Alexandre Filipe Zambujo de Brito

Licenciado em Ciências da Engenharia Electrotécnica e Computadores

Localization and Trajectory Control Algorithms Applied on Drones

Dissertação para obtenção do Grau de Mestre em

Engenharia Electrotécnica e Computadores

Orientador: Doutor Luís Filipe Figueira de Brito Palma, Professor Auxiliar, Faculdade de Ciências e Tecnologia da Universidade Nova de Lisboa

Co-orientador: Doutor Fernando José Vieira do Coito, Professor Associado, Faculdade de Ciências e Tecnologia da Universidade Nova de Lisboa

Júri

Presidente: Luis Augusto Bica Gomes de Oliveira, Professor Auxiliar com Agregação
Arguente: João Almeida das Rosas, Professor Auxiliar
Vogal: Fernando José Vieira do Coito, Professor Associado

FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

Março, 2018

Localization and Trajectory Control Algorithms Applied on Drones

Este documento foi gerado utilizando o processador (pdf)LaTeX, com base no template "unlthesis" [1] desenvolvido no Dep. Informática da FCT-NOVA [2].
[1] https://github.com/joaomlourenco/unlthesis [2] http://www.di.fct.unl.pt

ACKNOWLEDGEMENTS

The realization of this dissertation represents the end and the start of a big step in my life, so I would like to thank everyone that was part of it and that provided me this wonderful experience.

First of all, I would like to thank my advisor Professor Luís Filipe Figueira Brito Palma and my co-advisor Professor Fernando José Vieira do Coito for the opportunity, patience, effort and transmitted knowledge along the past year, that helped the realization of this dissertation. A special word to MSc. Vasco Brito, who also provided very useful help and guidance throughout this work.

I would also like to show my gratitude to my colleagues and friends, Alexandre Silva, André Quintanova, Carina Dias, Guilherme Almeida, Luís Alves, Nuno Pereira, Tiago Santos, Yaniel Barbosa, among others for all the help and support not only for the thesis but for the whole university years and of course for their valuable friendship.

Special thanks to my parents, João Manuel de Brito and Teresa da Piedade Pereira Zambujo de Brito, and my sister Laura Isabel Zambujo de Brito for the unconditional love, support and dedication to raise me, which made me the person I am today.

A special acknowledgment to my friends Maria Cristina, Jorge Lourenço, Giovanny Rodrigues, Tozé Soares, Francisco Ferreira, Bernardo Ferreira, Andreia Santos, Rodrigo Caçorino for the friendship and time spent together that helped me unwind from time to time.

Last but not least, I want to thank Faculdade de Ciências e Tecnologia of Universidade Nova de Lisboa for allowing me to learn, grow and develop traits that will help me in my career and personal life.

ABSTRACT

In this dissertation, the trajectory control of the quadcopter is explored and developed with the objective of finding the best way travel in terms of speed and energy consumption. The sensor fusion of several GPS modules is implemented as an algorithm that provides better localization measurements and reduces noise. An attempt to identify the NAZA® attitude controller in order to obtain its mathematical model is also subjects of this thesis. Trajectory algorithms are designed and tested with and without faults in the motors, on the X8 configuration in Simulink®. The main contributions are the improved GPS signal reception and algorithms for an autonomous trajectory following quadcopter. Experiments in the real-world quadcopter were done in order to validate the performance of such contributions. The simulations and experiments presented good performance of the quadcopter's behavior when integrating the filtered GPS signal. Simulations show the continuous improvement for trajectory generation and following of the drone between the three controllers tested (from worst to best): PID, state space feedback and differential flatness.

Keywords: quadcopter, modeling, fault tolerance, fault tolerant control, kinematic and dynamic systems, trajectory planning and following, sensor fusion, system identification, Arduino, Matlab®, Simulink®.

RESUMO

Nesta dissertação, a exploração e o desenvolvimento de controlo de trajetória tem como objectivo encontrar a melhor forma de completar trajectorias em termos de velocidade e consumo de energia. A fusão sensorial de vários modulos GPS são implementados como algoritmo que melhora a precisão de localização e diminui o ruído. A tentativa de identificação do controlador de atitude NAZA® de forma a encontrar o seu modelo matemático é também um dos temas desta tese. Algoritmos de trajectoria são desenhados e testados, com e sem falhas nos motores, na configuração X8 em Simulink®. As principais contribuições são a melhor recepção de sinal GPS e algoritmos de seguimento de trajectória de quadcopteros de forma autónoma. Foram feitas experiências no drone real de forma a validar o desempenho de tais contribuições. As simulações e experiências revelaram um bom comportamento do quadcoptero ao integrar o sinal de GPS filtrado. As simulações revelaram uma continua melhoria em relação à geração e seguimento de trajectória entre controladores testados (do pior para o melhor): PID, retroação de espaço de estados e differential flatness.

Palavras-chave: quadcopter, modelação, tolerância a falhas, controlo tolerante a falhas, cinemática e dinâmica de sistemas, planeamento e seguimento de trajectória, fusão sensorial, identificação de sistemas, Arduino, Matlab®, Simulink®.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ACRONYMS

CPU	Central Processing Unit
CS	Checksum
CS1	Checksum byte 1
CS2	Checksum byte 2
DC	Direct Current
DF	Differential Flatness
DOP	Dilution of Precision
ESC	Electronic Speed Controller
GPS	Global Positioning System
ID	Identification
IMU	Inertia Measurement Unit
LED	Light Emitting Diode
LiPo	Lithium Polymer
LQR	Linear Quadratic Regulator
MIMO	Multiple Input, Multiple Output
NARX	Nonlinear Autoregressive Exogenous
NED	North-East-Down
NMEA	National Marine Electronics Association
NNSS	Navy Navigational Satellite System

PID	Proportional Integral Derivative
PWM	Pulse Width Modulation
R/C	Radio Communication
RX	Reception
SI	International System of Units
SPI	Serial Peripheral Interface
SRAM	Static Random Access Memory
SS	State Space
TDM	Time Division Multiplexing
TWI	Two Wire Interface
TX	Transmission
UAV	Unmanned Aerial Vehicle
USB	Universal Serial Bus
UTM	Universal Transverse Mercator
VTOL	Vertical Take-Off and Landing
VU	Versatile Unit

LIST OF SYMBOLS

e	Error
γ	Angular position
I	Inertia
J_t	Inertial moment of rotation
m	Body mass
ϕ	Roll
ψ	Yaw
T_i	Integration time
T_d	Derivative time
T_u	Period of oscilation
θ	Pitch
u	Input
w	Motor's rotation speed
x	State
ξ	Linear position
y	Output

CHAPTER 1

INTRODUCTION

1.1 Motivation

The operation of aircrafts, more specifically the quadcopter, requires a well-aware and efficient pilot, be it human or machine. This starts with the quality of the sensors: accelerometer, barometer, gyroscope, GPS, Inertia Measurement Unit (IMU) and Compass, which are crucial for the awareness of the pilot during the flight. Efficiency of the pilot can be interpreted in many ways, but in this case, it refers to how good his or its actions are performed on the drone.

One of the main topics of this dissertation is the trajectory planning and tracking with the drone, and this research will explore what are the best ways to do it related with the polynomial form of the path along the waypoints and how the accelerations should be manipulated for less flight time or less energy consumed during the flight.

The other main topic regards the fact that the GPS, being the most used localization system around the world for its innumerous advantages, still has its limitations. Since the quadcopter is highly location dependent, finding ways to fight these limitations is another objective of this thesis.

1.2 Main Goals and Contributions

The main goal of this dissertation is to design algorithms that successfully plan feasible trajectories from predefined waypoints and control the quadcopter through them, considering the energy consumption and flight time. Since this dissertation is a continuation of Vasco Brito thesis "Fault Tolerant Control of a X8-VB Quadcopter", the further validation of the fault tolerant aspect of the X8 quadcopter is also experimented here. The second goal of this thesis is related with location precision, for which the sensor fusion of several GPS modules is addressed. A third goal is to

identify the NAZA attitude controller (from the DJI commercial quadcopter kit) in order to obtain a more realistic model, since this controller operates on a real drone.

One of the main contributions of this thesis is the clarification that the classic control algorithms designed in the industry are not recommended or efficient for such a fast system like the quadcopter. This means that the overshoot and slow rise time are not tolerated by a system like this, which happens with any purely feedback control approach. To this end, the Differential Flatness was addressed for optimal trajectory following and continuity, thanks to its feedforward aspect which takes into account the system dynamic equations to calculate the best possible actuations. Another contribution is the development of location improvement algorithms, that uses several sensors for better localization of the quadcopter.

1.3 Dissertation Structure

This document is organized in five chapters and in the following structure:

- Chapter 1 presents the motivation, main goals, and contributions of the dissertation;
- Chapter 2 is the state-of-the-art, which starts by explaining background history of the quadcopters and their evolution, followed by the trajectory and localization concepts addressed in this research, with their respective control methodologies, and finally, the related work;
- Chapter 3 shows this research's development, from the hardware units specifications and descriptions to the quadcopter assembly and modelling and from the localization and trajectory algorithms design to their respective control approaches;
- Chapter 4 presents the closed loop simulations and experimental results of the control algorithms applied on the real-world quadcopter. Further comparison between trajectory algorithms' performances and improvement in localization precision are clarified;
- Chapter 5 covers the conclusions and the future work of this dissertation;

CHAPTER 2

STATE OF THE ART

2.1 Introduction

The state of the art will present fundaments of the work done in this dissertation, covering two main topics in the research area of quadcopters, trajectory and localization. In the trajectory area, algorithms of trajectory generation and optimization already studied, verified and experimented will be presented and discussed, alongside with their methods integrated in such an aerial vehicle and their respective control approaches.

The localization research area will also be a theme in this thesis, so a brief history of its appearance will be presented, how it evolved along the years and how it is now a powerful technology for a wide range of applications.

2.2 Multicopter History

The quadcopter is essentially a helicopter with four rotors displayed in a square formation equally displaced at the same distance from the center of its body. Just like the helicopter, the quadcopter has a means to sustain its body aloft by pushing the air downwards, which categorizes them as a Vertical Take-Off and Landing (VTOL) aircraft (Luukkonen, 2011).

The history of VTOL aircrafts starts in 1490 with a design considered to match the characteristics of a helicopter made by the Italian artist and scientist Leonardo da Vinci. The drawing can be observed in Figure 2.1. This helical or lifting screw was inspired by the Archimedes' water screw in which he believed that it could possibly take-off vertically (Heatly, 1986).

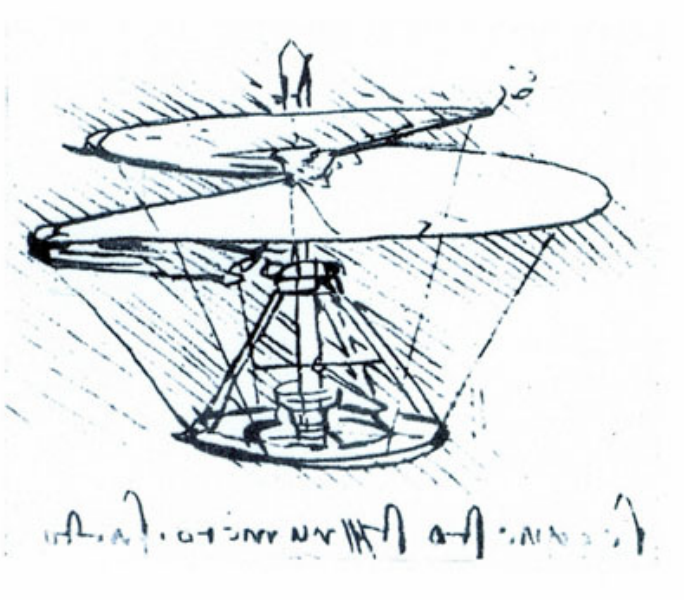

Figure 2.1: Leonardo da Vinci's drawing (Gibbs-Smith, 1978).

A wide variety of minor inventions contributed to the enhancement of the helicopter after the fifteenth century, presenting a bulky and heavy structure but lacking suitable power. Only in the latter half of the nineteenth century began to appear more realistic concepts used in nowadays helicopters, like the proposed rotor with adjustable pitch and the tail rotor or screw to counteract the torque provoked on the fuselage (Heatly, 1986). The aircraft propellers as we know today were pioneered by the Wright brothers, realizing that the propeller is essentially the same as a wing , as concluded from their wind tunnel experiments (Federal Aviation Administration, 2008).

The first known quadcopter, originally called gyroplane, has its debut appearance in France in the year of 1907, invented by the French brothers Louis and Jacques Breguet. The machine can be observed in Figure 2.2. This quadcopter was the first piloted aircraft able to lift vertically and it managed to fly several times (Brito, 2016).

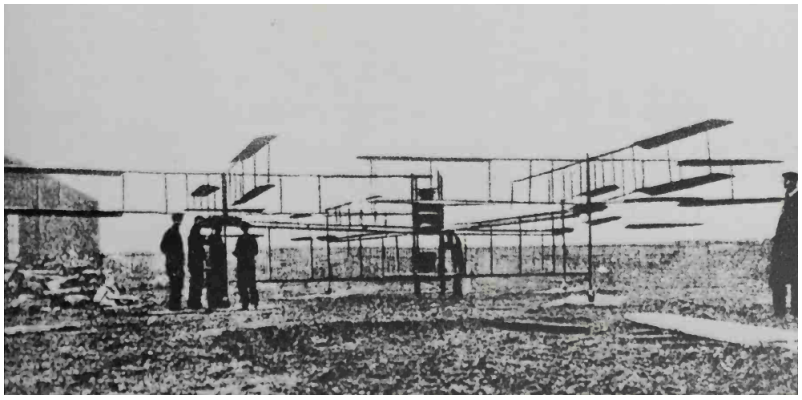

Figure 2.2: Breguet-Richet Gyroplane (Heatly, 1986).

Following this successful experiment, Etienne Oehmichen investigated on rotorcraft himself and built the Oehmichen No.2 with four rotors and eight propellers powered by a single engine as illustrated in Figure 2.3. This model presented a high level of stability and controllability for its time, performed more than a thousand of test flights and broke several flight records among helicopters (Spooner, 1924).

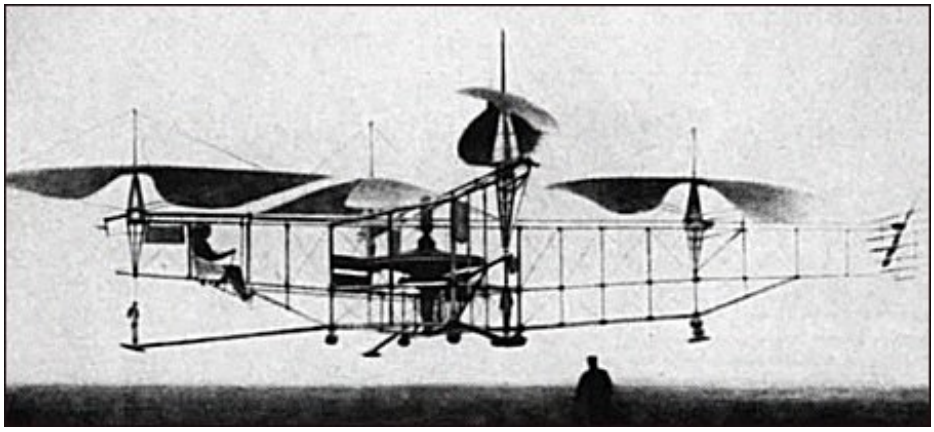

Figure 2.3: Oehmichen No.2 (Spooner, 1924).

Other quadcopter models were experimented, until the scientists realize that the single rotor helicopter with a tail rotor to counterbalance the torque caused on the fuselage was the better option, because of three major factors:

- The single rotor with tail rotor is naturally stable due to the weight being supported by only one attachment point to the rotor, causing it to naturally hang straight down and correct unwanted tilting with the force of gravity;
- The quadcopter is not naturally stable, therefore requiring some form of additional control to keep it stable at all times, either by the pilot or by a computer, which was non-existent at that time;
- More rotors imply more connections to the engines in the middle of the fuselage which requires more and long extensions of belt to the arms, while on single rotor helicopters only required a small connection to only one rotor (Krossblade Aerospace, 2017).

In the last decades, quadcopters have become more popular as Unmanned Aerial Vehicles (UAV) for various applications. These smaller vehicles fill the niche of what full-sized manned aircrafts like helicopters and airplanes can't or are inadequate for the task. Applications like routine surveillance, attack strategies in the military field and civilian applications from border protection to search and rescue, are nowadays commonly performed by quadcopters (Bouabdallah, 2007). One example of the most recent UAVs is the latest DJI Phantom 4 Pro represented in Figure 2.4.

Figure 2.4: DJI's Phantom 4 Pro (DJI, 2016).

2.3 Trajectory

Creating algorithms capable of generating collision-free trajectories in less time has been quite the challenge since the quadcopters emerged. The constantly evolving technology finally allowed fast and efficient controllers to actuate these naturally unstable aerial robots. The algorithms can be classified as:

- Algorithms based on geometric path primitives, in which the waypoints from start to final position are defined in space and then parameterizing these geometric path primitives along the way such as lines, polynomials or splines;
- Algorithms based on minimum velocity trajectory, where the control input constraints the velocities during flight through the differential flatness property of the quadcopter which will approve its feasibility;
- Optimal control considering the non-linearity of the system, which consists in control algorithms that directly considers the nonlinear dynamics of the quadcopter (Hehn and D 'andrea, 2015).

2.3.1 Algorithms based on geometric path primitives

The trajectory generation based on geometric path primitives accounts with the predefined set of waypoints along the way. The quadcopter can travel consecutively between waypoints in a line, facing directly the next target destination in the sequence, in the 3D space as illustrated in Figure 2.5.

In this example the quadcopter travels along the path P from a desired waypoint x_i^d to x_{i+1}^d according to the along track u_{at} and cross track u_{ct} inputs, which represents the most simplistic

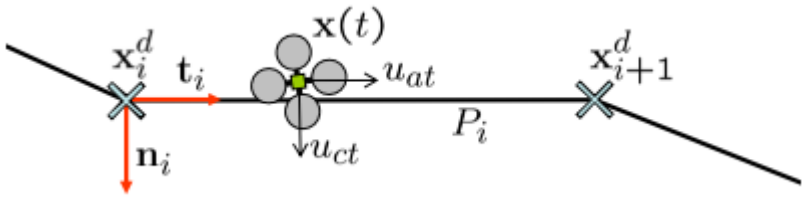

Figure 2.5: Quadcopter path definition with lines as geometric path primitives between waypoints (Hoffmann et al., 2008).

form of travelling but also the most inefficient. Because of the infinite curvature existing in each waypoint the quadcopter is forced to stop in each of these occasions (Hoffmann et al., 2008).

Another kind of path parametrization is the polynomial, characterized by smoothing the edges existing along the waypoints, resulting in a less clunky, more graceful way of travel. Among the several types of polynomials, the elementary and Chebyshev have been particularly explored in the research of trajectory optimization (Cowling et al., 2007).

In a typical case of the elementary polynomial where up to the second derivative of Cartesian coordinates must be satisfied at both ends of the trajectory, the third-order polynomial represents the appropriate smoothness desired. Integrating this polynomial twice results in a 4^{th} and 5^{th} order polynomials for the first derivative and i^{th} coordinate. This means that, if an additional flexibility is needed, the order of approximation can be increased resulting in higher order derivatives and additional variable parameters, as illustrated in Figure 2.6 (Yakimenko, 2006).

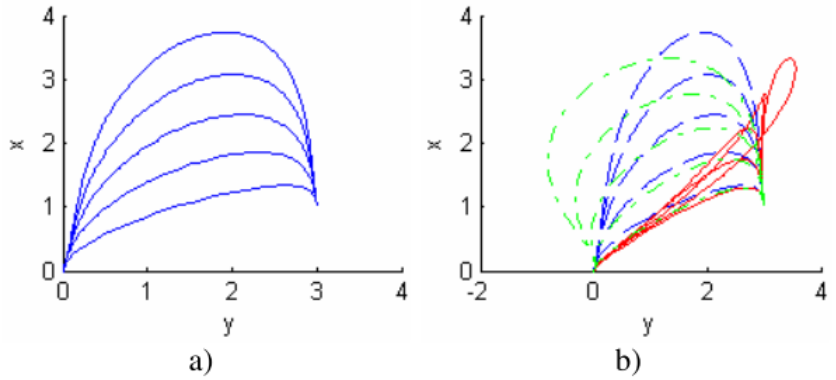

Figure 2.6: Elementary polynomials: a) Varying only one coefficient; b) Varying an additional coefficient for more flexibility (Yakimenko, 2006).

The Chebyshev polynomial method is one way to construct a polynomial approximation to the solution, in which both the states and control variables are expanded in terms of Chebyshev polynomials with unknown generalized Fourier coefficients. The use of the properties of this method implies the conversion of state equations, performance index and boundary conditions into

algebraic or transcendental equations in terms of unknown coefficients (Fahroo and Ross, 2002).

Another method used in the trajectory generation algorithms is the B-Spline or Basic Spline polynomials, that starts by defining the waypoints (except the first and last) as control points and then generating an approximation of B-spline functions fitting the control points, resulting in a pathway that nearly crosses each waypoint for a more smooth set of transitions (Bouktir et al., 2008), as represented in Figure 2.7.

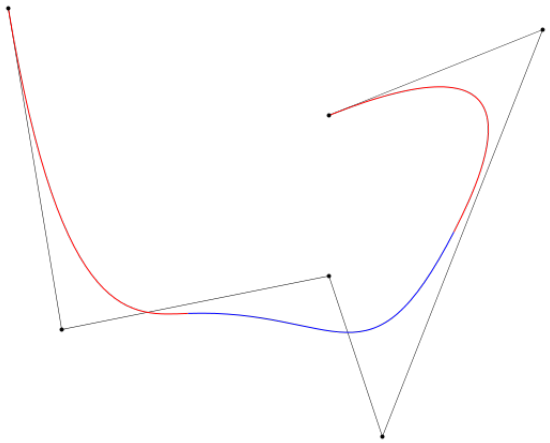

Figure 2.7: B-spline with the dots as control points.

2.3.2 Algorithms that minimize the derivative of the position trajectory

These kinds of algorithms use the quadcopter's differential flatness property through the control inputs to minimize the derivative of the position trajectory, directly affecting its feasibility. One example of this method is the minimum snap trajectory generation, that minimizes functionals of the trajectories described by basis functions, presented by (Mellinger, 2011). This approach is very useful for differential flatness control methodologies, since the trajectory references and their derivatives that form the optimal and smoothest way to travel between waypoints, can be provided by the trajectory generator. Another convenience of this method is the possibility of adding constraints in the trajectory corresponding to environment objects and walls, as it can be observed in Figure 2.8.

Another work that solves this optimization problem through a similar method is done by minimizing the weighted sum of derivatives (Richter et al., 2013). In Figure 2.9 this process of iterative refinement of segment times is illustrated, varying the weights of the derivatives for a faster traveling time. This algorithm also adjusts these weights automatically depending on the environment or when it knows it has to slow down to navigate through tight spaces without incurring excessive snap.

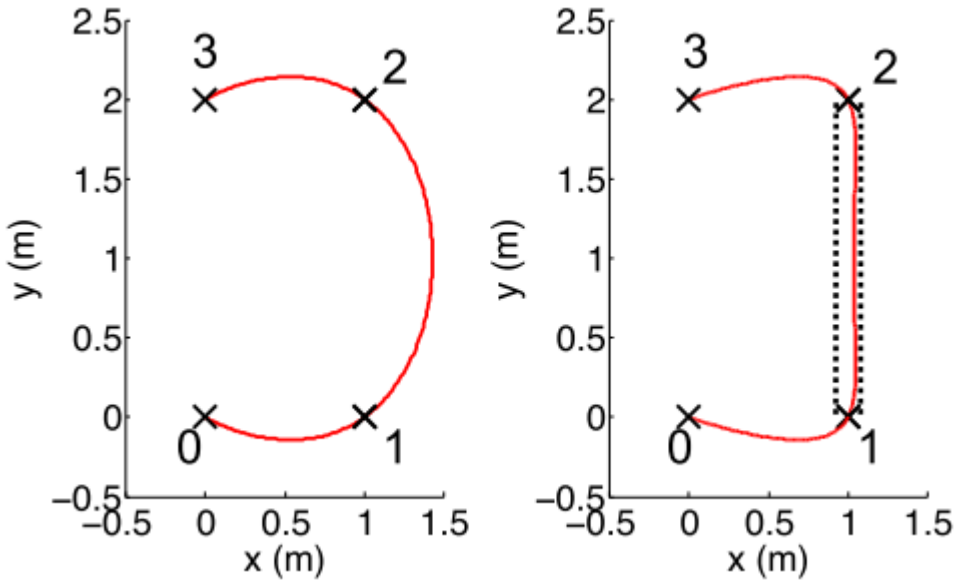

Figure 2.8: Optimal trajectories. Left: no corridor constraints; Right: corridor constraints between waypoints 1 and 2 (Mellinger, 2011).

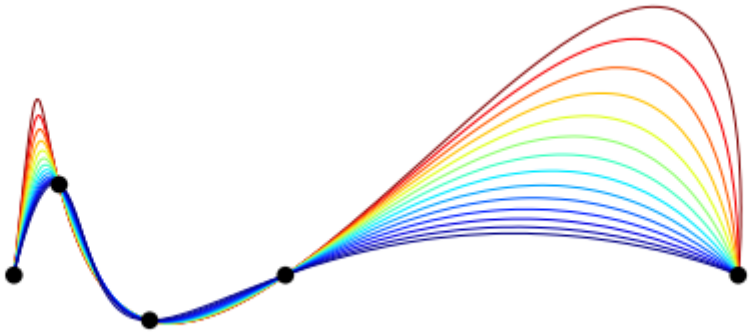

Figure 2.9: Iterative time refinement of a pathway through waypoints (Richter et al., 2013).

2.3.3 Optimal control considering the non-linearity of the system

There is also another way to perform a trajectory with a quadcopter, not taking for granted the linearized behavior provided by the controller, since these kinds of trajectory planning accounts with the non-linear aspect of the quadcopter itself. This brings faster response from the quadcopter movements since the controller will use its natural instability to perform faster rotations. Such approach is done by Ritz (Ritz et al., 2011) using custom electronics that allow the deployment of custom control algorithms. These algorithms generate control input based on the planned trajectory and the feedback obtained from the board gyroscopes to control the quadcopter rotational rates. Figure 2.10 illustrates the rotational rate (pitch) existing during the displacement of various distances and as it can be observed, the angle obtained during these transitions is very aggressive

when comparing with the common quadcopter's overall agility.

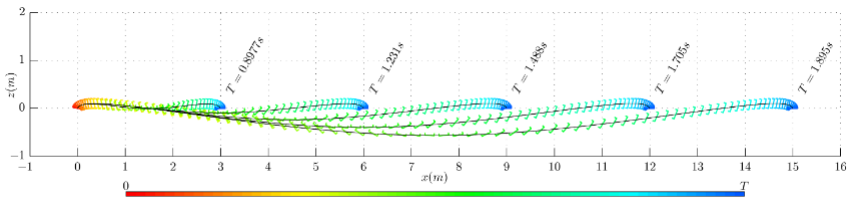

Figure 2.10: Illustration of maneuvers for horizontal displacement (Ritz et al., 2011).

2.4 Localization

The UAVs nowadays rely heavily on the Global Positioning System (GPS) if its localization/navigation across the surface of the Earth is needed. This system is composed by a satellite constellation that, in order to provide continuous global positioning, requires at least four satellites that are always electronically visible by the GPS receiver. After several constellation schemes proposed, the scientists came to a consensus that 21 evenly spaced satellites in circular 12-hour orbits can provide the global coverage with the least expense. The position coordinates provided by the GPS are expressed by latitude, longitude and elevation, which are calculated through the distances measured from the center of the earth to each satellite as illustrated in Figure 2.11. These satellites are equipped with atomic clocks, which are the most accurate time and frequency standards known, being able to provide good accuracy in the GPS position calculations (Hofmann-Wellenhof et al., 1994).

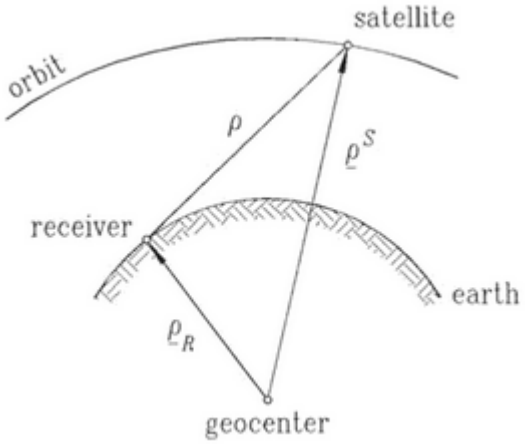

Figure 2.11: Principle of satellite positioning (Hofmann-Wellenhof et al., 1994).

2.4.1 History

The pioneer technique in electromagnetic localization was the High Ranging (HIRAN) system developed during the World War II for aircraft positioning purposes. Some years later, this system was used to attempt to determine the difference between Europe and America's datum, measuring HIRAN's arcs of trilateration, which caused a technological breakthrough when the scientists experienced the Doppler Effect in the signal broadcasts, and came to a conclusion of its usefulness for accurate time determining of satellite's position. This knowledge alongside with the Kepler's laws led to the present capabilities of near-instantaneous and precise localization incorporated in the GPS. The Navy Navigational Satellite System (NNSS) was the successor of this new technology, developed in the U.S. Military and initially for military exclusive use to determine the positioning of vessels and aircrafts. This system composed by six satellites is still operational and used by thousands of small vessels and aircrafts, military or civilian, to determine their position worldwide (Hofmann-Wellenhof et al., 1994).

Later in the year 1973, the GPS project was launched to overcome the limitations existing in the NNSS, mainly the large time gaps on its global coverage and its poor navigation accuracy. It was concluded that the previous system was not able to provide continuous positioning data to the user with only six satellites. The new Global Positioning System was able to provide this with very accurate time, position and velocity of the receiver device, using a total of 24 operational satellites in the constellation according with the present policy (Hofmann-Wellenhof et al., 1994). Other similar global positioning systems were later developed and launched by other organizations, namely the Russian GLONASS (Polischuk et al., 2002), Chinese BeiDou (Nowakowski, 2015) and the European Galileo ((GPS/Daily), 2014) satellite navigation systems, launched in 1982, 2000 and 2011, respectively.

2.4.2 GPS concept and fundamentals

As stated before, the current GPS constellation is formed by 24 satellites that communicate through radio-frequency signals to provide the current position to its user accurately for an easy, fast and safe navigation. This system can be found in cars, boats, airplanes and even smartphones.

The system itself relies upon the trilateration method to detect the position of the receiver. This method is illustrated in Figure 2.12 and can be described first by imagining that the distance from the position that has to be determined to one of the satellites is R1, and this position seen from the satellite could be anywhere on the sphere of radius R1. If another satellite is to measure that position, it will obtain R2 by chance and the distance from the desired coordinates to both satellites will be reduced to their intersection, forming a circle in space of possible coordinates. Adding a third satellite with distance R3 will reduce the possible coordinates to two and from these, there could be a fourth measurement but usually one of these points is a ridiculous answer due to

impossible position or velocity (El-Rabbany, 2002).

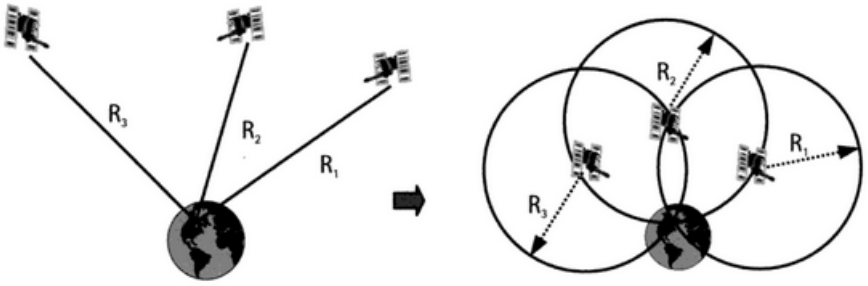

Figure 2.12: GPS's trilateration method (El-Rabbany, 2002).

The GPS recievers obtain these metric measurements by calculating the time that the signal takes to arrive from the satelites. To obtain accurate timing, the satellites are equipped with atomic clocks, however the GPS receivers are not, which is why there must be a fourth satellites for timing calculations, otherwise every GPS receiver had to have a built in atomic clock, which would make this technology unaffordable for most users (El-Rabbany, 2002).

The signal propagation in vacuum, where the satellites orbit, happens at speed of light, but when the signal crosses the atmosphere it gets slowed down by the ionosphere which causes uncertainties in the metric distance calculations. This delay is minimized with the dual frequency measurement where the signals are sent at two different frequencies and then compared by their relative speeds. Although this system is very sophisticated and convenient to a lot of users around the world for navigation, it is not perfect, having its limitations. The very high accuracy of the atomic clocks is not perfect and can account for some tiny errors. The system suffers from multipath error resulting from the reflection of the signal bouncing back to the receiver as illustrated in Figure 2.13, and fails to work in indoor, forest, or urban environments due to blockage of line-of-sight (El-Rabbany, 2002).

2.4.3 GPS Coordinates and metric Conversion

The metric system was invented by the French and many times reformulated around a bar that would serve as a model for the SI unit, the meter. After one last convention with most countries in Europe, it was finally defined that 10 million meters would be the distance between the equator and a pole (Alder, 2002). As for the GPS coordinates, dividing this distance by the 90º that forms between the equator and the pole, yields 111111 meters per degree. Since the coordinates are given in degrees for the decimal degrees format, one can intuitively calculate the precision of each decimal digit:

- tens digit – precision of about 1111 Km;

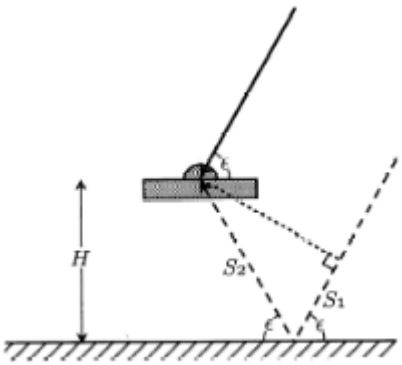

Figure 2.13: Signal reflection from the satellites resulting in the Multipath error (Floyd and Palamartchouk, 2015).

- units digit – precision of about 111 Km, as calculated above;
- first decimal digit – precision of about 11 Km;
- second decimal digit – precision of about 1111 m;
- third decimal digit – precision of about 111 m;
- fourth decimal digit – precision of about 11 m;
- fifth decimal digit – precision of about 1,111 m;
- sixth decimal digit – precision of about 0,111 m;
- seventh decimal digit – precision of about 11 mm.

This conversion makes it easier for humans to understand how much precision a GPS module yields, depending on the number of decimal places, and how impactful each decimal place is related to the conventional SI unit of distance (meter).

2.4.4 GPS Coordinate formats

There are several formats to display the coordinates. Table 2.1 presents an example of some of these formats for the coordinates of the Aerodynamics and Control laboratory at the Electrical Engineering Department in FCT UNL Campus.

The GPS modules provide data in the formats enumerated in Table 2.1 and more, but the ones that were subject of this dissertation only used the examples listed.

Table 2.1: GPS coordinate formats of the Aerodynamics and Control laboratory at the Electrical Engineering Department in FCT UNL Campus.

Coordinates format	Latitude	Longitude
Decimal Degrees	38.6604117	-9.205056
Degrees, min, sec	N 38º39'37.5"	W 9º12'18.2"
Decimal Minutes	N 3839.625	W 912.303
UTM	29N 482159	4279113

2.5 Control Approaches

2.5.1 Proportional Integral Derivative Control

The Proportional Integral Derivative or PID controller is a very common solution in feedback control of industrial processes. As designated in its name, it is composed by three elements:

- The proportional element, which deals directly with the difference between the desired setpoint and the measured process variable with a proportional gain;
- The integral element, that can be interpreted by the accumulation of the past error and used in the present iterations;
- The derivative element, is predicting the future error and using this knowledge to act on the control input preemptively (Araki, 2002).

These three components form the PID and the control input is described by the equation (2.1).

$$u(t) = K\Big(e(t) + \frac{1}{T_i}\int_0^t e(\tau)d\tau + T_d\frac{de(t)}{dt}\Big) \tag{2.1}$$

In the continuous time, u is the input, e is the error or the difference between the reference and the measured output signal, K is the proportional gain, T_i is the integration time and T_d is the derivative time.

This controller was originally developed analogically, with pneumatic valves, relays, motors, transistors and integrated circuits, through a lot of experiences and development stages. The PID, nowadays, are mainly digital due to the digital technology advancements, in which the appearance of microprocessors accelerated drastically the development of this controllers and allowed the development of additional features like automatic tuning, gain scheduling and continuous adaptation (Åström and Hägglund, 2006).

2.5.1.1 Ziegler-Nichols method

The PID will not work if the gains related with each of the three elements are left unattended. This mean that K, T_i and T_d must be chosen somehow. To choose these gains, one of the two classical heuristic methods like the step-response method or the ultimate-sensitivity method is usually used to tune efficiently a PID controller.

In the step-response method, the step response of the open-loop process is experimentally obtained and from the intersection L of the steepest slope R during the rising time is marked on the graphic illustrated in Figure 2.14.

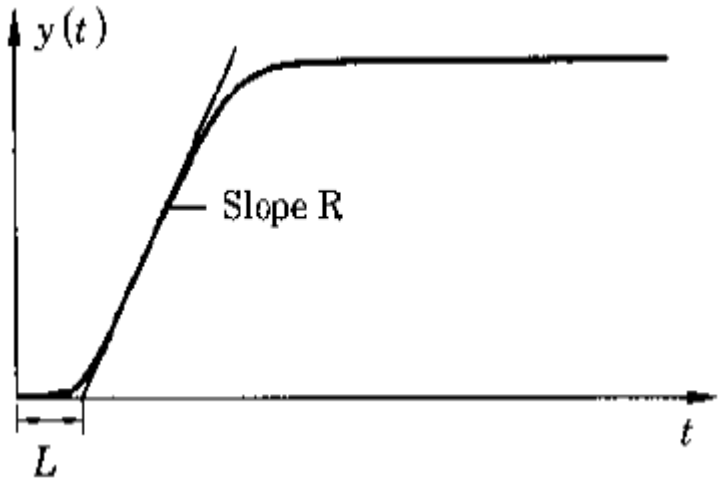

Figure 2.14: Determination of the parameter R and L (Åström and Wittænmark, 1997).

With R and L, one can calculate $a = RL$ and obtain the gains for each element of the PID through Table 2.2.

Table 2.2: Controller parameters obtained from the Ziegler-Nichols step response method.

Controller	K	T_i	T_d
P	$\frac{1}{a}$		
PI	$\frac{0.9}{a}$	$3L$	
PID	$\frac{1.2}{a}$	$2L$	$\frac{L}{2}$

As for the ultimate-sensitivity method, the key idea is to find the limit of stability of the process with the controller in closed-loop by varying the proportional gain alone. The ultimate gain K_u and period of oscillation T_u obtained in this experience are then used to calculate the gains of the PID according with Table 2.3 (Åström and Hägglund, 2006).

Table 2.3: Controller parameters obtained from the Ziegler-Nichols ultimate-sensitivity method.

Controller	K	T_i	T_d
P	$0.5K_u$		
PI	$0.4K_u$	$\frac{T_u}{1.2}$	
PID	$0.6K_u$	$\frac{T_u}{2}$	$\frac{T_u}{8}$

2.5.2 Model Predictive Control

The model predictive control is essentially the prediction of the process behavior based on its equivalent model. This feedback control method will observe the process response of a given input and knowing its model behavior, it will iteratively apply additional control input to obtain the desired output, as illustrated in Figure 2.5.2. It is mostly used for sampled systems, due to its simplicity and convenience that the control input and output constraints can be taken into account (Åström and Hägglund, 2006).

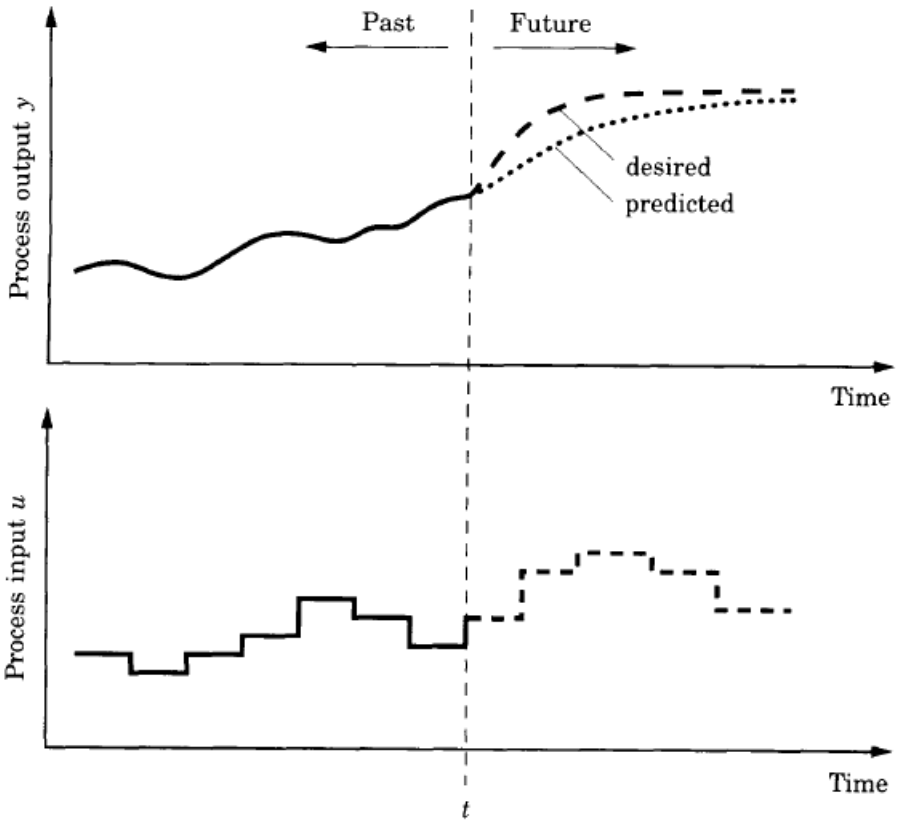

Figure 2.15: Illustration of model predictive control (Åström and Hägglund, 2006).

This predictive control method can be approached in a basic formulation, assuming the process model is linear, the cost function is quadratic and the constraints are presented as linear inequalities. These will be represented in state-space and are assumed time-invariant.

Considering the model of a plant represented by the set of equations (2.2), (2.3) and (2.4),

$$x(k+1) = Ax(k) + Bu(k) \tag{2.2}$$

$$y(k) = C_y x(k) \tag{2.3}$$

$$z(k) = C_z x(k) \tag{2.4}$$

where x is a n-dimensional state vector, u is a l-dimensional input vector, y is a m_y-dimensional estimated output vector and z is a m_z-dimensional vector of outputs which are to be controlled.

Model predictive control repeatedly optimizes the current time-step input taking into account the future time-step inputs and response of the system, and as such, the cost function related with this optimization must be minimized. The cost function will penalize the deviations of the predicted controlled outputs $\hat{z}(k+i)$ related to the reference trajectory $r(k+i)$ and is defined by equation (2.5),

$$V(k) = \sum_{i=H_w}^{H_p} ||\hat{z}(k+i) - r(k+i)||^2_{Q(i)} + \sum_{i=0}^{H_u-1} ||\Delta\hat{u}(k+i)||^2_{R(i)} \tag{2.5}$$

where $\hat{z}$ is an estimation of z, Q and R are the quadratic forms ($||x||^2_Q$ and $||u||^2_R$ respectively), H_w and H_p are the beginning and end of the prediction horizon, respectively, and H_u is the control horizon.

The constraints can be denoted in the following form to hold the control and perdition horizons:

$$E\ vec(\Delta\hat{u}(k), ..., \Delta\hat{u}(k+H_u-1), 1) < vec(0) \tag{2.6}$$

$$F\ vec(\hat{u}(k), ..., \hat{u}(k+H_u-1), 1) < vec(0) \tag{2.7}$$

$$G\ vec(\hat{z}(k+H_w), ..., \hat{z}(k+H_p), 1) < vec(0) \tag{2.8}$$

in which E, F and G are matrices of suitable dimensions and vec(0) denotes an empty vector of suitable dimensions. This kind of representation can be interpreted as actuator slew rates, actuator ranges and constraints on the controlled variables.

This model-based predictive control methodology is on of the few advanced control technique to have a major impact in the process control industry. The reason being that it is a generic control method able to deal with equipment and their safety constraints, routinely. It is also easy to understand, to extend its formulation to multivariable processes and it is more powerful than PID

control overall, without being much harder to tune. The operation in the constraint environment of the model predictive control allows for the most efficient and therefore, most profitable operation in many cases (Maciejowski, 2002).

2.5.3 State Space Feedback Control

The state of a system, in state space feedback control, is defined as the full description of it at any time. A well formulated model of a system, in state space, must include the relevant information about its input u(t), output y(t) and state x(t) variables (Ogata, 1970).

Generically, a system can be described through the system dynamics equation (2.9) and the measurement equation (2.10),

$$\dot{x}(t) = Ax(t) + Bu(t) \tag{2.9}$$

$$y(t) = Cx(t) + Du(t) \tag{2.10}$$

where A, B, C and D are constant matrices of appropriate dimensions, in which:

- A stands for the dynamic matrix;
- B is the input matrix;
- C is the output matrix;
- D represents the direct transmission matrix.

This formulation is illustrated in Figure 2.16 and represents how the system reacts about its own set of rules, the rules of the surrounding environment and when applying a set of inputs, in which the transition from one state to another must obey all of these rules.

Before implementing control feedback in the system, one must assure that all the needed state variables are measurable and available for feedback. The controllability of the system should also be verified, as this will guarantee that the poles can be placed as desired in the closed-loop system, through an adequate state feedback gain matrix K. This feedback control is represented by equation (2.11).

$$u(t) = -Kx(t) \tag{2.11}$$

The system is completely state controllable if equation (2.12) is verified.

$$rank[B|AB|...|A^{n-1}B] = n \tag{2.12}$$

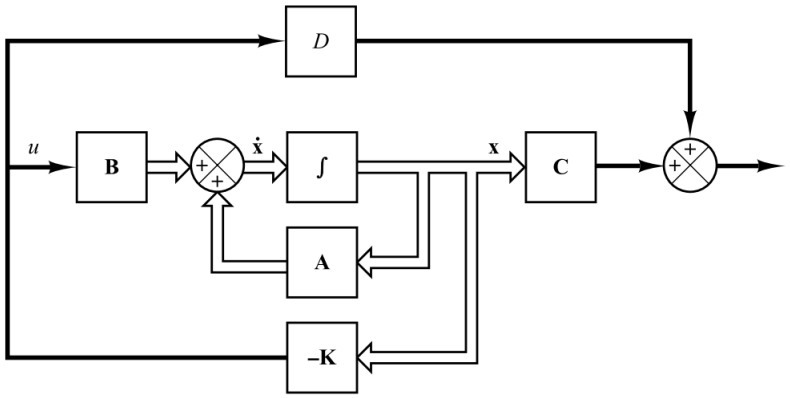

Figure 2.16: State space feedback control architecture (Ogata, 1970).

The objective is to find the K gain matrix that places the poles of the system in the left half of the s plane, which drives the equation (2.13)

$$\dot{x}(t) = (A - BK)x(t) \tag{2.13}$$

into a stable controllable state. This means that the stability and transient response of the system is determined by the eigen values of the matrix A-BK (Ogata, 1970).

2.5.3.1 Linear Quadratic Regulator

The Linear Quadratic Regulator is a powerful and popular design method among MIMO (multiple-input, multiple output) systems. This methodology was chosen to find the K matrix in the previous section, since it automatically ensures a stable closed-loop system, provides guaranteed levels of stability and is simple to compute (Anderson B., 1990).

Considering the system described in equations (2.9) and (2.10), the quadratic cost functional (2.14)

$$J = \int_0^\infty \left(x^T Q x + u^T R u \right) dt \tag{2.14}$$

is to be minimized, subject to the dynamics constraints of the open-loop system dynamics, providing this way, the optimal full state feedback control law.

In the quadratic cost functional J, Q is the matrix that attributes weights to each of the state variables x and R is the weight of the input u.

Solving the algebraic Riccati equation (2.15) for S:

$$A^T S + SA - SBR^{-1}B^T S + Q = 0 \tag{2.15}$$

where S is the unique, symmetric, positive semidefinite solution, which will be resorted to find the gain matrix K through equation (2.16).

$$K = R^{-1}B^T S \tag{2.16}$$

The gain matrix K produced this way guarantees an asymptotically stable closed loop dynamics of the system.

2.5.4 Differential Flatness Control

Differential flatness can be interpreted as the expression of the state and control variables of a system in terms of the output values (Fliess, 1990). A nonlinear system characterized by equation (2.17) is differentially flat if there exists variables $y \in R^m$, designated by flat outputs, in the form of (2.18) such that $x \in R^n$ and $u \in R^m$ can be recovered from by the equations (2.19) and (2.20), respectively,

$$\dot{x} = f(x,u) \tag{2.17}$$

$$y = h(x,u,\dot{u},...,u^{(r)}) \tag{2.18}$$

$$x = l(y,\dot{y},...,y^{(q)}) \tag{2.19}$$

$$u = g(y,\dot{y},...,y^{(q)}) \tag{2.20}$$

where x is the system's state, u is the input vector, and h, l and g are smooth functions.

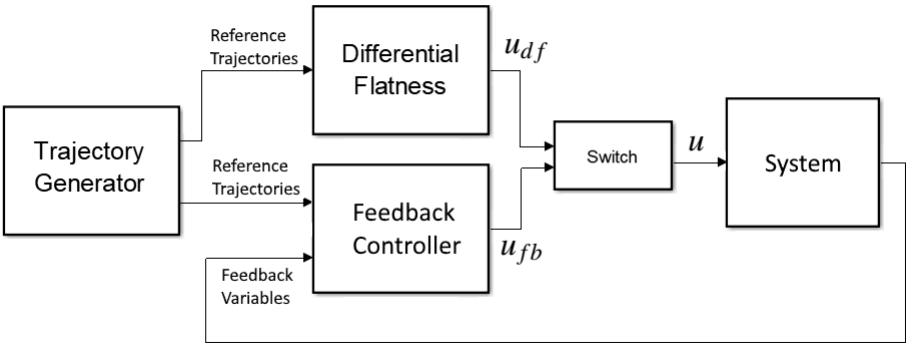

Figure 2.17: Differential Flatness Control architecture with feedback control as a stabilizer.

Given the initial x_0 and final x_f conditions of the states of a system x and theirs derivatives, the flat outputs y and their derivatives can provide the ideal inputs u that will drive the system to perform a smooth transition from one state to another while also respecting such conditions. This

control algorithm is categorized as feedforward control methodology, which actuates the inputs of the system while being unaware of the system's state. For this reason, feedforward control algorithms are, more often than not, complemented by a feedback control method to guarantee that any inconsistency or perturbation in the system doesn't deviate it from the desired setpoint. The differential flatness architecture is illustrated in Figure 2.17

2.5.4.1 Trajectory Generation

The smooth functions h, f and g mentioned in the previous section have to be produced somehow, so that the differential flatness controller can compute the inputs based on the reference. Such problem is usually approached by resorting to basis functions, that together form the polynomial desired as the reference. The polynomial can be calculated by means of simple monomials (Yakimenko, 2006), Chebyshev polynomials (Vlassenbroeck and Van Dooren, 1988), Laguerre polynomials (Balaji, 2007) among others. For this thesis, the monomial methodology was resorted, in which the resulting polynomial $P(t)$ can be described as equation (2.21).

$$P(t) = \sum_{n=0}^{M} a_k B_k(t) \tag{2.21}$$

where a_k is the coefficient aggregated with the respective monomial and $B_k(t)$ is the monomial itself. Associating this polynomial with the desired reference, one can simple derivate or integrate as many times as the differential flatness controller requires.

2.6 Sensor Fusion

Sensor fusion is the combination process of information acquired from multiple sources. This technique combines data from multiple sensors and related databases to achieve more accurate and detailed inferences than it would with only one sensor. This is the main advantage that the sensor fusion provides and the reason why it is needed overall.

This concept has its origins from humans and animals in the sense that they evolved with the capability of using various biological sensors (vision, smell, taste, audio and touch) for survival. Because they are equipped with various different sensors that provide information about different characteristics of the environment, a more complete perception of it is possible to understand (Hall and Llinas, 1997).

There are several techniques to computationally perform sensor fusion regarding signal, image and symbols, namely arithmetic mean, Kalman filter, logic filter and Fuzzy logic.

The Kalman filter is essentially a group of mathematical equations that provide an efficient way to estimate the process states in a recursive fashion. This filter can estimate the past, present

and future of these states, even when the model of the system is not completely known.

The filter was invented and later published in 1960 by the Mathematician Rudolf Kalman. The published paper described a recursive solution to the discrete-data linear filtering problem, which since then, due to the recent development of the digital computer, has been an extensive research subject in many investigation areas. This control approach is characterized by its estimation about the process and the proper correction (Welch and Bishop, 2006).

Considering the discrete-time state $x \in R^n$ of a given process described by the linear stochastic difference equation (2.23),

$$x_k = Ax_{k-1} + Bu_{k-1} + w_{k-1} \tag{2.22}$$

with a measurement as equation (2.23),

$$z_k = Cx_k + v_k \tag{2.23}$$

where w_k and v_k represent the process and measurement noise, respectively. These are assumed independent from each other, white and with normal probability distributions (2.24) and (2.25),

$$p(w) \sim N(0, Q) \tag{2.24}$$

$$p(v) \sim N(0, R) \tag{2.25}$$

in which Q is the process noise co-variance and R is the measurement noise co-variance and both are assumed constant, although they might change with each iteration.

The discrete Kalman filter algorithm is formed by two key process updates: the *a priori* estimation $\hat{x}_k$ of the next time step k and *a posteriori* correction considering the current measurement z_k. The time update equations project the state ahead (2.26) as well as the co-variance error (2.27).

$$\hat{x}_{k-1} = A\hat{x}_{k-1} + Bu_{k-1} \tag{2.26}$$

$$P_{k-1} = AP_{k-1}A^T + Q \tag{2.27}$$

The measurement update calculates the Kalman gain (2.28), updates the estimate with the measurement (2.29) and updates the covariance error (2.30),

$$K_k = P_{k-1}C^T(CP_{k-1}C^T + R)^{-1} \tag{2.28}$$

$$\hat{x}_k = \hat{x}_{k-1} + K_k(z_k - C\hat{x}_{k-1}) \tag{2.29}$$

$$P_k = (I - K_k C)P_{k-1} \tag{2.30}$$

where $z_k - C\hat{x}_{k-1}$ represents the measurement innovation which reflects the discrepancy between the predicted measurement $C\hat{x}_{k-1}$ and the obtained measurement z_k. K is the Kalman gain which minimizes the *a posteriori* co-variance error.

These two process updates are computed in every iteration, taking into account the previous *a posteriori* estimates used to predict the new a *priori* estimates. This algorithm is not only great at what it does but is also very practical and convenient, since it can be applied on a system and filter its data in real time, in which the information about the previous and present iteration is enough for full performance.

2.7 Related Work

Along the years, countless researches and developments have been done on drones, since this system became such a trend in the 2010's. This system attracts scientists and engineers due to its complex dynamics and also due to the promising future as a solution for an ever increasing amount of applications. Some research works have been done regarding the energy efficiency in drones and also in the polynomial trajectories while avoiding obstacles. One of which was done by Fabio Morbidi, Roel Cano and David Lara for the IEEE International Conference on Robotics and Automation, presenting algorithms for minimum-energy path between the initial and final configuration of a quadrotor by solving an optimal control problem with respect to the angular accelerations of the four propellers. The quadcopter used in this research was a DJI Phantom 2. In Figure 2.18 can be observed a bundle of trajectories minimizing energy consumption as concluded in their work (Morbidi et al., 2016).

Another work related with energy efficiency is the research done by Daniel Gurdan et al. which tackled the energy problems of the quadcopter by constructing one with minimal requirements to fly, in all aspects. This quadcopter is mainly composed by low-cost hardware and software and runs at low frequencies which cause it to have uncertainties in position control and unstable behavior. The disadvantages of this trade-off are fixed by their highly-optimized control algorithms (Gurdan et al., 2007). Their quadcopter is illustrated in Figure 2.19.

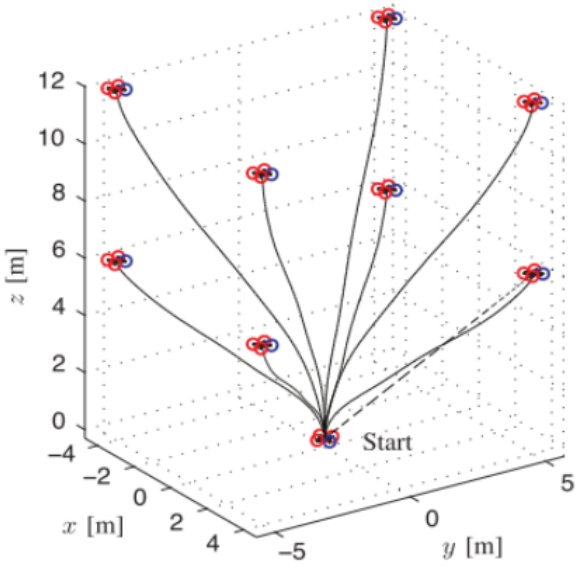

Figure 2.18: Bundle of minimum-energy trajectories of the quadrotor (Morbidi et al., 2016).

Figure 2.19: Low-cost quadrotor (Gurdan et al., 2007).

Several works can be found in the literature about sensor fusion for a wide range of applications. This methodology usually brings increased accuracy in tracking the true value and significantly attenuates noise. One of such works was done by a research group in France, led by Francois Caron, that developed a GPS/IMU multisensor fusion algorithm while validating contextual variables through fuzzy logic. Tests considering poor GPS data reception and drift were done, in which the algorithm weights how accurate the readings are in relation to the IMU data. The results of this sensor fusion algorithm can be observed in Figure 2.21, where it shows the error obtained for north, east and south readings of the GPS/IMU and the fused data error in the middle of each graph (Caron et al., 2006).

Identification of systems procedures can be approached in several ways and usually implies extensive work to find the right tools that allow for a precise identification, especially when it's a black box identification and there are various hidden variable changing the outputs. The work done by Parikh shows such case where the objective is to identify a quadruple tank system in closed loop.

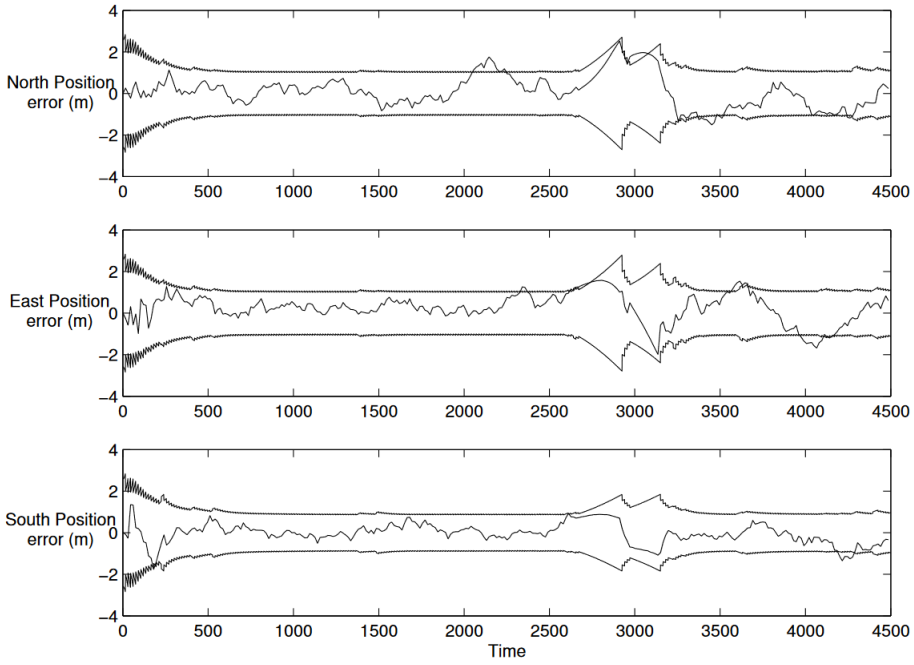

Figure 2.20: Sensor fusion of GPS and IMU through fuzzy logic (Caron et al., 2006).

The methodology to identify this system is illustrated in Figure 2.21. The identification process starts by replacing the multivariable controller with suitable equivalent proportional feedback controllers and accounts with noise signals (dither) added to the inputs and controller outputs. During the process, the identification is done by generating an estimate of the multivariable plant dynamic using the identification data related with the noise, the output and inherent relationships of the closed loop.

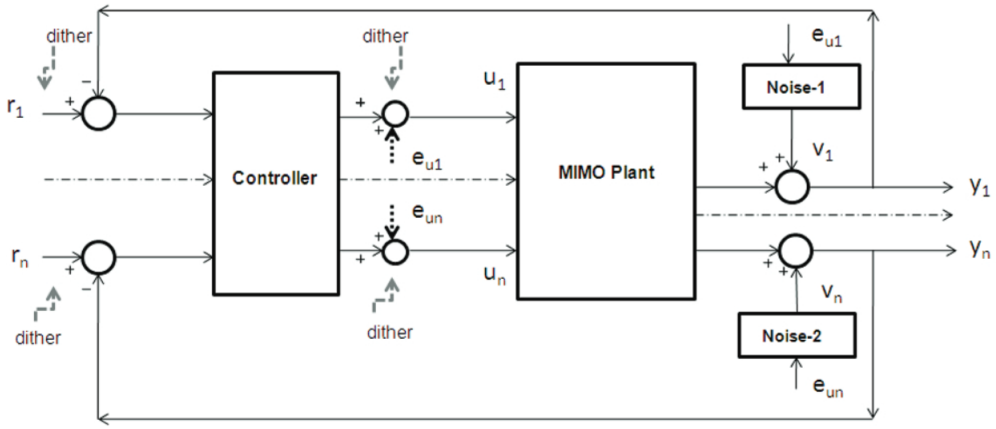

Figure 2.21: Architecture for the identification of the closed loop quadruple tank system (Parikh et al., 2012).

CHAPTER 3

SYSTEM'S STRUCTURE AND MODELING

3.1 Introduction

This dissertation follows a previous work done by Vasco Brito (Brito, 2016) regarding his adaptation of the NAZA quadcopter kit into a X8-VB quadcopter, and as such, all of its structure and components will be briefly described. The description of every piece of technology integrated in a quadcopter and additional parts used during the development of this dissertation will be illustrated and explained in this section. Additionally, the moddeling of the quadcopter X8 will be described in detail with the objective of elucidating the reader about the complex dynamic and kinematic model of this system.

3.2 Architecture Overview

Everything done in this dissertation is resolved around the NAZA controller illustrated in Figure 3.1. All trajectory tracking algorithms are designed to control the inputs (roll, pitch, throttle and yaw) so that the quadcopter can perform trajectories automatically. The GPS signal it receives is also a subject of this dissertation and the objective was to improve its precision by integrating more GPS signals and fuse them. Another objective was to create a model of the NAZA controller itself by exciting the inputs and observing the outputs, in order to identify it and obtain a more realistic model of the controller, instead of the previously obtained strictly from mathematical equations.

The NAZA possesses several inputs and outputs, but the ones relevant to this dissertation are illustrated in Figure 3.1. The LED device is responsible to indicate all kinds of states of the quadcopter namely: calibration, GPS signal, faulty motors, unplugged devices, flight mode, etc.

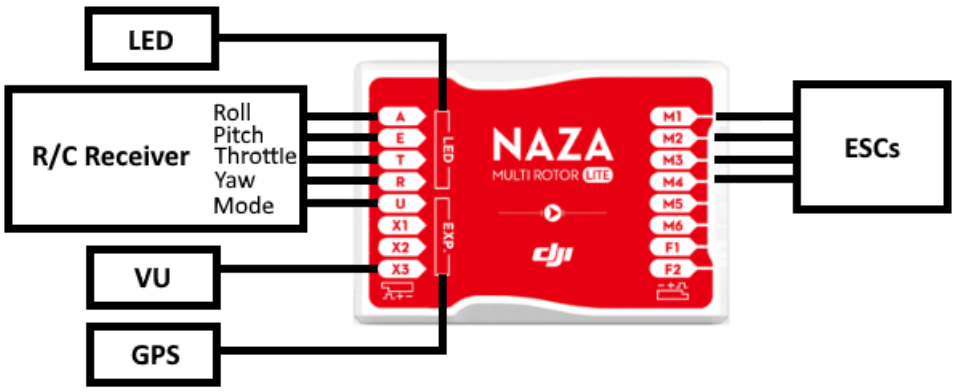

Figure 3.1: NAZA-M Lite controller and respective connections.

The Radio Communication receiver (R/C) is the device that receives the commands from the pilot and acts on the inputs of the NAZA controller, including the flight mode (manual, attitute or GPS). The Electronic Speed Controllers (ESCs) are responsible to provide the power to the motors as well as the respective PWM signals. The Versatile Unit (VU) is responsible to transform the power that comes from the battery to the adequate power for each device (NAZA controller, ESCs and R/C receiver).

3.3 Hardware Architecture and Components

3.3.1 Quadcopter structure

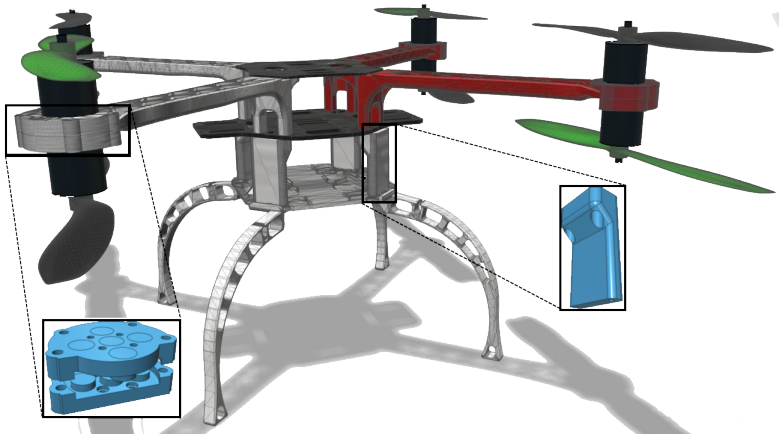

Figure 3.2: X8-VB Quadcopter 3D model (Brito, 2016).

The X8-VB Quadcopter was adapted from the DJI Flamewheel 450 kit by assembling additional pieces so that four extra rotors could be attached to the lower side of each arm, allowing this way, the fault tolerance aspect of the X8-VB Quadcopter. The result of this adaptation can be observed in Figure 3.2 along with its arm adapters and body extensions so that the propellers on

the bottom side of each arm won't intersect the drone's legs.

3.3.2 Attitude Controller

The DJI quadcopter kit provides the attitude controller NAZA-M Lite, which contains 3-axis gyroscope, 3-axis accelerometer and a barometer. This way, the controller can measure altitude and attitude and take action on the motors to maintain the structure stable at the desired state. The NAZA-M Lite is illustrated in Figure 3.3.

Figure 3.3: NAZA-M Lite attitude controller.

3.3.3 Arduino Uno and Due

The arduino Uno, illustrated in Figure 3.4, is arguably one of the most consumed micro controllers in the market due to its simplistic interaction between software and hardware. The software is easy to use and flexible enough for more complex tasks.

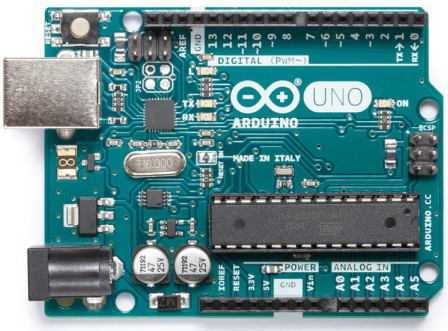

Figure 3.4: Arduino Uno (Arduino, 2018).

This Micro controller is equipped with analog and digital input and output pins that interface with various expansion boards, shields and other circuits. It interfaces with the computer with USB and can communicate with other devices through Serial (TX, RX), PWM, SPI and TWI. The pins 2 and 3 can also provide communication through interrupts (Arduino, 2018).

The arduino Due, illustrated in 3.5, provides a more powerful performance in many ways, when compared to arduino Uno. This comparison can be observed in 3.1.

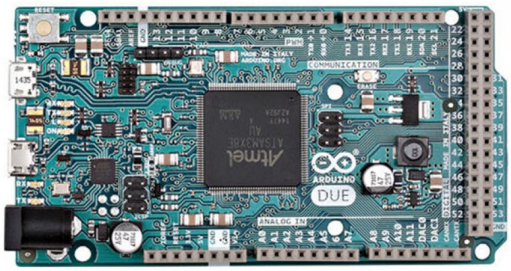

Figure 3.5: Arduino Due (Arduino, 2018).

Table 3.1: Arduino Uno versus Arduino Due specifications.

Arduino	Uno	Due
Processor	ATmega328P	ATSAM3X8E
CPU Speed [MHz]	16	84
Analog Pins(In/Out)	6/0	12/2
Digital Pins(In/Out)	14/6	54/12
SRAM [kB]	2	96
Flash [kB]	32	512

3.3.4 Radio Communication devices

3.3.4.1 Radio Controller and Receiver

In order to actuate on the NAZA controller with the desired control inputs, there needs to be a form of communication between the human and the quadcopter attitude controller. The acquired FrSky Taranis X9D Plus, illustrated in Figure, will serve for this purpose.

This radio Controller communicates with its receiver through radio signal, using its hopping technology that covers the whole 2.4 GHz band to provide a wide range and reliability in its communication. It features 8 output channels, allowing multiple important information to be acquired

Figure 3.6: FrSky Taranis X9D Plus on the left and its receiver on the right.

during the flight and displayed on the screen, namely: altitude, GPS coordinates, battery status.

The receiver provides the PWM inputs required by the NAZA controller: roll, pitch, throttle, yaw and mode.

3.3.4.2 Radio Frequency Communication

In order to extract real-time data from the quadcopter during the flight, a communication line via radio was deployed. The chosen device to do so was the 3DR radio V2 , illustrated in Figure 3.7 with the respective wiring, that communicates at 433 Mhz. The two modules of this device communicate in a two way full-duplex through adaptive TDM (Time Division Multiplexing), which enable transmitting and receiving of data, although it was used in a one way data transmition. The communication between the receptor and the arduino is done via serial at a 57600 baud rate.

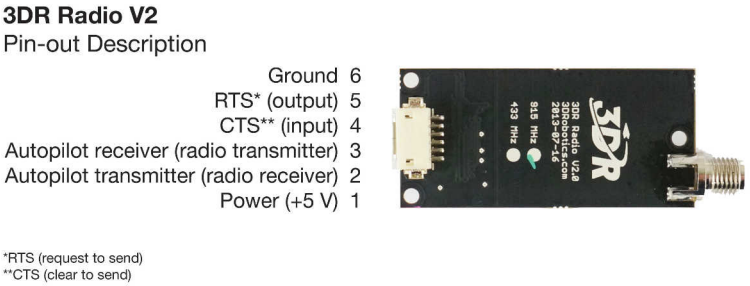

Figure 3.7: Wiring description of the 3DR radio V2.

3.3.5 GPS modules

3.3.5.1 Shield GPS Logger V2

The Shield GPS Logger V2, illustrated in Figure 3.8, has its GPS GTOP PA6B integrated along with a micro SD port for data logging. It provides several GPS related information through the NMEA (National Marine Electronic Association) protocol such as, time, coordinates, altitude, number of satellites, precision dilution, etc. Its specifications are listed in Table 3.2.

Figure 3.8: Shield GPS Logger V2.

Table 3.2: Shield GPS Logger V2 specifications.

Specifications	Value
Sensitivity [dBm]	-165
Frequency [Hz]	10
Channels	66
Current draw [mA]	20

3.3.5.2 Mini Locator RoyalTek REB-5216

This GPS module is capable to track the GPS and Glonass constelations at the same time, which was the main reason for its purchase. It also features 52 track verification channels and an active jammer removal to select the which constellation to track or both. To be noted that this device lacks documentation on how to use it, which is why it wasn't used in the sensor fusion algorithm.

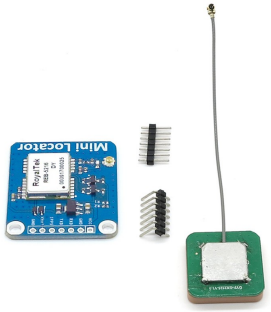

Figure 3.9: Mini Locator RoyalTek REB-5216.

3.3.5.3 Ultimate GPS Breakout V3

This GPS device is illustrated in Figure 3.10 and has the specifications listed in Table 3.3.

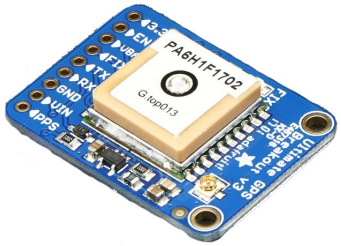

Figure 3.10: Ultimate GPS Breakout V3.

Table 3.3: Ultimate GPS Breakout V3 specifications.

Specifications	Value
Sensitivity [dBm]	-165
Frequency [Hz]	10
Channels	66
Current draw [mA]	20

3.3.5.4 NAZA GPS Module

The NAZA GPS module is illustrated in Figure 3.11 with its respective wiring description. This device provides both GPS and Compass data as soon as its powered, at a 115200 baud rate (115.2 KHz) with the PWM protocol. It sends GPS coordinates in the decimal degrees format exemplified in Table 2.1 after being decoded as explained further ahead in Section 4.3.3.

Figure 3.11: NAZA GPS wiring.

3.3.6 Absolute Orientation Sensor

The 9 Axes Motion Shield, illustrated in Figure 3.12, is based on the BNO055 absolute orientation sensor from Bosch Sensortec. It features a triaxial accelerometer, triaxial gyroscope and triaxial geomagnetic sensor.

Figure 3.12: 9 Axes Motion Shield.

3.3.7 Power Supply

The LiPo batteries were chosen to power the quadcopter, since this kind of power supply provides the power density needed to the brush-less DC motors. The power units used in this research are illustrated in Figure 3.13. The characteristics of the batteries are further described in Table 3.4.

The LiPo batteries have a particular volatile characteristic, so there should be always extra caution when handling, charging, discharging and storing theses batteries. If at least one cell's voltage of the battery reaches 3V or lower, the battery might not work as intended and even get

Figure 3.13: LiPo Batteries: 6000mAh on the left; 5000mAh on the right.

Table 3.4: LiPo batteries characteristics.

Battery	6.0	5.0
Capacity (mAh)	6000	5000
Discharge Rate (C)	25-50	25-35
Nº cells	4	4
Voltage per cell (V)	3.7	3.7

damaged to a point of ignition.

3.4 X8-VB Quadcopter Modelling

The dynamic and kinematic model of the quadcopter will be presented in this section, where the four inputs of the drone itself are correlated with the torques of each motor and consequently its six degrees of freedom. This model will consider the X8-VB architecture, which has eight motors instead of the usual four. As it can be observed in Figure 3.15, this quadcopter is shaped like a cross, with two rotors at the end of each arm, anti-parallel to each other.

Before the mathematical model of the system is described, some assumptions have to be considered regarding that the structure is rigid and symmetric, and that the quadcopter's position is described by reference coordinates. These reference coordinates are composed by a fixed and a mobile referentials. The fixed referential is placed on earth (usually the starting coordinate of the quadcopter) and the mobile referential is placed at the center of mass of the system, both described by the NED ($North - East - Down$) system as observed in Figure 3.14 (Figueiredo et al., 2014).

The linear ξ (3.1) and angular γ (3.2) positions can be obtained by measuring the displacement

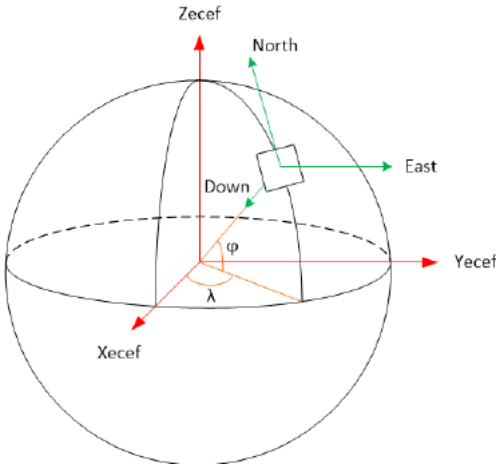

Figure 3.14: *North* − *East* − *Down* coordinate system (Figueiredo et al., 2014).

of the mobile referential related with the fixed referential. The mobile referential also provides the linear $\dot{\xi}$ and angular $\dot{\gamma}$ velocities, and forces f_n and torque t.

$$\xi = [x\ y\ z]^T \tag{3.1}$$

$$\gamma = [\phi\ \theta\ \psi]^T \tag{3.2}$$

where x, y and z represent the position of the center of mass and ϕ, θ and ψ its orientation.

In order to perform any rotational or linear movement, each rotors' torque must be varied accordingly. Each input U_1, U_2, U_3, U_4 is responsible for a combination of all eight rotors' thrust so that the quadcopter can change its attitude and move in any direction (Brito, 2016).

Table 3.5: Inputs of the Quadcopter X8.

Input	Effect	Movement
$U_1 = \sum_{n=1}^{8} f_n$	Throttle	Z axis
$U_2 = f_4 + f_8 - f_2 - f_6$	Roll	Y axis
$U_3 = f_3 + f_7 - f_1 - f_5$	Pitch	X axis
$U_4 = f_1 + f_3 + f_6 + f_8 - f_2 - f_4 - f_5 - f_7$	Yaw	-

The input vector $u = [U_1\ U_2\ U_3\ U_4]^T$ contains the four control inputs commonly known as throttle (U_1), roll (U_2), pitch (U_3) and yaw (U_4), which introduce torque contributions f_n into the respective n motors. The throttle is represented by equation (U_1) in Table 3.5 which contributes equally in all rotors in order to move in the Z axis. The Roll is represented in equation (U_2) which allows the quadrotor to perform a roll (ϕ) rotation represented as red in Figure 3.15, which allows the quadcopter to move in the y axis. The Pitch in equation (U_3) will provide the pitch (θ) rotation represented as green in the same Figure 3.15, which enables the movement in the x axis. The last equation (U_4) in Table 3.5 allows the drone to do a yaw (ψ) rotation around the vertical axis, also

represented in Figure 3.15 as blue.

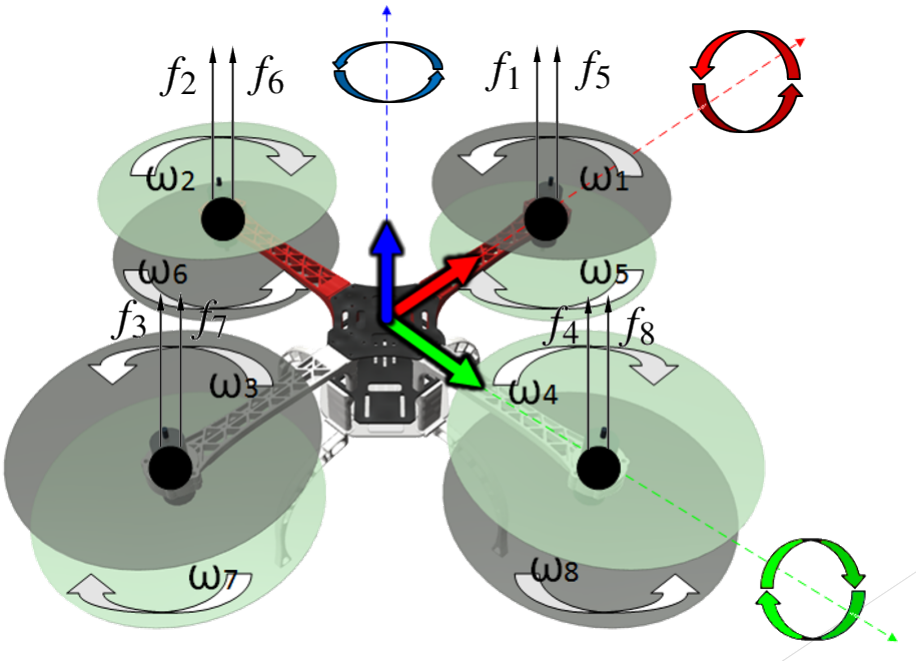

Figure 3.15: The three angular degrees of freedom of the quadcopter. The XYZ axis are represented with RGB colors, respectively (Brito, 2016).

The angular position or attitude (γ) is defined by the three rotations (ϕ, θ and ψ) of the mobile referential related with the fixed referential. Each of these rotations is mathematically described by the rotation matrices (Lee et al., 2009):

- Roll rotation matrix (3.3);

$$R_\phi = \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos(\phi) & -sin(\phi) \\ 0 & sin(\phi) & cos(\phi) \end{bmatrix} \tag{3.3}$$

- Pitch rotation matrix (3.4);

$$R_\theta = \begin{bmatrix} cos(\theta) & 0 & sin(\theta) \\ 0 & 1 & 0 \\ -sin(\theta) & 0 & cos(\theta) \end{bmatrix} \tag{3.4}$$

- Yaw rotation matrix (3.5).

$$R_\psi = \begin{bmatrix} cos(\psi) & -sin(\psi) & 0 \\ sin(\psi) & cos(\psi) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.5}$$

Therefore, the inertial position coordinates and the body reference coordinates can be obtained from the product of the three elementary rotation matrices, resulting in R_γ 3.6.

$$R_\gamma = \begin{bmatrix} cos(\theta)cos(\psi) & sin(\phi)sin(\theta)cos(\psi) - cos(\phi)sin(\psi) & cos(\phi)sin(\theta)cos(\psi) + sin(\phi)sin(\psi) \\ cos(\theta)sin(\psi) & sin(\phi)sin(\theta)sin(\psi) + cos(\phi)cos(\psi) & cos(\phi)sin(\theta)sin(\psi) - sin(\phi)cos(\psi) \\ -sin(\theta) & sin(\phi)cos(\theta) & cos(\phi)cos(\theta) \end{bmatrix} \tag{3.6}$$

Respecting the Newton-Euler equations of motion of a rigid body and defining the vectors containing the linear and angular position of the earth referential (3.7) and of the quadcopter referential (3.8), these are linked by equations (3.9) and (3.10), respectively.

$$\Upsilon = [\xi \ \gamma] = [x \ y \ z \ \phi \ \theta \ \psi]^T \tag{3.7}$$

$$\Lambda = [\zeta \ \chi] = [u \ v \ w \ p \ q \ r]^T \tag{3.8}$$

$$\dot{\xi} = R \cdot \zeta \tag{3.9}$$

$$\dot{\gamma} = T \cdot \chi \tag{3.10}$$

where T is the angular transformation matrix 3.11

$$T = \begin{bmatrix} 1 & sin(\phi)tan(\theta) & cos(\phi)tan(\theta) \\ 0 & cos(\phi) & -sin(\phi) \\ 0 & \frac{sin(\phi)}{cos(\theta)} & \frac{cos(\phi)}{cos(\theta)} \end{bmatrix} \tag{3.11}$$

The kinematic model of the quadcopter is described by the set of equations (3.12).

$$\begin{cases} \dot{x} = w[sin(\phi)sin(\psi) + cos(\phi)cos(\psi)sin(\theta)] - v[cos(\phi)sin(\psi) - cos(\psi)sin(\phi)sin(\theta)] + u[cos(\psi)cos(\theta)] \\ \dot{y} = v[cos(\phi)cos(\psi) + sin(\phi)sin(\psi)sin(\theta)] - w[cos(\psi)sin(\phi) - cos(\phi)sin(\psi)sin(\theta)] + u[cos(\theta)sin(\psi)] \\ \dot{z} = w[cos(\phi)cos(\theta)] - u[sin(\theta)] + v[cos(\theta)sin(\phi)] \\ \dot{\phi} = p + r[cos(\phi)tan(\theta)] + q[sin(\phi)tan(\theta)] \\ \dot{\theta} = q[cos(\phi)] - r[sin(\phi)] \\ \dot{\psi} = r[\frac{cos(\phi)}{cos(\theta)}] + q[\frac{sin(\phi)}{cos(\theta)}] \end{cases} \tag{3.12}$$

Now taking in consideration the forces, moments and gyroscopic effects acting on the structure's mass already explained in (Brito, 2016), the dynamical and kinematic model of the drone is described by the set of equations (3.13).

$$\begin{cases} \ddot{x} = \Big(sin(\phi)\sin(\psi) - cos(\phi)cos(\psi)sin(\theta)\Big)\frac{U_1 - F_{Dx}}{m} \\ \ddot{y} = \Big(cos(\psi)\sin(\phi) + cos(\phi)sin(\psi)sin(\theta)\Big)\frac{U_1 - F_{Dy}}{m} \\ \ddot{z} = -g + \Big(cos(\phi)\cos(\theta)\Big)\frac{U_1 - F_{Dz}}{m} \\ \ddot{\phi} = \frac{1}{I_{xx}}\Big((I_{yy} - I_{zz})\dot{\theta}\,\dot{\psi} - J_t\dot{\theta}\,\omega + U_2\Big) \\ \ddot{\theta} = \frac{1}{I_{yy}}\Big((I_{zz} - I_{xx})\dot{\phi}\,\dot{\psi} - J_t\dot{\phi}\,\omega + U_3\Big) \\ \ddot{\psi} = \frac{1}{I_{zz}}\Big((I_{xx} - I_{yy})\dot{\phi}\,\dot{\theta} + U_4\Big) \end{cases} \tag{3.13}$$

Where g is the gravity acceleration, m is its mass, I_{xx}, I_{yy} and I_{zz} represent the body inertia in each linear degree of freedom, J_t is the rotor inertia, ω is the resulting angular velocity in radians per second of all rotors and, F_{Dx}, F_{Dy} and F_{Dz} are the drag forces, contrary to its movement,

$$F_D = C\frac{\rho v^2}{2}A; \tag{3.14}$$

in which C refers to the drag coefficient, ρ the air density and A the impact area with air.

Figure 3.16 illustrates how the input u is translated into the torques of each motor which will affect the quadcopter attitude and position, considering faults and eventual perturbations of the wind.

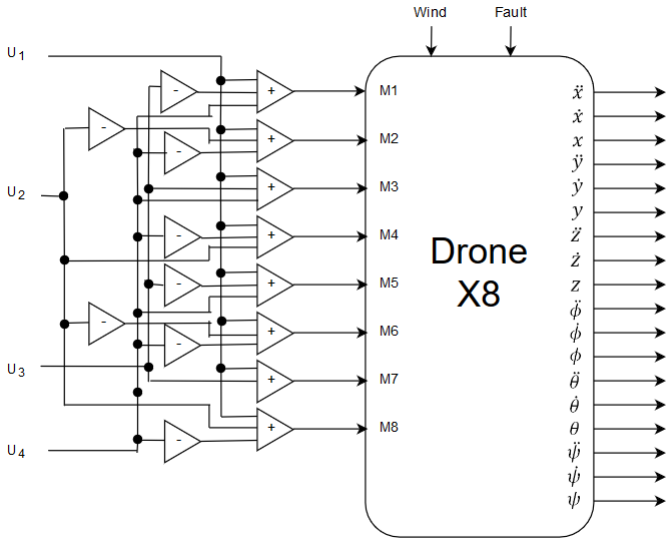

Figure 3.16: Quadcopter inputs translated to each *n* motor's torque

CHAPTER 4

IDENTIFICATION AND CONTROL

This chapter will present the theory behind the three major thematics of this dissertation: the identification of the mathematical model of the NAZA® attitude controller; trajectory control algorithms on the quadcopter X8; and the sensor fusion of the GPS modules.

4.1 NAZA Attitude Controller Identification

In order to identify the NAZA® controller, several experiments were done by applying the PWM inputs (roll, pitch, throttle and yaw) and observing its behavior (angle variations and rotor speeds).

As mentioned in Section 3.4 the control inputs $u = [U_1\ U_2\ U_3\ U_4]^T$ will apply their respective contributions to their respective motors. By varying these inputs slowly and for a long time, covering the whole spectrum of possible values and observing the right variables, it should be possible to identify the controller and replicate it as a model through NARX or Hammerstein-Wienner methods. The variables to be observed, in this case, should be the PWM values that the controller provides to the ESCs and the change in the orientation.

One of the experiments consists in maintaining every input idle except for the pitch input. The pitch value is changed between 1000 (inclined backwards) and 2000 (inclined forward) in PWM with small steps (5 PWM values) each second, covering all the spectrum of possible values. The PWM values given by the controller to the ESCs is measured between values 1000 (slowest rotor speed) and 2000 (fastest rotor speed). The orientations along time are also measured with a gyroscope.

As clarified by the pitch equation (U_3) in Table 3.5, any variations in U_3 will affect the motors

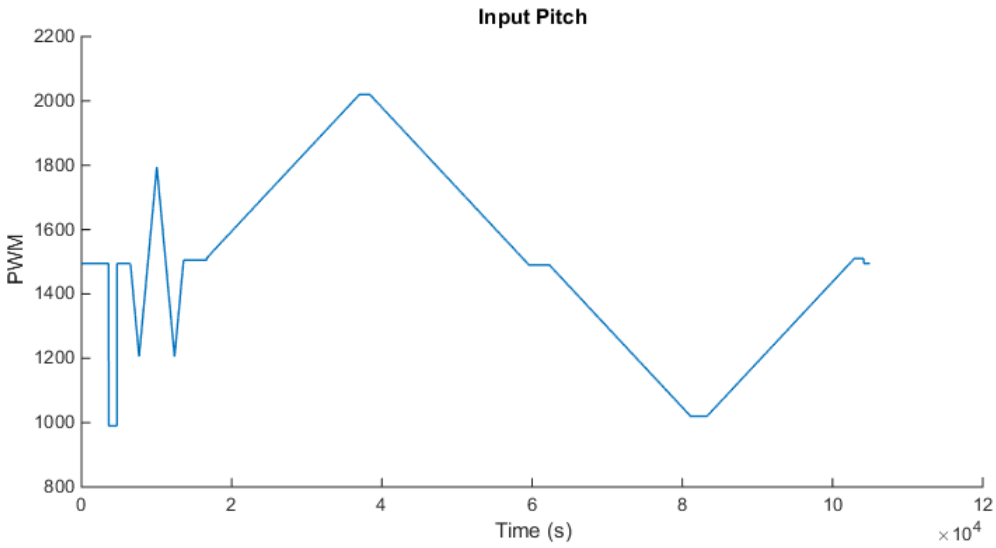

Figure 4.1: Pitch PWM values given to the NAZA controller.

1, 3, 5 and 7 accordingly. Since these contributions are equal or symmetric between rotors, the response of one of these motors is enough to model the pitch. A similar approach will happen for the other inputs U_1, U_2 and U_4.

In Figure 4.1, the begining of the signal where the PWM value reaches 1000 is related with the start of the motors. The start of the motors with the NAZA® controller happens when the throttle and pitch are held down (1000 PWM value) and the yaw and roll are held right or left (1000 or 2000 PWM values) for three seconds. Right after that and before the 1800 seconds there is a calibration process so that the nonlinearities of the NAZA® controller won't go to instability (the controller would apply high values of PWM and the motors would rotate really fast). The rest of the signal are the slow variations (5 PWM values) to cover all the possible values of pitch as explained earlier.

The measured orientation of the quadcopter when performing the pitch commanded along time can be observed in Figure 4.2. As it can be observed, the 9 Axes Motion Shield detected the pitch movements as well as the peak variations that the NAZA® controller causes sometimes. These peaks can be observed in the graphic 4.2 and are not desired for the modeling of the controller.

From the pitch experiments on the attitude controller, the signal given to motor 3 and 5 is illustrated in Figure 4.3. The response of the controller to the oposed motors(1 and 7) would have a symetric effect, in the sense that from 4000 to 7000 second, the signal would present higher PWM values (around 1250) and from 7000 to 10000 second it would stay as 1175 PWM values. This response is very noisy and inconsistent, which is also not desired for the controller modeling.

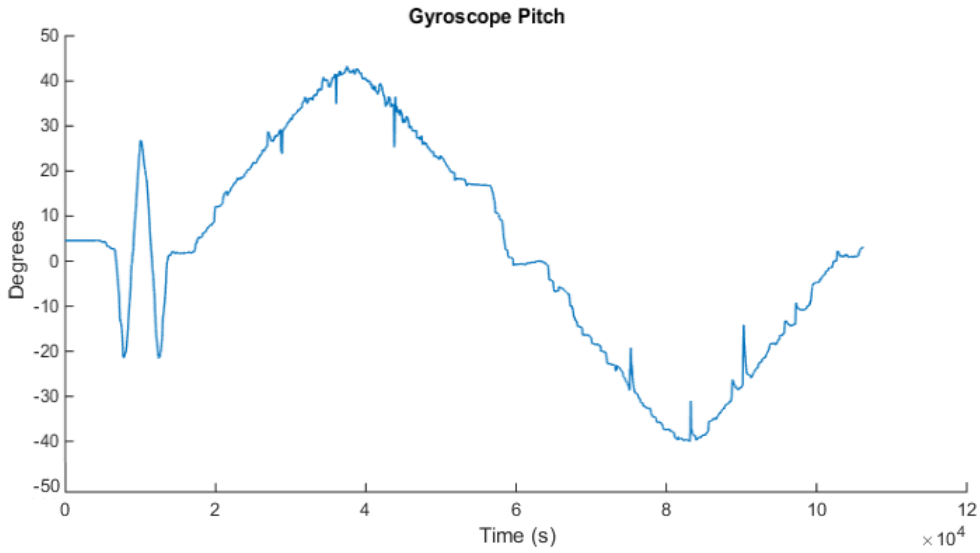

Figure 4.2: Pitch orientation of the structure along time.

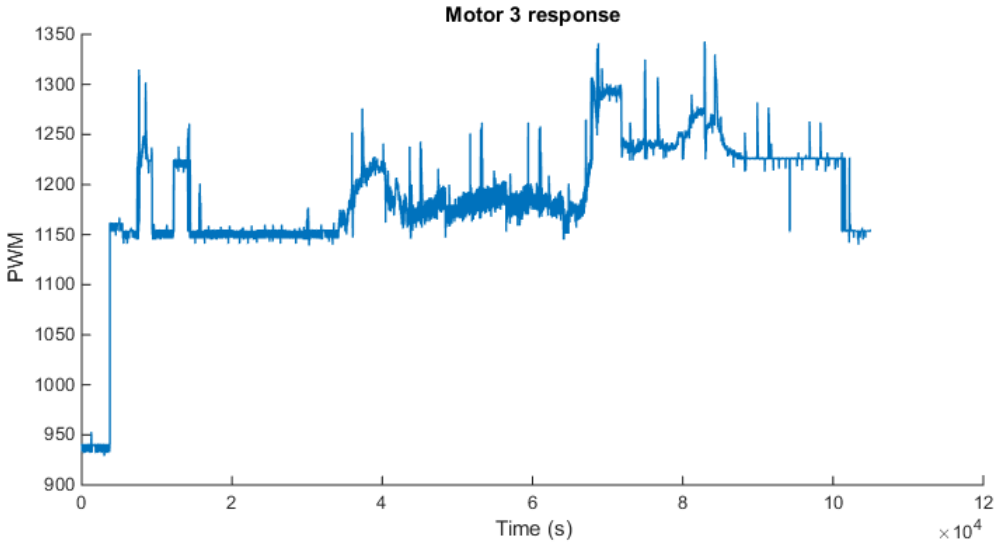

Figure 4.3: Response of the NAZA controller to the motor 3.

4.2 Trajectory Control Algorithms

The quadcopters must have an integrated attitude controller that stabilizes the structure horizontally with the rotors pointing up, by actuating on each of these rotors individually in order to maintain it at the equilibrium point. Essentially, the attitude controller maintains the quadcopter in hover state, where all of the system variables and inputs are constant. This attitude controller accepts u as the set of inputs, which is usually remotely controlled by a human. One of the main objectives of this thesis is to substitute the human in the loop for another controller that will change the inputs u to drive the quadcopter along a predefined trajectory. This method is called cascade

control, Figure 4.4, where the attitude controller will be controlled by a subsequent trajectory controller.

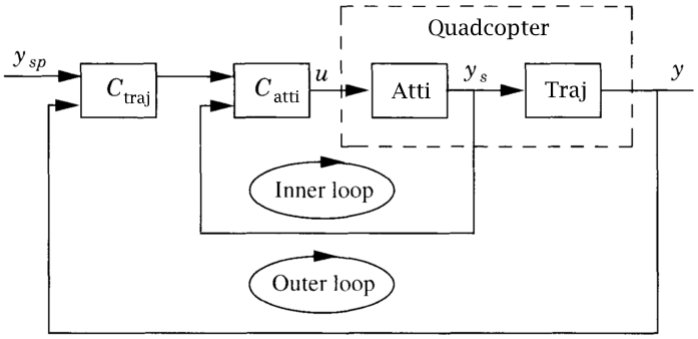

Figure 4.4: Block Diagram of the cascade control integrated in the quadcopter. Adapted from (Åström and Hägglund, 2006).

A well tuned and robust attitude controller is desired so that the outer loop won't be able to bring the whole system into instability. The tuning process of this control scheme follows the steps:

1. Tune the inner loop controller according to its process variable;
2. Set the inner loop controller to automatic with internal setpoint;
3. Tune the outer loop controller according to its process variable;
4. Switch the inner loop to external setpoint;
5. Switch the outer loop to automatic;

If this procedure is not followed properly, undesired switching transients may arise and corrupt the well functioning of the closed-loop system (Åström and Hägglund, 2006).

In this work, the attitude controller proposed by (Brito, 2016) was used as the inner loop and all the trajectory controllers presented in this section took its behavior into consideration. This attitude controller consists of a PID for each angular degree of freedom γ. The symmetry of the quadcopter is also true for its movement in the x and y axis, in the sense that the control algorithms developed for trajectory tracking in the x axis by varying the pitch (θ) can be replicated to control the movement in y by varying the roll (ϕ). Although this is true for the PID and State Space Controllers, the Differential Flatness is particularly different and this symmetry is explained further ahead in section 4.2.3. In all the trajectory controller, the yaw was stabilized by the attitude controller, which maintained it at zero degrees at all times.

4.2.1 Proportional Derivative Integral Control

The first controller designed to perform trajectory tracking of the quadcopter was the PID. The PID presents the most simple form of efficient control of any system, so as the first control attempt to apply on a system in a cascade architecture, it appeared to be a good starting tool. The PID also makes it easier to understand possible transients or inconsistencies happening in the cascade control loop, and it doesn't require the full knowledge or understanding of this very complex system that is the quadcopter. Three PIDs were designed to control each linear degree of freedom $\xi = [x\ y\ z]^T$ separately. This happens by comparing the distance between the reference and the actual position through the PID, providing a reference for the attitude controller. The PID architecture designed is illustrated in Figure 4.5.

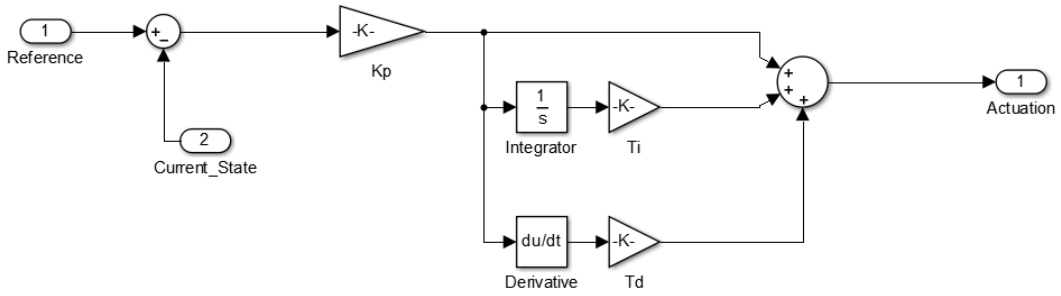

Figure 4.5: PID architecture used for trajectory tracking of the quadcopter.

4.2.1.1 Forward, Backward and Sideways Movement

The attitude controller actuates on the motors so that a pitch (θ) rotation is performed to move in the x axis, and similarly, a roll (ϕ) rotation to move in the y axis. Due to symmetry of the structure, the PID controller designed to move the quadcopter in the x axis through pitch (θ) actuations could be replicated for the movement in the y axis through (ϕ) actuations.

The PIDs were tuned by following the Ziegler-Nichols ultimate-sensitivity method described in section 2.5.1.1. The gains obtained were $K = 0.0001$, $T_i = 0.02$ and $T_d = 300$. The performance of this controller while guiding the quadcopter through the several predefined setpoints is illustrated in Figure 4.6, and the respective attitude references and actuation of the drone are represented in Figure 4.7. This controller presents a slow rising time and undesired overshoot with the gains obtained (at its best). It takes 10 seconds for the quadcopter to move 5 meters forward which is slow. From these figures and from the fact that the PID is a pure feedback methodology, its easy to understand why it is not adequate for this kind of problem.

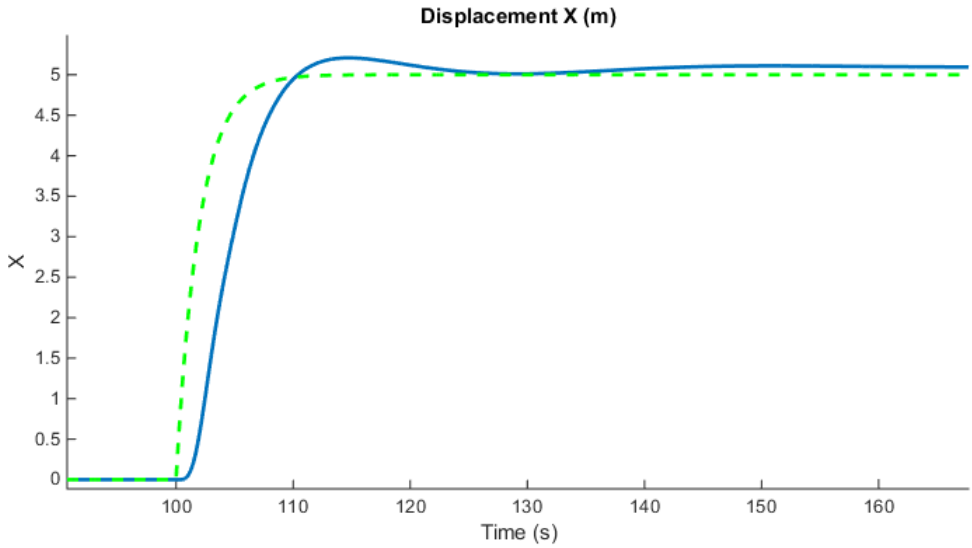

Figure 4.6: Trajectory tracking in the x axis with the PID controller (dotted green line is the reference and blue line is the actual trajectory).

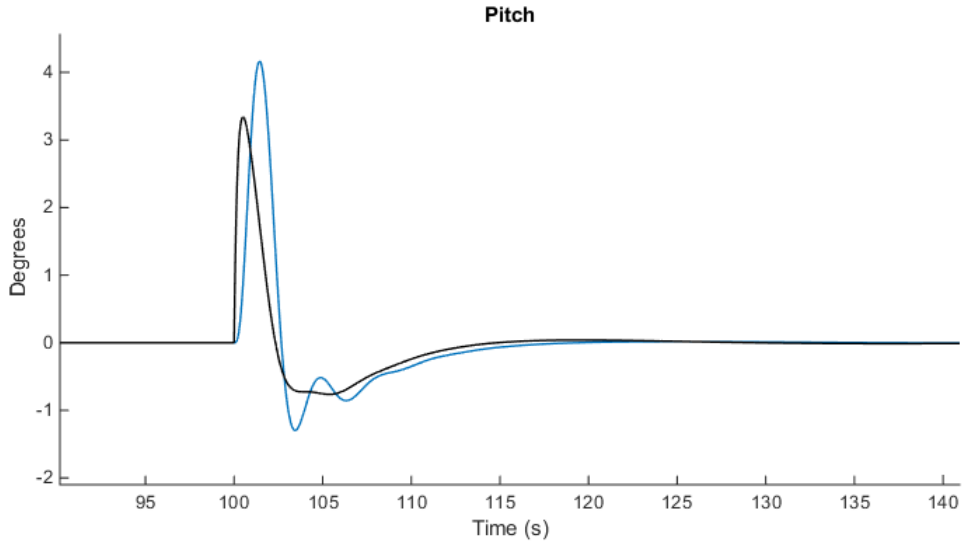

Figure 4.7: Pitch performed by the quadcopter with the PID controller (black line is the reference and blue line is the actual pitch).

4.2.1.2 Movement in Altitude

The quadcopter movement in the z axis is caused by varying the input U_1 which is directly related with the increase or decrease of the collective torque of the motors. The PID was designed to actuate on U_1 depending on the difference between the reference altitude and the current altitude. The gains obtained were: $K = 33$, $T_i = 0.1$ and $T_d = 6$. Figure 4.8 illustrates the performance of the PID while tracking the altitude reference. Just like the PIDs designed for the x and y movement, this one presents slow rising time and high overshoot.

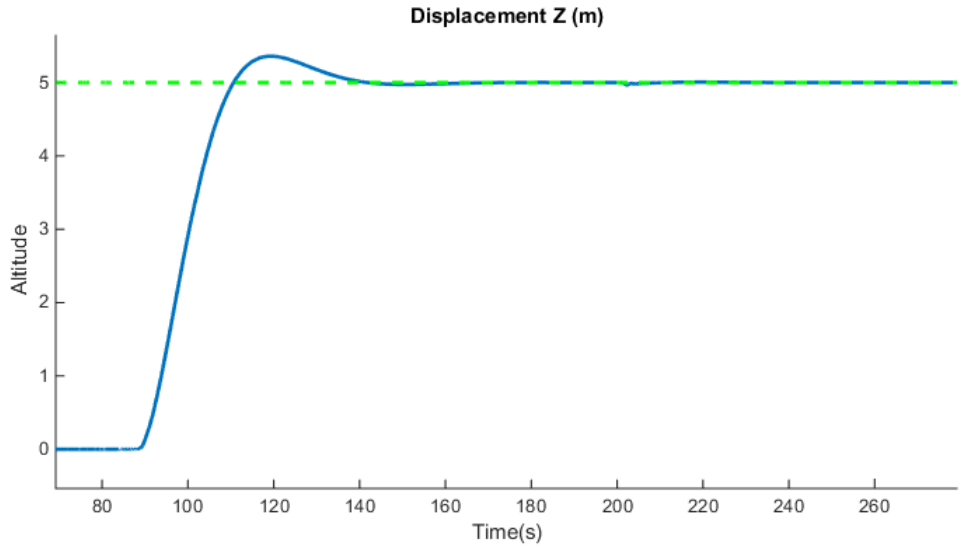

Figure 4.8: Trajectory tracking in the z axis with the PID controller (dotted green line is the reference and blue line is the actual trajectory).

4.2.2 State Space Feedback Control

Considering the set of equations (3.13) that describe the dynamic and kinematic open-loop system that is the quadcopter, its representation in state space was produced in order to posteriorly close the loop with a control algorithm. The objective is to control the altitude acceleration $\ddot{z}$ and angular accelerations $\ddot{\gamma} = [\ddot{\phi}\ \ddot{\theta}\ \ddot{\psi}]^T$, so that the $\xi = [x\ y\ z]^T$ and ψ of the quadcoper is changed to the desired state. The system to be controlled in this problem is nonlinear and as such, it has to be linearized about an operating point (Maccarleyi, 1984). The chosen operating point for the quadcopter was its hover state, characterized by equations (4.1) (4.2) (4.3) and (4.4),

$$\Upsilon_0 = [x_0\ y_0\ z_0\ 0\ 0\ 0]^T \tag{4.1}$$

$$\dot{\Upsilon}_0 = [0\ 0\ 0\ 0\ 0\ 0]^T \tag{4.2}$$

$$\ddot{\Upsilon}_0 = [0\ 0\ -g\ 0\ 0\ 0]^T \tag{4.3}$$

$$u_0 = [g\ 0\ 0\ 0]^T \tag{4.4}$$

where Υ_0 is the equilibrium point of the quadcopter at hover state, $\dot{\Upsilon}_0$ and $\ddot{\Upsilon}_0$ its derivatives (velocity and acceleration), u_0 the input values at hover state and g is the gravitational force. With this operating point in mind, the next step is to linearize the system's equations (3.13) around it. This

linearization comes from the fact that the system's equations are heavily dominated by trigonometric functions. Since the difference between the argument and the result of the trigonometric functions are negligible, these can be approximated to the argument when working with small values of roll, pitch and yaw. This way, the following assumptions took place: $\sin(\alpha) \approx \alpha$ and $\cos(\alpha) \approx 1 - \frac{\alpha^2}{2}$, (Boas, 2005). With the linearized model, one can develop each linear degree of freedom in state space, along with the respective angular degree of freedom that directly affects it.

4.2.2.1 Forward, Backward and Sideways Movement

The quadcopter movement in the x axis is caused by changing its pitch (θ) orientation (assuming the other attitude variables $\phi = \psi = 0$ for simplification), then its state space representation is described in (4.10).

$$\begin{bmatrix} \dot{x} \\ \ddot{x} \\ \dot{\theta} \\ \ddot{\theta} \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & \frac{U_1 - F_{Dx}}{m} & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x \\ \dot{x} \\ \theta \\ \dot{\theta} \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0 \\ \frac{1}{I_{yy}} \end{bmatrix} U_3 \tag{4.5}$$

The controllability matrix is then calculated to verify that (2.12) is true:

$$rank[B \mid AB \mid A^2B \mid A^3B] = rank \begin{bmatrix} 0 & 0 & 0 & \frac{U_1 - F_{Dx}}{mI_{zz}} \\ 0 & 0 & \frac{U_1 - F_{Dx}}{mI_{zz}} & 0 \\ 0 & \frac{1}{I_{zz}} & 0 & 0 \\ \frac{1}{I_{zz}} & 0 & 0 & 0 \end{bmatrix} = 4 \tag{4.6}$$

After confirming that the movement in the x axis is controllable by pitch actuations, the LQR method was resorted to find the optimal control gain matrix by minimizing the quadratic cost function (2.14). In this system, Q is the matrix that atributes weights to each of the state variables and R is the weight of the input u. These values were tuned according to the system dynamics contraints, how fast is the response of each state variable (Q) and how aggressive is the input signal (R). After some iterative attempts and consideration for the energy consumption and flight time, the values that presented the best results were:

$$Q = \begin{bmatrix} 0.2 & 0 & 0 & 0 \\ 0 & 0.5 & 0 & 0 \\ 0 & 0 & 0.01 & 0 \\ 0 & 0 & 0 & 0.01 \end{bmatrix} \tag{4.7}$$

$$R = 100 \tag{4.8}$$

by calculating the gain matrix K through (2.15) and (2.16), which resulted in $K = [0.6 \;\; 1.5 \;\; 9 \;\; 0.6]^T$.

As explained in section 4.2, due to symmetry of the system, the controller for the movement in the x axis by changing the pitch (θ) will be replicated for the movement in the y axis by varying the roll (ϕ). The trajectory tracking of the state space feedback controller is illustrated in 4.9 and its pitch orientation along the time can be observed in 4.10.

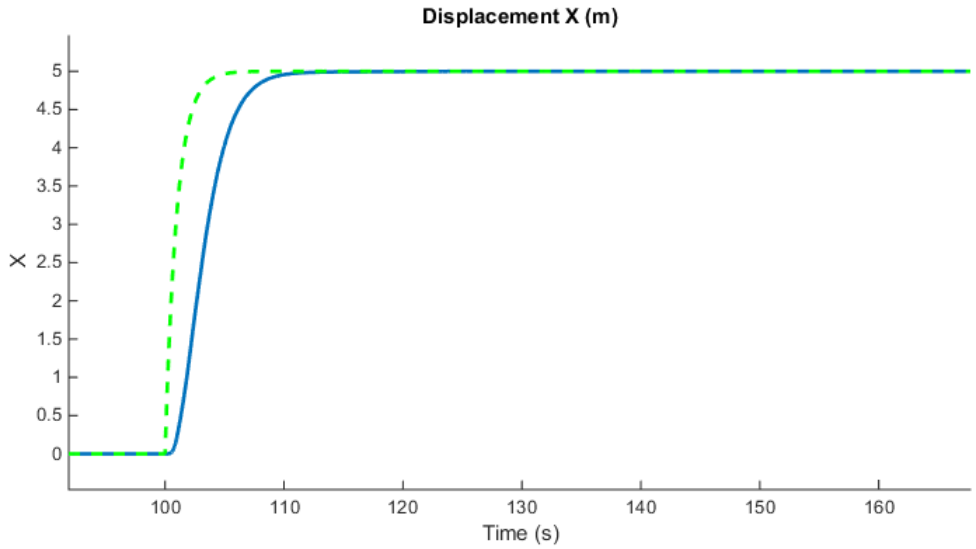

Figure 4.9: Trajectory tracking in the x axis with the state space feedback controller (dotted green line is the reference and blue line is the actual trajectory).

From Figure 4.9 it can be understood that the controller performs well in tracking the desired reference with less than 10 seconds of rising time and absolute dampness of the overshoot. Figure 4.10 presents an oscillatory actuation of the pitch due to the correlation between the attitude controller and the trajectory controller. Nevertheless, this pitch actuation guides the quadcopter fast and smooth to the 5 meter reference.

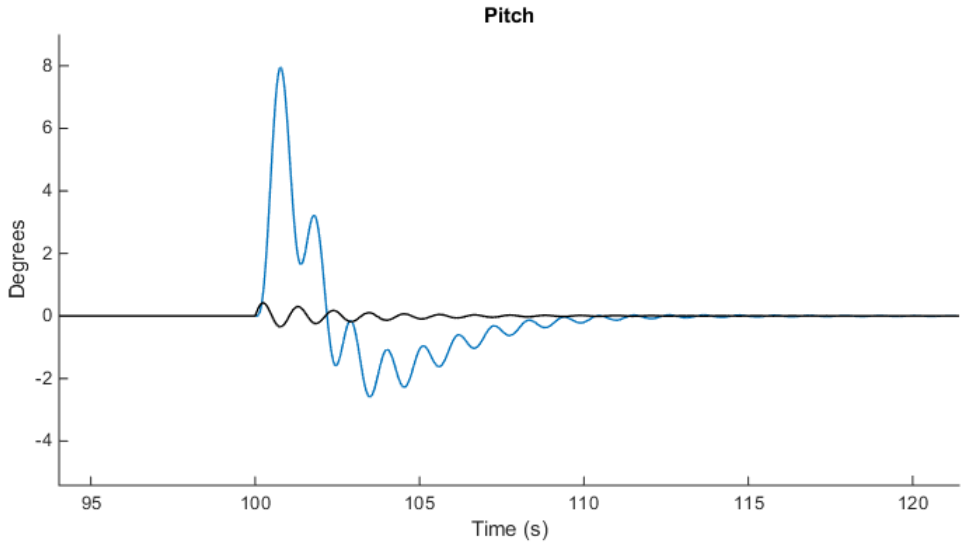

Figure 4.10: Pitch performed by the quadcopter with the state space feedback controller (black line is the reference and blue line is the actual pitch).

4.2.2.2 Movement in Altitude

The quadcopter movement in the z axis is caused by varying the input U_1 which is directly related with the increase or decrease of the collective torque of the motors. The state space representation is described in (4.9).

$$\begin{bmatrix} \dot{z} \\ \ddot{z} \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} z \\ \dot{z} \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{1}{m} \end{bmatrix} U_1 \tag{4.9}$$

The controllability matrix is then calculated to verify that (2.12) is true:

$$rank[B \mid AB] = rank \begin{bmatrix} 0 & \frac{1}{m} \\ \frac{1}{m} & 0 \end{bmatrix} = 2 \tag{4.10}$$

After confirming that the movement in the z axis is controllable, the LQR method was resorted to find the optimal control gain matrix K. After some iterative attempts and consideration for the energy consumption and flight time, the values that presented the best results were:

$$Q = \begin{bmatrix} 1 & 0 \\ 0 & 10 \end{bmatrix} \tag{4.11}$$

$$R = 0.01 \tag{4.12}$$

by calculating the gain matrix K through (2.15) and (2.16), which resulted in $K = [10 \;\; 32.4037]^T$.

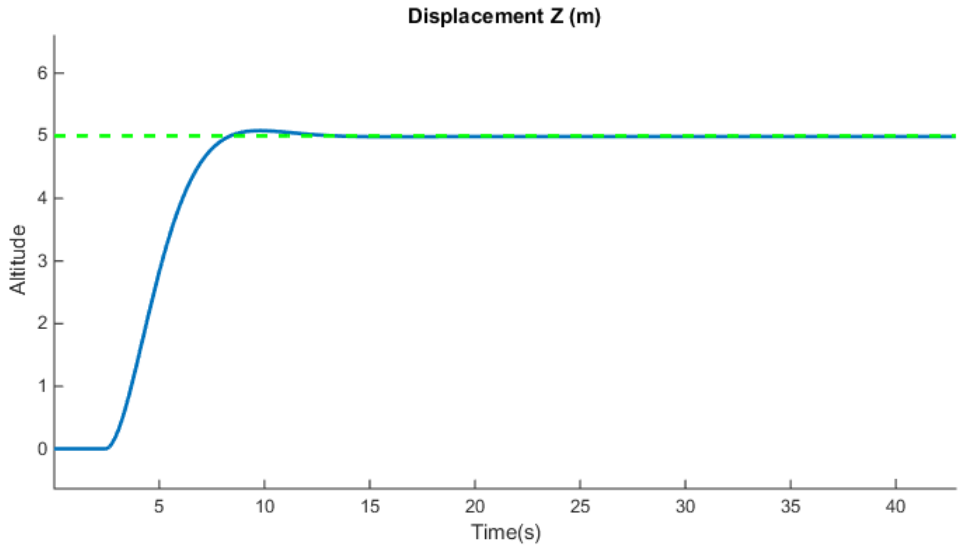

Figure 4.11: Trajectory tracking in the z axis with the state space feedback controller (dotted green line is the reference and blue line is the actual trajectory).

The trajectory tracking of the state space feedback controller is illustrated in Figure 4.11. It presents a rising time of 5 seconds and negligible overshoot, which is adequate for this kind of problem where speed and precision in trajectory tracking is crucial to avoid obstacles.

4.2.3 Differential Flatness Controller

From the set of equations (3.13) that describe the kinematics and dynamics of the quadcopter and from the intput u equations in Table 3.5, one can obtain the control vector u expressed in terms of the system's states and their derivatives, as represented in equation (4.13).

$$\begin{cases} U_1 = \sqrt{\ddot{x}_{ref}^2 + \ddot{y}_{ref}^2 + (g + \ddot{z}_{ref})^2} \\ U_2 = \ddot{\phi}_{ref} \\ U_3 = \ddot{\theta}_{ref} \\ U_4 = \ddot{\psi}_{ref} \end{cases} \tag{4.13}$$

This means that input U_1 is related with the desired accelerations in the three linear degrees of freedom and inputs U_2, U_3 and U_4 are tied with the desired angular accelerations, respectively.

Since the quadcopter is differentially flat (Mellinger, 2011), the attitude variables ϕ and θ can be calculated from (3.13), by following the properties of differential flatness described in section 2.5.4:

$$\phi_{ref} = \arcsin\left(\frac{\ddot{y}_{ref}}{\sqrt{\ddot{x}_{ref}^2 + \ddot{y}_{ref}^2 + (g + \ddot{z}_{ref})^2}}\right) \tag{4.14}$$

$$\theta_{ref} = \arctan\left(\frac{-\ddot{x}_{ref}}{g+\ddot{z}_{ref}}\right) \tag{4.15}$$

and after some calculations, the first and second derivatives of (4.14) and (4.15) can be obtained and hereby complete the set of control inputs u needed for the differential flatness controller (Formentin and Lovera, 2011).

In order to compute any control input u_{df}, there needs to be a smooth reference that brings the system from one state to another. For this reason, a trajectory generator was developed, in which the various setpoints are defined in space and time, and it's the trajectory generator's task to create smooth functions that should be the optimal or quasi-optimal references for the linear position ξ_{ref} and its derivatives $\ddot{\xi}_{ref}$.

The trajectory generator is responsible to create ideal trajectories through each couple of setpoints. The basis functions' technique was used in this approach, in which the initial and final state must be declared, in order to find the ideal coefficients that drive the basis functions into the optimal reference trajectory. As mentioned in section 2.5.4.1 the monomial basis functions were chosen for this problem, which can be expressed as equation (2.21). Since the quadcopter has a total of six degrees of freedom, at least six monomials ($M = 5$) are recommended for full approximation. Starting by approximating the second derivative of each component of ξ results in equation (4.16).

$$\ddot{\xi}(t) = a_{x2} + a_{x3}t + a_{x4}t^2 + a_{x5}t^3 = \sum_{n=2}^{M} a_k t^{n-2} \tag{4.16}$$

Their respective velocity $\dot{\xi}$ and position ξ can be obtained by simple integrations. The coefficients can be obtained by solving equation (4.16) in order to a_k.

Defining the position, velocity and acceleration of the aircraft for a specific linear degree of freedom in matrix (4.17),

$$X = \begin{bmatrix} X_0 \\ \dot{X}_0 \\ \ddot{X}_0 \\ X_f \\ \dot{X}_f \\ \ddot{X}_f \end{bmatrix} \tag{4.17}$$

where X_0 and X_f are the starting and final positions, respectively. And defining the time t_f that requires the quadcopter to move from X_0 to X_f, the time matrix (4.18) can be developed so that the coefficients a_{xi} can be calculated.

$$T = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & t_f & \frac{t_f^2}{2} & \frac{t_f^3}{6} & \frac{t_f^4}{12} & \frac{t_f^5}{20} \\ 0 & 1 & t_f & \frac{t_f^2}{2} & \frac{t_f^3}{3} & \frac{t_f^4}{4} \\ 0 & 0 & 1 & t_f & t_f^2 & t_f^3 \end{bmatrix} \tag{4.18}$$

Matrix A, which contains all the coefficients a_{xi}, can be calculated through $TA = X$ and thereafter, included in $\ddot{\xi}(t)$ 4.16 and its integrations $\dot{\xi}(t)$ and $\xi(t)$. Given these references, a control input $u_{df} = \ddot{\gamma}_{ref}$ can be generated from the differential flatness directly to the quadcopter, bypassing the attitude controller through a switch. This switch will let the state space feedback and attitude controller provide its u_{ss} only when the differential flatness controller is idle. The differential flatness arquitecture is illustrated in Figure 4.12.

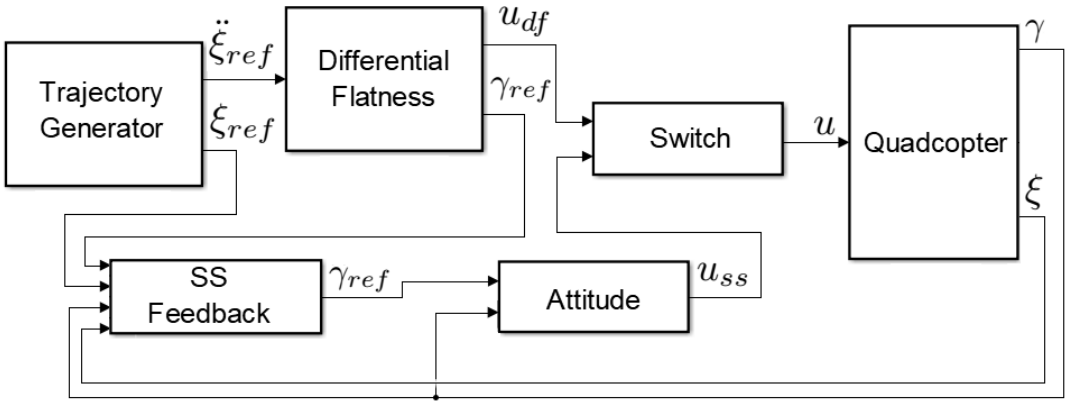

Figure 4.12: Differential Flatness controller with State Space Feedback as the feedback stabilizer.

This kind of feedforward controller is indeed not sufficient to control a volatile and unstable system like this, as any perturbation from the wind or from other sources will deviate the quadcopter's state (ξ, γ) from the desired one (ξ_{ref}, γ_{ref}). To solve this inconvinience, a state space feedback controller was added into the control loop, correcting any deviation from the desired state while also providing stability.

4.2.3.1 Forward, Backward and Sideways Movement

The reference functions for sideways movement will be produced the same way as the forward and backward reference functions, so the following descriptions of the trajectory generator algorithm can be understood for the movements in both degrees of freedom. The first step is to define the starting X_0 position, the ending X_f position and the time t_f to move from one position to the

other. With $X_0 = 0$, $X_f = 5$ and $t_f = 5$ it was obtained the ξ_{ref} and $\ddot{\xi}_{ref}$ references (Figure 4.13) that brings the drone from one position to the other while respecting the starting and ending points' conditions.

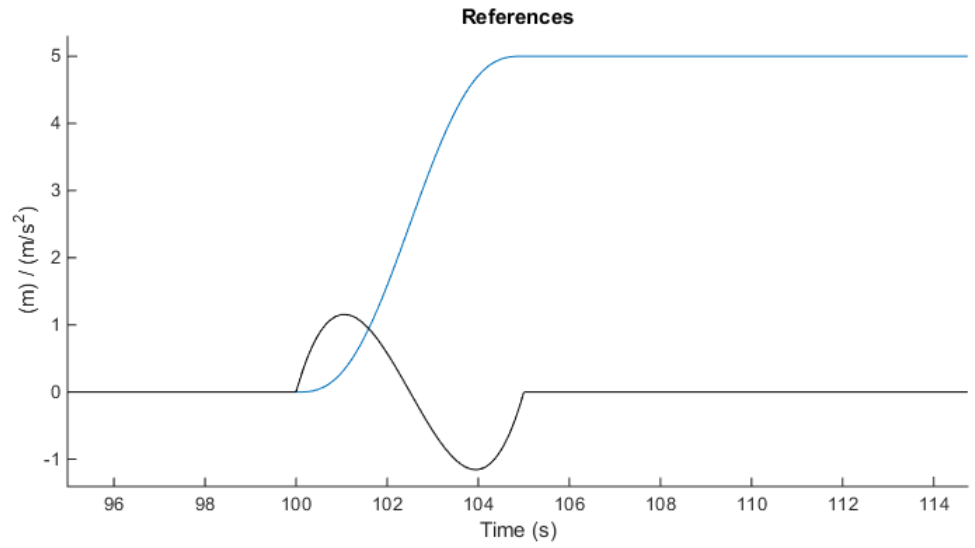

Figure 4.13: Trajectories generated for ξ_{ref} (blue) and $\ddot{\xi}_{ref}$ (black).

Given the reference trajectories needed for the differential flatness controller, the $U2 = \ddot{\phi}_{ref}$ and $U3 = \ddot{\theta}_{ref}$ can be calculated through equations (4.14) and (4.15), respectively.

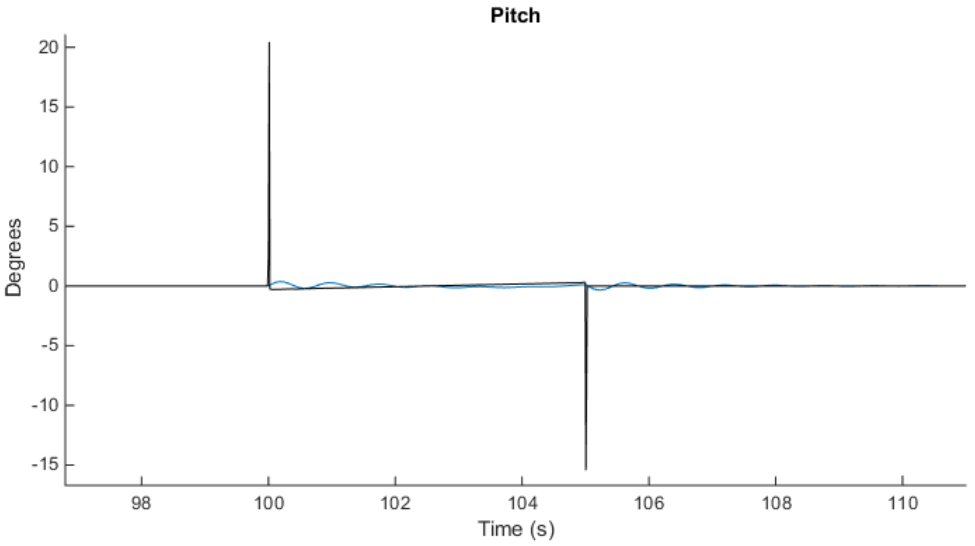

Figure 4.14: Reference generated for γ_{ref} (blue) and control input $\ddot{\gamma}_{ref}$ (black).

Following the architecture represented in Figure 4.12, the reference trajectories for both linear

movement (ξ_{ref} and $\ddot{\xi}_{ref}$) and angular orientation (γ_{ref} and $\ddot{\gamma}_{ref}$) along time can be fed to the switch and to the state space feedback controller. The performance of this control approach is illustrated in Figure 4.15 and 4.16.

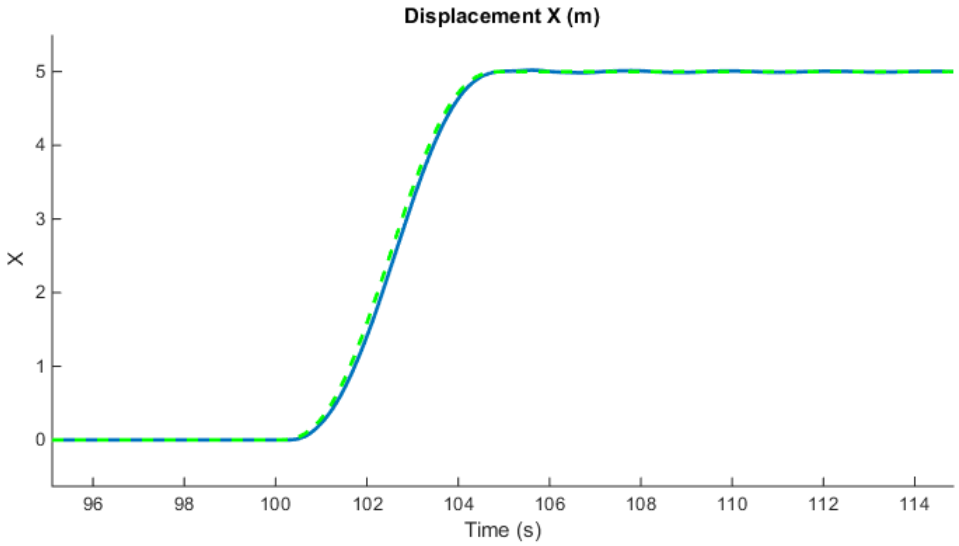

Figure 4.15: Trajectory tracking in the x axis with the differential flatness controller (dotted green line is the reference and blue line is the actual trajectory).

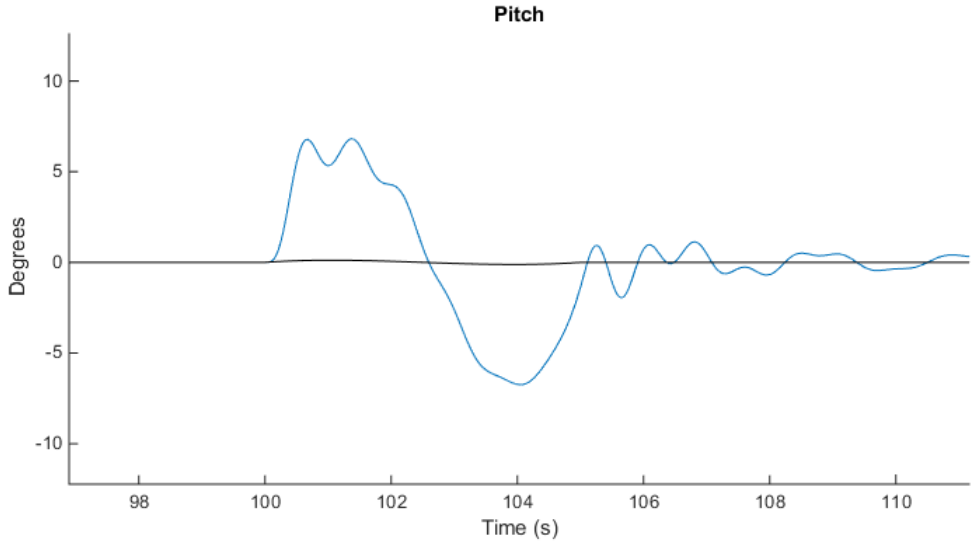

Figure 4.16: Pitch performed by the quadcopter with the differential flatness controller (black line is the reference and blue line is the actual pitch).

As expected, the quadcopter follows perfectly the reference, since the feedforward part of the controller is providing the optimal control inputs to do so, while the feedback controller is

maintaining this control scheme stable. The pitch actuation follows what the differential flatness controller commanded: to do a parabolic pitch forward in the first 2.5 seconds and the symmetric in the next 2.5 seconds. After second 105 in the graphic, the state feedback controller took control of the system, maintaining it stable at the desired setpoint. The figure presents noisy actuation resulting from the fact that this is still a cascade control scheme.

4.2.3.2 Movement in Altitude

The movement in altitude (z axis) will require the same kind of linear references (ξ_{ref} and $\ddot{\xi}_{ref}$) as the movements described in section 4.2.3.1, except the fact that this movement is not related with any angular rotation. The quadcopter movement in the z axis is caused by varying the input U_1 which is directly related with the increase or decrease of the collective torque of the motors. Once again, the starting and ending states are defined, although with a particular difference that is the influence of gravity. The states were defined as $Z_0 = 0, \dot{Z}_0 = 0, \ddot{Z}_0 = g, Z_f = 5, \dot{Z}_0 = 0, \ddot{Z}_0 = g$ and $t_f = 5$, where g is the force of gravity. By giving these starting and ending conditions to the trajectory generator, the references Z_{ref} and $\ddot{Z}_{ref}$ are formed (Figure 4.17).

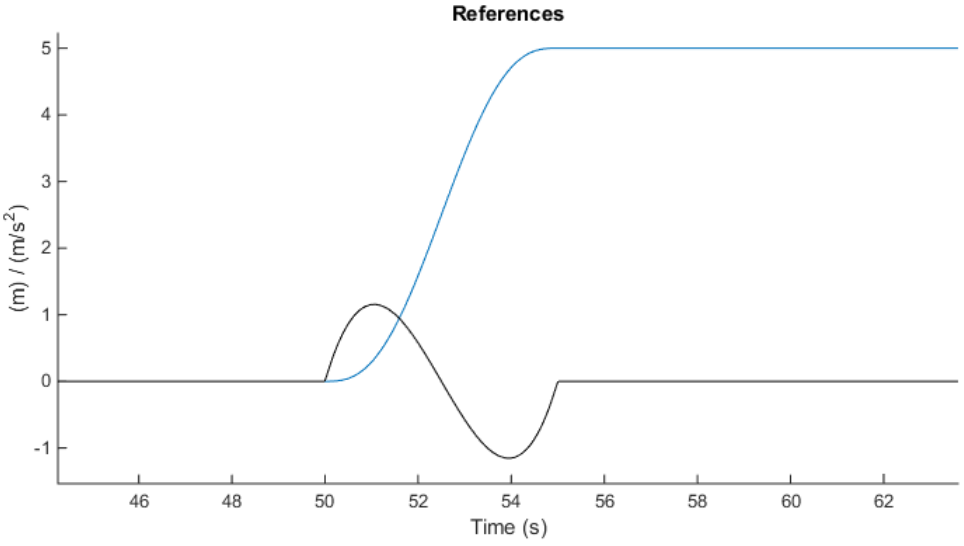

Figure 4.17: Trajectories generated for Z_{ref} (blue) and $\ddot{Z}_{ref}$ (black).

Feeding this trajectory reference to the differential flatness controller as illustrated Figure 4.12 will result in the throttle actuation $U1$ required to move the quadcopter from Z_0 to Z_f. The trajectory performed by the quadcopter in the simulation can be observed in Figure 4.18.

Once again the differential flatness controller performs almost perfectly, the feedforward part provides perfect actuation which results in near optimal trajectory tracking in the z axis. After the quadcopter reaches the reference, it appears to have an undershoot due to the fact that the state space feedback controller is not taking into account the force of gravity.

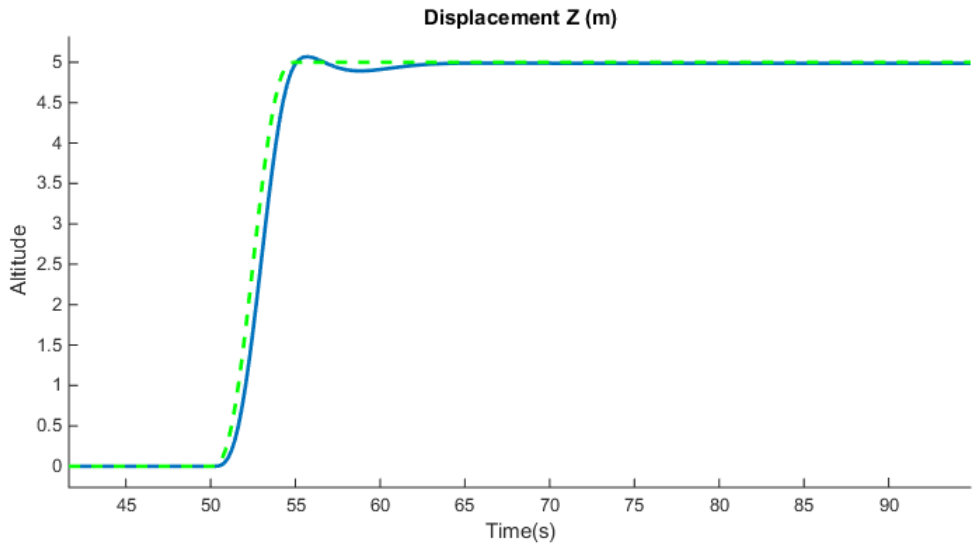

Figure 4.18: Trajectory tracking in the z axis with the differential flatness controller (dotted green line is the reference and blue line is the actual trajectory).

4.3 Localization Algorithms

The main objective of this development regarding the GPS data acquired by the drone, is to improve the precision of this data related with the true location. The fusion of several GPS Modules should and will improve the accuracy to locate the drone in the three dimensions (Latitude, Longitude and Altitude). The following sections related with localization will present the several GPS modules explored, the ones chosen to bring the data closest to the true location, and the algorithms used to fuse them.

Several GPS modules were used and tested in this dissertation, such as the Mini Locator RoyalTek REB-5216, Ultimate GPS Breakout v3, Shield GPS Logger V2 and the NAZA GPS module, but none of them performed as well as the latter. This GPS module grants a precision of 11 mm (7 decimal digits) versus the 0.111 m (6 decimal digits) of the other GPS modules available.

Considering all these GPS modules, the final decision was to use the two best GPS modules (both NAZA GPS). The addition of more or all the GPS modules in the sensor fusion would corrupt the localization data due to lack of precision.

4.3.1 Naza GPS Data Output

The Naza GPS Module sends data continuously as soon as it is powered, at a 115200 baud rate (Pawelsky, 2013). The device will send data related with its location (GPS) and orientation (compass) as illustrated in Figure 4.19, in which the compass data is sent every 30 milliseconds,

following the sequence:

55 AA 20 06 CX CX CY CY CZ CZ CS CS

where each couple of characters represents a byte, which have the following meaning, respectively:

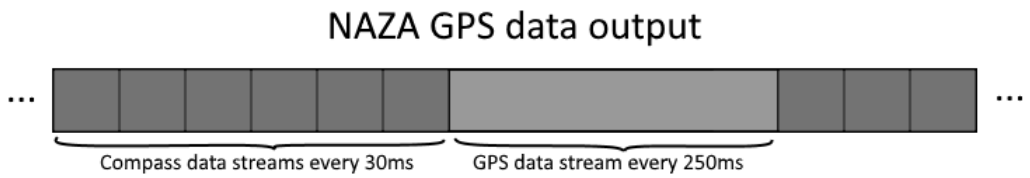

Figure 4.19: NAZA GPS module data output.

Header:

- byte 1-2: message header (always 55 AA);
- byte 3: message ID (20 for compass message);
- byte 4: length of the payload (06 for compass message).

Payload:

- byte 5-6 (CX): compass X axis data;
- byte 7-8 (CY): compass Y axis data;
- byte 9-10 (CZ): compass Z axis data.

Checksum:

- byte 11-12 (CS): checksum.

The payload (except the 9th byte) is XORed with a mask that is calculated based on the 9th byte. The GPS data is sent every 250 milliseconds, following the sequence:

55 AA 10 3A DT DT DT DT LO LO LO LO LA LA LA LA AL AL AL AL HA HA HA HA VA VA VA VA XX XX XX XX NV NV NV NV EV EV EV EV DV DV DV DV PD PD VD VD ND ND ED ED NS XX FT XX SF XX XX XM SN SN CS CS

which follows the same guidelines as the compass data streams:

Header:

- byte 1-2: message header (always 55 AA);
- byte 3: message ID (10 for GPS message);
- byte 4: length of the payload (3A for GPS message).

Payload:

- byte 5-8 (DT): date and time;
- byte 9-12 (LO): longitude (comes multiplied by 10^7, decimal degrees format);
- byte 13-16 (LA): latitude (comes multiplied by 10^7, decimal degrees format);
- byte 17-20 (AL): altitude (comes in millimeters);
- byte 21-24 (HA): horizontal accuracy estimate;
- byte 25-28 (VA): vertical accuracy estimate;
- byte 29-32 (XX): always zero;
- byte 33-36 (NV): NED (North, East, Down) north velocity;
- byte 37-40 (EV): NED east velocity;
- byte 41-44 (DV): NED down velocity;
- byte 45-46 (PD): position DOP;
- byte 47-48 (VD): vertical DOP;
- byte 49-50 (ND): northing DOP;
- byte 51-52 (ED): easting DOP;
- byte 53 (NS): number of satellites (not XORed);
- byte 54 (XX): always zero;
- byte 55 (FT): fix type (0 – no lock, 2 – 2D lock, 3 – 3D lock);
- byte 56 (XX): always zero;
- byte 57 (SF): fix status flags;
- byte 58-59(XX): always zero;
- byte 60 (XM): (XOR mask);
- byte 61-62 (SN): sequence number (not XORed);

Checksum:

- byte 63-64 (CS): checksum.

The payload is again XORed with the mask (byte 60). The latitude and longitude coordinates are obtained and calculated with 7 decimal digits, which guarantees a precision of 11 millimeters.

4.3.2 Sensor Fusion

With two NAZA GPS modules providing data, it is possible to fuse them to improve the localization precision (Magnusson and Odenman, 2012). The Kalman filter enables the use of several sensors and build an estimation based on them, which results in the so called sensor fusion. This method will estimate the state variables of the GPS signal from incomplete noisy measurements and fuse them to improve the estimation of the present state values, in an online recursive fashion.

In this procedure, the Arduino Due reads both GPS signal through Serial protocol, processes it and returns it in a 3rd serial connection to the controller. After reading, the longitude, latitude and altitude are extracted from both GPS data streams and included in the Kalman filter algorithm, which returns the filtered data for each of the three location parameters. All steps of this algorithm will be explained further ahead, from the decoding of the GPS modules to the fusion of both datum and the process of reconstructing the data stream that provides localization to the quadcopter.

The first step is to set up the Kalman filters for latitude, longitude and altitude:

- $$Q = 0.00001 \tag{4.19}$$
- $$R = \begin{bmatrix} 0.0001 & 0 \\ 0 & 0.0001 \end{bmatrix} \tag{4.20}$$
- $$P = 0 \tag{4.21}$$
- $$x = x_0 \tag{4.22}$$
- $$C = \begin{bmatrix} 1 \\ 1 \end{bmatrix} \tag{4.23}$$

The Q is the process noise co-variance and R is the measurement noise co-variance, in which both were determined experimentally by measuring the amplitude and variance of the noise for the latitude, longitude and altitude. The covariance error starts at $P = 0$ and the algorithm will constantly update it to the true value. The starting position $x = x_0$ is set to be the location just outside the Aerodynamics and Control lab at the Electronics Engineering Department in the FCT UNL campus: $Lat = 38.6604170$ (degrees), $Lon = -9.2050560$ (degrees) and $Alt = 145$ (meters). The C matrix is the output matrix and in this case, two sensors are being measured and they are both 1 because they contribute equally for the sensor fusion process.

After initializing the Kalman filter, the measurements of both GPS modules have to be read and decoded. A circular vector for each GPS module was created to save the bytes of the respective data streams. These bytes would be decoded by following the descriptions in section 4.3.1 to extract the localization parameters. Now with the measurements m_1 and m_2 available and after defining the measurement matrix Z, the Kalman update takes place, following the algorithm described in section **??**.

$$Z = \begin{bmatrix} m_1 & 0 \\ 0 & m_2 \end{bmatrix} \tag{4.24}$$

With the measurements being fused in every iteration, the final task of this algorithm is to encode the data back to the data stream of bytes so that the NAZA controller can understand it. The decoding and encoding of these data streams will be explained in the next section 4.3.3. This procedure describes the sensor fusion of two sensors, but it can be easily expanded for more sensors by adjusting the respective matrices mentioned in this section.

4.3.3 Data Management, Decoding and Encoding

As mentioned in section 4.3.2, the sensor fusion algorithms accounts with the decoding of the NAZA GPS module output, data management of it and finally the encoding the fused data so that the NAZA attitude controller can understand the fused data. All this procedure starts with the decoding of the GPS module output, byte by byte, of the data streams related with the GPS enumerated in section 4.3.1.

For each GPS module, each byte is stored in a cyclic vector so that the old bytes won't progressively occupy the storage of the processor and can be erased automatically. The data is extracted from each vector by decoding it (XOR each byte with the mask), fused by the Kalman filter and then saved in an extra vector. This extra vector will have the same information as one of the others but after the fusion occurs, the fused data will be overwritten where the GPS data stream is supposed to be, as illustrated in figure 4.20.

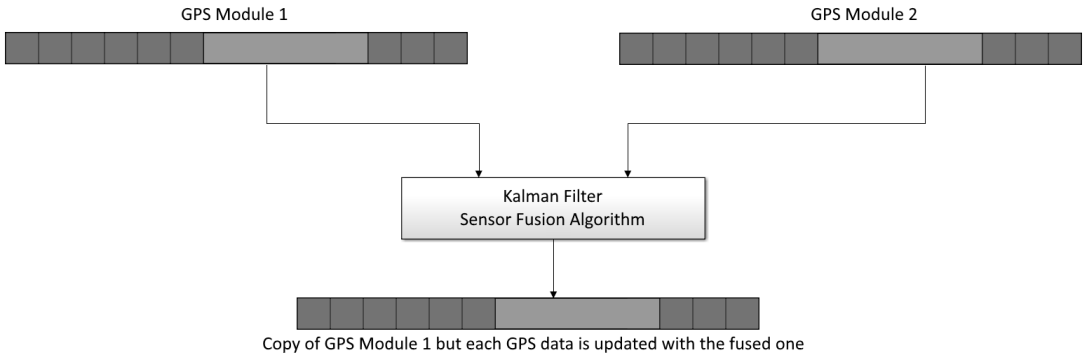

Figure 4.20: Sensor fusion of two NAZA GPS modules' data.

The checksum of each data stream (last two bytes of each data stream $CS1$ and $CS2$) are the validating bytes of the whole stream. If the data would become corrupted or missing, the checksum of that stream wouldn't correspond with the rest of the data, so it would end up ignored. Because the filtered GPS data stream is a copy of the GPS Module 1 (so that the Compass data can remain intact like its coming directly from the GPS module) and the new filtered GPS location is re-encoded where the previous location data was, the checksum will not correspond with the payload.

The checksum follows the iterative instructions for each byte of the payload:

$$CS1 = CS1 + input;$$

$$CS2 = CS2 + CS1;$$

where the $input$ is a byte of the payload, $CS1$ and $CS2$ are the bytes 63 and 64, respectively. After these calculations, the resulting checksum can be updated so that the NAZA attitude controller accepts the newly fused localization data.

CHAPTER 5

SIMULATIONS AND EXPERIMENTAL RESULTS

The simulations and experimental results are the proof of the theory presented in Chapters 2 and 3. This Chapter will present all the simulations related with trajectory tracking algorithms as well as localization algorithms. A comparison of the performance between trajectory tracking controllers and an overview of how the controllers behaved in simulations and experiments in the real drone will take place at the end of this section. The trajectory algorithms were not experimented in the real quadcopter X8 since the safeguard conditions were not met. In order to guarantee the safety of both the quadcopter and the environment around, there needs to be a safe zone where the quadcopter can fly freely but restrained in case any unpredicted behavior might cause damage (cut something with the fast spinning propellers or damage any part of the structure due to eventual collisions).

5.1 Simulations

5.1.1 Trajectory Tracking Algorithms

The trajectories chosen to test the performance of the controllers proposed were the same to make it easier to understand their different behaviors. These controllers were developed in Matlab and Simulink with the solver ode45 and 0.01 seconds time step. All simulations have a fault in rotor 1 at time 350 seconds. According to (Brito, 2016), to correct the quadcopter's behavior when a rotor fault happens, in order to keep it flying safely, the opposite rotor (motor 7) should be deactivated.

The simulations also take into account the values of the quadcopter obtained during modeling in the work (Brito, 2016). These values are presented in Table 5.1 along with their respective value.

Table 5.1: X8-VB Quadcopter's characteristics.

Constant	Description	Value
$J_t[Kgm^2]$	Inertial moment of rotation	5
$w[rad/s]$	Motor's rotation speed	0
$m[Kg]$	Body mass	0.02
$I_{xx}[Nms^2]$	X-axis inertia	0.006
$I_{yy}[Nms^2]$	Y-axis inertia	0.006
$I_{zz}[Nms^2]$	Z-axis inertia	0.0166

5.1.1.1 Trajectory Tracking with PID Controller

The PID controllers were tuned through the Ziegler-Nichols method as explained in Section 4.2.1. The gains obtained are listed in Table 5.2.

Table 5.2: Controller parameters obtained from the Ziegler-Nichols method.

Controller	K	T_i	T_d
Altitude	33	0.1	6
Roll	0.0001	0.02	300
Pitch	0.0001	0.02	300

The performance of this controller can be observed in in Figure 5.1, and the respective attitude references and actuation of the drone are represented in Fig. 5.2.

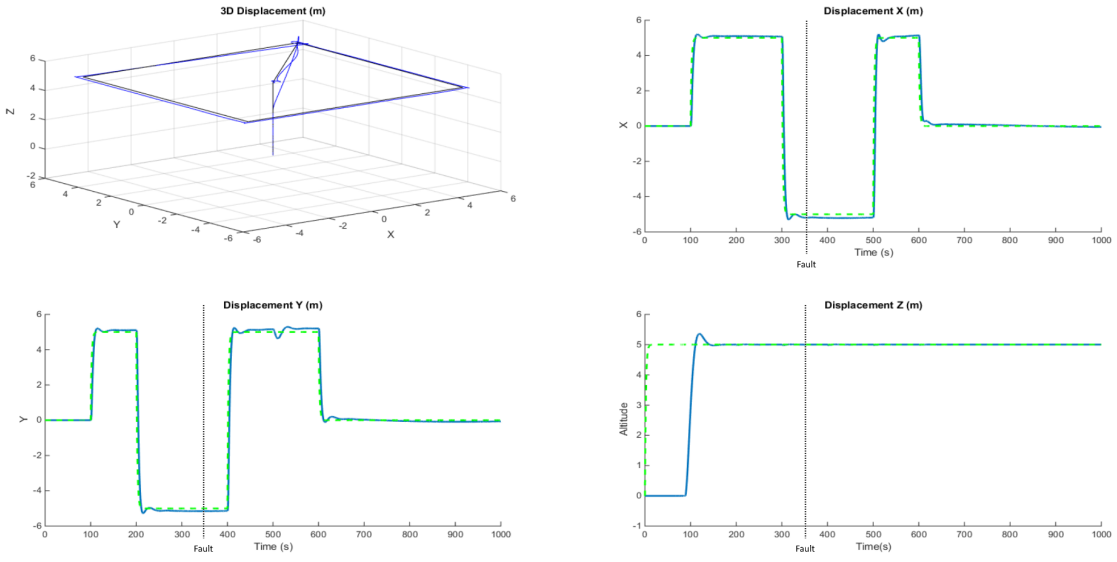

Figure 5.1: PID controller for trajectory tracking of the drone (dotted green line is the reference and blue line is the actual trajectory).

As it can be observed, the fault at time 350 seconds does not compromise the quadcopter's

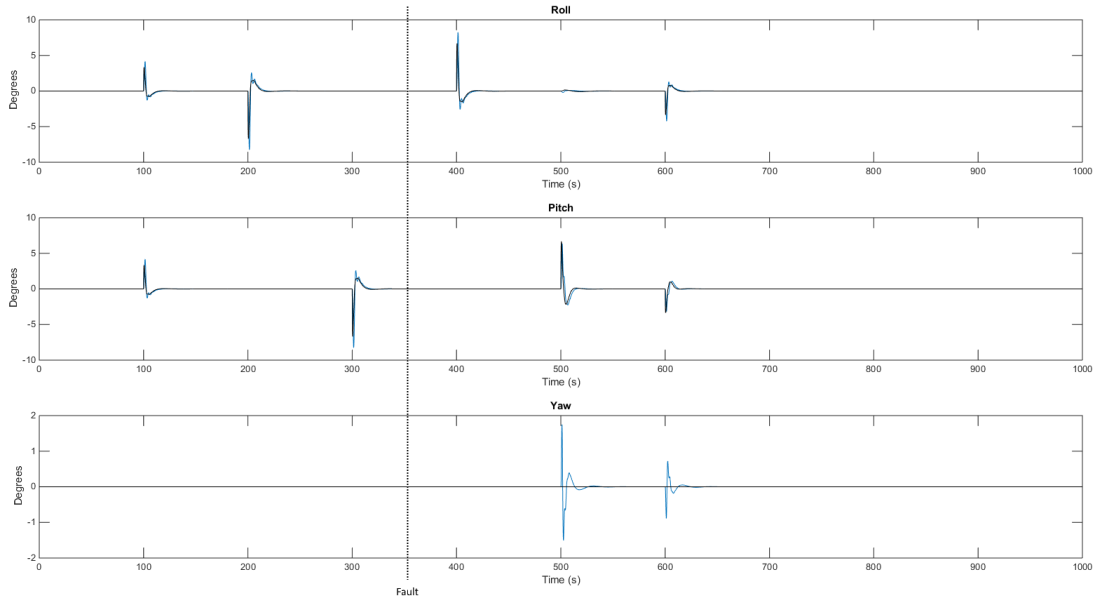

Figure 5.2: PID actuation for trajectory tracking of the drone (black line is the reference and blue line is the actuation).

well being, in fact, it is almost unaffected by it, as it continues to fly and completes the planned trajectory, although the yaw controller seems to have difficulty in keeping the desired setpoint. This happens because after the fault, two rotors spinning in the same direction were deactivated and thus the total yaw torque is unbalanced and the yaw PID is not prepared for this occasion. Overall, the controller presents slow actuation and some overshoot while tracking the predefined trajectories, which is not desired in any trajectory performance of any aircraft. Since the PID is purely feedback oriented and works with only some of the relevant variables (e.g. error between the desired and the actual pitch), it will always underperform in such a fast system and in trajectory tracking in particular.

5.1.1.2 Trajectory Tracking with State Space feedback Controller

The State Space was tuned through the LQR method as explained in Section 4.2.2. The weightings of each parameter were set with a balanced energy consumption versus flight time trade off. The gains obtained are listed in Table 5.3.

Table 5.3: Controller parameters obtained from the LQR method.

Controller	ξ	$\dot{\xi}$	γ	$\dot{\gamma}$
Altitude	10	32.4	-	-
Roll	0.6	1.5	9	0.6
Pitch	0.6	1.5	9	0.6

The performance of the State Space feedback tuned with the LQR method can be observed in Figure 5.3 while moving along the predefined waypoints. The actuations that caused these movements are illustrated in Figure 5.4.

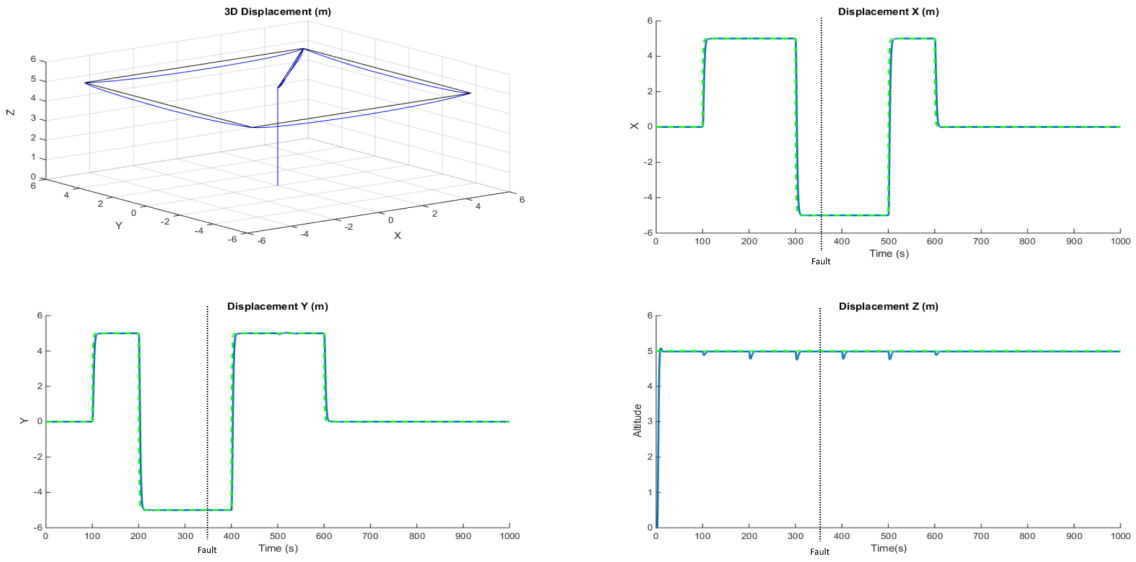

Figure 5.3: State Space feedback controller for trajectory tracking of the drone (reference represented as dotted green and the actual trajectory as continuous blue).

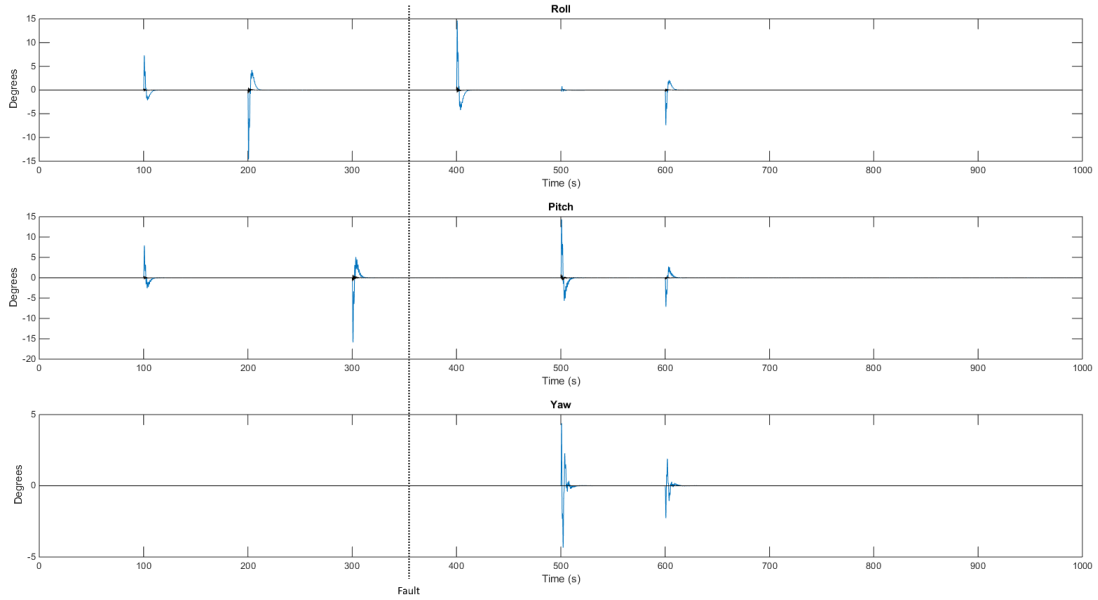

Figure 5.4: State Space feedback actuation for trajectory tracking of the drone (reference represented as black and actuation as blue).

From these simulations, it can be concluded that the state space controller performs good

trajectory tracking throughout the setpoints. This controller presents fast actuation in order to move to the next waypoint rapidly and has complete dampness in the overshoot, since the pitch and roll actuation are allowed to have that much responsiveness on the state of the system. However, this comes at a cost of altitude tracking issues, since the loss of sustentation is imperative with such aggressive roll and pitch actuations. Once again, the fault at time 350 seconds does not compromise the quadcopter's safety, as it continues to fly and completes the planned trajectory, although the yaw controller seems to have the same difficulties in keeping the desired setpoint as the PID controller.

5.1.1.3 Trajectory Tracking with Differential Flatness Controller

The performance of the Differential Flatness controller with the State Space architecture as feedback to stabilize the system's response can be observed in Figure 5.5 while moving along the predefined waypoints. The actuations that caused these movements are illustrated in Figure 5.6.

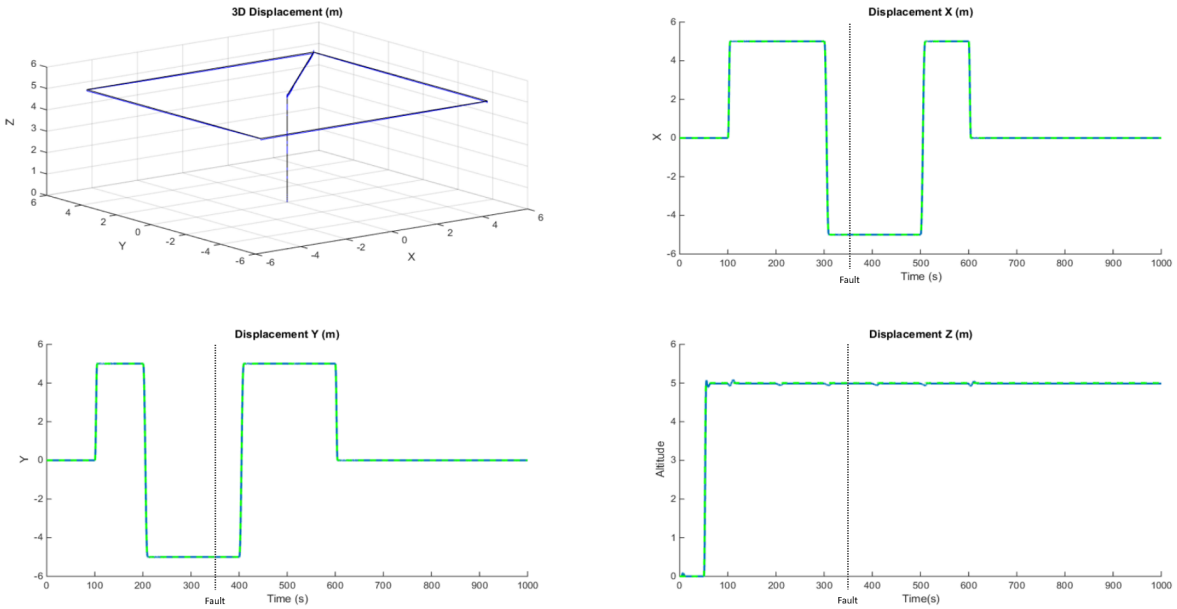

Figure 5.5: Differential Flatness controller for trajectory tracking of the drone (reference represented as dotted green and the actual trajectory as continuous blue).

Just like the previous controllers, this one is unaffected by the fault at time 350 seconds and has difficulty in maintaining the yaw angle. Through the graphics it can be observed that the trajectory of the drone is almost on par with the reference, having a near perfect trajectory tracking. This happens because the trajectory generator along with the differential flatness block provide the optimal references and actuations to move the drone from one set point to another while respecting the initial and final conditions of hover state (all attitude and linear variables are constant).

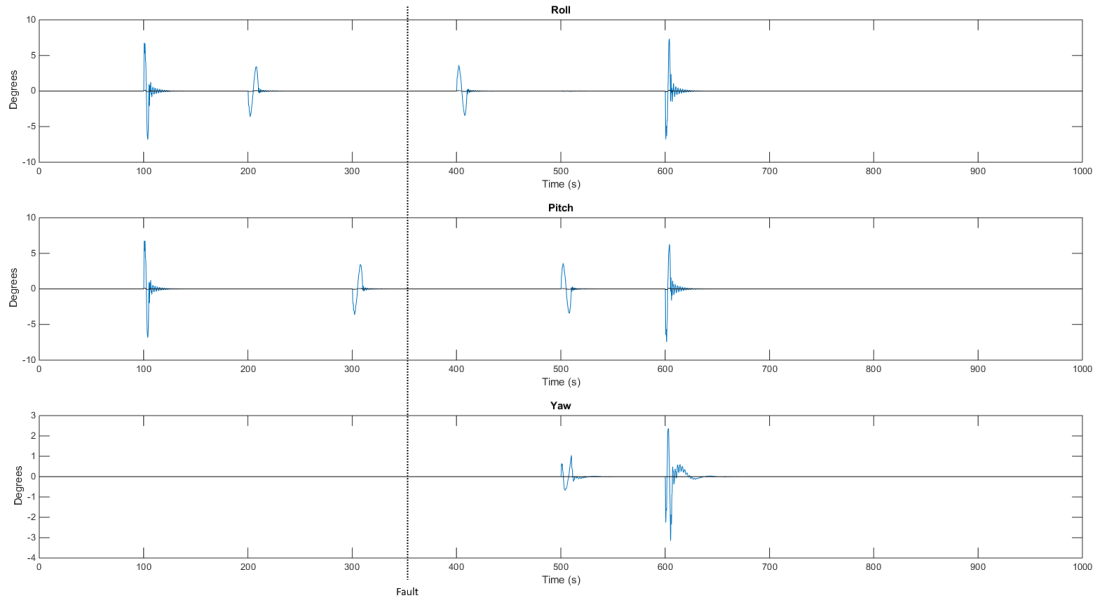

Figure 5.6: Differential Flatness actuation for trajectory tracking of the drone (reference represented as black and actuation as blue).

5.1.1.4 Comparison between Trajectory Tracking Controllers

Three trajectory tracking control approaches were developed and compared in this work. All three controllers were unaffected by the motor fault throughout the full trajectory, although all of them presented issues in tracking the yaw orientation after it happened at time 350 seconds. The PID is the slowest controller, providing a low actuation in the inputs and still presenting overshoot, which clarifies that this methodology is not indicated for trajectory tracking purposes. The State Space controller has complete dampness in the overshoot, even with aggressive roll and pitch actuations. These actuations bring considerable altitude loss, which makes this controller adequate up to a certain degree. The Differential Flatness has the best performance, since the optimal control inputs are computed beforehand considering each couple of waypoints conditions. This results in almost perfect reference tracking, with no overshoot and negligible deviation from the reference. This outstanding performance explains why most solutions for trajectory tracking on drones, and practically every problem related with trajectory, is solved by differential flatness. The comparison of rising times and overshoots are presented in Table 5.4.

5.1.2 Localization Algorithms

To test and validate the sensor fusion algorithm, an experiment took place at the Aerodynamics and Control laboratory at the Electronics Engineering Department at the FCT UNL Campus. The result of this experiment can be observed in Figures 5.7 and 5.8 which show the latitude, longitude

Table 5.4: Rising time and overshoot comparison between controllers (x axis).

Controller	Rising Time [seconds]	Overshoot [percentage]
PID	10	0.05
SS	10	0
DF	5	0

and altitude of both received from both GPS modules and their fusion. The blue signal is the NAZA GPS module 1 and as red is the NAZA GPS module 2. The yellow signal is the sensor fusion between the two GPS modules. The filtered signal presents a smoother localization and more precise with the reality. The values obtained from this experience were validated with Google Maps for the latitude and longitude coordinates and with the FCT UNL topographic map for the altitude. Figure 5.8 also shows the satellites in view which will be relevant for the precision of the signals.

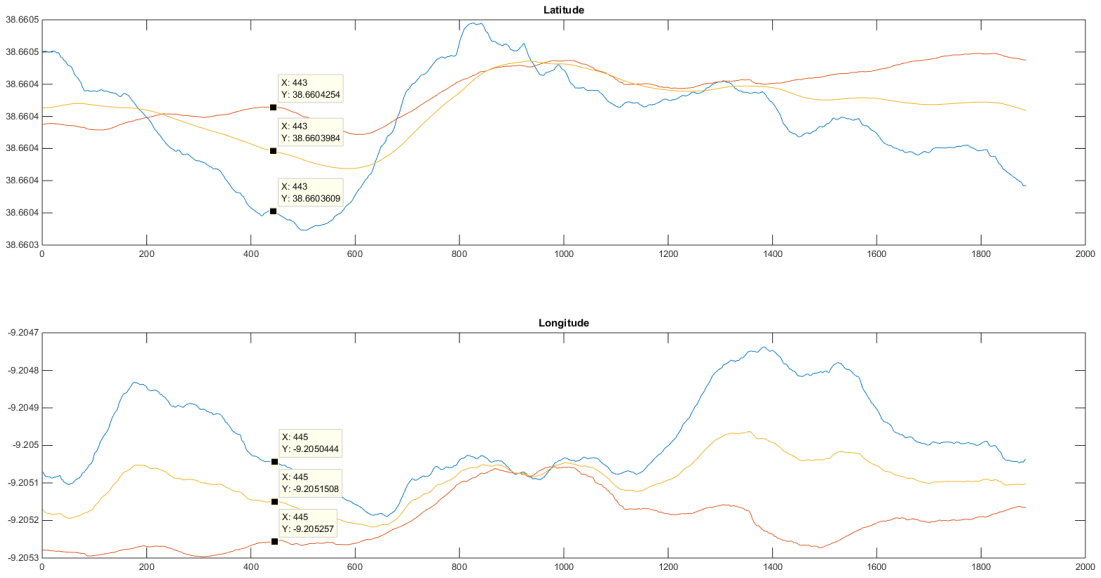

Figure 5.7: Latitude and longitude of the position received by the two NAZA GPS modules, along with its Kalman filtering.

The filtered signal (yellow) presents considerably less noise than the data received directly from the NAZA GPS modules (blue and red) and corresponds to the true value, according with Google Maps and the FCT UNL topographic map. The filtered signal presents an outstanding performance with the altitude in particular, since the difference between the two senors reaches the 20 meters. In Figure 5.9 can be observed the true latitude and longitude. The altitude of the sensor can also be obtained by adding the height of the building at the second floor at that place ($\sim$ 10

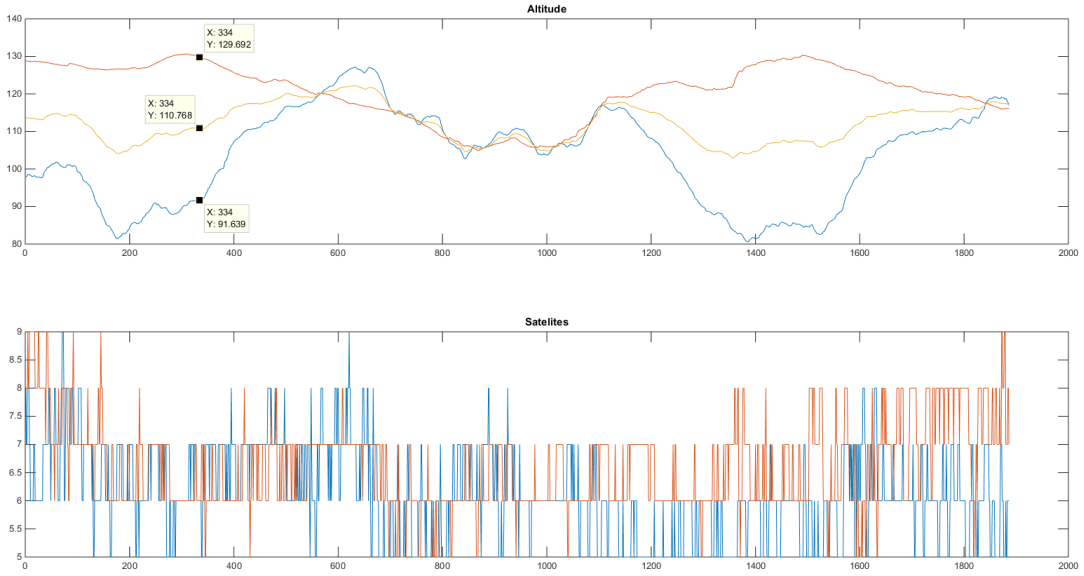

Figure 5.8: Satelites in view and altitude of the position received by the two NAZA GPS modules, along with its Kalman filtering.

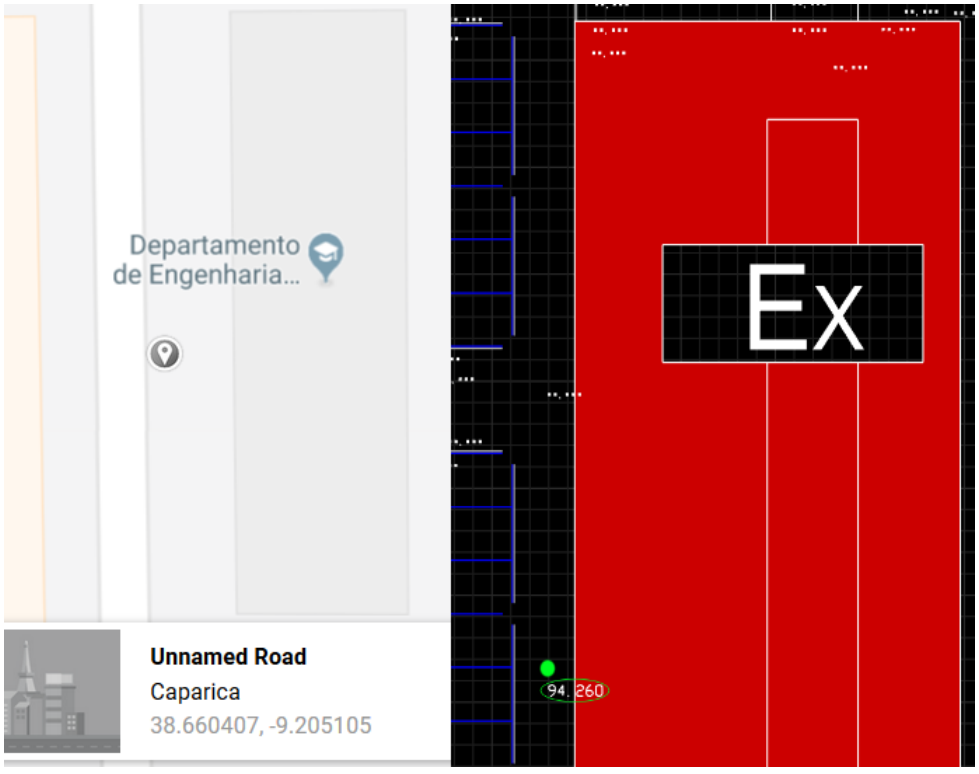

Figure 5.9: Location of the GPS module on Google Maps on the left and its altitude on the right (Green dot).

meters).

5.2 Experimental Results

This section will present the experimental results by applying the control algorithms simulated before, on the quadcopter and observe its behavior. The X8-VB Quadcopter subject to the experiments of this dissertation is illustrated in Figure 5.10 featuring the two NAZA GPS modules for improved localization data through sensor fusion.

Figure 5.10: X8-VB Quadcopter featuring two GPS sensors for improved localization data.

5.2.1 NAZA Controller Identification

In order to identify the NAZA controller, several experiments were done by applying the PWM inputs (roll, pitch, throttle and yaw) and observing its behavior (angle variations and rotor speeds). This was possible by assembling a test bench that would allow the NAZA controller to actuate on the motors and the whole structure could change its orientation accordingly. This test bench can be observed in Figure 5.11. To model the pitch of the controller, two experiments were performed in order to get estimation and validation data Figure5.12.

Due to the nonlinearity of the controller and its noisy output signals, there was no method that could model the pitch of the quadcopter. These signals could possibly be filtered but at the cost of information, which is not recommended to such a fast system. Some of the best results of this experiment can be observed in Figure 5.13. Several attempts to model it with these noisy and inconsistent signals were done, with NARX (various combinations of number of terms for the input and output) and Hammerstein-Wiener (several combinations of number of units for the input and

Figure 5.11: Test bench for the NAZA controller identification.

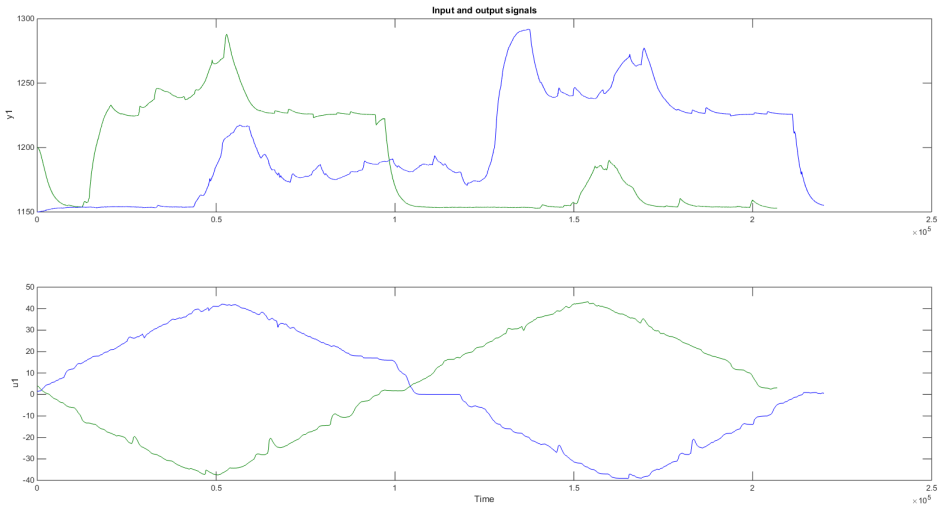

Figure 5.12: Estimation (blue) and validation (green) data of the pitch input (below) and torque motor output (above).

output) but none of them presented good enough results to serve as a model for the quadcopter's controller.

After looking closely to the results obtained, it appears that the PWM signals generated from the arduino Uno and Due seem to have inconsistencies, which might be the cause of the noise obtained from the NAZA controller and an alternative for a PWM generator should be explored. Also, the NAZA controller has several sensors integrated (such as gyroscope, accelerometer and barometer) that might influence the output signals, so a further look into how to identify the controller taking into account these internal variables will stay as future work. Table 5.5 describes which methods were used to create the models illustrated in Figure 5.13.

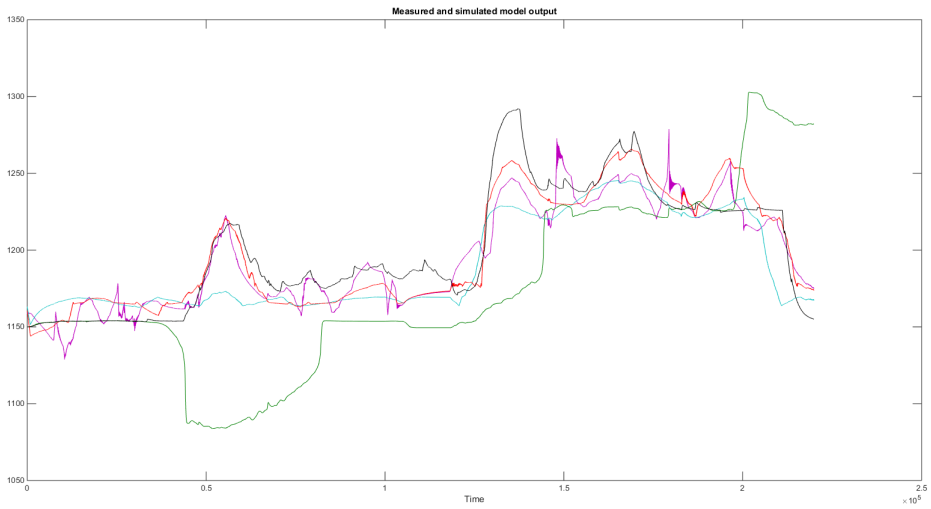

Figure 5.13: Model attempts of the NAZA pitch actuation.

Table 5.5: Model attempts of the NAZA pitch actuation signals.

Signal color	Method	Input terms	Output terms	Fit percentage
Black	motor 3 reference	-	-	-
Red	Hammerstein-Wiener	10	10	58.36
Purple	Hammerstein-Wiener	12	12	44.87
Blue	Hammerstein-Wiener	8	8	44.87
Green	NARX	2	2	- 48.9

5.2.2 Sensor Fusion

The Sensor fusion algorithm featuring the Kalman filter was deployed successfully on the quadcopter, the NAZA controller accepted the encoded signal with the filtered data from the two GPS modules and performed a smooth flight trajectory illustrated in Figure 5.14.

The Kalman filter does a good approximation of the real location on the quadcopter along the trajectory, while minimizing the noise present in each GPS's signal.

The Trajectory illustrated in 5.14 was performed near the FCT UNL Library. A view from above can be observed in Figure 5.15 and was validated with the latitude and longitude in Google Maps as can be observed in Figure 5.16 and its altitude in Figure 5.17.

As observed in Figure 5.16, the obtained coordinates match the trajectory performed with the quadcopter. The Kalman filter brings a better approximation and less noisy GPS signal, which will result in smoother flights, with less vibrations on the structure when in GPS mode.

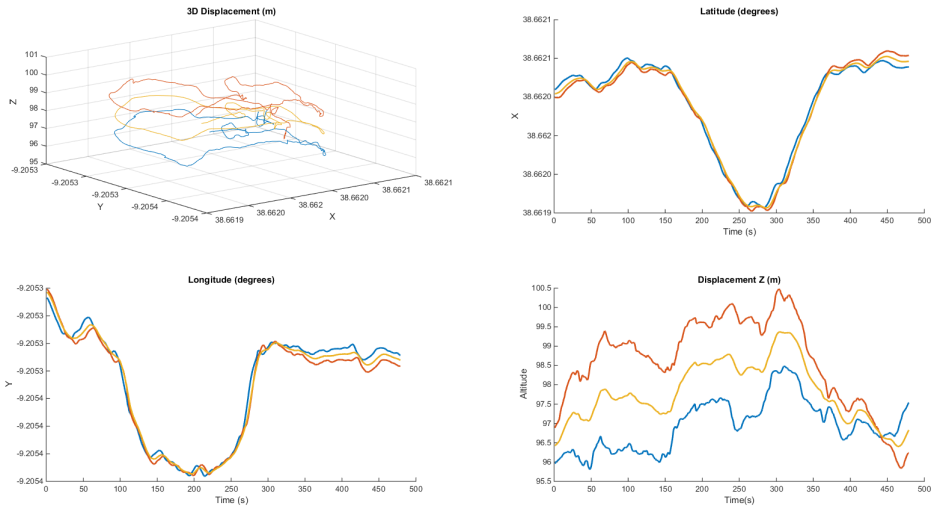

Figure 5.14: Sensor fusion of two NAZA GPS modules in latitude, longitude and altitude (GPS1-Blue, GPS2-Red, Filtered-Yellow.

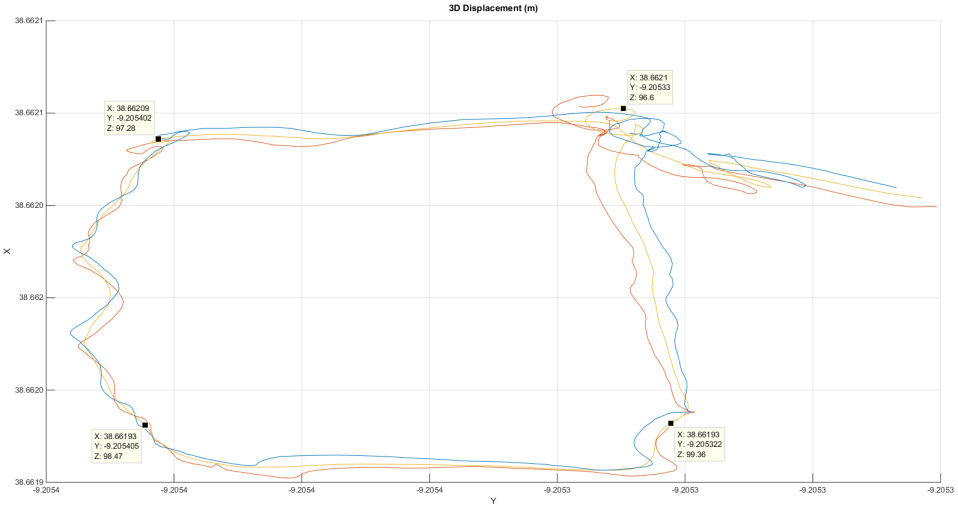

Figure 5.15: Trajectory performed by the quadcopter with the sensor fusion providing the filtered coordinates.

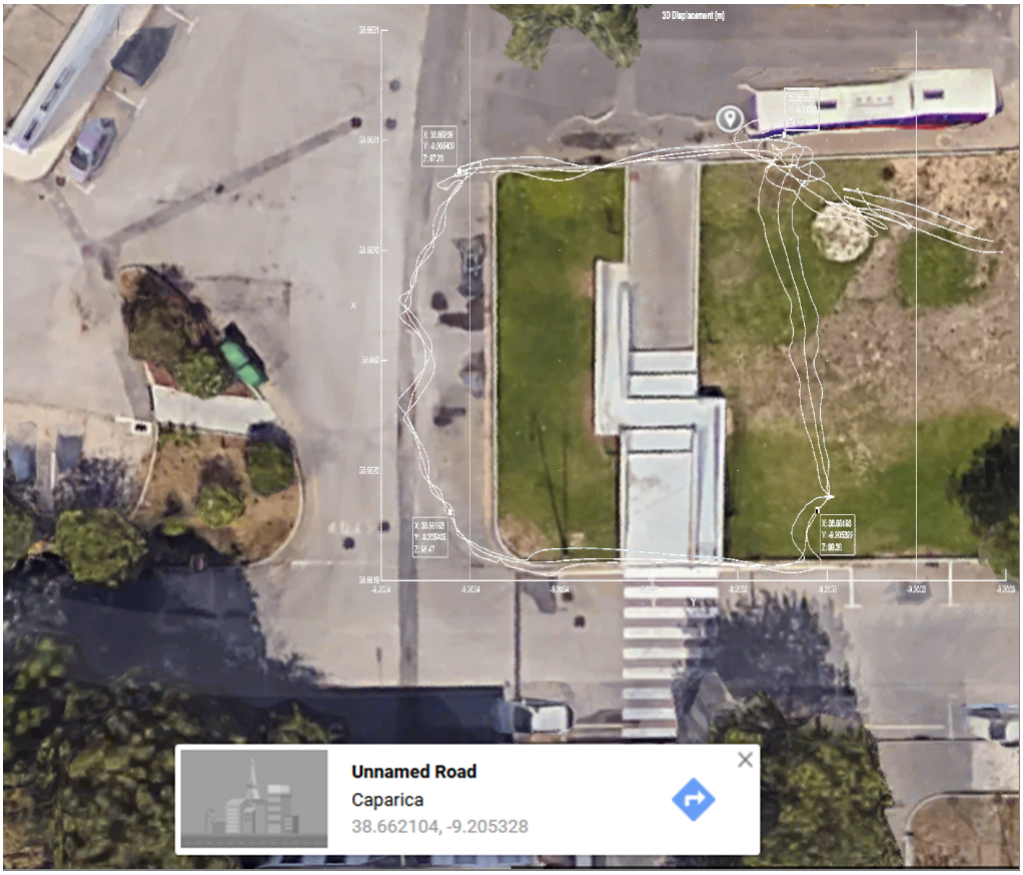

Figure 5.16: Validation of the coordinates obtained from the algorithm with Google Maps.

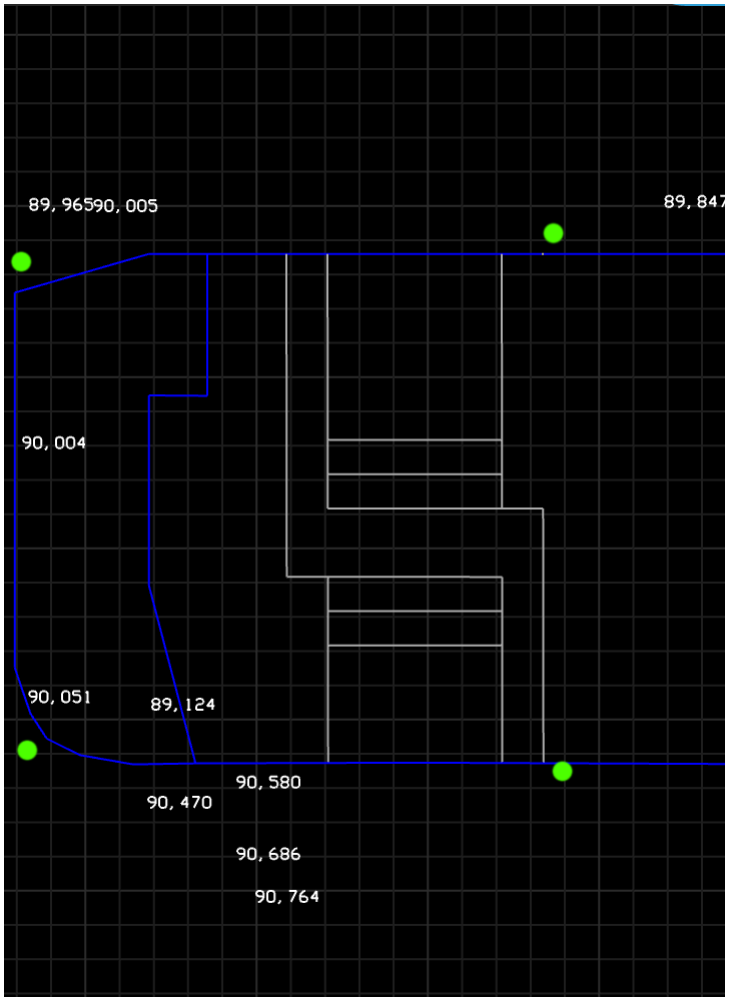

Figure 5.17: Validation of the altitude obtained from the algorithm with the topographic map of the FCT UNL Campus (Green dots are the vertices of the square.

CHAPTER 6

CONCLUSIONS AND FUTURE WORK

During this work, several trajectory and localization algorithms were proposed for the X8-VB quadcopter. The main goal of this dissertation was to improve the GPS data acquired by the quadcopter and to design control algorithms that control the quadcopter along a trajectory considering the flight time, energy consumption and eventual faults.

The localization of the quadcopter was improved and tested in real experiments with good results. The NAZA® controller accepted the filtered signal and flew with less perturbations caused by the noise in the GPS signal. Among trajectory tracking algorithms, the differential flatness controller presented the best performance in the simulations, following the reference perfectly. The state space controller, as a purely feedback controller, follows the reference with no overshoot and with average rising time. From the simulations one can understand that the PID controller is not adequate of this kind of problem, since it is a purely feedback algorithm and is limited to just some of the variables of the system. The modeling of the NAZA® attitude controller was not successful due to inconsistencies and noise in the output signals (motors). The trajectory algorithms were not tested in the real quadcopter as there weren't conditions to safeguard the drone or the environment if something were to happen out of what was planned. Although, the construction of a custom room for quadcopter testing is in the works and soon all the algorithms related with attitude and trajectory control of the quadcopter shall be tested and further developed.

From the work done in this dissertation it can be concluded that the usual feedback controllers are enough to plan and execute trajectories with aircrafts but present a far from optimal performance. The addition of feedforward control brings the precision and speed that is required for this kind of problem. Also, the sensor fusion of GPS modules does in fact improve the localization precision and attenuates noise.

As for future work, there are some aspects about this work that can be explored, namely: improvement of the differential flatness architecture; test of the trajectory controllers in the real quadcopter X8; and identification of the NAZA® attitude controller.

The trajectory algorithms developed in this dissertation contributed to a paper published on IEEE and presented in the international conference named "YEF-ECE 2018 - 2nd International Young Engineers Forum on Electrical and Computer Engineering" (Alexandre Brito, Vasco Brito, Luis Brito Palma, Fernando Coito, Paulo Gil).

BIBLIOGRAPHY

Alder, K. (2002). "The_Measure_of_All_Things_The_Seven-Year_Odyssey". In:

Anderson B., M. J. (1990). *Anderson B., Moore J.-Optimal control_ Linear quadratic methods (no p.229)-PH (1989)(1).pdf*.

Araki, M (2002). "PID control". In: *Control systems, robotics and automation 2* II, pp. 1–23.

Arduino (2018). *Arduino Uno*. URL: https://store.arduino.cc/arduino-uno-rev3 (visited on 03/12/2018).

Åström, K. J. and B. Wittænmark (1997). *Computer Control System - Theory and Design*. Third Edit. Prentice Hall, p. 569. ISBN: 7302050082. DOI: 10.1002/1521-3773(20010316)40:6<9823::AID-ANIE9823>3.3.CO;2-C. arXiv: arXiv:1011.1669v3.

Åström, K. J. and T. Hägglund (2006). *Advanced PID Control*, p. 460. ISBN: 1556179421. DOI: 978-1-55617-942-6.

Balaji, D. (Nov. 2007). "Multivariable Laguerre-Based Indirect Adaptive Predictive Control A Reliable Practical Solution for Process Control". In:

Boas, M. L. (2005). *Mathematical Methods in The Physics Sciences*. Vol. 17, p. 839. arXiv: 0605511.

Bouabdallah, S (2007). "Design and Control of Quadrotors With Application To Autonomous Flying". PhD thesis. Phd Thesis, École Polytechnique Fédérale de Lausanne, p. 61. DOI: 10.5075/epfl-thesis-3727. URL: http://biblion.epfl.ch/EPFL/theses/2007/3727/EPFL{_}TH3727.pdf(2015-10-24).

Bouktir, Y., M. Haddad, and T. Chettibi (2008). "Trajectory planning for a quadrotor helicopter". In: *2008 16th Mediterranean Conference on Control and Automation*, pp. 1258–1263. DOI: 10.1109/MED.2008.4602025.

Brito, V. (2016). "Fault Tolerant Control of a X8-VB Quadcopter". MSc. FCT-UNL.

Caron, F., E. Duflos, D. Pomorski, and P. Vanheeghe (2006). "GPS/IMU data fusion using multisensor Kalman ltering: introduction of contextual aspects". In: *Information Fusion* 7(2), pp. 221–230. ISSN: 15662535. DOI: 10.1016/j.in.

Cowling, I. D., O. a. Yakimenko, J. F. Whidborne, and A. K. Cooke (2007). "A Prototype of an Autonomous Controller for a Quadrotor UAV". In: *European Control Conference*, pp. 1–8.

DJI (2016). *Phantom 4*. URL: http://store.dji.com/new-release?from=store{\%}7B{_}{\%}7Dindex{\%}7B{_}{\%}7Dbanner.

El-Rabbany, A. (2002). *Introduction to GPS : the Global Positioning System*. Artech House, p. 176. ISBN: 1596930160.

Fahroo, F. and I. M. Ross (2002). "Direct Trajectory Optimization by a Chebyshev Pseudospectral Method". In: *Journal of Guidance, Control, and Dynamics* 25(1), pp. 160–166. ISSN: 0731-5090. DOI: 10.2514/2.4862. URL: http://arc.aiaa.org/doi/10.2514/2.4862.

Federal Aviation Administration (2008). *Pilot ' s Handbook of Aeronautical Knowledge*. Oklahoma, p. 7. ISBN: 1602397805. DOI: 10.1016/S0740-8315(86)80070-5.

Figueiredo, H., A. Bittar, and O. Saotome (2014). "Platform for quadrirotors: Analysis and applications". In: *2014 International Conference on Unmanned Aircraft Systems, ICUAS 2014 - Conference Proceedings*(May), pp. 848–856. DOI: 10.1109/ICUAS.2014.6842332.

Fliess, M. (1990). "Generalized Controller Canonical Forms for Linear and Nonlinear Dynamics". In: *IEEE Transactions on Automatic Control* 35(9), pp. 994–1001. ISSN: 15582523. DOI: 10.1109/9.58527.

Floyd, M and K Palamartchouk (2015). "Fundamentals of GPS for geodesy". In:

Formentin, S. and M. Lovera (2011). "Flatness-based control of a quadrotor helicopter via feedforward linearization". In: *Proceedings of the IEEE Conference on Decision and Control*, pp. 6171–6176. ISSN: 01912216. DOI: 10.1109/CDC.2011.6160828.

Gibbs-Smith, C. (1978). *The Inventions of Leonardo Da Vinci*.

(GPS/Daily) (2014). *Galileo works, and works well*. URL: http://www.gpsdaily.com/reports/Galileo{_}works{_}and{_}works{_}well{_}999.html (visited on 01/31/2017).

Gurdan, D., J. Stumpf, M. Achtelik, K. M. Doth, G. Hirzinger, and D. Rus (2007). "Energy-efficient autonomous four-rotor flying robot controlled at 1 kHz". In: *Proceedings - IEEE International Conference on Robotics and Automation*(April), pp. 361–366. ISSN: 10504729. DOI: 10.1109/ROBOT.2007.363813.

Hall, D. L. and J Llinas (1997). "An introduction to multisensor data fusion". In: *Proc. of the IEEE* 85(1), pp. 6–23. ISSN: 00189219. DOI: 10.1109/5.554205. arXiv: 9605103 [cs].

Heatly, M. (1986). *Illustrated History of Helicopters*. ISBN: 0-671-07527-6.

Hehn, M. and R. D 'andrea (2015). "Real-Time Trajectory Generation for Quadrocopters". In: *Ieee Transactions on Robotics* 31(4), pp. 877–892. ISSN: 1552-3098. DOI: 10.1109/TRO.2015.2432611.

Hoffmann, G. M., S. L. Waslander, and C. J. Tomlin (2008). "Quadrotor Helicopter Trajectory Tracking Control". In: *Electrical Engineering* 44(August), pp. 1–14. ISSN: 0022-3727. DOI: 10.1088/0022-3727/44/20/205001. URL: http://hoffmann.stanford.edu/papers/GNC08{_}QuadTraj.pdf.

Hofmann-Wellenhof, B., H. Lichtenegger, and J. Collins (1994). *Global Positioning System : Theory and Practice*. Springer Vienna. ISBN: 3709133114.

Krossblade Aerospace (2017). *History of Quadcopters and Multirotors — Krossblade Aerospace Systems*. URL: http://www.krossblade.com/history-of-quadcopters-and-multirotors/ (visited on 01/11/2017).

Lee, D., T. C. Burg, D. M. Dawson, D. Shu, B. Xian, and E. Tatlicioglu (2009). "Robust tracking control of an underactuated quadrotor aerial-robot based on a parametric uncertain model". In: *Conference Proceedings - IEEE International Conference on Systems, Man and Cybernetics*(October), pp. 3187–3192. ISSN: 1062922X. DOI: 10.1109/ICSMC.2009.5346158.

Luukkonen, T. (2011). "Modelling and Control of Quadcopter". In: *Journal of the American Society for Mass Spectrometry* 22(7), pp. 1134–45. ISSN: 1879-1123. DOI: 10.1007/s13361-011-0148-2.

Maccarleyi, C. A. (1984). "State-feedback control of non-linear systems]". In:

Maciejowski, J. (2002). *[Jan_Maciejowski]_Predictive_Control_with_Constrai(BookZZ.org).pdf*.

Magnusson, N. and T. Odenman (2012). "Improving absolute position estimates of an au- tomotive vehicle using GPS in sensor fusion". In: p. 82.

Mellinger, D. (2011). "Trajectory Generation and Control for Quadrotors". In: *2011 IEEE International Conference on Robotics and Automation*, pp. 2520–2525. ISSN: 1050-4729. DOI: 10.1109/ICRA.2011.5980409.

Morbidi, F., R. Cano, D. Lara, F. Morbidi, R. Cano, D. Lara, M.-e. P. Generation, F. Morbidi, R. Cano, and D. Lara (2016). "Minimum-Energy Path Generation for a Quadrotor UAV To cite this version : Minimum-Energy Path Generation for a Quadrotor UAV". In: *International Conference on Robotics and Automation*(May), pp. 2–8. ISSN: 10504729. DOI: 10.1109/ICRA.2016.7487285.

Nowakowski, T. (2015). *China launches Long March 3B rocket with Beidou-3 navigation satellite - SpaceFlight Insider*. URL: http://www.spaceflightinsider.com/

missions/commercial/china-launches-long-march-3b-rocket-with-beidou-3-navigation-satellite/ (visited on 01/31/2017).

Ogata, K. (1970). *Modern Control Engineering*. Vol. 17, p. 912. ISBN: 9780136156734. DOI: 10.1109/TAC.1972.1100013. arXiv: 0605511 [cond-mat]. URL: http://www.pearsonhighered.com/educator/product/Modern-Control-Engineering/9780136156734.page.

Parikh, N. N., S. C. Patwardhan, and R. D. Gudi (2012). *Closed loop identification of quadruple tank system using an improved indirect approach*. Vol. 8. PART 1. IFAC, pp. 355–360. ISBN: 9783902823052. DOI: 10.3182/20120710-4-SG-2026.00177. URL: http://dx.doi.org/10.3182/20120710-4-SG-2026.00177.

Pawelsky (2013). *DJI NAZA GPS communication protocol - NazaDecoder Arduino library - RC Groups*. URL: https://www.rcgroups.com/forums/showthread.php?1995704-DJI-NAZA-GPS-communication-protocol-NazaDecoder-Arduino-library (visited on 03/23/2018).

Polischuk, G. M., V. I. Kozlov, V. V. Ilitchov, a. G. Kozlov, V. Bartenev, V. E. Kossenko, N. a. Anphimov, S. Revnivykh, S. B. Pisarev, a. E. Tyulyakov, B. V. Shebshaevitch, a. B. Basevitch, and Y. L. Vorokhovsky (2002). "The Global Navigation Satellite System GLONASS: Development And Usage In The 21st Century". In: *34th Annual Precise Time and Time Interval (PTTI) Meeting*, pp. 151–160. URL: http://www.dtic.mil/cgi-bin/GetTRDoc?Location=U2{\&}doc=GetTRDoc.pdf{\&}AD=ADA484380.

Richter, C., A. Bry, and N. Roy (2013). "Polynomial trajectory planning for quadrotor flight". In: *International Conference on Robotics and Automation*(Isrr), pp. 1–16. URL: http://www.michigancmes.org/papers/roy7.pdf.

Ritz, R., M. Hehn, S. Lupashin, and R. D'Andrea (2011). "Quadrocopter performance benchmarking using optimal control". In: *IEEE International Conference on Intelligent Robots and Systems*, pp. 5179–5186. ISSN: 2153-0858. DOI: 10.1109/IROS.2011.6048382.

Spooner, S. (1924). "1956 - 1564 - Flight Archive". In: *Flight*. URL: http://www.flightglobal.com/pdfarchive/view/1956/1956-1564.html.

Vlassenbroeck, J. and R. Van Dooren (1988). "A Chebyshev technique for solving nonlinear optimal control problems". In: *IEEE Transactions on Automatic Control* 33(4), pp. 333–340. ISSN: 00189286. DOI: 10.1109/9.192187. URL: http://ieeexplore.ieee.org/document/192187/.

Welch, G. and G. Bishop (2006). "An Introduction to the Kalman Filter". In: *In Practice* 7(1), pp. 1–16. ISSN: 10069313. DOI: 10.1.1.117.6808.

Yakimenko, O. A. (2006). “Direct Method for Real-Time Prototyping of Optimal Control”. In: *Proceedings of the International Conference "Control 2006"*. URL: http://www.nps.navy.mil/faculty/yakimenko/RapidPrototyping/Control2006-Yakimenko-ID190.pdf.

APPENDIX A

SENSOR FUSION CODE (ARDUINO)

```
1  #include <NazaDecoderLib.h>
2
3  int k = 0;
4  int h = 0;
5  int n = 0;
6  int m = 0;
7  int o = 0;
8  int newData = 0;
9  int newData2 = 0;
10 int ready1 = 0;
11 int ready2 = 0;
12 int data;
13 int payload[60];
14 int trama[1000];
15 int trama2[1000];
16 int start = 0;
17 int start2 = 0;
18 double latitude2;
19 double lat;
20 double longitude2;
21 double lon;
22 double altitude2;
23 double alt;
24 double absoluteLat;
25 double prevLat2;
26 double absoluteLon;
27 double prevLon2;
28 double absoluteAlt;
29 double prevAlt2;
30 int ini = 0;
31 uint8_t mask;
```

```
32
33 int seq = 0;
34 int cnt = 0;
35 int msgId = 0;
36 int msgLen = 0;
37 int cs1 = 0;
38 int cs2 = 0;
39
40 double lati;
41 double longi;
42 double gpsAlt;
43 double spd;
44 //NazaDecoderLib::fixType_t  NazaDecoderLib::getFixType() { return fix; }
45 uint8_t sat;
46 double headingNc;
47 double cog;
48 double gpsVsi;
49 double hdop;
50 double vdop;
51 uint8_t year;
52 uint8_t month;
53 uint8_t day;
54 uint8_t hour;
55 uint8_t minute;
56 uint8_t second;
57
58
59 //kalman state struct
60 struct Kstate{
61   double q; //process noise covariance
62   double r[2][2]; //measurement noise covariance
63   double x; //value
64   double p; //estimation error covariance
65   double k[1][2]; //kalman gain
66 };
67 struct Kstate Lat, Lon, Alt;
68
69 double C[2][1];
70 double Z[2][1];
71
72 //auxiliar matrices
73 double Ct[1][2];
74 double PxCt[1][2];
75 double CxP[2][1];
76 double CxPxCt[2][2];
77 double PeR[2][2];
78 double CxE[2][1];
79 double ZmE[2][1];
80 double KxZ[1][1];
81 double KxC[1][1];
```

```
82
83  //kalman vars
84  double q = 0.00001;
85  double r1 = 0.01;
86  double r2 = 0.01;
87  double p = 0;
88  double init_lat = 38.6604170;
89  double init_lon = -9.2050560;
90  double init_alt = 145.000;
91
92  //kalman function declarations
93  void kalman_init(Kstate* result, double Q, double R1, double R2,...
94   double P, double intial_value);
95
96  void kalman_update(Kstate* state, double measurement1, double measurement2);
97
98  //kalman functions
99  void kalman_init(Kstate* result, double Q, double R1, double R2,...
100  double P, double intial_value)
101 {
102   result->q = Q;
103   result->r[0][0] = R1*R1;
104   result->r[0][1] = 0;
105   result->r[1][0] = 0;
106   result->r[1][1] = R2*R2;
107   result->p = P;
108   result->x = intial_value;
109 }
110
111 void kalman_update(Kstate* state, double measurement1, double measurement2)
112 {
113                       //prediction phase
114   //omit x = x
115   state->p = state->p + state->q;
116
117                       //measurement update
118   //Z = [Z1(k);Z2(k)];
119   Z[0][0] = measurement1;
120   Z[1][0] = measurement2;
121
122                       //update phase
123   //state->k = state->p / (state->p + state->r);
124   //K = Pka*C'*((C*Pka*C'+R)^-1)
125   Trans((double*)C, 2, 1, (double*)Ct);
126   Mult(&state->p, (double*)Ct, 1, 1, 2, (double*)PxCt);
127   Mult((double*)C, &state->p, 2, 1, 1, (double*)CxP);
128   Mult((double*)CxP, (double*)Ct, 2, 1, 2, (double*)CxPxCt);
129   Ad((double*)CxPxCt, (double*)state->r, 2, 2, (double*)PeR);
130   Inv((double*)PeR, 2);
131   Mult((double*)PxCt, (double*)PeR, 1, 2, 2, (double*)state->k);
```

```
132
133 //  Print((double*)Z, 2, 1, "Z");
134 //  Print(&state->p, 1, 1, "P");
135 //  Print((double*)C, 2, 1, "C");
136 //  Print((double*)state->k, 1, 2, "K");
137
138   //state->x = state->x + state->k * (measurement - state->x);
139   //Ek = Eka + K*(Z - C*Eka);
140   Mult((double*)C, &state->x, 2, 1, 1, (double*)CxE);
141   Sub((double*)Z, (double*)CxE, 2, 1, (double*)ZmE);
142   Mult((double*)state->k, (double*)ZmE, 1, 2, 1, (double*)KxZ);
143   double KXZ = KxZ[0][0];       //transform 1 by 1 matrix to number
144   state->x = state->x + KXZ;
145
146   //state->p = (1 - state->k) * state->p;
147   //Pk = (1-K*C)*Pka;
148   Mult((double*)state->k, (double*)C, 1, 2, 1, (double*)KxC);
149   double KXC = KxC[0][0];       //transform 1 by 1 matrix to number
150   double mKxC = 1 - KXC;
151   state->p = mKxC*state->p;
152
153 //  Serial.print("P");Serial.println(state->p,6);
154 //  Serial.print("Q");Serial.println(state->q,6);
155 //  Serial.print("R");Serial.println(state->r,6);
156 //  Serial.print("X");Serial.println(state->x,6);
157 //  Serial.print("K");Serial.println(state->k,6);
158 //  Serial.print("M");Serial.println(measurement,6);
159 }
160
161 void codeLong(int idx,double value)
162 {
163   union { uint32_t l; uint8_t b[4]; } val;
164   val.l = value;
165   for(int p = 0; p < 4; p++)
166   {
167     trama[idx + p] = val.b[p] ^ mask ;
168   }
169 }
170
171 void calcNewCS()
172 {
173   union { uint32_t l; uint8_t b[4]; } CS1;
174   union { uint32_t l; uint8_t b[4]; } CS2;
175   union { uint16_t l; uint8_t b[2]; } CSA;
176   union { uint16_t l; uint8_t b[2]; } CSB;
177   union { uint32_t l; uint16_t b[2]; } aux;
178   aux.b[1] = 0;
179   for(int p = 0; p < 60; p++)
180   {
181     aux.b[0] = trama[k - 61 + p];
```

```
182 //    Serial.println(aux.l);
183     CS1.l += aux.l;
184     CS2.l += CS1.l;
185   }
186   CSA.b[0] = CS1.b[0];
187   CSA.b[1] = 0;
188   CSB.b[0] = CS2.b[0];
189   CSB.b[1] = 0;
190   trama[k-1] = CSA.l;
191   trama[k] = CSB.l;
192 }
193
194 int32_t decodeLong(uint8_t idx, uint8_t mask)
195 {
196   union { uint32_t l; uint8_t b[4]; } val;
197   for(int i = 0; i < 4; i++) val.b[i] = payload[idx + i] ^ mask;
198   return val.l;
199 }
200
201 int16_t decodeShort(uint8_t idx, uint8_t mask)
202 {
203   union { uint16_t s; uint8_t b[2]; } val;
204   for(int i = 0; i < 2; i++) val.b[i] = payload[idx + i] ^ mask;
205   return val.s;
206 }
207
208 void updateCS(int input)
209 {
210   cs1 += input;
211   cs2 += cs1;
212 }
213
214 double getLat() { return lati; }
215 double getLon() { return longi; }
216 double getGpsAlt() { return gpsAlt; }
217 double getSpeed() { return spd; }
218 //NazaDecoderLib::fixType_t  NazaDecoderLib::getFixType() { return fix; }
219 uint8_t getNumSat() { return sat; }
220 double getHeadingNc() { return headingNc; }
221 double getCog() { return cog; }
222 double getGpsVsi() { return gpsVsi; }
223 double getHdop() { return hdop; }
224 double getVdop() { return vdop; }
225 uint8_t getYear() { return year; }
226 uint8_t getMonth() { return month; }
227 uint8_t getDay() { return day; }
228 uint8_t getHour() { return hour; }
229 uint8_t getMinute() { return minute; }
230 uint8_t getSecond() { return second; }
231
```

```
232
233 void decoder(int input)
234 {
235   if((seq == 0) && (input == 85)) { seq++; }
    // header (part 1 - 0x55)
236   else if((seq == 1) && (input == 170)) { cs1 = 0; cs2 = 0; seq++; }
    // header (part 2 - 0xAA)
237   else if(seq == 2) { msgId = input; updateCS(input); seq++; }
    // message id
238   else if((seq == 3) && ((msgId == 16) && (input == 58))) ...
239   { msgLen = input;  cnt = 0; updateCS(input); seq++; }
240   else if(seq == 4) { payload[cnt++] = input; updateCS(input);...
241    if(cnt >= msgLen) { seq++; } }             // store payload in buffer
242   else if(seq == 5) { seq++; }
    // verify checksum #1
243   else if(seq == 6) { seq++; }
    // verify checksum #2
244   else seq = 0;
245
246   if(seq == 7) // all data in buffer
247   {
248     seq = 0;
249     // Decode GPS data
250     if(msgId == 16)
251     {
252       uint8_t mask2 = payload[55];
253       uint32_t time = decodeLong(0, mask2);
254       second = time & 0b00111111; time >>= 6;
255       minute = time & 0b00111111; time >>= 6;
256       hour = time & 0b00001111; time >>= 4;
257       day = time & 0b00011111; time >>= 5; if(hour > 7) day++;
258       month = time & 0b00001111; time >>= 4;
259       year = time & 0b01111111;
260       longi = (double)decodeLong(4, mask2) / 10000000;
261       lati = (double)decodeLong(8, mask2) / 10000000;
262       gpsAlt = (double)decodeLong(12, mask2) / 1000;
263       double nVel = (double)decodeLong(28, mask2) / 100;
264       double eVel = (double)decodeLong(32, mask2) / 100;
265       spd = sqrt(nVel * nVel + eVel * eVel);
266       cog = atan2(eVel, nVel) * 180.0 / 3.1415;
267       if(cog < 0) cog += 360.0;
268       gpsVsi = -(double)decodeLong(36, mask2) / 100;
269       vdop = (double)decodeShort(42, mask2) / 100;
270       double ndop = (double)decodeShort(44, mask2) / 100;
271       double edop = (double)decodeShort(46, mask2) / 100;
272       hdop = sqrt(ndop * ndop + edop * edop);
273       sat  = payload[48];
274       uint8_t fixType = payload[50] ^ mask2;
275       uint8_t fixFlags = payload[52] ^ mask2;
276 //      switch(fixType)
```

```
277 //      {
278 //        case 2 : fix = FIX_2D; break;
279 //        case 3 : fix = FIX_3D; break;
280 //        default: fix = NO_FIX; break;
281 //      }
282 //      if((fix != NO_FIX) && (fixFlags & 0x02)) fix = FIX_DGPS;
283       newData2 = 1;
284     }
285   }
286 }
287
288
289 void setup()
290 {
291   Serial.begin(57600);
292   Serial1.begin(115200);
293   Serial2.begin(115200);
294   Serial3.begin(115200);
295
296   //Kalman init
297   kalman_init(&Lat,q,r1,r2,p,init_lat);
298   kalman_init(&Lon,q,r1,r2,p,init_lon);
299   kalman_init(&Alt,q,r1,r2,p,init_alt);
300
301   //C matrix setup
302   C[0][0] = 1;
303   C[1][0] = 1;
304 }
305
306 void loop()
307 {
308
309   if(Serial1.available())
310   {
311     if(start == 1)
312     {
313       if(k > 999)
314       {
315         Serial3.write(trama[0]);
316 //        Serial.println(trama[0]);
317       }
318       else
319       {
320         Serial3.write(trama[k+1]);
321 //        Serial.println(trama[k+1]);
322       }
323     }
324 //    Serial.println(trama[k]);
325     trama[k] = Serial1.read();
326     int input = trama[k];
```

```
327     uint8_t decodedMessage = NazaDecoder.decode(input);
328
329     switch (decodedMessage)
330     {
331       case NAZA_MESSAGE_GPS:
332       newData = 1;
333     }
334
335     if(newData == 1)
336     {
337       mask = trama[k-4];
338
339       newData = 0;
340       ready1 = 1;
341     }
342
343     k++;
344
345     if(k>=1000)
346     {
347       k = 0;
348       start = 1;
349     }
350   }
351
352   if(Serial2.available())
353   {
354     trama2[h] = Serial2.read();
355
356     if(start2)
357     {
358       if(h > 999)
359         decoder(trama2[0]);
360       else
361         decoder(trama2[h+1]);
362     }
363
364     if(newData2)
365     {
366 //      Serial.print(getHour());Serial.print(":");
367 //      Serial.print(getMinute());Serial.print(":");
368 //      Serial.print(getSecond());Serial.print(" - ");
369 //      Serial.print("GPS2 - ");
370 //      Serial.print(getLat(), 7);
371 //      Serial.print("    ");Serial.println(h);
372       latitude2 = getLat();
373       longitude2 = getLon();
374       altitude2 = getGpsAlt();
375
376       newData2 = 0;
```

```
377         ready2 = 1;
378       }
379
380       h++;
381
382       if(h>=1000)
383       {
384         h = 0;
385         start2 = 1;
386       }
387     }
388
389     if(ready1 && ready2)
390     {
391
392         if(!ini)
393         {
394           prevLat2 = latitude2;
395           prevLon2 = longitude2;
396           prevAlt2 = altitude2;
397           ini = 1;
398         }
399
400         absoluteLat = abs(prevLat2 - latitude2);
401         absoluteLon = abs(prevLon2 - longitude2);
402         absoluteAlt = abs(prevAlt2 - altitude2);
403
404 //        Serial.println(absoluteLat);
405 //        Serial.println(absoluteLon);
406 //        Serial.println(absoluteAlt);
407
408         if(absoluteLat > 0.00002)
409           latitude2 = prevLat2;
410
411         if(absoluteLon > 0.00002)
412           longitude2 = prevLon2;
413
414         if(absoluteAlt > 2)
415           altitude2 = prevAlt2;
416
417
418 //        Serial.print(NazaDecoder.getHour());Serial.print(":");
419 //        Serial.print(NazaDecoder.getMinute());Serial.print(":");
420 //        Serial.print(NazaDecoder.getSecond());Serial.print(" - ");
421
422 //        Serial.print("Lat1 - ");
423         Serial.print(NazaDecoder.getLat(), 7);Serial.print(";");
424 //        Serial.print("Lat2 - ");
425         Serial.print(latitude2, 7);Serial.print(";");
426 //        Serial.print("LatF - ");
```

```
427       kalman_update(&Lat, NazaDecoder.getLat(), latitude2);
428       Serial.print(Lat.x, 7);Serial.print(";");
429       lat = Lat.x*10000000;
430
431 //      Serial.print("Lon1 - ");
432       Serial.print(NazaDecoder.getLon(), 7);Serial.print(";");
433 //      Serial.print("Lon2 - ");
434       Serial.print(longitude2, 7);Serial.print(";");
435 //      Serial.print("LonF - ");
436       kalman_update(&Lon, NazaDecoder.getLon(), longitude2);
437       Serial.print(Lon.x, 7);Serial.print(";");
438       lon = Lon.x*10000000;
439
440 //      Serial.print("Alt1 - ");
441       Serial.print(NazaDecoder.getGpsAlt(), 3);Serial.print(";");
442 //      Serial.print("Alt2 - ");
443       Serial.print(altitude2, 3);Serial.print(";");
444 //      Serial.print("AltF - ");
445       kalman_update(&Alt, NazaDecoder.getGpsAlt(), altitude2);
446       Serial.print(Alt.x, 3);Serial.print(";");
447       alt = Alt.x*10000000;
448
449
450 //      Serial.print("NumSat1 - ");
451       Serial.print(NazaDecoder.getNumSat());Serial.print(";");
452 //      Serial.print("NumSat2 - ");
453       Serial.print(getNumSat());Serial.println(";");
454
455 //      Serial.println(k);
456
457       prevLat2 = latitude2;
458       prevLon2 = longitude2;
459       prevAlt2 = altitude2;
460
461       if(k<53)
462       {
463         n = k+946;
464         codeLong(n, lat);
465       }
466
467       if(k>=53)
468       {
469         n = k-52;
470         codeLong(n, lat);
471       }
472
473       if(k<57)
474       {
475         m = k+942;
476         codeLong(m, lon);
```

```
477         }
478
479         if(k>=57)
480         {
481           m = k-56;
482           codeLong(m, lon);
483         }
484
485         if(k<49)
486         {
487           o = k+950;
488           codeLong(o, alt);
489         }
490
491         if(k>=49)
492         {
493           o = k-48;
494           codeLong(o, alt);
495         }
496
497         calcNewCS();
498         ready1 = 0;
499         ready2 = 0;
500     }
501 }
```

APPENDIX B

SIMULINK - BLOCK DIAGRAM OF THE CONTROL ARCHITECTURES

This chapter will present the control architectures developed in *MATLAB R2014b - Simulink*.

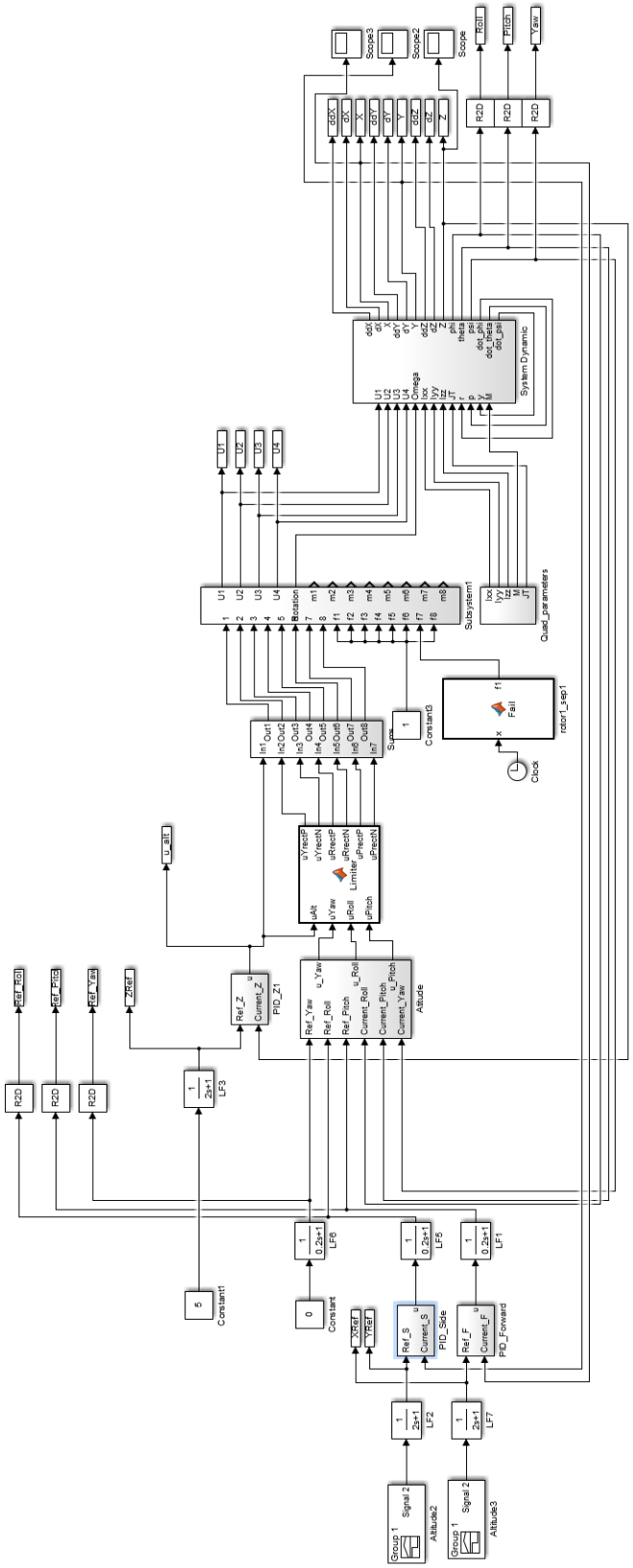

Figure B.1: Simulink block diagram of the PID architecture.

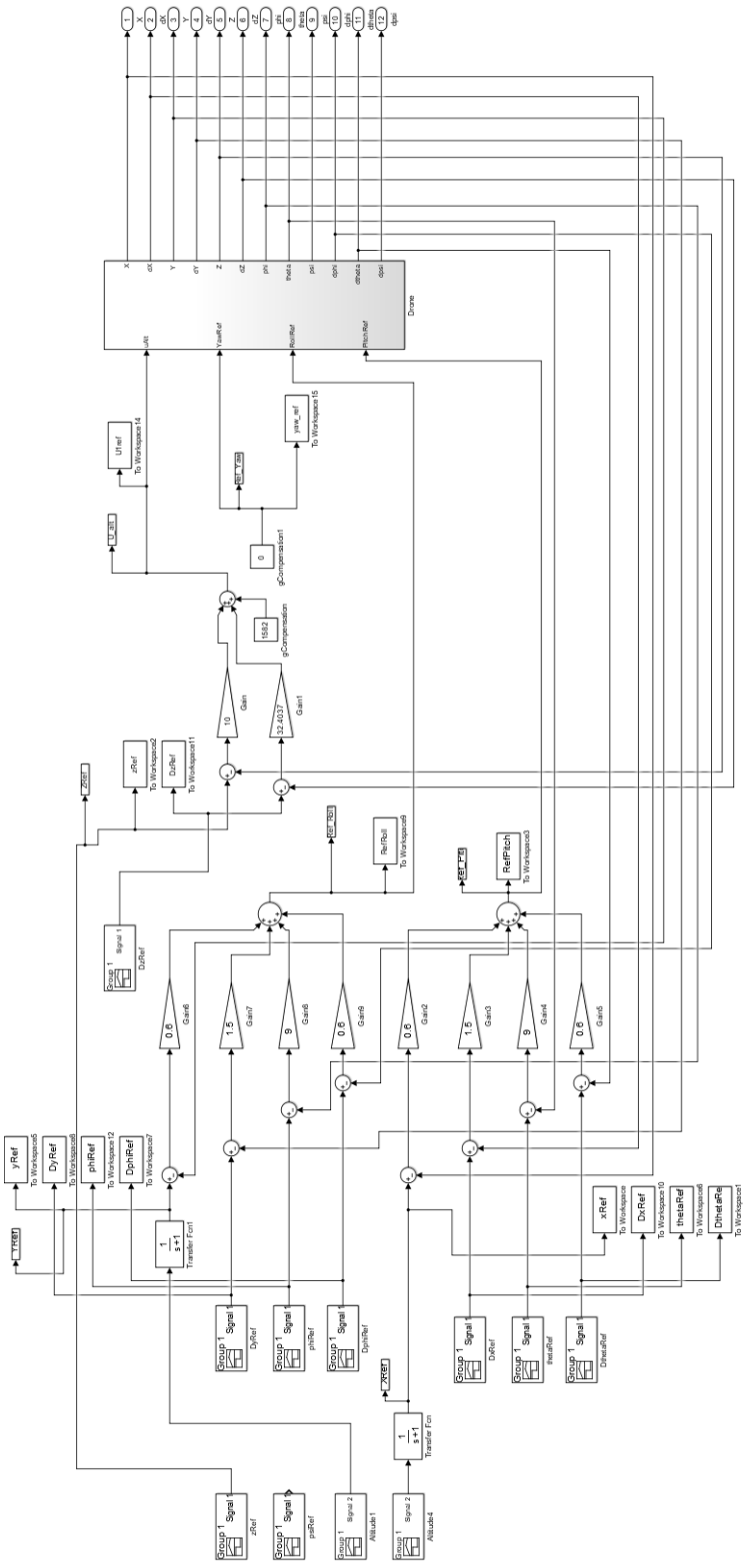

Figure B.2: Simulink block diagram of the state space feedback architecture.

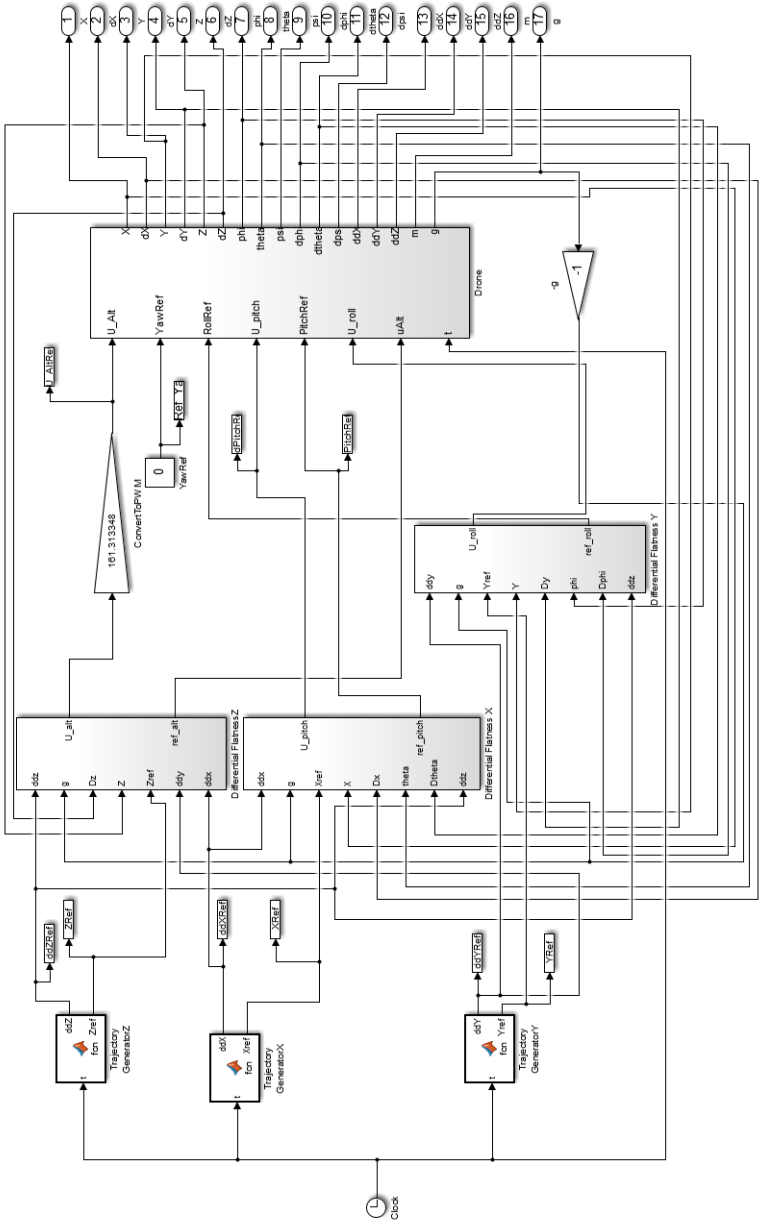

Figure B.3: Simulink block diagram of the differential flatness architecture.