Hugo Alexandre Rodrigues Cabrita

Electrical and Computer Engineer

Design and control of a rotary wing drone testbed

Dissertation submitted in partial fulfillment
of the requirements for the degree of

Master of Science in
Electrical Engineering

Adviser: Bruno João Nogueira Guerreiro, Auxiliar Professor, NOVA University of Lisbon

Co-advisers: Luís Filipe Figueira Brito Palma, Auxiliar Professor, NOVA University of Lisbon

Fernando José Vieira do Coito, Associate Professor, NOVA University of Lisbon

Examination Committee

Chairperson: Professor João Paulo Branquinho Pimentão
Raporteur: Professor José António Barata de Oliveira
Member: Professor Bruno João Nogueira Guerreiro

FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

November, 2020

Design and control of a rotary wing drone testbed

This document was created using the (pdf)LaTeX processor, based in the "novathesis" template[1], developed at the Dep. Informática of FCT-NOVA [2].
[1] https://github.com/joaomlourenco/novathesis [2] http://www.di.fct.unl.pt

Acknowledgements

It is with my great pleasure that I thanks all the staff from the Faculty of Science and Technology for the excellent environment and learning conditions offered throughout the realization of my academic studies. A special thanks to my thesis coordinator Prof. Bruno Guerreiro for his tireless support and for the constant interest during the accomplishment of this master's thesis.

Furthermore, I would like to thanks all my friends that accompanied me during these times, that provided some distraction when I most needed, specially when things weren't going as good as planned and allowed me to restore my confidence. Specially to my friend Bruno with whom I could share knowledge with in the beginning of the thesis and to my friends Rúben, Raquel and Francisco that managed to encourage and help me when I most needed.

An enormous thanks to all my family that supported me since I can remember, specially to my parents, my sister and my girlfriend that never doubted of my capabilities and always believed me and pushed me to maintain positive even on the hardest moments.

This work was partially funded by the FCT project REPLACE (LISBOA-01-0145-FEDER-032107) which includes Lisboa 2020 and PIDDAC funds, and also project CTS (UIDB/EEA/00066/2020).

Abstract

In order to improve the development of unmanned aerial vehicles it is mandatory to make, not only virtual simulations using proper software, but also real experimental trials. Furthermore, to enable rapid prototyping, a controlled environment with all the necessary hardware and software must be thoroughly prepared. The creation of a testbed will solve this need and that is precisely one of the objectives of this thesis, allowing future drone developments in the Faculty of Science and Technology. Simultaneously, another aim of this thesis is to design a quadrotor controller that will be implemented and tested, building on existing methods, but introducing some changes.

Overall, the goal is to show what's involved in the process of developing a drone controller and create a space where it can be safely tested. This work will also show different ways on how to integrate Marvelmind, which is motion capture system, with the drone autopilot, in this case PX4. This testbed allows to replace the GPS system and furthermore test the controller, while at the same time providing the space so that future developments can take place.

As this is a recent area of study and there are not many known publications regarding some aspects mentioned in this thesis, the presented material might help others by providing with some important information and guidance in the right direction.

Keywords: Drone; quadcopter; control; Marvelmind; PX4; autopilot; PixHawk 4 Mini; testbed; Motion Capture system.

Resumo

De modo a melhorar o desenvolvimento de veículos aéreos não tripulados é necessário não só a realização de simulações virtuais utilizando programas adequados, mas também proceder à realização de testes de caráter real. De modo a permitir a prototipagem rápida, um ambiente controlado com todo o *hardware* e *software* deve ser preparado com máximo rigor. A criação de uma arena de testes resolve este problema, sendo que este é precisamente um dos objetivos desta dissertação, permitindo ainda desenvolvimentos futuros na área de drones, dentro da Faculdade de Ciências e Tecnologia. Simultaneamente, outro objetivo que se pretende alcançar é o dimensionamento e teste de um controlador para um quadricóptero com base num outro controlador já existente, adicionando algumas alterações.

De um modo geral, o grande objetivo é mostrar os processos envolvidos no dimensionamento de um controlador de drones, e a criação de um espaço onde é possível testar os mesmos. Pretende-se também exemplificar diferentes métodos para integrar entre um sistema de captura de movimento, baseado no sistema Marvelmind, com um módulo de piloto automático, sendo que o utilizado foi o PixHawk 4 Mini. Esta arena de voo permite então a remoção do módulo de GPS e, adicionalmente, testar não só o controlador dimensionado como também o presente no piloto automático referido.

Visto que esta é uma área com um foco de estudo recente, ainda não existe muita investigação relacionada com certos aspetos mencionados, podendo a contribuição presente nesta Dissertação de Mestrado servir como inspiração, motivação e ponto de partida para investigação futura neste mesmo ramo.

Palavras-chave: Drone; quadricóptero; controlo; Marvelmind; PX4; piloto automático; PixHawk 4 Mini; arena de testes; Sistema de captura de movimento;

Contents

List of Figures

List of Tables

Glossary

Dirac	Comes from the Delta Dirac function, which represents an unit impulse symbol. An example of this function is that its value is always zero except in the origin where it takes the infinite value.
Gimbal lock	In a three-dimensional space, when two of the axes are moved into a parallel configuration, a degree of freedom is lost, resulting in a "locked" two-dimensional system.
Ground Control Station	Control center that provides the environment so that humans can control UAVs or any other unmanned vehicles.
Hover	Maintaining an aerial vehicle in the air in the same place
Linux	Open source operational system based on Kernel Linux
MAVLink	Low complexity and very lightweight messaging protocol allowing the communication between UAVs and between their components
MAVROS	ROS node that provides a communication driver for various autopilots with MAVLink communication protocol
NMEA 0183	Set of electrical and data specifications for the communication of marine electronical devices such as GPS receivers, sonars, gyroscopes, autopilots, etc.
Odometry	Usage of data from different motion sensors with the objective of estimating the position over the course of time
Quadrotor	Also referred as quadcopters, are helycopters with four rotors, commonly referred to drones.
Testbed	Pack of equipment used to test machinery, mainly aerial vehicles such as quadrotors

Acronyms

2D	Two-dimensional
3D	Three-dimensional
3DR	3D Robotics
AI	Artificial Intelligence
EKF2	Extended Kalman Filter 2
FCU	Flight Control Unit
GCS	Ground Control Station
GND	Graduated neutral density
GPS	Global positioning system
IMU	Inertial measurement unit
IP	Internet Protocol
LED	Light-emitting diode
LPE	Local Position Estimator
LQR	Linear-Quadratic regulator
MATLAB	MATrix LABoratory
MoCap	Motion Capture
MPC	Model predictive control
NED	North East Down
PID	Proportional-Integral-Derivative
PX4	PixHawk 4
QGC	QGroundControl

RC	Radio control
ROS	Robot Operating System
RTLS	Real-time location system
RUAV	Rotary-wing unmaned Aerial vehicle
SMC	Sliding Mode Controller
UART	Universal asynchronous receiver/transmitter
UAV	Unmaned aerial vehicle
UDP	User Datagram Protocol
USB	Universal Serial Bus
UWB	Ultra-wide band
VIO	Visual Inertial Odometry
VSC	Variable Structure Control
Wi-Fi	Wireless Fidelity

Symbols

Subscripts and Superscripts

$\vee$	Vee map
$\mathcal{B}$	Body frame
$\mathcal{C}$	Camera frame
des	Desired
max	Maximum value allowed
min	Minimum value allowed
ref	Reference
T	Transpose
$\mathcal{W}$	World frame
x, y, z	Cartesian components

Greek symbols

ω_B	Angular velocity in the body frame
ω_i	Angular velocity of rotor i
ϕ, θ, ψ	Roll, pitch and yaw Euler angles

Roman symbols

${}^{a}R_{b}$	Rotation matrix of b described in relation to frame a
a	Acceleration in the world frame

e_P, e_V, e_R	Position error, Velocity error and Rotation error
F_{des}	Desired force of the controller
F_i	Individual force in rotor i
M_i	Momentum in rotor i
$\mathcal{I}$	Inertial frame of the body
K_p, K_R, K_ω, K_v	Controller gains
m	Mass
u	Control input
u_1	Net body force
$\mathrm{u}_2, \mathrm{u}_3, \mathrm{u}_4$	Body moments in the x, y and z axis
v	Velocity in the world frame

Mathematical notation

$\times$	Cross product
dim	Dimension of array
$\hat{x}_B$	Skew matrix of x
$\dot{x}$	Derivative of x
$\ddot{x}$	Second derivative of x
x^*	Complex conjugate of x
$\overline{x}$	Mean of x
x^{-1}	Inverse matrix of x
x^T	Transpose matrix of x

Chapter 1

Introduction

The evolution of technology in the world we live nowadays has been exponentially growing. With that growth there's always the need of something more, something better, something able to solve more problems and in other situations to improve the way things are done, both in terms of time and efficiency.

Rotary-wing unmanned aerial vehicles (RUAVs), commonly called drones, appeared with the capacity of solving a lot of problems and improving the way things are done. The list of cases in which this recently developed technology acts is vast. Examples of this technology include architectural 3D rendering, something that before was a time-consuming and difficult task, nowadays with the usage of drones it became an extremely simplified task, or the plethora of military applications, such as supporting forest surveillance in the critical fire period and aerial photography/footages. Another dramatically increasing use of these devices are for transport and delivery purposes, for example to assist vaccine supply chains in low and middle income countries that face enormous challenges [22], and with the evolution and over the years the applications will go even further.

1.1 Motivation

Although their applications are enormous, they require a lot of researching and development, and obviously, often face some difficulties, such as obtaining an accurate position of the drone in real time, or even perform a trajectory planning, obstacle avoidance, or also coordination between multiple drones and research regarding battery autonomy that limits the flight time.

The need of implementing a controller that is able to satisfy our needs and is able to perform consistently is a fundamental part of the design of an UAV and is currently one of the aspects that has been the scope of a significant amount of research over the last

decade and still needs to be improved.

Additionally, in order to test the developed controller and after the virtual simulations have been successfully done with good results, there is a need to make real experimental trials with drones, therefore it is required to have a reliable testbed which offers the needed conditions to accomplish it, contemplating the necessary software and hardware.

As explained in [36], testbeds have a big importance when it comes to multi-robot research with UAVs as they are usually lightweight and easily influenced by the dynamics of the environment and can change their state in a matter of milliseconds. This testbed consists in a sort of arena that has all the required sensors to be able to perform tests with UAVs in the best conditions possible, even allowing trials with multiple UAVs. This structure will provide the conditions to experiment controllers that are already designed in autopilot modules or even custom controllers that haven't been tested before.

It is also important to refer that in order to have an autonomous UAV flying, the controller can be implemented in the autopilot module, or it can be running outside the autopilot and for that the custom controller must send, for example, velocity or attitude instructions to the autopilot (if the autopilot is flexible enough to allow it).

This autopilot module is basically what allows the UAV to guide itself without human assistance. It grands the UAV the ability to autonomously fly by itself and achieve a set of certain goals implemented by humans accordingly to the necessities of each experiment.

1.2 Objectives

Throughout the development of this thesis, some main objectives are considered:

- Implement a non-linear controller robust enough to control the UAV in a real environment.
- Design and create a drone testbed that provides the necessary conditions to perform real experimental trials so that UAV controllers can be tested, using a Motion Capture System to get the drone's position in real-time.
- Show different possibilities of integrations between a MoCap system and a quadrotor autopilot.

In this first stage of implementing the non-linear controller, some considerations will be taken in order to achieve a robust and well designed controller that is able to adapt to environmental changes such as wind and moving objects. One of the main goals is to have the possibility of following trajectories in an autonomous flight.

The second stage of this thesis is the creation of the testbed that will follow a couple of steps. In first place is the design which consists in choosing its dimensions and the most adequate sensors, while having in consideration the relation between their price and features/performance offered. After the design is done, the physically implementation is the next step, so that not only the drone controller made in the first stage can be tested,

but also allow future developments in the area of drones, as it provides a safe environment where aerial vehicles can safely fly.

Upon that, some steps of how to proceed with the integration of the systems that play a role in the testbed will be shown, working as a guide to others that might be wanting to achieve similar goals. Furthermore some experimental trials of the systems will be shown.

1.3 Problem statement and proposed solution

The control of an unmanned aerial vehicles is always a complex task as it is constantly facing the gravitational force, apart from all the other factors as wind and moving objects, and it needs to be able to keep a steady movement and be able to cancel these environmental forces. Having this in consideration, the implementation of a controller is a crucial part of these aerial vehicles, and to be able to fulfill these requirements it is needed a non-linear controller as they can actively adapt to these changes which is a considerable advantage when comparing to linear controllers. The decision of what controller to use is an important task as there are quite some non-linear controllers that can be used to do this task, each of them having their benefits and disadvantages, but this matter will be discussed ahead in the thesis.

In order to test a drone controller it is required a place that is able to provide the best environmental conditions before outdoors tests. One problem with having autonomous flying aerial vehicles is to know their exact position, for example if you would to fly a drone outdoors, the GPS system can be used even though it has an intrinsic uncertainty, as the conditions in outdoor trials usually allow to have uncertainties around a couple of meters, without compromising safety. Nonetheless, this can be a problem if there is an autonomous drone near obstacles.

Using this GPS module indoors would be completely unbearable as the conditions don't allow to have such position uncertainty, while the GPS signal reception quality is so degraded that the module might not even be able to compute a position solution. In order to avoid this problem, the testbed should come handy allowing the test of multiple drones in the same area as they should be able to have an irrelevant imprecision in their position. In order to achieve this goal, the choice of what positioning system to use is an important matter as it can be done by many methods, examples of these systems will be shown ahead in the thesis.

As for the solution encountered during the next chapters, it includes the usage of Marvelmind system as being the motion capture system responsible by following the drone's trajectory and capture its position. Furthermore, the position will be transmitted to a ROS topic and sent to the PixHawk 4 Mini autopilot. With this solution, and using a proper estimation algorithm that will fuse the data from the sensors belonging to the IMU with the external position coming from Marvelmind, the quadrotor will be able to know

its local position and all its states, allowing the quadrotor to perform a safe autonomous flight.

1.4 Thesis structure

The structure of the remaining chapters of this thesis, are as follows:

- Chapter 2 shows solutions already implemented applying different strategies, both in terms of drone controlling and technologies to apply in testbeds.
- Chapter 3 presents some concepts that are fundamental to understand the course of the thesis. Furthermore, it shows a mathematical model and a complementary controller able to perform trajectory following and some simulation results are shown to verify its robustness.
- Chapter 4 exposes a vast list of steps that had to be done so that a testbed providing a safe environment for real experimental trials could be made. Furthermore, tests were made to find if the accuracy announced by the manufacturer (Marvelmind) was being achieved.
- Chapter 5 reveals simulation results, not only using the controller from Chapter 3 but also using the controller provided by PX4. Additionally, it shows two possible ways of how to integrate Marvelmind system with PX4. Finally, some configurations needed to have the environment ready for real simulations are shown and a first real experimental test is made.
- Chapter 6 sums up the goals that were achieved during the whole thesis and discusses some examples of possible future works that could be done.

Chapter 2

Related work

The research in this area has been exponentially growing in the last decade, hence there's already a lot of developed work that will be analysed and taken into consideration. In this section some of these studies will be stated and ideas will be taken from them. This section will be divided into two subsections, first the related work with implementations of controllers and then related studies with drone testbeds.

2.1 Drone control methods

In order to have a fully operating UAV it is expected that there is a controller that will take responsibility for its movement, including not only simple movements but also all types of maneuvering that might be expected from the UAV, such as performing complex trajectories and hovering and obstacle avoidance. In this matter there is a need to define how we are going to model the UAV. In terms of controllers, these can be separated into two big categories: linear controllers and non-linear controllers.

As it is expected, each of these categories have their benefits and cons, the main advantage of using linear controllers is that the complexity is much reduced in comparison to non-linear algorithms, but in the other hand, non-linear algorithms demonstrate to be a much more robust solution as it allows the UAV to have a more stable behaviour and a lot more possibilities in terms of realizing hard tasks such as trajectory planning, multi-drone coordination and most importantly it shows better performance when it comes to environmental disturbances.

2.1.1 Linear control

In terms of linear algorithms, there are a few algorithms that have shown already good results given their small complexity. Usually these linear control techniques come from linearizing the dynamics around a certain operating time, normally being the hover.

The **proportional–integral–derivative (PID)** controller is a good example of a linear controller that is able to show acceptable results for the hover controller, as shown in [36]. This controller is vastly used in industrial control and essentially it is continuously calculating the error *e(t)* between the output value and the desired setpoint, applying a correction based on the proportional, integral and derivative terms. The figure 2.1 shows a schematic of the operation of this controlling method.

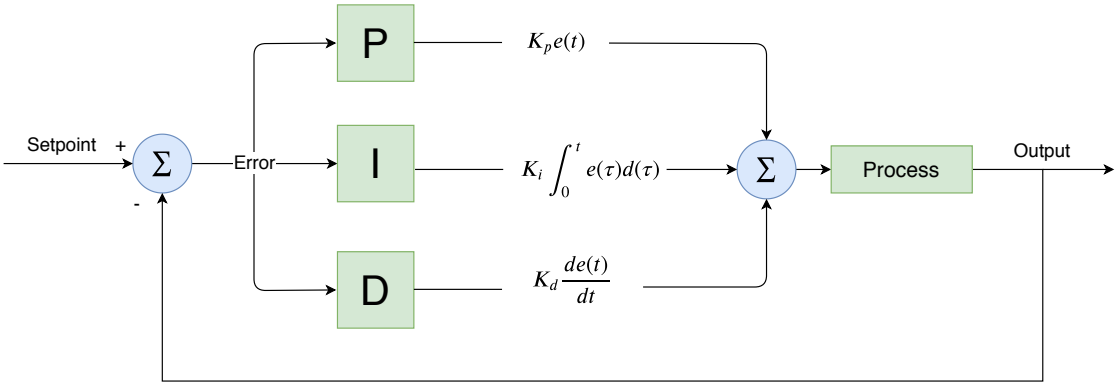


Figure 2.1: Block diagram of PID controller

Another linear controller is the **Linear-Quadratic Regulator (LQR)** which explanation and usage is well demonstrated in [13]. In its essence, this method aims to calculate the optimal feedback gain K which can be determined by tuning Q and R based on real test flight experiment results rather than simulation [49].

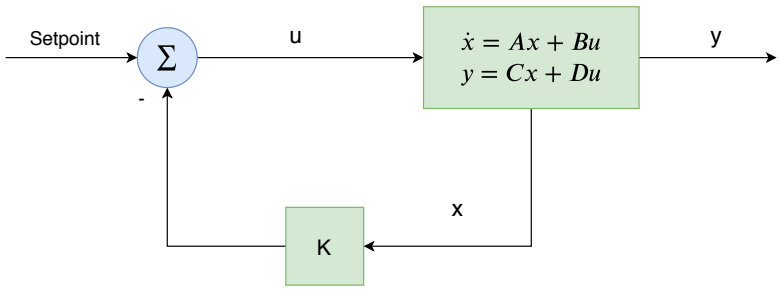


Figure 2.2: Block diagram of LQR controller

Model predictive control (MPC) is also a valid option to take into account as it has shown to be a robust and reliable option, as shown in [3, 24]. MPC is essentially an advanced method of process control that is used to control a process while satisfying a set of constraints. The main advantage of MPC is the fact that it allows the current timeslot to be optimized, while keeping future timeslots in account. This is achieved by optimizing

a finite time-horizon, but only implementing the current timeslot and then optimizing again, repeatedly. Another big advantage is that it has the ability to anticipate future events and take control actions accordingly. Figure 2.3 shows the block diagram of this method.

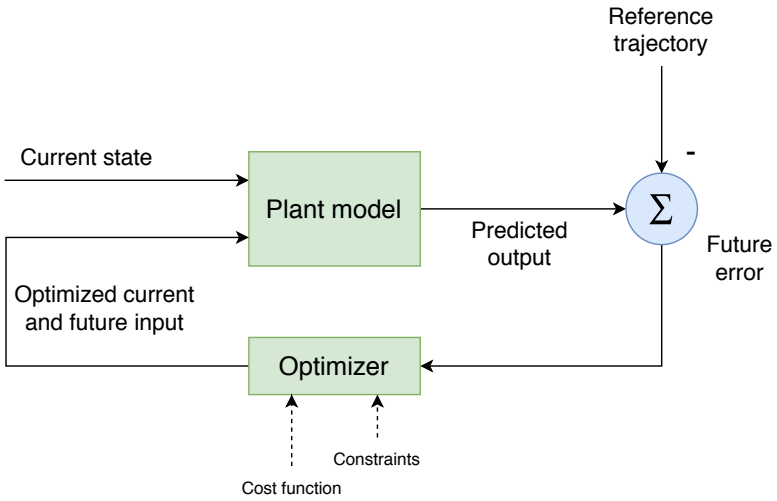


Figure 2.3: Block diagram of MPC

2.1.2 Non-linear control

When it comes to non-linear based controllers, there are a couple of algorithms/techniques that have gained popularity when it comes to UAV controlling. We have the example of Backstepping based controllers, SMC (Sliding Mode Controller) and feedback linearization.

Backstepping is a technique that allows to design stable controllers for non-linear systems. It has been applied in many situations when it comes to trajectory tracking as it allows the UAV to make smooth routes while at the same time it takes into consideration disturbances, e.g. wind. Theoretically, as it has a recursive structure, the designer can start the design process at the known-stable system and "back out" new controllers that progressively stabilize each outer subsystem. A good example of implementation of backstepping can be seen in [8]. [43] shows an example of strict-feedback systems that can be expressed by the following generic formulas

$$\begin{cases} \dot{\xi}_1 = f1(\xi_1) + g_1(\xi_1) \\ \dot{\xi}_2 = f2(\xi_1, \xi_2) + g_2(\xi_1, \xi_2)\xi_3 \\ \quad \vdots \\ \dot{\xi}_{r-1} = f_{r-1}(\xi_1, \xi_2, \ldots, \xi_{r-1}) + g_{r-1}(\xi_1, \xi_2, \ldots, \xi_{r-1})\xi_r \\ \dot{\xi}_r = f_r(\xi_1, \xi_2, \ldots, \xi_r) + g_r(\xi_1, \xi_2, \ldots, \xi_r)u \end{cases} \tag{2.1}$$

where $\xi_1, \ldots, \xi_r \in \mathbb{R}$ is the control input and f_i, g_i for $i = 1, \ldots, r$ are known functions and u is a scalar input to the system. A closer look at the architecture and functionality of this algorithm can be seen in [28, 29].

Sliding mode controller (SMC) is another non-linear controller able to show very good results when it comes to control quadcopters in real life approaches. This controller is a type of variable structure control systems (VSC system) that have been used in the design of robust regulators, model-reference systems, adaptive schemes, tracking systems, state observers and fault detection schemes. This control method changes the dynamics of a given dynamical system (linear or non-linear) by applying a discontinuous control signal that forces the system to "slide" along a cross-section of the system's normal behaviour. This is a method that has been used for over 50 years in various types of research in scientific and industrial areas due to its simplicity and robustness. Further reading and applied results about this this controller can be seen in [16, 52].

Feedback linearization is another approach to non-linear control design being its objective to algebraically transform non-linear system dynamics into linear ones, so that linear techniques can be applied [49]. Consider the state-space models of the form

$$\begin{cases} \dot{\mathbf{x}} = A \cdot \mathrm{x} + B \cdot \mathrm{u} \\ \mathrm{y} = C \cdot \mathrm{x} \end{cases} \tag{2.2}$$

where $\mathrm{x} \in \mathbb{R}^n$ is the vector of state variables, $\mathrm{u} \in \mathbb{R}^m$ is the vector of control input variables and $\mathrm{y} \in \mathbb{R}^n$ is the vector of outputs. The goal is then to achieve a control law of the form

$$\mathrm{u} = \alpha(\mathrm{x}) + \beta(\mathrm{x}) \cdot \mathrm{v}$$

that results in a linear input-output map with $\mathrm{v} \in \mathbb{R}$ being the input and y the output. Examples of this method where inner feedback linearization control loop is used can be seen in [12].

Despite the satisfactory results given by the linear controllers, the complexity of these controllers are limited when it comes to environmental changes and other characteristics that we're looking for to implement further in this thesis.

Given these facts, due to the need of more complex tasks our focus will go to non-linear algorithms from now on.

2.2 Testbed

We can take The Flying Machine Arena at ETH Zurich [19, 31] as the example of one of the biggest UAV testbed created so far that has allowed a huge development in this area such as high-performance maneuvering, aerial construction which basically consisted in creating a rope bridge using only drones, cooperating flying, improving the battery that is and will always be an aspect to improve, iterative learning that consists in the drone learning how to behave when it needs, for example, to follow a certain trajectory and

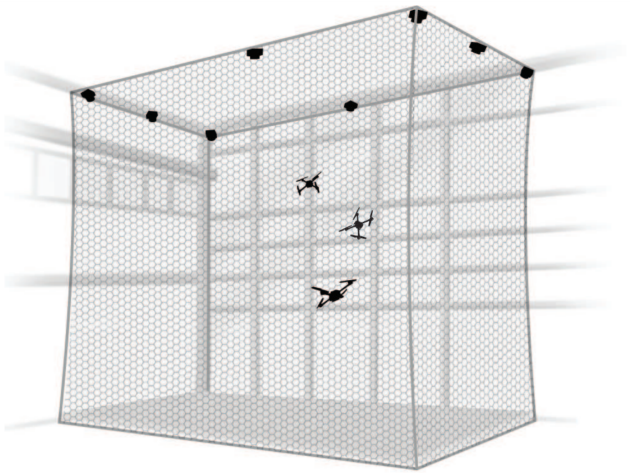


Figure 2.4: The Flying Machine Arena at ETH Zurich [1]

there are perturbations in the environment which is also one very important aspect to take into account as the environment in most real-case scenarios is not static. Figure 2.4 shows a sketch of this arena.

When we think about the testbed, we expect it to be able to localize the UAVs that are being tested. In this matter, which is actually not a trivial problem, there are some viable solutions. Each of these solutions presenting their benefits and disadvantages, we'll discuss a couple of these options, having into consideration the ratio between performance achievable and its cost.

Ultra-wide band (UWB) techniques can be used to position tracking indoors, although this system carries a heavy position error which is unbearable for small indoor spaces, [27] mentions a median error around 40cm and an average error of roughly 56 cm, which are values way above the expected for our purposes. Despite this system having an affordable price overall, the performance achievable is just not yet good enough for unassisted quadrotor flight purposes.

Another interesting solution for the problem in case are the conventional 2D laser scanners. If you think about them, it makes no sense to use them out of the box as they do not offer the needed 3D range, yet if these lasers are coupled to a rotation servo they are able to provide an extra freedom degree resulting in a 3D perception. This solution is not often seen in tracking quadrotors as the speed shown by this solution is usually not enough to follow a moving aerial vehicle. In terms of costs of this solution, even though they are not the most expensive, the results reached with this technique are not the best, an example of tracking the position of a mini-helicopter using this method can be seen in [32]. Figure 2.5 helps visualizing the implementation of this method in a more concise way.

Another possible and inexpensive method is using **visual tags** [1] placed on the

[1] Adapted from [48]
[2] Adapted from [32]

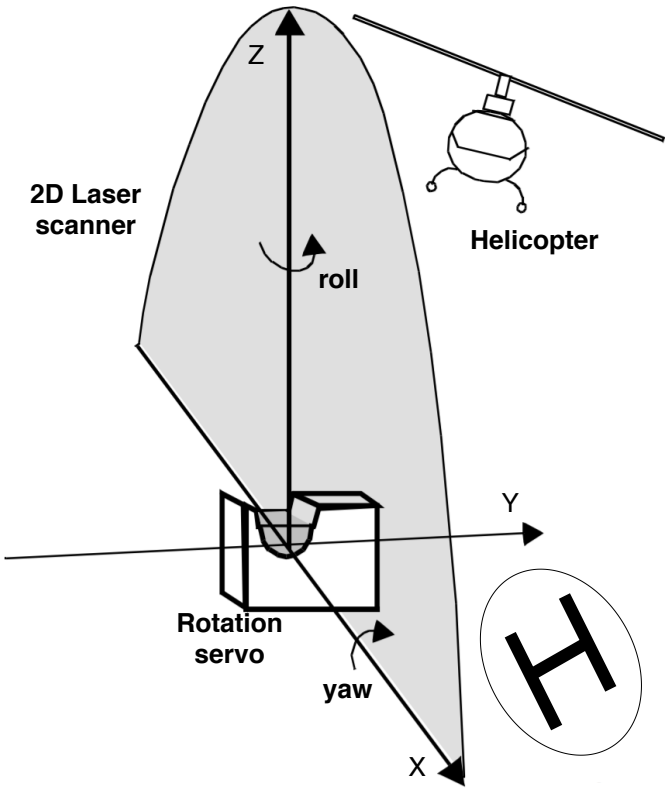


Figure 2.5: 2D laser scanner coupled to a rotation servo [2]

ground at known positions that are dynamically recognized through a camera attached to the bottom of the drone. These tags provide localization and orientation information in the 2D plane. Altitude information is obtained from ultrasound sensors, which are part of a drone's IMU and are employed to this end also in real deployments. Another use of visual tags are the ArUco markers [30], which are basically synthetic square tags composed by a dark border containing an inner binary matrix, a research using this method can be seen in [6] where they announce a precision of less than 7cm which is actually pretty decent having into consideration that it's a super inexpensive and easy method to apply, yet it's not quite the most solid solution.

Motion capture systems, which are composed of high-precision sensors able to give the position in real-time of one or more UAVs, provide one of the best achievable accuracy among the existing solutions, with the counterpart of being usually very expensive. These systems can use a set of high refresh rate cameras using infrared, placed all over a certain area. A known set of these sensors is the *VICON Motion Capture System* [36]. It is fast as it is able to reach a maximum of 375Hz, precise as the deviations of position estimates for single static markers, as they are on the order of 50 microns and robust. Even if all but one camera were occluded, the system will be able to maintain tracking.

Another known capture system of this type is the *OptiTrack Motion Capture* and a couple of accomplished experimental research done with it can be seen in [5, 10].

Another solution that is between the presented ones, as it is not so expensive but is

Figure 2.6: Marvelmind Indoor Navigation System

also able to provide an impressive precision of 2cm is the *Marvelmind Indoor Navigation System* [45] seen in Figure 2.6. This is an off-the-shelf indoor real-time location system (RTLS), designed to provide precise location data to autonomous robots and UAVs mainly. The navigation system consists of a network of stationary ultrasonic beacons interconnected via radio interface in a license-free band, one or more mobile beacons installed on objects to be tracked, and a modem providing gateway to the system from a computer. It calculates the location of the vehicle, drone, human or wherever the beacon is placed in based on the propagation delay of ultrasonic signals between the stationary beacons and mobile beacons (the one(s) attached to the moving part) using trilateration.

Some experiments using this capture system are shown in [46], and [4] shows interesting comparisons with other equivalent systems, such as Pozyx (which announces an accuracy of 10cm) [39], the already mentioned Aruco Marker and VIVE Lighthouse system by HTC that allows a sub-millimetric precision. Experiments seen in [21] shows that the system is capable of achieving impressive precision with a higher rate when comparing with Marvelmind system, while maintaining a similar price point.

Chapter 3

Vehicle components, modeling, and control

This section will have the objective of explaining the composition of the drones, which is essential knowledge if one is making research on this area, and also explaining a bit of the theory behind the movement and rotation of the aerial vehicles. Some of the concepts that will be presented in this chapter are important to easily understand the course of the thesis.

Furthermore, as described in previous chapters, the first objective will focus on the control of the UAV. The controller is expected to be able to have a smooth and accurate movement from one position to another. The control method developed in this chapter is based on the work presented in [34, 35, 50]. This part of the thesis has a great importance, as understanding the concepts behind the control of a drone is a very important basis to start researching in this area.

3.1 Quadcopter components

In order to understand what's behind all the drone controlling process it is crucial to be familiarized with the parts that compose a quadcopter. Apart from the obvious parts such as, propellers, motors, battery and all the video and communication related components, let's focus on the needed sensors and flight controller unit (FCU).

In order to achieve a stable and smooth flying, the usage of a vast list of sensors are mandatory. These sensors are then used by the **flight controller** which is basically the *brain* of the quadcopter, it reads the data from all the sensors to proceed with the control.

One of the most important sensors that must be part of the composition of a quadrotor is the **gyroscope**, which main function is to improve the drone's flight capabilities. The gyroscope needs to work almost instantly to the angular moments acting on the drone to keep it stabilized. The gyroscope provides essential navigational information to the

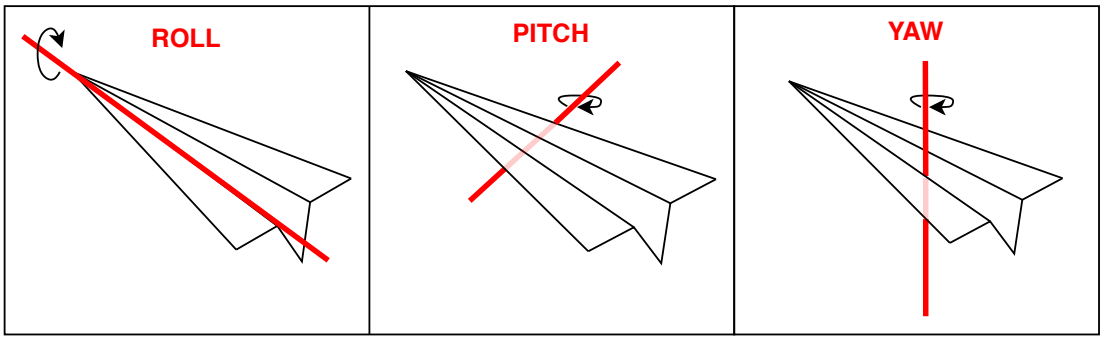


Figure 3.1: Roll Pitch and Yaw angles

central flight control systems [11] providing measurements of rotation rate around 3 axes:

- Roll: rotation around the back-to-back axis.
- Pitch: rotation around the left-to-right axis.
- Yaw: rotation around the vertical axis.

To have a clearer idea of how these rotations are applied to an aerial vehicle, Figure 3.1 helps to visualize them.

3D accelerometer is another crucial sensor with the main objective of measuring the orientation of a drone relative to earth's surface. This device provides the linear accelerations of the vehicle relative to the inertial frame. It works by sensing all the forces applied to the vehicle, including the earth gravity. These tiny electro-mechanical structures can easily interface to electronics, allowing engineers to build some pretty amazing devices on an extremely small space. More information regarding these devices can be seen in [11, 38].

A **barometer** can also be used as it is an air pressure sensor, in a quadrotor it can be used to measure its altitude from the ground. The measurement outcome from a barometer is so sensitive that it is able to detect changes even when the movement is just a couple of centimeters.

The **Inertial Measurement Unit**, commonly referred as the IMU, is an electronic device that measures and reports the UAVs specific force, angular rate and the orientation by using a combination of accelerometers, gyroscopes (as specified before) and sometimes magnetometers to assist calibration against orientation drift. IMU detects changes in rotational attributes like pitch, roll and yaw using one or more gyroscopes.

The **autopilot** system, in short words, is a system that allows the UAV to fly autonomously. The autopilot uses the measures given by all sensors and uses them to do certain tasks, such as object tracking, routes, collision avoidance, landing and take-off, etc. Examples of autopilots open source software stacks include Ardupilot and PX4, where the later plays a big role in the course of this thesis. For instance, PX4's flexibility

provides compatibility both to autopilot software stacks and autopilot hardware implementations, such as Holybro Pixhawk 4 Mini (used in this thesis), CUAV V5+, CUAV V5 nano, etc. [18]

The capability of the UAV knowing its location is crucial when it comes to tasks that are not human controlled. There are quite a lot of systems, the most used when it comes to outdoor utilization is the GPS system, despite the shown deviations it is still a cheap yet effective way to implement a localization system in a UAV. But when it comes to indoor maneuvering, the GPS system becomes inefficient so there's a need to use a more reliable localization system and that's where the testbed with a motion capture system implemented comes in handy.

With the flight controller operating and a working localization system, the quadcopter should now have the necessary components in order for the processor calculate at what speed each motor should spin in order to maintain a steady flight and perform the needed tasks. On the other hand, if there's no localization system and the tasks should be human performed there might be the need to use a normal RC controller capable of operating in the desired directions with a receiver and transmitter.

3.2 Rotation representation

The rotation representation is an important matter, with that said it's important to take into consideration the available options. Throughout the thesis several coordinate systems will be used to represent the orientations, the chosen coordinate system that will be used needs to have into consideration if it is suitable for the object in cause. For example, the *world frame* $\mathcal{W}$ which will be an arbitrary fixed point on the ground, its orientation must be defined so that, for example, the orientation of the *body frame* $\mathcal{B}$,representing the UAV itself, is known based on this world frame.

On top of the rotation representation, there are multiple reference frames. As for ROS, the common references used are **FLU** (X Forward, Y Left, Z Up) or **ENU** (X East, Y North and Z Up). In terms of PX4 it commonly uses whether **FRD** (X Forward, Y Right and Z Down) or **NED** (X North, Y East, Z Down). When integrating with different systems it is very important to make sure that the references are taken into account.

The two most used methods of representing angles are the Euler angles and the quaternion representation. Each of them have their benefits and disadvantages.

3.2.1 Euler angles and rotation matrices

A rotation of a body is originally represented by rotation matrix that can be defined by

$$\mathbf{R} = \begin{bmatrix} \hat{x} & \hat{y} & \hat{z} \end{bmatrix} \in \mathbb{SO}(3)$$

where $\hat{x}, \hat{y}$ and $\hat{z} \in \mathbb{R}^3$ and as each of these are unit vectors we get

$$\|\hat{x}\| = \|\hat{y}\| = \|\hat{z}\| = 1$$

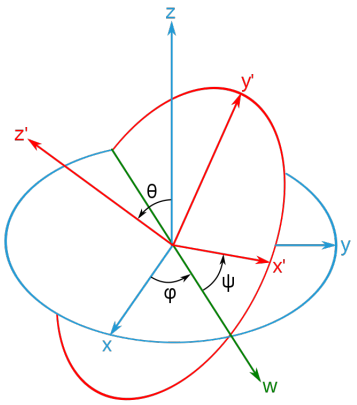


Figure 3.2: Euler Angles [1]

We can now define three rotations, one around each axis, obtaining

- ϕ - Roll (around x-axis)
- θ - Pitch (around y-axis)
- ψ - Yaw (around z-axis)

Defining the elementary rotations around each of these axis, we now obtain

$$R_x(\phi) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\phi) & -\sin(\phi) \\ 0 & \sin(\phi) & \cos(\phi) \end{bmatrix} \tag{3.1}$$

$$R_y(\theta) = \begin{bmatrix} \cos(\theta) & 0 & \sin(\theta) \\ 0 & 1 & 0 \\ -\sin(\theta) & 0 & \cos(\theta) \end{bmatrix} \tag{3.2}$$

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

Knowing that a rotation matrix is formed by three sequential rotations around the primary axes and that the order of the rotations have impact on the final orientation, it's easy to understand that there are a total 12 possible combinations. Six of these combinations rotate around one of the axis twice (XYZ, YXY, ZXZ, XZX, YZY, ZYZ) and these are called the Euler angles. The other group of combinations, where the rotations are done around all three axis, are called Tait-Bryan angles which won't be focus of study.

Euler angles were introduced and explained by Leonhard Euler and are used to describe a rotation of a rigid body, using the ϕ, θ and ψ angles, with respect to a fixed coordinate system [14]. This representation can be visualized in Figure 3.2.

[1] License CC BY 3.0, Adapted from https://commons.wikimedia.org/wiki/File:Eulerangles.svg

The final rotation matrix is then given by the multiplication of the three matrices, while having in consideration the order of the rotation. An example of parametrization, considering a rotation given by Z - X - Y, the rotation matrix obtained from the rigid body to the world is:

$$R_{zxy}(\phi,\theta,\psi) = R_z(\psi)\cdot R_x(\phi)\cdot R_y(\theta) \tag{3.4}$$

3.2.2 Quaternion representation

William Rowan Hamilton introduced quaternions [23] in the 19th century, a more complex way of representing rotations of a body in a 4 dimensional vector space. A quaternion $\mathbf{q} \in \mathbb{H}$, with $|\mathbf{q}| = 1$, can be represented as:

$$\mathbf{q} = q_w + q_x\mathrm{i} + q_y\mathrm{j} + q_z\mathrm{k} \tag{3.5}$$

where i, j and k are complex numbers and $i^2 = j^2 = k^2 = ijk = -1$. This quaternion representation is usually done with computational resources as the mathematics adjacent to this method is complex. Further readings can be seen in [14].

A quaternion can be obtained from Euler angles using

$$\mathbf{q}_{313}(\phi,\theta,\psi) = \begin{bmatrix} q_w \\ q_x \\ q_y \\ q_z \end{bmatrix} = \begin{bmatrix} \cos(\frac{\phi}{2})\cos(\frac{\theta}{2})\cos(\frac{\psi}{2}) - \sin(\frac{\phi}{2})\cos(\frac{\theta}{2})\sin(\frac{\psi}{2}) \\ \cos(\frac{\phi}{2})\cos(\frac{\psi}{2})\sin(\frac{\theta}{2}) + \sin(\frac{\phi}{2})\sin(\frac{\theta}{2})\sin(\frac{\psi}{2}) \\ \cos(\frac{\phi}{2})\sin(\frac{\theta}{2})\sin(\frac{\psi}{2}) - \sin(\frac{\phi}{2})\cos(\frac{\psi}{2})\sin(\frac{\theta}{2}) \\ \cos(\frac{\phi}{2})\cos(\frac{\theta}{2})\sin(\frac{\psi}{2}) + \cos(\frac{\theta}{2})\cos(\frac{\psi}{2})\sin(\frac{\phi}{2}) \end{bmatrix} \tag{3.6}$$

3.2.3 Comparison of methods

In order to have a successful state of the system, the method to represent rotations of the UAV is an important matter and the choice should not be done without any considerations. So, in order to represent the state of our system, both Euler Angles, Quaternions and rotation matrices were considered. Despite the low storage memory needed with using Euler angles, it carries problems with the discontinuities and singularities present in this method, resulting in a gimbal lock.

Despite quaternions have a more complex mathematical calculations, the storage needed for these operations favors this parameterization as for quaternions there's only the need of 4 scalars against 9 for the rotation matrices. Speed is another advantage of quaternions as it is much faster to multiply scalars than matrices. A performance example between quaternion and Euler angles can be seen in [2] where clearly quaternion shows to be better.

In terms of conversion between these three methods, further readings with very detailed information can be seen in [14].

3.3 Modelling

As referred in Chapter 3.2 there will be two coordinate systems used, the body frame of the UAV, $\mathcal{B}$, defined by (x_B, y_B, z_B) and the world frame, $\mathcal{W}$, with the position coordinates (x_W, y_W, z_W). A rotation matrix ${}^{W}R_B$ describes the orientation of frame $\mathcal{B}$ in frame $\mathcal{W}$, which can be parameterized by Z - X - Y Euler angles, where first we rotate about the z_w axis by the yaw angle, ψ, then about the x_W axis by the roll angle, ϕ, and in last about the y_W axis by the pitch angle, θ. This rotation parametrization can be described by

$$ {}^{W}R_B = \begin{bmatrix} c(\psi)c(\theta) - s(\phi)s(\psi)s(\theta) & -c(\phi)s(\psi) & c(\psi)s(\theta) + c(\theta)s(\phi)s(\psi) \\ c(\theta)s(\psi) + c(\psi)s(\phi)s(\theta) & c(\phi)c(\psi) & s(\psi)s(\theta) - c(\psi)c(\theta)s(\phi) \\ -c(\phi)s(\theta) & s(\phi) & c(\phi)c(\theta) \end{bmatrix} \tag{3.7} $$

where c denotes the *cosine* and s is the *sine*.

Furthermore, the angular velocity ω_B with components p, q and r can be calculated using the derivatives of the roll, pitch and yaw angles, given by

$$ \omega_B = \begin{bmatrix} p \\ q \\ r \end{bmatrix} = \begin{bmatrix} \cos(\theta) & 0 & -\cos(\theta) \\ 0 & 1 & \sin(\phi) \\ \sin(\theta) & 0 & \cos(\phi)\cos(\theta) \end{bmatrix} \begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} \tag{3.8} $$

Being $\mathbf{r}$ the vector which represents the center of mass of the UAV and having into account the gravitational force acting $-z_W$ direction, the acceleration is defined by

$$ m\ddot{\mathbf{r}} = \begin{bmatrix} 0 \\ 0 \\ -mg \end{bmatrix} + R \begin{bmatrix} 0 \\ 0 \\ \sum F_i \end{bmatrix} \tag{3.9} $$

Each of the rotors of the quadcopter produces an angular speed ω_i thus it produces a force F_i and a momentum M_i, respectively defined by

$$ F_i = k_F \omega_i^2 \tag{3.10} $$

$$ M_i = k_M \omega_i^2 \tag{3.11} $$

The system constants k_F and k_M depend on the relation between the rotor speed and the respective lift forces.

Being the rotor speeds the system inputs, we can define $\mathbf{u}$, being u_1 the net body force that represents the sum of the forces of the 4 rotors and u_2, u_3, u_4 are the body moments which can be expressed to the rotor speeds by

$$ \mathbf{u} = \begin{bmatrix} k_F & k_F & k_F & k_F \\ 0 & k_F L & 0 & -k_F L \\ -k_F L & 0 & k_F L & 0 \\ k_M & -k_M & k_M & -k_M \end{bmatrix} \begin{bmatrix} \omega_1^2 \\ \omega_2^2 \\ \omega_3^2 \\ \omega_4^2 \end{bmatrix} \tag{3.12} $$

where L is the distance between the centre of the quadcopter and the axis of rotation of the rotors.

The acceleration of the centre of mass can now be defined using Newton's equations being the sum of the gravitational force (3.9) and the net body force u_1

$$m\ddot{\mathbf{r}} = -mg\mathbf{z}_w + u_1\mathbf{z}_B \tag{3.13}$$

Now that we have all the necessary parameters, the state of the system can be defined by x. This state is given by the position, orientation (locally parametrized by a rotation matrix), linear velocity and the angular velocity of the quadcopter, yielding

$$\mathrm{x} = [x, y, z, \phi, \theta, \psi, \dot{x}, \dot{y}, \dot{z}, p, q, r]^T$$

We can now define the system dynamics of the quadcopter given the components of the system state and the control inputs described by the equations:

$$\begin{aligned}
\dot{\mathbf{p}} &= \mathrm{v} \\
\dot{\mathbf{v}} &= -g\mathbf{z}_w + \frac{u_1}{m}\mathbf{z}_B \\
\dot{\omega}_B &= \mathcal{I}^{-1}(-\hat{\omega}_B\mathcal{I}\omega_B + u_r) \\
{}^W\dot{R}_B &= {}^W R_B\hat{\omega}_B
\end{aligned}$$

with gravitational force $g = 9.8ms^{-2}$, moment of inertia $\mathcal{I}$ and $\hat{\omega}_B$ being the skew matrix obtained from ω_B

$$\hat{\omega}_B = \begin{bmatrix} 0 & -r & q \\ r & 0 & -p \\ -q & p & 0 \end{bmatrix} \tag{3.14}$$

After defining these dynamics equations, these dynamics were then implemented in *Simulink*. Figure 3.3 shows this implementation.

3.4 Controlling

The objective of this section is to implement a controller to control along defined trajectories, $\sigma_T = [r_T(t), \psi_T(t)]^T$, similar to the one developed in [34, 35, 50] that is able to hover and move from one point to another, obviously with as much smoothness and precision as possible, having as basis the system dynamics shown in Section 3.3.

It's important to define the position and velocity errors, these can be defined, respectively by:

$$e_p = r - r_T$$

$$e_v = \dot{r} - \dot{r}_T$$

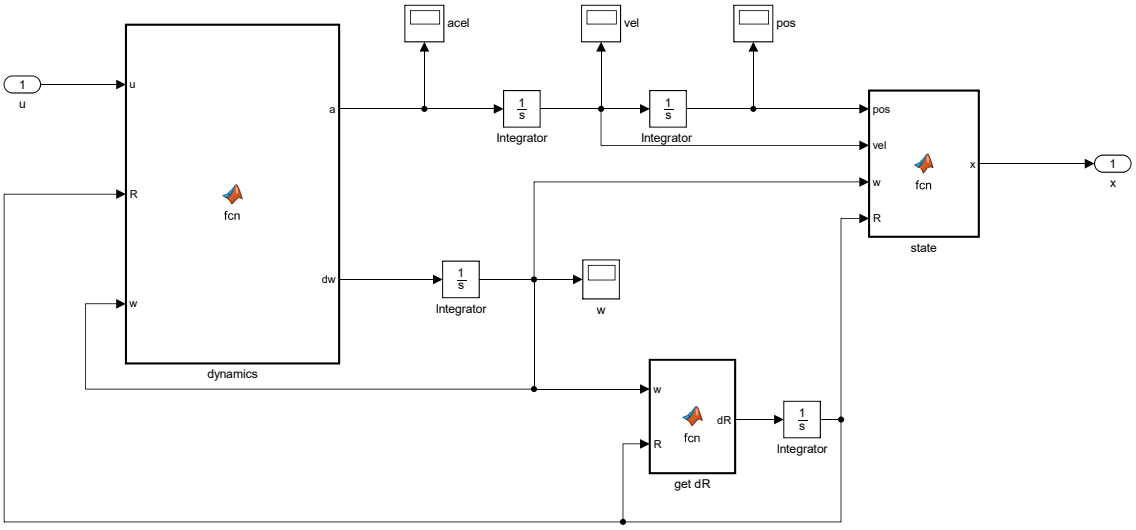


Figure 3.3: Quadcopter dynamics implemented in *Simulink*

and the desired force vector, F_{des}, can be defined by

$$F_{des} = -K_p e_p - K_v e_v + mgz_W + m\ddot{r}_T \tag{3.15}$$

with K_p and K_v being the gain definite matrices of position and velocity, respectively. The controller input u_1 can now be defined as

$$u_1 = F_{des} \cdot z_B \tag{3.16}$$

In order to calculate the other three inputs, the rotation errors must considered. Starting with z_B, we observe that it has the same direction as the force vector F_{des}, so

$$z_{B,des} = \frac{F_{des}}{\|F_{des}\|} \tag{3.17}$$

with $\|F_{des}\| \neq 0$. Consider now an intermediate coordinate frame A being the transformation of the world frame after a few yaw rotations, with the $x_{A,des}$ component calculated by

$$x_{A,des} = [\cos(\psi_{ref}), \sin(\psi_{ref}), 0]^T \tag{3.18}$$

Knowing the specified yaw angle along the trajectory ψ_T, we are in conditions to calculate $x_{B,des}$ and $y_{B,des}$

$$y_{B,des} = \frac{z_{B,des} \times x_{A,des}}{\left\| z_{B,des} \times x_{A,des} \right\|}, \tag{3.19}$$

$$x_{B,des} = y_{B,des} \times z_{B,des} \tag{3.20}$$

The rotation of the body can now be written as

$$^{\mathcal{W}}R_{B,des} = [x_{B,des}, y_{B,des}, z_{B,des}] \tag{3.21}$$

and the rotation error is defined as

$$e_R = \frac{1}{2}(^{W}R_{Bdes}^{T} .^{W} R_B - ^{W} R_B^T .^{W} R_{Bdes})^{\vee} \tag{3.22}$$

where $^{\vee}$ represents the *vee map* (inverse operation of the skew matrix) which takes elements of $\mathbb{SO}(3)$ to $\mathbb{R}^3$. As for the angular velocity, $\omega_{B,des}$, it is needed to manipulate the derivative of the acceleration, $\dot{a}_{ref}$

$$m\dot{a}_{ref} = \dot{u}_1 \cdot z_{B,des} + ^{W} R_{B,des} \cdot \omega_{B,des} \times u_1 \cdot z_{B,des} \tag{3.23}$$

We can project this expression along z_B, and knowing that $\dot{u}_1 = z_B \cdot m\dot{a}$, we can define the vector $\mathbf{h}_\omega$, substituting $\dot{u}_1$ into (3.23):

$$\mathbf{h}_\omega = \omega_{\mathcal{BW}} \times z_{B,des} = \frac{m}{u_1}(\dot{a}_{ref} - (z_{B,des} \cdot \dot{a}_{ref}) z_{B,des}) \tag{3.24}$$

where $\mathbf{h}_\omega$ is the projecting of $\frac{m}{u_1}\dot{a}$ onto the $x_B - y_B$ plane. The first two desired body frame components of the angular velocity can be calculated respectively by

$$p_{des} = -\mathbf{h}_\omega \cdot \mathbf{y}_B$$

$$q_{des} = \mathbf{h}_\omega \cdot \mathbf{x}_B$$

In order to find the third component of the angular velocity, $\mathbf{r}_{des}$, it is needed to analyze the equation relating the derivatives of the Euler angles to the angular velocity given by

$$^{W}R_{B,des} \cdot \omega_{\beta,des} = \begin{bmatrix} x_{C,des} & y_{B,des} & z_W \end{bmatrix} \begin{bmatrix} \dot{\phi}_{des} \\ \dot{\theta}_{des} \\ \dot{\psi}_{des} \end{bmatrix}$$

where $\omega_{\beta,des} = [p_{des}, q_{des}, r_{des}]^T$, therefore

$$^{W}R_{B,des} \begin{bmatrix} p_{des} \\ q_{des} \\ r_{des} \end{bmatrix} = \begin{bmatrix} x_{C,des} & y_{B,des} & z_W \end{bmatrix} \begin{bmatrix} \dot{\phi}_{des} \\ \dot{\theta}_{des} \\ \dot{\psi}_{des} \end{bmatrix} \tag{3.25}$$

The angular velocity error can then be defined by the difference between the actual and desired angular velocities

$$e_\omega = \omega_B - \omega_{B,des} \tag{3.26}$$

Finally, the other 3 control inputs $u_T = [u_{2,des}, u_{3,des}, u_{4,des}]^T$ are calculated by

$$u_T = -K_R e_R - K_\omega e_\omega \tag{3.27}$$

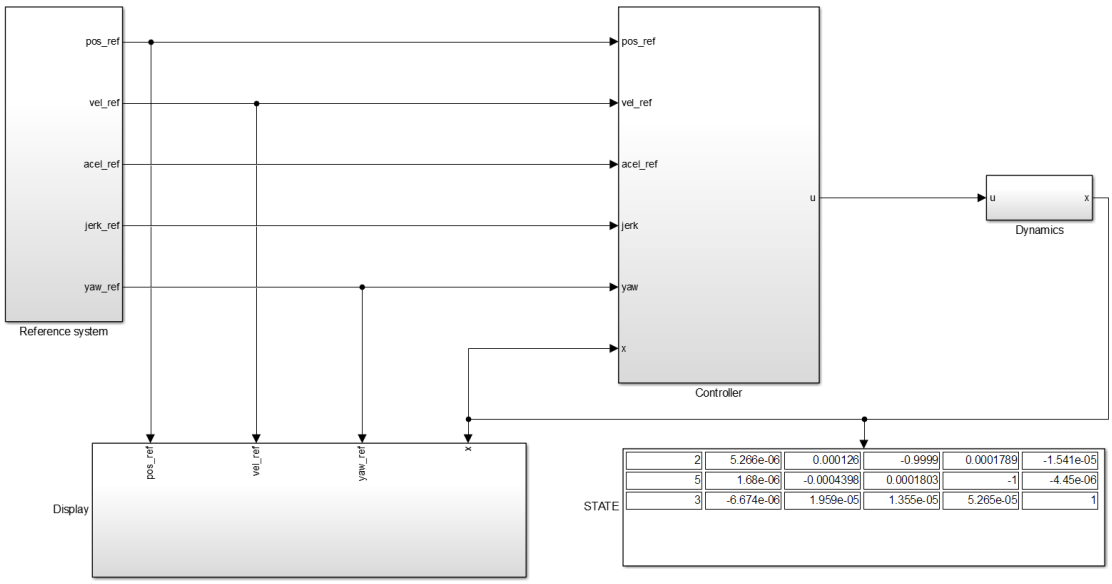


Figure 3.4: Quadcopter controller in *Simulink*

with K_R and K_ω being diagonal gain matrices. With all the equations needed to implement the controller, the conditions to perform some operations as hovering and trajectory tracking should now be met.

The controller implemented in *Simulink* is shown in Figure 3.4 where it can be seen several blocks. Namely a block that is responsible to give the reference of the system, in other words, the final state we want the quadcopter to achieve, the *Controller* block is responsible to achieve this state and is constantly receiving the current state of the quadcopter by the *Dynamics* block. The *Display* block is able to show us a graphical evolution of the position, velocity and yaw states. The *State* block is just a table that shows the final values/state of the quadcopter in a table, used only to make a quick analysis in order to know if the final state of the quadcopter is the one we want to achieve.

In order to have a better understanding of the controller structure, it's important to understand that the outer loop is responsible for the trajectory following that controls the position and the velocity, and notice there is an inner loop that is essentially the attitude controller responsible for the behaviour of the quadcopter in terms of rotations and angular velocities. Figure 3.5 shows a block diagram of this structure, where state $\mathbf{x}$ is calculating at each time step, and based on this state, a new set of inputs $\mathbf{u}$ are calculated.

3.5 Simulation results

Using the *Display* block described from Figure 3.4 it is possible to see the performance of our controller. Firstly, the reference system will start with all parameters at zero, one seconds later, a step in the x-axis, y-axis and z-axis of the position reference are applied and the results can be seen, respectively, in Figure 3.6a, Figure 3.6b and Figure 3.6c.

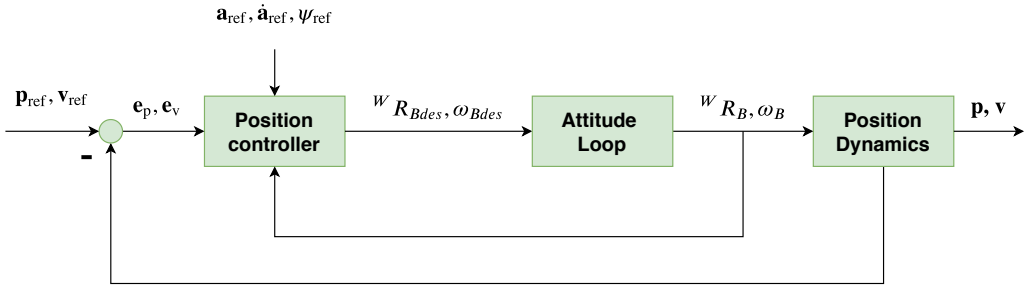
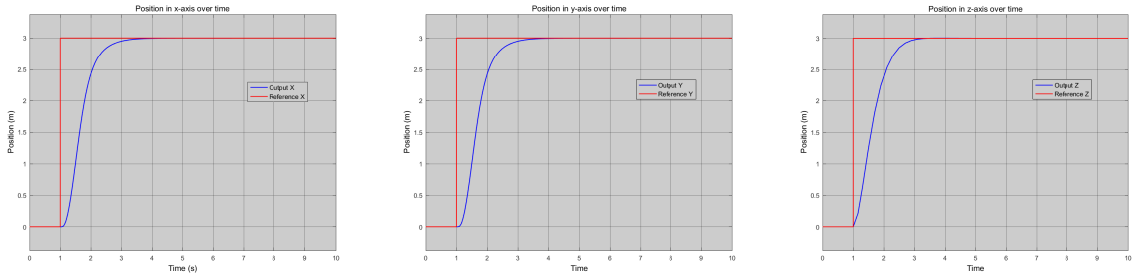


Figure 3.5: Block diagram of the controller



(a) Position over time in x-axis

(b) Position over time in y-axis

(c) Position over time in z-axis

Figure 3.6: Individual step trajectory following of X, Y and Z axis

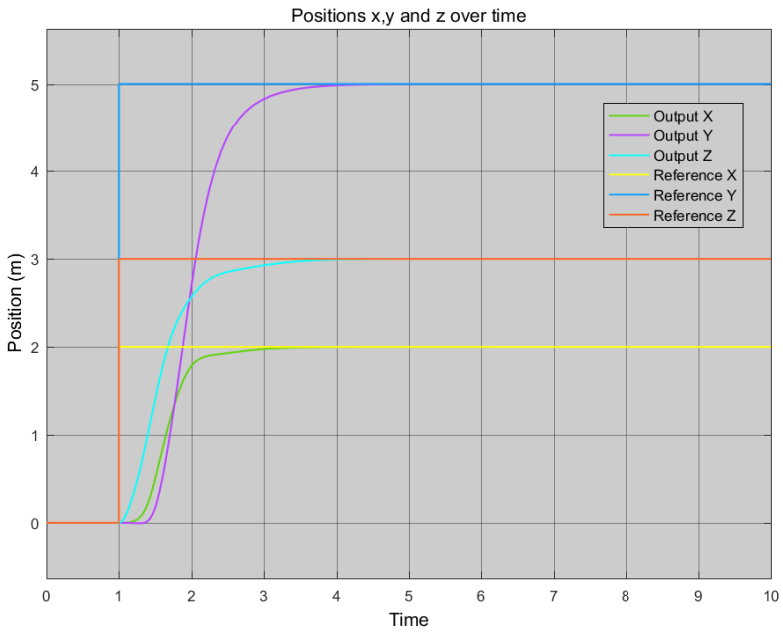


Figure 3.7: Position over time

A trajectory tracking that involves all the three axis can be seen in Figure 3.7.

In terms of achieved performance of this controller, it could be expected to achieve better results in terms of speed, because as it can be seen, it takes around 1-3 seconds

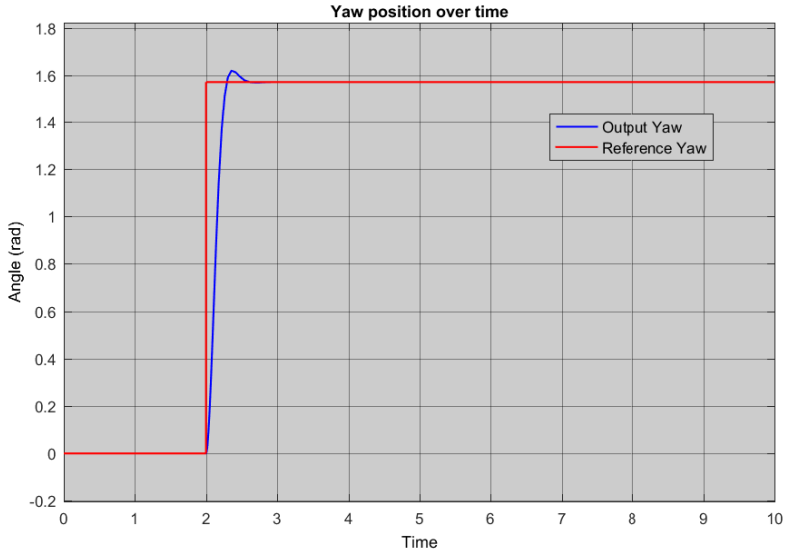


Figure 3.8: Yaw angle over time

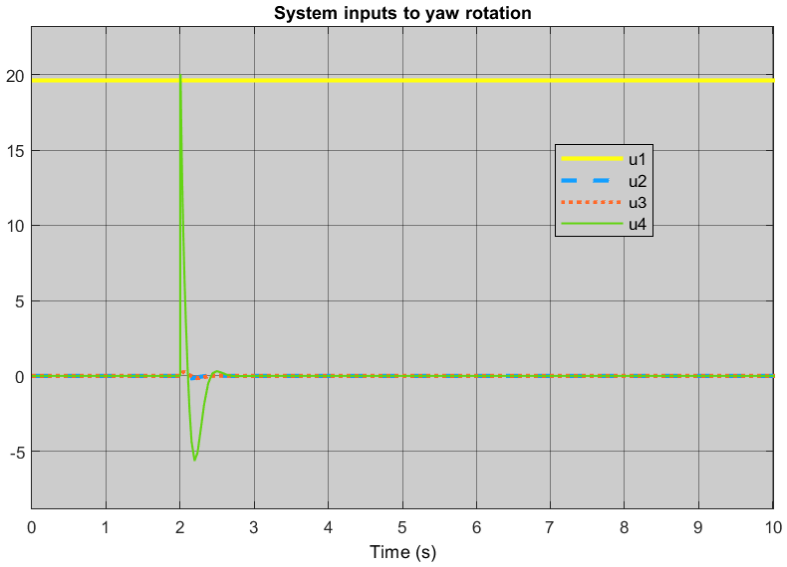


Figure 3.9: System inputs to achieve a yaw rotation

to achieve the desired result, depending of course in the distance between the positions we're attempting to reach. The gains used in the controller play an important role when it comes to performance, it would be easy to achieve faster results with the counterpart of adding overshoot or oscillations. Even though the speed is not the best, the smoothness is pretty good hence the choice of these controller gains.

The performance of the controller in terms of the yaw angle was also tested, these results were obtained and can be seen in Figure 3.8. The input of the system to reach achieve this state can be seen in 3.9.

The performance shown by the controller when it comes to reference following of the yaw angle is pretty fast. In the shown example the reference changes from an angle of

K_p	K_v	K_R	K_ω
$\begin{bmatrix} 15 & 0 & 0 \\ 0 & 15 & 0 \\ 0 & 0 & 15 \end{bmatrix}$	$\begin{bmatrix} 10 & 0 & 0 \\ 0 & 10 & 0 \\ 0 & 0 & 10 \end{bmatrix}$	$\begin{bmatrix} 20 & 0 & 0 \\ 0 & 20 & 0 \\ 0 & 0 & 20 \end{bmatrix}$	$\begin{bmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 2 \end{bmatrix}$

Table 3.1: Controller gains

zero degrees to $\frac{\pi}{2}$ rad (90 degrees) after two seconds, and the quadcopter takes about half a second to stabilize in the reference, showing a small amount of overshoot to reach this value, yet these are pretty acceptable results. The results obtained in these tests are better than we can expect in reality as the system inputs are not properly limited to simulate real rotors, even though they are similar.

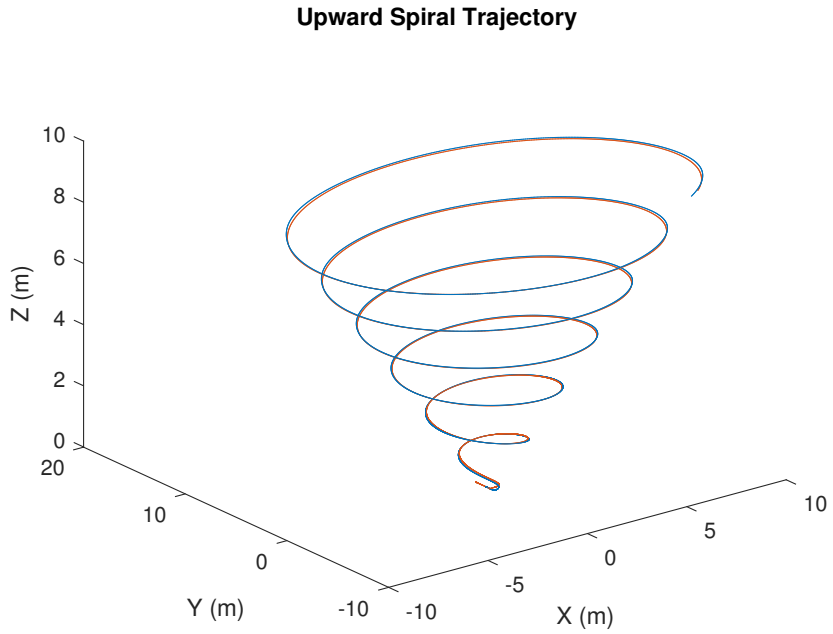
The parameters used to test this controller were the following, the mass of the drone was considered to be $m = 2kg$, the gravitational force $g = 9.81ms^{-1}$ and the inertial matrix of the vehicle was considered to be

$$\mathcal{I} = \begin{bmatrix} 0.06 & 0.003 & 0.0006 \\ 0.003 & 0.05 & 0.002 \\ 0.002 & 0.003 & 0.1 \end{bmatrix}$$

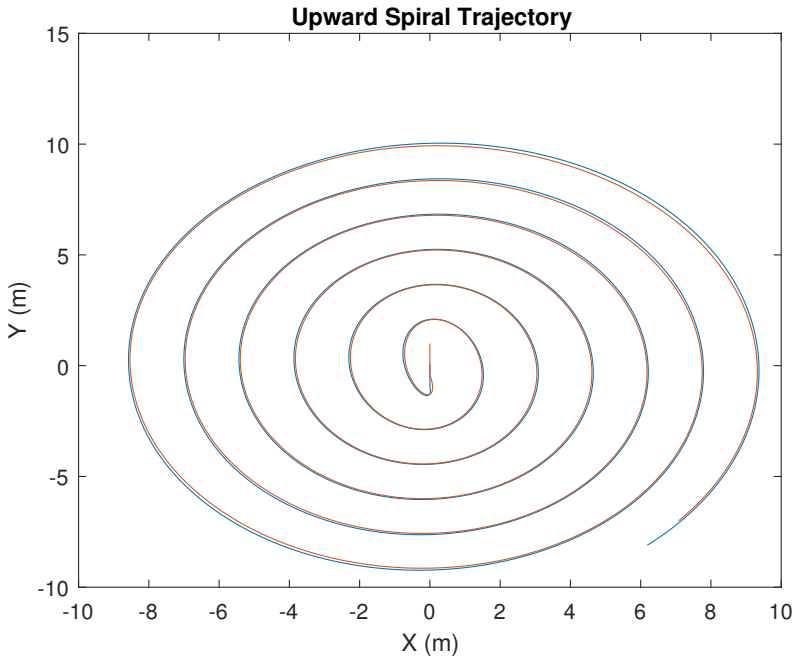
If the results were very important, gain tuning techniques could have been applied to obtain the most adequate outcome results from the controller, but the aim of this controller does not require such effort to maximize speed and efficiency. The used gains to achieve these results, the position gain K_p and velocity gain K_v seen in Equation (3.15) were adjusted in order to obtain a trajectory with as low overshoots as possible. The gains K_R and K_ω were adjusted in order to obtain a response with the minimum oscillations possible. The values of all the used gains in the simulation are present in Table 3.1, these values were obtained after several intense testing while trying to obtain the best achievable results with the controller user.

A more complex trajectory was made in order to test the efficiency of the controller with non-linear trajectories. To comply this task, the gains had to be slightly adjusted, mainly the gains with respect to the position K_p and velocity K_v. The trajectory made is an upward spiral, which is a very good example of a complex trajectory and if the results obtained are coherent with the trajectory we are in conditions to say the performance achieved by the controller are utterly positive, the results obtained can be seen in Figure 3.10.

A point where the reliability of the controller was achieved, and the conditions to move on to more enthusiastic simulations are now met. With that said, further tests resourcing a virtual environment that simulates real conditions can be seen in Chapter 5 where the robustness and efficiency of the controller is put to the test.



(a) 3D Plane



(b) 2D Plane

Figure 3.10: Position tracking with a upward spiral reference trajectory

Chapter 4

Drone testbed

The list of required material, the necessary preparations, the hardware needed, and the software development needed in order to achieve a fully working drone testbed (arena) is a rather long, complicated and time-consuming process. It is precisely regarding this matter that this chapter emphasis. Throughout this chapter it will be explained in detail each of the phases to conclude this process, as well as some hints for someone who would desire to achieve similar results.

Starting from the beginning, to be able to design the drone testbed there's a physical need to have a place to properly mount it, as it's a rather big arena, the Faculty of Science and Technology provided the necessary installations, inside a room of the Department of Electrical and Computer Engineering. The room is big enough to create a decent testbed with dimensions approximately of 2m x 3.5m x 3m (width x length x height).

The testbed is delimited with a net which function is to protect the drones from crashing into unwanted places and most importantly works as a protection for the people who are controlling the quadrotors and whoever might be in the surroundings. Figure 4.1 shows a sketch how the testbed is implemented.

When controlling drones there is a need to know their location, if you were to fly a drone outdoors you could simply use a GPS module that would fairly give you the position of the drone. In this case, the objective is not to perform outdoor tests, but rather consider that all the tests will be conducted indoors, which invalidate the possibility to use the traditional GPS system. Therefore, it becomes necessary to have an alternative way to know the drone's location as accurately and latency-free as possible. Taking into consideration valid solutions for this problem, some of them presented in Chapter 2.2, also taking into care some aspects such as the compatibility with external programs, the ease of use and obviously the ratio between price and performance, the chosen system was the Marvelmind Indoor Navigation System, also mentioned in Chapter 2.2.

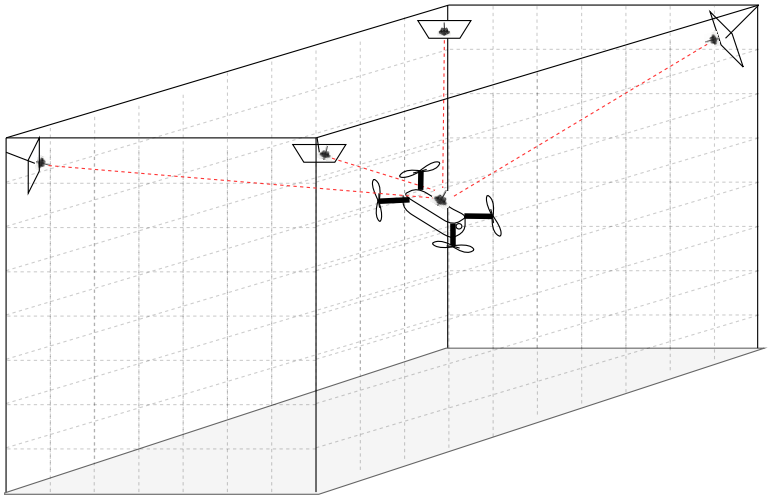


Figure 4.1: Testbed drawing

This system is composed by six elements: five ultrasonic beacons and a modem. In each of the four upper corners of this net surrounding the arena there will be a stationary beacon, mounted on a rotatable support that is roughly pointing to the center of the arena. These stationary beacons will be responsible for giving the drone's position as accurately as possible. The drone being tested will be carrying a moving beacon, which in practice is the same as the other stationary beacons, the only difference is that this one will be configured to be a moving/mobile beacon (commonly refereed as the hedgehog).

All these five beacons will then communicate with the modem, the refereed element of the Marvelmind Indoor Navigation System, this modem will be connected by USB to the computer running Linux. Marvelmind provides a simple software that goes by the name of "Dashboard" which facilitates the configuration of these beacons. It allows to configure a large list of settings such as the update rate (Hz) in which the beacons operate, choose an address for each individual beacon, choose whether a beacon is stationary/mobile and it also offers the possibility to monitor in real-time each of the beacons.

Furthermore, in order to start using Marvelmind, it was needed to set the system up. The steps are not too hard and can be done with enough ease upon carefully reading the instructions to do so. Firstly, there is the need to identify which hardware is being used, which in this case, each beacon is marked as being HW49 433MHz (as it is an European version) and the modem also being the HW49 433MHz. Upon that, the firmware from Marvelmind must be downloaded and flashed through *Dashboard* into each of the electronic devices (the firmware from the modem is distinct from beacon's firmware) by connecting an USB cable from the computer to each individual device. With that done and each of the beacons fixed in the right place, the software detects the position of each of the beacons, creating a submap with the possibility to configure a ton of parameters. Further information regarding the setup of this system can be seen in [45].

Another point worth mentioning is that safety is rule number one when maneuvering

with aerial vehicles, specially with quadcopters as they have blades that can easily cause serious damage. In this case, safety should be taken even more seriously as the goal of this testbed is to test controllers that haven't been prior tested, meaning that the chances of occurring unwanted behaviour from the drone is even higher. Therefore, to minimize the chances of happening any accidents, two cables have been placed on the testbed, and these cables should always be attached to the quadrotor when testing controllers that may have unpredictable behaviours, and one of these should be human handled. One of the cables is attached to the lower part of the drone, limiting the height of the drone so that it can never touch the ceiling. The other cable is the one that must be human handled, the cable is attached to the upper part of the drone's body and prevents the drone from crashing into the ground. If the operator sees that the drone is not doing what it's supposed to do and feels that something can go wrong, he must immediately push the cable. When the cable is pushed, the drone will be stuck between the two cables, limiting its movement and guaranteeing that the drone does not crash, and most importantly that it does not cause damage to any humans.

Some real images with regards to the developed testbed are attached in Appendix B.

4.1 Analysis of Marvelmind indoor navigation system

Marvelmind Indoor Navigation System, acting as an "indoor GPS system" will play a big role, not only in the performance of this drone testbed, but also in the course of this thesis. As mentioned before, the objective of this system is to provide the location of the drone(s) inside the arena, as for that, a vast sequence of real life tests will be conducted in this chapter in order to have an idea of how accurate and how fast the data retrieved by this system is.

In first place, an exhaustive test will be done in order to verify its real-life accuracy, remembering that the accuracy announced by the manufacturer is roughly 2cm. What is intended to determine is if the values announced are indeed similar when comparing to real life scenarios. Furthermore, the main goal with the intensive tests that will be done throughout this chapter are to understand the impacts some of the parameters have, in order to achieve the most optimal results from this system in terms of position's accuracy, while at the same time maintaining the latency as low as possible. These are two important aspects to have into consideration when flying with drones, specially in such small places.

As stated before, the accuracy of this system will be measured. The way this test will be conducted is by placing one of the beacons configured as a mobile beacon (hedgehog) in a certain position (x_0, y_0, z_0) and collect the position given by the system for a certain amount of time, collecting the data which contains the registered position during all the time span, and then properly analyze in order to retain the necessary conclusions from it. These tests will be done with different update rates (Hz), yet maintaining the

same environmental conditions throughout the realization of each individual tests. It's important to mention that the option to allow movement filtering was turned off.

In order to properly integrate Marvelmind with the work done in previous chapters, there was a need to feed ROS with the measured values from Marvelmind, and for that Marvelmind provides the necessary documentation with some code examples of possible ways on how to achieve the desired goal.These code examples are basically ROS nodes, each with a specific objective [47]. It was based on these examples that a ROS node was created in order to read the values from Marvelmind and then publish in a ROS topic. This node, named *hedge_receive_pos* is responsible for publishing a ROS message of type *geometry_msgs/Point* which includes the position in X, Y and Z axis in a topic named *drone_pos*. Despite the possibility of retrieving more data from Marvelmind, at this point the only need is to get the position from the beacon, hence the choice of this message type as it is enough to store the three position coordinates.

Having the data published to a ROS topic in real-time, it becomes easy to record the data from this topic, using for that, ROS bags. Upon that, the data can be read through MATLAB and subsequently plotted and analyzed.

The resulting output from these experiments, while varying the update rate of the system from 2 Hz, 8Hz and 16+Hz (maximum update rate) can be seen in Figure 4.2, all of these tests have roughly around ten or more minutes of duration.

Taking a quick glance at the results shown in Figure 4.2, it's rather obvious that the results are not as good as expected. Unwanted positions are retrieved quite often as there are *spikes* in the graphs, meaning that the position that Marvelmind outputs is quite wrong in those points, which could mislead the controller and cause undesired behaviour. It's worth mentioning that the perfect results would be three horizontal lines in each of the graphs since the object is stopped, hence the position should be linearly the same throughout the time span. It's also obvious that the results obtained with the higher update rate (16+Hz) provides less wrong position coordinates.

One thing that was interesting to check was that if the average of the position would change a lot throughout the time, basically if the signal would diverge or summing up errors during the time span of the test, which fortunately does not happen.

In order to properly take a look at the results without these *diracs*, a filtration on these results were done in order to calculate the accuracy with each of the update rates presented. The filtration was done by calculating the average of each of the axis (x, y, z) and also by calculating the standard deviation σ, which can be calculated using the following equation

$$\sigma = \sqrt{\frac{\sum_{i=1}^{n}(x_i - \overline{x})^2}{n-1}} \tag{4.1}$$

Then, in order to take only points that are really far from the average μ, the standard deviation σ was used to eliminate points that are not within the 99.7% $(\mu + 3\sigma)$ of the data. Upon this filtration, the results can be seen in Figure 4.3.

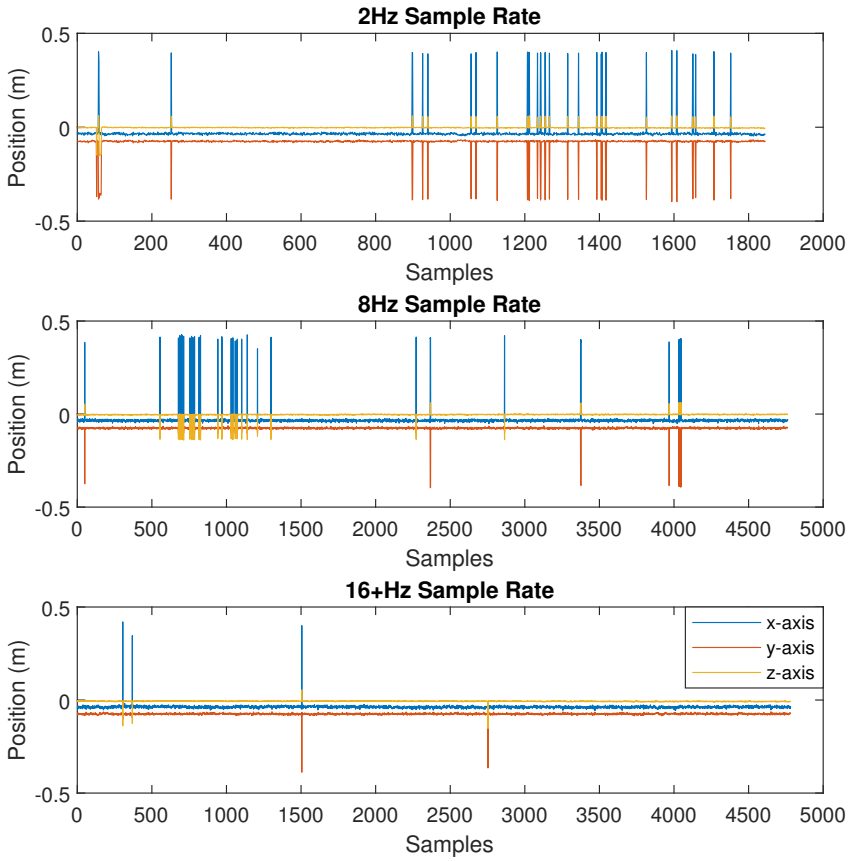


Figure 4.2: Output with different update rates (Raw data)

Regarding Figure 4.3 it is clear that the results are way clearer than Figure 4.2 and it becomes easier to analyze. It is obvious that the position in z-axis is way more precise than the other two axis, but let's take a look at the exact results obtained.

The method presented implies that the average of the whole signal is known which invalidates this method to be used in real time, but in this case, as the objective was only to analyze the data retrieved, it's a valid method to have an idea of the consistency of the results.

A way to filter the data in real time would be really helpful as the previous method can not do that job as mentioned. It is possible to use a median filter, commonly used with real-time data, as well as photography noise reduction and sound noise filtering, as it's a really simple, effective and has a low delay component which is important in this case. It's worth mentioning that every real-time filter imposes a certain delay in the received signal.

To use this filter, firstly an input window must be chosen, which is essentially the number that will be analyzed in each operation, noting that the greater this number is the greater the delay will be. Given this window size, the selected values will be sorted

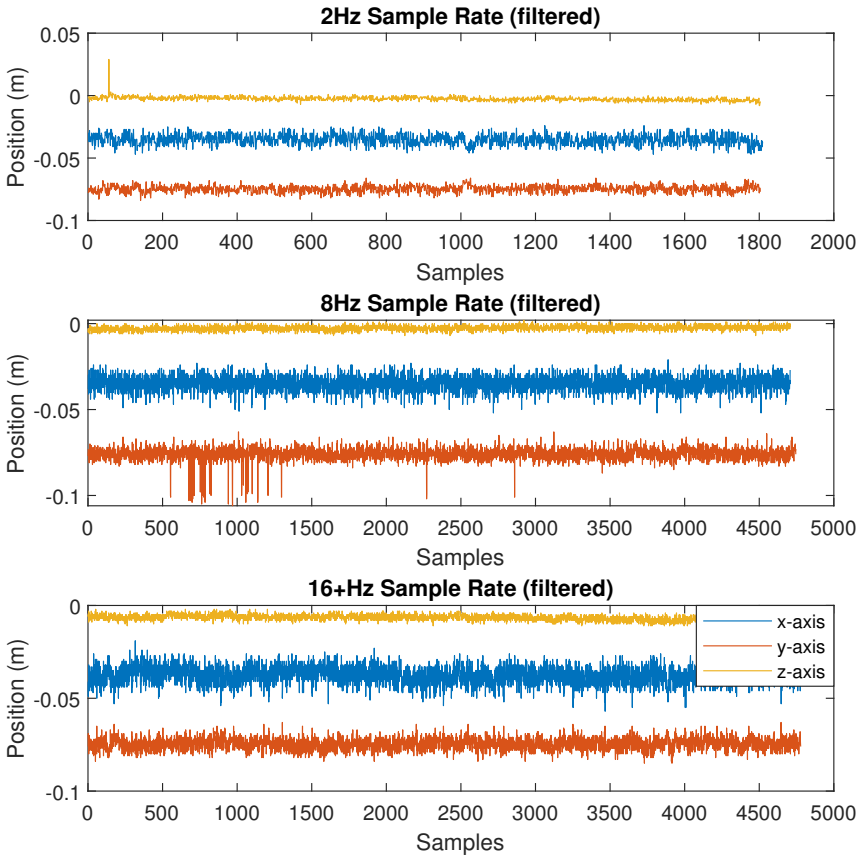


Figure 4.3: Output with different update rates (Filtered)

in ascending order and the median is calculated, this way the values that have bigger deviations will be removed, thus the quality of the signal is improved.

The choice of the input window, after analyzing a couple of options, the one that seems to achieve the desired objective without adding a lot of delay to the positioning signal is an input window of 5. This means that for every calculation it takes into consideration two past values, two future values and the present value. Saying this it becomes obvious that the delay introduced between the retrieved position by Marvelmind and actually using these values is 2 samples, which does not have a notorious impact in the control of the drone. An example of this median filter with an input window of 5 can be seen in Figure 4.4.

Taking a closer look at the introduced delay, for example, in a system of 16Hz, the duration between each sample is given by

$$T_{sample} = \frac{1s}{16Hz} = 62.5ms$$

hence, being the mentioned sample delay of two, we get

$$T_{delay} = T_{sample} \cdot 2 = 125ms$$

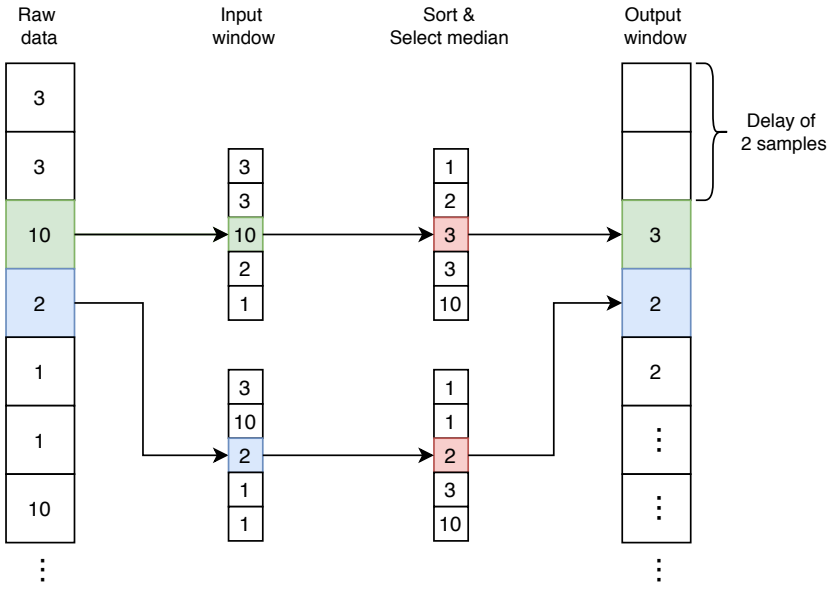


Figure 4.4: Median filter with input window of 5

which can be seen as a tolerable added delay to the system.

The results obtained using this method can be seen in Figure 4.5. It can be observed that the signal is smoothed and the unwanted values that were discussed beforehand have disappeared, except from the spikes seen in 2Hz rate (y-axis and z-axis). In conclusion, this method has proven to be quite useful in this real scenario.

The same results could have been achieved without adding delay if the method in use would be a rolling median filter that, instead of using future values, it would just use present and past values. The results achieved would be the same as shown, but in practice it would be much better since it would not make calculations on past time horizon depending on the chosen input window, resulting in a considerable reduction of the delay.

Another way to avoid unwanted values is by using the filtering method provided by Marvelmind's software itself, as shown in Figure 4.6. The results obtained were using an input window of 5, the same value as used before in Figure 4.5.

A first glance at Figure 4.6 shows that the results obtained using a sample rate of 2Hz and 8Hz are clearly worse when comparing with the update sample of 16+Hz, as this last one shows an incredible accuracy while at the same time avoiding the spikes seen in the other results.

As a conclusion to all these tests, it remains the decision of choosing the best possible settings with the final objective of having the most accurate system, while maintaining the minimum delay possible. To answer this question, we need an analytic method to evaluate each of the results obtained in all the tests. The chosen method to do the needed analysis was the mean deviation (also referred as mean absolute deviation). This is a very simple and effective method to analyze the accuracy of a signal, and it can be used on

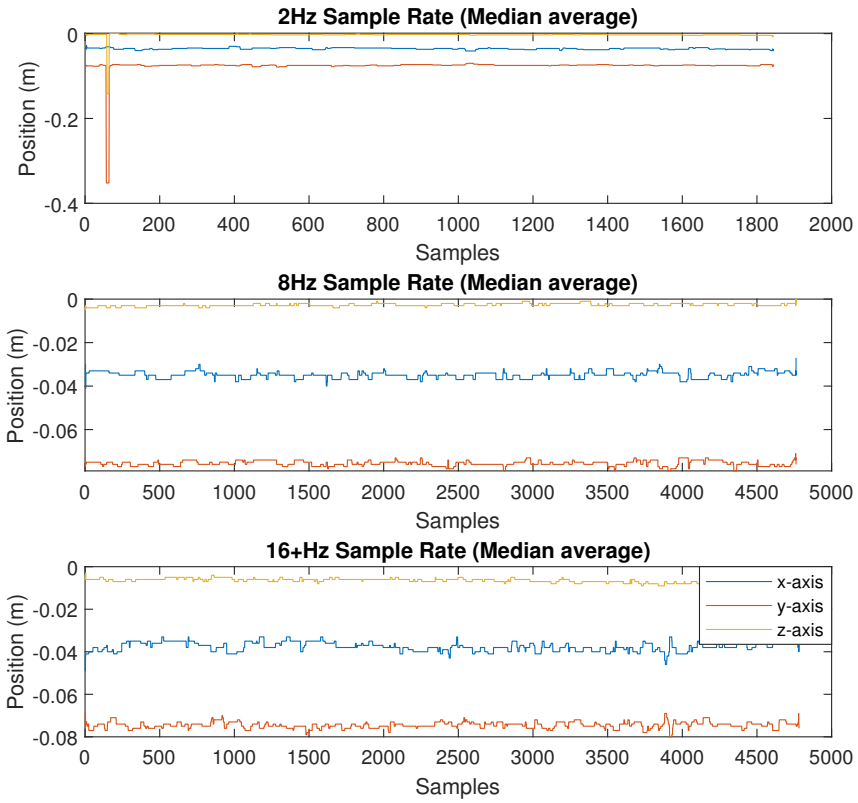


Figure 4.5: Output with different update rates (Median filter)

MATLAB by simply using the function *mad* and the formula which allows to calculate it is given by:

$$MD = \frac{1}{N}\sum_{i=1}^{N}|x_i - \overline{x}| \tag{4.2}$$

where N is the number of total samples and $\overline{x}$ is the mean of the whole signal.

Firstly, a calculation of the mean deviation for each of the signals (each axis of each update rate) was made with the internal filter of Marvelmind turned off, referring to Figures 4.2, 4.3 and 4.5. Table 4.1 helps to have a numeric evaluation of the previous figures, summarizing all the mean deviations, expressed in centimeters (cm). From this table we can see that the best results achieved are using the median filter, and using the higher update rate of 16+Hz.

Afterwards, Marvelmind's filtering option was turned on. Table 4.2 shows us the results of the mean average of each of the axis, also expressed in centimeters (cm) and in this table we can see that the best results were given by the higher sample rate of 16+Hz.

Given these outcomes, the choice of how the position given by the system will be filtered to avoid unwanted behaviours goes to the filter provided by Marvelmind as it provides slightly better results as the ones provided by the median filter applied *manually*,

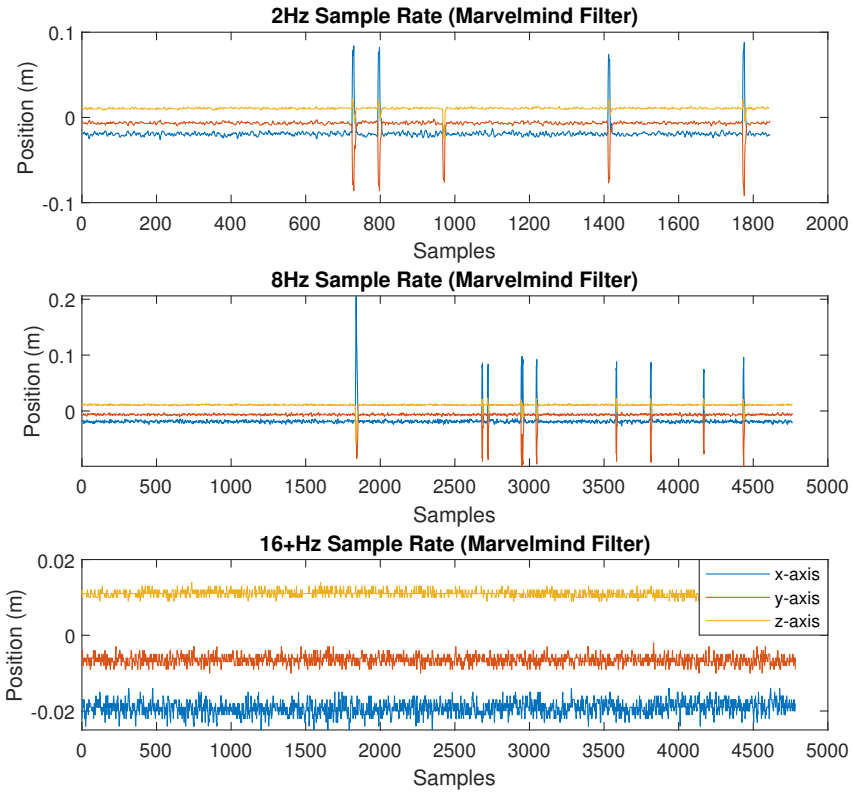


Figure 4.6: Output with different update rates (Marvelmind filter)

Marvelmind's filter turned off									
	Raw Data			Filtered (Standard Deviation)			Filtered (Median Filter)		
Rate	**2Hz**	**8Hz**	**16+Hz**	**2Hz**	**8Hz**	**16+Hz**	**2Hz**	**8Hz**	**16+Hz**
X	1.61	1.05	0.41	0.34	0.34	0.38	0.21	0.19	**0.18**
Y	1.36	0.39	0.27	0.23	0.25	0.25	0.26	0.17	**0.12**
Z	0.29	0.25	0.13	0.12	0.11	0.13	0.11	0.10	**0.06**

Table 4.1: Mean deviations of each set of signal with Marvelmind's filter off

Marvelmind's filter turned on			
	2Hz	**8Hz**	**16+Hz**
X	0.29	0.34	**0.14**
Y	0.25	0.24	**0.10**
Z	0.09	0.10	**0.06**

Table 4.2: Mean deviations of each set of signals with Marvelmind's filter on

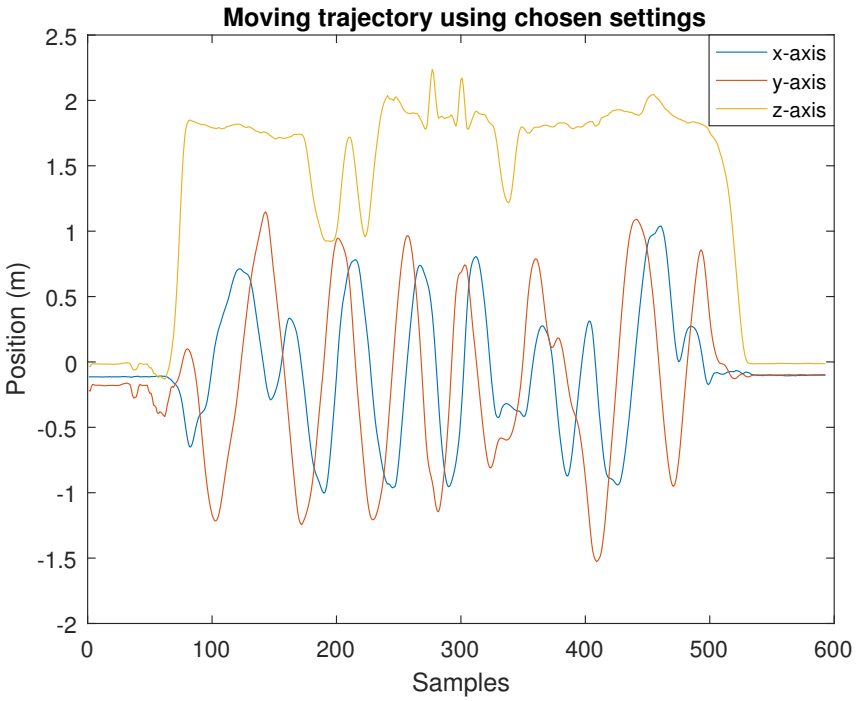


Figure 4.7: Moving trajectory using optimal settings

adding that it also saves effort due to its ease of use. The choise of the sample rate to use is also clear, as it's easy to analyze that the best outcome is given by the higher rate, which in this case is given by the option of 16+Hz, which is also good because the higher the sample rate the lower the latency.

Finalizing the accuracy experiments, a test was made while moving the hedgehog resulting in a rather random trajectory, although it's very interesting to reach the conclusion that no undesired results are shown in Figure 4.7, keeping the trajectory without oscillations and maintaining a smooth trajectory, while at the same time preventing the existence of any kind of *spikes* as seen in previous tests. This figure allows us to validate the smoothness of the position fetch via Marvelmind upon the proper filtration.

Now that the testbed has been properly tested and the results that can be expected are known, both in terms of precision and latency of the position gathered by the system while at the same time maintaining the needed security measures, we can say that the necessary conditions to begin making some more interesting tests are gathered.

In terms of delay, a more accurate measure will be shown ahead in Chapter 5, as it's an important value to know when controlling drones. The autopilot must know the difference between the real measurements that it is retrieving from the IMU and the data that it is receiving from external sources in order to properly estimate all the states of the quadrotor.

4.2 Testbed and autopilot coordinate systems calibration

So that it was possible to perform experimental trials in the testbed as shown in Chapter 5, the PX4 autopilot was gathered, and in order to achieve good results coming from the Extended Kalman Filter (EKF2) used for the state estimation of the quadrotor throughout autonomous flight missions inside the developed testbed, it is completely important that the coordinate system used in the testbed and in the autopilot is the same, so that the position received in the PX4 autopilot is the one expected.

Before the calibration, it was clearly noticeable that the coordinate system was not the same, because while maintaining the direction of the movement and varying the yaw angle of the autopilot module, different results were obtained. These results clearly led to understand that the coordinate system had to be adjusted so that we can avoid incoherences in the position between the systems that certainly would cause unwanted behaviour and increase the chances of crashing the drone.

As it can be seen in [51], the autopilot is expecting to receive the coordinates in NED system (X North, Y East, Z Down). The steps that were done to proceed with the coordination alignment were the following

- Establish a position (in the middle of the testbed) which coordinates would be the origin of the referential with $x = 0, y = 0, z = 0$ and mark the position on the ground so it's a visually known position
- Place the drone in this exact position with the hedgehog beacon attached to it
- Align the x-axis from Marvelmind with the magnetic north pole as accurately as possible making the needed rotations on the z-axis, which in our case was 35 degrees.
- Make the needed rotation around x-axis so that the z-axis is facing down - obtaining the NED coordinate system, in our case as the z-axis was facing up, the rotation was 180 degrees.
- Move Marvelmind's submap so that the established origin position would be the desired one and adjust the height so that the current position is the ground ($z = 0$).
- Move the quadrotor around each axis and visualize the movements in QGC whilst at the same time move its yaw angle to make sure everything is calibrated.

During these calibrations it was possible to see that when trying to align the quadrotor with the north magnetic (where yaw angle is equal to zero degrees) the results were diverging more than desired hence the need to re-calibrate the integrated compass, which can be done easily resourcing QGC. Upon this calibration the results obtained were quite accurate having small divergences, always lower than 5 degrees, which is pretty much an acceptable result.

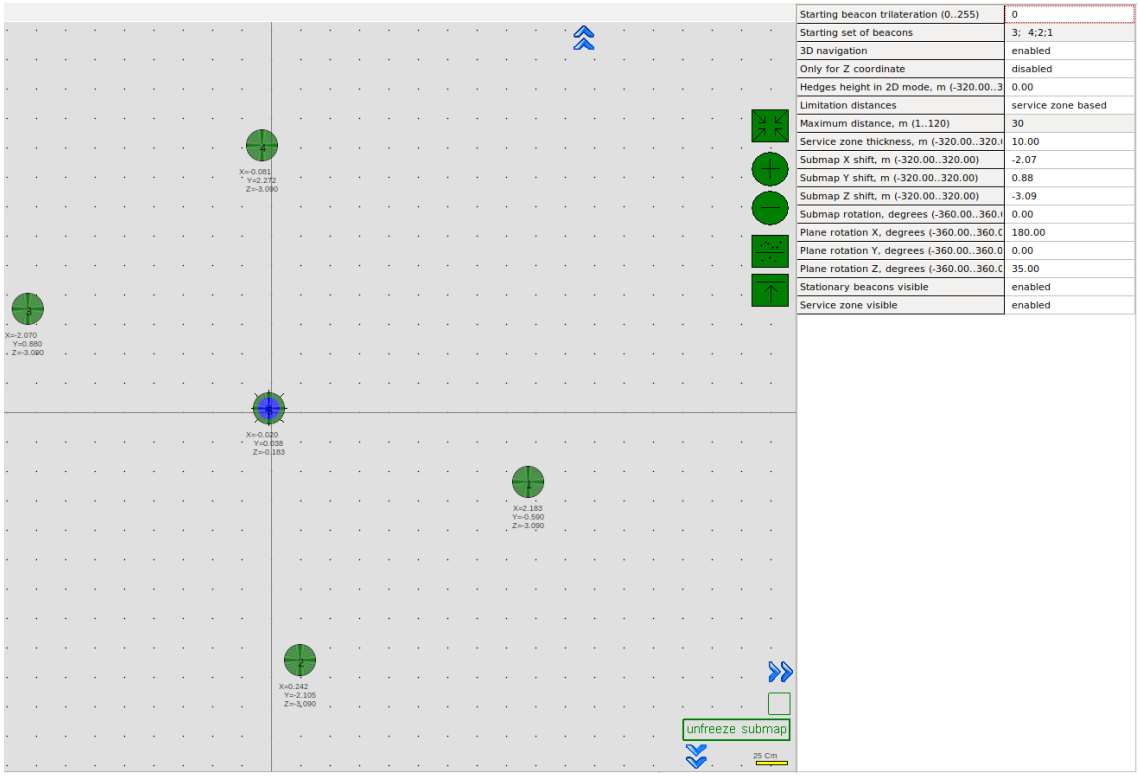


Figure 4.8: Marvelmind submap configuration

After all these important adjustments we achieve a coherent coordinate system that allows a safe autonomous flight inside the testbed. The final configurations achieved in *Dashboard* software from the Marvelmind side can be seen in Figure 4.8, where the rotations and shifts in all axis done in the submap can be seen.

Chapter 5

Drone control

This chapter will emphasis on joining the efforts of the previous chapters, namely to put to work both the designed drone controller in Chapter 3 and the drone testbed developed in Chapter 4, with the final objective of having a fully working drone controlled inside the testbed.

To give a bit of context, in first place, virtual simulations will be done using the Gazebo environment and, afterwards when successful results can be achieved from these, the next step is to move on to real experimental trials inside the testbed. This second objective will be achieved using the Pixhawk 4 Mini (with the software stack PX4) autopilot and the airframe used will be the Holybro QAV250. Both these components are part of a kit, which assembly instruction and respective configuration can be seen in [25]. A real picture, regarding this quadrotor is shown in Appendix B.

5.1 Virtual simulations

Upon multiple tests and achieving the desired results as shown in Figure 3.10 from Chapter 3, the next step is to achieve similar results but this time using a 3D simulator and for this matter, the chosen simulator is Gazebo provided by ROS and for this matter there's the need to use not only Simulink as before, but also Robot Operating System (ROS) and PX4 Firmware.

Robot Operating System (ROS) is in its essence a framework to write robot software. It provides a collection of tools, libraries, and conventions that allow to simplify the task of creating complex and robust robot behaviour across a wide variety of robotic platforms. In this particular case, it will allow to send inputs to the drone. Further reading regarding this software can be seen in [44].

One of the packages provided by ROS is an interface to the simulation environment

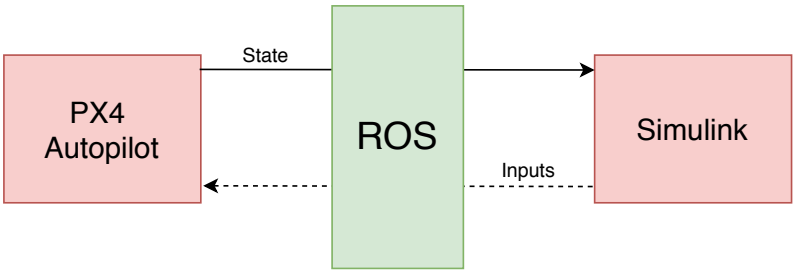


Figure 5.1: Interaction of Simulink with ROS and PX4

Gazebo, which delivers a well-designed virtual environment that simulates a real life test with the necessary conditions allowing to quickly test algorithms, controllers or even train AI systems. It provides a robust physics engine with multiple options available to simulate not only indoors but also outdoors, making the range of possibilities almost unlimited. In-depth information regarding this simulator can be seen in [20].

PX4 is used in a wide range of use-cases, from consumer drones to industrial applications and provides open source flight control software for drones and other unmanned vehicles such as under water vehicles and boats. In addition, PX4 will provide us a simulator where it's possible to test and improve the controller. It's worth mentioning that PX4 provides a handful of helpful tutorials, including a tutorial that shows how ROS can interact with PX4 using a MAVROS node. Further information regarding PX4 and its vast available capabilities can be explored in [40].

In order to use the firmware with all the functionalities provided by PX4, its source code can be cloned from their official repository in [41] and then it must be built. Furthermore, the complete steps for the initial setup can be seen in [7].

Figure 5.1 gives an idea on how the interaction between the different used software is done. In terms of connection between ROS and Simulink it's rather simple, as Simulink provides an easy and intuitive tool to test and establish the communication with ROS. It also provides a way to subscribe and publish topics to ROS which facilitates things, as receiving the current state of the drone and sending the desired inputs to the drone. Essentially, we can observe that ROS operates as a communication layer between Simulink which is operating the controller and PX4 Autopilot that is controlling the drone itself.

At this time, there's the need to use two physical computers, as the computational power needed to execute both software and running two operating systems at the same time is quite demanding.

Robotic Operating System and PX4 are both running in Linux environment (Ubuntu 18.04) in one computer. As for the MATLAB/Simulink software, it is running in the second computer using Windows operating system, shown to be much more efficient for real-time operation than its versions for Linux and Mac. This way, it's possible to avoid overload of a single computer while at the same time avoiding the usage of a virtual machine as this would be a solution to avoid the usage of two computers due to the need of using two separate operating systems.

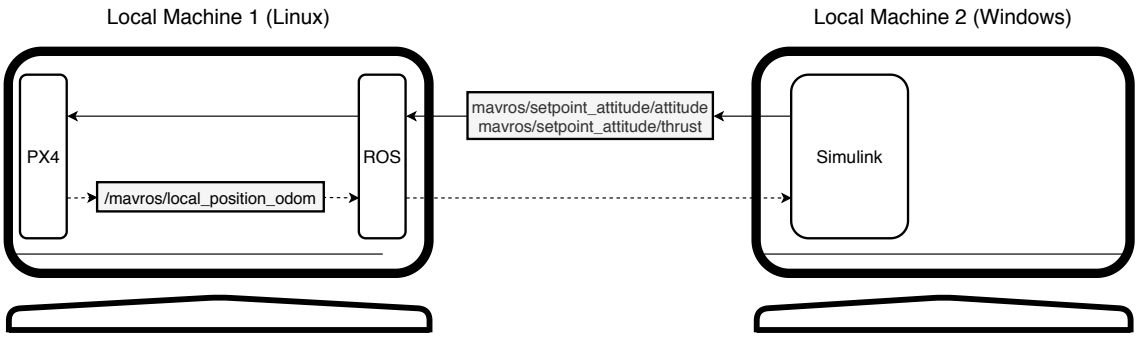


Figure 5.2: Interaction between the two local machines

With regards to communications between both machines, it's crucial to make sure both are operating in the same local network as the establishment of the communication of ROS and Simulink must be inside a local network. It's worth mentioning that a problem was faced when adopting this method, namely that ROS was not accepting messages coming from Simulink, the issue was solved via *command line* using the command *'sudo ufw allow from [IP of local machine]'*.

Figure 5.2 shows the mentioned interaction between the two local machines, with the most important MAVROS nodes needed to control the quadrotor.

There were some initial time-consuming requirements before starting to use all these software together and being able to send any type of inputs to the drone, such as setting up the whole environment and making sure that the communications between them were correctly being sent and received, as well as making sure that they were fast enough to properly fill the requirements to control the drone in real-time.

PX4 provides a list of drones in their simulator, the chosen drone to keep with the simulations is the 3DR Iris, it's a rather small, light and agile drone that allows to make fast maneuvering in small places which is precisely what's needed throughout the development of this research.

In order to proceed with the simulations, there was the need to activate the *offboard mode* to force the vehicle to obey the received inputs and allow controlling the vehicle's movement and attitude with the usage of a set of provided MAVLink messages. To achieve this requirement a ROS node *"offb_node"* was created with the responsibility of activating the offboard mode using the service *mavros/set_mode* and arming the vehicle (turning on the motors) using the service *mavros/cmd/arming*. Later, in this node there will also be created topics responsible for the data exchange between Simulink and PX4. Figure 5.3 gives a strong idea on the looks of the mentioned drone.

At this time, the vehicle is ready to receive inputs by the user and act accordingly, Figure 5.4 shows the Gazebo environment in which the tests are being conducted throughout these virtual simulations, only a couple of elements were added to the environment in order to maintain it simple and avoid spending unnecessary computational resources in

[1] Published by Adafruit Industries (https://www.flickr.com/photos/adafruit/10578616775/)

Figure 5.3: 3DR Iris [1]

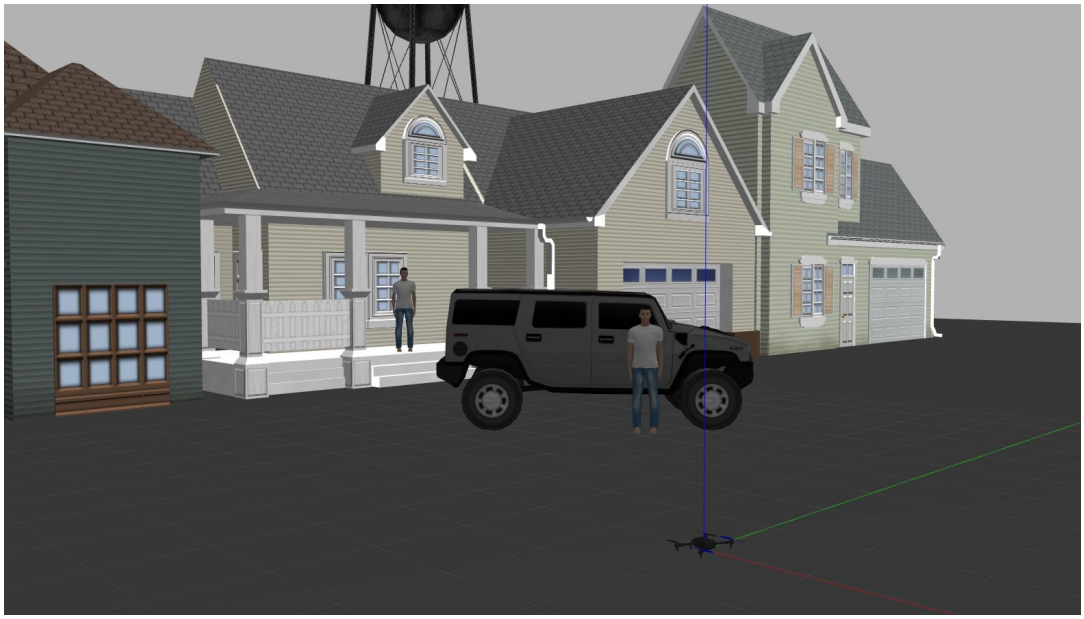


Figure 5.4: Gazebo environment

K_p	K_v	K_R	K_ω
$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$	$\begin{bmatrix} 1.5 & 0 & 0 \\ 0 & 1.5 & 0 \\ 0 & 0 & 1.5 \end{bmatrix}$	$\begin{bmatrix} 6 & 0 & 0 \\ 0 & 6 & 0 \\ 0 & 0 & 6 \end{bmatrix}$	$\begin{bmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 2 \end{bmatrix}$

Table 5.1: Controller gains for 3DR Iris

the course of the simulations, but at the same time adding some reality to it.

For this chapter, there was the need to adjust some parameters when comparing to the values used in Chapter 3. For this simulation, we were able to use the parameters provided by PX4, with this information the mass parameter was now readjusted to be

$$m = 1.5kg$$

and the gains used now were readjusted to completely different values than ones used before (Table 3.1), when comparing to new controller gains that can be seen in Table 5.1.

Regarding the inertial matrix used, it also had to be readjusted in order to respect the dynamics of the 3DR Iris quadrotor, the values used were retrieved from the file that has all the parameters regarding this aircraft, the resulting matrix is given by

$$\mathcal{I} = \begin{bmatrix} 0.0291 & 0 & 0 \\ 0 & 0.0291 & 0 \\ 0 & 0 & 0.055 \end{bmatrix} \tag{5.1}$$

In an early stage of these simulations, the initial objective was only to send position coordinates, making sure that the communications were being well executed. In this simulation, the controller in use is fully provided by PX4 which is not the desired objective.

The results are very smooth as expected due to the controller in use is a well developed and fully-tested one and it's being used in a perfect environment, free of obstacles and other disturbances such as wind. The results of this test can be seen in Figure 5.5.

After this simulation, a more interesting simulation was done, this time instead of controlling by position, the 3DR Iris was controlled by attitude, using the developed controller throughout Chapter 3.4. To control the drone by attitude there was the need to use two topics, *mavros/setpoint_attitude/attitude* that requires as input a quaternion being the rotation matrix and *mavros/setpoint_attitude/thrust* that accepts the value of thrust applied to the drone with value comprehended in the interval [0;1]. These two topics are being published by the *"offb_node"* node referred before.

The thrust applied in the drone was extracted from Equation (3.16) and the required quaternion is a conversion from the rotation matrix that comes from Equation (3.21) into a quaternion. It's worth to mention that the designed controller is not being fully used as the objective of the controller is to control only by torques, however at this moment a

[2]Note that the scale of time was retrieved from Simulink, which is faster than Gazebo environment, resulting in discordance in time

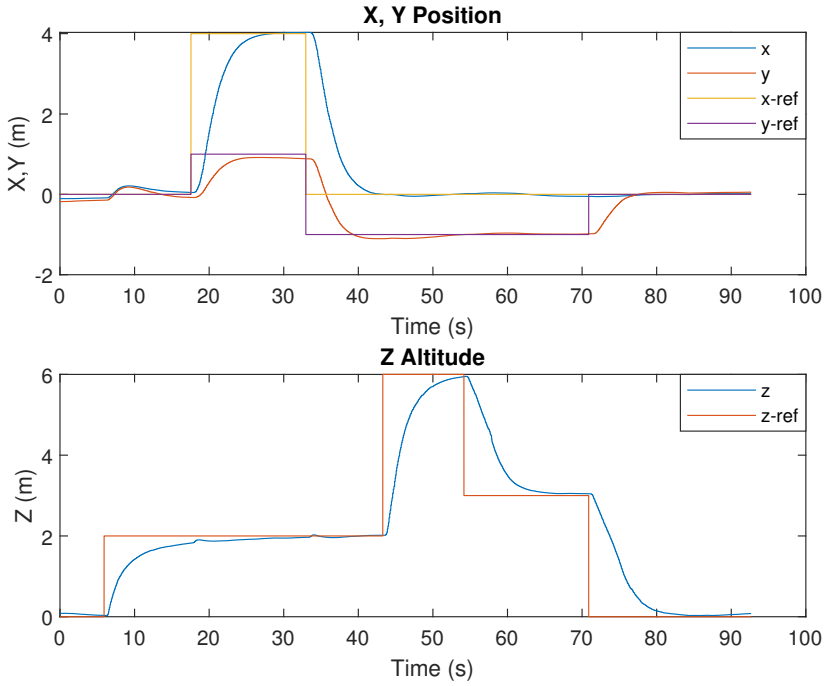


Figure 5.5: Trajectory following using PX4 controller [2]

good part of the controller in use is the referred one and the other part is still from the PX4. In order to achieve similar results as shown before it wasn't as easy as only sending position coordinates, there was a need to convert *u1* in a measure that the PX4 would properly respond. It was done by setting a lower limit of 0 and an upper limit of 20 to *u1* and then dividing by 20. The choice of this value had the mass of the quadrotor into consideration as the mass is directly proportional to this value. With this conversion, the thrust input sent to PX4 is u1 $\in [0;1]$.

Upon this change it was imperative to properly adjust the gains, namely the position gain K_p and velocity gain K_v, which were drastically lowered to values near 1. After these changes, the expected positions were not being achieved, as there was constant offsets. Consequently, in order to solve this issue there was the need of adding a static gain in order to better achieve the desired position coordinates. These static gains added in the x and y axis were roughly 15 centimeters.

Even after these changes, with a couple simulations it was possible to infer that the drone's behaviour was not the expected one, the most relevant stated issue was that if trying to moving the drone to a very far position, the roll and pitch angles would be way too aggressive causing the drone to flip and the gravity would to the rest of the work to crash the drone into the ground. This issue was then fixed by limiting the roll and pitch angles that the controller would send as input, before converting Equation (3.21) to a quaternion, a simple conversion from the rotation matrix to Euler angles was done, which is way easier to manipulate values, and then limiting the angles by the following

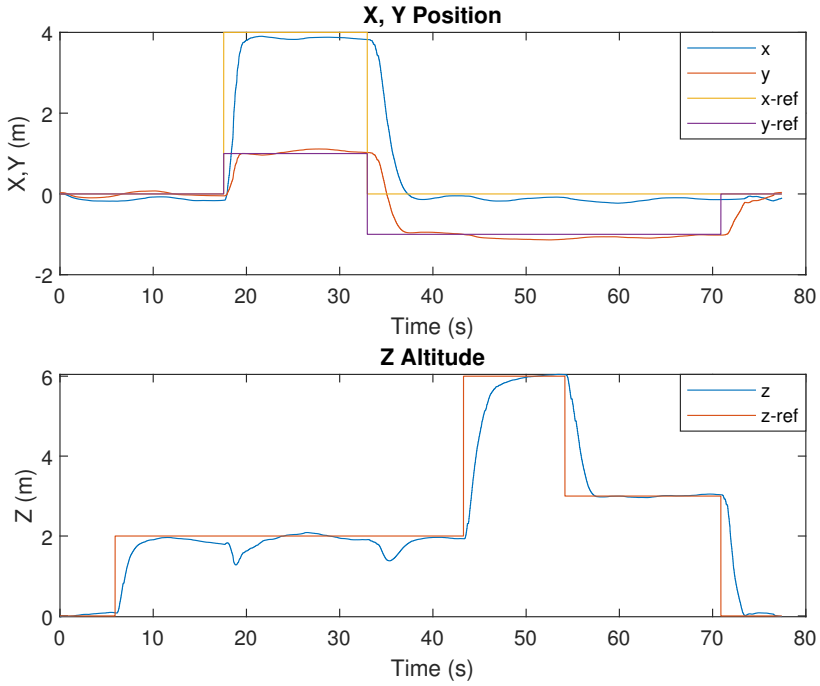


Figure 5.6: Trajectory following using attitude control

formula:

$$\theta_{des} \leq \frac{\pi}{3} rad$$
$$\phi_{des} \leq \frac{\pi}{3} rad$$

and finally the conversion to quaternion could be made. Despite this solution slowing down the drone in some occasions, it prevents the occurrence of unwanted situations, acting as a safety flight measure.

Figure 5.6 shows a trajectory following using the same trajectory presented in Figure 5.5, but this time instead of controlling only by position it's being controlled by attitude as explained before, the results are clearly worse than before. It's easy to conclude that the controller provided by PX4 is better designed than the one achieved in Chapter 3 which is expected, as PX4 provides a controller developed throughout years and offers integral effect which ours does not. Nevertheless, the achieved results are acceptable to be used in real case scenarios where the precision needed isn't the major concern, and we should also have into consideration that the reference shown is not quite the ideal as the positions change instantly and not smoothly, hence the tracking following is not easy to make in this case.

Another interesting point when comparing the results between Figure 5.5 where the controller from PX4 is being used with Figure 5.6 where our controller is being used, it is quite clear that PX4's controller is way slower hence allowing smoother trajectories while comparing with our controller that shows a quicker response.

Furthermore, it is still possible to increment the efficiency of this controller by tuning the gains and adjusting other parameters but it's always a battle between speed and precision. There's never the perfect adjusting as it depends on the final purpose of the controller and the gain tuning is a time consuming process and for the scope of this thesis the achieved results are pretty satisfactory.

5.2 Integration of Marvelmind system with PX4

The goal now is to move from 3D simulations made in Gazebo environment to real world tests using a real quadrotor controller, which in this case the chosen one to fulfill this task will be the PixHawk 4 Mini autopilot.

In an initial phase, the primary objective is to integrate the hedgehog beacon by Marvelmind with the PixHawk 4 Mini autopilot and furthermore to use the position provided by the beacon as a primary position resource, instead of the GPS as the tests made will be indoors, in the testbed shown in Chapter 4. Similar experiments can be seen in [26] with a difference that they use two beacons on top of the quadrotor in order retrieve yaw measurements, avoiding the drift in yaw measurements that occur with a magnetometer-based yaw measurement system.

To achieve this objective, two solutions will be presented, they differ in terms of the communication between the systems (Marvelmind and the autopilot). The first solution will be to electrically connect Marvelmind's hedgehog to the autopilot, directly feeding the autopilot with the position coordinates. On the other hand, the other solution doesn't require an electrical connection, instead it will depend on the communication through ROS. This last solution is more flexible as it is easier to adapt to other vehicles/controllers.

As these tests are meant to be done indoors we do not require a GPS module, hence we are in conditions to remove it and replace it by the hedgehog.

5.2.1 Communication through NMEA

This solution will show how to integrate Marvelmind with PX4 by electrically connecting the hedgehog directly to the autopilot. To achieve the desired goal, luckily Marvelmind provides instructions on how to make this integration between both hardware. These instructions provide a rather complete list of steps, starting from the beginning where it is explained how to update the firmware and set the necessary settings which is a rather quick process to make. Furthermore, it shows the pin connections that must be done in order to conclude all the steps. This requires a bit more effort as there are a lot of cables with small dimensions and they all need to be properly soldered (the complete list of steps can be observed in [33]). This configuration also mentions the importance of sensor calibrating such as accelerometer, compass and gyroscope as it is an important step in order to receive the most accurate data from the sensors onboard with a focus on maintaining the safety of any flying vehicle.

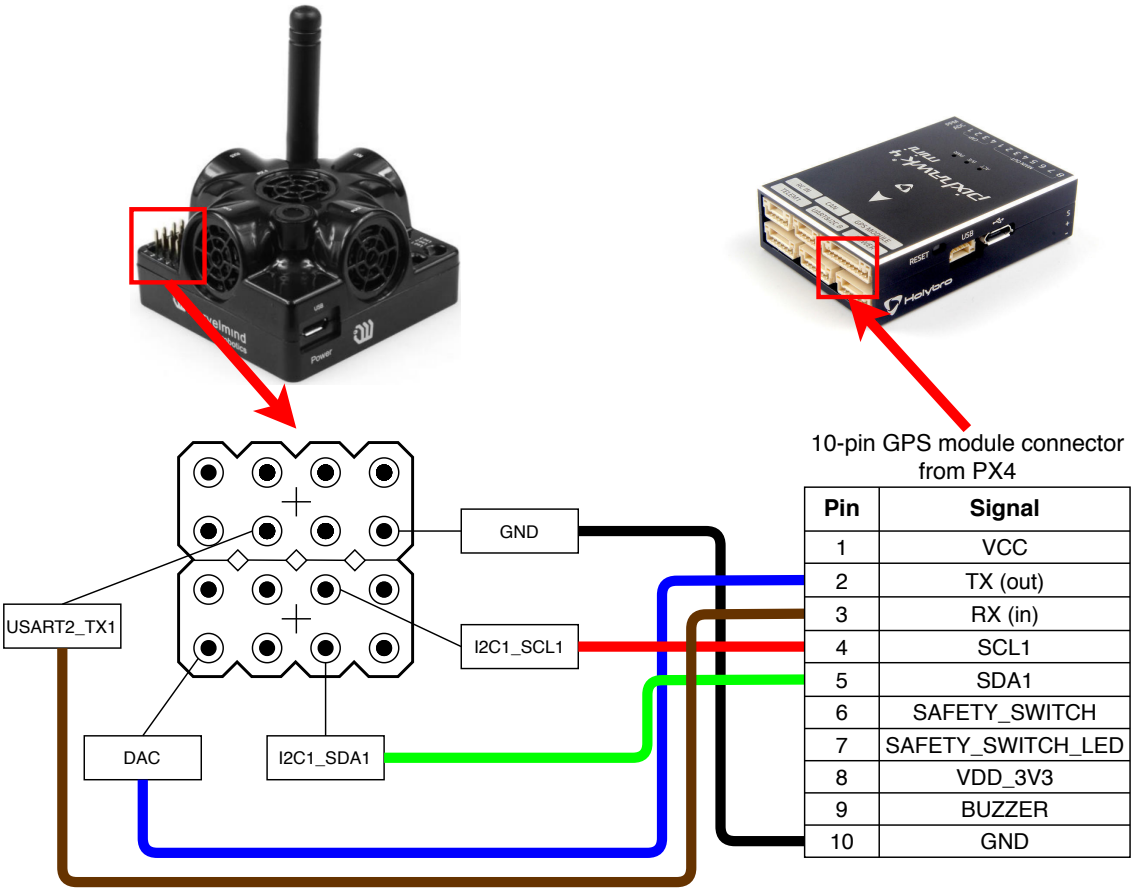


Figure 5.7: Electrical connection between Marvelmind and PX4

Upon having the necessary cable which has 10-pin GPS connector (provided by the autopilot's manufacturer) to individual servo female connectors, we are now in conditions to make the connection between the PX4 autopilot and the Marvelmind hedgehog beacon. In Figure 5.7 it is possible to see a scheme that allows to visualize the connections that were made.

Both the mentioned calibrations made and the configurations that had to be adjusted in order to ignore the real GPS module and accept the coordinates coming from Marvelmind had to be done in QGroundControl (QGC) which is basically an intuitive and powerful ground control station that provides full flight control as well as mission planning for MAVLink enabled drones. Its interface is rather easy to use and the features offered by this software are immense, allowing researchers to develop and improve drone controlling with good conditions [42].

In terms of configurations on Marvelmind's side, it was needed to change the protocol in which the data is being sent from Marvelmind's hedgehog to use NMEA 0183 protocol (unlike the UBX protocol mentioned in [33]), and the UART speed to 115200 bps. This configuration was done in *Dashboard* software and can be seen in Figure 5.8.

In order for the QGroundControl to place the drone in the real coordinates of the

Interfaces	(-) collapse
UART speed, bps	115200
Streaming output	USB+UART
Protocol on UART/USB output	NMEA 0183
NMEA message $GPRMC	enabled
NMEA message $GPGGA	enabled
NMEA message $GPVTG	enabled
NMEA message $GPZDA	enabled
NMEA message $GPHDT	enabled
Use IMU fusion for location in NMEA	enabled
Use IMU fusion for speed in NMEA	enabled
External device control	No control
PA15 pin function	SPI slave CS
Raw inertial sensors data	enabled
Processed IMU data	enabled
Raw distances data	enabled
Quality data stream	enabled
Telemetry stream	disabled
Telemetry interval, sec (1..255)	1
Locations of other hedgehogs	disabled
IMU via modem	(+) expand
User payload data size (0..48)	0

Figure 5.8: Marvelmind streaming data configurations through NMEA 0183

testbed, which is inside the campus of Faculty of Science and Technology from NOVA University of Lisbon. The coordinates (latitude and longitude) of the place were retrieved using Google Maps as accurately as possible so we can see the drone moving roughly in the position where it actually is. These configurations had to be added in the *Dashboard* software in the Georeferencing tab. It's worth mentioning that these coordinates have no impact on the results that we are expecting to achieve, the location of the drone in the world is not required, it's just a bonus feature as the tests are indoors.

At this point, there was the need to decide which estimator to choose and upon a careful glance at this matter, the estimator that stands out the most when comparing to the other options, was the EKF2. This choice is specially due to the fact that it can fuse data from sensors with different time delays which is really important in our case, even though this is a complex algorithm and demands higher computational resources. The second best option was the LPE but due to the needed requirements it wouldn't provide the performance that EKF2 is able to, hence the choice.

The way this solution is working is, as referred, through EKF2 algorithm which combines the information from all the sensors such as IMU, magnetometer and the position from Marvelmind and estimates all the states as position, velocity, the rotation between the body frame and local earth in X, Y and Z axis, earth magnetic field and other states.

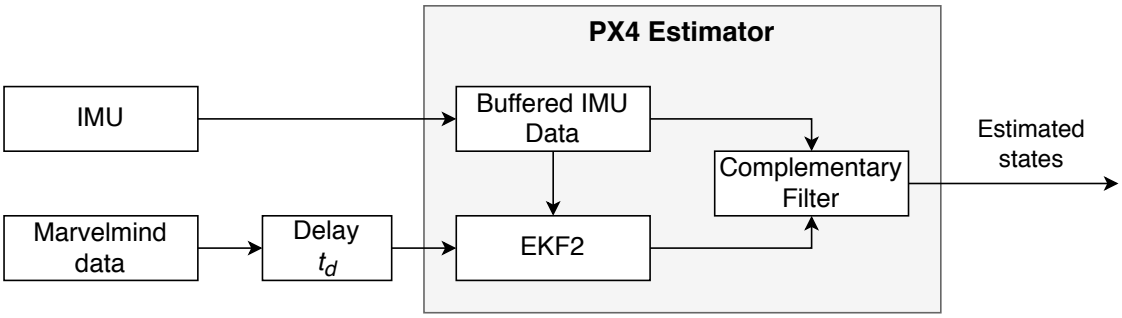


Figure 5.9: PX4 autopilot state estimation

```
vehicle_gps_position_s
        timestamp: 2687497020  (3.670653 seconds ago)
        time_utc_usec: 0
        lat: 386605167
        lon: -92047936
        alt: 426
        alt_ellipsoid: 0
```

Figure 5.10: Console output of the Geographic coordinates

It is this combination of data that allows the autopilot to properly control the drone. This algorithm works on a delayed *fusion time horizon* so that it can allow different delays on each of the sensors relatively to the IMU, hence the need to know the delay between the measurements and set it in the parameter EKF2_EV_DELAY (this step will be done further ahead). Furthermore, a more complete explanation on the functionality of EKF2 and how to tune it depending on the purposes aimed to achieve is provided by PX4 and can be seen in [15].

The complexity of the architecture of this algorithm is immense, nevertheless it's important to understand how the estimation of the states is being done, as well as understand the fusion of the data received by the autopilot, hence the creation of the schematic that can be seen in Figure 5.9 that represents in a simple and compact way the main important aspects about the state estimation.

Now that these configurations are done, the objective was to know if the communication between both systems is being done correctly, for that we need to make sure that the position coordinates are being received in the autopilot. With Marvelmind streaming the coordinates through NMEA 0183 and the autopilot connected to a local machine, using the output console provided by QGC it was possible to use the command *"gps status"* which outputs the Geographic coordinates that were established in Marvelmind's system, that gives us the confirmation that the communication is established and at the same time it validates that connections of the cable are properly done. The output given by the console in this initial test can be seen in Figure 5.10.

Although the communication between the systems is being well done, only with the Super Beacon from Marvelmind system it would be possible to use the coordinates

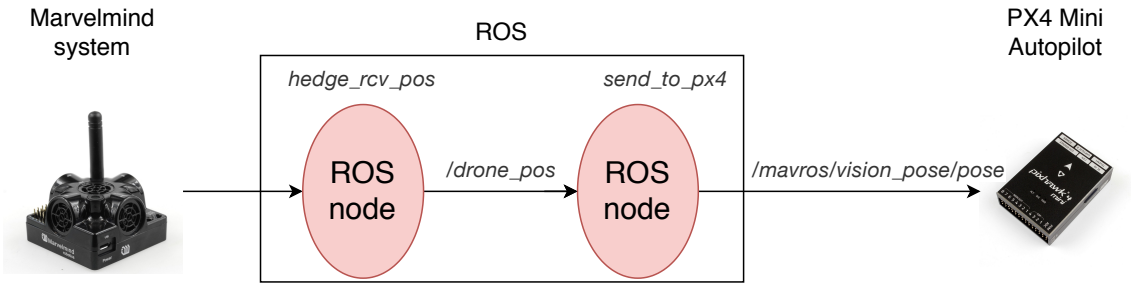


Figure 5.11: Communication between PX4, ROS and Marvelmind

received as being the local positions that PX4 should use while performing autonomous flights. Facing this setback derived from not having the appropriate devices to move forward with this solution, there is another possibility to achieve the same goal which is exactly what we can see in the following Chapter 5.2.2 where the integration between systems is being done using ROS.

5.2.2 Communication through ROS

As an alternative to the results obtained by electrically connecting Marvelmind's hedgehog directly to PX4 autopilot as shown in Chapter 5.2.1, another solution was considered and implemented. This method can be seen as being easier in terms of hardware implementation, as there is no need to make physical adaptions to allow the communication between the systems, but it requires a bit more of work in terms of software implementation as we will observe ahead. This solution will be presented a bit more in-depth as it is a more generic way to integrate the systems. Even if the autopilot would be changed it would not require as many changes as it would with the solution presented in Chapter 5.2.1, therefore the focus on this one.

As an alternative to the electrical connection between both hardware, this method is designed so that Marvelmind transmits the data towards a ROS topic via Marvelmind protocol (instead of NMEA 0183) and then ROS publishes this data to the PX4 autopilot. As it is easy to understand, ROS is being used as a bridge between the systems and is responsible by remapping the data so that PX4 knows how properly read and use it. This structure is simplified in Figure 5.11 allowing to easily visualize the communication between the systems. In this figure it's possible to visualize that the position retrieved from Marvelmind system is being published to a ROS node *hedge_rcv_pos* (this node is responsible by transferring the data from Marvelmind into a ROS topic and it has been already mentioned before in Chapter 4) and published into topic */drone_pos*, which then is received by another ROS node, *send_to_px4*, that was created with the objective of receiving the data from */drone_pos* topic and send it to PX4 Autopilot via */mavros/vision_pose/pose* topic.

As mentioned, to accomplish this method there was the need to create a new ROS node responsible by remapping the data into a topic that PX4 accepts as the local position

Interfaces	(-) collapse
UART speed, bps	115200
Streaming output	Disabled
Protocol on UART/USB output	Marvelmind
External device control	No control
PA15 pin function	SPI slave CS
Raw inertial sensors data	enabled
Processed IMU data	enabled
Raw distances data	enabled
Quality data stream	enabled
Telemetry stream	disabled
Telemetry interval, sec (1..255)	1
Locations of other hedgehogs	disabled
IMU via modem	(+) expand
User payload data size (0..48)	0

Figure 5.12: Marvelmind streaming data configuration through ROS

of the drone, this being */mavros/vision_pose/pose* as mentioned in [51] where most of the important steps to feed the autopilot with external sources as Visual Inertial Odometry (VIO) and Motion Capture (MoCap) are shown.

This mentioned ROS node receives the data from topic */drone_pos*, which as referred before is a message of type *geometry_msgs/Point* and then it must be published in */mavros/vision_pose/pose* topic, which is a message of type *geometry_msgs/PoseStamped* that is essentially composed by a message of type *geometry_msgs/Point* for the position and another message of type *geometry_msgs/Quaternion* for the orientation of the vehicle.

At this point, the remapping is only done for the position message, sending an empty message of orientation as this is not important in this case. The responsibility of obtaining the orientation of the quadrotor is given to the IMU of the autopilot hence the need of only receiving the position in the autopilot.

As in the previous solution, in this one there is also the need to configure parameters on QGC in order to properly use EKF2 to estimate most of the states of the quadrotor and combine the data with the position received from ROS during the autonomous flying experiments.

As explained, the communication is now made via Marvelmind protocol, hence the need of making the necessary adjustments in *Dashboard* software from Marvelmind, and these changes are resumed in Figure 5.12.

A drawback of this solution is that there is a bigger delay between the position retrieved by Marvelmind and the position that the autopilot is using as the autopilot is not

being fed with the data directly.

In order to test this solution, it is needed to run a MAVROS launch file (px4.launch), this file allows to establish the connection between the local machine and the autopilot, and for that we need to define the IP and the port of the computer running the simulation, which in this case is the local machine, meaning the IP parameter can be omitted.

So for the parameters of the Flight Control Unit, which in this case is the PX4 we are using the local machine IP and the port 14540, resulting in

fcu_url:=udp://:14540@

and there is also the need to specify the same parameters for the GCS which in this case is also running in the local machine and with port 14560, resulting in

gcs_url:=udp://@localhost:14560

so in order to run this command with the needed parameters we have

roslaunch mavros px4.launch fcu_url:=udp://:14540@ gcs_url:=udp://@localhost:14560

At this point, we are able to communicate with the autopilot and send commands to it, but due to some limitations we can not open the GCS at the same time, which in this case is the QGroundControl, hence the need to use a bridge to communicate both with MAVROS and QGroundControl. For that, MAVROS provides a node called *gcs_bridge*, where we must specify the parameters (IP and the port) for the connection, which must be same as the ones configured in the gcs_url in the launch file, therefore we can run the following command:

rosrun mavros gcs_bridge _gcs_url:=udp://@127.0.0.1:14560

where 14560 is the port being used and 127.0.0.1 is the IP of the local machine.

Furthermore, in QGroundControl, there was the need to establish the parameters for the connection as well. To do this, in tab *General* → *CommLinks*, a new UDP connection was created where the IP was specified as being the one from the local machine and the port was specified to be the one we connected the bridge to, being 14560.

We are now in conditions to visualize all the information of the PX4 autopilot in the GCS via the MAVLink inspector, where the information of all the available MAVLink messages can be seen in real time, which is really useful in order to check if the data we want to send is being received and used.

With all these settings properly configured, the conditions to start sending the coordinates that we want the PX4 to use are now met. In order to visualize if the data we are sending and the data that is being estimated in LOCAL_POSITION_NED is roughly the same, we must set the following parameter

MAV_ODOM_LP = 1

Parameter	Setting
SYS_MC_EST_GROUP	Make sure that the estimator is set to ekf2 (Default)
EKF2_AID_MASK	Enable *vision position fusion* and disable *inhibit IMU bias estimation* (12)
EKF2_HGT_MODE	Set to *Vision*
EKF2_EV_DELAY	Configure the delay of the position received from vision pose (Marvelmind)
MAV_ODOM_LP	Set to "1" to easily debug the results
COM_RC_IN_MODE	Set to "1" to deactivate the RC controller check
CBRK_IO_SAFETY	Set to "22027" to deactivate the need to push the safety switch from the GPS (that has been turned off)
CBRK_USB_CHK	Set to "159753" so that arming the vehicle while connected to USB cable is possible

Table 5.2: EKF2 Configuration

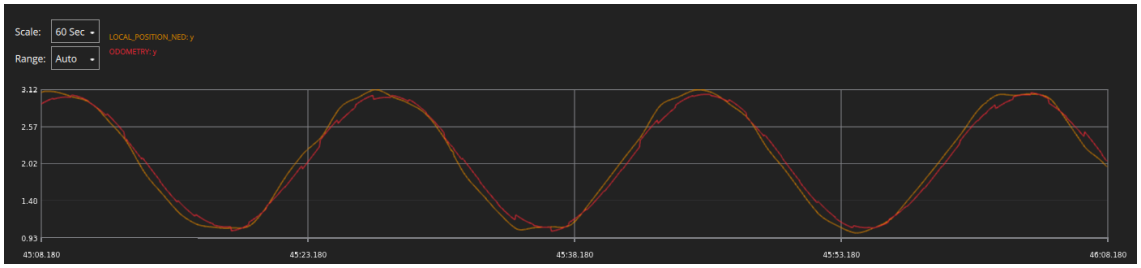
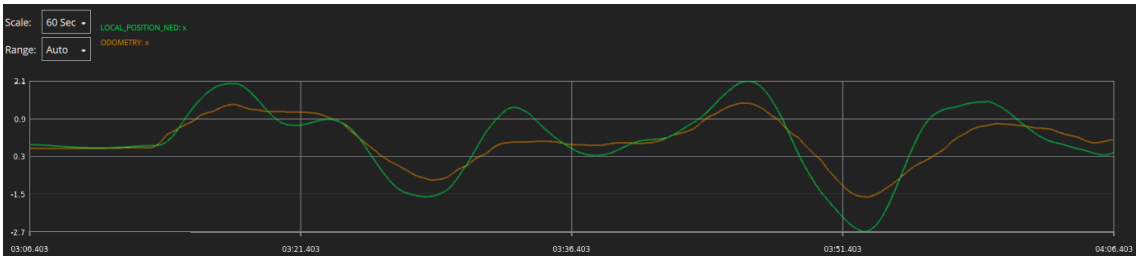


Figure 5.13: Performance test of EKF2

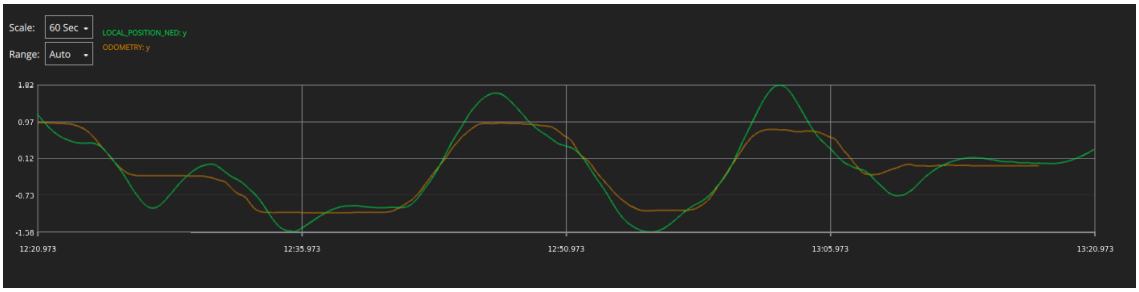
so that PX4 will stream back the received external pose as MAVLink ODOMETRY messages as explained in [51]. A simple list with some of the most important parameters to make this method work are shown in Table 5.2, some of the parameters' functions and their need will be better explained ahead.

Furthermore, in order to reach some conclusions in terms of the efficiency of this process, a simple test was made. This test consisted basically in sending hard-coded positions, with noise, in order to see how the estimator reacts to them. In this test we are supposed to observe a very small delay or even no delay between the position estimated by the EKF2 (local position) and the position that is being sent from the ROS node (vision position). This test was made using the plotting functionality of QGC, plotting in the same graph the y-axis from LOCAL_POSITION_NED and the y-axis of the ODOMETRY. Figure 5.13 shows the results of this simple test.

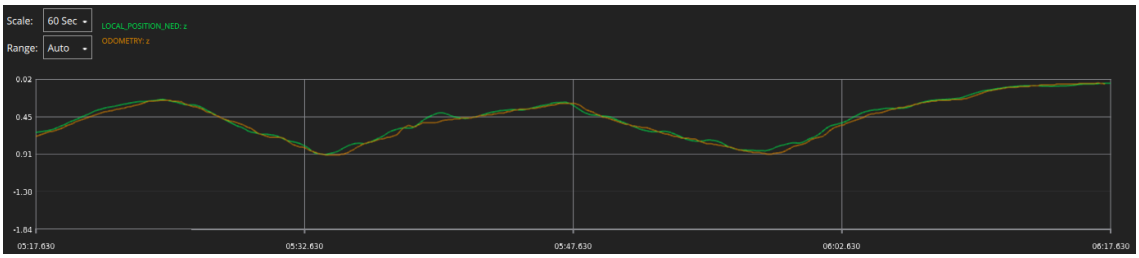
A quick glance at the results obtained in Figure 5.13 shows us that the estimation system in used is doing a very good job as the position being used is pretty much the same as the one being sent. We can also see that this estimator is acting as another layer



(a) x-axis



(b) y-axis



(c) z-axis

Figure 5.14: Performance test of EKF2 with real data (Delay not configured)

of filtering which is good to avoid unwanted behaviours from the drone.

Another important aspect to carry out with the real experiments is that there was the crucial need to proceed with the orientation coordination between the testbed and the autopilot, which was explained in detail in Chapter 4.2. With the calibration between the systems done, we now meet the conditions to make some more consistent tests in order to prove the reliability of the systems in cause.

It's time to feed the autopilot with the real values coming from Marvelmind and evaluate how it reacts to them. In a first approach, we will observe how well the EKF2 state estimator reacts to real values, with real movements and obviously, with real environmental disturbances.

In the course of these tests, Marvelmind's beacon was physically attached to the quadrotor, even though the drone was shut down, with no blades attached and was being human moved. The outcome results for each of the axis can be seen in Figure 5.14.

Taking a closer look at Figure 5.14a and Figure 5.14b it is clear that the results obtained are not good, the estimation done by EKF2 is really not the desired goal, yet the reason for these undesired results are clearly due to the delay of the received positions from Marvelmind, this can be concluded because in Figure 5.13 the results are very good. We

can clearly see that the estimation is taking into account the results coming from the IMU and then it should adjust its location based on the coordinates sent from the external source, but as the autopilot doesn't know that the coordinates are delayed it thinks that it is still moving hence the resulting overshoots when some harder movements are done.

In terms of the results on Figure 5.14c regarding the z-axis, they are definitely better than the other two axis, that's due to the fact that the movement done in the z-axis was slow and the barometer measurements are helping, hence the delay does not corrupt the estimation in this case, but if faster movements were done the results would be equally bad. A couple of other tests were made, and sometimes it was observable that the estimation of the local position was diverging a lot independently of the axis in case, sometimes more than 10 meters, which obviously is unusable.

However, this issue related to the delay can and will be solved, for that the parameter responsible to adjust the delay between the data from the IMU and the one received from the MoCap system must be changed, the parameter is

EKF2_EV_DELAY

as mentioned before. Starting from a delay of 0ms, multiple tests were made, progressively increasing the delay parameter by 50ms, until the best results were obtained, fixing the delay in

EKF2_EV_DELAY = 400ms

which is a bit higher than desired, yet good results can be obtained facing this delay value, as we can observe in Figure 5.15.

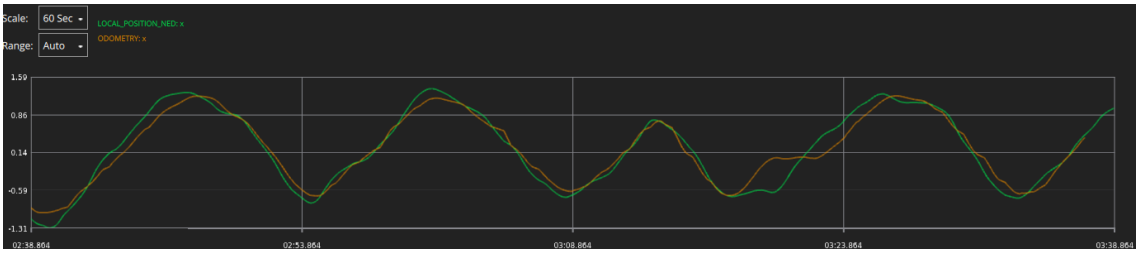
In terms of the results obtained in Figures 5.15a, 5.15b and 5.15c are way more acceptable than the ones seen prior to the delay calibration and now the estimation is being correctly done even when there are fast movements which is always important.

Another aspect to take into consideration is that, if there are continuous mismatch in any of the axis, it can be configured in EKF2 parameters. From what was observable, it was possible to see that there was a gap of roughly 1.5cm, therefore a small calibration of the z-axis was done, adjusting the following parameter

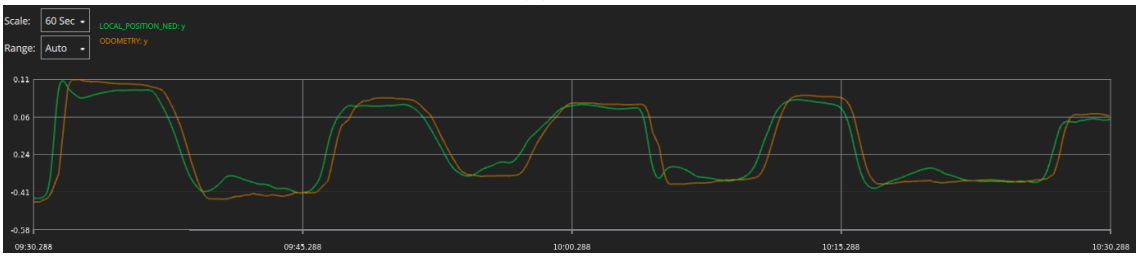
EKF_EV_POS_Z = 1.5 cm

and improving the precision of the position on the z-axis, making the outcome more consistent with the real local position.

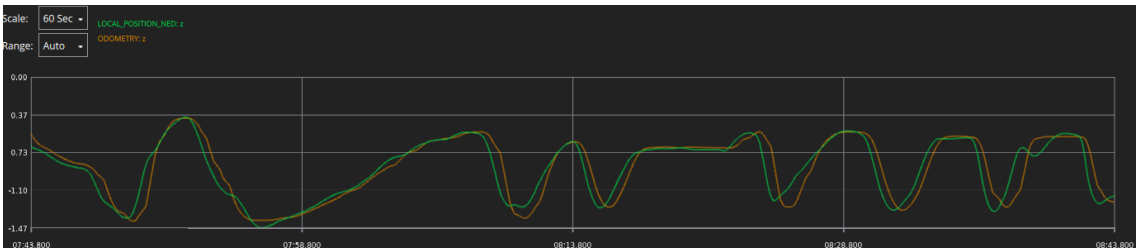
At this point, the results in terms of state estimation and testbed validation in order to allow safe flight to be made inside its perimeter is achieved. The only missing step is to proceed with real experimental trials, meaning to have a drone actually flying inside the testbed.



(a) x-axis



(b) y-axis



(c) z-axis

Figure 5.15: Performance test of EKF2 with real data

5.3 Real experimental trials using telemetry radio

So that we can autonomously fly a quadrotor inside the testbed, there were quite some changes that had to be done. In first place, we obviously must get rid of the USB connection to the computer and replace it with a Micro Transceiver Telemetry Radio Set, which is also a component that's included in the Holybro kit that's being used throughout this Chapter. Figure 5.16 shows this telemetry radio with the respective antenna, which works on the European 433MHz band (same band as Marvelmind system).

The installation of this telemetry radio is a rather simple process as it is plug-and-play and requires minimum configurations beforehand. Firstly, the USB cable that was being used before must be removed. Upon that, if the respective module that communicates with the telemetry radio is connected to the autopilot, we are already in conditions to receive and send messages from the autopilot to the local machine. With that said, the only thing that needs to be done now is to tell MAVROS where to communicate with the FCU. This can be done in multiple ways, one of the ways is to change the fcu_url parameter in file $px4.launch$ and change from ttyACM1 to ttyUSB0. With this configuration in place, and the communications established, QGC was opened to confirm that the data was being received from the autopilot.

Figure 5.16: Telemetry radio

The first impressions upon checking the MAVLink messages that were being shown in QGC was that the transmission rate was way lower than the ones observed when using USB cable. This can be a problem, as for example, the position we were sending from Marvelmind, was only being transmitted in a rate between 0.8Hz and 1Hz which is unviable, because in order to use the Offboard mode, PX4 requires a minimum rate of 2Hz when receiving "external vision" or MoCap pose [37]. To outcome these troubles, another communication as Wi-Fi modules can be used, Appendix A gives more details regarding the differences between these two communication types.

Even with this telemetry radio, it should be able to perform a simple experimental trial inside the testbed. To accomplish that, the blades were assembled, the battery was charged, the safety measures were double checked, specially the cables that are attached to the aircraft, and the drone was put in place to start the first experiment with an actual drone flying.

The objective of this initial test was simply to make a takeoff, hover for a small amount of time and then land it again. Despite the conditions to proceed with the test were met, the flight did not go exactly as planned, because upon arming the engines of the quadcopter, it automatically started flying, even without any *take off* command. Rapidly it was noticeable that something wrong had happened, hence the cable attached to the drone was pushed, holding it in place and preventing the drone to collapse in any hard material, while at the same time, maintaining people away from any potential harm caused by spinning blades.

Furthermore, it was crucial to understand what caused this undesired behaviour, and that's where the logs come handy. Taking a closer look at the registered logs it was possible to observe that the quadcopter entered in *failsafe mode*, attempting to perform some maneuver that shouldn't have been done. Nevertheless, we can observe the trajectory performed during this flight in Figure 5.17, as a rosbag subscribing all topics was recorded, allowing to recover all the data registered in the experimental trial.

Although the flight did not meet the expectations, the testbed proved to have the necessary safety measures.

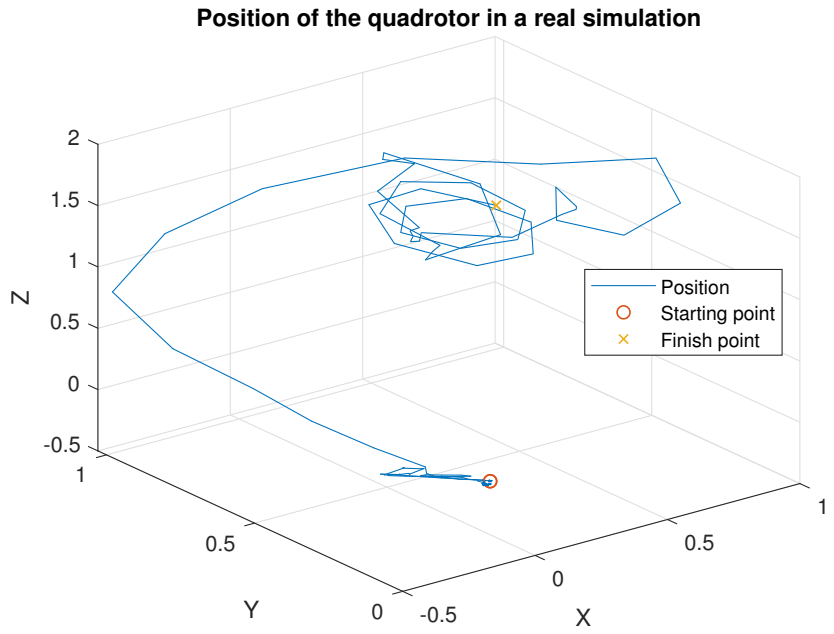


Figure 5.17: Experimental trial using telemetry radio

5.4 Real experimental trials using a WiFi module

The flight shown in Figure 5.17 was not concluded with the efficiency that was desired, mainly due to the fact that the telemetry radio in use was not capable of transmitting all the data at a speed rate enough to properly perform this type of flights. In order to overcome this problem, the telemetry radio was removed and replaced by a Wi-Fi microchip, the ESP2866, which can be seen in Figure 5.18.

This microchip operates at a speed of 2.4GHz, that way it is able to achieve much faster communication rates when comparing with the telemetry radio, which operates at 433MHz.

Fortunately, PX4 allows the implementation of this microchip with ease. A guide to the integration of a ESP2866 module can be seen in [17].

In first place, so that the Wi-Fi module is able to communicate MAVLink messages, there was the need to flash the pre build binaries provided in the mentioned guide. The FT232 adapter, which can be also be seen in Figure 5.18, provides the ability to make the communication between USB And UART, allowing to flash the mentioned file to the Wi-Fi module upon establishing the electrical connection between these two devices, which can be seen in the same figure. It's important to make sure that the FTDI module is set to 3.3V (instead of 5V) in order to avoid damaging ESP2866.

Upon connecting the FTDI module via USB to the computer using Linux operating system and making sure that *esptool* software is installed, it remains to set ESP2866 into *flashmode*, which can be done by simply pressing both *Reset* and *GPIO-0* buttons at the same time and then releasing the *Reset* button in first place. A red LED will start blinking

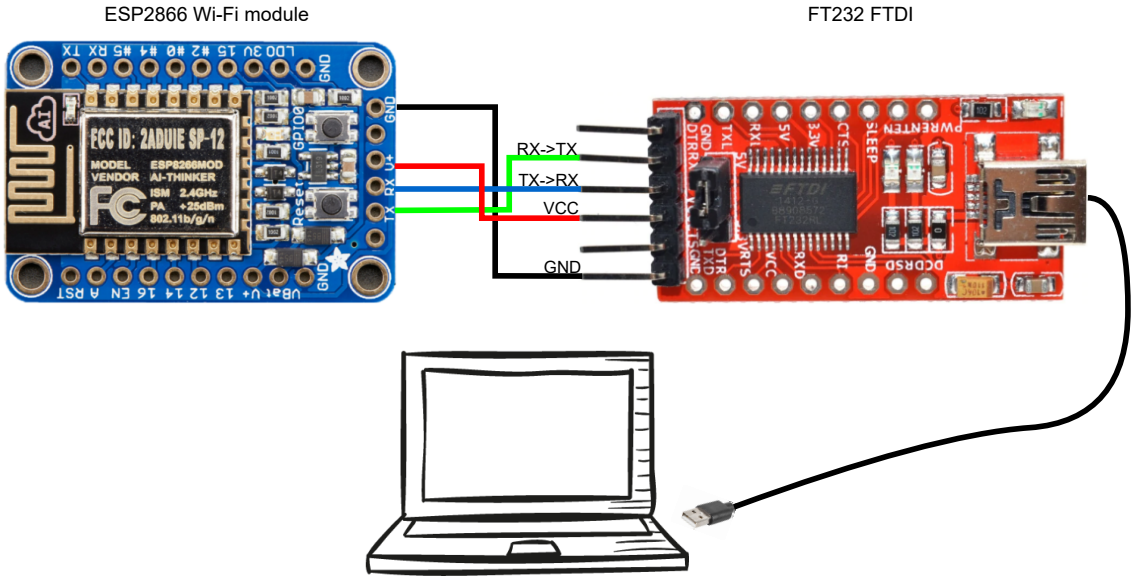


Figure 5.18: Electrical connection between ESP2866 and FT232 to flash pre built binaries

with low intensity ensuring that the *flashmode* is enabled.

On the console, the following command allows flashing the pre build binaries into ESP2866:

esptool.py –baud 921600 –port /dev/your_serial_port write_flash 0x00000 firmware_xxxxx.bin

where 921600 is the baud rate, *your_serial_port* is the USB port that is being used (e.g., /dev/ttyUSB0) and *firmware_xxxxx.bin* is the file previously downloaded.

After flashing the firmware, which is a process that takes less than a minute, ESP2866 was configured as an access point. This allows the computer to connect directly to its Wi-Fi network, which by default, has both the SSID and password set to *pixracer*. To check the status and configure some parameters it is possible to access this network's IP which is defined by 192.168.4.1.

Additionally, a cable was made to directly connect ESP2866 into TELEM1 port of PX4 autopilot, by soldering the TELEM1 connector to servo connectors. The autopilot was connected to the computer via USB cable to finish the configuration, as well as readjust the baud rate of TELEM1 port in QGC to match the one configured before (921600).

Finally, the USB cable was removed, the autopilot was connected to an external battery and QGC automatically connected to this network and start transmitting MAVLink messages.

The interaction between Marvelmind and the autopilot using this ESP8266 microchip can be seen in Figure 5.19.

Furthermore, it's important to ensure that the transmission of data between QGC and the autopilot is smooth and that the parameter MAV_0_MODE is set to External vision, allowing to maximize the number of messages that are transmitted.

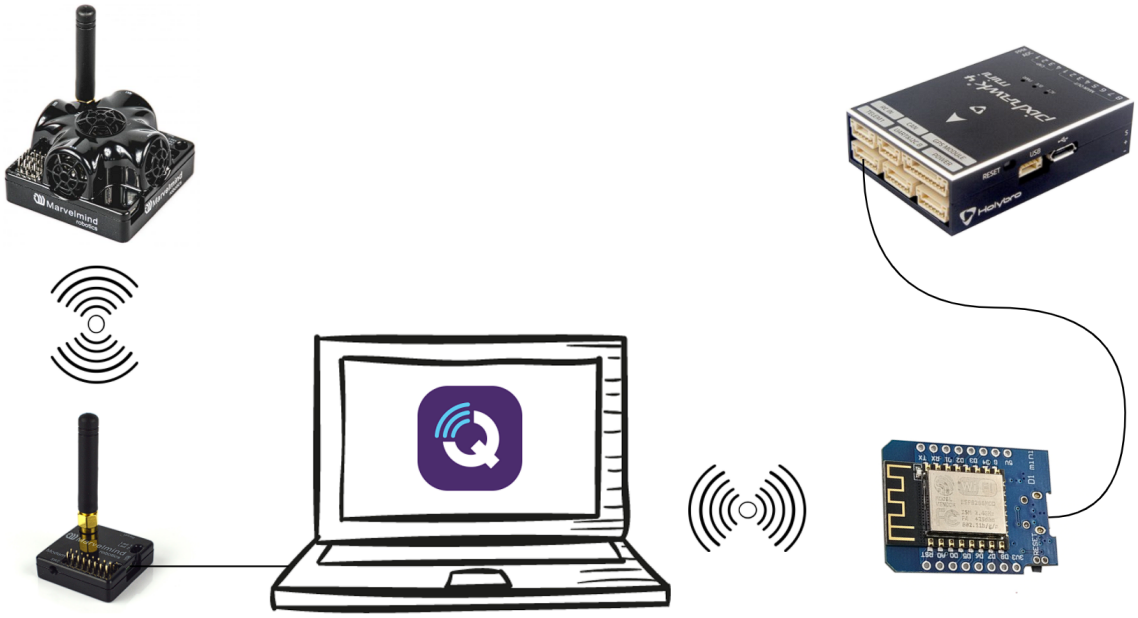


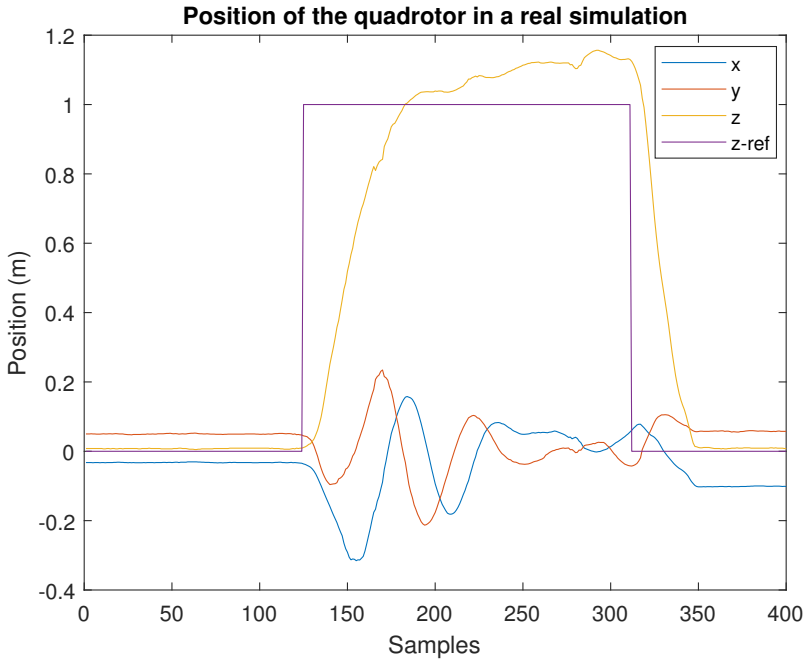
Figure 5.19: Interaction scheme between Marvelmind and ESP2866

The conditions to make a simple takeoff flight are achieved, hence the preparations to do so. The blades were attached, the battery was plugged and made sure it was charged enough to complete the whole flight, the protection ropes were attached to the aircraft, the safety measures were double-checked and the flight was done.

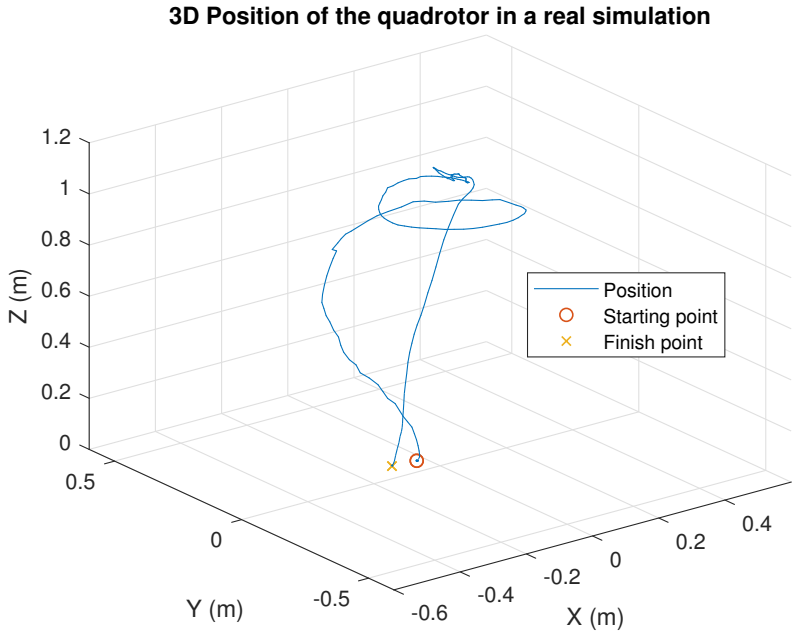
The objective of this experimental trial was easy, arm the rotors, takeoff to an altitude of 1 meter and then land again. To perform this trial the takeoff velocity was lowered, adding this to the fact that the drone has more weight than its intended to (Marvelmind beacon sitting on top of the aircraft) and might not be well balanced, which might cause it not to fully fly upwards with no oscillations.

The results of this trial were recorded to a *rosbag* and plotted in MATLAB so that it is easier to analyze the data. This data can be seen in Figure 5.20 where it is possible to see some small oscillation in X and Y axis (roughly 20cm) which might be a result of the mentioned facts. However, the results shown are very solid, the trial was completed with success, there was not unwanted movements and there was not the need of any external force to hold the quadcopter in place. Also regarding Figure 5.20, the reference of the z axis was manually added after the experimental trial, meaning it is not the exact reference but roughly what was intended to happen.

This experimental trial fortifies and validates the work developed in terms of the testbed, integration with the external MoCap system, and even the ESP8266 Wi-Fi module introduced in this section which drastically improved the performance of the autonomous flights using an external position system.



(a) 2D Position



(b) 3D Position

Figure 5.20: Experimental trial using ESP8266

Chapter 6

Conclusion

The first objective of this thesis was to understand how quadrotor controllers work and design one, taking into consideration an already implemented and tested one. That objective was concluded with success and really good simulation results using MATLAB were obtained, as these can be seen throughout Chapter 3.

The second objective, which was essentially to design a drone testbed with the capabilities to localize quadrotors, while providing good safety measures to test aerial vehicles inside its perimeter, was also successfully achieved throughout Chapter 4. Furthermore, intensive accuracy tests to the results obtained using Marvelmind system were done, with the objective of validating the performance announced by the manufacturer and also to know what kind of efficiency could be expected from this system through the experimental trials.

Upon that, controller simulations were done using Gazebo environment and also good results were obtained, not only using PX4 controller but also using the controller developed through Chapter 3.

Furthermore, Chapter 5.2 shows that the integration between the autopilot (PX4 Mini) and the MoCap system (Marvelmind) is possible in two different ways. The first presented way was by electrically connect both systems, however it was not possible to show further results due to material restrictions. Regarding the second way, in order to bypass the first method, another solution was presented using ROS as a bridge between the autopilot and the MoCap system and successful results were achieved. After a couple adjustments and calibrations, the outcome position estimation resulting from the autopilot was really satisfying as well.

Some simulations without mounting the blades on the drone were done, just to see if the rotors would react correctly to inputs sent and the results obtained showed to be consistent. Upon that, a real experimental trial using a telemetry radio was attempted

but unfortunately, upon checking the logs, it was possible to see that a *failsafe mode* was enabled resulting in a bad flight, however it confirmed the good safety measures provided by the testbed.

Finally, the telemetry radio was removed and replaced by a ESP8266 Wi-Fi module, boosting the transmission rate of MAVLink messages. Some experimental trials were done and the results were very good, validating the efforts done throughout the whole thesis in designing a safe environment to experiment quadrotors.

As this is a long-term project, a manual with instructions and the code used to perform the interaction between the systems have all been shared in a repository which will serve as guide and help whoever might want to use the testbed. This repository can be seen in [9].

6.1 Future work

There are some possible future developments that could follow the course of this thesis. The first one would be to make some more intense tests, gain tuning and add integral effect to the controller developed in Chapter 3 with the objective of improving its robustness.

The second point that could be done as a future work would be to use the controller developed in Chapter 3 to control a drone inside the testbed, in order to confirm that the controller is able to achieve simple autonomous tasks.

Furthermore, make some more experimental trials with more complex trajectories and even try using other controllers. Ultimately make simulations with more than one quadrotor inside the testbed would also be an interesting point to follow, but for that it would be necessary to get additional Marvelmind beacons to attach to the remaining vehicles.

Bibliography

[1] M. Afanasov, A. Djordjevic, F. Lui, and L. Mottola. "FlyZone: A Testbed for Experimenting with Aerial Drone Applications." In: *Proceedings of the 17th Annual International Conference on Mobile Systems, Applications, and Services*. ACM. 2019, pp. 67–78.

[2] A Alaimo, V Artale, C Milazzo, and A Ricciardello. "Comparison between euler and quaternion parametrization in uav dynamics." In: *AIP Conference Proceedings*. Vol. 1558. 1. American Institute of Physics. 2013, pp. 1228–1231.

[3] K. Alexis, G. Nikolakopoulos, and A. Tzes. "Switching model predictive attitude control for a quadrotor helicopter subject to atmospheric disturbances." In: *Control Engineering Practice* 19.10 (2011), pp. 1195–1207.

[4] R. Amsters, E. Demeester, N Stevens, Q Lauwers, and P Slaets. "Evaluation of Low-Cost/High-Accuracy Indoor Positioning Systems." In: *Proceedings of the 2019 International Conference on Advances in Sensors, Actuators, Metering and Sensing (ALLSENSORS), Athens, Greece*. 2019, pp. 24–28.

[5] M. E. Antonio-Toledo, E. N. Sanchez, A. Y. Alanis, J. Flórez, and M. A. Perez-Cisneros. "Real-time integral backstepping with sliding mode control for a quadrotor UAV." In: *IFAC-PapersOnLine* 51.13 (2018), pp. 549–554.

[6] J. Bacik, F. Durovsky, P. Fedor, and D. Perdukova. "Autonomous flying with quadrocopter using fuzzy control and ArUco markers." In: *Intelligent Service Robotics* 10.3 (2017), pp. 185–194.

[7] *Building PX4 Software*. URL: https://dev.px4.io/master/en/setup/building_px4.html. (Accessed: 04.10.2020).

[8] D. Cabecinhas, R. Cunha, and C. Silvestre. "A nonlinear quadrotor trajectory tracking controller with disturbance rejection." In: *Control Engineering Practice* 26 (2014), pp. 1–10.

[9] H. Cabrita. *Rotary wing drone testbed*. https://bitbucket.org/bguerreirostudents/hugo-cabrita-code/src/master/. 2020.

[10] K. Chang, J. H. Kim, S. A. Wilkerson, and S. A. Gadsden. "Motion Capture Control of a Nano Quadrotor." In: *15th LACCEI International Multi-Conference for Engineering, Education, and Technology*. 2017, pp. 19–21.

[11] F Corrigan. *Drone gyro stabilization, IMU and flight controllers explained*. 2018.

[12] A. Das, K. Subbarao, and F. Lewis. "Dynamic inversion with zero-dynamics stabilisation for quadrotor control." In: *IET control theory & applications* 3.3 (2009), pp. 303–314.

[13] A. Dharmawan, T. K. Priyambodo, et al. "Model of linear quadratic regulator (lqr) control method in hovering state of quadrotor." In: *Journal of Telecommunication, Electronic and Computer Engineering (JTEC)* 9.3 (2017), pp. 135–143.

[14] J. Diebel. "Representing attitude: Euler angles, unit quaternions, and rotation vectors." In: *Matrix* 58.15-16 (2006), pp. 1–35.

[15] *ECL/EKF Overview and Tuning*. URL: https://docs.px4.io/v1.9.0/en/advanced_config/tuning_the_ecl_ekf.html. (Accessed: 25.10.2020).

[16] C. Edwards and S. Spurgeon. *Sliding mode control: theory and applications*. Crc Press, 1998.

[17] *ESP8266 WiFi Module*. URL: http://docs.px4.io/master/en/telemetry/esp8266_wifi_module.html#flashing-the-esp8266-firmware/. (Accessed: 06.12.2020).

[18] *Flight Controller (Autopilot) Hardware*. URL: https://docs.px4.io/master/en/flight_controller/. (Accessed: 25.11.2020).

[19] *Flying Machine Arena at ETH Zurich*. URL: https://www.flyingmachinearena.ethz.ch/history/. (Accessed: 03.10.2020).

[20] *Gazebo - Robot simulation made easy*. URL: http://gazebosim.org/. (Accessed: 01.10.2020).

[21] M. Greiff, A. Robertsson, and K. Berntorp. "Performance Bounds in Positioning with the VIVE Lighthouse System." In: *2019 22th International Conference on Information Fusion (FUSION)*. IEEE. 2019, pp. 1–8.

[22] L. A. Haidari, S. T. Brown, M. Ferguson, E. Bancroft, M. Spiker, A. Wilcox, R. Ambikapathi, V. Sampath, D. L. Connor, and B. Y. Lee. "The economic and operational value of using drones to transport vaccines." In: *Vaccine* 34.34 (2016), pp. 4062–4067.

[23] W. R. Hamilton. "XI. On quaternions; or on a new system of imaginaries in algebra." In: *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science* 33.219 (1848), pp. 58–60.

[24] A. Hernandez, H. Murcia, C. Copot, and R. De Keyser. "Model predictive path-following control of an AR. Drone quadrotor." In: *XVI Latin American Control Conference The International Federation of Automatic Control, Cancun, Mexico*. 2014.

[25] *HolyBro QAV250 + Pixhawk4-Mini Build*. URL: https://docs.px4.io/master/en/frames_multicopter/holybro_qav250_pixhawk4_mini.html. (Accessed: 15.10.2020).

[26] B. Katona and K. Morgansen. "Navigation of Indoor Spaces Using Multiple Quadrotors." In: *AIAA Scitech 2019 Forum*. 2019, p. 1412.

[27] B. Kempke, P. Pannuto, and P. Dutta. "Polypoint: Guiding indoor quadrotors with ultra-wideband localization." In: *Proceedings of the 2nd International Workshop on Hot Topics in Wireless*. ACM. 2015, pp. 16–20.

[28] H. K. Khalil and J. W. Grizzle. *Nonlinear systems*. Vol. 3. Prentice hall Upper Saddle River, NJ, 2002.

[29] M. Krstic, P. V. Kokotovic, and I. Kanellakopoulos. *Nonlinear and adaptive control design*. John Wiley & Sons, Inc., 1995.

[30] G. C. La Delfa and V. Catania. "Accurate indoor navigation using smartphone, bluetooth low energy and visual tags." In: *Proceedings of the 2nd Conference on Mobile and Information Technologies in Medicine*. 2014.

[31] S. Lupashin, A. Schöllig, M. Hehn, and R. D'Andrea. "The flying machine arena as of 2010." In: *2011 IEEE International Conference on Robotics and Automation*. IEEE. 2011, pp. 2970–2971.

[32] J. Martinez, A Pequeno-Boter, A Mandow, A García-Cerezo, and J Morales. "Progress in mini-helicopter tracking with a 3D laser range-finder." In: *IFAC Proceedings Volumes* 38.1 (2005), pp. 648–653.

[33] *Marvelmind and PX4 integration*. URL: https://marvelmind.com/pics/marvelmind_px4_integration.pdf. (Accessed: 15.10.2020).

[34] D. Mellinger and V. Kumar. "Minimum snap trajectory generation and control for quadrotors." In: *2011 IEEE International Conference on Robotics and Automation*. IEEE. 2011, pp. 2520–2525.

[35] D. W. Mellinger. "Trajectory generation and control for quadrotors." Doctoral dissertation. University of Pennsylvania, 2012.

[36] N. Michael, D. Mellinger, Q. Lindsey, and V. Kumar. "The grasp multiple micro-uav testbed." In: *IEEE Robotics & Automation Magazine* 17.3 (2010), pp. 56–65.

[37] *Offboard Mode*. URL: https://docs.px4.io/master/en/flight_modes/offboard.html. (Accessed: 06.10.2020).

[38] J. M. Pflimlin, T. Hamel, and P. Souères. "Nonlinear attitude and gyroscope's bias estimation for a VTOL UAV." In: *International Journal of Systems Science* 38.3 (2007), pp. 197–210.

[39] R. Portela Suárez. "Estudio del error e posicionamiento de los dispositivos POZYX." Doctoral dissertation. 2017.

[40] *PX4*. URL: https://www.px4.io/. (Accessed: 20.08.2020).

[41] *PX4 source code repository*. URL: https://github.com/PX4/PX4-Autopilot. (Accessed: 04.10.2020).

[42] *QGC - QGroundControl*. URL: http://qgroundcontrol.com/. (Accessed: 03.10.2020).

[43] I. A. Raptis and K. P. Valavanis. *Linear and nonlinear control of small-scale unmanned helicopters*. Vol. 45. Springer Science & Business Media, 2010.

[44] *Robot Operating System (ROS)*. URL: https://www.ros.org/. (Accessed: 20.08.2020).

[45] M. Robotics. *Marvelmind Indoor Navigation System Operating manual*. Nov. 2020. URL: https://marvelmind.com/pics/marvelmind_navigation_system_manual.pdf.

[46] A. M. Romanov, M. P. Romanov, A. A. Morozov, and E. A. Slepynina. "A Navigation System for Intelligent Mobile Robots." In: *2019 IEEE Conference of Russian Young Researchers in Electrical and Electronic Engineering (EIConRus)*. IEEE. 2019, pp. 652–656.

[47] *ROS Marvelmind package*. URL: https://bitbucket.org/marvelmind_robotics/ros_marvelmind_package/src/master/. (Accessed: 04.10.2020).

[48] A. Schöllig, F. Augugliaro, and R. D'Andrea. "A platform for dance performances with multiple quadrocopters." In: *Improving Tracking Performance by Learning from Past Data* 147 (2012).

[49] F. Sebatino. "Quadrotor control: modeling, nonlinear control design, and simulation." Master's thesis. KTH Electrical Engineering, 2015.

[50] M. I. da Silva Marques. "Trajectory planning and control for drone replacement during formation flight." Master's thesis. Instituto Superior Técnico - Universidade de Lisboa, 2018.

[51] *Using Vision or Motion Capture Systems for Position Estimation*. URL: https://dev.px4.io/master/en/ros/external_position_estimation.html. (Accessed: 25.10.2020).

[52] S. Vaidyanathan and C.-H. Lien. *Applications of sliding mode control in science and engineering*. Vol. 709. Springer, 2017.

Appendix A

Comparison between communication methods

The importance of reducing the delay between communications is crucial because a quadcopter can change its state in milliseconds. If you think about the speed that these devices can reach, which is in the order of tens of meters per second, it's easy to realize that an update rate of 1Hz is very slow for these devices.

If the objective is to make autonomous flights inside a testbed (indoors), the communication between our aerial vehicle and the local machine should be highly stable and have an effective and high rate that allows to receive and send messages, in this case MAVLink messages, providing the conditions to achieve a safe flight. The device that makes the communication should also not get flooded with messages, as that could increase the delay even more.

In terms of ways to make the communication, we have two main options that are commonly used, each of these presenting advantages and disadvantages when comparing with one another. These two options are telemetry radios (which operate in 433MHz in European territory) and Wi-Fi modules (operates in ranges of 2.4GHz and 5GHz).

In Chapter 5, when there was the need to have communications between the autopilot and the local machine, a telemetry radio was used as shown in Figure 5.16, but as mentioned, this device is not the best suitable for our study case.

The reason that this device is not effective is that it does not allow us to transmit position data with the highest possible transmission rate, reaching around 1Hz when transmitting the position retrieved by Marvelmind (which is being sent from ROS at a rate of 20Hz).

Even though the radio does not provide the transmission rate that we require, it still provides some advantages when comparing with Wi-Fi modules, making a small comparison between these both devices we observe that

- Telemetry radios have higher operational range, reaching hundreds of meters, while Wi-Fi modules reach only a couple dozens of meters.
- Telemetry radios do not require other network configuration, whereas Wi-Fi modules require a local network where they can connect to (some can work as an access point).
- Wi-Fi modules are cheaper when comparing to telemetry radios.
- Wi-Fi modules are able to send and receive messages at a higher rate, avoiding flooding the channel and providing more reliable and up-to-date data to the aerial vehicle.
- In terms of line-of-sight propagation, telemetry radios show less restrictions when comparing to Wi-Fi modules.

A small conclusion for our current study case is that, to properly make flights inside the testbed, a Wi-Fi module makes much more sense, not only to achieve better experimental results but also it makes the experiments highly safer. This fact is strongly confirmed in Chapter 5.4.

Appendix B

Pictures of the testbed and the quadrotor

This appendix contains some images taken to the used material. Figure B.1 shows the aircraft Holybro QAV250 after being assembled with Marvelmind's beacon attached to it instead of the traditional GPS module. Figure B.2 and B.3 shows the implemented testbed from two different angles while standing inside its perimeter. Figure B.4 shows the supports that were used in order to attach all the stationary beacons that compose Marvelmind's kit.

Figure B.1: Holybro QAV250 with Marvelmind attached

Figure B.2: Testbed

Figure B.3: Testbed

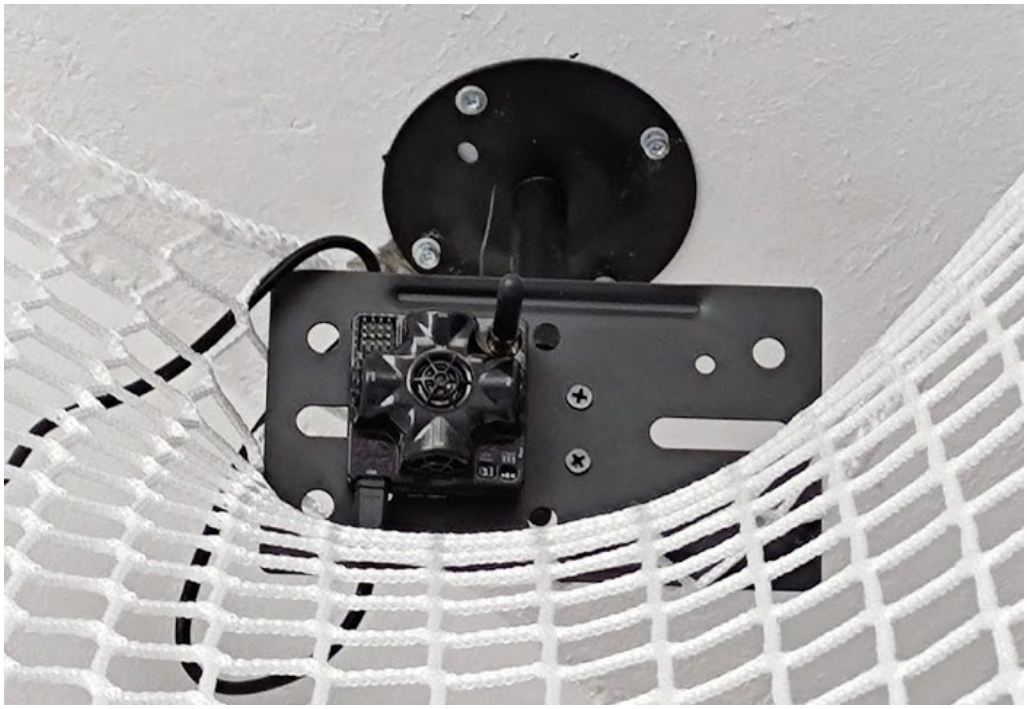


Figure B.4: Support holding Marvelmind's beacon