UCLouvain

epl

École polytechnique de Louvain

Development of an automatic guiding system for builder drones

Author: **José Antonio Alcaide Jiménez**
Supervisor: **Pierre Latteur**
Readers: **Igor Bouckaert , Ramin Sadre , Sébastien Goessens**
Academic year 2019–2020
Màster Universitari en Enginyeria Industrial

Abstract

Builder drones have the potential of aiding in construction sites by picking up and depositing masonry units. However, their accuracy is restrained due to drifting, wind and the uncertainty of positioning methods such as GPS.

This thesis explores the feasibility of a vision-based guidance system for builder drones. In particular, it examines the reliability of ArUco markers as a local positioning method, as well as different control approaches to move the drone towards them.

Finally, it creates a framework capable of executing a construction mission based on searching for a specific brick and depositing it on its correct position. Furthermore, complementary systems are developed to help future iterations of the project.

Acknowledgements

I want to start thanking Pr. Pierre Latteur for allowing me to work in this ongoing project at the EPL. This opportunity has been an unforgettable enrichment experience.

I would also like to thank the fantastic group of people that I met in Louvain-la-Neuve. I will never forget all the special moments that we shared.

Finally, I would like to thanks my friends, family and especially, Laura. They have stood with me every step of the way, and without their support, this project will not have been possible.

To all of you, thank you.

Contents

Chapter 1

Introduction

1.1 Context of the project

Working conditions in construction sites have improved over the last two centuries. The increase in safety concerns and the use of new technologies, such as BIM, has allowed for a more organised and safe development cycle.

However, the shortcomings of using human workforce remain. Lack of communication between the workers, lack of skilled labour, accidents or the effects of adverse weather, all can slow down and endanger operators. Automation can help reduce these difficulties by replacing ineffective and dangerous procedures with autonomous systems, increasing both efficiency and safety.

If we seek inspiration in nature for efficient building, we can observe the way birds construct their nest. These animals fly back and forth for resources, wood branches, and they deposit them to build their home. As some previous scientific teams have researched, replicating this process with flying robots no longer is an unreachable concept thanks to the utilisation of non-piloted aircraft [1] [2]. Figure 1.1 shows a flight assembled architecture developed in the university of ETH Zürich.

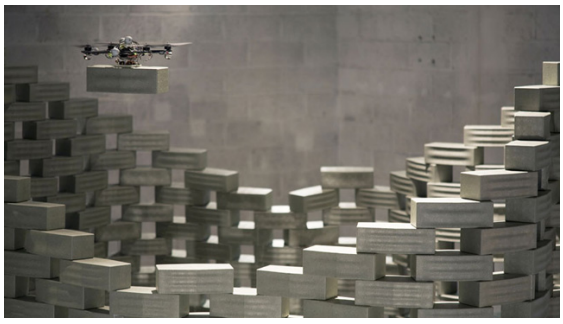
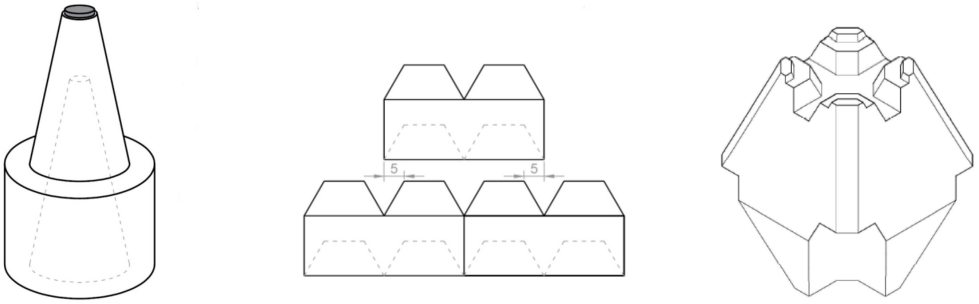


Figure 1.1: Flight assembled architecture by the university of ETH Zürich



(a) Circular drick (b) Rectangular dricks (c) Droxel

Figure 1.2: Drone-compatible bricks

Indeed, non-piloted aircraft, also known as drones, have existed since the mid-19th century and through history were used primarily as military armament. However, in the last years, they have suffered a substantial expansion of new uses, such as packet delivery or recreational purposes. Its popularisation opens the world to a new discipline of robotics and automation using UAVs.

The team of researchers in the Ecole Polytechnique de Louvain led by Pierre Latteur has been investigating the use of drones for building structures. The aim was to develop a robust system capable of overcoming the inaccuracy of these vehicles induced by drifting and wind. We can analyse their research from two angles: civil and automation engineering.

Regarding the civil part, several pieces of research investigated the shape and material of potential masonry elements that could work with drones [3] [4] [5] [6]. They developed several types of drone-compatible elements such as the dricks and droxels shown in Figure 1.2. These elements can compensate the imprecision that the drones introduce compared to a human worker. In particular, these construction units can work with a maximum error of 5 cm from their theoretical position, as shown in Figure 1.3.

Moreover, they developed drone-compatible lifting systems and tested the methods with a manual operator. The tests with the remote controller proved the feasibility of the blocks in constructions projects. A picture of the experiment can be seen in Figure 1.4.

Several pieces of research investigated the automation part simultaneously. The focus was on developing drone-compatible positioning systems that would provide better accuracy than a GPS module.

In 2017, Thibault Jacques and Fréderic Kaczynski studied the feasibility of using a camera-based local positioning system. Using computer vision, they analysed diverse approaches to locate a brick and examined both precision and performance.

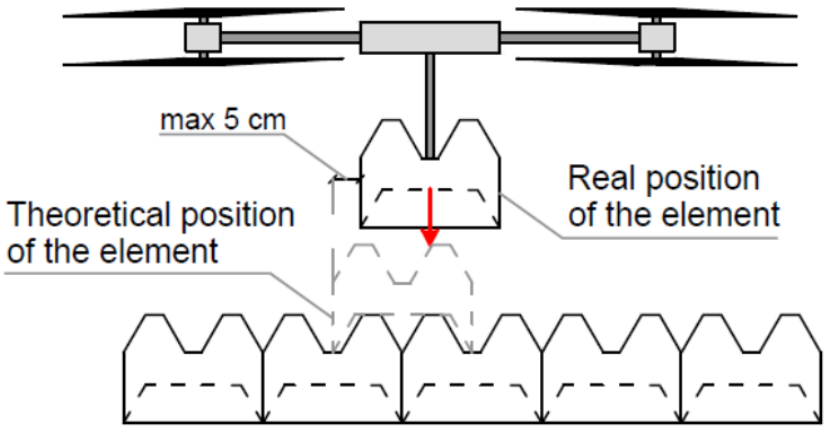


Figure 1.3: Installation of a block despite the imprecision of the drone

Figure 1.4: Manual test of a drone manipulating bricks

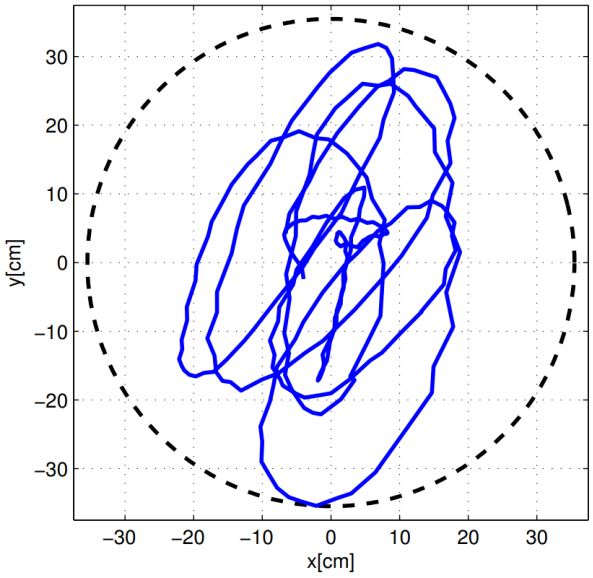


Figure 1.5: Trajectory of the drone during the stationary phase (48 sec)

They concluded that ArUco markers were a correct solution thanks to its accuracy and low processing power consumption. The error stayed below 3cm for a distance of two meters and a tag of 6.5cm-by-6.5cm[7]. These results validated the system for future implementation as a local positioning system.

From 2018 to 2019, several students developed a global positioning system using simultaneously three technologies: A Total Station (precision of 1 cm), a UWB (precision of 20 to 40 cm), and an Intel RealSense Depth camera (precision of over 50 cm). They developed a system that dynamically switches for the most accurate source available [8], [9], [10].

In 2019, the same students made the initial steps towards developing a guidance system capable of commanding the drone autonomously. However, the results were not great, as the drone presented difficulties to maintain a stationary position without drifting. Figure 1.5 shows the drift of the drone from the desired location for 48 seconds. The drone was only capable of staying within 35 cm of the desired position.

The guidance system commanded the drone to achieve the desired positions only using global positioning methods. The implementation did not include the local camera-based system and was far too basic. This project aims to fix these problems by creating a program capable of improving its stability, security and incorporating the local positioning system capable of working with an error smaller than 5 cm.

1.2 Objectives

This project aims to develop a vision-based guidance system capable of commanding a drone to build a masonry structure. It will use as a foundation the knowledge generated by previous projects and aim to fix the issues that former students encountered. We can establish three specific objectives for this thesis:

First, test the reliability of a drone-compatible local positioning system using ArUco markers. In particular, examine the relation between error, distance and marker size.

Second, examine different control approaches to position the drone respect to an ArUco tag and obtain a method that guarantees a maximum deviation of 5 cm from a given target.

Third, build a guidance system: a framework capable of combining global and local position methods to find bricks and their deposit position autonomously. Also, develop complementary programs to aid future development.

Chapter 2

Hardware Architecture

2.1 Introduction

A drone is a non-piloted aircraft capable of operating with distinct degrees of autonomy. These vehicles can perform an extensive list of tasks, from package delivery to leisure, and thus, massively vary on size and type.

This chapter aims to describe the system adopted in this project as well as its essential parts, its global positioning systems and its connections.

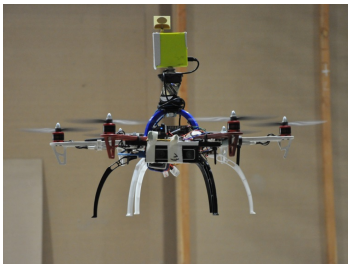
2.2 Selection of a suitable drone

Builder drones are a particular type of drone suitable for construction processes. They shall be able to transport construction material of considerable weight and have an accurate positioning system. The team of researchers in the EPL developed three drones suitable for diverse stages of development:

- **Big drone:** The big drone is a quadcopter with a personalised chassis that can move up to 40 kg of weight. Its usage is restricted for a system entirely robust. Figure 2.1a shows the big drone.
- **Medium-sized drone:** The medium-sized drone is a quadcopter that can carry weights ranging from 5 to 10 kg. Its usage is restrained to systems previously validated with the small-sized drone. Its dimensions allow it to test systems in a more realist scenario than the small drone. Figure 2.1b shows the medium drone.
- **Small-sized drone:** The small-sized drone is a hexacopter that can carry weights of around 1 kg. Its lightweight, 3D printed design, makes it a fit

(a) Big drone

(b) Medium drone



(c) Small drone

Figure 2.1: The drones of the UCLouvain

solution for testing novel systems. However, the results obtained with it should consider its modest proportions. Figure 2.1c shows the small drone.

Due to the new technology developed in this project, the small drone was early adopted as the project's drone regardless of its final availability limitations experienced at the end of the project. The following chapter will explain the parts of the drone.

2.3 Parts of the drone

Before developing a guidance system for the drone, it is essential to understand its diverse parts and the relations between them. For this thesis, we will explain the specific parts used in the small-sized drone, although different configurations are possible.

2.3.1 Flight controller

The flight controller is the unit in charge of handling the stabilisation of the vehicle and controlling its movement. It collects data from its internal measurement units (accelerometer, magnetometer, and gyroscope) and receives commands and information from other external sources (such as the onboard computer).

The flight controller is also in charge of processing the information it receives and sending the appropriate answer to the electronic speed controllers (ESC). These last ones will generate the required three-phase current to regulate the speed of the motors.

The flight controller used in the small-sized drone is a PX4 Cube. This unit is an open-source flight controller with a robust flight stack that provides flexible tools for developers. It has the advantage over other controllers of having an extensive list of compatible communications systems, peripheral integrations, and power management solutions. Figure 2.2 shows the Pixhawk 2 cube.

Figure 2.2: The PX4 Cube

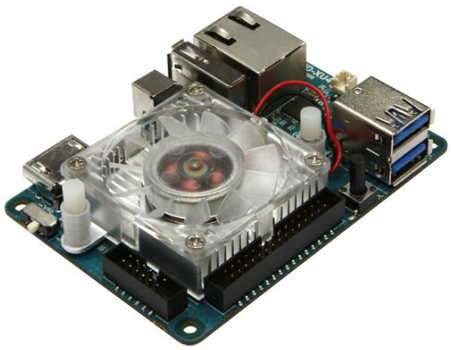


Figure 2.3: The Odroid XU4

2.3.2 Onboard computer

The onboard computer is the unit responsible for commanding the PX4 controller. It is in charge of handling the positioning systems, optimising the flight and guaranteeing security.

The small-sized drone has an Odroid XU4, an onboard computer with 2GB of RAM, an 8-core ARM Processor, and 3 USB ports. It does not dispose of an integrated WIFI card, and thus a WIFI dongle was installed. The system is running Ubuntu and the ROS framework to communicate with the different systems. It can be seen in Figure 2.3.

2.3.3 Onboard camera

The onboard camera is the unit in the drone responsible of capturing images during flight. Chapter 4 explains its integration as a local positioning system.

The unit mounted in the small-sized drone is a Logitech HD Webcam C270. This model is light enough to be suitable to be equipped into the small drone. It has a resolution of 720p, a field of view of 60 degrees and is able of transmitting at 30 fps. It is connected via USB to the Onboard computer where it sends the

Figure 2.4: Logitech HD Webcam C270

real-time recording.

2.3.4 Global positioning units

Multiple interchangeable units can be incorporated in the chassis to global position the drone. Some examples include a prism (with a Total Station), a UWB tag or a GPS. The functionality of these systems is explained in section 2.4.

2.3.5 Other systems

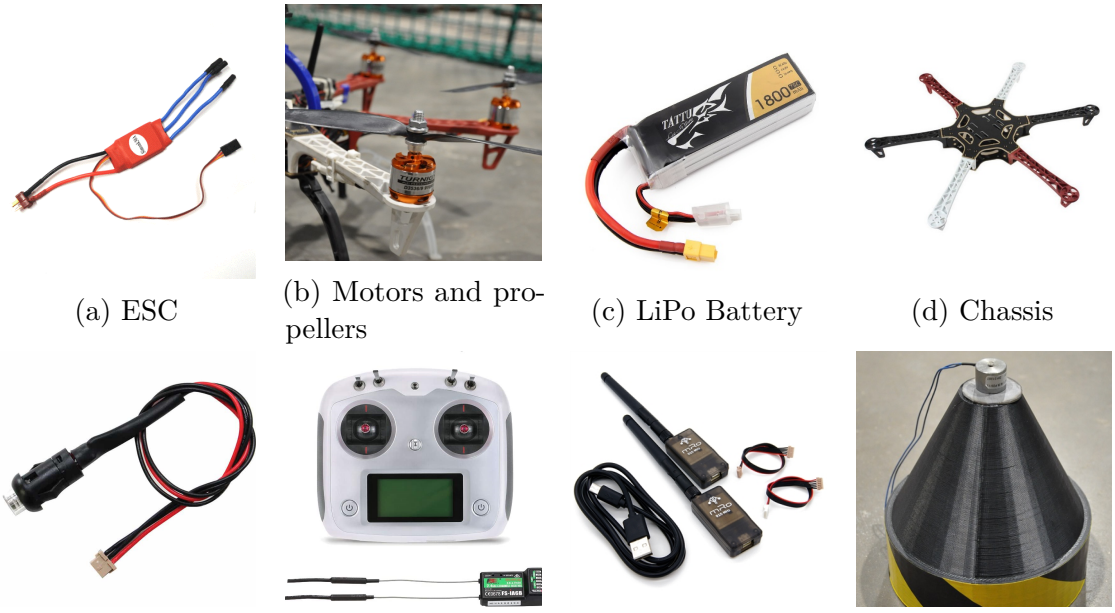
There are other complementary units necessary for the correct performance of the drone. A summary of their functionality is provided in the following list.

- **Electronic speed controllers (ESC):** An electronic speed controller is an electronic device capable of interpreting signals from the flight controller and altering the speed of the rotors accordingly. The ESCs are connected to the battery and operate by reading the voltage received from the flight controller and providing the three-phase current to the motors. There are as many ESC as motors in the drone. Figure 2.5a shows an image of an ESC unit.
- **Motors and Propellers:** Motors receive the electric current of the ESC units to spin the propellers. By moving the propellers at high speeds, they can generate the force to lift the drone. They are likewise responsible for inducing torque that needs to be compensated to avoid the rotation of the drone around itself. In the case of the small-sized drone, a hexacopter, three motors spin clockwise, and the other three spin anti-clockwise. The motors used are brushless motors due to their higher relation power/weigh. Figure 2.5b shows an image of the motors and propellers used in the hexacopter.

- **Battery:** The battery provides energy to the ESC systems and the flight controller. A LiPo battery is used for its large storage capabilities. However, they suffer from limited lifespan and long recharge times. Figure 2.5c shows an image of a LiPo Battery.
- **Chassis:** The chassis serves as a support for all the items on the drone. It can have a printed circuit that transmits the battery to the ESC. In the case of the small-sized drone, it is a modified chassis F550 with new parts made of 3D printing. Figure 2.5d shows an image of a drone chassis F550.
- **Security switch:** This switch is connected to the PX4 controller and functions as an additional security measure. It has to be pressed every time to start the arming procedure. Figure 2.5e shows an image of a PX4 security switch.
- **Transmitter Control and receiver:** The receiver allows commanding the drone with a controller. It enables manual flight and can help check the validity of hardware and the positioning systems before attempting to use an autonomous system. Figure 2.5f shows an image of a Transmitter Control and a receiver.
- **SiK Telemetry Radio:** A SiK Telemetry Radio is a small open-source radio platform that allows creating wireless MAVlink connections between the drone and the ground control station. It works in ranges of 300 meters, and it allows to send commands to the PX4 without the Odroid. It can also log information using QGroundControl. The units used on the small-sized drone are mRo SiK Telemetry Radio. One unit should be attached to the flight controller and the other to the computer. Figure 2.5g shows an image of a pair of mRo SiK Telemetry Radio.
- **Lifting System:** The lifting system consists of an electromagnet activated through an electronic card connected to the onboard computer. The output provided by the Odroid GPIO pins is not sufficient, and thus an XU4 Shifter Shield is added. This system will provide adequate current flow to command a switch allocated int the electronic card to activate the current flow from the battery. Figure 2.5h shows an image of the lifting system.

2.4 Global positioning system

The position of the drone is described by its three spatial coordinates (x_G, y_G, z_G), and its three orientation coordinates: $(roll, pitch, yaw)$. As seen in Figure 2.6, $roll$,



(a) ESC (b) Motors and propellers (c) LiPo Battery (d) Chassis

(e) Security Switch (f) Transmitter (g) Telemetry Radio (h) Lifting System

Figure 2.5: Complementary elements of the drone

pitch and *yaw* are defined as the rotation around the x, y and z-axis of the drone, respectively.

The most common approach to locate a drone is the use of GPS. This technology provides geolocation by continuously receiving radio signals containing time and data of a minimum of four satellites. However, this technology has the disadvantage of being very susceptible to disturbance, not providing acceptable accuracy and not working in GPS-denied areas.

To get around this problem, previous researches investigated the use of three different global positioning methods: UWB, Total Station, and an Intel Real Sense camera [8], [9], [10].

2.4.1 Total Station

The total station is a positioning system that works with infrared signals at multiple frequencies. The signals are reflected on a prism in the top of the drone. When the message returns, the station can determine the distance between the vehicle and itself. Figure 2.7 shows the setup of the positioning system.

This system provides excellent accuracy, with an error below 1 cm. However, if the drone moves to fast, or the drone gets out of direct sight from the station, the station will lose track of it. The low robustness over speed does not make the system ideal when an exact global precision is not required. Due to this, and

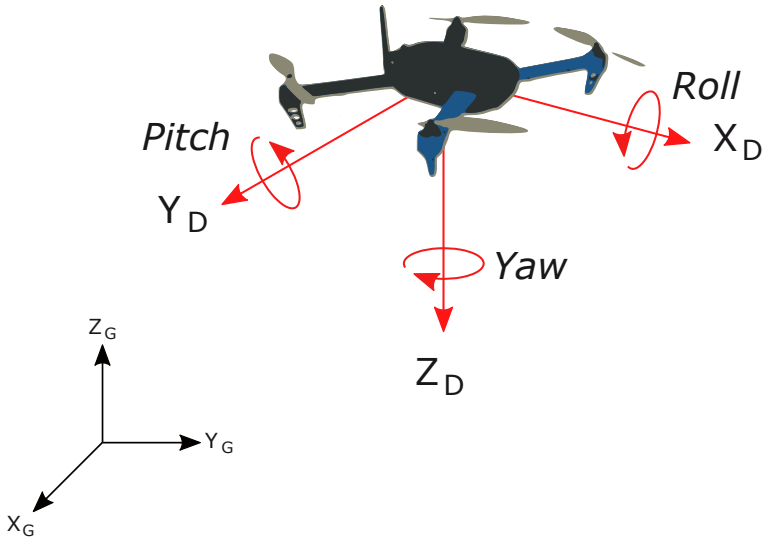


Figure 2.6: Coordinate system of the drone

availability issues, this method was not used in this project.

2.4.2 Intel RealSense Camera

This positioning system uses an Intel RealSense Camera to provide position and orientation. Specifically, the T265 model has two black-and-white cameras with a field of view of 163 degrees. By internally processing the images, it is possible to obtain the global position of the drone. Figure 2.8 shows an image of the T265 model.

When the drone was static on the spot, the error was lower than 1 cm. However, when it moved, the error was higher than 50 cm. Due to unavailability and the system not fitting to the project, this method was not used in it.

2.4.3 UWB

Ultra-wideband (UWB) is a radio technology that uses weak radio pulses at high-speeds. These signals are resistents to disturbance and provide high accuracy. They have the advantage of working in GPS denied areas with very light consumption in real-time functionalities. They are, however, unable to carry much information

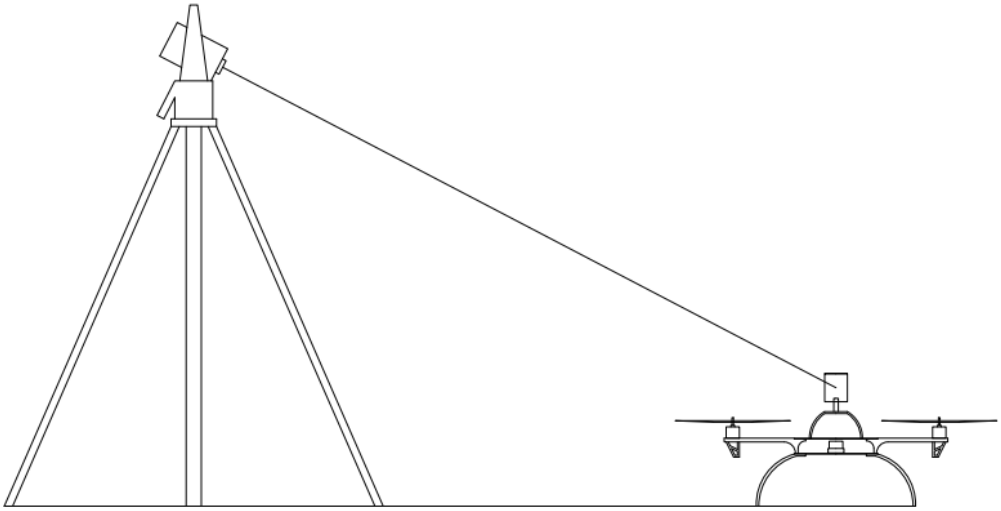


Figure 2.7: Total Station setup

Figure 2.8: Intel Camera

Figure 2.9: UWB antenna

and limited to operation distances between 10 and 100 meters.

There are multiple alternatives to use the UWB as a positioning system. One way to approach the problem is to use the Two-Way Ranging method. This process works by continuously sending a signal from an antenna mounted (the "tag") to the antennas mounted around them (the "anchors"). Once the message is received, the anchor will timestamp it and send it back to the tag. Finally, the distance is calculated by multiplying the time of flight of the UWB signal with the speed of light.

Several EPL students developed a positioning system using four UWB tags from decawave [11]. They located three anchors around the laboratory and one in top of the drone. Then, they computed the position through a mathematical problem called trilateration that allowed to position the tag. Figure 2.10 shows the setup for the trilateration problem.

The precision obtained ranged from 20 cm in x, y to 40 cm in z. These results are not ideal for executing precision tasks, but its robustness and ease of use make it an excellent solution to control the drone in GPS-denied areas and when global high precision is not required.

For this project, we will only use the UWB for being the more reliable source of information. This system does not provide information about the drone's orientation during flights, but it can be obtained using the local positioning system.

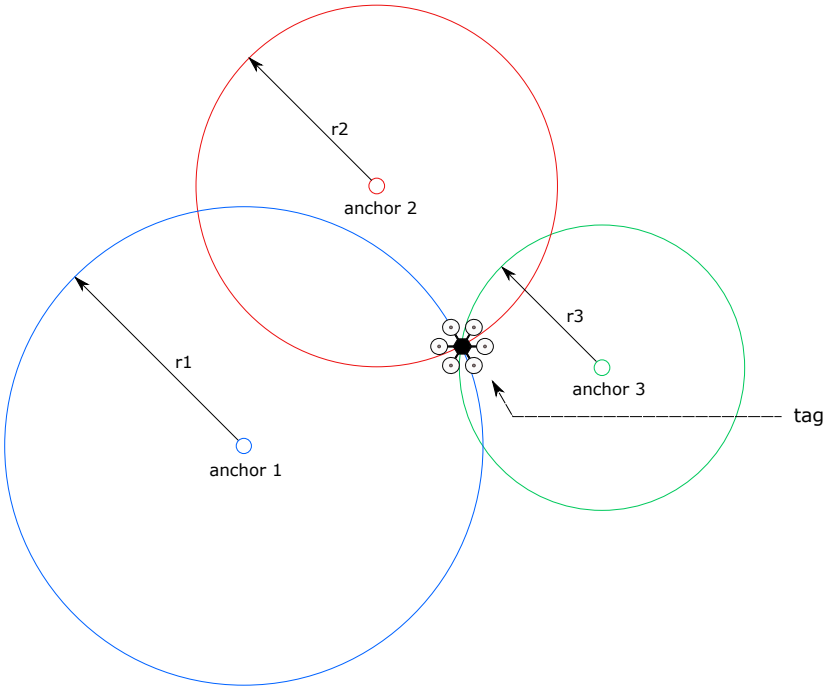


Figure 2.10: The trilateration problem

2.5 Hardware overview

An overview of the connections of all the units of the small-sized drone can be seen in Figure 2.11.

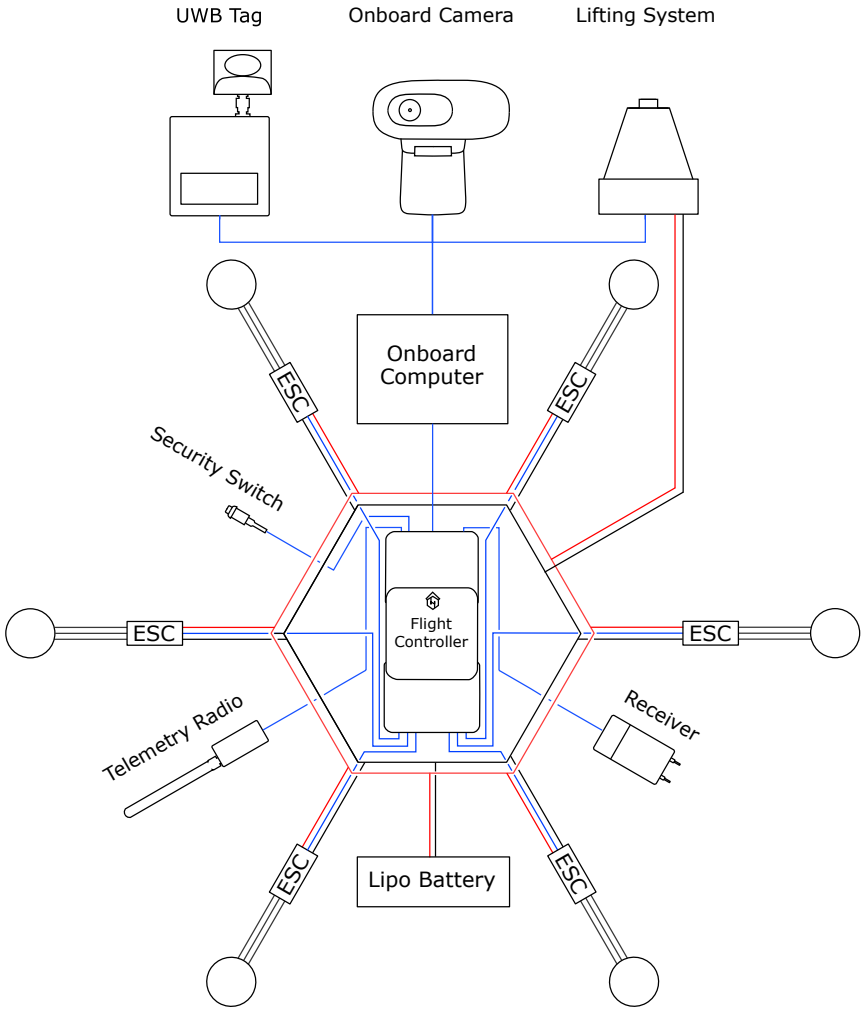


Figure 2.11: Hardware Overview

Chapter 3

Framework Architecture

3.1 Introduction

A standard transmission system between units is necessary for enabling the communication between the onboard computer and the flight controller. Furthermore, a global language allows simulation experiments which can help to test the behaviour of the drone in a safer scenario.

In this chapter, we will explain how we handle the communication between all systems using ROS. We will also explain the functionality of SITL PX4 simulation developed using Gazebo and its more important parts.

3.2 ROS

The flight controller of this project, the PX4, uses MAVlink (Micro Air Vehicle Communication Protocol) as a communication protocol. This protocol is designed to work with the drone ecosystem and its build towards transmission speed and security.

To be able to convert messages to the language of the controller, we use ROS as middleware (Robotic Operating System), running on top of Ubuntu Mate 18.04. ROS provides a flexible framework that allows the development of robot software. It is a collection of tools, libraries, and conventions that simplify creating complex and robust systems.

ROS framework works as a modular system where each processing task is executed in independents nodes. In our drone, we will have three different nodes besides the master node: a global positioning node, a local positioning node and the main guidance system node.

- **Global positioning node:** Node in charge of sending the global pose of the drone using any of the methods described in section 2.4.

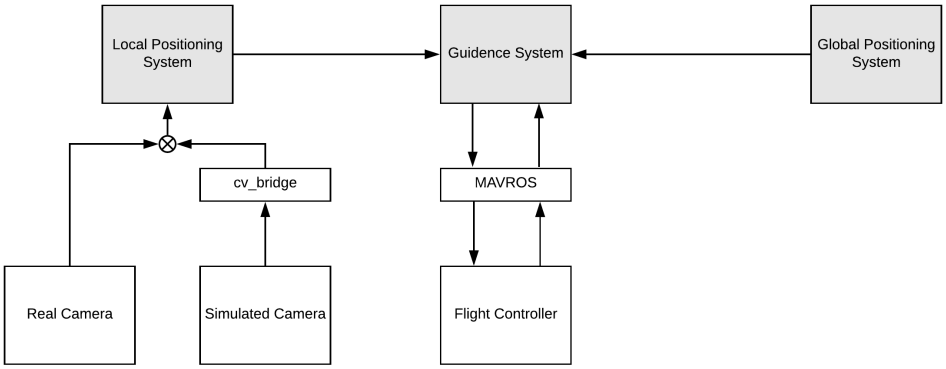


Figure 3.1: Nodes connections

- **Local positioning node:** Node responsible for processing camera images and sending the local pose. Developed in Chapter 4
- **Guidance system node:** Node in charge of commanding the position, speed and modes of the drone. Developed in Chapter 5

This way of operating allows easier manipulation of the code, and depurating errors more efficiently. All those nodes communicate with each other using a common communication channel and share information such as flight status. When a node needs to send information it *publishes* to a topic and when a node needs to obtain information it *subscribes* to a topic. Neither publishing or subscribing is restricted to one single node. That is to say, multiple nodes can subscribe to a topic and publish to it. Nodes can also communicate using services. This method consists of a client sending a request and waiting for an answer to the service.

ROS framework is open-source, which allows for the usage of libraries and drivers developed by the community, significantly increasing the speed of writing code. In particular, an essential library used in this project is MAVROS.

MAVROS is a Library that allows communication between MAVlink and ROS. It provides the necessary drivers that translate commands from the onboard computer to the PX4 language. It also enables the use of ground control stations such as QGroundControl.

Another essential library utilised is OpenCV. OpenCV is a library oriented to real-time computer vision with extensive documentation and support from the community. OpenCV can work with real and virtual images, but these last ones require the utilisation of the CVbridge library to "translate" the images to ROS. Figure 3.1 shows a simple diagram with the nodes connections.

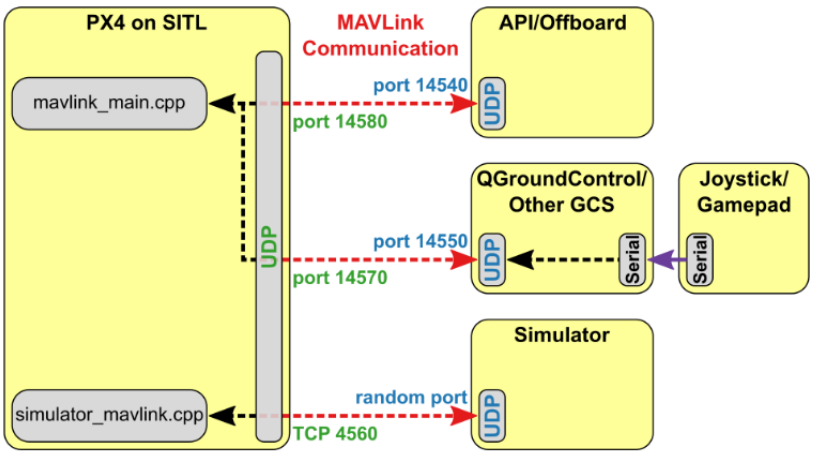


Figure 3.2: Schema of the simulation

3.3 Simulation

The PX4 code can control a computer modelled drone in a simulated environment, just as it will command a real-life drone. This way, it is possible to use a ground station, the offboard API or a radio control to test the drone. Thus, using the simulation allows a quick and safe way to validate automated systems before attempting a real-life flight.

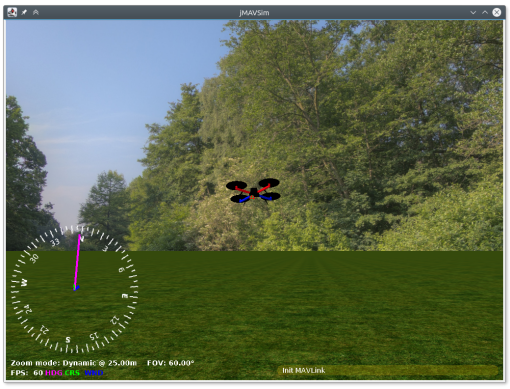
The simulation of the PX4 can be performed in two distinct ways:

- **Software in The Loop simulation (SITL):** In this type of simulation the flight stack software is directly run in the computer (can be run locally or in another computer connected to a local network).
- **Hardware in The Loop simulation (HITL):** In this type of simulation, the flight stack is run in a real PX4 board.

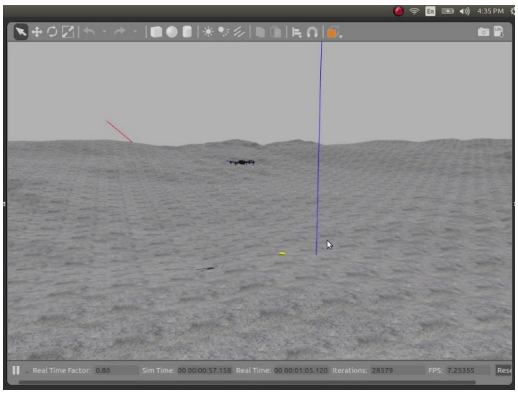
For this project SITL simulation was used. During SITL, sensor data comes from the simulation with communications using the UDP protocol. Figure 3.2 shows the SITL simulation communicating with the PX4 software.

There are several simulators which work with the PX4 firmware. In this project, two were studied: jMAVSim and Gazebo. [12]

- **jMAVSim Simulator:** jMAVSim is a simple simulator for multirotor systems. It has the advantage of being easy to set up for testing simple commands such as taking off, landing, or achieving a setpoint. It also enables to test the system under various fails conditions (for example, loss of position data) [13]. Figure 3.3a shows the simulator environment.



(a) jMAVSim simulator



(b) Gazebo simulator

Figure 3.3: Simulators compatibles with the PX4

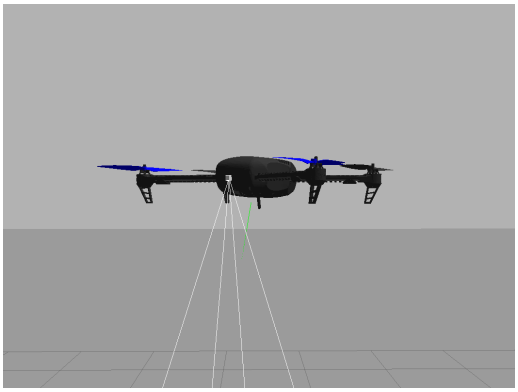
- **Gazebo Simulator:** Gazebo is a robust 3D simulator that can perform all jMAVsim features and provides the framework to develop more sophisticated simulations, including computer vision [14]. Figure 3.3b shows the simulator environment.

Initial tests were performed with the jMAVSim simulator for its simplicity. However, Gazebo was later adopted because it provided the framework for integrating a simulated camera.

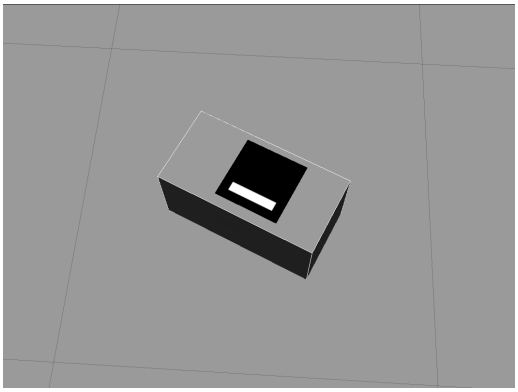
To describe the elements of the simulation, Gazebo utilises XML files called SDF (Simulation Description Format). The two more important components for this project were the simulated drone and the simulated ArUco Brick:

- **Simulated Drone:** The drone used in the simulation is the quadcopter iris, a model publically available in the gazebo models repository. It has an adapted virtual camera sensor from the gazebo models repository and a custom plugin to publish the ROS images into a ROS topic. The iris drone also includes a GPS module that sends the global position to the controller. It generates noise similar to the one typically found in real systems. Figure 3.4a shows an image of the simulated drone.

- **ArUco Brick:** The tag used in the simulation is an ArUco Tag attached to a Gazebo cube. It enables the simulation of the local positioning system. Figure 3.4b shows an image of the simulated block.



(a) Simulated drone



(b) Simulated block

Figure 3.4: Elements of the simulation

Chapter 4

Local Positioning System

4.1 Introduction

Picking up and depositing bricks requires a positioning system capable of correctly locating the drone. Global positioning systems, such as the Total Station or the UWB, are not capable of providing this information as the position of the construction units is unknown to them. This is why the construction of a system capable of finding bricks is essential.

This chapter aims to obtain a vision-based local positioning system capable of identifying ArUco markers. It will cover the selection of the system, the camera model and the ROS integration.

4.2 Selection of a system

For developing a vision-based local positioning system, we will use the Logitech onboard camera described in section 2.3.3 . Using cameras has the advantage of being very simple to set up as they only require an initial calibration (See calibration section 4.3).

Using an onboard camera transforms the problem into a computer vision one, and there are several ways to approach it. Jacques Thibault and Frederic Kaczynski compared ArUco markers against other systems in their research for a suitable system for builder drones [7].

Their results concluded that ArUco Markers had higher accuracy and performance with less memory consumption compared to other systems. Besides, they have the advantage of not restricting the system to one specific class of brick (such as using convolutional neural networks or feature detection would). This is why this project would focus on ArUco Markers to expand on their tests and develop the local positioning system.

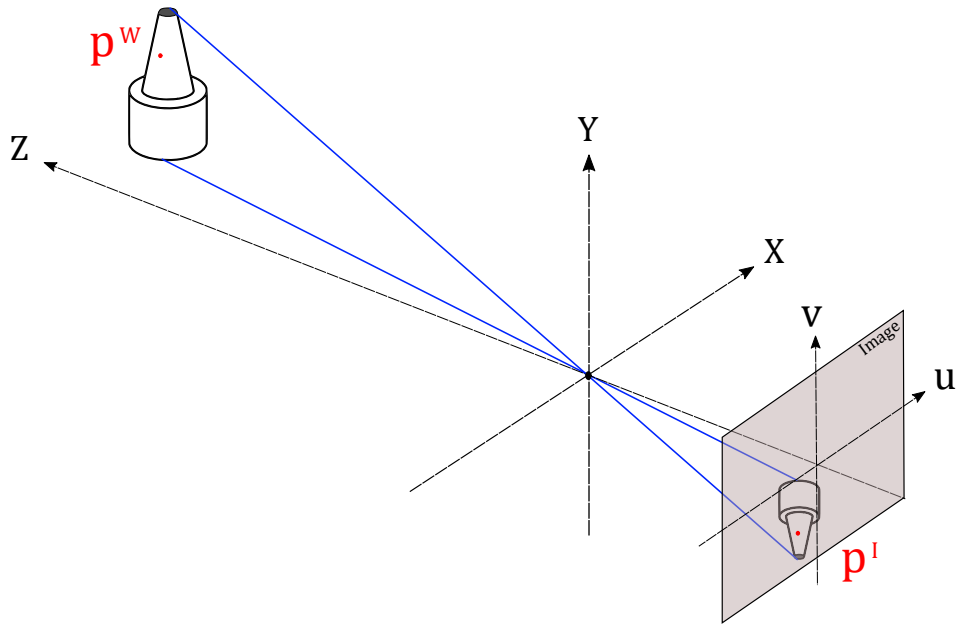


Figure 4.1: The pinhole model

4.3 Camera model

Cameras are devices capable of obtaining a 2D image from the real world. The most common model used to describe these devices is the pinhole camera. In this model, all light goes through a small infinitesimal aperture and projects an inverted image on the sensor.

This model establishes a perspective transformation between points in the 3D world $p^W = (X, Y, Z)$[m] and their projection into a 2D image $p^I = (u, v)$[pixels] as seen in Figure 4.1. This relation is defined in Eq. 4.1.

$$p^I = A\,[R|t]\,p^W \qquad \text{(Eq. 4.1)}$$

Where $[R|t]$, the matrix of extrinsic parameters, is the joint rotation-translation matrix that transforms points from the 3D world coordinate system $p^W = (X, Y, Z)$ [m] to the 3D camera coordinate system $p^I = (x, y)$ [m] (see Eq. 4.2). In other words, it describes the position of the camera around the scene.

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \quad \text{(Eq. 4.2)}$$

In the other hand, A is the matrix of intrinsic parameters or camera matrix. It contains four coefficients that define the specific camera model. These models are the focal length (f_x, f_y) and the optical centres (c_x, c_y) calculated in terms of pixels. These coefficients remain constant if the lens is not modified. See Eq. 4.3:

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \quad \text{(Eq. 4.3)}$$

If we combine Eq. 4.1 with Eq. 4.2 and Eq. 4.3 we obtain an extended look of the equation: Eq. 4.4.

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \underbrace{\begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}}_{Camera\,Matrix} \underbrace{\begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{bmatrix}}_{Extrinsic\,Matrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \quad \text{(Eq. 4.4)}$$

However, the relation in Eq. 4.4 is not totally accurate. In a real camera lens, there are imperfections that generate a distorted image of the original picture. Using the Brown-Conrady model, we can correct the distortions generated by the camera [15]. This method decomposes distortion into two sources: radial distortion and tangential distortion.

Radial distortion is caused by flawed radial curvature and makes straight lines curve as they move away from the centre of the image. This phenomenon happens when light rays start bending as they get distant from the optical centre. Eq. 4.5 shows the relation between the distorted x, y obtained by the camera and the fixed $x_{corrected}, y_{corrected}$.

$$\begin{aligned} x_{corrected} &= (x - x_c)(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) \\ y_{corrected} &= (y - y_d)(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) \end{aligned} \quad \text{(Eq. 4.5)}$$

Where (x_c, y_c) is the distortion center, r is the distance to the center of radial distortion (calculated as $r = \sqrt{(x - x_c)^2 + (y - y_c)^2}$) and k_1, k_2, k_3 are the radial coefficients to find.

On the other hand, tangential distortion occurs when the lens and the image plane are not parallel, and the distance is closer than expected. It is governed by

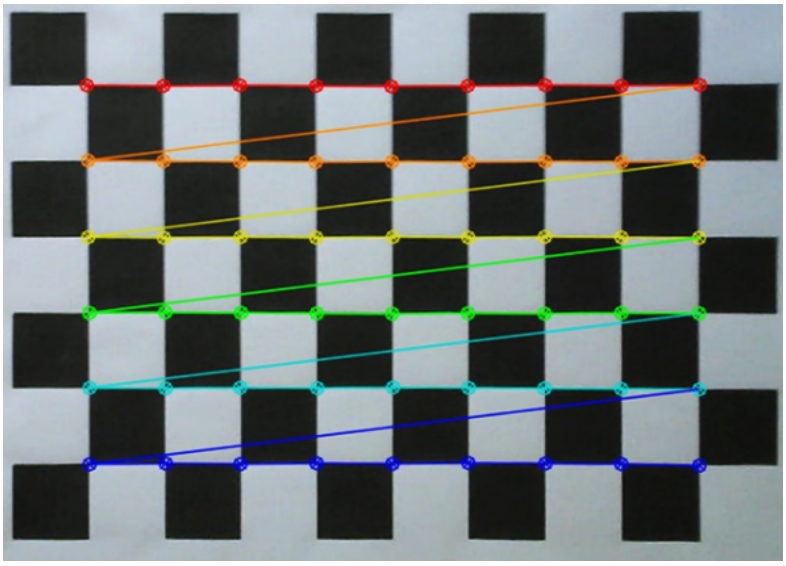


Figure 4.2: Image of the chessboard during calibration

Eq. 4.6 where p_1 and p_2 are the tangential coefficients to find.

$$\begin{aligned} x_{corrected} &= x + [2p_1(x - x_c)(y - y_d) + p_2(r^2 + 2(x - x_c)^2)] \\ y_{corrected} &= y + [p_1(r^2 + 2(y - y_d)^2) + 2p_2(x - x_c)(y - y_d)] \end{aligned} \qquad \text{(Eq. 4.6)}$$

Neither of these coefficients $(k_1, k_2, k_3, p_1, p_2)$ alters depending on the scene. They remain constant, and they need to be found in a process called camera calibration. To obtain the values of the distortion coefficients we need to use images of an object with known geometry.

OpenCV offers support for calibration with a chessboard [16]. Taking 30 pictures of a chessboard with the camera, we can determinate its values and calibrate it. Figure 4.2 shows the chessboard during calibration in this project.

4.4 ArUco library

ArUco library is an Open Source Computer Vision library aimed at Augmented Reality applications developed by the Universidad de Córdoba. It works by identifying ArUco markers and obtaining information from them.

An ArUco marker is a square-shaped matrix barcode consisting of a black border to help detection and a binary matrix code describing its ID. It has the advantage of being easily identifiable through computer vision, working with open-source libraries and performing rapid and accurate pose estimation [17] [18]. Figure 4.3 shows a collection of ArUco Markers.

Figure 4.3: Example of ArUco markers

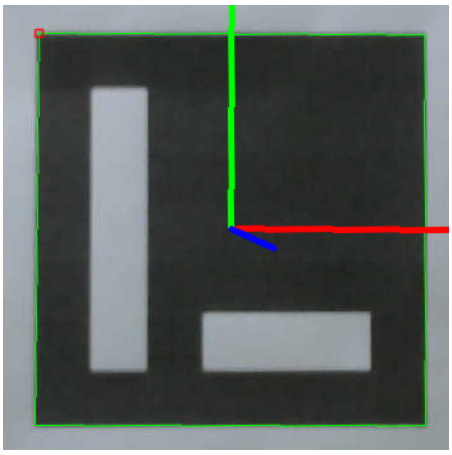


Figure 4.4: Identified ArUco Marker

4.5 Marker's detection

In order to obtain the position and orientation of the drone, we used the original ArUco libraries to obtain the markers. For the real camera, we printed the markers in conventional paper. For the virtual camera, these markers were inserted in the simulation (see section 3.3).

The detection algorithm uses a built-in function of the OpenCV library: *detectMarkers()* [19]. This function returns a list of detected markers from an image containing ArUco markers. The algorithm has two major parts:

1. First, detection of marker candidates occurs. The image is thresholded to find square shapes that could be possible markers.

2. Second, their internal codification is analysed to see if they belong to the ArUco library. If they do not belong to the dictionary, it rejects the candidate.

For each marker identified, a vector of its four corners is returned as well as its integer identifier. Figure 4.4 shows a picture of an ArUco identified.

4.6 Pose estimation

To perform camera pose estimation, we use the built-in function *estimatePoseSingleMarkers()*. It takes the coefficients of camera model as inputs, as well as the corners obtained from *detectMarkers()* and the length of the markers' side.

In return, this function outputs the translation vector and rotation vector that transforms points from the marker frame to the camera frame. In other words, the translation and rotations vectors allow getting the position and rotation of the marker in the camera frame.

The rotation vector is just a more compact and optimised representation of the rotation matrix. Using the CV built-in function *Rodrigues*, it is possible to obtain the rotation matrix from the marker frame to the camera frame. This matrix contains the *roll*, *pitch* and *yaw* angles of the marker in the camera frame.

4.7 ROS integration

We build the local positioning system as a ROS node to integrate it with the guidance system. It will communicate with the guidance system node to obtain the ID to find. Once it receives a valid ID, it will start searching it in the image and publishing its relative position into a ROS topic. This information is an essential part of the search ID method explained in section 5.2.3

This system can work with real and virtual images, but in the case of the virtual camera, the library CVBridge converts the simulated images to compatible images for OpenCV (see section 3.2).

Algorithm 1 presents the basic functionality of the local positioning method

Algorithm 1 Local Estimation algorithm

```
1: procedure LOCAL_ESTIMATION
2:     frame ← video
3:     ID ← GuidanceSystem
4:     ID_list, corners_list ← DETECT_MARKERS(frame)
5:     if ID in ID_list then
6:         vector ← ESTIMATEPOSESINGLEMARKERS(dist, corner, tag_length)
7:         translation_vector, rotation_vector ← vector
8:         rotation_matrix ← RODRIGUES(rotation_vector)
9:         PUBLISH(Local Position)
```

It is essential to consider that the relative distance is calculated between the marker and the camera, not the marker and the drone. This implies that special care should be taken to position the camera and that a frame transformation should

be performed. This is not handled for the local positioning node, but rather, for the guidance system. In particular, the search ID function should be modified to include this transformation (this will be further discussed in section 5.2.3).

Chapter 5

Guidance System

5.1 Introduction

The guidance system is the system in charge of receiving the status of a vehicle and sending commands to control its movement and actions. It allows to close the control feedback and ensure the desired behaviour. Without a proper guidance system, the autonomous control of a vehicle is impossible, and its stability and security can be compromised.

In our case of study, the aim is to develop a guidance system in charge of controlling a builder drone. That means developing a system capable of managing the positioning of a drone as well as picking and positioning drone-compatible elements.

In this chapter, we will talk about the new positioning system, its parts, how it handles the control and its safety measures.

5.2 Overview of the mission

Building a masonry structure using a drone is a complex mission. We can simplify the task by reducing the guidance system to eight straightforward phases:

1. **Start system:** Initialise all systems for autonomous control and take off.
2. **Move to pickup area:** Use the global positioning system to move the vehicle to the pickup area.
3. **Pickup the brick:** Use the local positioning system to search the blocks and activate the lifting system.
4. **Move to deposit area:** Use the global positioning system to move the vehicle to a deposit area.

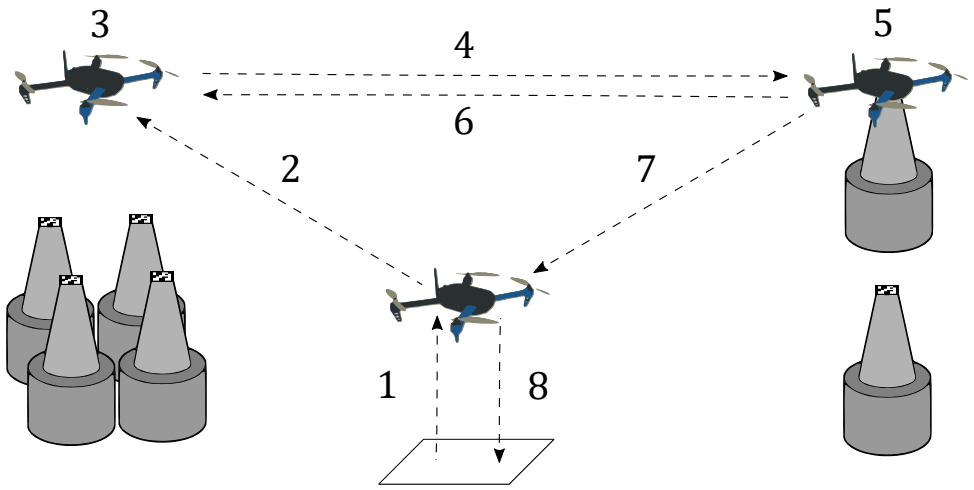


Figure 5.1: Overview of the mission

5. **Leave the brick:** Use the local positioning system to search the blocks deposit area and deactivate the lifting system.

6. **Return to pickup area and repeat:** Return to the pickup area and repeat phase 3 to 5 until the last brick is positioned.

7. **Return to home position:** If all bricks have been positioned return to the home position.

8. **End system:** Land vehicle and end the autonomous control.

In summary, the drone will navigate between the stack and the construction location, carrying and depositing the bricks and executing all previously described phases. This procedure will continue until positioning the last block. Figure 5.1 shows a visual overview of the mission.

To execute this mission with the drone, we first need to "translate it" to a language that it will understand. That is why we simplify the tasks into five more straightforward functions for the drone:

- **Start system (*mission*):** Phase 1, explained in section 5.2.1
- **Move to a global position (*X, Y, Z, Yaw*):** Phases 2, 4, 6 and 7, explained in section 5.2.2

- **Search ID (*id*):** Phase 3 and 4, explained in section 5.2.3
- **Set lifting system (*active/inactive*):** Phase 3 and 4, explained in section 5.2.4
- **End system:** Phase 8, explained in section 5.2.5

These five elementary operations are sufficient to create a functional guidance system. Now, the only thing lacking is the inputs of such functions. In our case, the following six elements completely define our mission:

- **Brick list** $[[id_{p1}, id_{d1}], [id_{p2}, id_{d2}, ...]]$ **:** List of all bricks with their identifier tag and the tag that identifies their deposit area.
- **Pickup position** p_1^p, p_2^p **:** The two opposites corners of the pickup area where the bricks are located.
- **Deposit position** p_1^d, p_2^d **:** The two opposites corners of the deposit area where the bricks will be located.
- **Pickup Scan height** h_{sp} **:** The height at which the drone will search for the bricks.
- **Deposit Scan height** h_{sd} **:** The height at which the drone will search for the deposit position.
- **Flight height** h_f **:** The general flying height of the drone.

For defining the position of the pickup/deposit areas and the flying heights, it is necessary to establish the centre of our coordinate system. In our case, the centre will always be placed on the initial position of the drone. Figure 5.2 shows the position variables in a diagram.

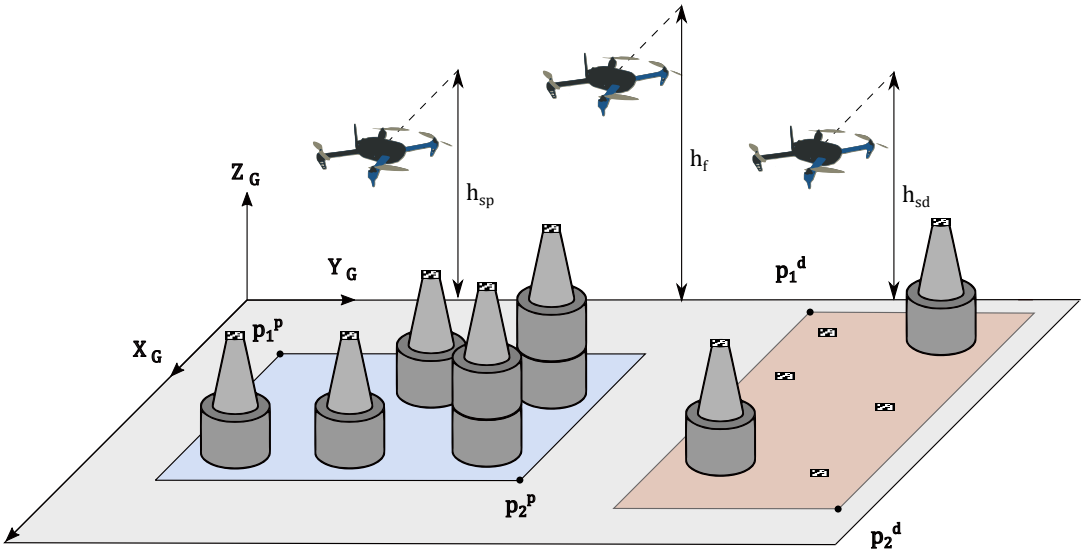


Figure 5.2: Overview of the variables

Finally, combining the six input variables with the five elementary functions it is possible to obtain a working system as shown in Algorithm 2.

Algorithm 2 Guidance System algorithm

```
1: procedure GUIDANCE_SYSTEM(mission)
2:     START_SYSTEM(mission)
3:     for all brick in bricks_list do
4:         pickup_position, deposit_position, brick_id, deposit_id ← brick
5:         MOVE_TO_GLOBAL_POSITION(pickup_position)
6:         SEARCH_ID(brick_id))
7:         SET_LIFTING_SYSTEM(active)
8:         MOVE_TO_GLOBAL_POSITION(deposit_position)
9:         SEARCH_ID(deposit_id))
10:        SET_LIFTING_SYSTEM(inactive)
11:    END_SYSTEM( )
```

The next sections will describe the functionality of the five elementary operations previously described.

5.2.1 Start the system

The guidance system runs in the onboard computer connected via serial cable to the flight controller. On execution, it will start as a ROS node and will attempt to load the mission file of the flight.

The mission file is a document containing all the necessary data for the execution of the program. The specifics of this information were explained in the previous section (see 5.2).

These parameters are transformed into a series of simpler commands and are presented to the user for its verification. Figure 5.3 shows an image of the response displayed to the user.

```
python guidance_system.py
[INFO] [...]: ========================================================================
[INFO] [...]: Guidance system starting
[INFO] [...]: ========================================================================
[INFO] [...]: The parameters of the mission are:
[INFO] [...]: List of Pickups ID: ['5', '3']
[INFO] [...]: List of Deposits ID: ['4', '6']
[INFO] [...]: Pickup Positions: [[1.0, 2.0], [2.0, 4.0]]
[INFO] [...]: Deposit Positions: [[1.0, 6.0], [2.0, 8.0]]
[INFO] [...]: Pickup Scan height: 1.2
[INFO] [...]: Deposit Scan height: 1
[INFO] [...]: Flight height: 2
[INFO] [...]: ========================================================================
[INFO] [...]: The mission programmed is the following:
[INFO] [...]: 1: Position [x: +1.000, y: +2.000, z: +2.000] accuracy = +0.200m
[INFO] [...]: 2: Especial command: Aruco Search ID: 5.0
[INFO] [...]: 3: Especial command: Activate Electromagnet
[INFO] [...]: 4: Position [x: +1.000, y: +6.000, z: +2.000] accuracy = +0.200m
[INFO] [...]: 5: Especial command: Aruco Search ID: 4.0
[INFO] [...]: 6: Especial command: Deactivate Electromagnet
[INFO] [...]: 7: Position [x: +1.000, y: +2.000, z: +2.000] accuracy = +0.200m
[INFO] [...]: 8: Especial command: Aruco Search ID: 3.0
[INFO] [...]: 9: Especial command: Activate Electromagnet
[INFO] [...]: 10: Position [x: +1.000, y: +6.000, z: +2.000] accuracy = +0.200m
[INFO] [...]: 11: Especial command: Aruco Search ID: 6.0
[INFO] [...]: 12: Especial command: Deactivate Electromagnet
[INFO] [...]: ========================================================================
[INFO] [...]: Is this OK? [y,n]
```

Figure 5.3: Mission information of the Start System

If the user does not accept this information, the program closes immediately. Otherwise, it starts the initial checks. First, it verifies that the folowing services and subscribers are working correctly with the flight controller using the MAVLink protocol:

1. **Services**
    - **Arming Service:** Service that change the arming state of the drone (arm/disarmed).

- **Set Mode Service:** Service that change the flying mode of the drone (Offboard, manual, auto.land, etc.)

2. **Subscribers**

   - **Drone State/Extended State Subscribers:** Subscribers that receive information about the status of the drone.
   - **Global Positioning System:** Subscriber that receives information about the global position of the drone.
   - **Local Positioning System:** Subscriber that receives information about the local position of the drone.

Then, using the information from the drone state subscriber, it checks that the vehicle is landed. Once verified, it initialise the publishers of the ROS node:

- **Position Publisher:** Publisher that sends setpoints to the drone.
- **Speed Publisher:** Publisher that commands the speed of the drone.
- **ID Publisher:** Publisher that sends the desired ID to the local positioning node.

Next, it initialise the threads that will handle sending the setpoints and reading the keyboard. Finally, it saves its origin position, switches to OFFBOARD mode, arms itself, and takes off. Figure 5.4 displays the information on the screen during these checks.

```
[INFO] [...]: ==============================================================
[INFO] [...]: Waiting for set_mode service
[INFO] [...]: Waiting for arming service
[INFO] [...]: Services OK
[INFO] [...]: Waiting for subscribers
[INFO] [...]: Subscribers OK
[INFO] [...]: Checking landed state
[INFO] [...]: Confirmed landed state
[INFO] [...]: All initial checks OK
[INFO] [...]: ==============================================================
[INFO] [...]: Started publishing setpoints
[INFO] [...]: Keyboard Thread ACTIVE
[INFO] [...]: Setpoints Thread ACTIVE
[INFO] [...]: Change mode to OFFBOARD ...
[INFO] [...]: Change mode to  OFFBOARD SUCCEED (operation time: 1.0 seconds)
[INFO] [...]: Change state to armed...
[INFO] [...]: Change state to armed SUCCEED (operation time: 1.0 seconds)
```

Figure 5.4: Preliminar checks of the Start System

5.2.2 Move to a global position

To move the drone to a global position, we send to the PX4 a setpoint composed of a target location ($r_{sp} = (X, Y, Z)$) and a target orientation (ψ_{sp} =$yaw\ angle$). This process is handled via the setpoint thread operating at a rate superior of 2 Hz to avoid the PX4 timeout of 500 ms between two offboard commands.

Then, the PX4 internal position controller compares the setpoint with the estimated values of the position ($\hat{r}$), and orientation ($\hat{\psi}$) obtained from the global position method (see section 2.4) and the PX4 internal sensors. This response computes the vertical thrust and attitude to position the drone.

The guidance system will keep sending setpoints until it verifies that the drone has arrived at the correct position. It will compare the target location with the actual position of the drone and see if the error is smaller than a user-defined threshold. If the drone reaches the setpoint during a limit of time, the program execution continues. Figure 5.5 shows a succesfull global positioning operation. If the drone fails to reach the setpoint, a timeout error occurs, and the drone executes the end system method (see the end system section 5.2.5)

Lower threshold values will result in a more slow and precise positioning. However, as we will see in the results section (see section 6.4), there is a limit to the maximum accuracy that the global position method and the PX4 control can obtain by themselves.

This method is reserved for general flying operation between the pickup, deposit and home area. The Search ID method will handle the positioning system for the more delicate operations (see 5.2.3).

```
[INFO] [...]: ========================================================================
[INFO] [...]: Mission n. 1 of 8
[INFO] [...]: Position command executed
[INFO] [...]: actual pose is   x: +0.011, y: −0.010, z: −0.015 [m]
[INFO] [...]: trying to reach  x: +0.003, y: +0.001, z: +1.982 [m]
[INFO] [...]: setpoint REACHED x: +0.003, y: +0.001, z: +1.982 (3.55 seconds)
[INFO] [...]: ========================================================================
```

Figure 5.5: Image of the move to a global position method

5.2.3 Search ID

This method allows the drone to scan the pickup or deposit area, identify a tag and descend in a controlled manoeuvre. It utilises the local position tool developed in Chapter 4. It consists of two phases: The scan phase and the descending phase:

1. **Scan phase:** The drone moves around the pickup or deposit area using the move to a global position function (See previous section 5.2.2). These areas

are defined by its two opposing corners set in the mission file. Figure 5.6 represents the scan phase.

The system communicates with the local positioning system and demands the ID to find. If a correct ID is not found, the mission is aborted. If the local position sees a correct ID, the drone will stop and start the descending phase.

2. **Descending Phase:** If a correct ID is detected, the local positioning system will start sending the relative position of the marker with respect to the camera and the drone will attempt to centre itself around the ArUco tag. For doing so, there are multiple approaches to this problem. During this thesis, we explored the problem in three different ways.

    - **Setpoint Error Control:** This method consists in using the global positioning system developed in section 5.2.2 to adjust the new setpoint repeatedly. That is to say, obtain the actual position of the drone, add the distance to the marker, and send it as a new setpoint.
    - **PI Speed Control:** This approach does not use the global positioning at all. Instead, it commands the drone's speed on its axis to obtain a precise positioning. We establish it as a PI controller.
    - **PI Setpoint Control:** This method is a variation of the setpoint error control with a PI controller. It modifies x,y and z setpoints accordingly to the distance error obtained by the local positioning system.

If using any of the explained methods the drone successfully descends and obtain an accepted error, the mission continues. Figure 5.7 shows a successful search operation. The failures scenarios are explored in the safety section 5.3.

As an important note, the user needs to define the position of the camera relative to the center of the drone. Without this information the drone will hover around the camera instead of the actual center of the drone.

```
[INFO] [...]: ========================================================================
[INFO] [...]: Mission n. 3 of 8
[INFO] [...]: Searching pickup ID : 3.0
[INFO] [...]: Found ID
[INFO] [...]: Local Positioning SUCCEED (operation time: 25.05 seconds)
[INFO] [...]: ========================================================================
```

Figure 5.7: Image of the search ID method

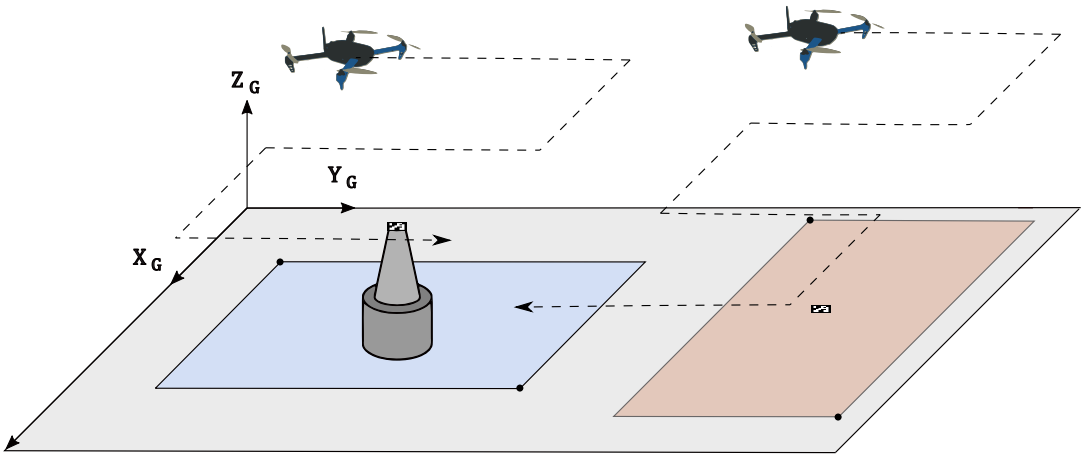


Figure 5.6: Representation of the search of the brick and the deposit position

5.2.4 Set the lifting system

Once the drone has positioned itself on top of a tag, the lifting system will activate or deactivate depending on the mission. This system commands the pins of the offboard computer to begin or interrupt supplying electric current to the electromagnet.

5.2.5 End the system

This system will execute if there are no more bricks to locate, the Move or Search methods fail, or the interrupt command is pressed. The drone will change to RTL mode, i.e. it will return to its home position and land on the spot. When the status subscriber informs that the drone is stable in the ground, it will activate the disarming service and close the guidance system node. Figure 5.8 shows the output of the system ending method.

The configuration of the RTL mode can be set using QGroundControl Tool. Specifically, the following four parameters define the RTL behaviour:

- **RTL_RETURN_ALT**: Safe altitude at what the drone will ascend before returning. The default value is 60 m.
- **RTL_DESCEND_ALT**: Altitude at which the drone will stop its initial

descend. The default value is 30 m.

- **RTL_LAND_DELAY**: Time that the drone will hover at the descend altitude (RTL_DESCEND_ALT) before start descending. The default value is 0,5s. If set to -1, the drone will not stop hovering.
- **RTL_MIN_DIST**: Minimum horizontal distance at which the drone will elevate to the safe altitude (RTL_RETURN_ALT) instead of directly flying towards the home position.

```
[INFO] [...]: ========================================================================
[INFO] [...]: Program ended, landing...
[INFO] [...]: change mode to AUTO.RTL ...
[INFO] [...]: change mode to  AUTO.RTL SUCCEED    (operation time: 1.0 seconds)
[INFO] [...]: Keyboard Thread CLOSED
[INFO] [...]: Setpoints Thread CLOSED
[INFO] [...]: LOG Thread CLOSED
[INFO] [...]: Landed, exiting...
[INFO] [...]: ========================================================================
```

Figure 5.8: Image of the end system method

5.3 Safety measures

Safety is a primary concern when working with robots. Losing control of an autonomous vehicle can compromise the security of itself and the people around them. Thus, it is essential to take into account all possible conditions that could suppose a dangerous situation.

In case of unexpected behaviour during the execution of the guidance system, it is possible to trigger an emergency landing using the keyboard. This command will trigger the ending sequence described in section 5.2.5. That is to say; the vehicle will go to its home position, change to the land mode, and proceed to disarm. Figure 5.9 shows the response of the system.

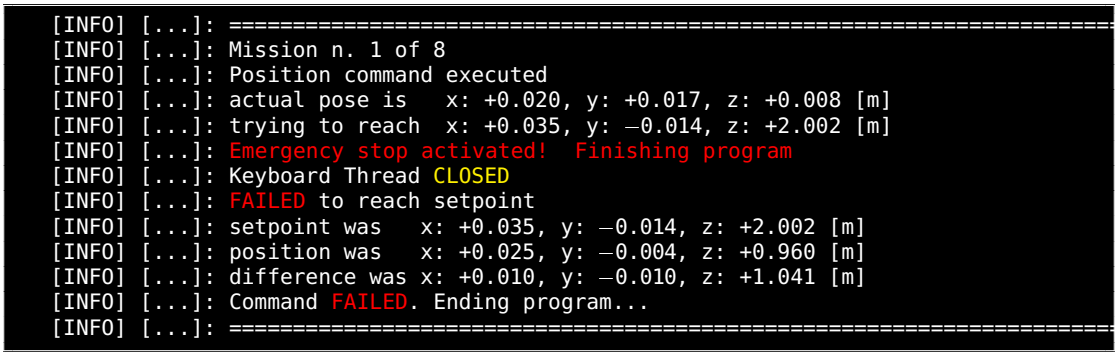


Figure 5.9: Image of the Keyboard Interruption

There are as well, multiple failure scenarios contemplated. Some of them are:

- Lost connection with subscribers. Activates the end mode. (Figure 5.10)
- ID not found in the provided area. Activates the end mode. (Figure 5.11)
- ID initially found and then lost. Sends the drone to the first position where the marker was spotted. (Figure 5.12)
- Services and commands timeouts. Activates the end mode. (Figure 5.13)

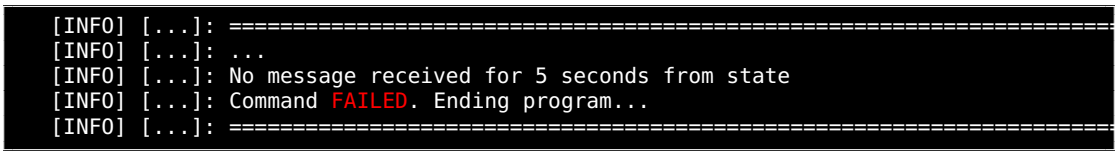


Figure 5.10: Lost connection with subscribers. System activates the end method.

Figure 5.11: ID not found in the provided area

```
[INFO] [...]: ========================================================================
[INFO] [...]: Mission n. 3 of 8
[INFO] [...]: Searching pickup ID : 3.0
[INFO] [...]: Found ID
[INFO] [...]: ID lost. Going to the first position where the marker was spotted
[INFO] [...]: ...
[INFO] [...]: ========================================================================
```

Figure 5.12: ID initially found and then lost. Sends the drone to the first position where the marker was spotted.

```
[INFO] [...]: ========================================================================
[INFO] [...]: Mission n. 3 of 8
[INFO] [...]: Searching pickup ID : 3.0
[INFO] [...]: Found ID
[INFO] [...]: Local Positioning FAILED (timeout: 50 seconds)
[INFO] [...]: Command FAILED. Ending program...
[INFO] [...]: ========================================================================
```

Figure 5.13: Command timeout. Activates the end mode.

In case of being a problem publishing data, the flight controller will be the one triggering an offboard lost failsafe. This event occurs when the PX4 detects that the command publishing rate is not consistent and has dropped below 2Hz. Consequently, the flight controller executes a series of commands to preserve safety. This commands can be personalised modifying the PX4 parameters using the QGroundControl tool. Some of the are of them:

- **COM_OF_LOSS_T:** Timeout in seconds before triggering offboard lost failsafe.
- **COM_OBL_ACT:** Mode to switch during the failsafe and not connected to RC Control (Land, hold or return).

The PX4 also allows for establishing a series of security parameters such as a safe perimeter or a maximum height. This commands should as well be personalised using QGroundControl before attempting to fly.

5.4 Complementary systems

Even if all software precautions are taken into account, hardware problems can still compromise the safety of any operation. For this reason, the following complementary systems were developed. They can help test the reliability of the drone iteratively before attempting to execute the guidance system.

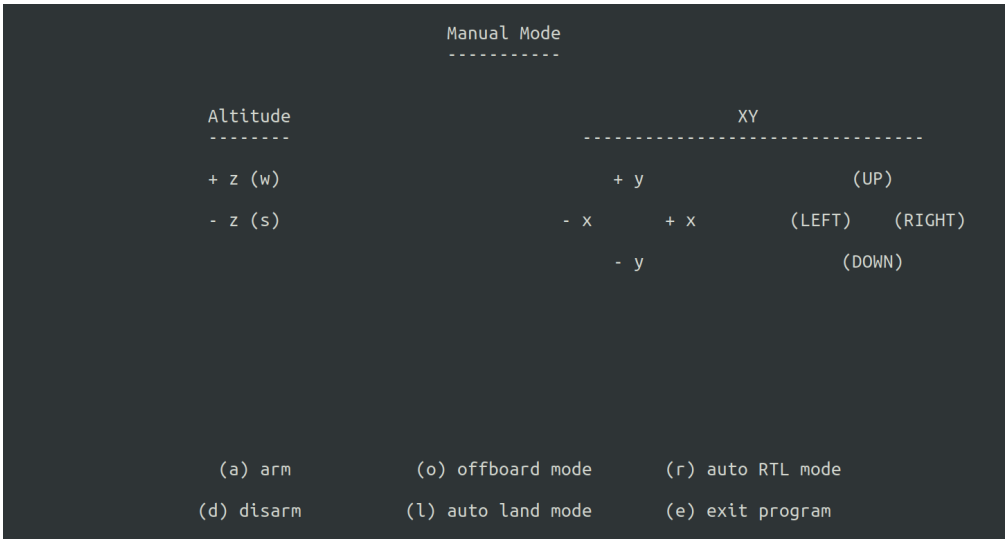


Figure 5.14: Manual mode interface

5.4.1 Manual System

The manual mode is a modified version of the global positioning function seen in section 5.2.2. It works by sending a constant flow of setpoints that the drone attempts to follow using only global positioning systems. The setpoints are changed incrementally with the keyboard.

The system can also activate two autonomous modes: AUTO.Land mode lands the drone in the spot and AUTO.RTL returns the drone to its home position.

There is a simple interface that runs inside the terminal using the python module curses to help the user identify the correct commands as shown in Figure 5.14

5.4.2 Command system

The command system is a modified version of the guidance system (see Algorithm 2). Instead of receiving the mission parameters described in section 5.2.1, it reads a direct list of commands from a text file. These commands are:

- **Move to (X, Y, Z, precision).** Same procedure as the Move method from section 5.2.2
- **Local positioning (id)**. A modified version of the search ID function to only includes its second phase 5.2.3. It searches the ID on the spot and attempts to lower the distance between the marker and the drone.

- **Lifting system (Activate/Deactivate)**. Same procedure as the lifting system from section 5.2.4

Its functionality can be seen in Algorithm 3. Its functionality should be tested previous to execute the guidance system.

Algorithm 3 Command System algorithm

```
1: procedure COMMAND_SYSTEM(mission)
2:     command_list ← START_SYSTEM(mission)
3:     for all command in command_list do
4:         if type(command)=position then
5:             MOVE_TO(command)
6:         if type(command)=id then
7:             SEARCH_ID(id))
8:         if type(command)=activate/deactivate then
9:             SET_LIFTING_SYSTEM(activate/deactivate)
10:    END_SYSTEM( )
```

Chapter 6

Experiments

6.1 Introduction

In this chapter, we will explain the results obtained both from the local positioning system and the guidance system. A summary of the setup is given plus a discussion of the results.

6.2 Initial experiments

The first set of experiments took place in the laboratory. This tests focused on examining both the accuracy of the UWB positioning system and the stability of the small-sized drone.

Initially, for safety reasons, the propellers were removed until verification of the correct functionality of the drone. Then, we prepared the drone, the wireless area and the three anchors system (Appendix A explains the setup procedure).

We placed the UWB antenna on top of the drone vertically and proceeded to test its reliability. Figure 6.1 shows an image of the test. We manually moved the drone around the laboratory and checked that its accuracy was consistent with the results obtained by the previous students (20 cm in x, y to 40 cm in z). However, these results greatly varied in function of the position and orientation of the UWB antenna.

Once the functionality of the global positioning system was confirmed, we started analysing the behaviour of the drone's motors. For this test, we kept the propellers off and used a manual controller. The results showed that the engines spun inconsistently due to a calibration error of the PX4. The re-calibration of the flight controller fixed these issues.

Finally, we reassembled the propellers, and we tested the ability of the hexacopter to keep a steady flight. The analysis was performed inside the drone area at the

Figure 6.1: UWB Test

EPL with the manual controller. Figure 6.2 shows an image of the manual test. In this scenario, the effect of wind should be insignificant and yet, the drone presented an unstable behaviour. It showed difficulty to keep altitude, position and to take off gently.

This test was performed after a PX4 calibration, and thus, the focus shifted to a hardware problem. In particular, it was observed that the motors would spin too abruptly and that they might be responsible for the nervous behaviour of the system. Choosing engines of lower power should fix the problem. Unfortunetly, the COVID-19 restrictions limited further access to the drone, but future projects should take that into account (see Chapter 7: Future Work).

6.3 The local positioning system

To test the local positioning system, we made use of the virtual camera running an identic version of the identification code, just modified to extract the images from Gazebo. This test enabled for more straightforward analysis as the exact position of every item is known in the simulation. Hence, we can compare the estimated distance of the algorithm with the exact real distance.

The setup consisted of the tag fixed in the ground, and the camera movement automated. The test was performed moving the camera to up to five meters, with

Figure 6.2: Manual Test of the small drone

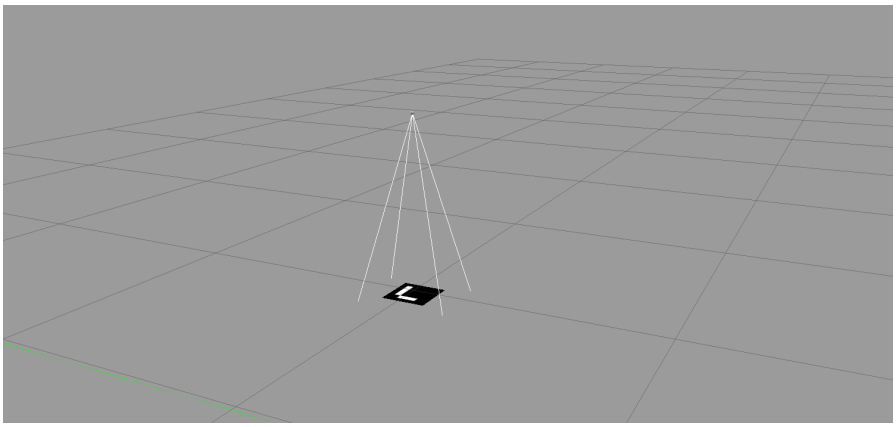
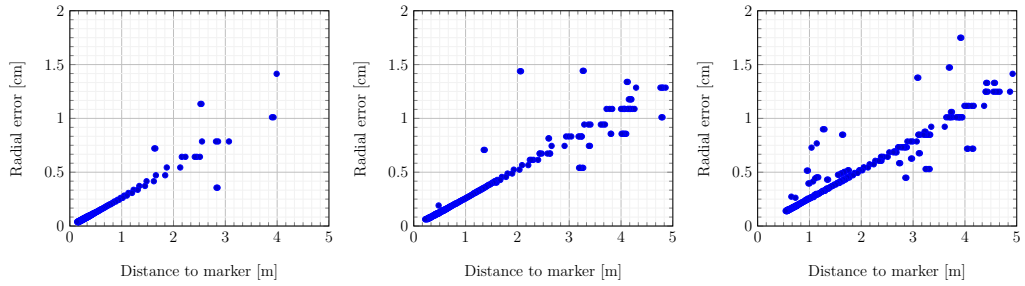


Figure 6.3: Set up of the local positioning test

eight different yaw angles, and two different markers sizes. Figure 6.3 shows the setup.

We measured the radial error as: $e_r = \sqrt{e_x^2 + e_y^2}$. Where e_x and e_y are the X and Y errors of the camera respect to the tag, respectively. The results showed that the radial accuracy decreased consistently with distance but remained under acceptable limits (below 5 cm). See Figure 6.4. The error was invariant to yaw angle and the size of the marker.

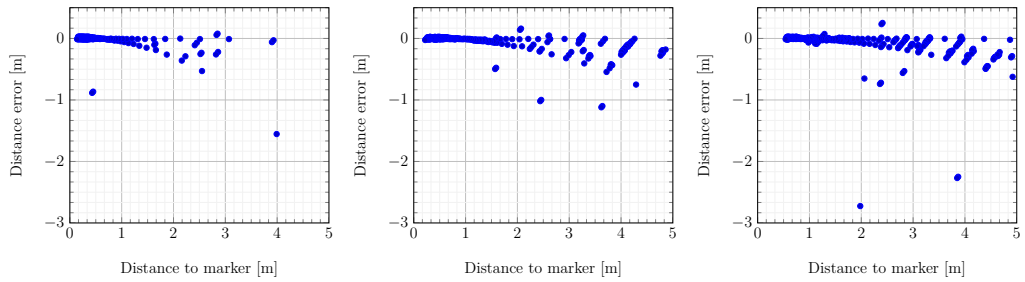


(a) 10cm marker (b) 20cm marker (c) 30cm marker

Figure 6.4: Evolution of the radial error when increasing the distance to the marker with different tag sizes

This test allowed to obtain the operating range of the positioning system. Indeed, the resolution and field of view of the camera constrain the distance at which the markers are identifiable. When the camera is very close to the tag, it is not able to keep it in all in the image. For long distances, the resolution does not allow the pattern to work correctly. For a marker of 10x10cm, the virtual camera was able to recognize the tag from 20 centimetres to up to 3 meters reliably. In comparison, a marker of 30x30cm identified the marker from 50 centimetres to up to more than 5 meters reliably.

We also measured the distance accuracy calculated as the difference between the estimated distance and its real value. The results showed that the distance error increases much faster than the radial error (see Figure 6.5). The error was also invariant to the yaw angle, but marker size play an important role.



(a) 10cm marker (b) 20cm marker (c) 30cm marker

Figure 6.5: Evolution of the distance error when increasing the distance to the marker with different tag sizes.

It is interesting to see that, at closer ranges, from 0 to 1 meter, there is not a substantial difference of the distribution of error. What is more, greater size markers actually are less convenient, as they lose sight of the marker sooner. At

larger ranges, the size of the marker plays a more important role as shown in Figure 6.6. However, the accuracy at these distances is not as important as the limitation of losing sight of the marker during the descending procedure.

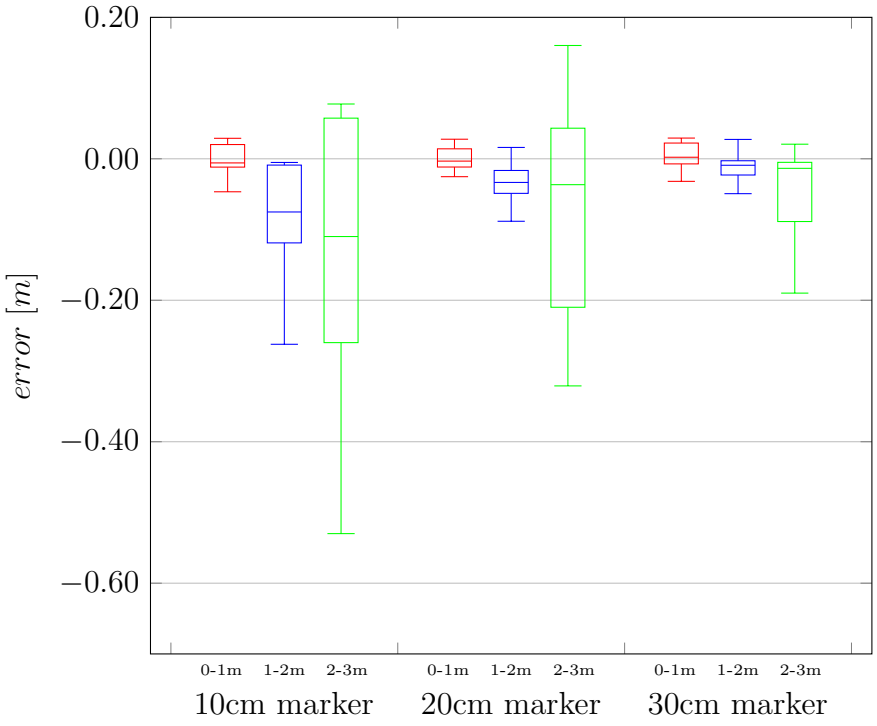


Figure 6.6: Error distribution in function of the size of the marker and distance between the camera an the tag

The accuracy results obtained in the simulation were validated using the real camera. The results were also coherent with the virtual camera and with the works of Jacques Thibault and Frederic Kaczynski[7] .

These results showed that using ArUco markers for a local positioning system is a feasible solution. The error obtained in closer ranges is small enough to accomplish the target precision (under 5 cm). Bigger tags carry better accuracy in the distance estimation at farthest distances. However, it is at the expense of losing sight of the marker at closer ranges.

6.4 The control method

The control method test aimed to examine the ability of multiple control methods to keep the drone centred around one point during 50 seconds. The aim was to see if the drone was able to maintain itself under 5 cm of the target, the maximum

acceptable error for the actual drone-compatible systems already developed [10]. The experiment was enterally performed using the SITL Gazebo simulation using a 20x20cm marker. The setup of the experiment can be seen in Figure 6.7

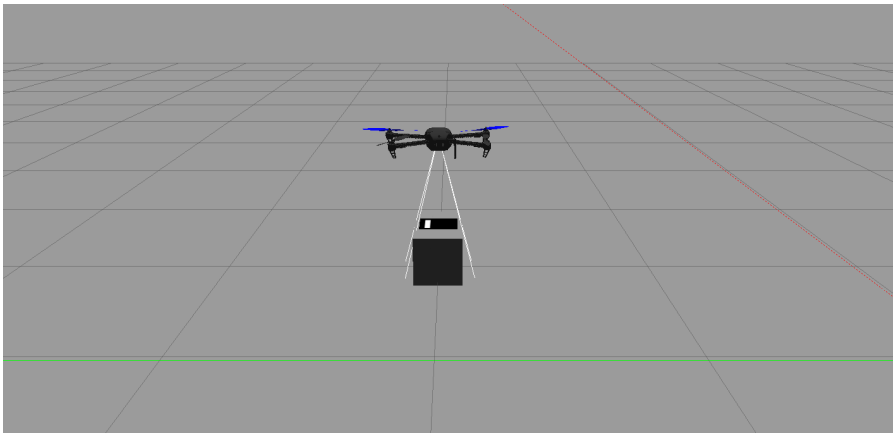


Figure 6.7: Set up of the control method test

Initially, we examined the results of only utilising a global positioning system to see the base performance without a local positioning system. For doing so, we commanded the drone to achieve an invariant global setpoint using the method developed in 5.2.2.

The results showed that, even if the PX4 controller attempts to close the distance between the drone position and the setpoint, the results are not great. Without information to close the control feedback, we are unable to cancel the uncertainty of the global position method. As shown in Figure 6.8, the drone was not able to keep itself inside the 5 cm allowed radio, and the error escalated to up to 10 cm.

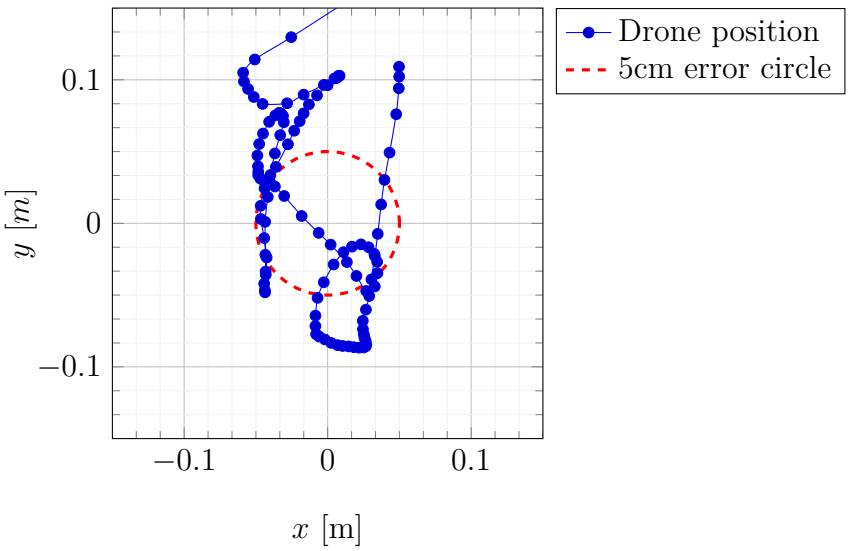


Figure 6.8: Positioning error of the drone using the global position method during 50 s

The sole usage of a global positioning method is not enough for the execution of precision manoeuvres. The implementation of a local positioning method can significantly improve these results, and there are multiple ways to proceed. For this project, we focused on exploring three different control methods: Error Setpoint, Speed Control and PI Setpoint.

The Error Setpoint control method calculates the distance between the drone and the marker, the distance error, using the information of the local positioning method. Every new iteration it will command the drone to move to a new setpoint calculated as the actual estimated global position of the drone plus the distance to the local location.

The results obtained better accuracy than using just the global positioning system. This control method allowed the system to achieve the setpoint fast and stably. However, the response of the system was not consistent under the 5 cm error limit. Figure 6.9 shows the XY position of the drone and the time evolution of error.

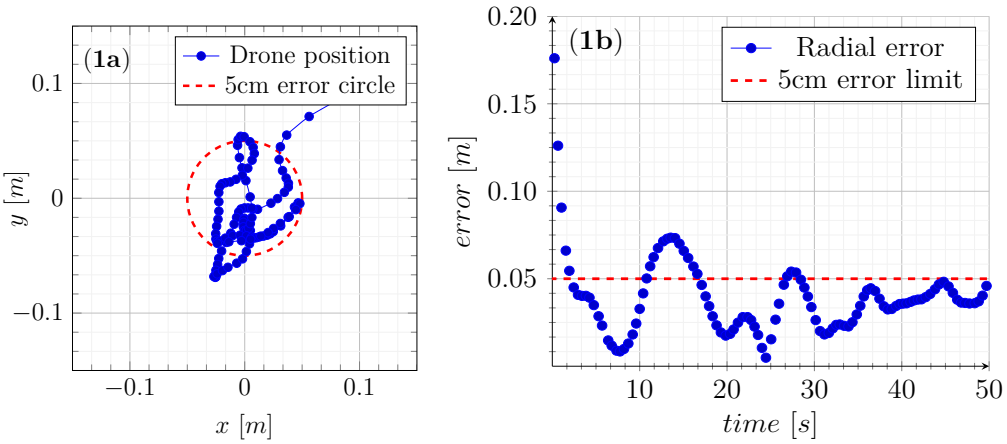


Figure 6.9: Positioning error of the drone for 50 seconds using the error setpoint control method. Image labelled as (1a) represents the XY position of the drone through time. Image labelled as (1b) represents the time evolution of error.

The PI speed control method consisted of commanding the drone with their speed. The system will stop sending global setpoints and instead will momentarily command the drone to move at speeds proportional to the distance error. The results showed that the system was able to keep itself consistently under the 5 cm range. This method has the advantage of not depending on the accuracy of the positioning method. However, the way the PX4 commanded the speed resulted in more sharp manoeuvres witch could not be ideal when carrying weight. Figure 6.10 shows the XY position of the drone and the time evolution of error.

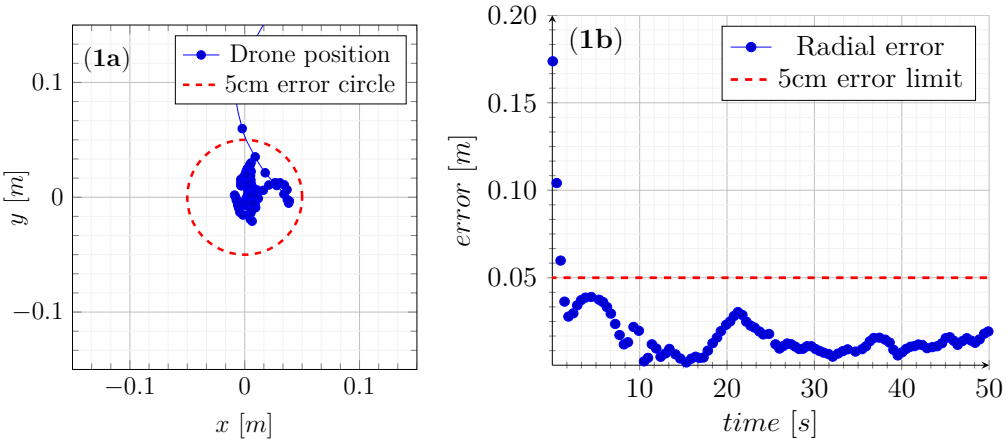


Figure 6.10: Positioning error of the drone for 50 seconds using the PI speed control method. Image labelled as (1a) represents the XY position of the drone through time. Image labelled as (1b) represents the time evolution of error.

Finally, the PI controller functions similarly to Error Setpoint Method. This system computed the distance error and updated the setpoints proportionally. However, thanks to the integral part of the controller, we obtained better results. The error kept consistently under the 5 cm target, with a fast approach, and stable manoeuvres. Figure 6.11 shows the XY position of the drone and the time evolution of error.

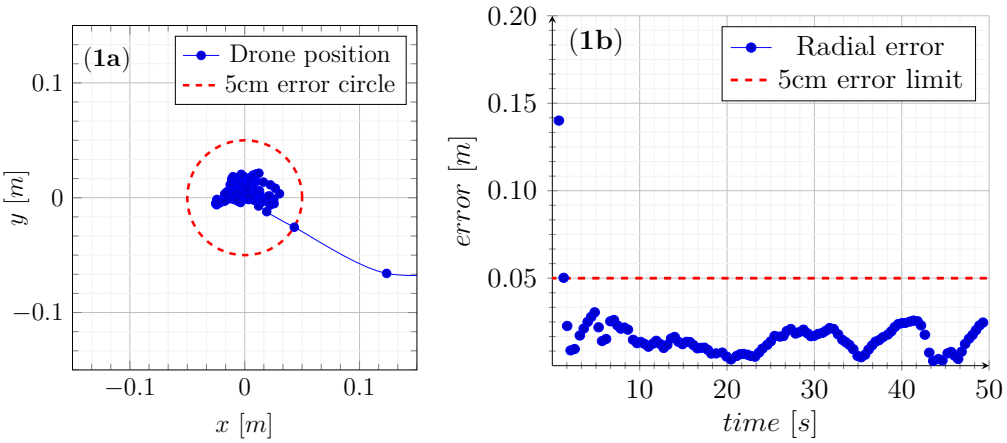


Figure 6.11: Positioning error of the drone for 50 seconds using the PI control method. Image labelled as (1a) represents the XY position of the drone through time. Image labelled as (1b) represents the time evolution of error.

In Appendix B.2, more experiments can be found using these control methods. The summary of these results can be seen in Figure 6.12. We can see that both the PI speed control method and the setpoint PI method present a substantial improvement over the Error Setpoint method. Even if the PI Setpoint method presents better results in this scenario, the speed control method should not be disregarded, and both options should be tested with the real drone to verify their accuracy.

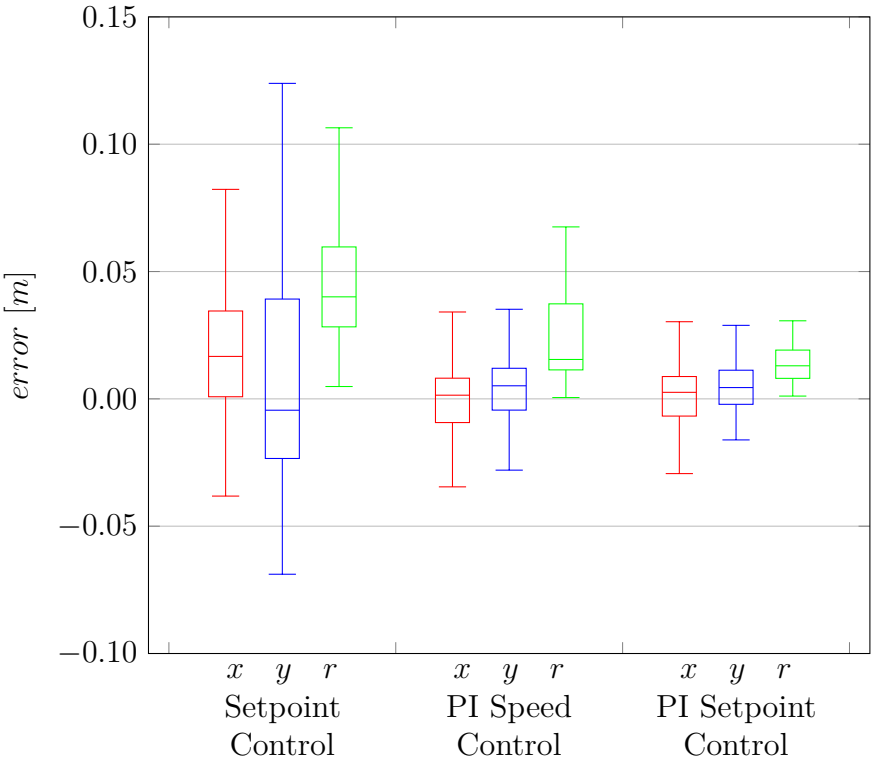


Figure 6.12: Error distribution for different control approaches

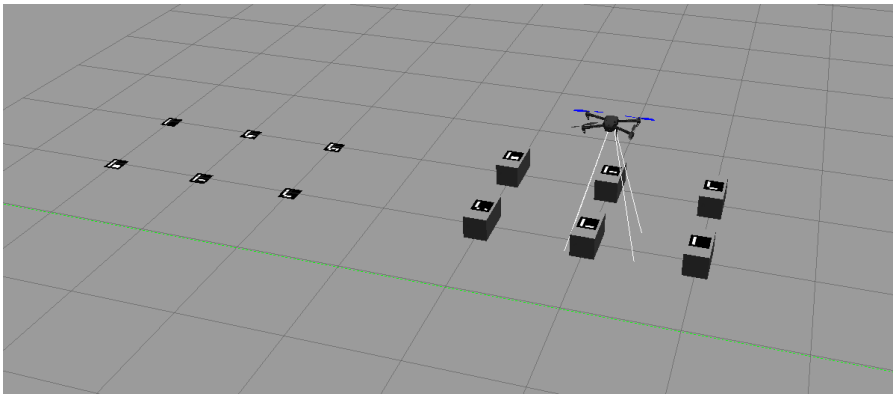


Figure 6.13: Image of the guidance system test

6.5 The guidance system

The last test consisted of checking the functionality of the guidance system. Like the previous test, the system was carried on the SITL simulation. It was tested with a stack of bricks and their deposit positions, both using 20x20cm ArUco markers, as shown in Figure 6.13. The control method selected was the PI Setpoint method, as it presented the best results in previous experiments.

The test showed that the drone was capable of executing the system without problems. We tested the system under full cycles and on all safety measures worked correctly (see section 5.3).

For the sake of clarity, we will only show the trajectory of the drone searching one single brick and its deposit positions. Figure 6.14 shows the 3D representation of the trajectory and Figures 6.15 and 6.16 its 2D projection. As observed, the system works as expected.

Finally, we can also examine the system from a time point of view. The drone was able to execute the task in a bit more than 70s. The descending time was of 10 seconds each time, see Figure 6.17. Future iterations of the project could examine the balance between time and precision.

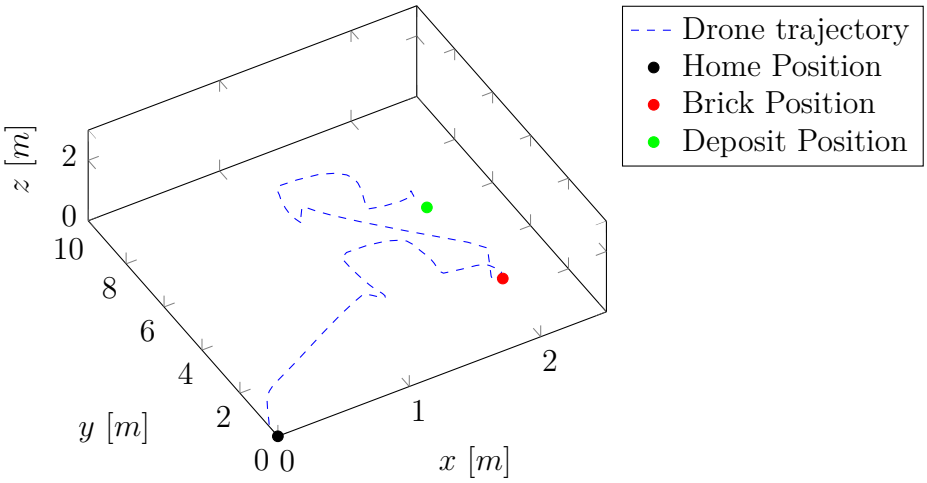


Figure 6.14: 3D trajectory of the drone during the guidance system

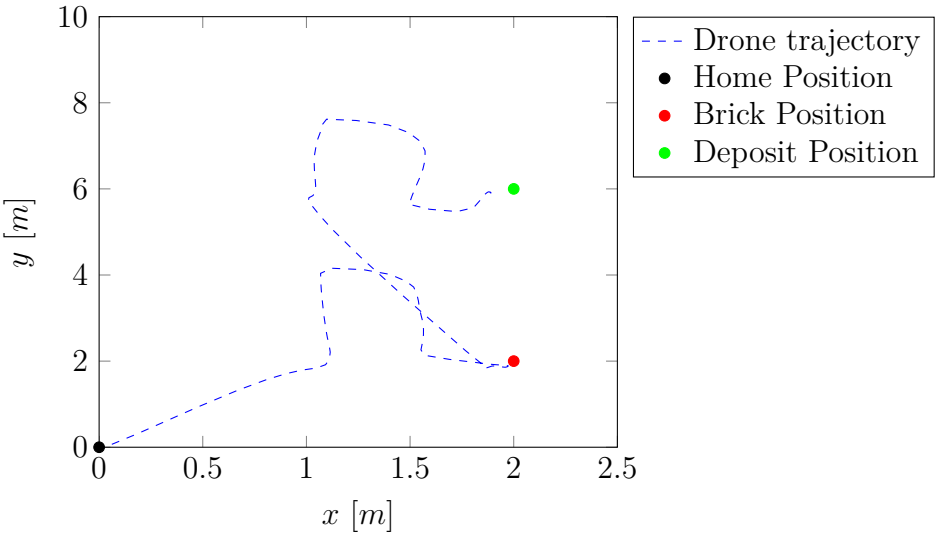


Figure 6.15: 2D trajectory of the drone during the guidance system

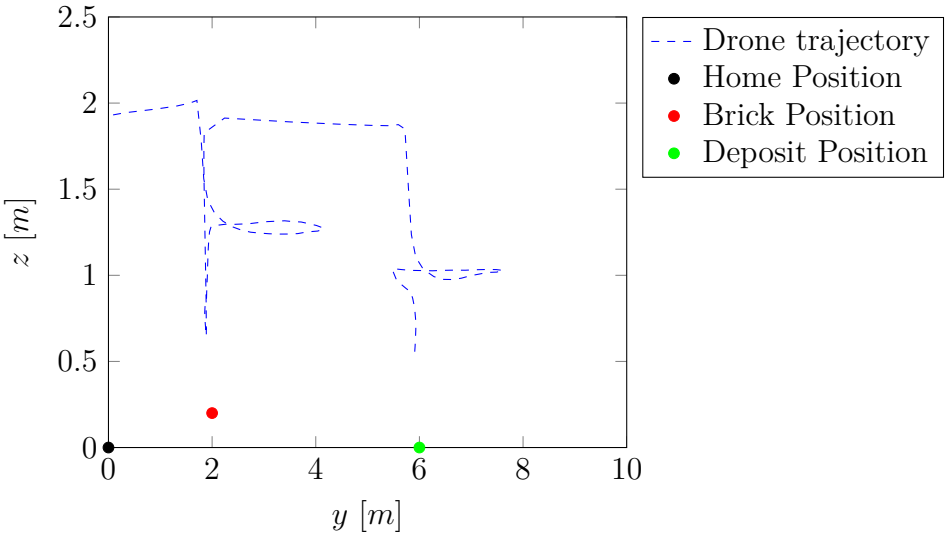


Figure 6.16: 2D trajectory of the drone during the guidance system

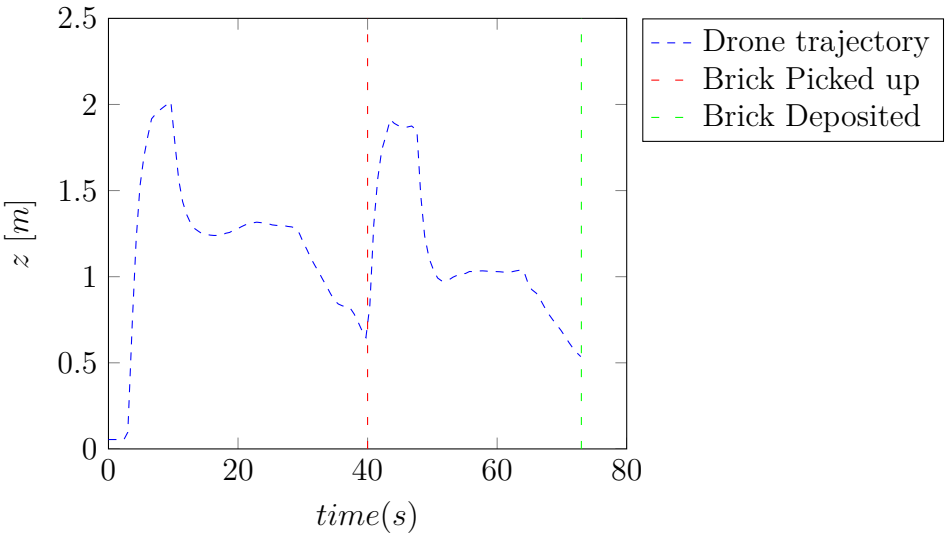


Figure 6.17: Time evolution of the z coordinate during the guidance system

Chapter 7

Future work

Due to the COVID-19 crisis lived in the spring of 2020, much of the planned field experiments had to be cancelled. The results obtained using the simulation shall not be discarded, but it is essential to replicate them to confirm the conclusions of this project.

First, future work should focus on reviewing the equipment of the small-sized drone before attempting to control it autonomously. The initial experiments of the project revealed that the hardware of the small-sized drone was a bit unreliable. There are multiple possible reasons why it could have behaved like this, but the first focus should be the motors, which do not seem fit for the system. Also, the chasis and connections have endured some wear and should be analysed. Another possible approach could be to adjust the internal values of the PX4 PID.

Alternatively, constructing a new drone using renovated parts as well as the new models of the PX4 flight controller should be studied. A user manual can be found at the end of this thesis and at the end of its 2019 predecessor [10]. This tutorial can ease the development of a new system.

When the hardware has been proven reliable in manual flights, iterative tests should be done with the drone. The user should familiarise itself with the code and set the security parameters accordingly (see 5.3).

The first test should be made with the manual mode developed in section 5.4.1. This system will allow testing the drone only under a global positioning system, such as the UWB. If the system is working properly, the local positioning system can be tested with the command mode (see section 5.4.2). The PI control parameters should be recalculated and recheck system performance. Once there is acceptable behaviour, the guidance system could be used as a framework to try to obtain more refined solutions such as the incorporation of a web-based user interface.

Chapter 8

Conclusions

This thesis developed the necessary systems to operate builder drones. The aim was to obtain a method capable of manipulating the vehicle while keeping an accuracy higher than 5 cm.

For doing so, we first developed a local positioning system using ArUco markers. Using a camera attached to the drone, we estimated the position of these ArUco tags and tested their performance with a real and a simulated camera.

The results showed that the radial error of the estimated position was proportional to the distance with the marker, i.e., as the drone moves away, the precision of the system lowers. The radial error remained similar for different markers sizes and showed excellent accuracy in the simulation.

Regarding the estimated distance, its accuracy decreased at a much higher speed compared to the radial error, and markers of bigger size allowed for much better accuracy at more considerable distances. However, larger tags have the disadvantage that, at closer ranges, the camera is not capable of identifying them in the image.

All in all, it was proven that a 20x20 marker is a good position estimator. At distances lower than 2 meters, is has a radial error smaller than 1 cm, which is lower than the 5 cm radial error necessary for this thesis. Regarding the distance error, it is consistently lower than 3 cm at a distance lower than 1 meter.

The next step was to develop a control method capable of utilising the local positioning system. We tested multiple control methods inside the simulation and demonstrated that it was possible to improve the performance of the PX4 basic setpoint command. Using both an altered setpoint control method and a speed control method, we were able to keep the drone under the 5 cm error perimeter consistently.

Finally, we developed the guidance system, a program capable of commanding the drone in building missions. This program is capable of searching for a specific brick in a given area and transporting it to its deposit position. During development,

we took special attention to consider multiple security methods and to design complementary systems to aid future iterations of these projects. All of them were tested under the simulation and returned satisfactory results.

To conclude, the results obtained in this project prove that it is possible to get higher positioning accuracy than the one provided with the PX4 controller. The combined use of the local position method and the guidance system proved successful. The simulated drone executed the construction manoeuvres satisfactory and with the required accuracy.

Certainly, COVID-19 profoundly affected the field results of this project. It was not possible to verify the exact extent of its results, but they should not be disregarded for this reason. The methods developed in this thesis serve as a solid foundation for future iterations of the project, and as a step closer to obtain builder drones.

Bibliography

[1] Artyom Maxim et al. "UAV Guidance with Robotic Total Station for Architectural Fabrication Processes". In: *researchgate.net* 86.October (2017), pp. 145–161. URL: https://www.researchgate.net/publication/320491203.

[2] Frederico Augugliaro et al. "The flight assembled architecture installation: Cooperative contruction with flying machines". In: *IEEE Control Systems* 34.4 (2014), pp. 46–64. ISSN: 1066033X. DOI: 10.1109/MCS.2014.2320359. URL: https://ieeexplore.ieee.org/abstract/document/6853477/.

[3] Justin Leplat and Jean-Sébastien Breton. "Feasibility study for drone - based manufacturing of architectural structures". In: August (2015).

[4] Sébastien GOESSENS. "Drone-compatible additive manufacturing of building-scale structures". In: (2019).

[5] Pierre Latteur, Sébastien Goessens, and Caitlin Mueller. "Masonry construction with drones". In: *Proceedings of IASS Annual Symposia*. Vol. 2016. 7. 2016, pp. 1–10. ISBN: 2518-6582. URL: https://www.researchgate.net/publication/316107815%7B%5C_%7DMasonry%7B%5C_%7Dconstruction%7B%5C_%7Dwith%7B%5C_%7Ddrones.

[6] Adrien Naveau and Amaury Moncourrier. "Development of "drone-compatible" masonry construction system". 2016.

[7] Thibault Jacques et al. *Development of a local positioning system for builder drones with image processing*. 2018.

[8] N Sorensen et al. "Development of a guidance and automatic positioning system for builder drones with a theodolite". In: *dial.uclouvain.be* (). URL: https://dial.uclouvain.be/downloader/downloader.php?pid=thesis%7B%5C%%7D3A14688%7B%5C&%7Ddatastream=PDF%7B%5C_%7D01.

[9] Mathieu MONNART. "Développement et validation d ' un système de guidage autonome pour l ' assemblage d ' éléments". In: (2019).

[10] Nicolas Vanvyve et al. "Design of a positioning and a mission management system for builder drones". In: (2019).

[11] Decawave. *Technology - Decawave*. 2019. URL: https://www.decawave.com/technology1/ (visited on 05/02/2020).

[12] *Simulation · PX4 v1.9.0 Developer Guide*. URL: https://dev.px4.io/v1.9.0/en/simulation/ (visited on 05/03/2020).

[13] *jMAVSim Simulation · PX4 v1.9.0 Developer Guide*. URL: https://dev.px4.io/v1.9.0/en/simulation/jmavsim.html (visited on 05/03/2020).

[14] *Gazebo*. URL: http://gazebosim.org/ (visited on 05/03/2020).

[15] Duane Brown. "Decentering Distortion of Lenses - The Prism Effect Encountered in Metric Cameras can be Overcome Through Analytic Calibration". In: *Photometric Engineering* (1966).

[16] OpenCV. *OpenCV: Camera Calibration and 3D Reconstruction*. 2016. URL: https://docs.opencv.org/3.1.0/d9/d0c/group%7B%5C_%7D%7B%5C_%7Dcalib3d.html%7B%5C#%7Dga61585db663d9da06b68e70cfbf6a1eac%20http://docs.opencv.org/master/d9/d0c/group%7B%5C_%7D%7B%5C_%7Dcalib3d.html%7B%5C#%7Dga549c2075fac14829ff4a58bc931c033d.

[17] S. Garrido-Jurado et al. "Generation of fiducial marker dictionaries using Mixed Integer Linear Programming". In: *Pattern Recognition* 51 (Mar. 2016), pp. 481–491. ISSN: 00313203. DOI: 10.1016/j.patcog.2015.09.023.

[18] Francisco J. Romero-Ramirez, Rafael Muñoz-Salinas, and Rafael Medina-Carnicer. "Speeded up detection of squared fiducial markers". In: *Image and Vision Computing* 76 (Aug. 2018), pp. 38–47. ISSN: 02628856. DOI: 10.1016/j.imavis.2018.05.004.

[19] *OpenCV: ArUco Marker Detection*. URL: https://docs.opencv.org/master/d9/d6a/group%7B%5C_%7D%7B%5C_%7Daruco.html%7B%5C#%7Dgab9159aa69250d8d3642593e508cb6baa (visited on 05/03/2020).

Appendix A

User Manual

This tutorial supposes that you are using the small-sized drone seen in Chapter 2. The small drone has a PX4 cube controller and an Odroid running Ubuntu Mate 18.04 already connected to a local network.

If needed, it is possible to set up the system working on other configurations. In that case, the first place to start is in the PX4 Development guide [12]. You will need to set up both the controller and the onboard computer running ROS on top of Ubuntu.

A.1 Drone setup

In order to execute the autonomous systems of the drone, you need to verify its connections.

First, make sure that odroid has the wifi dongle connected to its port near the power supply. This will enable wireless communication between the Odroid and a computer, and it is necessary to launch the systems remotely.

Then check that the lower port of the Odroid is connected to the controller and upper port is connected to the USB Hub. In the USB Hub connect the Logitech camera and the UWB antenna.

Make sure that the UWB antenna is correctly fixed perpendicular to the drone and the three UWB anchors are activated. Likewise, make sure that the camera is fixed, parallel to the drone plane, and that is decalage is consistent with the one of the program (see section 6.3).

Finally, you will need to connect the two cables of the electronic board to the Shifter Shield. The ground on pin 9 and the second on pin 11. This will enable the Lifting system.

Finally, verify that the batteries are charged and connect them to the drone.

A.2 Connection to the Odroid.

To command the Odroid, it is necessary to establish a connection between the unit and a computer. One way of proceeding is by creating an SSH wireless connection between them.

For doing so, you must connect a wifi modem to the power supply and connect both the Odroid and a computer. If you use the TP-Wifi modem of the laboratory, the odroid will connect automatically supposing that it has the wifi dongle connected. If you use another Wifi modem, you will first need to establish a connection between both. If you need help with this procedure, you can consult the Appendix of [10] .

Once the Odroid is connected, connect to the network with the computer. The credentials of the network are:

- **Name**: IMU
- **Password**: odroidadmin

Once both systems are connected to the same network, execute the ssh command to connect to the Odroid:

```
$ ssh odroid@ip-wifi
```

Where the IP-wifi is the fixed IP address of the wifi key (In the TP modem of the laboratory is 192.168.0.150). The password of the Odroid is "odroid".

Once the connection has been established, you will be able to access the Odroid folder and launch its systems.

A.3 Launch the systems

To launch the systems, you will first need to place the updated systems in the Odroid. They should be placed in the following folder of the Odroid:

```
catkin_ws/src/SmartDrone/launch
```

Then you will need to open three different terminals. One for each node:

1. **Launch Positioning System:** Execute the following command to launch the positioning system:

   ```
   roslaunch smart_drone system.launch --screen
   ```

   Then, navigate to Widgets in the Mavlink Inspector of Qground-Control and verify that the following two messages have similar values:

- LOCAL_POSITION_NED
- VISION_POSITION_ESTIMATE

If they do not have similar values, go to parameters in QgroundControl and change LPE_FUSION to include the barometer, wait until values converge, and deactivate it again.

2. **Launch the local positioning system:**

   Navigate to the folder where the file is and execute it:

   ```
   $ cd catkin_ws/src/SmartDrone/src
   $ local_positioning.py
   ```

3. **Launch the guidance system:**

   Navigate to the folder, modify the mission file and execute it:

   ```
   $ cd catkin_ws/src/SmartDrone/src
   $ atom guidance_mission.txt
   $ ./init.sh
   $ guidance_system.py
   ```

Appendix B

Complementary Experiments Data

B.1 Vision results

The following pages present a closer look of the results of the distance error of the local positioniong system:

1. Error at closer distances (<1 m) B.1
2. Error at medium distances (1-2 m) B.2
3. Error at larges distances (2-3 m) B.3

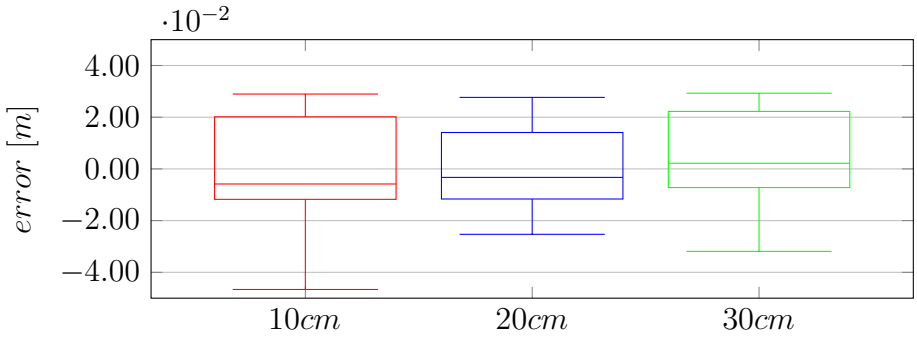


Figure B.1: Error distribution for a distance smaller than 1 meter

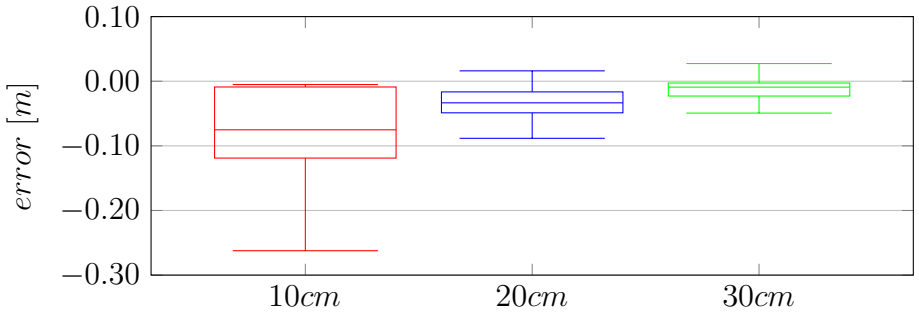


Figure B.2: Error distribution for a distance between 1 and 2 meters

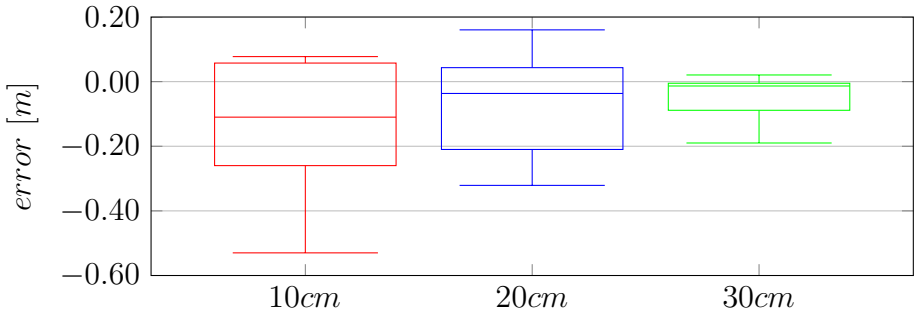


Figure B.3: Error distribution for a distance between 2 and 3 meters

B.2 Control results

The following pages represent the results of the three types of control studied in this project:

1. Setpoint Control B.4
2. PI Speed Control B.5
3. PI Setpoint Control B.6

The conditions of the test are explained in Chapter 6.

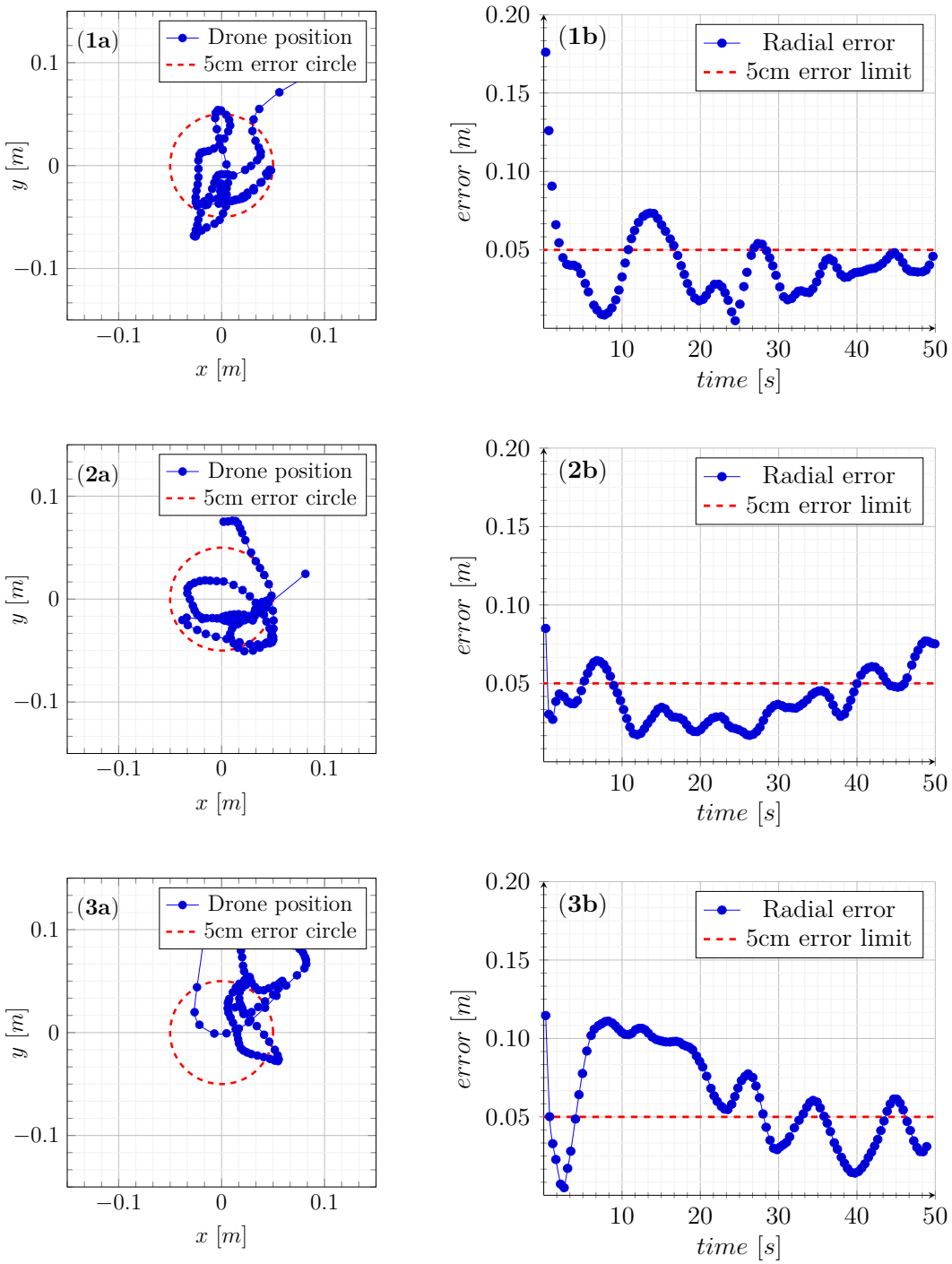


Figure B.4: Positioning error of the drone for 50 seconds using the simple setpoint method. Images labelled as (a) represent the XY position of the drone. Images labelled as b represent the error to the marker respect to the time

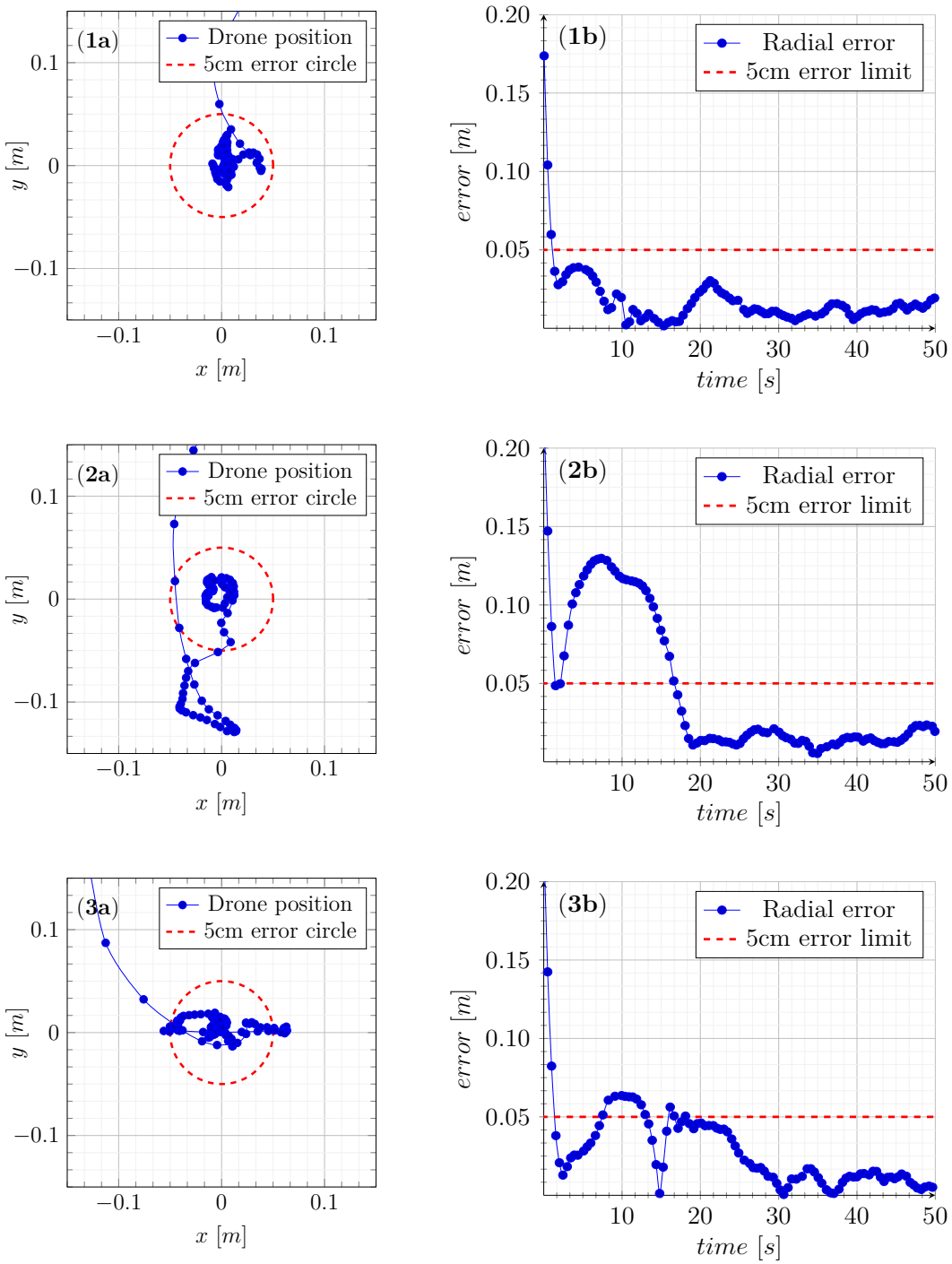


Figure B.5: Positioning error of the drone for 50 seconds using the PI speed controlling method. Images labelled as (a) represent the XY position of the drone. Images labelled as b represent the error to the marker respect to the time

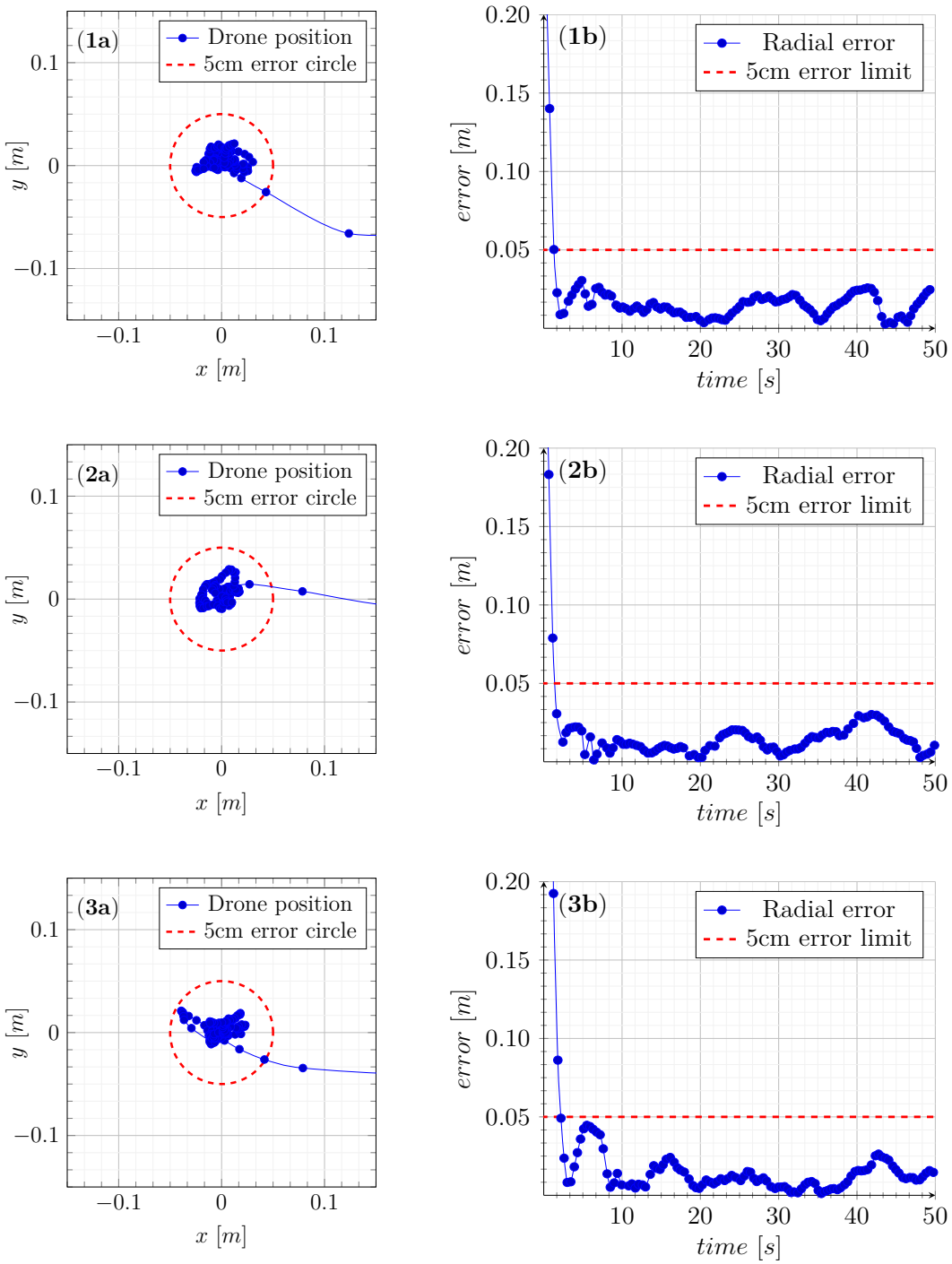


Figure B.6: Positioning error of the drone for 50 seconds using the PI setpoint control. Images labelled as (a) represent the XY position of the drone. Images labelled as b represent the error to the marker respect to the time

UNIVERSITÉ CATHOLIQUE DE LOUVAIN
École polytechnique de Louvain
Rue Archimède, 1 bte L6.11.01, 1348 Louvain-la-Neuve, Belgique | **www.uclouvain.be/epl**