Modeling and Identification of Crazyflie 2.0

Bachelor-Thesis

Guillem Montilla Garcia

TECHNISCHE
UNIVERSITÄT
DARMSTADT

Fachbereich ETiT
Institut für Automatisierungstechnik
Prof. Dr.-Ing. Jürgen Adamy

Modeling and Identification of Crazyflie2.0
Bachelor-Thesis

Eingereicht von Guillem Montilla Garcia
Tag der Einreichung: 27. September 2017

Gutachter: Prof. Dr.-Ing. Jürgen Adamy
Betreuer: Raul Godoy

Technische Universität Darmstadt
Fachbereich ETiT
Institut für Automatisierungstechnik
Prof. Dr.-Ing. Jürgen Adamy

Ehrenwörtliche Erklärung

Hiermit versichere ich, die vorliegende Bachelor-Thesis ohne Hilfe Dritter und nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die aus den Quellen entnommen wurden, sind als solche kenntlich gemacht worden. Diese Arbeit hat in dieser oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Darmstadt, den 27. September 2017 Guillem Montilla Garcia

Contents

List of Figures

List of Tables and Codes

Tables:

Codes:

List of Variables

State variables:

x	State vector
x, y, z	Translational movement
ϑ, φ, ψ	Pitch angle (y-axis), roll angle (x-axis) and yaw angle (z-axis)
R	Rotation Matrix
u, y	Input vector, output vector
A	State matrix
B, C	Input and output matrix
G (s)	Transfer function
n_0, d_0, d_1	Parameters of the transfer function
v_x, v_y	Velocity in x-direction and y-direction
$\sum \boldsymbol{T}$	Total thrust of the rotors
m	Mass of the quadcopter
a	Acceleration of the quadcopter
g	Gravity acceleration
F	Forces applied to the quadcopter

Abstract

In this thesis, it is realized the modeling and identification of a modified *Crazyflie 2.0* mini-quadcopter and it is created a generic methodology to model and identify other quadcopters.

Crazyflie 2.0 is a versatile open source flying development platform produced by *Bitcraze* with reduced weight and size that is equipped with low-latency/long-range radio and Bluetooth LE that can be used to communicate the platform to mobiles, computers or radiofrequency controllers.

The *Crazyflie 2.0* used on this thesis contains a camera attached on the body frame in order to obtain the relative position from an ArUco marker and it does not behave as the standard version of the quadrotor.

The quadcopter is controlled by a ground station that runs the ROS package *crazyflie-ros* to establish an autonomous control of the quadrotor and the experiments realized to obtain the model of the vehicle are based on this package.

The data obtained from different experiments is processed by Matlab to obtain the desired transfer functions to identify the platform and there are compared two ways to obtain these transfer functions to know in future scenarios which is better to use.

After the model of the quadrotor is found, a simulation platform is created to make a study of the quadcopter flight behavior with the objective to improve the control.

To improve the control of the quadcopter, new PID values are found for the position controller that make the quadcopter perform better. This simulation platform includes a 3D trajectory plotter that can show in three dimensions the evolution of the vehicle's position and it can be extrapolated to other quadcopters.

At the end of the thesis are shown the result of the improvements of the platform and in which applications could be used.

1 Introduction

The Unmanned Aerial Vehicle (UAV), known it also as drone, is an unmanned aircraft that can navigate autonomously, without human control or beyond line of sight.

The firsts UAV started to appear in the First World War as a military tool that was used to realize shooting drills and also to defend versus zeppelins.

In the Second World War, the UAV were improved but when the real revolution started was in around XX century in the Gulf War. In that war, the UAV started to have all the tools that we can see in them nowadays. [3]

The basic characteristics or components that must have an UAV are:

- Body: that component of the UAV is the one that join all the other components and it can be designed in a way to be more aerodynamic.
- Power Supply: this part is in charge to store and distribute all the energy needed to feed the UAV.
- Sensors: these components are in charge to obtain different data from the environment in order to be able to interact with the surroundings.
- Actuators: these components are the ones in charge to make the UAV fly and modify the movement trajectories.
- Software and computing: UAV software, called the flight stack or autopilot, are real-time systems that require rapid response to process the data of the sensors in order to obtain a good flight behavior.
- Loop control: the loop control is used to correct the errors that appear between the wanted behavior, movement and positioning of the UAV and the real ones.
- Communications: communications are in charge to establish a connection between the computing, the sensors and the actuators.

The UAV can be classified according to different characteristics in four categories:

The first category is the fixed-wings UAV, which refer to UAV that need a runway to realize the take-off and landing. Generally, these kinds of vehicles have long endurance and can fly at high speeds.

Rotary-wing UAV, also called rotorcraft UAV, are the types of UAV that have the skill to realize a vertical take-off and landing. Rotorcrafts have the skill of hovering and a high maneuverability.

Next category is the blimps. These UAV are balloons and airships, which are lighter than air and float on the air, have a long endurance and low speed and normally have a large size.

Flapping-wings UAV are the type of UAV that tries to imitate the flying way of some animals like birds or insects and they have the wings inspired on that. [2]

Nowadays the most popular applications for the UAV are on military, surveying and mapping, video and photography, search and rescue and finally on research and investigation

One of the most emerging rotorcraft concept for unmanned aerial vehicle (UAV) are the quadrotors, also known as quadcopters.

The design of that vehicles consists of four rotors that are located on the corners of the quadcopter that have two pairs of counter-rotating.

The first quadcopter that presents and stable flight was created by Dr. George de Bothezat and Ivan Jerome in 1922. Some experimental versions were created before but they did not present a long time of flight and the control of the vehicle was not good. [10]

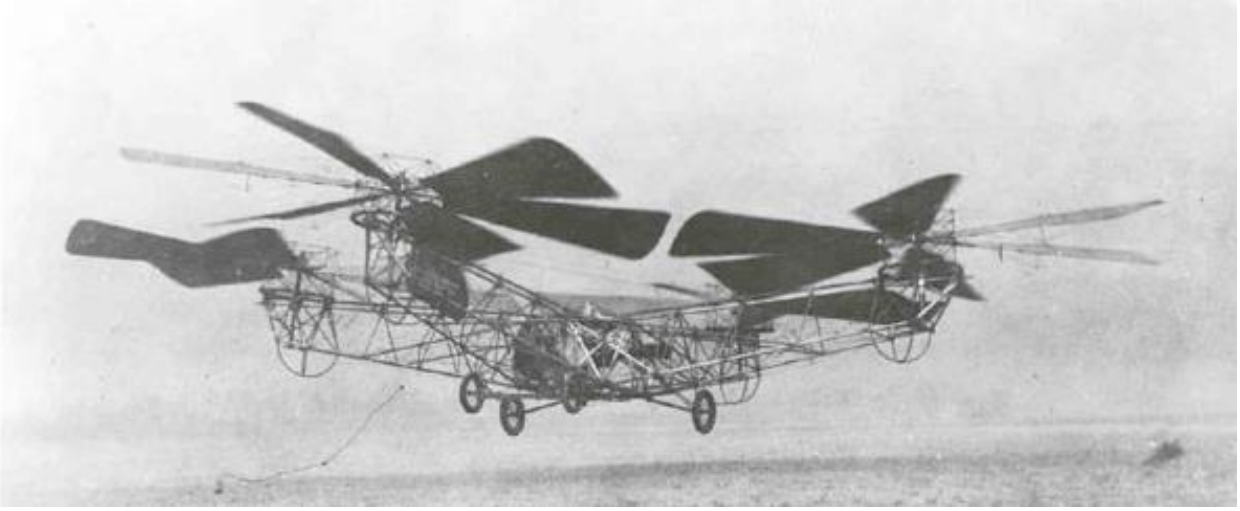

Figure 1.1: This image shows the first quadrotor created by Dr. George de Bothezat and Ivan Jerome while it was realizing its first flight in 18 of December of 1922. Extracted from [10].

This quadrotor had a dimension of 20x20x3m and it was equipped with 4 six bladed rotors located at the end of a X-shaped structure.

The vehicle weighed 1633 Kg and it could achieve a speed around 48 Km/h with a performance time of 2 minutes and 45 seconds.

From that first model, the quadrotors were keep evolving and nowadays, the research community has a particular interest in that kind of UAV because it has some important advantages versus the other flying platform models.

The most important reason is that the quadcopters have an easy and simple mechanical structure because the rotors do not require a complex mechanical control linkage to operate, and that simplifies the design and maintenance of that kind of drones.

Also, another important reason is that the quadcopters have a vertical takeoff and landing, the ability to hover and some models can be small.

The combination of that three characteristics make this type of vehicle perfect for flying indoors places, like laboratories, where it is easy to realize experiments. [1]

1.1 Objectives

The goal of this thesis is to realize a modeling and identification of the *Crazyflie 2.0* that will able, in a future, to realize a better autonomous control of the flying platform in an indoor environment using the *Crazyflie-Ros* package.

A secondary objective of the thesis is to improve the flying behavior of the *Crazyflie* 2.0 when it uses an ArUco marker finding new values for the PID that control the position of the vehicle in order to be faster, smother and more accurate.

Problem overview

The model of Crazyflie 2.0 used in the Robotic Laboratory of TU Darmstadt is not a standard version of this platform. This version contains a camera assembled to the platform that points to the floor.

Having this extra camera affects at the normal behavior of the quadcopter and it presents a different response to the same input signal than the normal one.

The *Crazyflie 2.0* is a versatile flying development platform and it could offer a lot of research possibilities but without a good modeling of the vehicle, the doors for future applications are closed

Application

The work done in this project, could be useful in the future to realize an autonomous control platform for this quadrotor.

It is expected that with the model of the quadcopter extracted in this thesis, one can obtain a very accurate position controller for the *Crazyflie 2.0* that can be used in other future applications that requires a control of the vehicle using fiducial marker or extern pose estimation.

Framework

To achieve the final objective of this thesis, that it is to realize a modeling and identification of the quadrotor, some processes need to be done.

Firstly, the quadrotor must be programed to realize some testing flights. Then, when the quadrotor is able to fly, the next step is to extract the data from the sensors of the drone and save it.

Once the data of the different test flights is recorded, the next step is to process that information to acquire the different parameters desired to obtain a good model.

Finally, it is used different tools to obtain the transfer functions needed to identify the quadrotor and it is done a comparison between the real behavior of the quadrotor and the behavior obtained by simulation using the obtained transfer functions of the model.

1.2 Outline

To achieve the mentioned goals this thesis is divided into 10 chapters.

1. **Chapter 1** focuses on making an introduction to the UAV and explaining the objectives and problems that are attended on this thesis.

2. **In chapter 2** it is explained the laboratory environment and all the things that are necessary in order to try to reply the experiments realized on this thesis and also it is introduced ROS.

3. **Chapter 3** describes the platform Crazyflie 2.0 that is the one used to realize the experiments. In this chapter, it is explained the hardware, the different sensors that has the platform and how it works the software and the control structure of this one.

4. **In chapter 4** the dynamics of the quadcopter and the equations of motion are explained. These equations can be expressed as a linearized model. This linearized model can be identified for the Crazyflie 2.0 realizing experimental flights.

5. **Chapter 5** shows the different linearized models of the Crazyflie and the results of the experiments that were needed to obtain these models. This chapter contains also an explanation of how there were realized the different experiments and shows a *Matlab* platform that can simulate the behavior of the quadcopter.

6. **Chapter 6** explain the different processes to transform the data obtained from the Crazyflie2.0 into the desired information needed to be able to obtain the transfer functions of the quadrotor.

7. **Chapter 7** explains the process done to improve the flight of the quadcopter and make a comparison between how flew before and after the improvements.

8. **Chapter 8** explains how can this work help in future projects and what could be the applications of that work.

9. **Chapter 9** shows the different conclusions obtained from this work and how could have been improve this work.

10. At the end of the thesis is shown all the bibliography used on this thesis.

1.3 Crazyflie 2.0

The *Crazyflie 2.0* is a versatile open source flying development platform produced by *Bitcraze* that only weighs 27g and has a 92x92x29mm dimensions.

Crazyflie 2.0 is equipped with low-latency/long-range radio and Bluetooth LE that can be used to communicate the platform to mobiles, computers or radiofrequency controllers.

That platform is ideal for researchers because of its low price, adaptability, resistance and, due the low weight, it is very safe to fly on indoor places.

The version of Crazyflie 2.0 used on this thesis is not the standard platform sold by *Bitcraze*. The platform used is a customized version that contains a camera in order to control the pose.

More details of the platform and the modifications are shown in Chapter 3.

2 Laboratory Set-up

In this chapter, it is described the equipment used in the TU Darmstadt robotic laboratory to realize the different experiments of this work.

In section 2.1 appears all the necessary knowledge for other researchers that want to realize an identical set-up.

In section 2.2 it is provided an overview about ROS and which packages were used to realize the different experiments.

In section 2.3 it is presented a list of the used standard units and the coordinate frame used to show the results.

2.1 Laboratory configuration

The experiments were realized in 2 different locations. The first one was the Roboterlabor at RMR TU Darmstadt. The Roboterlabor is a room of 5x5m with a height of 2.8m. In that room were realized all that experiments that did not need a long horizontal translation distance.

The second location for the experiments was the 4th floor corridor of the RMR TU Darmstadt building. This place has a long length that makes possible the realization of long translations distance experiments.

During the course of the experiments the next equipment was used:

1. Ground station: the ground station was computer with *Ubuntu 16.04 LTS* and has connected a *Crazyradio PA*, in order to send and receive information of the *Crazyflie2.0*, a *Logitech F710 Wireless Gamepad*, in order to control the flight modes of the quadrotor, and a video receiver *ImmersionRC UNO5800*.

2. *Crazyflie 2.0* modified: as was mentioned before, the quadrotor used in this project is not a standard one. The model used is a version made by some students of TU Darmstadt [6].

   That model contains a camera model *FX797T* from *FXT Technologies* suited in a customized casing that points to the floor in order to know the position of the platform.

3. Flying Area: to realize the experiments, it is needed a ArUco marker that is going to be localized by the camera attached to the quadrotor.

   The ArUco markers had a dimension of 197x197mm and each experiment had his own configuration of this markers. Also, in order to obtain good results, it was tried to minimize the wind turbulence in the different areas where the quadrotor flew.

For the communication between the Crazyflie 2.0 and the ground station it was created a radio channel using the *uri* radio: "*//0/120/2M/E7E7E7E701".*

2.2 ROS

Robot operating system, also known as ROS, is an open source software-framework for robots that provides the services that one expects from an operating system including hardware abstraction, low-level device control, implementation of commonly-used functionality, message-passing between processes and package management.

It also provides tools and libraries for obtaining, building, writing, and running code across multiple computers. [7]

Nowadays, ROS is the standard tool focused on robotic research and investigation and has a big community of researchers that discuss and publish their packages to extend the features and resources of ROS.

ROS has a modular distribution, that means that the user can run more than one process simultaneously.

To understand how ROS works, it is going to be shown the main concepts and how they work.

Packages: the main unit for organize software in ROS are the packages. The packages contain ROS runtime nodes, a ROS library, datasets, configuration files or anything else that is usefully organized.

Nodes: The processes that are working on ROS have the name of nodes. As it was said before, ROS is designed to be modular, that is why normally there are going to be running many nodes at the same time.

The nodes are executed, compiled and controlled individually and they have only one purpose. Nodes can publish and subscribe to a topic.

Topics: The name that is used to identify the content of the message it is called topic. Nodes communicate between them using topics and the ratio between publisher subscriber is 1 – N, one publisher and infinite subscribers.

Publishers and Subscribers: Nodes can be classified as publishers, subscribers or both. The publishers are that nodes that send topics to other nodes.

On contrary, the nodes that receive the topics are called subscribers. Note that one publisher can be also a subscriber of another topic.

Messages: a message is simply a data structure comprising typed fields. These messages are sent via topic and can be primitive types (integer, Boolean, floating point, etc.) or an array of these ones.

Services: the message that are used to synchronize the process of request-reply between the nodes receive the name of services. Services are defined by a couple of messages, one for request and another for reply.

Bags: The format for recording and playing back ROS message data is called bag. Bags are useful to process the data received of the experiments realized by ROS. One of the best points of the bag format is that can be edited by Matlab.

ROS Master: ROS Master is who stays in charge to run all the nodes and stablish the necessary communications. ROS Master provides the necessary information to the nodes to make possible the transmission of messages between them.

Every time a node appears, it connects to the master to register the details of the transmission, like which topics are sent or which topics wants to be received, and when appears a match, it stablishes the pertinent communication.

2.2.1 Example of how ROS works

To make better the comprehension of how ROS works it is going to be shown and easy example with explanations of a simple ROS execution.

This example consists of 2 nodes, the *Camera* node and the *Image_viewer* node, and it pretends to simulate a real ROS package where one camera takes an image and shows it.

As it was mentioned, every node has a specific task, in this case the *Camera* node is in charge to take a picture and the *Image_viewer* node is in charge to show that picture.

At the beginning of the process all the nodes of the package are going to be executed and it starts the checking process of these nodes with the master.

In this case, the *Camera* node says to ROS Master that it is publishing the topic *Images* and it starts to publish it but it cannot be received by any other node.

Figure 2.1: Scheme that represents how ROS notifies that a topic is available.

After that, the *Camera* node is sending the *Image* topic, but anybody is receiving that information but when the *Image_viewer* node is initialized, it is going to start the checking process and then it is going to stablish a communication with the ROS Master and it is going to say that it wants to subscribe to *Images* topic.

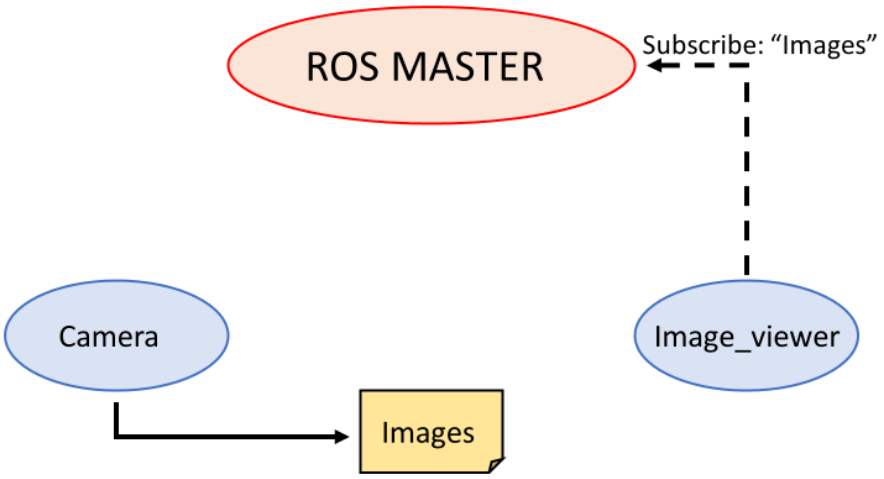

Figure 2.2: Scheme that represents how ROS notifies that a topic is required.

Once all this process has been done, ROS Master detects that there is a match, so it is going to notify to both nodes of that and advice that they can initiate a communication.

When this communication starts, it is going to be establish a relation as a Publisher-Subscriber where the node *Camera* is going to become a publisher that sends the Image topic to the *Image_viewer*, that it is going to become a subscriber.

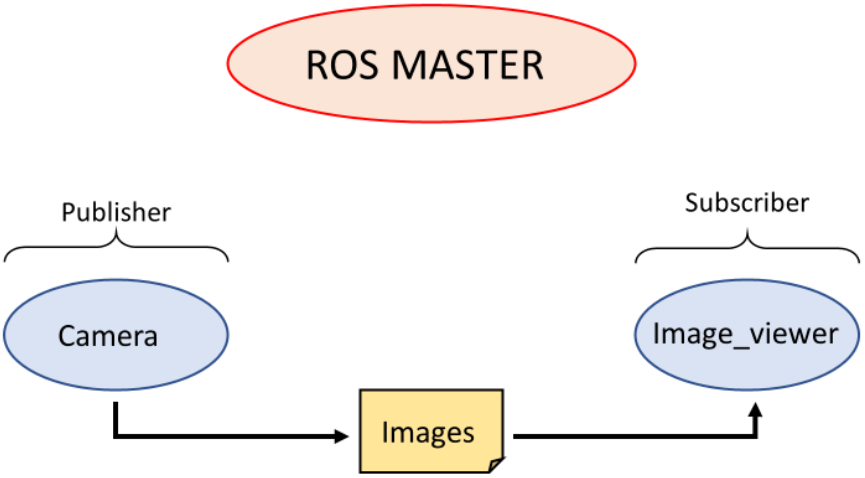

Figure 2.3: Scheme that represents how ROS establish a communication beetwen a publisher and a subscriber.

2.3 Standard Units of Measure and Coordinate Conventions

In this work, all the equations are expressed following the International System of Units to provide a simple and clarified view and understanding.

The coordinate frame used to express the location and orientation of the quadrotor is set-up as the x-direction pointing to the right, the y-direction pointing forward and the z-direction pointing up.

As it was said, all the equations are expressed following the International System of Units but this not applies for the communication of ROS and the quadcopter.

The measurements obtained from the imu of the quadcopter follow the International System of Unit, that means that the angular speed measured by this one is in rad/s and the acceleration is expressed in m/s^2.

On the contrary, the commands or signals that have to be sent to the quadrotor in order to control it are not following that convention.

The angles of reference sent to the quadrotor by ROS have to be in degree and the angular speed in degree per second.

3 Hardware: Quadcopter

3.1 Crazyflie 2.0

The Crazyflie 2.0 was introduced as a second iteration of the open source Crazyflie quadcopter in 2013 by Marcus Eliasson, Arnaud Taffanel and Tobias Antonsson and it is characterized for being a versatile flying development platform with a small size (92x92x29mm) and low weight (27g).

Its advanced functionality and its low price, 200€, make this platform ideal for developers and researchers and with its small size and weight can easily fly in indoor environment like laboratories or houses.

The platform was designed as a solderless kit, that means one can quickly assembled the quadrotor by simply attaching the different parts to the circuit-board frame and it would be ready to fly.

The next sections are going to explain with more detail the different characteristics of the Crazyflie 2.0 platform version used for the realization of the thesis.

In the first section, the hardware of the Crazyflie 2.0 is going to be described and the modifications realized to the quadrotor are going to be explained.

In the second section, the different sensors that contains the platform are going to be listed and described it and the camera attached to the frame of the quadcopter is going to be described.

In the third section, the software of the platform is going to be described and the different ways to control the vehicle are going to be explained.

In the last section, the control architecture used to dominate the trajectory of the quadcopter is going to be explained.

3.1.1 Hardware

The Crazyflie 2.0 is designed as a circuit board-frame in a x-configuration (see in figure 3.1) that has four rotors fixed on the 4 edges of the frame.

The standard boxing of the Crazyflie 2.0 that one can buy in the website of the company contains 1 Crazyflie 2.0 control board, 1 LiPo battery (240mAh), 5 DC motors of 7mm, 6 motor mounts, 10 propellers, 1 battery holder and different male connectors (see in figure 3.1).

Figure 3.1: This image shows the structure of the Crazyflie 2.0 (on the left) and the different components that contain the bundle when Crazyflie 2.0 is bought. Extracted from [9]

The quadcopter weights 27g and has a size of 92x92x29mm and using the battery provided by Bitcraze without modifying the platform, it can fly for 7 minutes and the charging time is 40 minutes.

The motors of the quadcopter have a diameter of 7mm, a length of 16mm and a weight 2.7mm. These motors have a rated voltage of 4.2V, a rated current of 1000mA and moves at 14 000rpm/V.

The propellers of the drone have a size of 45mm with a fits shaft of 0.8mm and there are 2 different models, the Clockwise propellers (CW) and the Counter-Clockwise propellers (CCW).

The battery provided by Bitcraze has a capacity of 240mAh with a nominal voltage of 3.7V per cell, weights 7.1g and has a size of 20x7x30mm without including the cable.

Modifications

The quadcopter used in this thesis was modified in order to be able use a on board camera and it has 2 differences on the hardware in comparison of the original one.

The first difference is that the TU Darmstadt's model has a camera that is attached to the frame using a custom casing.

That casing was designed specially to fit with the camera model *FX979T* from *FXT Technologies* and was created using a 3d printer to reduce the weight as much as possible, only 1.3g, and achieving a perfect assembly with the platform.

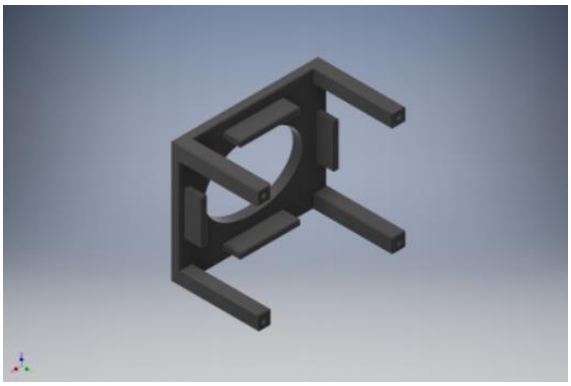

Figure 3.2: This image shows a CAD model of the case used to join the camera FX979T into the Crazyflie 2.0 [6].

The second difference is that to avoid the casing hits the ground, it was added 4 extra ground legs of foam under every rotor.

That pieces of foam weight 1g and have a dimension of 1x1x2cm. In the next figure appears the final assembly of the customized version of the quadcopter.

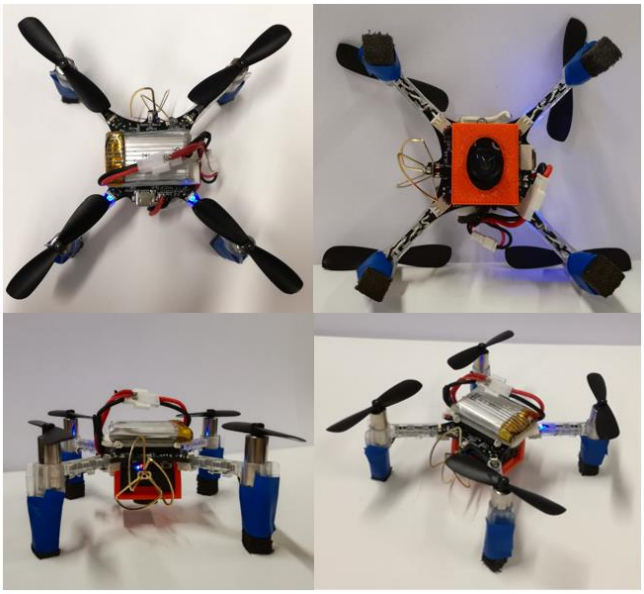

Figure 3.3: Image that shows a perspective of the modified Crazyflie 2.0 and its three plans of view.

3.1.2 Sensor equipment and camera

The original Crazyflie 2.0 is equipped only with an Inertial Measurement Unit (IMU) and a high precision pressure sensor, but the TU Darmstadt customized version has also a camera.

Inertial Measurement Unit (IMU)

The inertial measurement unit of the Crazyflie is the model MPU-9250 and consists of a 3-axis gyroscope, a 3-axis accelerometer and a 3-axis magnetometer.

The gyroscope has a user-programmable full-scale range of ±250, ±500, ±1000 and ±2000 º/s; an operating current of 3.2mA and has incorporated a digitally-programmable low-pass filter.

The accelerometer has a user-programmable full-scale range of ±2g, ±4g, ±8g and ±16g and it has a normal operating current of 450µA.

The magnetometer is a monolithic Hall-effect magnetic sensor and it has a full-scale measurement range of ±4800µT.

Pressure Sensor

The Crazyflie 2.0 has included a LPS25H pressure sensor that it is an ultracompact absolute piezoresistive pressure sensor that includes a monolithic sensing element and a IC interface. This sensor has an absolute pressure range from 260 to 1260 hPa.

Camera

The camera used in the customized version of the Crazyflie 2.0 is the model FX797T from FXT Technology. This camera has a maximum resolution of 720x480 pixels with 30 fps and a viewing angle of 120º.

The overall weight of the camera is 4.5g with dimensions of 20x17 mm and the prize is around 40€.

To power on the camera, it is used the same battery of the quadcopter and that makes the battery discharge early and the flying time decreases from 7 minutes to 4 minutes. [6]

3.1.3 Software

The Crazyflie 2.0 is an open project, that means that it exists a community that supports the project creating supported firmware, software and APIs written in Java, Ruby, C++, C# and Javascript.

Bitcraze provides a Crazyflie client used for controlling the Crazyflie, flashing firmware, setting parameters and logging data but is not the only way to control the platform.

The firmware and software is continuously being updated with various improvements and new features added. The platform supports wireless firmware updates via radio and Bluetooth LE, so when a new firmware is released it's a breeze to update it.

Supporting multiple radio protocols, the Crazyflie 2.0 can be used from a Bluetooth LE enabled mobile device or from a computer using the Crazyradio or Crazyradio PA.

To control the Crazyflie and realize the different experiments, it is used a package for ROS named *crazyflie_ros* and some more packages for ROS that are used to obtain and process the images of the camera.

The *crazyflie_ros* package supports Crazyflie 1.0 and Crazyflie 2.0 and uses the Crazyradio PA to receive and publish on-board sensors data. The used version of this package was a modified version provided on Github by the user "Denkrau".

That version of the package contains a PID control for the Crazyflie using Aruco markers and it was modified in each experiment in order to obtain the necessary information.

3.1.4 Control architecture

As it was explained in the last section, in the experiments, the Crazyflie 2.0 is controlled by the *crazyflie_ros* package by "Denkrau".

This package needs other packages to obtain all the necessary data in order to work properly. The structure of the control system is the next shown on the next figure.

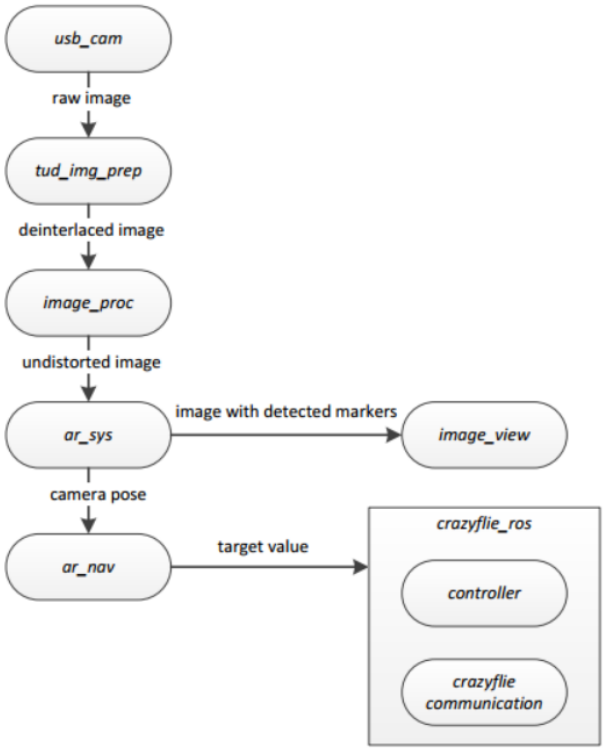

Figure 3.4: Scheme that shows all the nodes and topics of ROS used to run the *crazyflie-ros* package. Nodes are represented by elipses and topics by arrows. Extracted from [6].

As it appears in the last scheme, 7 different packages are used to control the Crazyflie 2.0. These packages are *usb_cam, tud_img_prep, image_proc, ar_sys, image_view, ar_nav* and finally the known *crazyflie_ros*.

To understand the whole way of working it is going to be explain what does every one of these packages.

Usb_cam

The usb_cam package interfaces with standard USB cameras and publish the images as a *sensor_msgs*.

Tud_image_prep

The *tud_image_prep* package is a set of tools for processing camera images with techniques that include deinterlace of analog images and filtering.

Image_proc

This package contains the *image_proc* node that removes camera distortion from the raw image stream, and if necessary will convert Bayer or YUV422 format image data to color.

Ar_sys

This package is used to estimate the 3D pose of an image using ArUco marker boards.

Image_view

The image_view package is a simple viewer for ROS image topics that also includes a specialized viewer for stereo and disparity images.

Ar_nav

Ar_nav is a ROS package that offers an autonomous flight including waypoint navigation with a quadcopter using ArUco markers.

Crazyflie_ros

This package provides different tools to work with the Crazyflie 1.0 and 2.0 nano quadrotor from Bitcraze. The package contains core drivers to use the Crazyflie with ROS, a URDF model of the quadrotor, a simple navigation to goal if it is known the external position and different demos ready to run for Crazyflie.

Once it is known the function of every package, the general process can be explained in a short resume.

The system obtains an image from the camera that is attached on the Crazyflie 2.0 and after that, the image is filtered and processed to improve the its quality and the distortion is removed.

Once the image is ready, it is detected an ArUco marker in this image and the 3D pose of the platform is estimated and shown on the computer.

After that, the necessary inputs are sent to the Crazyflie in order to drive the platform to the goal position.

To understand better how the *crazyflie_ros* package can control the position of the quadcopter the next figure contains a scheme of the processes realized.

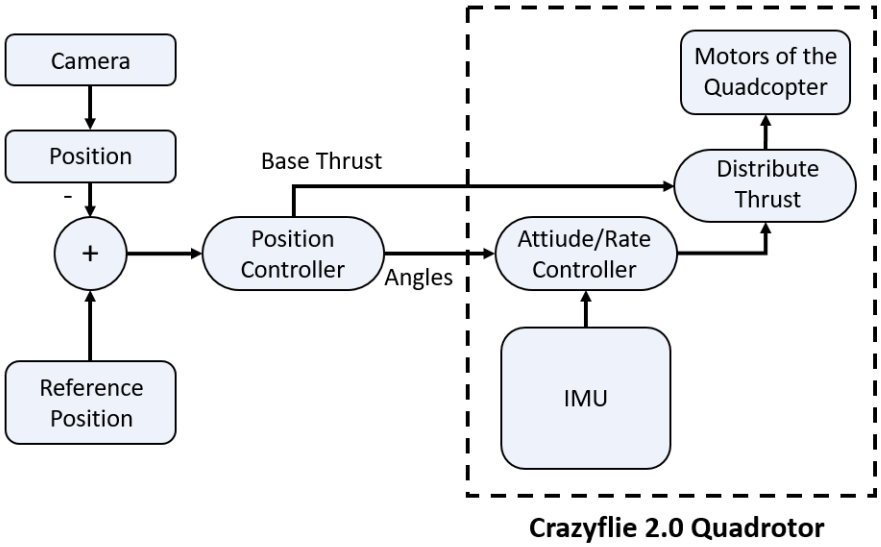

Figure 3.5: Scheme that shows how the *crazyflie_ros* package controlls the position of the quadcopter.

As one can see, the position obtained by the camera is compared with the reference positions and the error between them is going to be sent into the position controller.

Once the controller receives the error, it uses some PIDs to calculate the base thrust that has to send to achieve the required height and which angles are needed to change the position of the quadrotor into the desired one.

Finally, the thrust is distributed into the four rotors and that produces a movement of the quadrotor that makes the position change and the control loop starts again.

4 Modeling and Identification of Crazyflie2.0

In this chapter, it is going to be described the dynamic equations for a general quadcopter derived around the hover regime from the equations of motion.

4.1 Dynamics

The principle of operation of a quadrotor consists in the generation of a net force using the rotation from the 4 rotors. The quadcopter is characterized using four identical rotors. The motors of the quadcopter rotate in different ways, the motor 1 and 3 rotates in counter-clockwise direction and the motors 2 and 4 rotates in clockwise direction.

Changing the rotational speed of each rotor separately makes able a control of the rotational and translational dynamics of the quadcopter and there are 4 possible cases of actuation.

1. Hover: this case describes a state behavior of the quadcopter which maintains a constant altitude and the velocities on the 3 axes are zero. During this state, the four motors are rotating with the same angular speed.

2. Tilt of the quadcopter: to produce a translation to left or right and to forward and backward, it is necessary a tilt of the quadcopter.

   For example, to produce a linear movement in x-direction, the quadcopter must present an angle around the y-axis, also known as pitch. To produce that angle the motors have to spin at different speeds.

   To produce a pitch, the motors 1 and 2 have to spin faster than the motors 3 and 4 or vice versa, on the contrary, to produce a roll (rotation on the x-axis) the motors 1 and 4 rotates faster than the motors 2 and 3 or vice versa.

   Figure 4.1 shows the number of the motors and the different combinations of movements.

3. Rotation around z-axis: the rotation of each rotor produces thrust and torque in the center of this one. In the hover state, all the rotors spin with the same speed, that means the momentum produced in the center of mass is balanced and any movement is produced.

   On contrary, when the rotors 1 and 3 spin faster than the 2 and 4 or vice versa, the momentum in the center of mass is not balanced and produces a rotation around the z-axis also known as yaw.

4. Altitude variation: to control the altitude the quadrotor acts similar to hover state. In hover state, all the rotors spin at the same speed but the altitude is constant achieving an equilibrium.

   Increasing or decreasing this speed of the four rotors brakes the equilibrium of the quadrotor and produces a movement on z-axis, in other words, the quadcopter goes up or down depending if the speed of all rotors increases or decreases.

To understand better how reacts the quadrotor to the different rotations of the rotors, figure 4.1 shows a scheme of the different movements that the drone can realize depending on the rotor spinning situation.

As one can see looking at figure 4.1, on case A and B a pitch is produced. The pitch moves the quadcopter forward or backward and it is produced increasing the spinning of the rotors 1 and 2, to move forward, or the rotors 3 and 4, to move backward.

On case C and D, a roll is produced. The roll moves the quadcopter left or right and it is produced increasing the spinning of the rotors 2 and 3, to produce a movement to the right, or the rotors 1 and 4, to produce a movement to the left.

Cases G and H show how depending on how fast spin all the rotors at the same time, the altitude can increase or decrease.

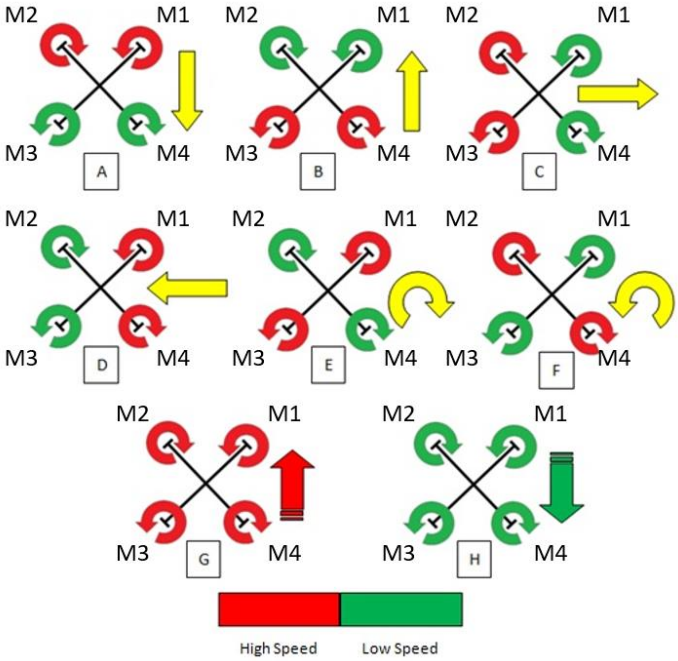

Figure 4.1: This image shows the different rotation directions of the rotors from the quadcopter and how it reacts to this rotations.

4.1.1 Preliminaries

Coordinate Frames

The dynamics of the quadrotor are expressed using the terms of body-frame. The origin of the coordinate frame is located at the center of mass and the velocities and orientation are expressed using that frame.

In that frame, the x direction points to the right, the y direction to forward and the z direction points up. In figure 4.2 it is shown a scheme of the coordinate frame that was described for the Crazyflie 2.0.

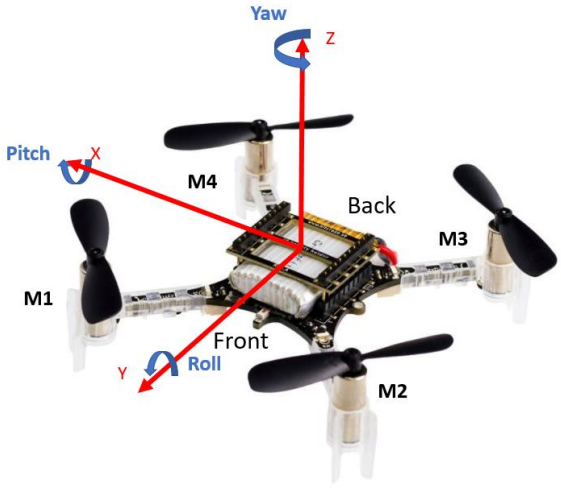

Figure 4.2: Image that shows the different axes and rotations of the Crazyflie 2.0 and the numeration of its rotors.

Notice that this coordinate frame showed is the one for the quadrotor but this frame is not the same as the camera coordinate frame.

The origin of the camera coordinate frame is aligned with the x and y position of the quadrotor frame but it is displaced 2 cm down over the z direction and does not share the same direction frame because in the camera frame, the z-axis points to the ground.

That body-frame explained before it is used to express the dynamics of the quadcopter but it does not provide a global position, to do that it is created a world frame in order to represent the position of the quadcopter in the space.

The world frame follows the same conventions than the body frame.

Both frames use the *right-hand rule* to represent the angles of rotation. These angles of rotation, that are also shown on figure 4.2, are the next:

1. Pitch: pitch angle is represented using the character θ and it is defined as a rotation around the y-axis.

2. Roll: roll angle is represented using the character ϕ and it is defined as a rotation around the x-axis.

3. Yaw: yaw angle is represented using the character ψ and it is defined as a rotation around the z-axis.

As one can see, the systems has six degrees of freedom formed by three translational degrees of freedom (x, y and z) and three rotational degrees of freedom (θ, ϕ and ψ).

Rotation in the reference frame

To express the transformation between the body-frame and the world-frame it is used a 3x3 rotation matrix and a 3x1 translation vector. The rotation matrix **R**(ϕ, θ, ψ) is made by three independent rotations and the translation vector $\mathbf{x}_0$ describes the offset between the origin of the world-frame and the body-frame in x, y and z direction.

$$\mathbf{R}(\phi,\theta,\psi) = \begin{pmatrix} \cos(\theta)\cos(\psi) & \cos(\theta)\sin(\psi) & -\sin(\psi) \\ -\cos(\phi)\sin(\psi)+\sin(\phi)\sin(\theta)\cos(\psi) & \cos(\phi)\cos(\psi)+\sin(\phi)\sin(\theta)\sin(\psi) & \sin(\phi)\cos(\theta) \\ \sin(\phi)\sin(\psi)+\cos(\phi)\sin(\theta)\cos(\psi) & -\sin(\phi)\cos(\psi)+\cos(\phi)\sin(\theta)\sin(\psi) & \cos(\phi)\cos(\theta) \end{pmatrix}$$

(4.1)

The rotation between both frames is given through **R**(ϕ, θ, ψ) = **R**(ϕ) **R**(θ) **R**(ψ). First, the world-frame has to be rotated around the x-axis, so it is made a ϕ rotation.

Secondly, it has to be rotated the world-frame around the y-axis, so it is made a θ rotation and finally it is made the same for z-axis making a ψ rotation. Notice that these rotations does not affect to the coordinate points.

As it was said before, the equation 4.1 represents three rotations. These individual rotations are called basic rotations or elemental rotations and are represented in equation 4.2 and the rotations follow the right-hand rule.

$$\mathbf{Rx}(\phi) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\phi) & -\sin(\phi) \\ 0 & \sin(\phi) & \cos(\phi) \end{bmatrix}$$

$$\mathbf{Ry}(\theta) = \begin{bmatrix} \cos(\theta) & 0 & \sin(\theta) \\ 0 & 1 & 0 \\ -\sin(\theta) & 0 & \cos(\theta) \end{bmatrix}$$

$$\mathbf{Rz}(\psi) = \begin{bmatrix} \cos(\psi) & -\sin(\psi) & 0 \\ \sin(\psi) & \cos(\psi) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

(4.2)

4.2 Modeling of the Crazyflie 2.0

In this section, the nonlinear dynamics of the quadrotor are represented in a simplified way for an operating regime around the hover point.

4.2.1 Quadrotor model near hover operation

When the quadrotor is operating on a state near the hoover operation, the aerodynamic effects could be ignored and threat the vehicle using the basic principles motion.

While the quadcopter is operating on hoover state, two forces are acting on the vehicle body: the gravity of the quadcopter, that follows the Second Law of Newton ***F** = mg,* that points to the floor and the combination of the thrust from the four rotors of the vehicle, $\sum \boldsymbol{T}$,that points up.

Notice that when the quadcopter has a tilt, the force of the gravity of the quadcopter stills pointing to the ground but the combination of all the thrust points to the perpendicular direction of the tilt angle as it is shown on figure 4.3.

To control the Crazyflie 2.0 it is needed to be able to control movement of orientation of this one. To change the speed in x and y direction it is needed to control the θ and ϕ angles because these produces an acceleration on x-axis and y-axis.

To control the altitude, it is needed to modify the height rate, and to modify the angle on z-axis is going to be controlled by the yaw rate. Knowing that, the inputs of the system can be represented as the equation 4.3.

$$\boldsymbol{u} = \begin{pmatrix} \phi ref \\ \theta ref \\ \Psi\, ref \\ \dot{z}\, ref \end{pmatrix} \tag{4.3}$$

The input **u** is a 1x4 vector that contains the input reference values for the roll angle, the pitch angle, the yaw rate and the height rate.

4.2.2 Attitude dynamics

To model the attitude dynamics, it is used a simplified second order model. This model represents a stable system and theoretically, the pitch and roll dynamics are identical one to the other one.

The attitude dynamics on x-axis and y-axis are represented as the SISO system equation showed on equations 4.4.

$$G_{\theta}(s) = \frac{\theta}{\theta ref} = \frac{n_0}{s^2 + sd_1 + d_0}$$

$$G_{\phi}(s) = \frac{\phi}{\phi ref} = \frac{n_0}{s^2 + sd_1 + d_0} \tag{4.4}$$

4.2.3 Lateral translation dynamics

The lateral translation is produced through a tilt of the quadcopter that creates a horizontal acceleration. The forces that are acting on a lateral translation are the gravity force and the thrust of the props.

Developing the Second Law of Newton, ***F*** = *mg,* for the both forces, it leads to the next 3 equations:

$$Fz = \sum T * \cos(\theta)\,;$$

$$Fx = \sum T * \sin(\theta)\,;$$

$$Fg = mg;$$

(4.5)

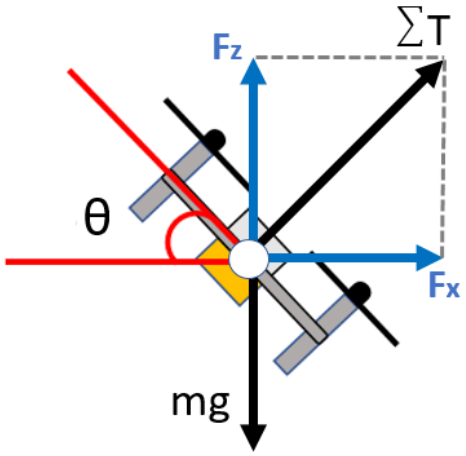

Figure 4.3: Image that shows the different forces acting into the quadrotor.

Looking the 4.5 equations, leads to understand that with a tilt angle of zero degrees, the force of the thrust is equal to the gravity force and maintains a constant height and no velocity on x-axis.

Increasing the tilt angle, the situation changes. The force of the thrust on z-axis is going to be equal to the gravity force in order to keep the height constant and the force of thrust on x-axis is going to produce an acceleration.

Knowing that, the 4.5 equations can be developed:

$$Fz = Fg = \sum T * \cos(\theta) = mg$$

$$\sum T * \cos(\theta) = mg \rightarrow \sum T = \frac{mg}{\cos(\theta)}$$

(4.6)

The develop of the equation 4.6 leads to know that the overall force of thrust can be expressed as the force of the gravity divided by the cosine of the tilt angle. Knowing that, another expression can be developed.

$$\begin{cases} Fx = \sum T * \sin(\theta) \\ \\ \sum T = \dfrac{mg}{\cos(\theta)} \end{cases} \rightarrow \quad Fx = mg * \frac{\sin(\theta)}{\cos(\theta)} \tag{4.7}$$

The equation 4.7 shows another way to express the force on x-axis of the quadrotor. Applying the Second Law of Newton and basic trigonometry leads to:

$$\begin{cases} Fx = mg * \dfrac{\sin(\theta)}{\cos(\theta)} \\ \\ \tan(\theta) = \dfrac{\sin(\theta)}{\cos(\theta)} \\ \\ Fx = a_x * m \end{cases} \rightarrow \quad a_x = g * \tan(\theta) \tag{4.8}$$

The quadrotor is operating near the hovering point and if it is assumed that to control the quadrotor small angles are used, it can be realized the approximation $\tan(\theta) \approx \theta$. This approximation leads to the next equation.

$$a_x = g * \theta \tag{4.9}$$

The same approach is used to model the same assumptions to y-axis using the ϕ angle. It is assumed that the air drags depending on the second order velocity is oblivious because of the low velocities around the hover operating regime and that leads to the transfer function for the translational dynamics:

$$G_{vel\,x}(\mathrm{s}) = \frac{v_x}{\theta} = \frac{g}{s} \qquad G_{vel\,y}(\mathrm{s}) = \frac{v_y}{\phi} = \frac{g}{s} \tag{4.10}$$

The transfer function showed on equation 4.10 relate the speed on *x* and *y* with their relative angles. A transfer function that relates the reference angle and the speed can be developed combining the equation 4.4 and the equation 4.10.

That leads to the next equation:

$$G_{vel}(\mathrm{s}) = \frac{v_x}{\theta ref} = \frac{v_y}{\phi ref} = \frac{n_0}{s^2 + sd_1 + d_0} * \frac{g}{s}$$

(4.11)

4.2.4 Vertical Translation dynamics and Yaw dynamics

To model the vertical translation dynamics, it is used a simplified first order model and it behave mostly linear. There are two transfer functions that can represent the vertical translation.

The first way uses the PWM thrust of the rotors as an input and the second way uses a reference height as a input. The equations of these models are represented on the next equations:

$$G_{height}(\mathrm{s}) = \frac{z}{\dot{z}_{ref}} = \frac{n_z}{sd_{z1} + d_{z0}} ; \qquad G_{height\ PWM}(\mathrm{s}) = \frac{z}{PWM} = offset\ PWM + \frac{n_{z\ PWM}}{sd_{z1\ PWM} + d_{z0\ PWM}}$$

(4.12)

In the PWM height transfer function, one has to notice that the transfer function has an offset. That offset represents the minimum PWM thrust that needs the quadrotor to initiate the elevation because until the quadrotor does not achieve that PWM value, there is not a linear behavior.

The yaw dynamic is also modeled as a first order system because it behaves mostly linear too. The input of that function is the reference yaw rate of the quadcopter and the output is the yaw angle. Knowing that, the yaw dynamic is modeled as follows:

$$G_{Yaw}(\mathrm{s}) = \frac{\psi}{\dot{\psi}_{ref}} = \frac{n_\psi}{sd_{\psi 1} + d_{\psi 0}} ;$$

(4.13)

4.2.5 State space model of the quadcopter

To represent the general model of the quadrotor it is used a state space model. The state space model is a mathematical model of a physical system that contains inputs, outputs and state variables related by first order differential equations.

The state space model of the quadrotor consists of 12 states contained in ***x*** vector and is represented by the equation 4.14.

$$\dot{x} = A * x + B * u; \qquad y = C * x;$$

(4.14)

In last equation, ***A*** represents the state matrix, ***B*** represents the input matrix and ***C*** represents the output matrix. The input and output vectors are represented by ***u*** and ***y***.

The content of the matrixes and vectors used to express equation 4.14 are shown on equation 4.15.

$$x = \begin{pmatrix} x \\ y \\ z \\ \phi \\ \theta \\ \psi \\ v_x \\ v_y \\ v_z \\ \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{pmatrix}; \quad u = \begin{pmatrix} \phi_{ref} \\ \theta_{ref} \\ \dot{\psi}_{ref} \\ \dot{z}_{ref} \end{pmatrix}; \quad y = \begin{pmatrix} z \\ \phi \\ \theta \\ \psi \\ v_x \\ v_y \end{pmatrix}; \quad C = \begin{pmatrix} 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{pmatrix};$$

$$A = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & g & 0 & \frac{-k_1}{m} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & g & 0 & 0 & 0 & \frac{-k_1}{m} & 0 & 0 & 0 & 0 \\ 0 & 0 & -d_{z0} & 0 & 0 & 0 & 0 & 0 & -d_{z1} & 0 & 0 & 0 \\ 0 & 0 & 0 & -d_0 & 0 & 0 & 0 & 0 & 0 & -d_1 & 0 & 0 \\ 0 & 0 & 0 & 0 & -d_0 & 0 & 0 & 0 & 0 & 0 & -d_1 & 0 \\ 0 & 0 & 0 & 0 & 0 & -d_{y0} & 0 & 0 & 0 & 0 & 0 & -d_{y1} \end{pmatrix}; \quad B = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & n_{z0} \\ n_0 & 0 & 0 & 0 \\ 0 & n_0 & 0 & 0 \\ 0 & 0 & n_{y0} & 0 \end{pmatrix};$$

(4.15)

The measured outputs of this model are: the tilt angles (ϕ, θ, ψ), the velocities in x-y plane (v_x, v_y) and the height (z).

5 Results and Experiments

In this chapter, all the experiments realized to obtain the different transfer functions of the quadcopter are explained and it shows the models obtained.

5.1 Attitude Dynamics Experiments

This section contains a description of the experiments realized to obtain the attitude dynamics model of the quadcopter and shows the results obtained from these experiments.

5.1.1 Pitch and Roll Experiments

To realize the pitch and roll experiment, the controller of the *crazyflie_ros* package was modified in order to behave as it is explained below.

In this experiment, the Crazyflie 2.0 starts to fly, search the ArUco marker and stabiles a position at 70 cm over the marker.

After 10 seconds, the quadcopter is hovering over the marker very stable and starts to set an angle of 7.5° in y axe for the pitch experiment, and 7.5° in x axes for the roll experiment.

In figure 5.1, it is shown a scheme of the experiment that reproduces the behavior of the quadrotor while it is realizing this one.

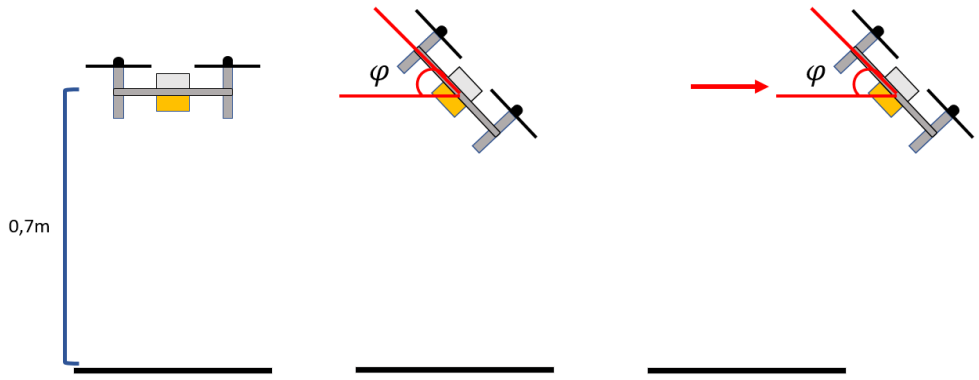

Figure 5.1: This image shows an scheme of the behavior of the quadcopter when it is realizing the pitch and roll experiments.

The input was sent using the topic *crazyflie/cmd_vel/linear.y* and *crazyflie/cmd_vel/linear.x* and the output was received using the topic crazyflie/imu and processing the data to obtain the function of the x and y angle evolution on time.

In the figure 5.2 it is shown the comparative between the command of the desired angle and the real one from the pitch experiment.

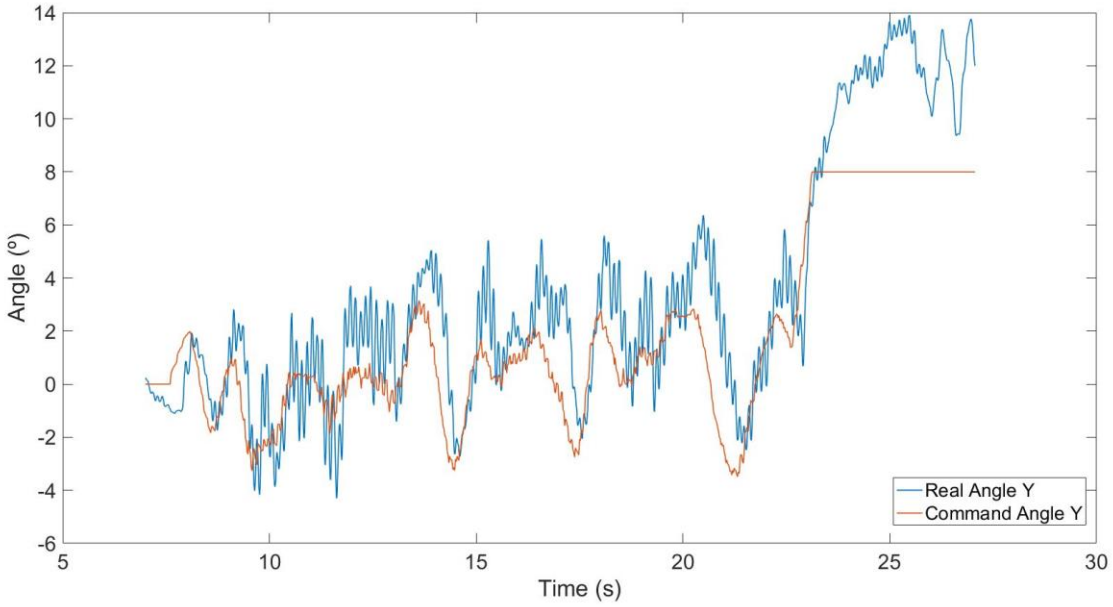

Figure 5.2: This figure shows the response of the angle in y-axis of the quadrotor for an certain input command sent. In blue appears the angle on y-axis detected by the imu of the quadrotor and in red appears the command input sent to the quadrotor.

Processing both signals by Matlab using the Matlab System Identification Tool, it can be obtained the transfer function of the pitch and the result is:

$$G_{pitch}(s) = G_{roll}(s) = \frac{123.2}{s^2 + 31.86s + 81.85} \tag{5.1}$$

This transfer function fits on a 69.33% with the real behavior that was seen in the experiment. To see if this transfer function presents a good behavior, a Simulink Matlab system was created and it is shown in the next figure.

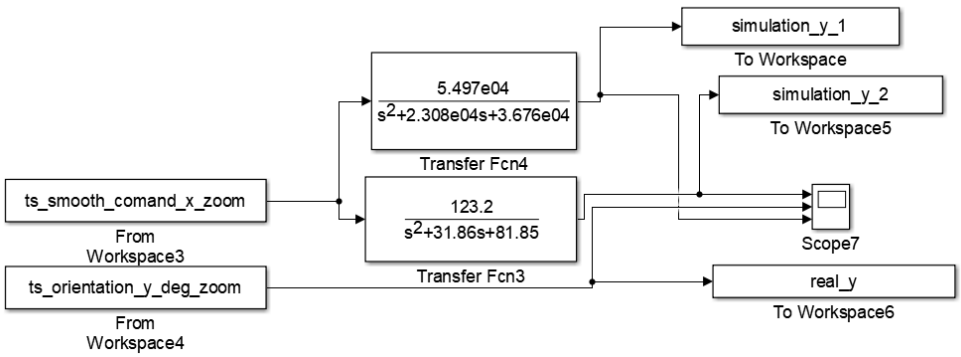

Figure 5.3: Simulink scheme used to compare the real angle measured by the imu and the simulation of that angle.

A comparison between the output obtained in the experiment and the output of a simulation using that transfer function is shown in the next figure.

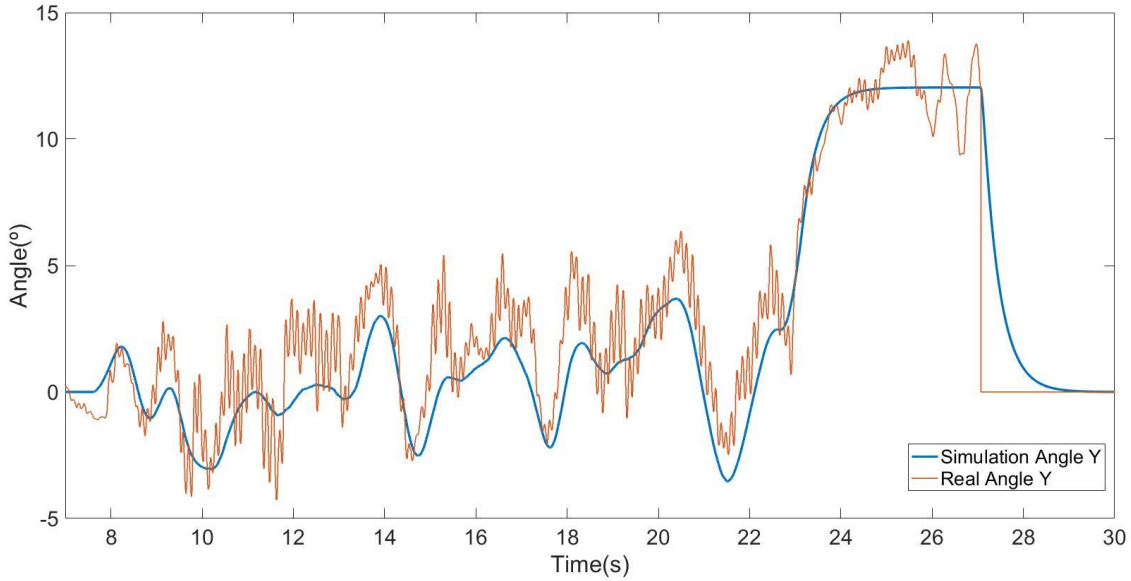

Figure 5.4: This graph shows a comparison between the angle on y-axis measured by the imu of the quadrotor and the simulation obtained of this angle. The real angle measured is shown in red and the angle obtained by simulation in blue.

As one can see in this figure, for irregular inputs the difference is bigger but for a constant or slower input the behavior fits better. Also notice that the output of the simulation presents a better response to the command input.

The same experiment was realized for the roll and the transfer function obtained fits on a 65.36% and was similar to the pitch transfer equation.

In order to simplify the model, the transfer function with better fit (the pitch equation) it is going to be used for both, pitch and roll, because in theory both behaviors are equals and the little difference could be an error of the fitting tool.

In the next figure appears the difference between the both transfer functions for the same input.

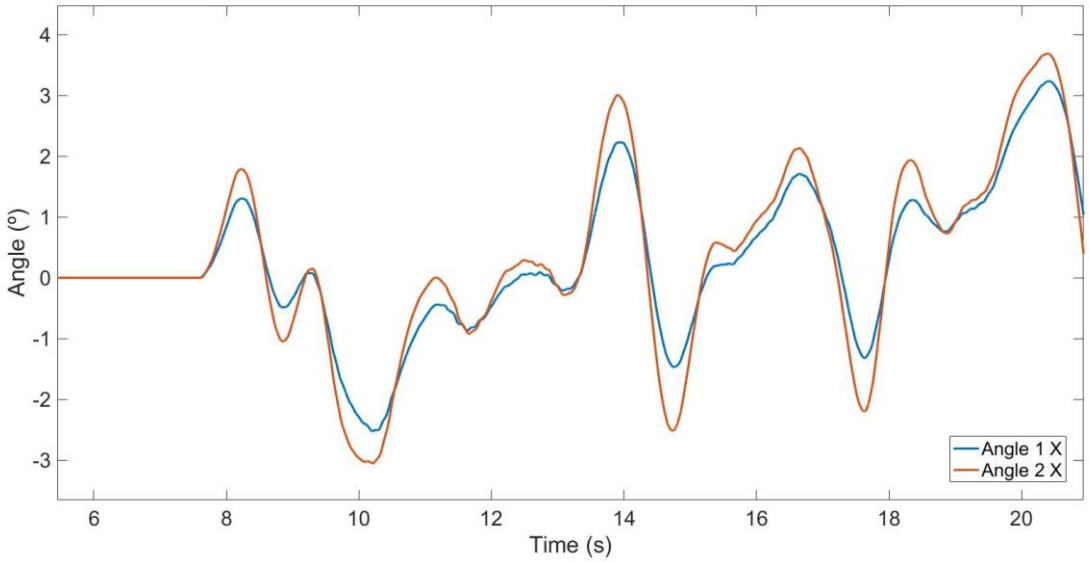

Figure 5.5: This graph shows a comprasion of the response of the x axis angle between two simulations for the same input using different transfer functions. In red it is shown the response using the transfer function obtained for the pitch and in blue it is shown the response using the transfer fuction obtained for the roll.

As one can see in the last figure, the error between both is very small, lower than 1°, so it is accepted to use the equation that fits better for both.

5.1.2 Yaw Experiment

To realize the Yaw experiment, the controller of the *crazyflie_ros* package was modified in order to behave as it is explained below.

In this experiment, the Crazyflie 2.0 starts to fly, search the ArUco marker and it stabiles a position at 70 cm over the marker. After 10 seconds, the quadcopter hoovers over the marker very stable and it starts to spin on the z axis without stopping with an angular speed of 36°/s as it is shown on the next picture.

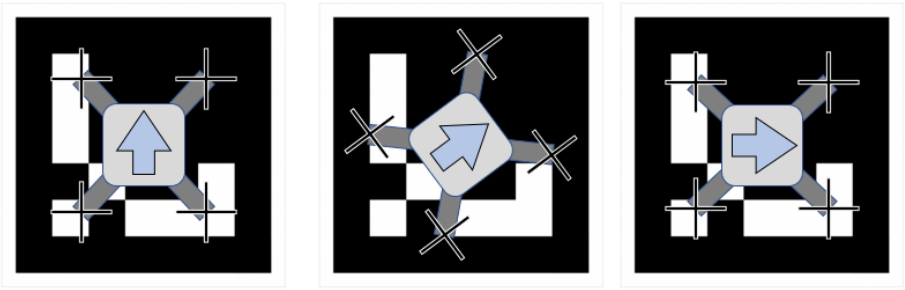

Figure 5.6: This image shows a represetation of the behavior of the quadrotor when it is realizing the yaw experiment.

The input was sent using the topic *crazyflie/cmd_vel/angular.z* and the output was received using the topic *crazyflie/imu* and the data is processed to obtain the function of the z angle evolution on time.

It was tried to obtain the transfer function using the Matlab function *tfest* but the fit percentage was too low (20%) so it was searched manually.

To realize that, a Matlab Simulink system was done using the same input of the experiment, the output of the experiment and adjusting a transfer function block to fit as better as it could be. The scheme of the Simulink system is shown in the next figure.

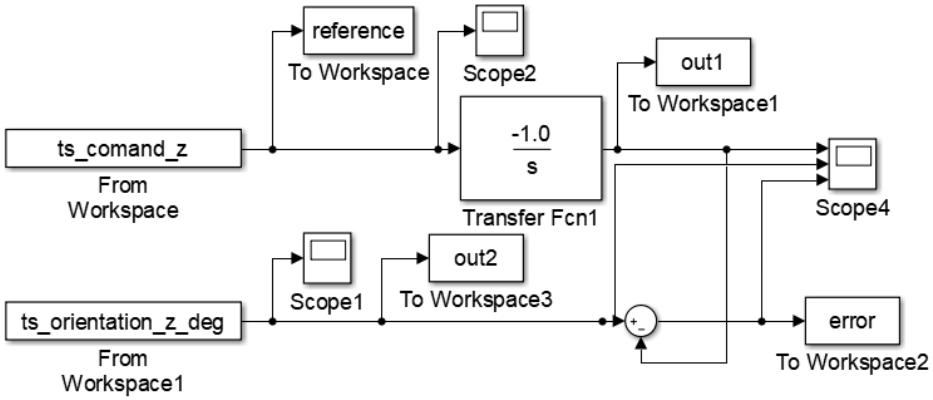

Figure 5.7: Simulink Scheme used to compare the real yaw measured by the imu and the simulation of that angle .

This Simulink system provides a simulated output for the yaw angle. To obtain a good transfer function, the transfer function block was adjusted manually until the best fit was found. The transfer function obtained for the yaw is the next one:

$$G_{Yaw}(s) = -\frac{1.0}{s} \tag{5.2}$$

As one can see, the equation seems to be perfect but if it is compared the output of the yaw simulation and the output of the real yaw (Figure 5.8), one can see that the error is very low so that equation can be accepted as the yaw transfer function.

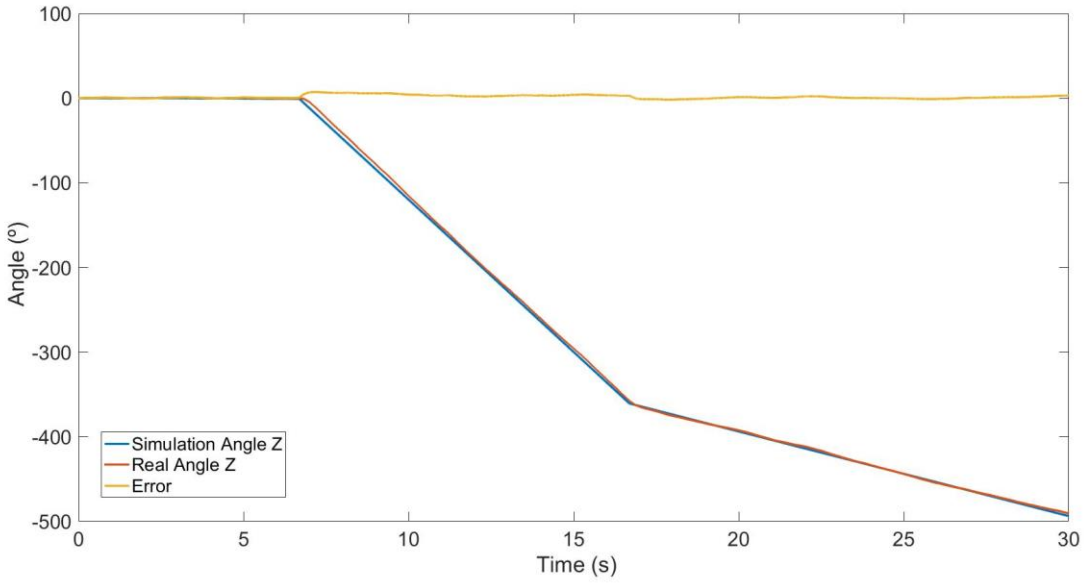

Figure 5.8: Graph that shows the comparison between the real yaw measured by the imu and the simulated yaw. The real angle measured is shown in red, the simulated angle in blue and the error between them in yellow.

5.1.3 Height Experiment

To realize the height experiment, the controller of the *crazyflie_ros* package was modified in order to behave as it is explained now.

In this experiment, the Crazyflie 2.0 starts to fly, search the ArUco marker and stabiles a position on 70 cm over the marker. After 10 seconds, the quadcopter is hovering over the marker very stable and starts to increase its height until hoover at 80cm over the marker as it is shown on the next picture.

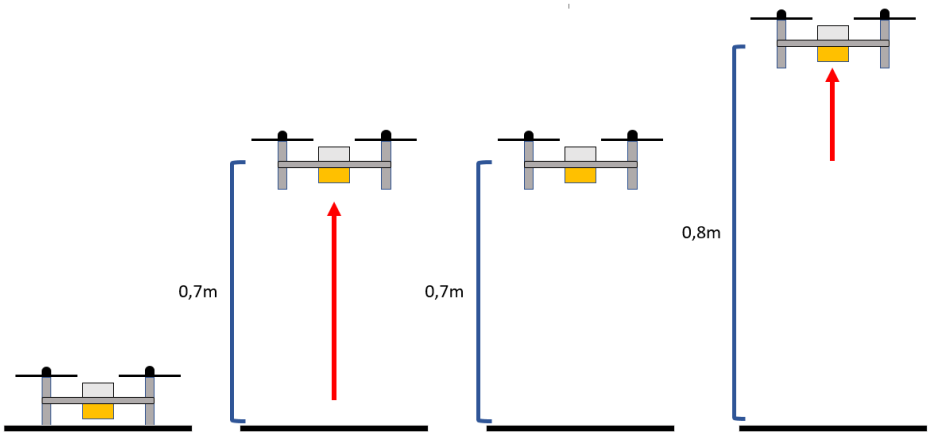

Figure 5.9: This image shows a scheme of the behavior of the quadcopter when it is realizing the height experiment.

The input was sent modifying the desired height inside the controller and the output was received using the topic ar_single_board/pose and processing the data to obtain the function of the z distance evolution on time.

In the next figure, it is shown the evolution of the desired height and the real height in the experiment.

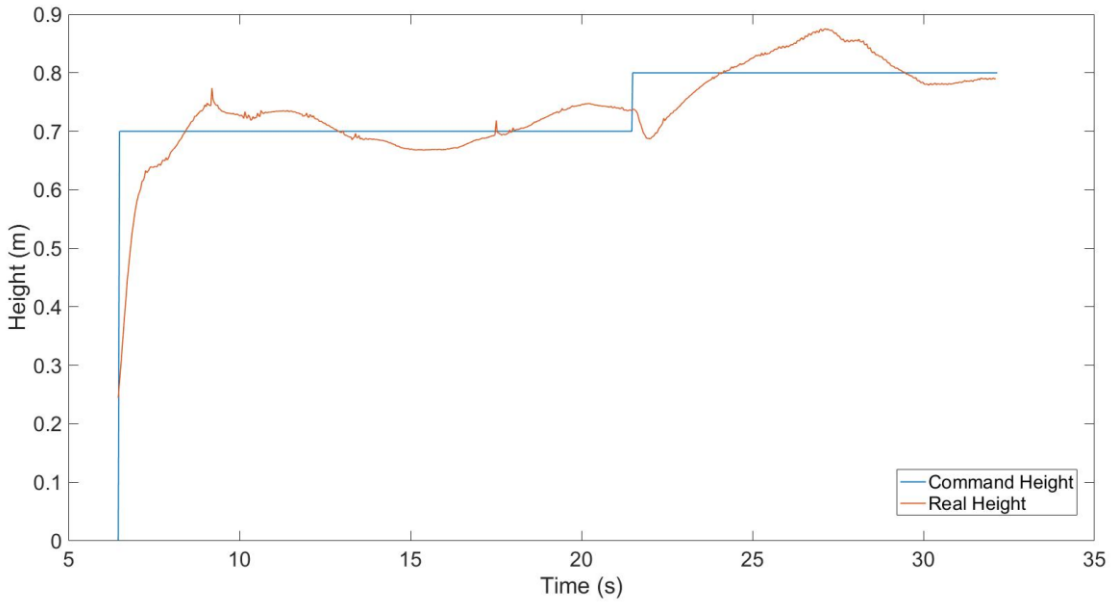

Figure 5.10: This graph shows the response of the height of the quadcopter for a certain height input command. In red appears the angle height detected by the quadrotor and in blue appears the command input sent to the quadrotor.

Processing both signals by Matlab using the Matlab System Identification Tool, it can be obtained the transfer function of the pitch and the result is:

$$G_{height}(s) = \frac{1.615}{s + 1.592} \tag{5.3}$$

This transfer function fits on a 77.76% with the real behavior that was seen in the experiment. To see if this transfer function presents a good behavior, a Simulink Matlab system was created and it is shown in the next figure.

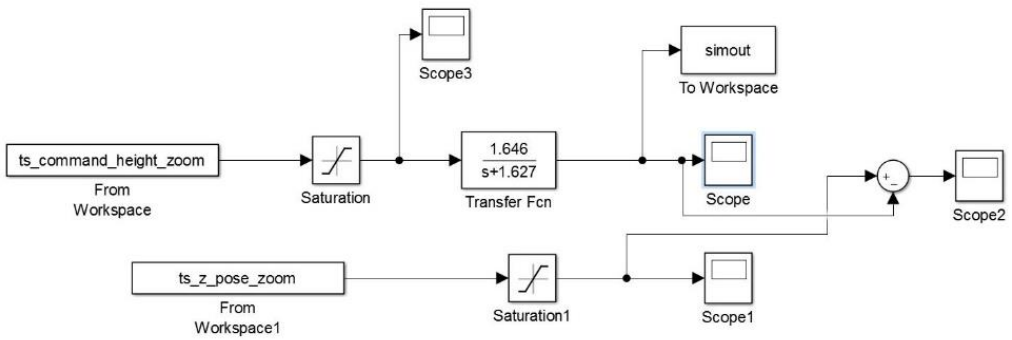

Figure 5.11: Simulink Scheme used to compare the height measured by the quadrotor and the simulation of that height.

This Simulink Matlab system uses the input command for the height used on the experiment and the real height measured by the camera on the experiment.

The transfer function used is the one found before and this system shows a comparison between the output obtained in the experiment, the output of a simulation using that transfer function and the error between them in the next figure.

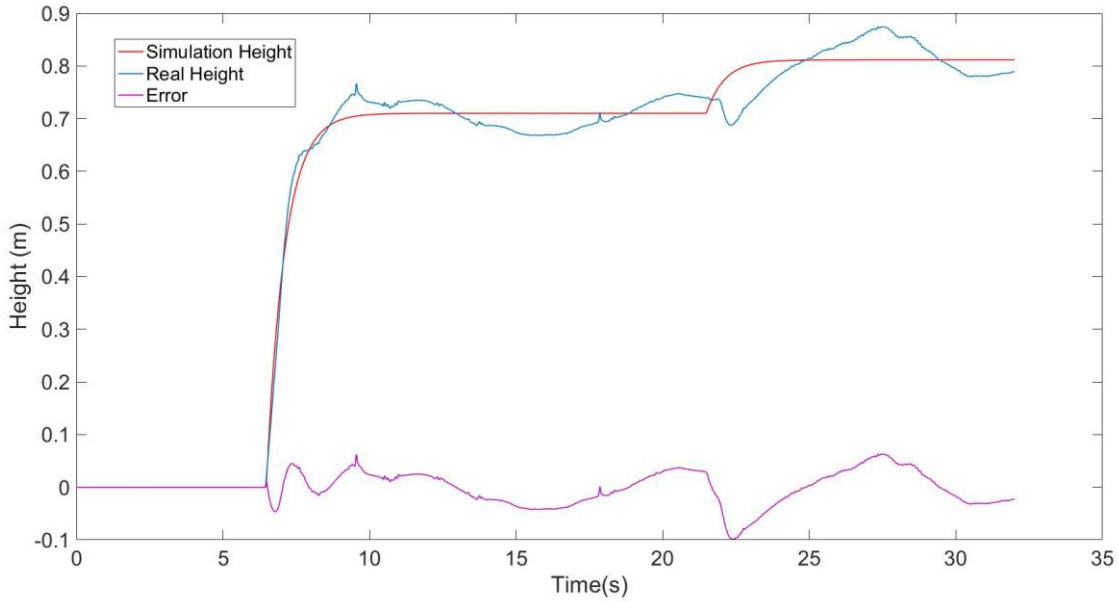

Figure 5.12: Graph that shows the comparison between the height measured by the quadrotor and the simulated height. The height measured is shown in blue, the simulated height in red and the error between them in purple.

As we can see the approximation of the transfer function is quite good and the error between them could be because the detection of the ArUco markers can present an error of 10cm and as it appears in the last figure, the biggest error is about 10 cm.

Another transfer that could be useful is the one that establish a relation between the PWM signal sent to the motors of the quadrotor and the height.

In order to find this transfer function, it has to be realized the same experiment realized before, so the same data can be used.

The input signal used is sent using the topic *crazyflie/cmd_vel/linear.z* and it is shown in the figure 5.13. To obtain the height it is going to be used the same topic used before, the *ar_single_board/pose/z.*

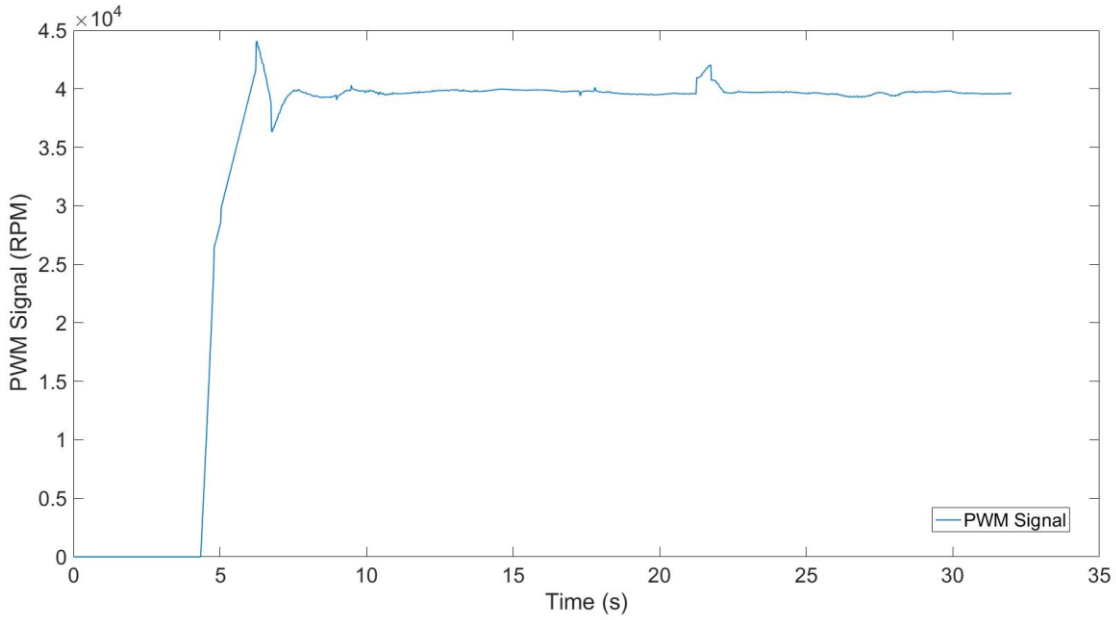

Figure 5.13: This graph shows the thrust signal PWM sent to the rotors to obtain a linear movement in z-axis.

When the quadcopter is trying to take off, there is a moment that the props are turning at a high revolution but the quadrotor doesn't take off.

That is because there is an offset PWM and only when the rotors are higher from that offset value, the vertical movement starts. In this quadcopter, it was made an approximation of this offset and the result was 35000 rpm.

Knowing that, to obtain the transfer function one has to subtract that offset to the function and apply the Matlab Identification tool and find a first order transfer function model.

Doing that, the next transfer function was found:

$$G_{height\ PWM}(s) = \frac{9.87 * 10^{-5}}{s + 0.6123} \tag{5.4}$$

To compare the transfer function and see if it has a good fit, the Simulink scheme shown on Figure 5.14 was used.

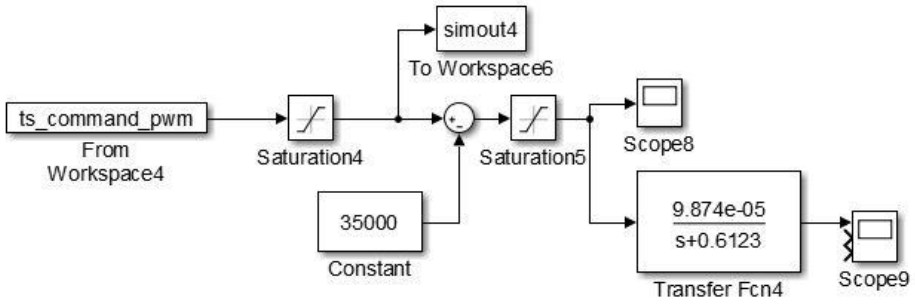

Figure 5.14: Simulink Scheme used to obtain the simulated height measured by the quadrotor using the PWM transfer function.

In the figure 5.15, it is shown the comparison between the real height and the simulated height using the PWM transfer function.

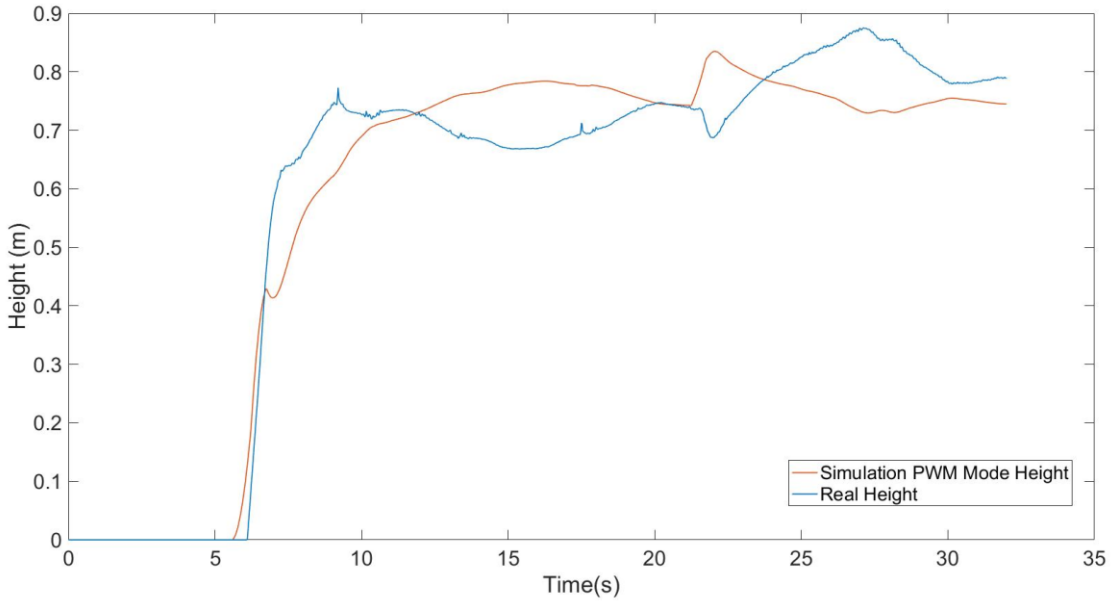

Figure 5.15: This graph shows a comparison between the height measured by the quadcopter and the height obtained by simulation using the PWM transfer function. The measured height is shown in blue and the simulated height is shown in red.

This transfer function presents an approximation of the height but is not as accurate as the other transfer functions, that is because there are too many errors at the time to estimate the transfer function.

The PWM signal, is a large signal, that means that errors and noise in the signal, become bigger after the transfer function.

Also, it is really hard to measure the offset of the taking off and that affects to the approximation and the ArUco markers could have an error of 10cm or even higher on the pose estimation.

5.1.4 Angle-speed Experiments

To realize the Angle-speed experiment, the controller of the *crazyflie_ros* package was modified in order to behave as it is explained now.

In this experiment, the Crazyflie 2.0 starts to fly, search the ArUco marker and stabiles a position at 70 cm over the marker.

After 10 seconds, the quadcopter is hovering over the marker very stable and it starts to set an angle of 7.5° in y-axis and it accelerate until the quadcopter lost the position of the ArUco marker.

In the Figure 5.16, it is shown a Simulink scheme to obtain the desired information.

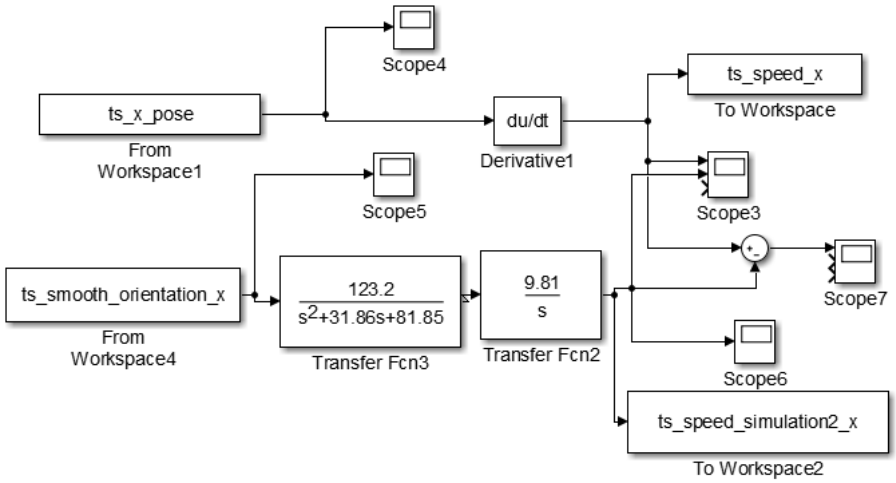

Figure 5.16: Simulink Scheme used to obtain the speed by derivating the measured pose of the quadrotor and compare it with the simulated speed.

To obtain this transfer function it is not used an identification method because as it was shown on chapter 4.2.3, to obtain the transfer function that relates the angle of the quadrotor with the speed of this one the only thing that one has to do is multiply the pitch and roll transfer function by $\frac{g}{s}$. Knowing that, the transfer function is expressed like the next equation:

$$G(s) = \frac{123.2}{s^2 + 31.86s + 81.85} * \frac{g}{s} = \frac{1208.592}{s^3 + 31.86s^2 + 81.85s} \quad \textbf{(5.5)}$$

To compare the simulation and the real speed it is going to be used the Simulink scheme used before. To obtain the real speed, it is going to be derivate the position captured by the ArUco marker among the time.

In Figure 5.17 appears the comparison between the simulation and the real speed.

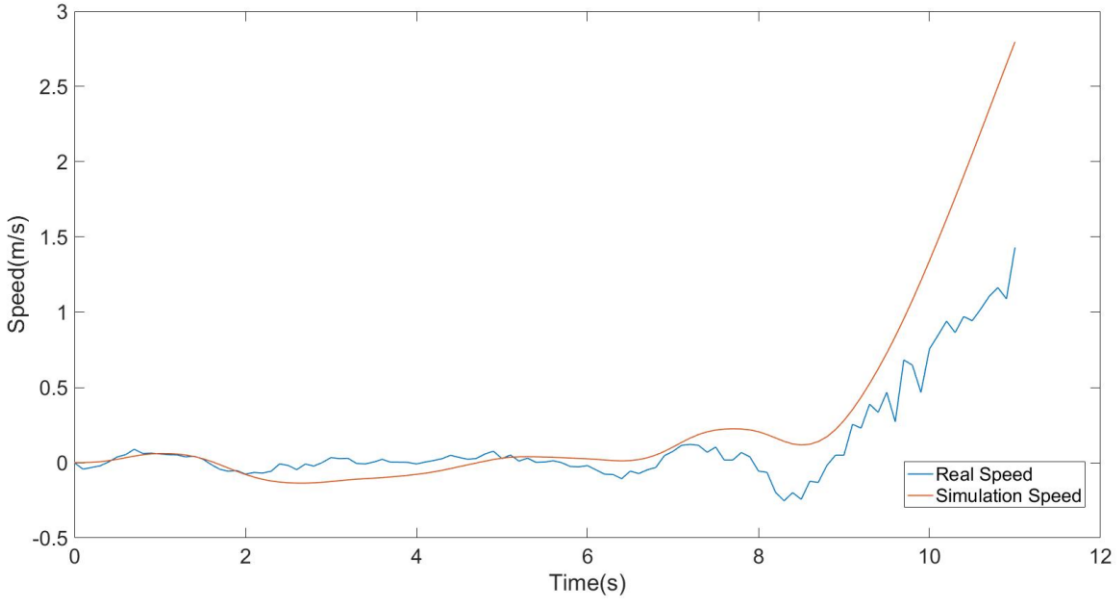

Figure 5.17: Graph that shows the comparison between the speed obtained by derivating the measured pose of the quadrotor and the simulated speed. The measured speed is shown in blue and the simulated speed in red.

As one can notice, the fit between the simulation and the real speed is not as good as wanted but it has an explanation. To obtain the real speed, it was used the position captured by the ArUco marker.

It is notice that the ArUco markers in a distance between the 0.6 meters and 1 meter could have an error of 10cm or even higher.

These errors can appear without one notice it and it is hard to know if the measurements are good or not and that means that the speed that in theory has to be the real of the quadcopter, it could present a big difference from the reality.

One way to improve that transfer function could be use an external way of location like a system of webcams at the roof of the room to obtain a better measurement of the quadcopter's position.

6 Data Processing

This section explains the different processes to transform and adapt the data obtained from the Crazyflie 2.0 platform to obtain the different transfer function models.

To realize the different experiments, it was created a Matlab code for each experiment and a simulation system using the tool Simulink.

In this chapter, there are some commented code lines of the different Matlab programs created and all codes are shown completely in the annexes of the thesis.

6.1 Recording Data

In order to save the data of the experiments to process it later, this information is recorded using the ROS command *rosbag record*.

The *rosbag record* command subscribes to a topic and creates a bag file where all the messages published on those topics are going to be save.

Recording the data into a bag file is very useful because there are a lot of tools for processing data that can work with bag files. One of this tools that can work with bag files, and it is the one used on this thesis, is Matlab.

The topics recorded in the bag files to realize the experiments are the next ones:

- ar_single_board/pose: this topic contains the information of the coordinates pose of the quadcopter on the three axes. This topic is necessary to realize the angle-speed experiment and the height experiment.

- crazyflie/cmd_vel: this topic contains the information of the inputs sent to the quadcopter. This topic is necessary to realize all the experiments because is the only input topic that it is used to control the trajectory of the quadcopter.

- crazyflie/imu: this topic contains the information that measured by the imu of the Crazyflie2.0. This topic is used in all the experiments except for the height experiment.

Notice that the bag files have reference time inside in order to synchronize all the topics on a posterior processing.

6.2 Extracting Data into Matlab

As it was mentioned in the section before, Matlab is used on this thesis to process the experiment data. The first step to process the data is extracting the information inside the bag files and convert it to another type of data better to handle it.

On the code created for each experiment, it is used some functions to extract the different data from the bag files and separate the different topics that are contained. An example code lines from one of the Matlab program are shown and explained below.

```
1. filepath = fullfile(fileparts(which('Matlab_code')),'\','Experiment.bag');
2. bag = rosbag(filepath);
3. bagselect = select(bag,'Topic','/crazyflie/cmd_vel');
4. ts = timeseries(bagselect);
```

Code 6.1: This code shows an example of how to import a bag file on Matlab and extrac a specific topic.

The code 6.1 shows a Matlab program abstract that, executed, imports a bag file saved in a folder of the pc and it extracts a desired topic into a *timeseries* variable.

On line one of the file *Matlab_code*, the function *fullfile* looks for the *Experiment.bag* file and saves the information on the variable *filepath*.

On second line, the function *rosbag* opens and parses bag file selected and saves the file into a BagSelection. After that, using the function select, a topic contained into the bag file, in this case the topic */crazyflie/cmd_vel*, is extracted and save it into an timeseries variable in order to be easily to handle.

6.3 Transforming Data

Once the data is extracted, it has to be transformed because that data is raw. The basic transformations realized are going to be shown below.

6.3.1 Unit Conversion

One of the basic data transformations realized by Matlab has the objective of realize a conversion of units. The different unit conversions used in the Matlab code are going to be explained below.

Once the data of the *imu* topic is extracted, one can obtain the angular speed in radians for the three axes. To be able to obtain the different transfer functions, the three angles on axes must be obtained.

To do that some Matlab functions have to be applied on the *imu* topic data to convert that angular speed in radians into angle position of the quadcopter in degree.

The first function used to do that is *cumptrapz.* As it is known, integrating the angular speed, the angle can be obtained and this is what that function does.

This function computes an approximation of the cumulative integral of the angular speed using the trapezoidal method with unit spacing. The data obtained from the imu has not a unit spacing and it has to be divided by a conversion number.

Once the angles in radians are obtained, it is needed to transform that angles in radians to degree. To do that the function *rad2deg* is used. That function converts automatically from radians to degree. The code lines used are shown below:

```
1. orientation_x = (cumtrapz(x_ang))/100;
2. orientation_x_deg = rad2deg(orientation_x);
```

Code 6.2: This code shows an example of how to convert an angular speed in radians into an angle in degree.

The code 6.2 shows an example of the functions that were explained before. On line one, the function *cumtrapz* is used to integrate an angular speed saved in the variable *x_ang* and saves the result, that is going to be an angle in radians, in the variable *orientation_x*.

On second line, it is used the function *rad2deg* to transform the angle in radians obtained before into degree.

6.3.2 Type Conversion

Sometimes, there are some Matlab functions that are not compatible with certain types of data and to solve that, the data type has to be changed. One of the most common case is related to the *timeseries* type.

The timeseries is a data vector sampled over time that often is used to plot graphics and it is not compatible with many Matlab functions.

To solve that problem, the timeseries has to be separated into the data part and the time part and after that, the pertinent functions are going to be applied into that divisions. Once the operations are done, the timeseries can be joined again.

6.3.3 Sample Conversion

In the experiments realized, there were used more than one topic, that means that not all the data comes from the same topic and that could create a problem.

Some topics have not the same rate frequency of communication, that means when the topics are recorded not all have the same number of samples.

There are some different options to find a transfer function using Matlab but all of them have the same requisite, and this requisite is that the input and output data used to find that transfer function, it must have the same number of samples.

To solve that problem, there are two different solutions. The first solution is to create a Matlab code that modifies the data with less sample by filling it using an interpolated data. That code was created and it is shown in Annex 2.

The second option is to use the Matlab tool Simulink and import all the data to create new data with a desired sample ratio.

This is the best option in terms of quality but that makes that you cannot run all the code at one and you should execute the code by parts.

6.3.4 Obtaining Transfer Function

Once all the data is processed, it is time to obtain the different transfer functions. Matlab provides two ways to obtain these transfers functions.

First way, and the simplest one, is to use the Matlab tool called "*System Identification*". That tool can be found on the apps menu of Matlab. To use that tool, it is needed the input and output of the system that one wants to identify.

There are some requirements to use that tool. The first requirement is that the input and output of the system that one wants to identify have to be a in a time domain variable.

The second requirement is that the input and the output must have the same number of samples and they have to be in a constant time rate.

Finally, the tool also need to know the number of poles and zeros of the function model that one wants to find.

One of the best ways to prepare the data to use that tool is using Simulink because it is easy to obtain different data with the same ratio and number of samples.

That tool provides a percentage of fit between the output of the system measured and the output of the system using the simulated transfer function found by the tool.

Other way to obtain the transfer function is using the Matlab function *tfest.* To use that function, one has to implement the next structure in the code:

```
1. time_sample = 0.02;
2. np = 1;
3. nz = 0;
4. data = iddata (output, input, time_sample);
5. sys = tfest (data, np, nz);
```

Code 6.3: This code shows an example of how to obtain a estimated transfer function in Matlab.

On code 6.3, *data* is an *iddata* type variable that contains the input and the output of the system with the same number of samples and the same time ratio, *np* is the number of poles that has the transfer function that one wants to identify and *nz* is the number of zeros of the transfer function that one wants to identify.

Once that function is executed, Matlab shows a percentage of fit between the output of the system measured and the output of the system using the simulated transfer function found by the tool.

To find the different transfer functions of this thesis, both methods were used and it was arrived at the conclusion that in the majority of the cases, the percentage of fit of the two methods were equal or very similar.

Looking that results one can think that the transfer functions obtained could have the same behavior but if one realizes the simulations with Simulink using the transfer functions obtained from each method, it can be notice that, at the majority of the cases, exist a discrepancy between both responses.

To see one of this example and understand the magnitude of this problem, the figure 6.1 is shown:

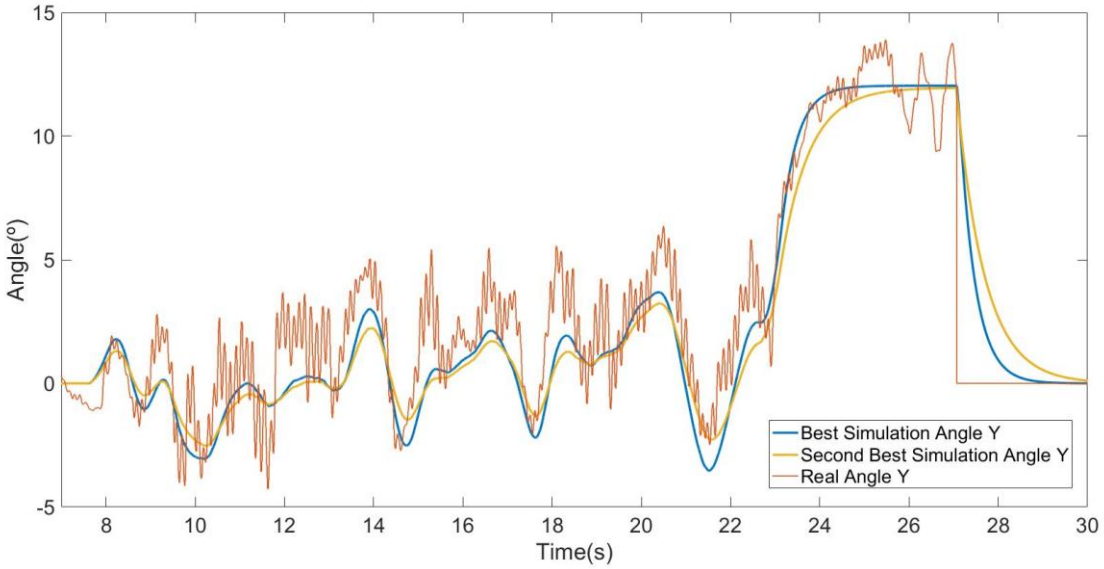

Figure 6.1: This image shows a comparison between the angle in y-axis measured by the imu and the angles in y-axis simulated using two different methods. The measured angle is shown in red, the different simulations appear in blue and orange.

In that case, both transfer function presented the same percentage fit, but at one can see easily that one transfer function fits much better to the reality than the other one.

After observing that, it was arrived at the conclusion that the best way to obtain the different transfer functions was using the "*System Identification*" tool.

7 Improvement of the Flight

In this chapter, it is going to be explained the process done to improve the flight of the quadcopter.

That process consists on the creation of a simulation platform, the calibration of the different PID of the quadcopter and finally the evaluation of the improvements on the performance.

7.1 Overall Simulation Matlab System

To control better the quadrotor, a Simulation platform was created. This platform is made by Matlab and simulates the behavior among the time of the quadcopter's trajectory of in 3D.

That platform can be used to adapt or find some PID values to make the flight of the quadcopter smoother, quicker or behaves as the user wants and also it can show the reaction of the quadcopter to a specific command inputs.

The platform is made by 2 parts: the Quadrotor Simulation part and the 3D Trajectory Animation.

7.1.1 Quadrotor Simulation

The Quadrotor Simulation is a system created by Simulink that uses the transfer functions of the model to simulate the response of the quadrotor for a certain input.

That system has two blocks and each one realizes a different function. The first block uses command inputs, that can be obtained from an experiment and recorded into a bag file, to obtain a simulation of the speed, position, orientation and angle of the quadcopter.

That block could be used to obtain a first idea of how the quadrotor reacts to certain inputs and it can show an approximate idea of the trajectory that the quadrotor realizes for these inputs.

Using the outputs of this block, the 3D Trajectory Animation can be used to show the response of the quadcopter in a 3D plot. The Simulink diagram is shown in the next figure:

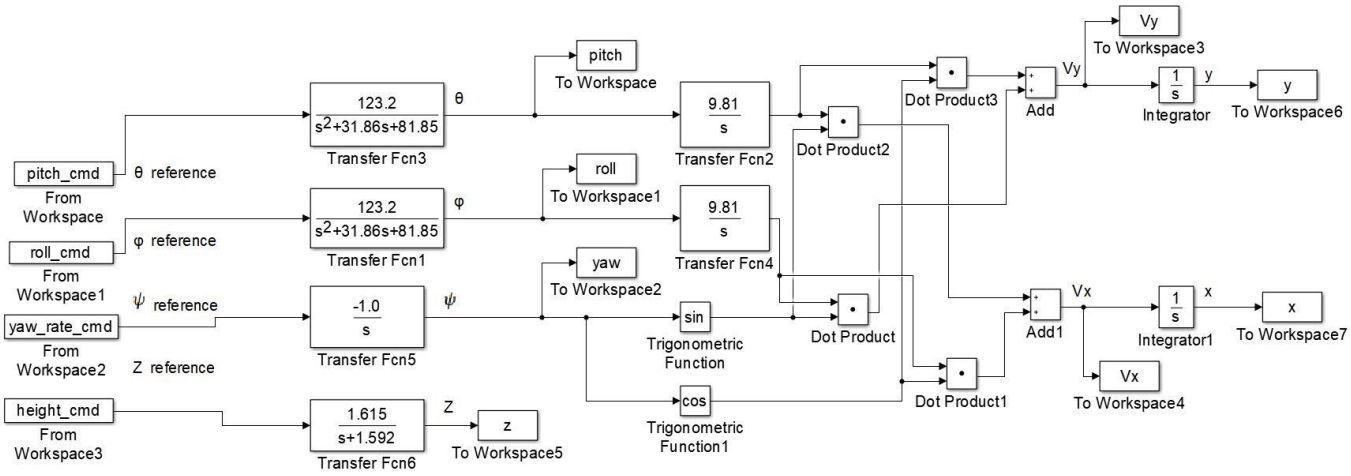

Figure 7.1: Simulink scheme used to obtain the outputs evolution of the quadrotor for a certain inputs.

As one can see in the last figure, the inputs of the system are: the pitch command, the roll command, the yaw rate command and the height command.

On the other side, the outputs of the system are: the pitch, roll and yaw, the height, the speed on x-axis and y-axis and the position on x-axis and y-axis.

The second block tries to simulate the behavior of the quadcopter when this one is controlled by the ROS package *crazyflie_ros* created by *Denkraum*.

When the quadcopter is running that package, it makes a comparison between the actual position and the goal position of the vehicle and uses some PID to correct that error and drive it into the correct place.

This block has different inputs as the previous one, instead of using the command inputs, it uses reference positions.

With that reference positions, it is created a close loop that uses a PID to transform the error between the reference and goal positions into commands for the different variables that are necessary to control the quadcopter.

The Simulink diagram of that block is shown in the next figure:

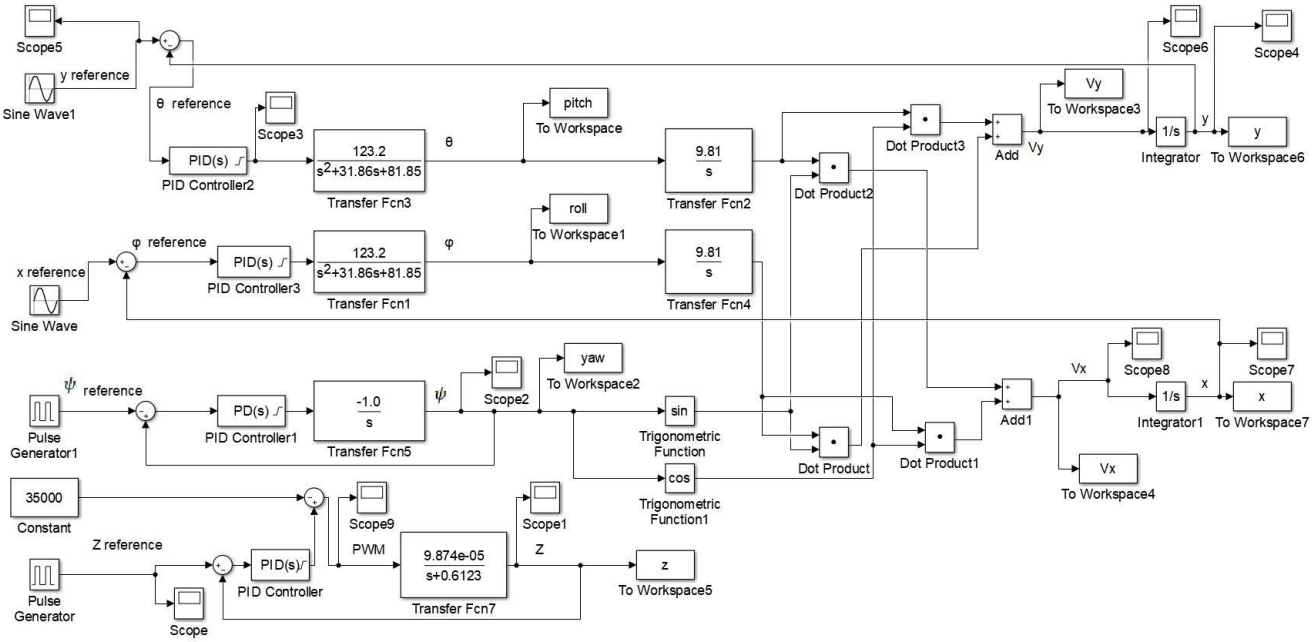

Figure 7.2: Simulink scheme that represents the control system of the quadrotor with its PID controllers and shows the evolution of the outputs for reference values .

As one can see in the last figure, the inputs of the system are: the x and y position reference, the yaw reference and the height reference.

On the other side, the outputs of the system are: the pitch, roll and yaw, the height, the speed on x-axis and y-axis and the position on x-axis and y-axis.

This block is very useful to take a preview of the flight of the quadcopter using different PID values or adjust PID values to change the behavior of the vehicle.

7.1.2 3D Trajectory Animation

In order to represent the flight of the quadrotor, it was created a Matlab code to show the trajectory of the vehicle in a dynamic way.

The 3D trajectory animation uses the 3D pose obtained in the Simulink simulations and creates a 3D animation of the trajectory of the quadcopter among the time, that means that one can see an estimation of how could be the flight of the quadcopter and also it could plot one trajectory recorded to see a comparison.

The representation of that trajectory is dynamic, that means that time is also represented on the 3D plot using the drawing speed of the graph. To understand it better, Figure 5.20 is shown:

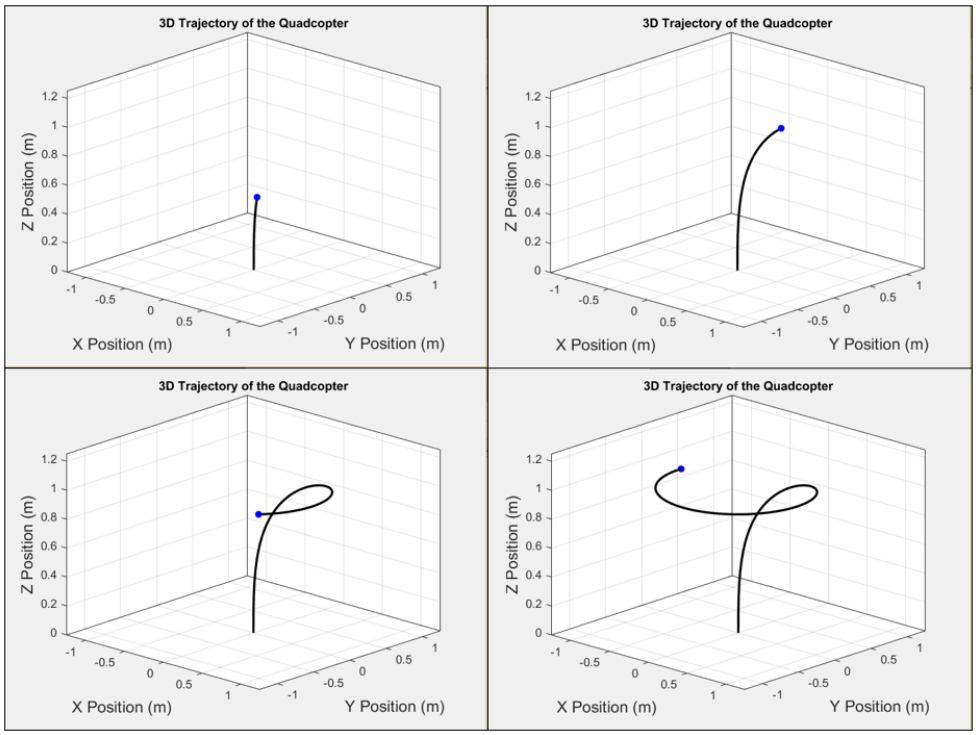

Figure 7.3: Image that shows the evolution of the simulated trajectory on the 3D trajectory animation.

Figure 5.20 shows the simulated trajectory of the quadcopter captured at different moments. The blue marker represents the position of the quadcopter and the speed of plotting is synchronized with the speed of the quadrotor.

That means if the marker goes from one point to another in a certain time, that time is going to be the same as the flight duration of the quadcopter.

7.2 PID Calibration

As it was mentioned on Chapter 3.1.3, in the experiments it was used a ROS package to control the Crazyflie and stabilize it over an ArUco Marker using a PID controller.

When the quadcopter was tested, it was noticed that the PID values used were not the best for this application because the stabilization was too fast and that produced overshoots.

Once the transfer functions were found and the Overall Simulation Matlab System was created, a new configuration for the PID can be found.

To do that, it was used a Simulink tool called “PID Tuner”. This tool analyzes the plant function and shows different responses depending of the transient behavior and the response time.

To find a good configuration, it was taken the parameters that makes the faster response of the system without having an overshoot and a robust transient behavior.

This procedure was realized for all the PID of the system and the parameters of each one is shown in the next table:

	Kp	Ki	Kd	Upper Limit	Lower Limit
Yaw PID	20.00	0.00	5.00	200	-200
Pitch PID	12.00	0.06	10.00	10	-10
Roll PID	-12.00	-0.06	-10.00	10	-10
Height PID	-10781.90	-5196.65	-638.14	60000	0

Table 7.1: This table shows the new parameters found for the different PID of the control system.

7.3 Evaluation of the Improvements

Once the new PID values are found, these are going to be tested realizing different test flights and the results are going to be compared with the old values to see the magnitude of the improvement.

One of the problems that were notified before starting the thesis was that the controller of the quadrotor worked good hoovering over the ArUco marker but it had a bad response when the vehicle had to go from a far position to the ArUco marker.

The old values of the PID that control the positioning of the quadrotor presented a big overshoot at the time to stabilize it over the marker.

To see if the new values of the PID present a better behavior, one test flight was done where it was compared the behavior of the old PID values versus the new ones.

On this test flight, the quadrotor is going to be located 80 cm away from the marker in the x-axis and after taking off, the quadcopter is going to fly until it stabilizes over the ArUco marker with a height of 70 cm.

The objective of this test is compare the positioning overshoot between the old and the new values of the PID. In figure 7.4 appears the trajectories done by the quadrotor in the test using both PID values.

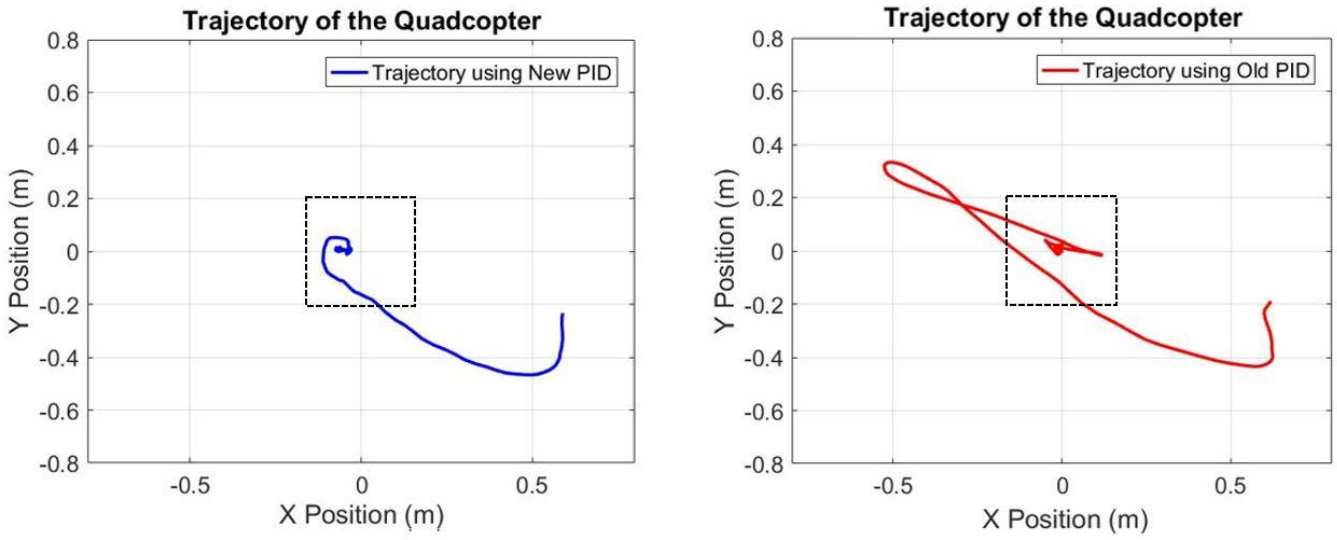

Figure 7.4: These images show the comparison between the trajectories of the quadrotor on x-y plane of the new PID values versus the old PID values**.**

Last figure represents the trajectories of the flights, the blue graph represents the trajectory using the new PID values and the red graph represents the trajectory using the old PID values.

The ArUco marker is located in the center of the axes and it is represented using a black rectangle. If one takes a look into the old PID values trajectory, it presents a huge overshoot of almost 50 cm in the trajectory and in some situations that overshoot could be so big that the camera could lost the marker of its field of vision.

On contrary, the new PID values trajectory presents a smooth behave and a small overshoot. That overshoot is so small that in any moment the quadrotor leaves the ArUco marker zone once it is over.

On figure 7.4, it can be compared which values of the PID produce less overshoot and a better trajectory but it cannot be evaluated the time response.

To evaluate the time response, it is going to be compared the evolution of the x and y position among time from the cases exposed on figure 7.4.

In order to appraise the time response, there are going to be compared the times that each case requires to stabilize the position of the quadcopter into the marker area without fluctuations.

To realize this comparison the next figures are going to be shown:

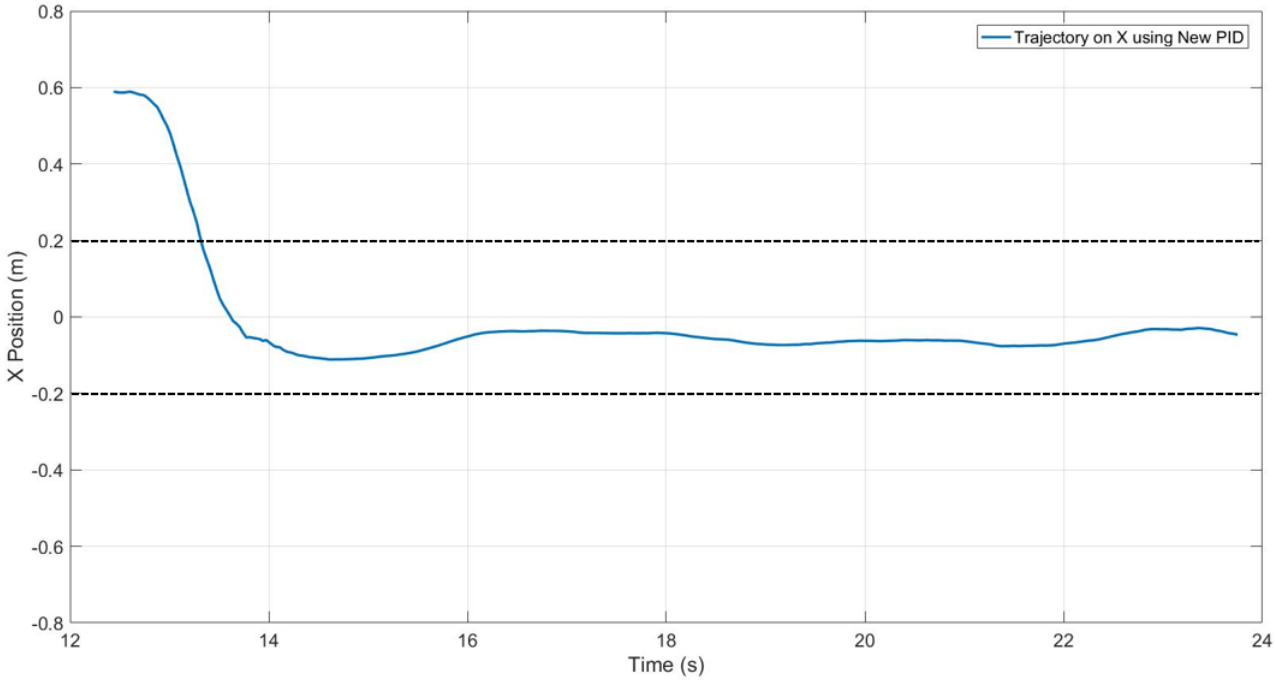

Figure 7.5: This image shows the x position evolution among time while the quadcopter is realizing the trajectory showed on figure 7.4 using the new PID values.

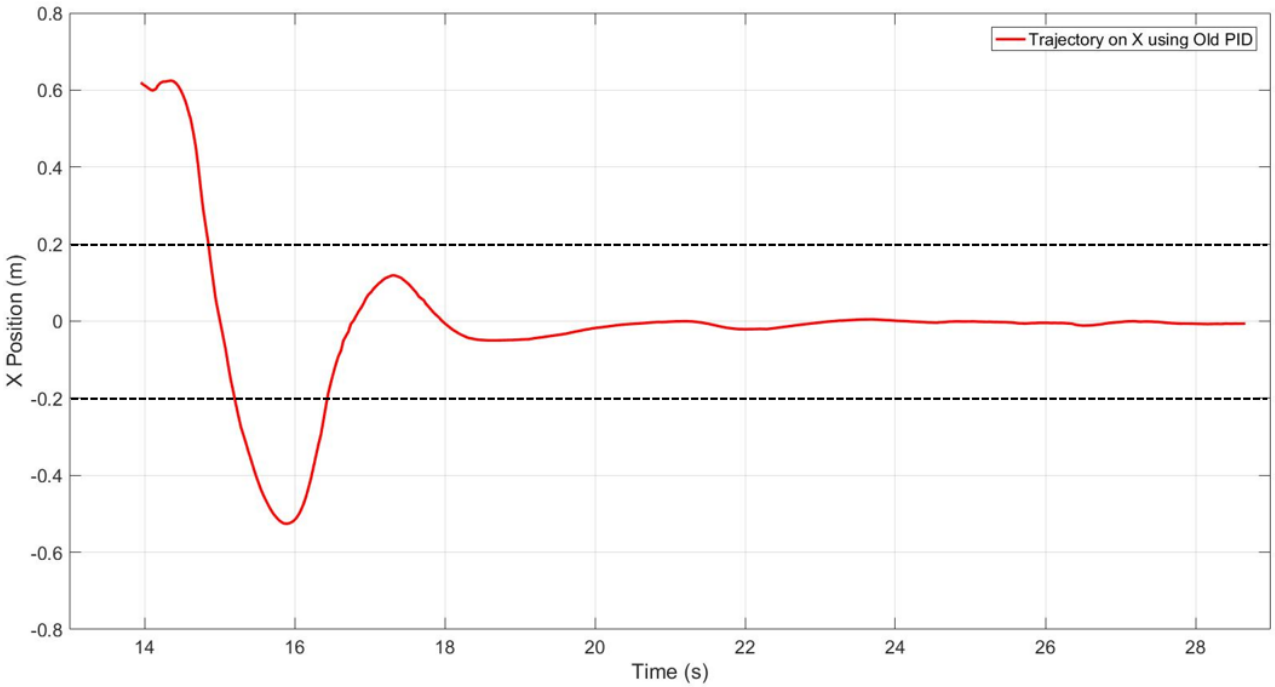

Figure 7.6: This image shows the x position evolution among time while the quadcopter is realizing the trajectory showed on figure 7.4 using the old PID values.

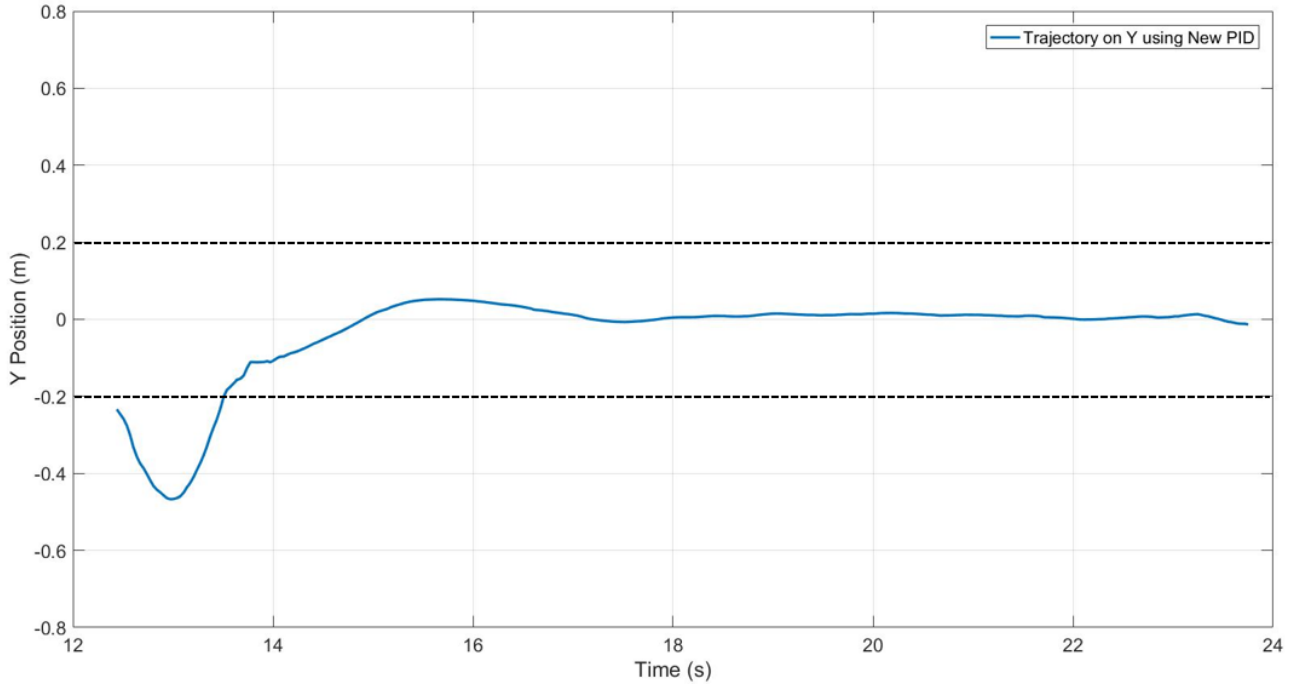

Figure 7.7: This image shows the y position evolution among time while the quadcopter is realizing the trajectory showed on figure 7.4 using the new PID values.

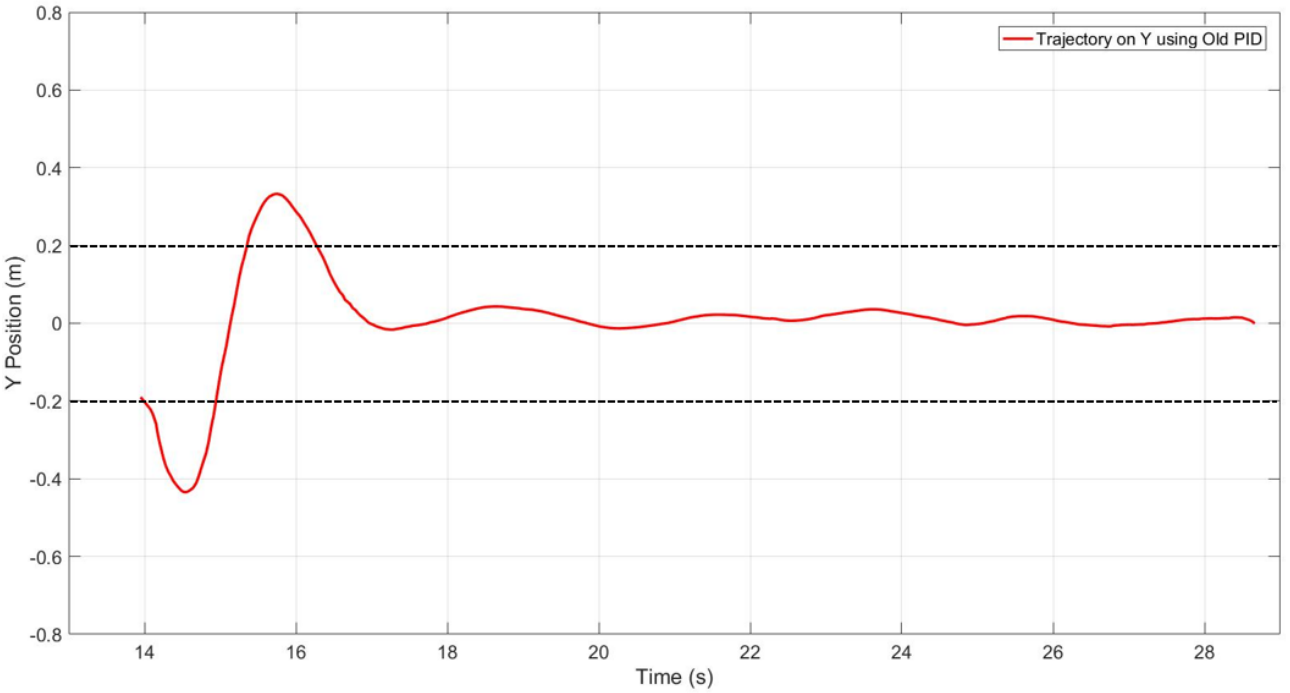

Figure 7.8: This image shows the y position evolution among time while the quadcopter is realizing the trajectory showed on figure 7.4 using the old PID values.

Taking a look on figure 7.5 and 7.6, the time response for the stabilization between both cases can be compared.

Figure 7.5, that it represents the case of the new PID values, presents a stabilization time of approximately 2 seconds from the detection of the ArUco marker until the stabilization over it.

Figure 7.6, that presents the case of the old PID values, presents a stabilization time of approximately 4.5 seconds from the detection of the ArUco marker until the stabilization over it.

That means the new PID values make behave the quadcopter with a better time response.

Looking figures 7.7 and 7.8, it is obtained a similar comparison like the one commented before. The distance lower than the last case but it stays still the same, the new PID values make behave the quadcopter with a better time response.

Another aspect that was wanted to improve of the quadrotor control was the height stability. With the old values of the PID, the height control takes too much time to stabilize and present some fluctuations before the stabilization.

To see if with the new values of the PID present a better behavior, a test flight was done.

On this test flight, the quadrotor is going to be located in the middle of the ArUco marker on the ground and is going to take off in order to achieve a height of 70 cm over the marker.

The objective of this test is compare the stability of height control between the old and the new values of the PID.

In figure 7.5 appears the height trajectories done by the quadrotor in the test using both PID values.

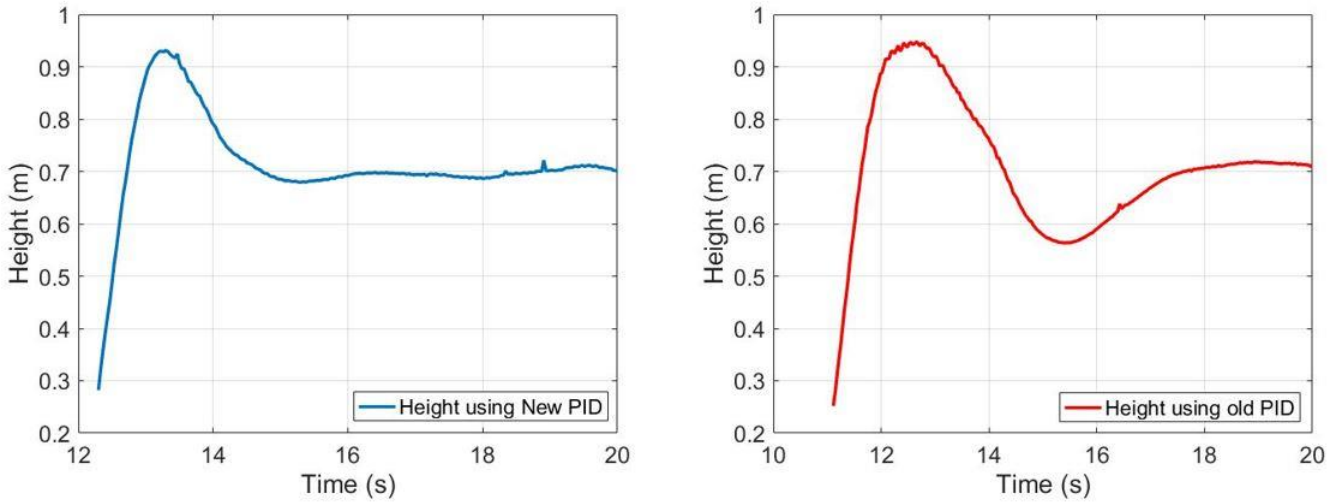

Figure 7.9: These images show the comparison of the height stabilization of the quadrotor between the new PID values and the old PID values.

Last figure represents the height trajectories of the flights, the blue graph represents the trajectory using the new PID values and the red graph represents the trajectory using the old PID values.

If one takes a look into the old PID values trajectory, it presents an overshoot on the takeoff of almost 25 cm in the trajectory and after that, it presents a fluctuation of 10 cm before to stabilize in the reference height that is 70 cm.

Notice also that the time that takes to stabilize is more or less 7 seconds after the takeoff.

On contrary, the new PID values trajectory presents a smooth behave but the initial overshoot remains. This overshoot was not tried to change in order to avoid the ground effect on the takeoff because to delete this overshoot, the speed of the taking off has to be less and that means the ground effect is going to be higher and the quadrotor is going to lose control on the takeoff.

Apart of that, this new trajectory does not present any fluctuation and it has a quick stabilization of more or less 3 seconds after the takeoff.

Finally, another aspect of the flight control is going to be compared. With the old values of the PID, the positioning control when the quadcopter was hovering over the ArUco marker was quite good.

To see if with the new values of the PID presents an improved behavior, a test flight was done.

On this test flight, the quadrotor is going to be located in the middle of the ArUco marker on the ground and is going to take off in order to achieve a height of 70 cm over the marker.

The objective of this test is compare the stability of the positioning control between the old and the new values of the PID when the quadcopter is near the goal position.

In figure 7.10 appears the trajectories done by the quadrotor in the test using both PID values.

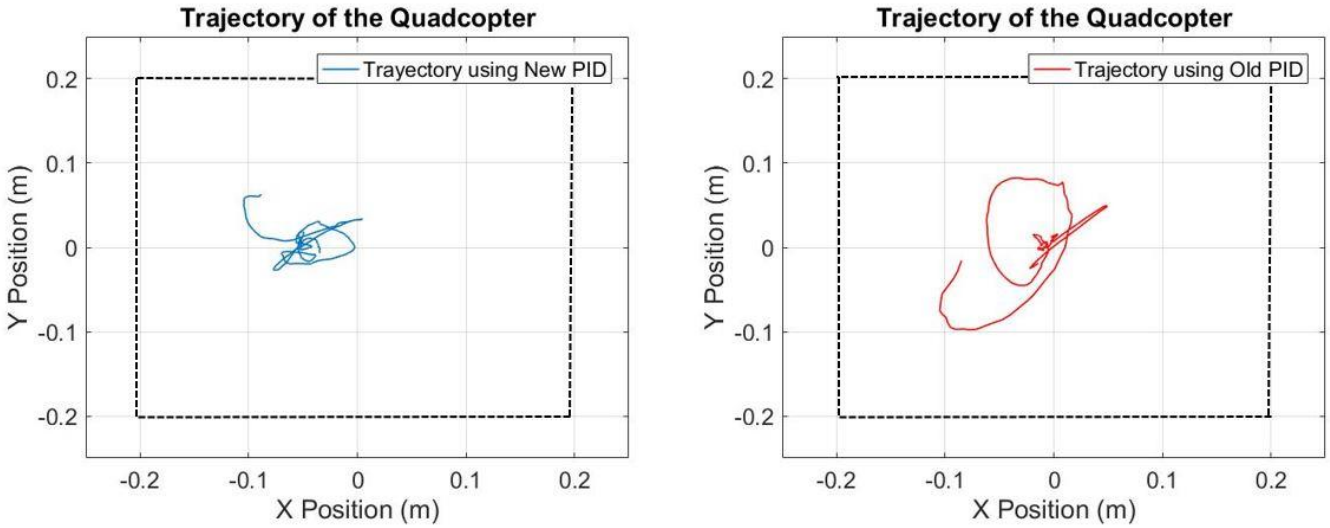

Figure 7.10: These images show the comparison of the trajectories of the quadrotor on x-y plane when is hoovering the ArUco marker between the new PID values and the old PID values.

Last figure represents the trajectories of the flights, the blue graph represents the trajectory using the new PID values and the red graph represents the trajectory using the old PID values.

The ArUco marker is located in the center of the axes and is represented using a black rectangle. If one takes a look into the old PID values trajectory, it presents a small overshoot but it has a good behavior and stabilization and the movements are smooth.

On the other side, the new PID values trajectory presents a more accurate positioning and a minimum overshoot but the movement are brusquer.

On figure 7.10, it can be compared which values of the PID produce a better stabilized trajectory but it cannot be evaluated the time response.

To evaluate the time response, it is going to be compared the evolution of the x and y position among time from the cases exposed on figure 7.10.

The objective of this comparison is to observe which PID presents a better behavior when the quadcopter is stabilized on a position over the ArUco marker.

To realize this comparison the next figures are going to be shown:

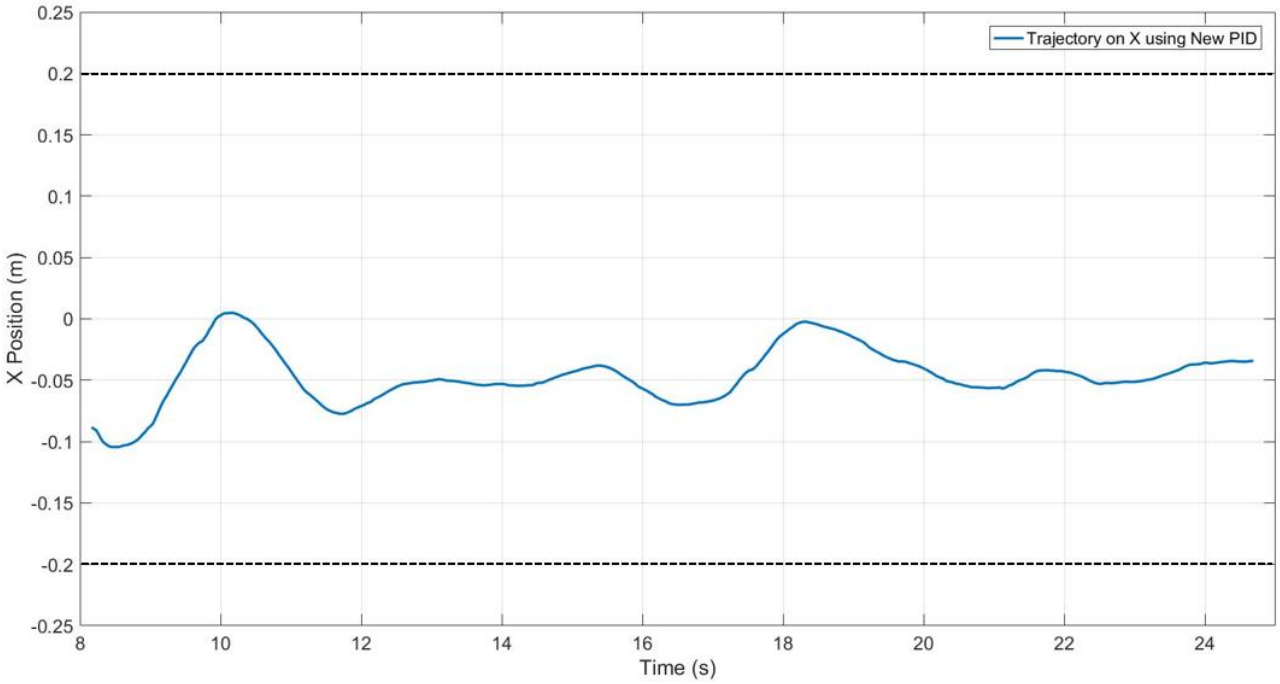

Figure 7.11: This image shows the x position evolution among time while the quadcopter is realizing the trajectory showed on figure 7.10 using the new PID values.

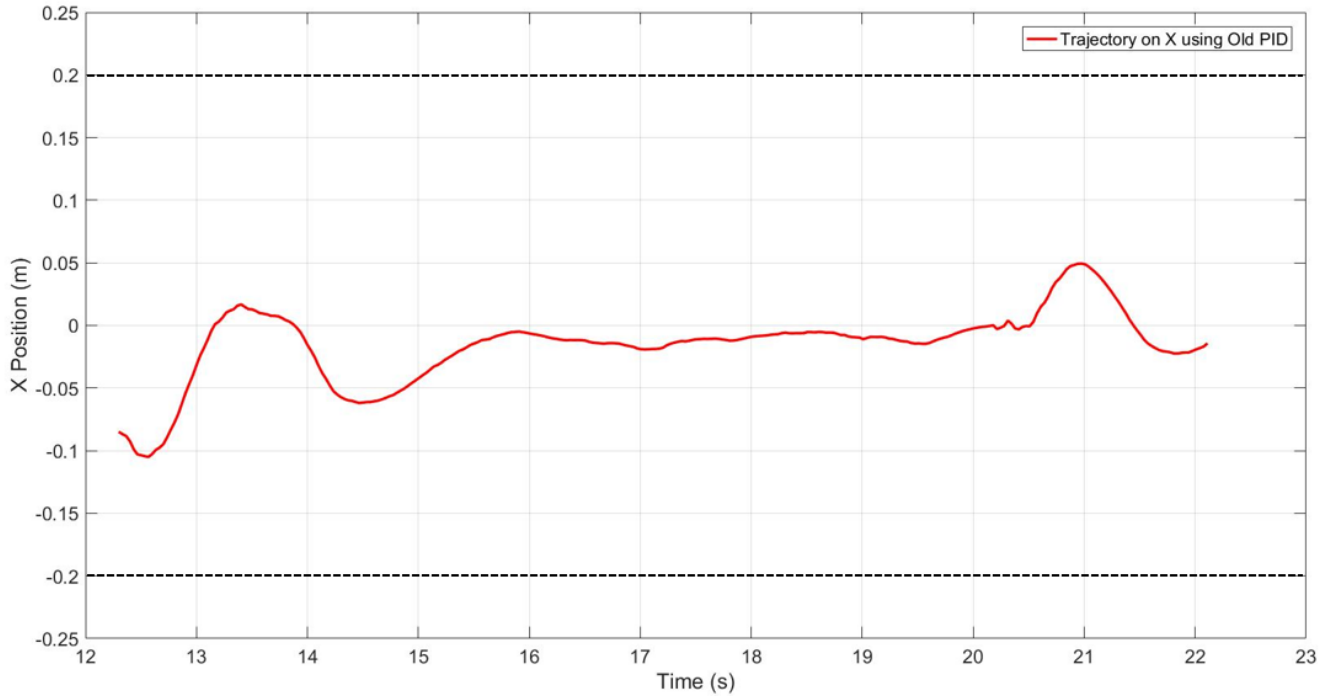

Figure 7.12: This image shows the x position evolution among time while the quadcopter is realizing the trajectory showed on figure 7.10 using the old PID values.

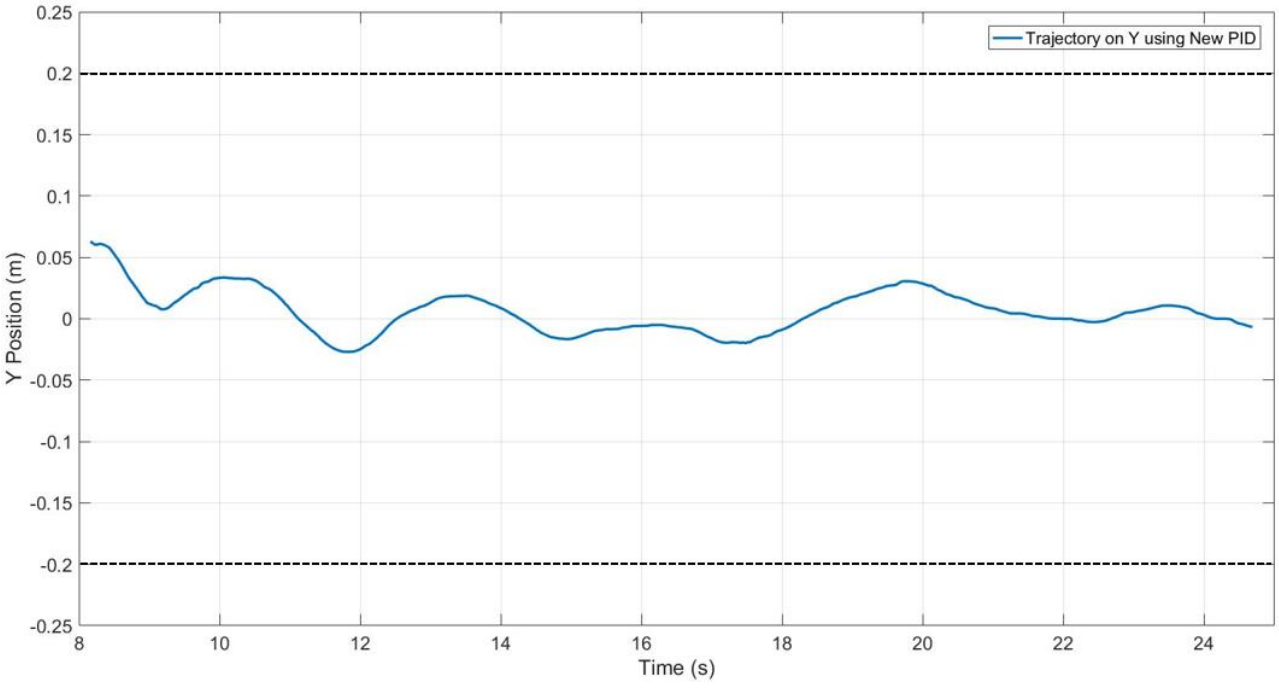

Figure 7.13: This image shows the y position evolution among time while the quadcopter is realizing the trajectory showed on figure 7.10 using the new PID values.

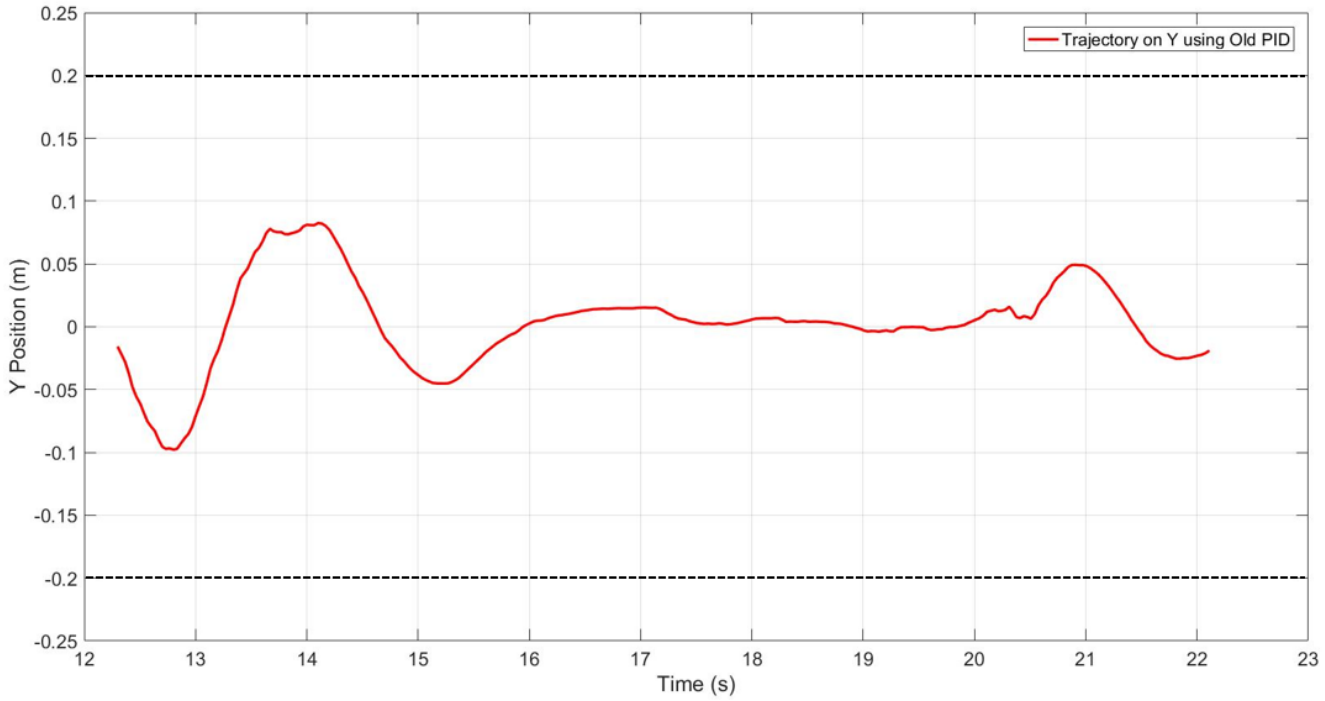

Figure 7.14: This image shows the y position evolution among time while the quadcopter is realizing the trajectory showed on figure 7.10 using the old PID values.

After comparing last figures, it is noticed that the difference between using the new or the old PID values does not make a remarkable variance in the behavior of the quadrotor when it is stabilized over the ArUco marker.

Using the new PID values, the behavior is very stable and it does not seem that it presents problems in time response.

The same happen for the behavior using the old PID values, the only little difference showed is that with the new values the controller makes more corrections of the trajectory and, on contrary, the old PID values presents a smother trajectory with a minimum number of corrections.

After looking all this tests and comparison, the conclusion is that with the new PID values that were find, the flight of the platform has improved in all the aspects in comparison of the old values.

With the new values of the PID controller, the quadrotor flies faster and more accurate. That means the quadcopter can be stabilized over the ArUco marker with an inferior time and it does not present any overshoot on the x-y position controller.

The height control of the quadcopter has also been improved achieving an inferior stabilization time and the overshoots on the height position controller were reduced but not eliminated.

8 Future Work

The work realized on this thesis could be used to implement other research projects of interest. The possible extensions, modifications and improvements are listed as follows:

1. The location system used to obtain the position of the quadcopter with ArUco markers presents considerable errors on the estimation. It would be of interest to analyze other possibilities to improve the location system like changing the ArUco markers for dynamic markers or combined markers.

2. The flight of the quadcopter produces vibrations on its body and affects the chasing of the camera that it moves and produces errors in the location. It would be good to design and implement a better chasing for the camera with a stronger join and good absorption of vibrations.

3. The designed 3D plotter that shows the trajectory of the quadcopter works properly but it doesn't show the orientation of the vehicle. It would be good to improve this tool by adding a 3D model of the quadcopter at the trajectory to be able to see the angle tilts of the quadcopter.

4. It would be interesting to implement a velocity control and make a comparison of the performance versus the actual controller.

5. Nowadays, one of the most interesting fields for the quadcopters are the cooperative work and it is full of possibilities. An interesting future research could focus on synchronize more than one Crazyflie 2.0 to cooperate in duties or flight at the same time without collisions.

6. The method used to find the different transfer functions of the quadrotor could be extrapolated for other models and the simulator created could be adapted to these new quadrotors very easily.

9 Conclusions

The main objective of this thesis was to realize a modeling and identification of the modified Crazyflie 2.0 from the ETiT department of Technische Universität Darmstadt and improve the behavior of this one.

After realizing the thesis, I can say that the main objective was achieved by founding the different transfer functions that define the model with an elevate grade of concordance.

The methodology used in this work to find the model of the quadcopter is documented in order to be reproduced again or be extrapolated into another project and guide step by step through the different experiments needed to realize this identification.

Some tools and programs were created to realize the identification and they can be easily modified to be used in other projects.

Using the tools and programs created, there were found new values for the PID that control the position of the Crazyflie when this one is flying over ArUco markers.

These new values were tested and the results were good and it was achieved a better behavior for the flight of the quadcopter.

In conclusion, I think the objectives of the thesis were achieved. It was found a good modeling and identification of the Crazyflie 2.0 and the flight control of this one suffers a considerable improvement.

The methodology used was tried to be as much accurate as it could to the reality and the different processes used on the thesis are explained clearly in order to be able to be reproduced again in the same platform or in another different.

Bibliography

[1] Hoffman, G.; Huang, H.; Waslander, S.L.; Tomlin, C.J. (20–23 August 2007). *Quadrotor Helicopter Flight Dynamics and Control Theory and Experiment*. In the Conference of the American Institute of Aeronautics and Astronautics. Hilton Head, South Carolina.

[2] Kenzo Nonami, Farid Kendoul, Satoshi Suzuki, Wei Wang, and Daisuke Nakazawa. 2010. *Autonomous Flying Robots: Unmanned Aerial Vehicles and Micro Aerial Vehicles (1st ed.)*. Springer Publishing Company, Incorporated.

[3] Ian G. R. Shaw, (2014), "The Rise of the Predator Empire: Tracing the History of U.S. Drones", Understanding Empire, https://understandingempire.wordpress.com/2-0-a-brief-history-of-u-s-drones/ . September 2017.

[4] Tüylü, Tobias. 2016. *Velocity controller design for a fly-by-wireless quadcopter*. Master-thesis, Technische Universität Darmstadt.

[5] http://wiki.ros.org/. ROS Wiki. September 2017.

[6] Didion, M.; Eisenhauer, R; Kraus,D; Lechner, G. 2017. *Integration of a Mini Camera into a Tiny Quadrotor for a Navigation with Visual Markers*. Projektseminar Automatisierungstechnik, Technische Universität Darmstadt.

[7] Teppo Luukonen (22th August 2011). *Modelling and control of quadcopter*. Aalto University.

[8] www.bitcraze.io. Crazyflie 2.0. https://www.bitcraze.io/crazyflie-2/. September 2017.

[9] I. 2017 Bitcraze ©1994-2017 The Mathworks. Crazyflie wiki, 2017.

[10] Leishman, J. Gordon. *Principles of Helicopter Aerodynamics*. Cambridge University Press, 2006.

[11] Mahony, R; Kumar, V; Corke, P. 2012, *Multirotor Aerial Vehicles: Modeling, Estimation, and Control of Quadrotor*, IEEE Robotics and Automation Magazine. September 2012.

[12] Gang Hua ; Hervé Jégou. *Autonomous flight system using marker recognition on drone*. Computer Vision - ECCV 2016 Workshops - Amsterdam, 2016.

Annexes

Annexes Content

Annex 1: 3D Pose Animation Matlab Code

```
%Matlab code for the 3D pose animation of the quadcopter.

%The positions in x,y and z obtained by simulation are extracted.
x1 = x.data;
y1 = y.data;
z1 = z.data;

%It is created an animated 3D line with X,Y and Z axes.
figure('Name','3D Trajectory of the Quadcopter','NumberTitle','on')
curve = animatedline('LineWidth',2);
set(gca,'XLim',[-2 2],'YLim',[-2 2],'ZLim',[0 1.25]);
grid on;
box on;
ax = gca;
ax.YLabel.String = 'Y Position (m)';
ax.YLabel.FontSize = 14;
ax.XLabel.String = 'X Position (m)';
ax.XLabel.FontSize = 14;
ax.ZLabel.String = 'Z Position (m)';
ax.ZLabel.FontSize = 14;
title('3D Trajectory of the Quadcopter')
view(43,24);
hold on;
%It is created a loop to plot the pose of the quadcopter in real-time.
for i=1:length(z1)
    addpoints(curve,x1(i),y1(i),z1(i));
    head = scatter3(x1(i),y1(i),z1(i),'filled','MarkerFaceColor','b');
    drawnow
    %A pause is added to calibrate the execution time for an specific PC
    pause(0.02);
    delete(head);
    z.time(i)
end
hold off
%A new figure is created with an overall view of the quadrotor trajectory.
figure()
 hold on
 grid on;
view(43,24);
 plot3(x1,y1,z1)
 hold off
```

Annex 2: Resampling and Transfer Function Finding Matlab Code

```
%Matlab code to resample the variables into the same length time vector and finds
the transfer function between the two variables.

%First there are selected the two vectors to adjust with their respective times.
A = comand_time;
B= selection_comand_y;
C= test_time;
D=selection_x;
i=1;
j=1;
time1=A;
time2=C;
value1=B;
value2=D;
[max_row1,max_column1] = size(value1);
[max_row2,max_column2] = size(value2);

%A matrix is created with the same length as the longest vector
Matrix_resample_time= ones(max_row1,1);
Matrix_resample_data= ones(max_row1,1);
%To adjust the length of the matrix is it used a extrapolation of the values to
fill the empty spaces of the matrix.
while(i < max_row2)
      if(time1(j) >= time2(i))
      Matrix_resample_time(i) = time2(i);
      Matrix_resample_data(i) = value1(j);
    end
    if(j<max_row1)
     if(time1(j) < time2(i))
         j=j+1;
         Matrix_resample_data(i) = value1(j);
         Matrix_resample_time(i) = time2(i);
     end
    end
   i=i+1;
end
%The result is going to be saved into a timeseries variable and plotted.
Matrix_resample_time(i) = time2(i);
Matrix_resample_data(i) = value1(j);
ts_resample_comand_y_zoom =timeseries(Matrix_resample_data,Matrix_resample_time);
figure()
hold on
plot(ts_resample_comand_y_zoom)
plot(ts_orientation_x_deg_zoom)
hold off
%Once both vectors have the same length and time ratio, function sys is used to
obtain the transfer function between them.
data = iddata(selection_x,Matrix_resample_data,0.05)
np=2;
nz=0;
sys = tfest(data,np,nz)
stepplot(sys)
```

Annex 3: Extracting Data from Rosbag file Matlab Code

```
% Matlab code to extract an convert the data from different ROS topics.
filepath = fullfile(fileparts(which('Plot_Experimento_Y')),'\',
'Experimento_cuerda3.bag');
bag = rosbag(filepath)
%Select the topic you want to plot. Example: /turtle1/pose
bagselect1= select(bag,'Time',[bag.StartTime
bag.EndTime],'Topic','/crazyflie/imu');% Selection of topic IMU of the crazyflie
bagselect2= select(bag,'Topic','/crazyflie/cmd_vel');%Selection of topic cmd_vel
of the crazyflie
%Exctract the data as a time series data
ts=timeseries(bagselect1);%ts==> timeseries of the IMU Data
ts2=timeseries(bagselect2);%ts2==> timeseries of the cmd_vel Data
ts.Name = 'IMU DATA';
ts2.Name = 'CMD_VEL DATA';
%Setting the initial ROS time to 0
offset_time = bagselect2.StartTime - bagselect1.StartTime
ts.Time = ts.Time - ts.Time(1);
ts2.Time = ts2.Time - ts2.Time(1) + offset_time;

%Exctract the specific data from the bag
x_ang= getcolumn(ts.Data,8); %Angular velocity in X
y_ang= getcolumn(ts.Data,9); %Angular velocity in Y
z_ang= getcolumn(ts.Data,10);%Angular velocity in Z
comand_x = getcolumn(ts2.Data,1);%Angular Command in X
comand_y = getcolumn(ts2.Data,2);%Angular Command in Y
comand_z = getcolumn(ts2.Data,6);%Angular Command in Z
%Extract the specific Data as a timeseries Data
ts_x_ang=timeseries(x_ang,ts.Time);
ts_y_ang=timeseries(y_ang,ts.Time);
ts_z_ang=timeseries(z_ang,ts.Time);

%Plot the specific Data
figure('Name','X and Y Angular Velocity','NumberTitle','off')
hold on
plot(ts_x_ang)
plot(ts_y_ang)
hold off

%This is the angles in radians
orientation_x= (cumtrapz(x_ang))/100; % Angular Position in radians in X
orientation_y= (cumtrapz(y_ang))/100; %Angular Position in radians in Y
orientation_z= (cumtrapz(z_ang))/100; %Angular Position in radians in z
%This is the angles in degree
orientation_x_deg = rad2deg(orientation_x);%Angular Position in degree in X
orientation_y_deg = rad2deg(orientation_y);%Angular Position in degree in Y
orientation_z_deg = rad2deg(orientation_z);%Angular Position in degree in z

.
.
.
```

```
%Extract the Angular Position Data as a timeseries Data
ts_orientation_x=timeseries(orientation_x,ts.Time);
ts_orientation_y=timeseries(orientation_y,ts.Time);
ts_orientation_z=timeseries(orientation_z,ts.Time);
ts_comand_x = timeseries(comand_x,ts2.Time);
ts_comand_y = timeseries(comand_y,ts2.Time);
ts_comand_z = timeseries(comand_z,ts2.Time);
ts_orientation_x_deg=timeseries(orientation_x_deg,ts.Time);
ts_orientation_y_deg=timeseries(orientation_y_deg,ts.Time);
ts_orientation_z_deg=timeseries(orientation_z_deg,ts.Time);
%Plot the Angular Position Data in Radians
figure('Name','X and Y Angular Positions in Radians','NumberTitle','off')
hold on
plot(ts_orientation_x)
plot(ts_orientation_y)
plot(ts_comand_x)
plot(ts_comand_y)
%plot(orientation_z)
hold off
%Plot the Y Angular Position Data in Degree and the X comand
figure('Name','Y Angular Position Data in Degree and the X
comand','NumberTitle','off')
hold on
plot(ts.Time,orientation_y_deg)
plot(ts_comand_x)
hold off

%Plot the  X Angular Position Data in Degree and the Y comand
figure('Name','X Angular Position Data in Degree and the Y
comand','NumberTitle','off')
hold on
plot(ts.Time,orientation_x_deg)
%plot(ts.Time,orientation_y_deg)
%plot(ts_comand_x)
plot(ts_comand_y)
%plot(orientation_z_deg)
hold off

test_time = ts.time(695:2682);
selection_x = orientation_x_deg(695:2682);
selection_y = orientation_y_deg(695:2682);
selection_comand_x = comand_x(352:1354);
selection_comand_y = comand_y(352:1354);
comand_time = ts2.time(352:1354);
ts_orientation_x_deg_zoom =timeseries(selection_x,test_time);
ts_orientation_y_deg_zoom =timeseries(selection_y,test_time);
ts_comand_x_zoom = timeseries(selection_comand_x,comand_time);
ts_comand_y_zoom = timeseries(selection_comand_y,comand_time);


.
.
.
```

```
%The signals recive a smother filter in order to see it better
smooth_selection_comand_x=sgolayfilt(selection_comand_x,1,25);
ts_smooth_comand_x_zoom = timeseries(smooth_selection_comand_x,comand_time);
smooth_selection_comand_y=sgolayfilt(selection_comand_y,1,25);
ts_smooth_comand_y_zoom = timeseries(smooth_selection_comand_y,comand_time);
figure('Name','Smooth Zoom angular Y Position in Degree and Comand
X','NumberTitle','off')
hold on
    plot(ts_orientation_y_deg_zoom)
    plot(ts_smooth_comand_x_zoom)
hold off
figure('Name','Smooth Zoom angular X Position in Degree and Comand
Y','NumberTitle','off')
hold on
    plot(ts_orientation_x_deg_zoom)
    plot(ts_smooth_comand_y_zoom)
hold off
```