FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

# Indoor Tracking for Drone

**Sérgio Emanuel Costa Vinha**

MSC DISSERTATION


Mestrado Integrado em Engenharia Eletrotécnica e de Computadores

Supervisor: Fernando Fontes

Co-Supervisor: Luís Paiva

July 23, 2019

# Abstract

Currently, the drone field is receiving a special attention because of its potential of application. We can use drones in photography, film making, agriculture, as a hobby, racing, precision industry, etc. These applications keep growing every day. However, existing commercial drone technology is not yet mature to autonomously fly in indoor environments.

This work addresses the tracking of autonomous drones in complex environments. The indoor tracking solution to enable a simple cargo transport within warehouses is a motivation that contributes to development new methodologies and logistics management. The present work suggests some improvements to this technology.

Since autonomous drones are recent in robotics, there are still work to progress. To adapt these type of flying robots to automation is a difficult task, since indoor tracking is still in big development and high accuracy is an essential requirement for the application.

The problem consists in providing indoor tracking for drone in warehouse environment. For this a solution based in a drone construction combined with ultra wide-band beacons is implemented. The beacons technology is provided by *Marvelmind* that is fused with IMU measurements provided by the Pixhawk flight controller. The flight stack is PX4, and its integration with Marvelmind, as proposed here, is innovative.

# Resumo

Atualmente, a área dos drones está a receber especial atenção devido ao seu potencial de aplicação. Nós podemos usar drones em fotografia, gravações de cinema, agricultura, como *hobby*, em corridas, industria de precisão, etc. Estas aplicações continuam a crescer todos os dias. No entanto, as soluções de drones comerciais não estão preparadas para voar em ambiente *indoor*.

Este trabalho endereça a localização de drones autónomos dentro de ambientes complexos. A solução de "indoor tracking" para a localização de carregamentos dentro dos armazéns constitui uma motivação que contribui para o desenvolvimento de novas metodologias e gestão de logisticas. O presente trabalho sugere algumas melhorias para esta tecnologia.

Como os drones autónomos são recentes em robótica, há ainda algum trabalho a ser realizado com este propósito. Para adaptar este tipo de robôs voadores a processos autónomos é uma tarefa difícil, pois este tipo de localização ainda está em desenvolvimento e alta precisão é necessária.

O problema consiste em fornecer localização a um drone dentro de um armazém. Para isso, a solução implementada é feita através da construção de um drone, combinada com o uso de *beacons* baseados em comunicações rádio de banda ultra larga. A tecnologia dos beacons é fornecida pela *Marvelmind*, que é fundida com medições do sensor IMU, fornecidas pelo controlador de voo Pixhawk. A biblioteca de voo é o PX4, e a sua integração com o *Marvelmind*, como proposto aqui, é inovadora.

# Acknowledgments

I would like to thank my supervisors Fernando Fontes and Luís Tiago Paiva for the astounding help they gave me during this period of work. Besides the help with the construction of the document and the opinions, they also gave me tips to write that I consider to be decisive to a well organized thesis. They were always available to everything I needed in the course of this year and gave me the motivation to complete this work.

I must as well thank the company where I've done my dissertation. They gave me the opportunity to develop this project in their company and also financed the components used in this project. There were no problem to have two or three coffees per day in the company, constituted the main reason to finish the thesis and never give up. The time I spend there was important to develop communication and organization skills, among other things, essential for my future.

It is fundamental to thank my family as well. There were difficult days in the last months, so I must thank them for all the tremendous support and motivation they gave me. In special to my father, to my mother and to my brother, not only during the thesis, but for all these five years too. I must thank my girlfriend for all the unconditional help, care and affection she gave me along this time. I would like to thank my parents for give me the opportunity to accomplish this important ambition in my life that would be impossible without them.

Finally, I want to thank to all my friends inside the university that I shared good moments with. Specially to my work colleague with who I shared emotions and opinions while working in the company. Lastly, I want to thank my friends outside the university that constituted an important pillar in my life. All the great moments together, all the craziness and all the work made the person I am today, a person that I am pride of.

The support of FCT through FEDER/Compete2020 grant PTDC-EEI-AUT/31447/2017 - UP-WIND in research associated to this project is also acknowledged.

Thank You!

Sérgio Vinha

*"I have not failed.*
*I've just found 10,000 ways that won't work."*

Thomas A. Edison

# Contents

# List of Figures

# List of Tables

# List of Code

# Acronyms

| | |
|---|---|
| BSD | Berkeley Software Distribution |
| ESC | Electronic Speed Controller |
| GPL | General Public License |
| GPS | Global Positioning System |
| IMU | Inertial Measurement Unit |
| IPS | Indoor Positioning System |
| NFC | Near Field Communication |
| NMEA | National Marine Electronics Association |
| RTLS | Real Time Locating System |
| RFID | Radio-Frequency Identification |
| SLAM | Simultaneous Localization and Mapping |
| TDOA | Time Difference Of Arrival |
| TOA | Time Of Arrival |
| UAV | Unmanned Aerial Vehicle |
| UWB | Ultra Wide Band |
| VTOL | Vertical Take-off and Landing |

# Chapter 1

# Introduction

## 1.1 Context and Motivation

Industry is an important field in automation processes. It includes not only the fabrication methods, but also the way goods move between factories, cities and even continents. With this, warehouse management and logistics sector is a vital task in order to improve response time shipment of a product. The movement of goods and parts can be done using autonomous robots. This robots are present in autonomous processes since the beginning of a product construction to its expedition. With the 4.0 industry, flexible and smart manufacturing is required. The research to find and to create solutions has grown, because of complexity expansion of processes in industrial environment.

In fact, we can already associate autonomous robots to drones. They perform autonomous flights, programmed or in real time. Industry, agriculture, surveillance and film making are only examples of the application fields. Cargo applications and delivery solutions contribute to better and faster responses in the logistics sector, growing the economy. By being autonomous and flexible, they can be faster and more agile within a complex environment in comparison against ground robots.

However, the autonomous robotic system accurate localization is a challenging step in robot navigation field. The mobile device should avoid dangerous situations, such as unsafe conditions and collisions. Localization is a fundamental problem in robotics due to the need of the robot position in practically all performed tasks. Thus, to solve this problem, the robotic system needs to extract environment characteristics. In an outdoor environment, GPS solution have already been implemented, being the basic of localization. But in an indoor surrounding we do not have GPS at our disposal. Another technologies are necessary to improve the localization methods in indoor space. Indoor tracking have algorithms that are capable of doing tracking with high accuracy and with high update rate, meaning better accuracy of the results. However, in 3D location it is harder to get good results. This motivates our work.

Indoor real time locating systems have been gaining relevance due to the widespread advances of devices and technologies and the necessity for seamless solutions in location-based services.

Alongside, the integration of cargo drones is not an innovative plan. But the fact of these cargo being transported in indoor environment raises some problems whose resolution is still under development. There are some solutions, but the more recent ones and with acceptable results are still subject of research without any widely accepted standard framework.

An indoor tracking solution is proposed, as well as a drone assembly and programming. Our main problem consists on integrate an indoor tracking technique to the autonomously driving drone, while having a precise and accurate localization of it. For that we use ultra wide band beacons to establish the indoor localization. The main work innovation consists in the integration of the *Marvelmind* location system and a drone equipped with PX4 flight stack. The solution carried out incorporates the customer requirements and needs to be combined with other company plans.

## 1.2 Goals

This dissertation main goal is to design a system capable of autonomously controlling a drone within a indoor space, while knowing his 3D location. This theme is the subject of recent research, see for example the works in [2] and [3].

To do so, some tasks are considered as part of our work plan:

1. To review technologies and principles of indoor tracking;
2. To review the indoor localization algorithm;
3. To assembly and control a certain drone;
4. To implement the integration of the localization device with the drone;
5. To analyze and to test the proposed solution.

## 1.3 Document Structure

Besides this introduction chapter, there are seven more chapters. In Chapter 2, we start with a literature review. First by doing an introduction to the technologies and methodologies used in indoor tracking. Still, inside this chapter, we also include a revision on drones components and its important characteristics.

Then, in Chapter 3, we define the problem and propose a preliminary solution, imposed by the company.

In Chapter 4, we propose a solution for the location device. Later, in Chapters 5 and 6 we introduce the implementation aspects and control architecture of the drone, respectively.

The system integration between *Marvelmind* and PX4 is done in Chapter 7. Some discussion about the implementation is also carried out.

Lastly, in Chapter 8 we summarize the conclusions and suggest some future work.

## 1.4 Contributions

Several contributions were made inside this thesis work. It starts by a review of existing technologies related to indoor tracking and also drones. This thesis is related to a company specialization. We had to study the market solutions available for this project. It was also import to identify relevant features of the final solution when developing this thesis, in order to have a product with commercial value. So, it was necessary to select and adapt existing and available technologies into a solution that satisfies the customer requirements.

The result demanded hardware selection and also assembly. We started by identifying a drone and how it could be integrated with an indoor tracking device. The software field also required some investigation in order to determine whether the hardware available could meet the customer requirements.

The final result consisted in the integration of *Marvelmind* beacons with the PX4 flight stack. Such implementation, to the best of our knowledge, has never been done before. Therefore, the solution proposed and developed in this dissertation is innovative.

# Chapter 2

# Literature Review

In this chapter we introduce the indoor tracking solutions for drones available at present time. The main concepts of indoor tracking are described in section 2.1. The technologies and types of drones used in all type of applications are characterized in section 2.2.

## 2.1 Indoor Tracking

Indoor real time location systems (RTLS) [4] have been gaining relevance due to the advances in robotics, partially in drones. The navigation is a crucial and an inherent task to robots, so they can perform missions. For that, there are some solutions used in 2D and 3D environments.

### 2.1.1 Type of Measurements

There are three types of measurements, that are distinguished by their way of measure:

#### 2.1.1.1 Geometric related measurements

These come from measurements directly associated to geometric constraints between nodes.

- **Received Signal Strength (RSS):** it estimates the distance by measuring the signal strength between two nodes. In other words, it is possible to acquire the distance based on signal strength transmitted by an emitter. The signal has a propagation loss correspondingly to the distance travelled. This is commonly used in low-power systems.

- **Time of Arrival (TOA):** this measures the time a signal takes to travel to a wall and return to the sensor. This sensor can be a laser, for example. Another type of technique based in this measure is the use of a transmitter and a receiver. A time synchronization is strongly required by both ends. Another equivalent methods are usually accomplished using a two-way time-of-arrival ranging protocol or time difference-of-arrival. These yield the most accurate results [5].

- **Angle of Arrival (AOA):** consists in estimations of positions with the angles of arrival to the measured node, sent by a transmitter. It has an advantage in 2D environments that relies on a simple use of only two nodes to get a measure. The accuracy is attenuated when the target is too far from the measuring node. However, system complexity and accuracy can be intensified with time difference of arrival. This uses more than one time to get the result that is based on the difference arrival of waves with known propagation velocity.

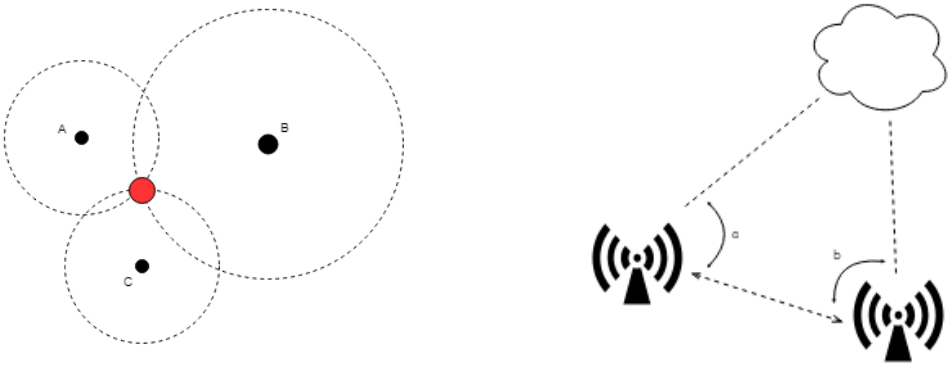
- **Phase Difference of Arrival:** it relies on measuring phase difference of the arrival signals, transmitted with different frequencies. These phases are proportional to the distance traveled, and so better estimations of the measurements are obtained.

#### 2.1.1.2 Position related measurements

Position related measurements can be divided in different types of categories showed below:

- **Trilateration:** it calculates the distance to the nodes (Fig. 2.1a), based in TOA or RSS. Having three nodes, translates in an absolute value, meaning that can have more accurate results.

- **Triangulation:** it makes use of AOA to estimate the position of the target. As stated below (Fig. 2.1b), two stations (A and B) measure the angle incident to the target, knowing the distance of A to B. This is translated in high accuracy but has the throwback of being expensive.




(a) Trilateration

(b) Triangulation

Figure 2.1: Different measurements based in trilateration and triangulation

- **Proximity:** another tracking can be achieved with this technique. This is a simple way of detecting objects. For example, when a tag crosses near a magnetic field, the system throws a binary detection. With this, the sensors know that an object is near their range, so it is inside the room. This particular measure is useful to detect presence in room. This feature

exploits radio-frequency identification (RFID) tracking [6], and allows the use of multiple tags [7] without compromising the effectiveness of the system.

- **Vision:** this method uses video cameras to capture images from the surrounding environment. With this, vision algorithms can estimate position by recognizing objects, like the work done in [2] and [8]. The images obtained can be compared against other images in databases, stored with information of position of each pixel. This technique is used by surveillance cameras to detect tags in certain objects, allowing tracking. The big disadvantage of these systems is the high computer resources required.

- **Scene Analysis:** like vision algorithms, scene analysis also compares information with data previously recorded and stored, like magnetic fields from the room, for example. One technique based on scene analysis is the Fingerprinting method, that requires low hardware requirements and gives high accuracy. The disadvantage of this method is the time consumed in data collection. This is the main reason to hindering the use of this technique [9].

Signals produced by Wi-Fi, FM or AM radio, cellular networks and lights present in all buildings can be utilized for tracking. Ambient light [10], sound [11], [12] and ultrasounds are only examples of what can be used to track objects. These appear as signals of opportunity.

#### 2.1.1.3 Self measurements

**Measurements with inertial devices** have a crucial appliance in robotics. Nowadays, most of the devices have a small inertial measurement unit (IMU) capable of measuring angular velocities and forces. They can work as three-dimensional magnetometers, altimeters and barometer. With this, inertial navigation [13] can be performed, only by knowing the initial position, velocity and orientation. It is difficult to attenuate the error over the time, leading to imprecise measurements. Moreover, this measurements can be fused with other measures by adding filters [8], achieving higher accuracy. Bayesian tracking consists in a Bayesian filter, a state estimation and for last, some filtering algorithms like extended Kalman filter or particle filtering [8] to get good performances.

There are also methods for distributed and cooperative tracking, adding fingerprinting and methods for simultaneous localization and mapping [2].

### 2.1.2 Technologies build on measurements

There are some advanced technologies present in our lives that we can not live without and they are capable of being used to track all kind of objects. They are build on the types of measurements refered to before.

They are growing from the old and exploited **IR** and **Ultrasounds**, to the more recent impulse radio **Ultra Wide-Band** (UWB) techniques [7], [14]. The last one consists of narrow time-domain pulses of very short duration, typically on the order of nanosecond, transmitting signal over a wide frequency band larger than 500 MHz.

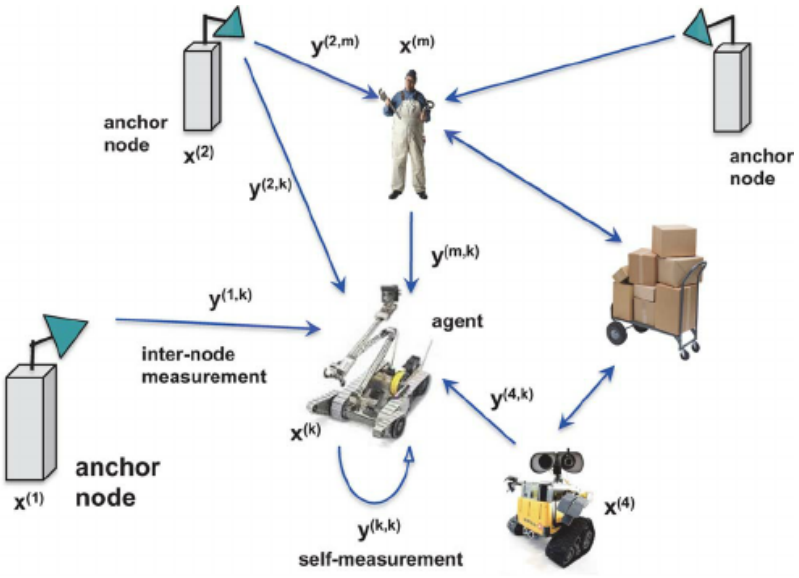



Figure 2.2: A system with anchors and mobile robots simplified. It is also represented a robot with self measurement based in IMU [1].

Another technology developed is **Near-Field Electromagnetic Ranging** that allows waves with low frequencies and long wavelengths exploit the deterministic relationship of the free space within a close range to the transmitter and receiver. In addition, short-range **Wireless** technologies are the base of many standards that offer wireless connectivity. The most used and advanced one is Wi-Fi for wireless networks applications [15]. **RFID** [6] and **Near Field Communication** (NFC) are used for object and items identification, since they are low cost and **Bluetooth** is used for close range and fast connectivity.

The importance in robotics of the tracking took us to systems capable of doing simultaneous localization and mapping - **SLAM** with various technologies like laser and vision [2]. Furthermore, the use of deep neural networks to navigation [16] can also be a solution to some problems that indoor tracking exposes.

### 2.1.3 Technologies Assessement

There are several ways to measure the quality and effectiveness of the techniques presented before.

- **Accuracy and Precision:**

Accuracy is based on how far a value is from its reference target (Fig. 2.3). It is one of the most important factors while considering evaluation for a indoor tracking system.

From another point of view, precision evaluates how the values are distributed around a certain performance.

- **Complexity and Robustness:**

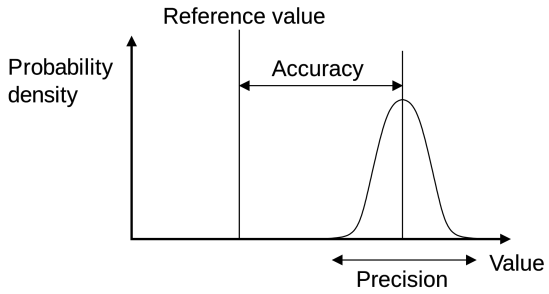



Figure 2.3: Accuracy vs Precision [Source]

While evaluating indoor tracking systems, we have to consider the resources needed to do it. That being said, complexity determines the hardware requisitions when we are increasing, for example, the number of nodes. This describes how easy is to increase the use more nodes to track. More robustness means more reliable measurements and higher ability for tolerating perturbations that might affect the normal conditions of the system. It appears as an important factor in extreme conditions of some environments.

- **Range and Refresh Rate:**

Range is how far an object can be located, while refresh rate is the time needed to update the measurement. These features are normally related to each other. While adding range, the refresh rate of the location decreases. It is also an important factor to increase accuracy.

- **Cost:**

Typically, while adding more nodes or more accuracy, it becomes more expensive to scale the system. However, that is not always the case. The costs are associated not only with hardware, but also with software.

### 2.1.4 Discussion

Considering industrial applications, the outstanding solution includes robustness and really low maintenance. We need to care about this topic.

SLAM is a preeminent solution nowadays, but the payload needed to transport a camera and other hardware make this technology a bad choice for a small drone application. Also, the hardware recommended to process the images is usually expensive.

RFID, NFC and other tags based can be used to track objects, since they provide a high scalability to complex systems. They can be very low powered, standing for almost 3 years in some cases, but are not as precise and accurate as the customer requirements.

On the other hand, technologies using beacons to estimate positions by UWB can be very accurate, since the main calculations are done in a low resourced computer. Like IR and Ultrasounds, by using UWB, we provide a higher range using high frequencies capable of trespassing objects.

These frequencies are not interfered by other networks like warehouse Wi-Fi and cellular communications. They are really fast when comparing against alternative technologies, accomplishing high refresh rates, required by the nature of this project.

## 2.2 Drones

As we continue this thesis, a topic of discussion is Drones, also known as Unnamed Aerial Vehicles (UAV). We are going to introduce to some types and a few crucial characteristics [17].

### 2.2.1 Body

The body comes in first as the most important characterizing feature of a drone. It can adopt several configurations in order to fly. The most recognized ones are helicopters, multicopter and planes.

- **Helicopter:** it typically has only a single main rotor. Even with only one rotor, it is capable of transporting high payloads. The helicopter is not a good option when stability is needed. They are known for being expensive.

Figure 2.4: Example of an helicopter [Source]

- **Multicopter:** it can adopt different sizes, depending on how many rotors it has (Fig. 2.5), namely tricopter, quadcopter, hexacopter and so on.

  The most used one is the quadcopter, for several reasons. It is not complex to control, due to its mechanical simplicity. Having a compact size, makes it easy to control. Due to its four rotors, it is very stable. Inside the quadcopter, there are some different frames due to the position of the rotors in the frame. When we are talking about autonomous flight and higher velocities, the most commonly used frames inside quadcopters are the "plus" and "X" configurations,

- **Fixed wing:** frequently recognized as planes, these have big size and fast velocities. They are not a proper option when the main application is indoor space since it needs a big space to take-off. There are some options with VTOL, that allows vertical take-off and landing like

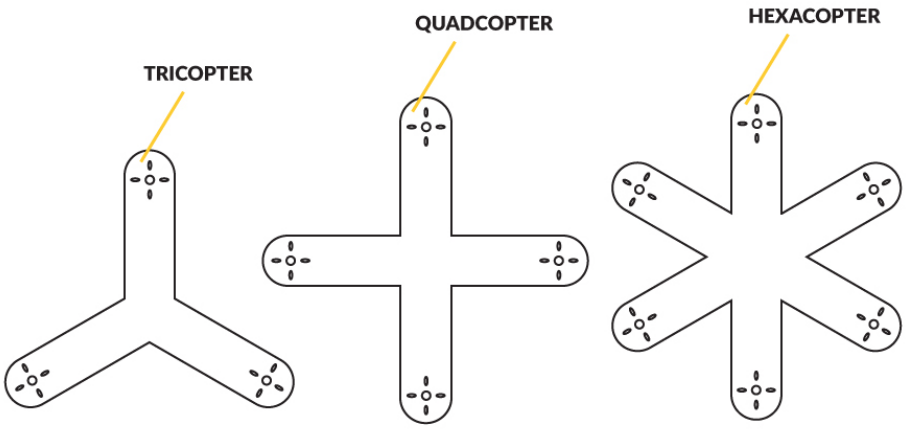



Figure 2.5: Different examples of multicopter frames [Source]

a multicopter and fly like a plane. These are expensive and hard to control for the required application.

(a) Plane [Source]

(b) Plane with VTOL [Source]

Figure 2.6: Different examples of fixed wing UAV

### 2.2.2 Cargo Types

When the main application is photography and film making, we normally select multicopters. And that is not wrong, since they can provide good options for payloads. It can transport a simple camera and its gimbal (Fig. 2.7), sensors and cargo, making them the best option inside drones. Since it has space between rotors, it does not affect the air mass circulation needed for the motors.

On the other hand, planes make it difficult to transport payload, because of their characteristic type of take-off and high control needed.

Figure 2.7: Example of a drone with camera gimbal [Source]

### 2.2.3 Hardware

Firstly, we have the **flight controller**. It is a small circuit board of varying complexity that is a crucial part inside a drone. It is the brain of the UAV. Because of many sensors it has, like gyroscopes, accelerometers and so on, it can estimate how the craft is moving. It has the job to distribute signal to the different rotors, controlling each individually.

There are many brands available, like Pixhawk and APM (Fig. 2.8), each one with their own advantages. Pixhawk is very user friendly, bringing high quality and low cost to their users. It is a reliable open-hardware for autopilot. There are after-market options available too. Some of them are based in microcontrollers like Arduino and Raspberry Pi (Fig. 2.8b and 2.8c).

Another important component is the Electronic Speed Controller, that coverts signals from the flight controller into phased electrical pulses to determine the speed of the brushless motor.

### 2.2.4 Power Supply

Batteries are used to power the UAVs. The most used type is LiPo batteries because of the high energy density and high discharge rate. LiPo batteries are classified with discharging rate (C), nominal voltage, cell count and capacity. The battery is the heaviest component inside a UAV. More time of flight does not mean more capacity, since adding capacity adds weight to the battery.

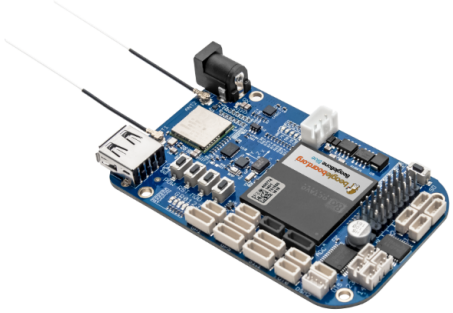
When choosing batteries, we have to keep in mind the maximum current it can provide. It can be calculated by multiplying the battery capacity (Ah) with the discharge rate (C).

### 2.2.5 Software

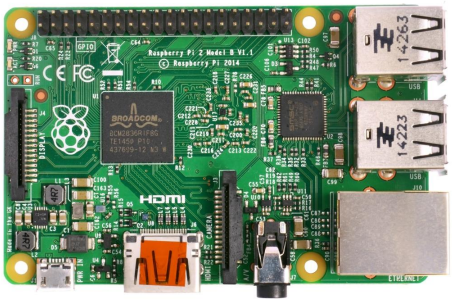
Besides hardware, the flight controller needs a running software to control the UAV. PX4 and Ardupilot are open-software (Fig. 2.9), capable of controlling a large variety of unnamed autonomous vehicles. It offers full-featured and reliable autopilot used by industry.

The main difference between PX4 and Ardupilot are the licences. Since Ardupilot is guarded by GPL licence, it can not be sold without making the modifications open. In the other way, PX4 is covered by BSD licence, which makes a good option for a commercial purpose.

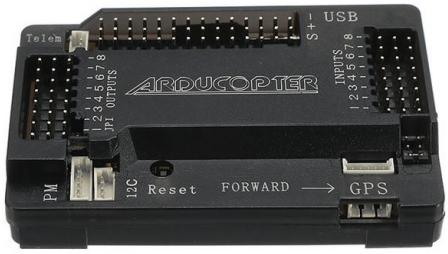

(a) Pixhawk V4 [Source]



(b) Beagle Bone Blue [Source]



(c) Raspberry Pi 2/3 Navio2 Autopilot - [Source]




(d) APM 2.8 - [Source]

Figure 2.8: Example of flight controllers


(a) PX4 [Source]


(b) Ardupilot [Source]

Figure 2.9: Different examples of flight stacks

These autopilots consist of navigation software running in the UAV, along with a ground station controlling software. These include Mission Planner, QGround Control and others. The flight controller shares data with the ground station using communication via Mavlink. This is an appropriate protocol used to communicate with autonomous vehicles, like drones.

### 2.2.6 Discussion

For a complex and relative small environment, like warehouses, planes are not a valuable preference. Even with VTOL, they need a big open space to fly.

Helicopters make it up, when a high payload is needed. Despite having typically only one motor, they are very hard to control. And having a top rotor makes it a disadvantage to transport cargo. Multicopters can achieve more cargo capacity, even knowing that adds more diameter and size to the multicopter. Good and light motors and propellers are necessary for this purpose.

Adding battery capacity insures more flight time. But this is the same of adding payload to the drone because of the battery weight. So it needs to be a balanced drone-battery weight ratio in order to improve flight time.

Bigger drones, like hexacopters and some quadcopters need big space to move because of his propellers. It is also a big concern when choosing a drone for an indoor environment. The clean space inside the drone is also bigger in quadcopters, depending on what type of propellers it has.

Lastly, Pixhawk offers the best option when it comes to user friendly options for UAV flight controller hardware. It has a relative low price comparison with other products in market, while giving a high performance.

# Chapter 3

# Problem Characterization

In this Chapter, we state the problem and list the customer requirements, since this has been a thesis with a company. This company works in the medical field, having specialized robots to transport and organize medical boxes inside pharmacies. The propose of the problem is to replace the traditional ways of transportation of cargo inside warehouses, like treadmill, with cargo drones. In the end, medical boxes can be transported by the help of drones.

## 3.1 Problem Statement

The main objective here is to locate a UAV within an indoor region, that means have a location of the autonomous vehicle, so it knows where to go. It needs to pick up a box, and transport it to a different point of the room. We are looking for high accuracy results in this process.

## 3.2 Customer Requirements

There were some technical specifications that the customer wanted to be implemented in the drone. These include having a **area** of 0.15 x 0.15 m in the middle of the drone, so we can transport small boxes, such as small packs of medicine. Also, an important reminder, is that it needs to pass corridors that have a **dimension** of 0.7 m and have a minimum **velocity** of 1 m/s. One last requirement related to the drone is the payload, of about 1 kg.

Relating to Fig. 3.1, the dimension $a$ is the total space the propellers need to rotate and the drone to move around. The propellers have a diameter of $r$. The total usable space in the top of the drone is given by the measure $d$. These are the main dimensions to take in consideration when choosing the drone.

Having discussed the drone requirements, we proceed to decide the location requirements. The first and essential condition is 0.4 m of **accuracy** (in the best case of 0.1 m). An accurate and robust device is the task to accomplish this work. The drone also needs to be located in a warehouse with **perimeter** of 20 by 20 meters. For this a device with high update rate is also demanded.

htb

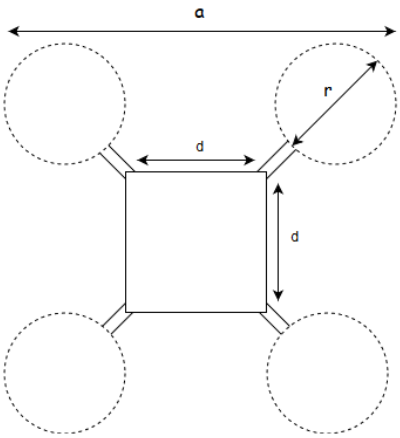



Figure 3.1: Dimensions of the drone

## 3.3 Proposed Solution

This thesis was done in the industry. As so, the company requested a market survey to decide the necessary components and with that a preliminary solution. The drone proposed as the leading solution is a quadcopter. With that, we get the flexibility necessary for the task, while simultaneously having a robust control. The localization device used is based on beacons. So, there are two main systems we have to integrate (Fig. 3.2).

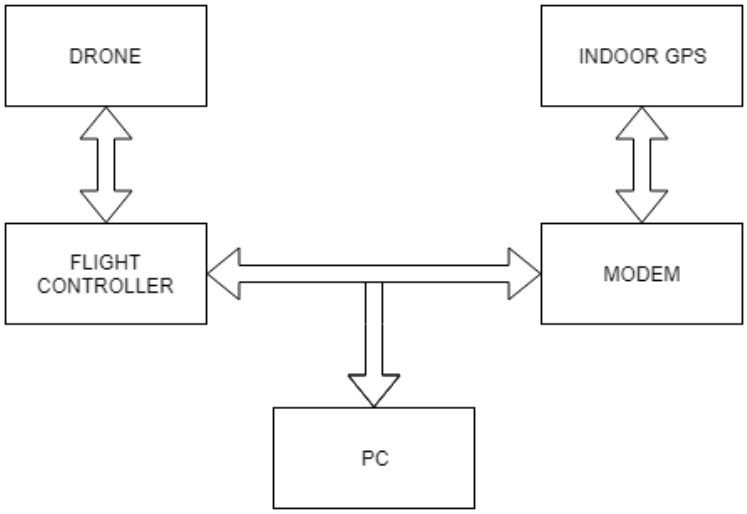



Figure 3.2: System Concept

The central dilemma in this job is to control the drone. For that we are going to use a type of autonomous flight using waypoints. A simple computer is sharing data to control the UAV, that

uses a flight controller. This flight controller controls all the actuators and reacts accordingly to the sensors that a drone owns.

Simultaneously, a device capable of 3D tracking is giving information to a modem that communicates with a computer. The computer does all the necessary computations using the measurements of all the beacons. This device is placed in the top of the drone, usually named as mobile beacon.

The proposed solution consists in convert the data from the modem device to "GPS coordinates", while turning this into indoor GPS. These coordinates, fused with IMU of the drone information and mobile beacon are ranked to give more accurate results [18]. With this, we will be capable of control the drone in an outdoor environment, like normal GPS via Mavlink. These details are going to be explained in detail in the next chapters.

# Chapter 4

# Location Solution

In this chapter, we describe the selected indoor tracking solution to implement in our project. We start by an introduction to the device and establishing the working method. Some tracking tests of the mobile beacon before integration with the drone are also reported.

## 4.1 Marvelmind

To complete the hardware in our project, we need to decide about the indoor tracking technology.

For that we use the ***Marvelmind*** system, that is capable of giving 2cm of accuracy. Also, the range goes up to 30m, accomplishing the customer requirements (section 3.2). The mobile beacon has a battery capacity up to 12 hours of continuous tracking. Note that other technologies were evaluated before choosing *Marvelmind*. A market survey related to indoor tracking solutions can be found in the appendix A, at the end of the document.


Figure 4.1: Marvelmind

The localization system consists of 4 beacons placed on walls and a mobile beacon placed on the drone. The 5 beacons communicate with the modem, that is a device connected to the

computer. The location of the mobile beacon is calculated based on the propagation delay of ultrasonic signal, using trilateration. As reported in Chapter 2, UWB technology is capable of the most accurate results. These measurements can be updated until 63ms, making it a really powerful indoor tracking device. This can be verified with the technology present in this device.

With the *Marvelmind* dashboard we have all the information related with the position of the mobile beacon. It is in this dashboard that we configure all the parameters and the working interface of the system, presented in the next section.

## 4.2 Configuration Parameters

We start by describing the most import parameters used to configure the beacons and the modem. These are configured in the dashboard installed on a *Windows* based computer, as mentioned in the manual [19].

Initializing the modem parameters, we have the update rate of the measurements (going from 1Hz to 16Hz). For this parameter we keep default value of 8Hz, being the most trusted frequency by Marvelmind in their tests. Then we have the window of averaging, averaging between location update measurements. More value means have less location jitter, but higher latency. Also, there are filters of distances. Higher value is equivalent to better filtering, but may be too conservative and "kill" good measurements. For these two parameters, we decided to preserve the default value of 4 and 3 respectively.

As for the beacon parameters, the definitions are the same, but adding the ultrasound parameters. We can change the duty of the signal emitted by the ultrasonic sensor from 1 to 99%. Raising the value, lowers the strength of ultrasonic signal. The value we chose is 50%. Another topic of relevance is the number of ultrasonic pulses the TX beacon emits. More pulses is stronger, but have longer echo. Since we are testing in a small room that does not exceeds 10 meters, also the default value chosen is 5 periods. Finally, the amplification of the arrived signals are set to automatic to be ready for any disruption of the environment of tests and integration.

Additionally, as we are developing the project in Europe, the adopted working frequency band imposed is 433MHz, instead of 915MHz. Inside radio, we can configure the radio profile between 38kbps (for better range and interference immunity, but slower), 153kbps (balanced) and 500kbps that is the fastest, but the lowest radio range and least immune to interference. These parameters were chose accordingly to the copter placement manual of *Marvelmind* [20].

## 4.3 Interfaces

Since the beginning of this chapter, we are discussing all the possibilities and configurations of the *Marvelmind* system. As so, there are different hardware interfaces and protocol of data exchange with mobile beacon - via USB, UART and SPI interfaces [21]. Since we are just transmitting data with the position to the drone, we do not need any feedback. To reduce complexity, we use UART, considering that only 2 cables are needed to transmit data.

The system works in the direct way. In other words, the four stationary beacons are placed in the walls, emit ultrasounds, and a fifth beacon (that is placed in the drone - mobile beacon) records the pulses received. Then, the mobile beacon sends to the modem, via radio, the relative distances to the other beacons. The modem is in charge to do all the mathematics to calculate the position of the mobile beacon. Therefore, this position is carried again via radio to the mobile beacon, that exchanges the position data with the flight controller.

For communication protocol [21] between the beacon and the flight controller, we have NMEA-0183 and streaming packet format.

- **NMEA0183 Message Protocol**

This is a type of GPS message protocol and NMEA is an acronym for the *National Marine Electronics Association*. It is a communication protocol compounded by several NMEA messages that transport different data. It is similar to the second option, being a streaming packet also. Like in streaming packets, there are packets with a code related. In NMEA messages (Fig. 4.2), theses codes are the id of the messages (normally starting by $GPxxx). Each message transfers different combinations of information. The *Marvelmind* is capable of sending the next messages:

- **$GPGGA:** Global Positioning System Fix Data;
- **$GPZDA:** Time, Hours and Data;
- **$GPVTG:** Course Over Ground and Ground Speed;
- **$GPRMC:** Recommended Minimum Specific GNSS Data.

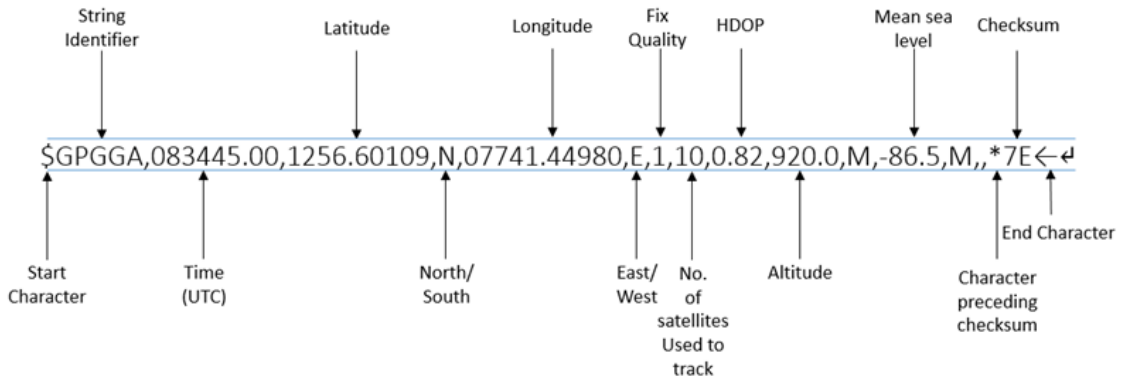



Figure 4.2: Example of $GPGGA message [Source]

Therefore, in the NMEA protocol, we can configure a modem option - *Georeferencing*, to attribute the GPS coordinates we want (latitude and longitude). The *Marvelmind* system measures position in form of rectangular Cartesian system coordinates (X, Y, Z), where Z in most cases is the height. For translation to GPS coordinates, the Z axis is directed up and means altitude above sea level. The Y axis is directed to north, so Y is latitude. Finally, X axis is directed to east, so X is longitude. The point (0, 0, 0) has GPS coordinates according to a georeference point we set.

- **Marvelmind Packets**

Here, the mobile beacon transmits the position of the drone in Cartesian coordinates. Unlike NMEA messages, we can broadcast every type of information of the beacon (position, velocity, rotation quaternion, acceleration, etc) in Cartesian coordinates (Fig. 4.3).

**1.5. Packet of processed IMU data (code of data 0x0005).**

This packet is transmitted when new inertial sensors data available, update rate 100 Hz.

| Offset | Size (bytes) | Type | Description | Value |
|---|---|---|---|---|
| 0 | 1 | uint8_t | Address | 0xff |
| 1 | 1 | uint8_t | Type of packet | 0x47 |
| 2 | 2 | uint16_t | Code of data in packet | 0x0005 |
| 4 | 1 | uint8_t | Number of bytes of data transmitting | 0x2a |
| 5 | 42 | | Data packet (see lower) | |
| 47 | 2 | uint16_t | CRC-16 (see appendix) | |

Format of data packet

| Offset | Size (bytes) | Type | Description | Value |
|---|---|---|---|---|
| 0 | 4 | int32_t | Coordinate X of beacon (fusion), mm | |
| 4 | 4 | int32_t | Coordinate Y of beacon (fusion), mm | |
| 8 | 4 | int32_t | Coordinate Z of beacon (fusion), mm | |
| 12 | 2 | int16_t | W field of rotation quaternion | |
| 14 | 2 | int16_t | X field of rotation quaternion | |
| 16 | 2 | int16_t | Y field of rotation quaternion | |
| 18 | 2 | int16_t | Z field of rotation quaternion | |
| 20 | 2 | int16_t | Velocity X of beacon (fusion), mm/s | |
| 22 | 2 | int16_t | Velocity Y of beacon (fusion), mm/s | |
| 24 | 2 | int16_t | Velocity Z of beacon (fusion), mm/s | |
| 26 | 2 | int16_t | Acceleration X of beacon, $mm/s^2$ | |
| 28 | 2 | int16_t | Acceleration Y of beacon, $mm/s^2$ | |
| 30 | 2 | int16_t | Acceleration Z of beacon, $mm/s^2$ | |
| 32 | 1 | uint8_t | Address of beacon | |
| 33 | 1 | 1 byte | Reserved (0) | |
| 34 | 4 | uint32_t | Timestamp, ms | |
| 38 | 4 | 4 bytes | Reserved (0) | |

Note: Quaternion is normalized to 10000 value.

Figure 4.3: Example of a Packet of processed IMU data

## 4.4 Preliminary Tests, Results and Discussion

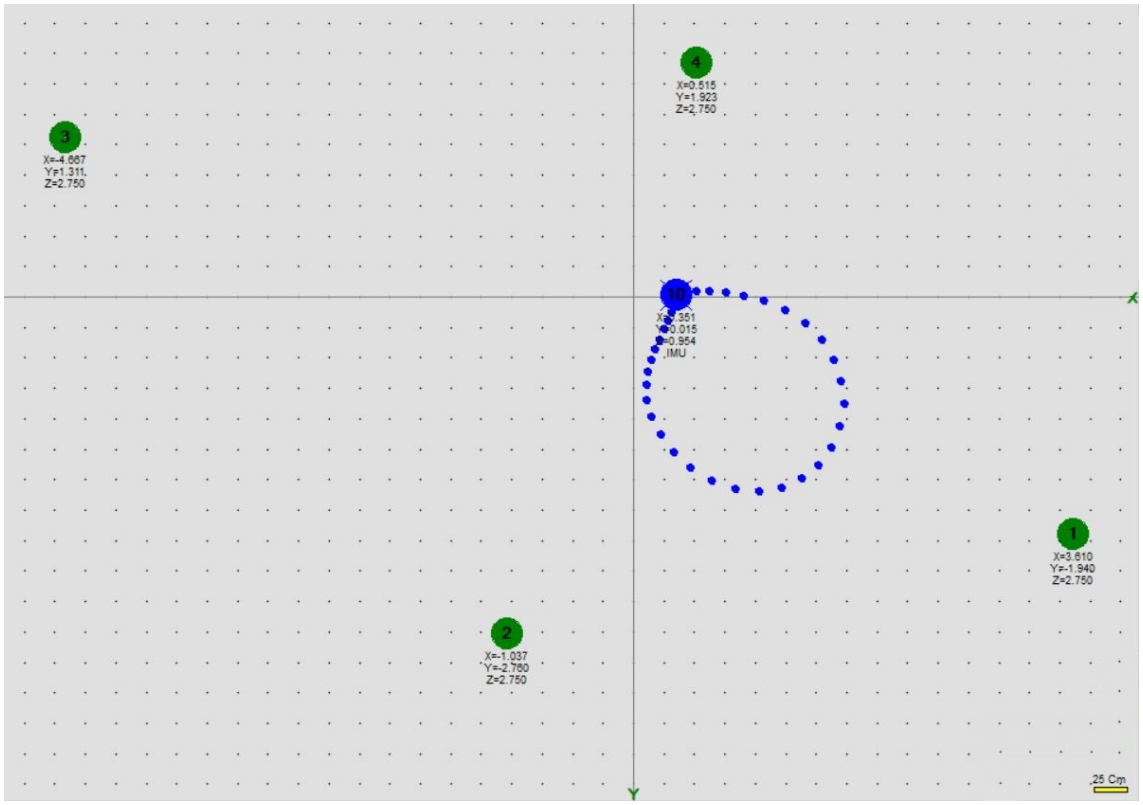
Before deciding on the communication protocol with the flight controller, some preliminary tests were done. For this, the stationary beacons were placed on the walls accordingly to the copter placing manual of *Marvelmind* [20]. Those were placed 2.75 meter height in the center of the room walls (Fig. 4.4).

Since the beacons are close to the ceiling, the sensors RX5 that are pointing to the ceiling were disabled. With this, we get less interference and less wrong measurements. Some first tests were done, and can be observed in the Figure 4.5.

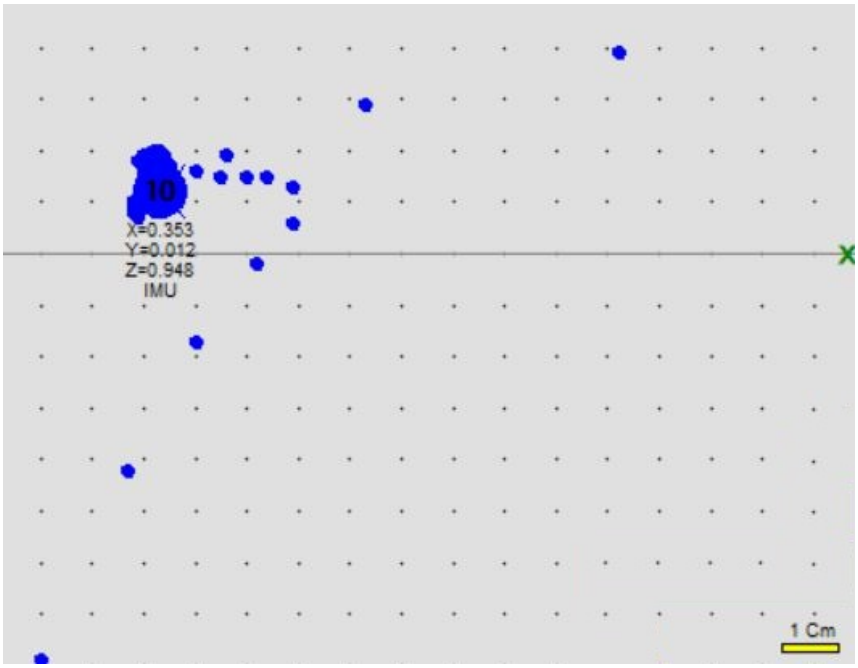
(a) Example 1

(b) Example 2

Figure 4.4: Beacons placement examples




(a) Scale of 25cm between points




(b) Scale of 1cm between points

Figure 4.5: Preliminary test example - Circle

We can achieve really accurate results, having less than +/-2cm of error. When the mobile beacon is stationary there are small deviations (Fig. 4.5b). Even though, the system is capable of tracking with precision the mobile beacon. We consider this a successful result. In chapter 7, this architecture will be more detailed, since some corrections with interference of the rotors of the drone are done.

# Chapter 5

# Drone Implementation Aspects

In this chapter, we describe the drone solution and all the necessary components, software and control of the drone, done after gathering the market options.

The main necessary components were reported in section 2.2, that include the drone frame, flight controller and the other parts needed, such as power supply. The drone match the requirements proposed by the company, reported in subsection 3.2 and is described in the next sections.

## 5.1 Hardware

Relating to customer requirements (subsection 3.2), the adopted drone is based in a Tarot 650 Sport frame. It is the result of a market survey of drones available in the market. Note that other possibilities considered are disposed in the appendix B, at the end of the document. The specifications of the chosen drone are listed below:

- **Motors:** Tarot 4006 620kV;
- **ESC:** 30A SimonK Brushless Speed Controllers;
- **Propellers:** 1355 Carbon Fibre;
- **Flight Controller:** Pixhawk Autopilot v1;
- **Battery:** Tattu 6750mAh 4S 50C.

The drone frame has four motors. Being a quadcopter we have the robustness needed to work inside a warehouse, where the boxes are droped. The chosen battery has 605g of weight. Despite this weight, the drone is capable of flying more than 20 minutes with 1kg of payload. The UAV has a total dimension of 0.65x0.65m. The compartment (space inside) has 0.2x0.2m, satisfying the customer requirements space to transport medical boxes. The motors, ESCs and the propellers are the ones recommended to the frame adopted. The flight controller is a Pixhawk based board. This is an old board, but the most stable one. Also, this brand is known for producing flight controllers for industry application. It supports the most known flight stacks like PX4 and Ardupilot.

Another component relating to our drone is the telemetry radio, that is a SiK Telemetry Radio from the brand 3DR. It is one of the easiest ways to setup a telemetry connection between the flight controller and a ground station. Finally, the drone was assembled and all the cables were tied up to the frame (figure 5.1), accordingly to pixhawk documentation [22].

(a) Components (b) Top

(c) Bottom (d) Result

Figure 5.1: Drone assemble

## 5.2 Software

In this section of the work, we decide on the main control of the drone. All software options have high impact on the solution we are developing. Firstly, we decide about how we control the drone, the firmware to install in the flight control board and then, we describe the simulation

software. Finally, since we need to monitor the drone, we need a ground station software, that is also reported.

So, in the next parts of this thesis, we clarify all the mentioned important components.

### 5.2.1 Drone Control Library

Drone technology has been developed over the years. And with that, all the sensors present in a drone were also developed. The way we see drones flying has been also upgraded. Now we do not only use an ordinary radio command, but also a telemetry radio to autonomously control the drone. These telemetry radios are used to communicate with the drone and use a library of messaging protocol. The most known one is Mavlink. It is a very lightweight messaging protocol for communicating with drones (and between onboard drone components). It is based in messages that are defined within XML files and transports data like mission protocols or parameter protocols. With this, we accomplish missions with the drone, while simultaneously see the drone position and drone sensors in real time. For that we use a ground station that provides full flight control and mission planning for any drone, installed in a remotely ground computer.

The control of the drone is in real time. And so, we assume that the orders that arrive at the drone, to move from one point to another, can be updated and the drone can either hold in air or proceed with the mission. The ground station already has these programmable options. However, the drawback of using a ground station is that we have to manually introduce all the properties of the mission we want the drone to perform. For instance, if we want to have a fully autonomous drone, we cannot just go to the ground station and select the points we want to go manually. So, to control the drone we use a platform named DronecodeSDK that is a MAVLink Library for the PX4 flight stack. The library provides a simple API for managing one or more vehicles, providing programmatic access to vehicle information and telemetry, and control over missions, movement and other operations [23].

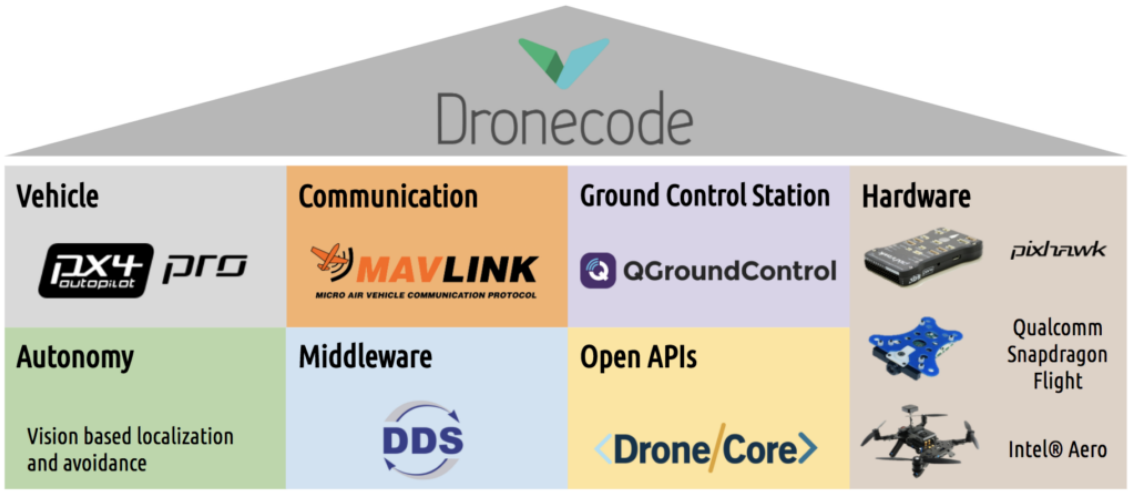



Figure 5.2: Dronecode SDK Full Platform [Source]

With these tools, we can extend the core C++ SDK using plugins in order to add any other required MAVLink API (for example, to integrate with custom cameras, gimbals, or other hardware over MAVLink). There are wrappers in Python and in Java, smoothing the user experience, if there are already present some knowledge with these languages. In our case, the SDK is used to send missions and control the drone inside a warehouse, based in a waypoint map defined by the user.

### 5.2.2 Flight Stack Firmware

In the early drones technologies, a simple drone was composed by a simple radio receiver that receives the parameters from a radio control and inputs the information to the drone, and this allowed to control each motor of the drone individually. That was a hard task to perform. Now, with a flight board the drones equipped with IMU can be more stabilized and controlled by a simple ground station. This is only possible with a core component that is the flight stack firmware. This is a group of libraries needed to be installed in the flight control board of the drone. It is in charge of correctly access all the sensors information, and to control the drone and all his actuators, depend on what information is arrived on it.

Since the board is a Pixhawk, we only have two recognized options to install - PX4 and Ardupilot. PX4 is a platform that ensures everything needed for a complete UAV solution. It is supported by *BSD 3-clause license*, that makes it a good option to the customers plan, since it can be sold as a commercial product. Ardupilot already has an integration with the *Marvelmind* system, but this only works with a ground station where a user inputs manually all the waypoints [24]. Since the company wanted a profitable integration, this was a challenge to work with. On the other hand, Ardupilot is more community friendly software, from where PX4 come from. PX4 flight stack is the largest industry-backed development community in the drone space today and is the only firmware supported by the DronecodeSDK, making it our decision/selection.

### 5.2.3 Ground Station and Simulation

As for ground station, to see all the real time information about the drone, we have two valuables options: QGround Control and Mission Planner. The first provides full flight control and vehicle setup for PX4 and Ardupilot powered vehicles. It provides easy and straightforward usage for beginners, while still delivering high end feature support for experienced users [25]. The second is not compatible with PX4 flight stack, making QGround Control the solution if we eventually need to do a simple tests with the drone.

Finally, to simulate the code that controls the drone mission we also have several options, but only one will be used: Gazebo. It is a powerful 3D simulation environment that is particularly suitable for testing object-avoidance and computer vision. It is recommended by PX4 community.

# Chapter 6

# Drone Control Architecture

We now introduce the architecture of the drone control. The system is composed by two main components (Fig. 6.1).

On one hand, we have the drone that we simulate in Gazebo. For that, two options are available to test. We can work with SITL - software in the loop in order to reproduce the PX4 drone responses, or use HITL - hardware in the loop. The second is often used to test the releases of the software directly in the flight controller. Since the procedure we do in the drone control is constituted by simple orders (take-off, land, go to location,...) that are already developed in the PX4 firmware, we test the drone with SITL.

On the other hand, we have the ground station that is in the same computer where the SITL is running. As referred to in the last chapter, the program used to do mission planning is the DronecodeSDK. That it is the core of the project. It is where the project mission planning is refined and the waypoint trajectory designed.

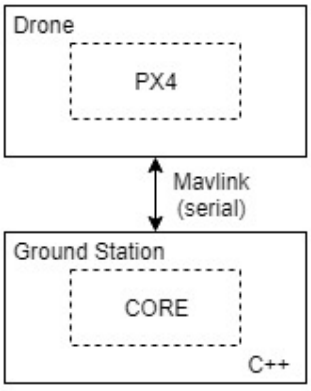



Figure 6.1: Initial Control Architecture

These main components connect to each other via Mavlink (Fig. 6.1). As invoked before, Mavlink stands for Micro Air Vehicle Link and it is a protocol for communicating with small unmanned vehicle. It is designed as a header-only message marshaling library. For simulation purposes, this connection is established via UDP (User Datagram Protocol) protocol, since the

drone and the core are in the same computer. When developing the drone in a real world, this connection will be via Serial, accomplished by the telemetry radio. Finally, TCP - Transmission Control Protocol is also available for Wi-Fi connections via MAVLink.

The Core is developed in C++ and has the option to get information about vehicles (vendor, software versions, product versions etc), get vehicle telemetry and state information (e.g. battery, GPS, RC connection, flight mode etc.) and set telemetry update rates. Also, the important commands like arm, disarm, kill, takeoff, land and return to launch can be send to the controller. Finally, we create and manage missions.

However, as mentioned in section 5.2.1, library wrappers are also available. And with that, a more simple user experience is available. The wrapper is based on a gRPC client communicating with the gRPC server written in C++. The server, also called backend, acts as the core cited before. Using an interface client-server we control the drone using a language wrapper available, that have almost all the same functionalities of the core. The client, also known as frontend, allows the user to do all the mission planning (Fig. 6.3).

The only drawback for using the server is the lack of a communication protocol required for this project. The backend only supports Mavlink communication with the drone via UDP, making it an useless option if serial is essential. But since the serial communication is already implemented in the core, explained in the next section, rebuild it should not be hard. Nonetheless, the control can be simulated in the computer, building the PX4 to run in our PC. Then, this build is in charge of disposing all the simulated sensors and actuators and send to Gazebo software, the simulation of the environment.

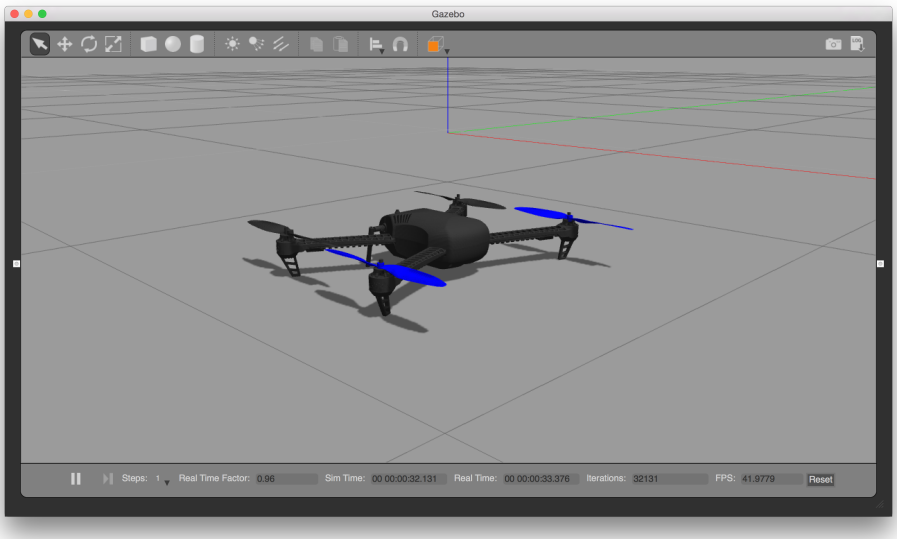



Figure 6.2: Gazebo [Source]

## 6.1 Backend Serial Communication

The DronecodeSDK C++ Core is the main library class. Access to drone information and control objects is provided by plugins (e.g. Telemetry, Action, Mission etc.). Plugin objects are instantiated with a specific *System* object. When initializing the system, we provide information about the protocol communication and port to be accessed. There are three (synchronous) connection methods: UDP, TCP and serial. The serial is the one that we use to connect to drone, when using telemetry radio, but not allowed when working in backend mode.

Detailing the core library, we have *telemetry*, *action* and *mission* classes. The first one is used to get vehicle telemetry, including state and flight mode information. The *action* class is used for commanding the vehicle to arm, takeoff, land, return home and land, disarm and kill. The last one allows to create, upload, download, run, pause, restart, and track missions. Missions can have multiple "mission items", each which may specify a position, altitude, fly-through behaviour, camera action, gimbal position, and the speed to use when traveling to the next position. For our work purposes we just consider the position, altitude and fly-through behaviour, since our drone do not have any gimbal or camera.

When considering the Backend and Frontend protocol, we still have the same plugin architecture. In our case, we work with Python. This wrapper benefits with Asyncio library. Asyncio provides a set of high-level APIs to run Python coroutines concurrently and still have full control over their execution. These executions are preferable for stopping and continuing a routine and easier to create loops, which are useful in our application method.

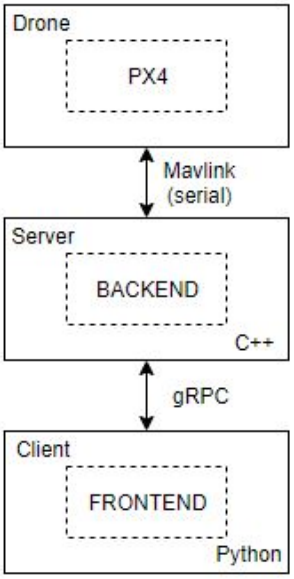



Figure 6.3: System Architecture

We need to build and run the backend which is the gRPC server that the Python wrapper will connect to. But by default, the backend connects using MAVLink on UDP port 14540 which is

running by default when PX4 is running in SITL (software in the loop simulation). In order to make it convenient for our goal, we need to add the functionality to connect via serial.

More detailed, in the class backend.cpp, when we call the function *connect()*, we simply change the port (used in UDP connection) to baudrate, since serial works with baudrates. When connecting via USB cable to the flight controller the baudrate preferred is 57600 baud and when using the telemetry radio we define 115400 baud as the speed of the connection.

Listing 6.1: Funtion connect

```
void connect(const int mavlink_listen_port)
    {
        _connection_initiator.start(_dc, baudrate);
        _connection_initiator.wait();
    }
```

Then, inside connection_initiator.h, the backend server default connection uses the function *add_udp_connection()* and we change it for *add_serial_connection()*, like in the next code excerpt:

Listing 6.2: Function start

```
bool start(DronecodeSDK &dc, const int baudrate)
    {
        init_mutex();
        init_timeout_logging(dc);
        _discovery_future = wrapped_register_on_discover(dc);

        if (!add_serial_connection(dc,baudrate)) {
            return false;
        }
        return true;
    }
```

Finally, once this modification is done in the SDK library, we can build and run backend server. This allows working with a real drone in an actual environment, using any of the language wrappers available.

## 6.2 Mission Planning

Inside a warehouse there are a lot of corridors. These corridors are usually unobstructed and free of persons. For simplicity and ideally purposes, we imagine these corridors free of persons or other type of distraction that could interfere in a normal fly of the drone.

As mentioned in the last section (6.1), mission planning is composed by a waypoint "vector" (*missionItem*) from where we want the drone to maneuver. Typically, we associate drones flying outdoor, having the default localization based on GPS devices. This involves a latitude, longitude and height (in meters) localization parameters. However, even with a high accuracy that we have

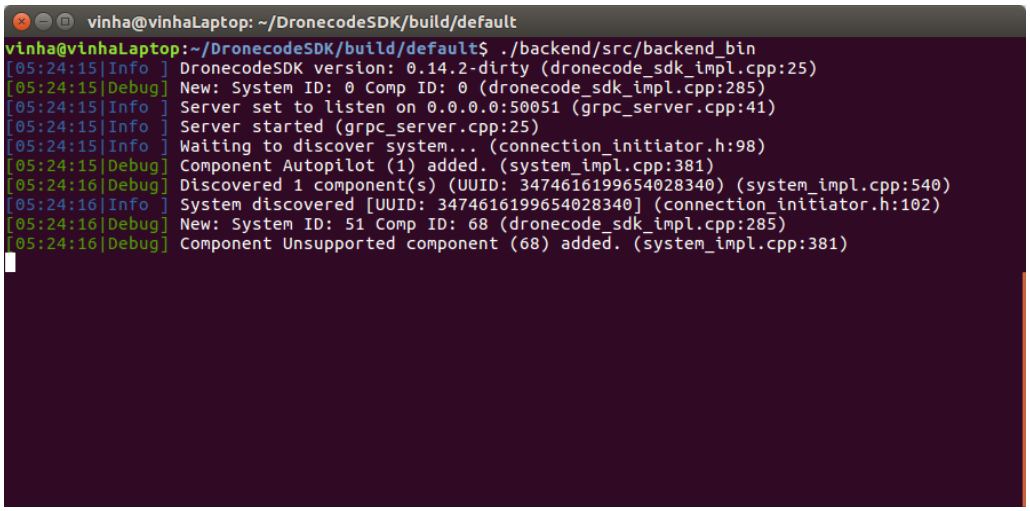



Figure 6.4: Successful connection of the backend via Serial

with these coordinates, inside a warehouse we demand the use of Cartesian coordinates (x,y,z in meters) for better understanding of the positions of the space.

According to the next equations, we convert the GPS coordinates to Cartesian coordinates, by using reference coordinates for latitude and longitude [21].

$$Lat = Lat_{ref} + y * 9.013373 \tag{6.1}$$

$$Long = Long_{ref} + \frac{x * 8.98315}{cos(\frac{Lat_{ref}}{1000000})} \tag{6.2}$$

The GPS coordinates and the references are in microdegrees and $x$ and $y$ values are in meters and correspond to latitude and longitude, respectively. The height is the same in the two coordinates quotations.

To test the equations present in the *Marvelmind* manual [21], we defined two geographic coordinates as reference and then 6 randoms points separated by 10 meters. The idea was to translate the geographic coordinates to cartesian coordinates and then print them in *Google Maps*.

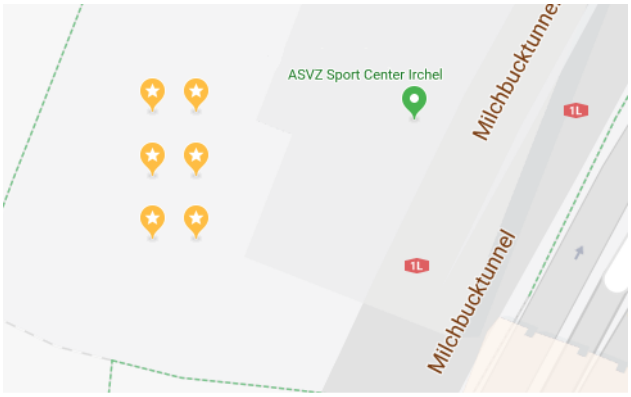



Figure 6.5: *Google Maps* test

The result is in figure 6.5. The points are not spaced as they should be. The reason is that the equations were wrong in the *Marvelmind* manual. After that, in contact with the *Marvelmind* support, they fixed the problem. Note that the equations 6.1 and 6.2 are correct, and the issue was fixed in the manual.

A simple example of a mission planning using Python is shown below:

Listing 6.3: Mission example loop

```
async def run():
    mission_items = []
    mission_items.append(MissionItem(47.398039859999997,
                                     8.5455725400000002,
                                     2,
                                     1.5,
                                     True,
                                     float('nan'),
                                     float('nan'),
                                     MissionItem.CameraAction.NONE,
                                     float('nan'),
                                     float('nan')))
    await drone.mission.set_return_to_launch_after_mission(True)
    await drone.mission.upload_mission(mission_items)
    await drone.action.arm()
    await drone.mission.start_mission()
```

This code excerpt demonstrates the main loop known by the function run(). Firstly, we create a simple vector of *mission_items*. Then, we append a *MissionItem* that has the latitude, longitude, height, velocity and other parameters that we do not use, declared as *null*. We set the mission as return to launch when the movement to the point (or points) desired is completed. Finally, we upload the mission to the drone and arm the drone to start the mission.

With the thought in a later application that reads waypoints from a user input, we create a function that converts points in meters to points in GPS coordinates. These coordinates come from a text file, but later can be programmed directly by a user interface. Then, using the equations mentioned before (6.1 and 6.2), we translate the coordinates. A function was created for this purpose and to read the points from a text file.

For simple use experience, only latitude and longitude parameters are required, but it is just a matter of adding more parameters to the text file in order to have height and velocity values before enter. These values are set by a default value of 2 meters and 1.5 meters per second, respectively, when creating the mission.

The next loop is joined to the main run() in order to wait for new waypoints to arrive and create a new mission planning:

Listing 6.4: Waypoint introduction loop

```
while items == []:
  items = read_safeWaypoints_from_txt("PointsGPS.txt")
  await asyncio.sleep(5)
  last_items = items
  for i in range(0, len(items)):
     mission_items.append(MissionItem(float(items[i][0]),
                                      float(items[i][1]),
                                      8,
                                      25,
                                      True,
                                      float('nan'),
                                      float('nan'),
                                      MissionItem.CameraAction.NONE,
                                      float('nan'),
                                      float('nan')))
  await drone.mission.set_return_to_launch_after_mission(False)
  await drone.mission.upload_mission(mission_items)
```

## 6.3 Path Planning

We begin the missions by adding the waypoints desired by a user. Inside this project, the mission planning is defined by a path planning. In other words, if we want to go from a point A to D, the drone cannot just simply go through the wall or the warehouse shelfs. So, we need to make a path planning. With this scope, a path finding algorithm is demanded. A* is a search algorithm recognized and known one. This find the shortest path from a point to another, given points from a map (Fig. 6.6).

The two methods have weight associated with the cross of some points desired. With this aspect, we can give more weight to some corridors, imposing the algorithm to choose the "cheaper" path, depending on what weights it chooses. A* is considered a "best first search" because it greedily chooses which vertex to explore next, according to the value of f(v) [f(v) = h(v) + g(v)] - where h is the heuristic and g is the cost so far. This is the main aspect to our decision for path planning. The first version code of the algorithm is based in the pseudo code found in *Wikipedia* page [26] and the warehouse map is defined by points spaced by one meter. To make the spaces that the drone cannot fly, we simply remove those points from the vector of the points from the map variable. Given the point of start and the finish point, the algorithm searches the shortest path relative to the last point. In the end, we have a path made by points separated by one meter. Since the drone interprets all the waypoints with a stop, in adjacent points we would have a lot of acceleration and deceleration of the drone. A compact function of the path is essential in order to improve the mission planning.

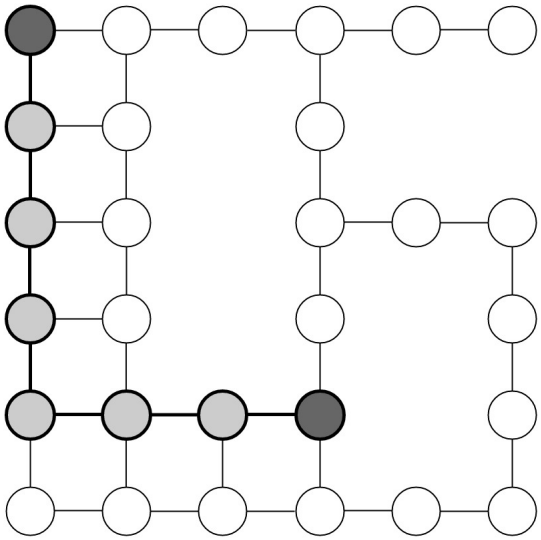


Figure 6.6: Example of grid map

However, this implementation is not suitable for the purposes of the project because of the limitations of the waypoints distances and travel directions. The drone can only go to the front, back, left or right. In the version two of the algorithm code, a change to the map coordinates variable is done. The map is now composed by edges and points coordinates. This allows to have points spaced with different values and corridors defined by the map properties of the warehouse (Fig. 6.7). Also, all directions are allowed in this version.

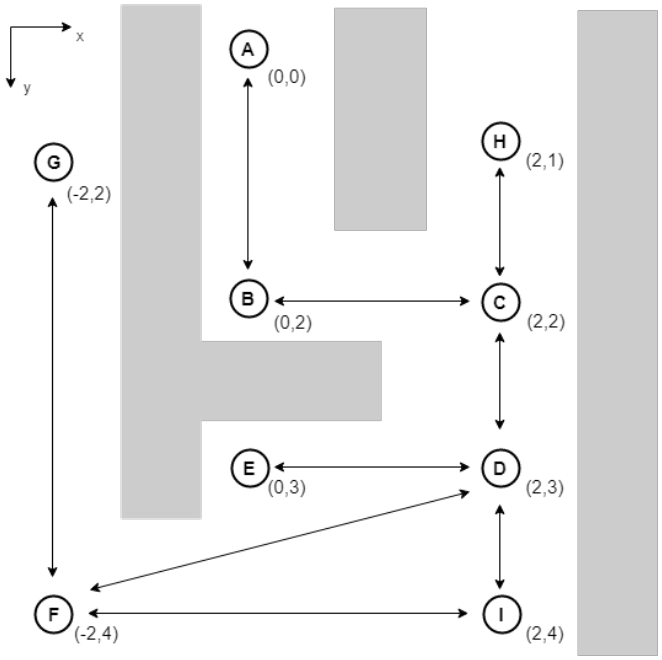



Figure 6.7: Example of a warehouse map - *Version 2*

Firstly, we declare the edges and the coordinates (more information below: [listing 6.6 and

6.5]). For example, the point A has the coordinates (0,0) and point F has coordinates (-2,4). When we are in the point D, the edges available to go are E, F and I. These coordinates and edges have to be declared within a variable, as the algorithm calculates the distances from each points coordinates. These have to be changed every time the layout of the warehouse is also changed, but gives more freedom to choose the waypoints. This implementation can be easily rebuild, making a good solution for customer requirements.

Listing 6.5: Edges declaration

```
map.edges = {
    'A': ['B'],
    'B': ['A', 'C'],
    'C': ['B', 'D', 'H'],
    'D': ['C', 'E', 'F', 'I'],
    'E': ['D'],
    'F': ['D', 'G', 'I'],
    'G': ['F'],
    'H': ['C'],
    'I': ['D', 'F']
    }
```

Listing 6.6: Coordinates declaration

```
map.coord = {
    'A': (0, 0),
    'B': (0, 2),
    'C': (2, 2),
    'D': (2, 3),
    'E': (0, 3),
    'F': (-2, 4),
    'G': (-2, 2),
    'H': (2, 1),
    'I': (2, 4)
    }
```

Then, after the variables created, the same algorithm as before is implemented. The first node is where we start the path. The distance from a node is the distance from the initial node to this node. A* algorithm will assign some initial distance values and will try to improve them step by step. This method is more detailed in the pseudo code available in the Wikipedia page about Dijkstra's algorithm [26].

After waypoints input from an user, like a function *go_to(initial_point, last_point)*, the algorithm starts to calculate the path. Then, the program converts the points from meters to degrees, as talked in last section. Finally, the mission is created and the upload done to the drone. When the mission is completed and the drone arrives to the last point, the loop checks if there is another user input. If not, the drone lands in that point and waits for a new mission.

Inspecting the figure 6.8, we can see on the left the ground station tracing the movement of the drone. On the right upside we can monitor the mission status, developed in python. Finally, on the lower side we have the simulation of the drone in Gazebo. As we can see, the mission have already been finished. No new waypoints have been added, so the drone finished the mission by landing. Note that the distances between waypoints were set to more than 10 meters, only to allow us to examine (in the ground station) the results with more accuracy.

## 6.4 Discussion

In this chapter the mission control for the drone was detailed. It consists of reading waypoints from an user input in order to make a new mission to the drone. We defined the main program-

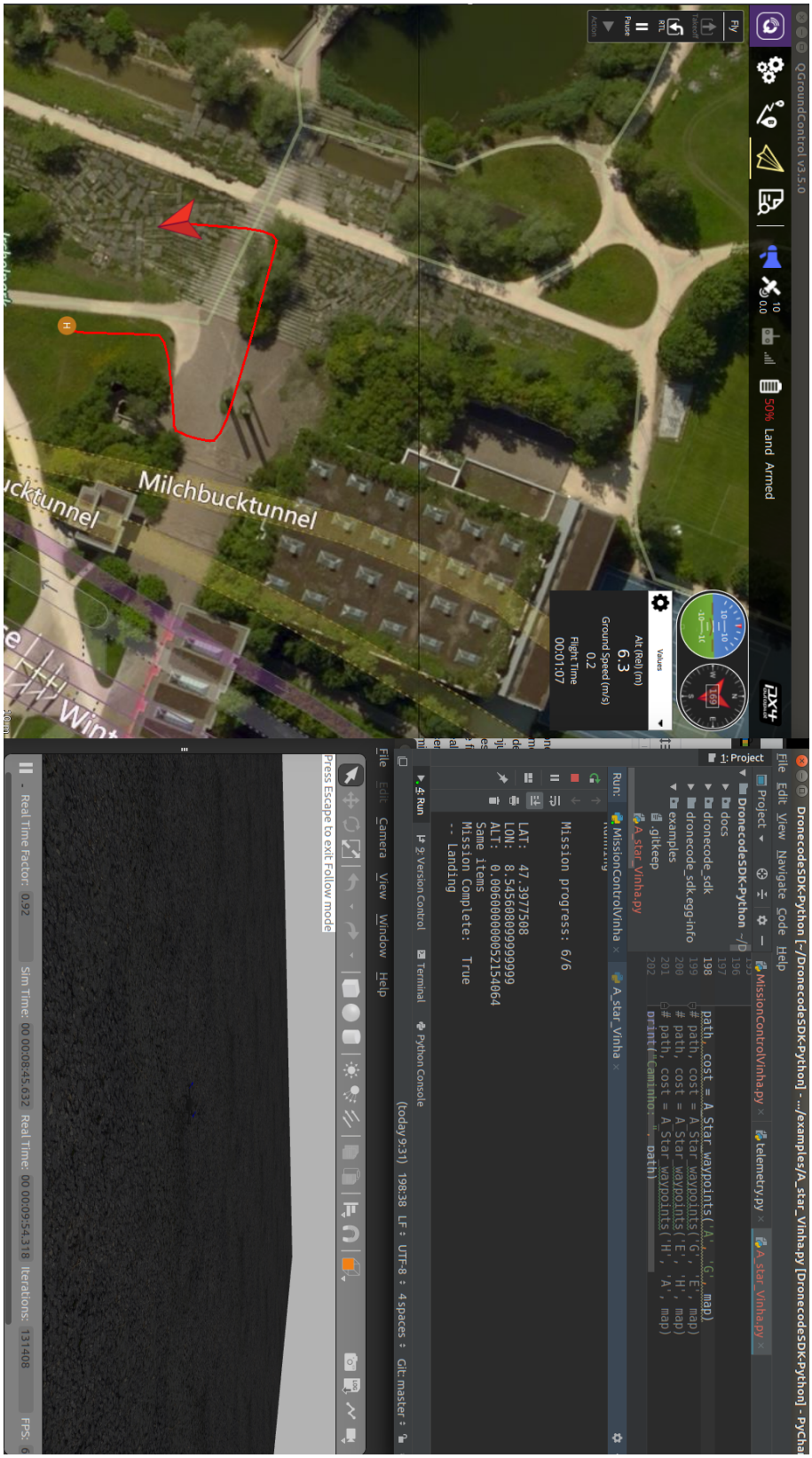



Figure 6.8: Simulation Example

ming language, that is Python. We described the main architecture when developing the drone management, that used a backend solution.

Also, a waypoint segmentation algorithm was developed. Even though the path algorithm is limited when choosing the waypoints, we can conclude that we can always find the best solution when navigating throughout the corridors. This can be applied to any warehouse, after the points and edges being declared.

An integration with any drone equipped with PX4 is allowed. This is an open entry to combine with an interface that can have access to the spaces available in real time and update the edges and points in real time, using a data base for example. Simple commands can be added to meet the customer requirements for a commercial product.

Relating to figure 6.8, we can see that the management of the drone control is successfully implemented. We consider the results to be very satisfying and this mode to be an effective way to handle warehouse travelling.

# Chapter 7

# System Integration

In this chapter, we integrate the localization measurements into the drone firmware. We start by describing the data exchange method with the flight controller. Then, we describe the fusion of the localization data and a survey of the parameters. Finally, simulation and tests are done to verify the system.

## 7.1 Location Update

The core objective of the project is to achieve capability to fly indoors. As mentioned in chapter [4], the indoor device consists of a mobile beacon to be placed in the drone. The drone can receive the localization data through two communication protocols (section 4.3) and fly like in an outdoor environment.

However, these two protocols require some work to be compatible with the PX4 firmware. Since the PX4 firmware present in the *Github* page [27] already has a driver for GPS communications with NMEA protocol, the reconstruction of these driver was not expected be hard. On the other hand, when using the *Marvelmind* packets, a new driver to parse the messages is needed to construct the driver. So, using the NMEA protocol came for the first option.

The connection among the pixhawk and the mobile beacon is executed by two cables, that connect the GPS port of the pixhawk and two pins of the mobile beacon, relative to the next figure. These connections can be consulted in *Marvelmind* manual [21].

The driver that we use in our project to interpret the NMEA messages is the *Astech Driver*. Since the core of the driver is already implemented, only the part that parses certain messages is required. As mentioned in section 4.3, the messages that the *Marvelmind* system outputs are four and the message required for navigation is $GPGGA - Global Positioning System Fix Data.

The driver is developed in a way that firstly reads the header of the message, according to the next sentence:

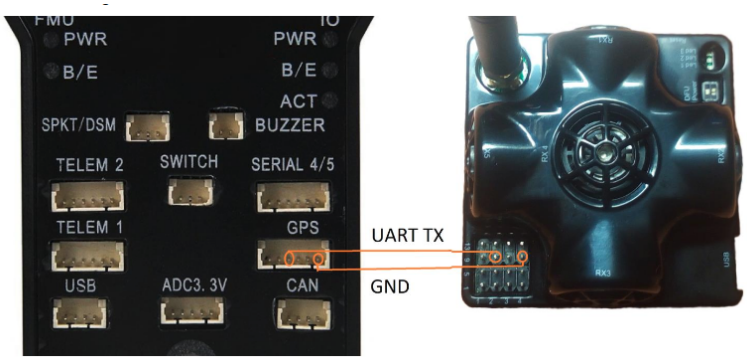



Figure 7.1: Connection between GPS port and beacon pins

Listing 7.1: Message parse

```
if ((memcmp(_rx_buffer + 3, "GGA,", 3) == 0) && (uiCalcComma == 14)
{
(...)
}
```

If the header is equal to a sentence and has "x" commas, it is a certain message and we start to isolate the parameters of the messages in between the commas, depending on the message. The baud rate selected is 115kbps and the driver is automatic chosen by the PX4 firmware when the system starts and the messages start to arrive.

Then, inside this condition, the parameters are updated according to their meaning inside the message commas, like in the next sentence:

Listing 7.2: Message parameter update example

```
if (bufptr && *(++bufptr) != ',')
{
      lat = strtod(bufptr, &endp); bufptr = endp;
}
```

Note that each time the buffer is read, it goes to the next comma. Also, the parameter North/-South and East/West determines if *lat/lon* is positive or negative. In the end, *lat* and *lon* are converted from degrees, minutes and seconds to degrees.

We integrated as quoted in the PX4 development guide [28] and then uploaded to the pixhawk board through the terminal of *Linux* and a USB cable. The version of the PX4 firmware is v1.8.2 (stable version). Later, we connected the board via telemetry radio to QGround Control and started the first configuration and calibration of the board [29].

## 7.2 Sensors Fusion

With the firmware installed and configured, we started to do a more advanced configuration in order to integrate PX4 and *Marvelmind*. In this section, we start with a small introduction of the

flight stack and then we describe sensors fusion.

The following diagram (Fig. 7.2) shows an overview of the building blocks of the flight stack. It contains the full pipeline from sensors, RC input and autonomous flight control (Navigator), down to the actuator.

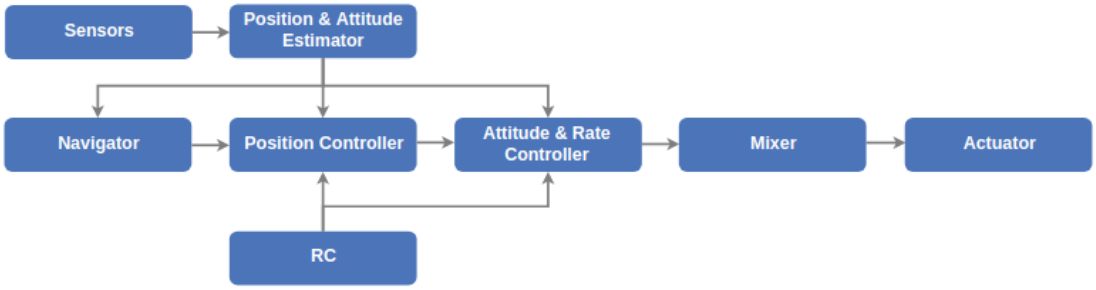



Figure 7.2: PX4 High-Level Flight Stack

The Pixhawk is equipped with some useful sensors for this project. These are the barometer, accelerometer, gyroscope, etc. The estimator takes one or more sensor inputs, combines them (for example IMO sensor data), and computes a vehicle state (for example the attitude). Another sensor that can be used is the GPS, that in our case is the "indoor GPS". After the localization data processed, these are fused with other measurements from the other sensors [30].

The controller is a component that takes a setpoint and a measurement or estimated state (process variable) as input. Its goal is to adjust the value of the process variable such that it matches the setpoint. The output is a correction to eventually reach that setpoint. For example the position controller takes position setpoints as inputs, the process variable is the currently estimated position, and the output is an attitude and thrust setpoint that move the vehicle towards the desired position.

However, the implementation with *Marvelmind* has never been done before with PX4. The fusion between GPS location data and the IMU is carried out by the precision and errors that a conventional GPS device has. And so, the parameters of the PX4 system are subject to change to get the best estimation of the position of the drone. They are used to store information about the configuration of the system algorithm.

The parameters are organized by their meaning to a relative sensor. In a initial part, only the sensors relative to the estimator of the position will be changed. The mission and controller parameters will still be the same, since the difference to an outdoor environment come from the localization protocol and we want the same performance when moving the drone around the way-points. For that, the parameters that we set up are relative to EKF2. EKF2 is an extended Kalman filter estimating attitude, 3D position, velocity and wind states, developed in the flight stack present in the PX4 firmware [31]. For introduction to this subject, a lecture from Extend Kalman Filter was attended [32].

The parameters subject to change are present in the table 7.1. The primary source of height data when flying outdoors is the barometer, however in the indoor environment, the barometer

measurements are not accurate. With that in mind, the primary source of height data was changed to GPS and since we cannot disable the barometer, we had to increase the measurement noise for barometric altitude to the maximum allowed. Also, since the GPS used has more precision, the measurement noise for GPS position had to be decreased from 1 meter to 0.08 meters. This value is chosen accordingly to the maximum error of the *Marvelmind* system that is +/- 2cm. Finally, all the GPS checks were disabled, because some values of the conventional GPS were not present in our implementation.

| Parameter | Description | Value |
|---|---|---|
| EKF2_BARO_NOISE | Measurement noise for barometric altitude | 15 m |
| EKF2_GPS_CHECK | Integer bitmask controlling GPS checks | 0: No checks |
| EKF2_GPS_P_GATE | Gate size for GPS horizontal position fusion | 4 |
| EKF2_GPS_P_NOISE | Measurement noise for GPS position | 0.08 m |
| EKF2_HGT_MODE | Determines the primary source of height data used by the EKF | 1: GPS |

Table 7.1: Important parameters configuration

## 7.3 Simulation and Data Analysis

A typical GPS device can output compass data to any pixhawk. However, our beacon do not have this functionality. Nevertheless, the pixhawk board is equipped with a compass, that will be used for our system to find its orientation. Because of that, we have to adjust the submap created in the *Marvelmind* system to align the geographic North.

When we move the drone aligned with the real North (Fig. 7.3), we can see in the dashboard that it is moved along the $y$ axis (Fig. 7.4). The same happens when we move the drone along the $x$ axis, that means East/West side.

The system is based in NED (North East Down) coordinates. It uses a coordinate system where the X-axis is pointing towards the true North, the Y-axis is pointing East and the Z-axis is pointing Down, completing the right-hand rule.

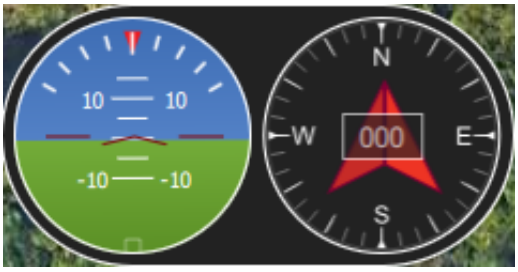



Figure 7.3: Gyroscope and Compass alignment on Pixhawk - *North*

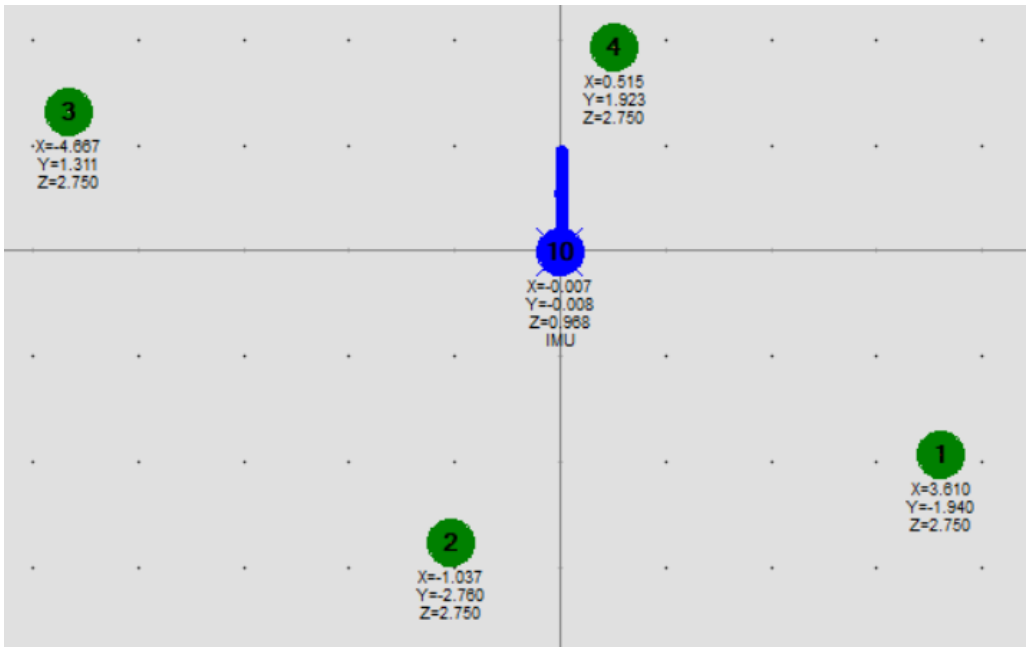



Figure 7.4: Submap test on *Marvelmind* Dashboard - *North test*

After the integration of the system, It was time to do a preliminary simulation of the movements. The PX4 is equipped with flight logs, where we see all parameters and variables of the system. PX4 logs detailed aircraft state and sensor data, which can be used to analyze performance issues.

To analyze the logs, we used a program named *PYflight Analyzer*, where we draw all the necessary values and data. Since the first time analyzing the integration of the two systems, we had to evaluate the raw data coming from the *Marvelmind* system that enters in the Pixhawk. This variable is named *vehicle_gps_position* and has the latitude, longitude and altitude coming directly from the beacon. After pixhawk receives this data, it is fused with data from the IMU sensor and the variable *vehicle_global_position* is created. Finally, since the autonomous navigator of PX4 works with local variables, the attribute *vehicle_local_position* is created exposing the x,y and z coordinates of the position.

We simulated a simple trajectory, where the drone moves from a initial point (0,0,0) to another coordinate, waits there approximately 10 seconds and then returns to the initial point. We have done different simulations that are illustrated in the next figures. The drone was moved with hands and no propellers have been set in the drone. As mentioned in last section, the drone was moved accordingly to North/South and East/West coordinates, that corresponds to *x*/*y*, *lat*/*lon* coordinates, respectively. This allowed a better analysis of the data log, when studying the data from the graphics.

- **Movement of 30cm along the North/South axis:**

Figure 7.5 displays the raw data coming from the beacon. Since we move the drone in the North/South axis, it has a step in the latitude graph, while the longitude graph has small variations.

Also, all the figures show quantified values coming from the beacons measurements. In the altitude chart (Fig. 7.5c), the measurements are in millimeters and had a variation when moving the drone from the initial position of about 2 centimeters. Then, after the fusion with data coming from the IMU sensor a new variable is created (Fig. 7.6). Now, these values are continuous and more stable. We can observe the step in the latitude graph and the longitude and altitude stable, as expected. In this variable, the altitude is shown in meters. We cannot conclude if the measurements are correct, because they are shown in geographic coordinates.

Finally, we have a more understandable chart, since the variables appear in meters (Fig. 7.7). Observing Figure 7.7, we can recognize that the values of the *X* and *Y* position are correct, and the system is capable of correctly fuse the indoor location with IMU measurements. The maximum error in the *Y* axis was +/- 4cm. It is not perfect but enough for our application. Nevertheless, these errors occur when we started to move the drone. This can be explained by the fusion with the IMU measurements that predicted a small movement in that way, even though no movement was done in that way.

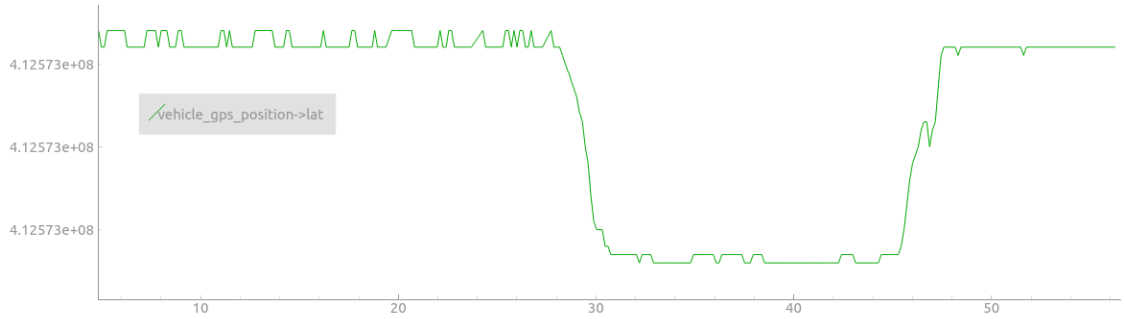
Note that this is not a true test, since the drone is not working, only moved by hand.

- **Movement of 50cm along East/West axis:**

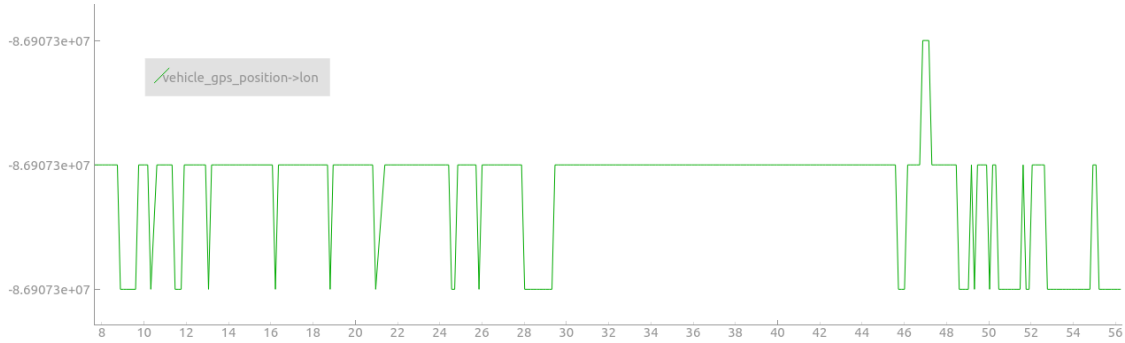
As expected, accordingly to Figure 7.8, there is a step in the longitude graphic. We assume this is correct, since we moved the drone in the East/West side. There is small perturbations in the latitude and altitude charts. In Figure 7.8c we see a fluctuation of about 60 millimeters. Again, these data fused with accelerometer and gyroscope readings created the plots in Figure 7.9. We observe the same small fluctuations in the latitude and altitude charts. Accordingly to the longitude plot (Fig. 7.9b), there is a small overshot when reaching the maximum movement. Finally, these values are converted to cartesian coordinates, shown in Figure 7.10. Analyzing the Y variable, we can conclude that the movement only reached 35 centimeters. In section 7.10, we mentioned an error with the equations related to the *Marvelmind* system that translate the real position of the mobile beacon (*x,y,z*) coordinates to *lat, lon* and altitude. After this issue being fixed, *Marvelmind* uploaded a new version of firmware and this was fixed (results in figure 7.11).

- **Movement of 30cm in altitude:**

A test on the altitude was also done. Relative to the raw data (Fig. 7.12a), we see 2 overshoots of about 10 cm in altitude, but a stable difference of 30cm from the initial point. On the other hand, when comparing the local variable of the altitude (Fig. 7.12b), we see a step in the altitude of about 0.3 meter, that result in a successful test. It has a little overshot, that was rapidly compensated with the fusion of the sensors to the correct altitude. Note that the system works with NED coordinates, so altitude Z takes negative values. The result is in the figure 7.12.



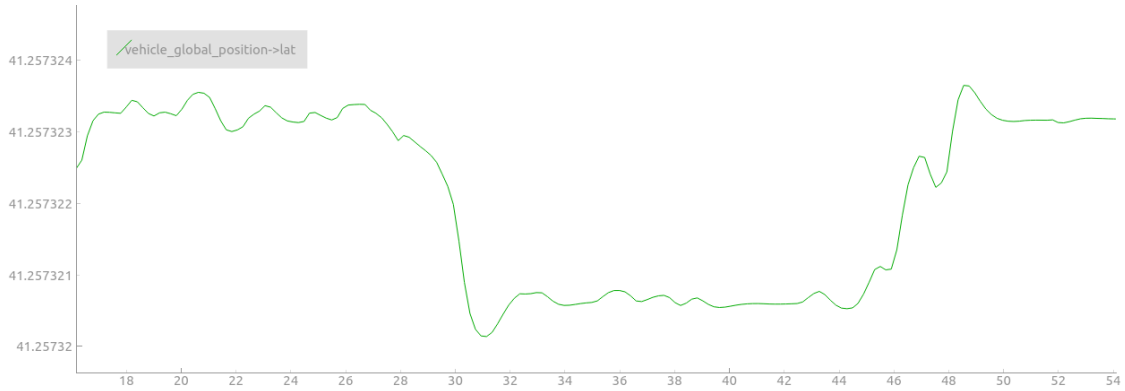

(a) Latitude



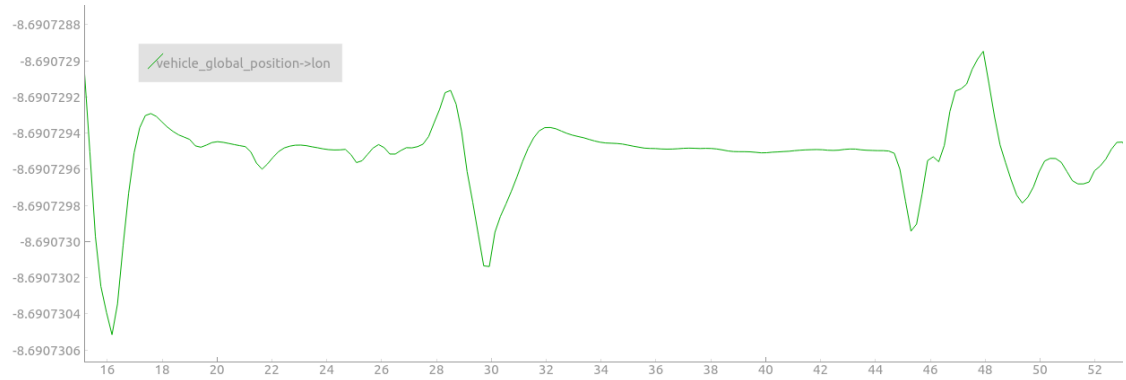

(b) Longitude


(c) Altitude

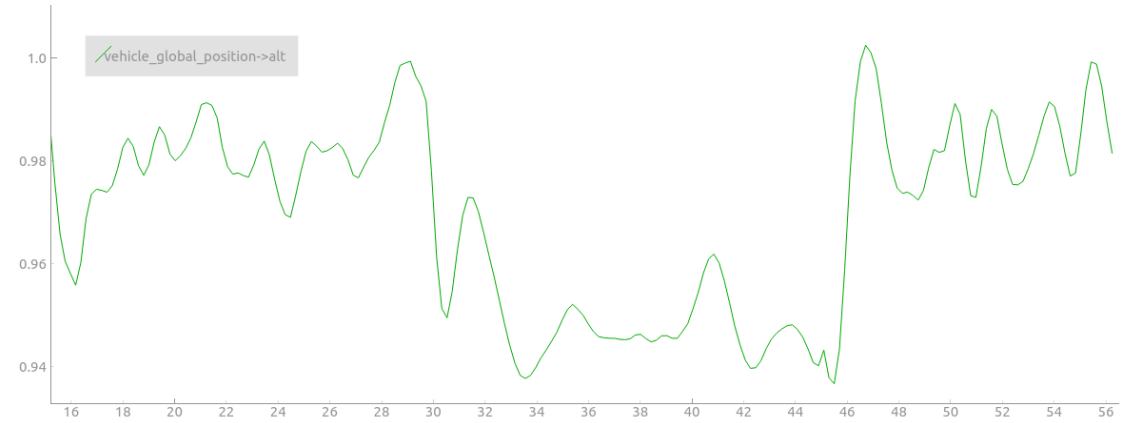
Figure 7.5: Vehicle GPS Position - Movement along North/South axis




(a) Latitude




(b) Longitude




(c) Altitude

Figure 7.6: Vehicle Global Position - Movement along North/South axis

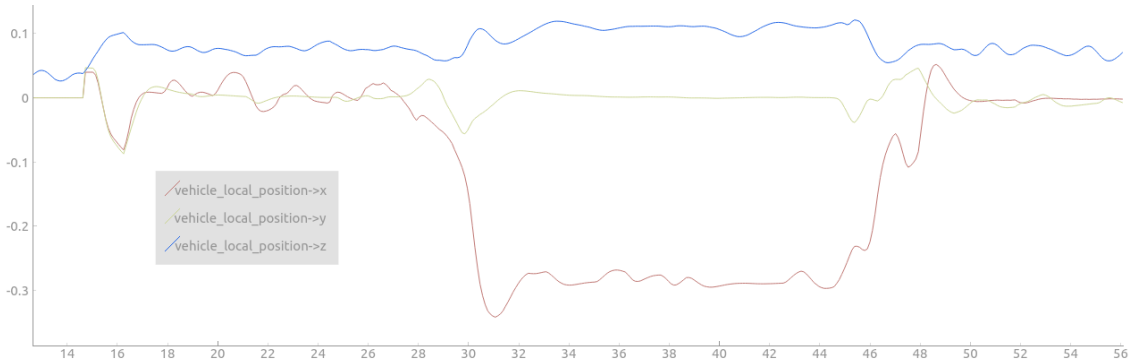
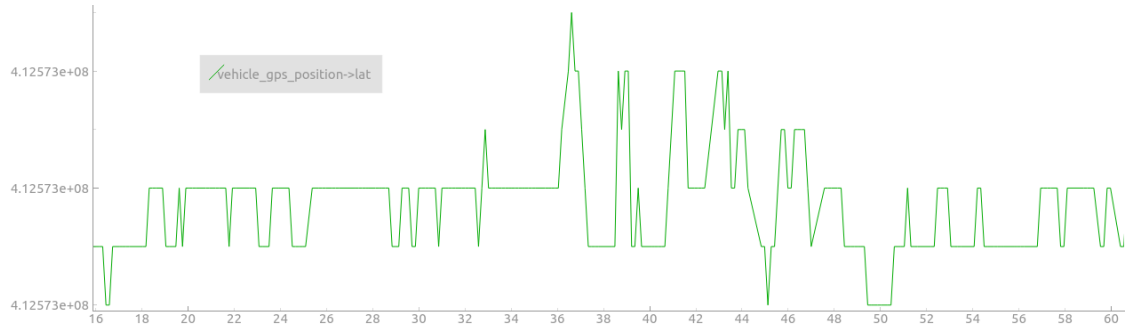
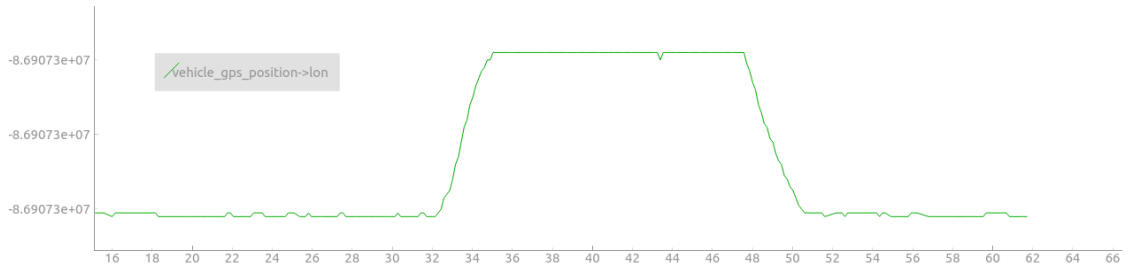


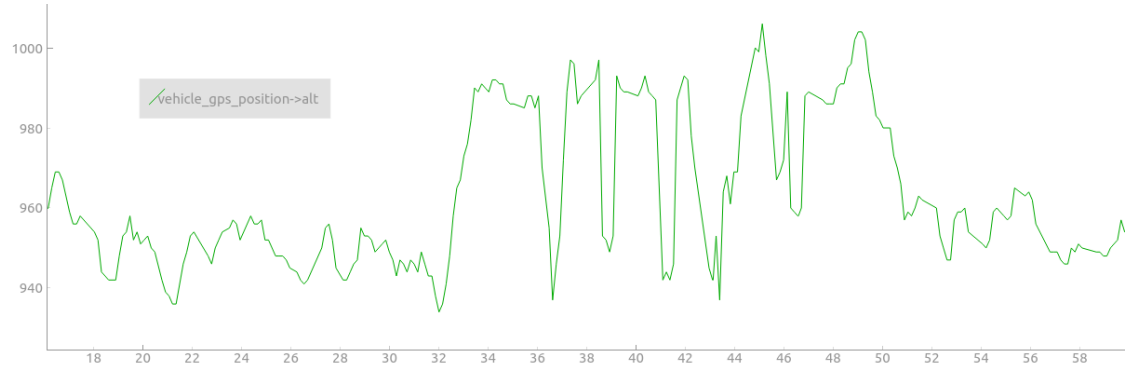

Figure 7.7: Vehicle Local Position - Movement along North/South axis




(a) Latitude

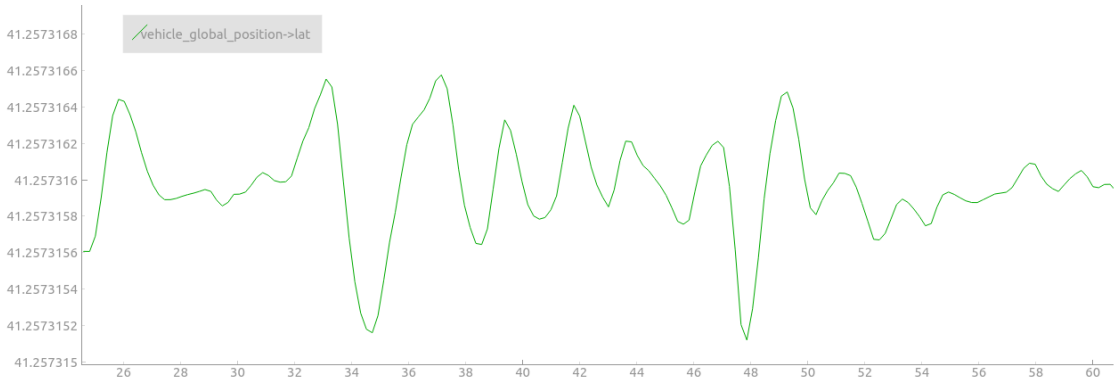



(b) Longitude

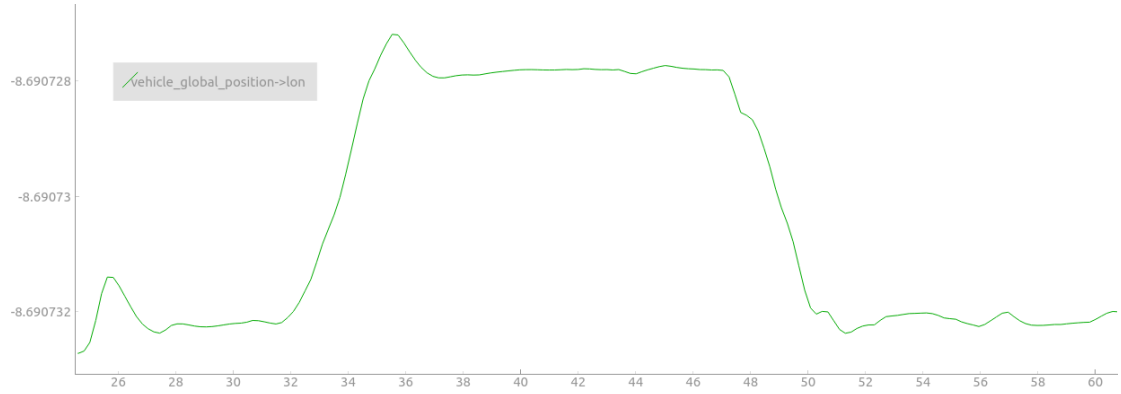



(c) Altitude

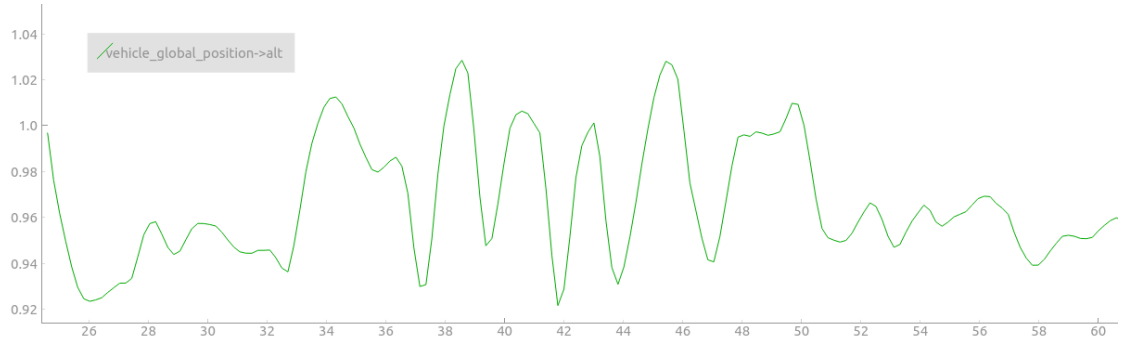
Figure 7.8: Vehicle GPS Position - Movement along East/West axis




(a) Latitude




(b) Longitude




(c) Altitude

Figure 7.9: Vehicle Global Position - Movement along East/West axis

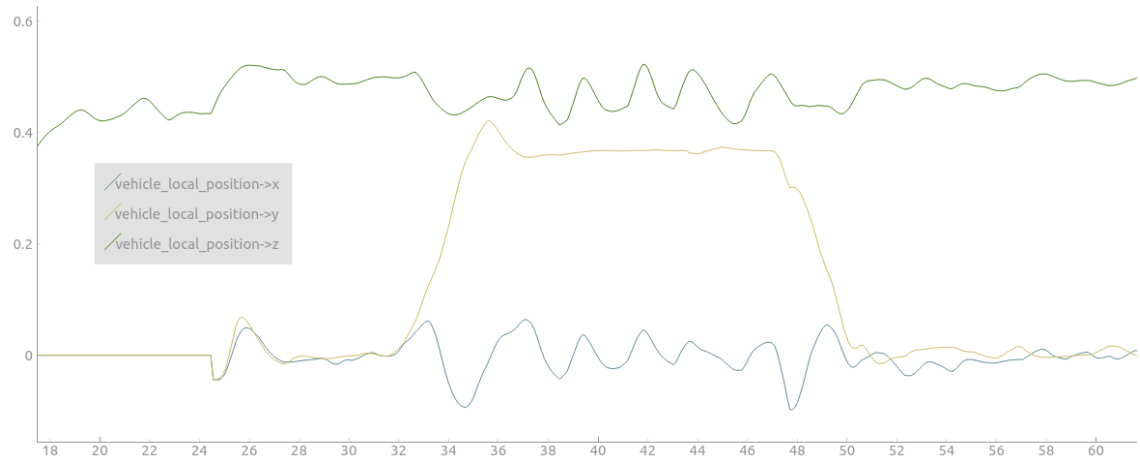



Figure 7.10: Vehicle Local Position - Movement along East/West axis

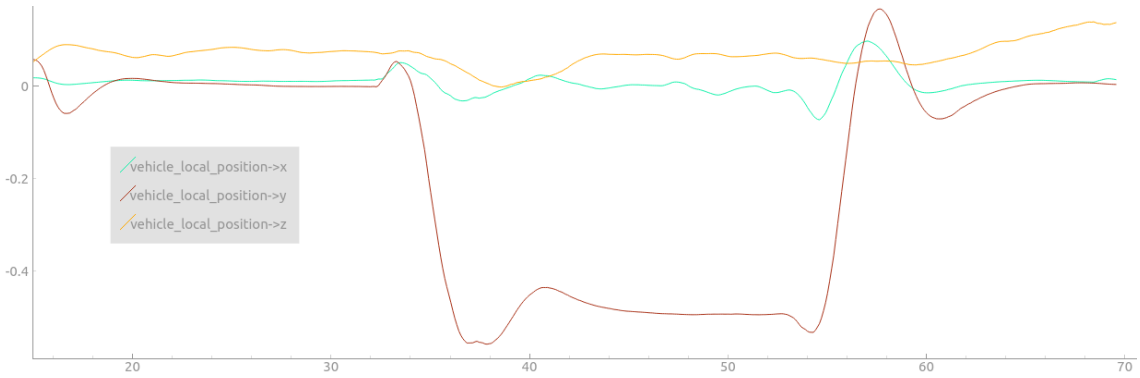
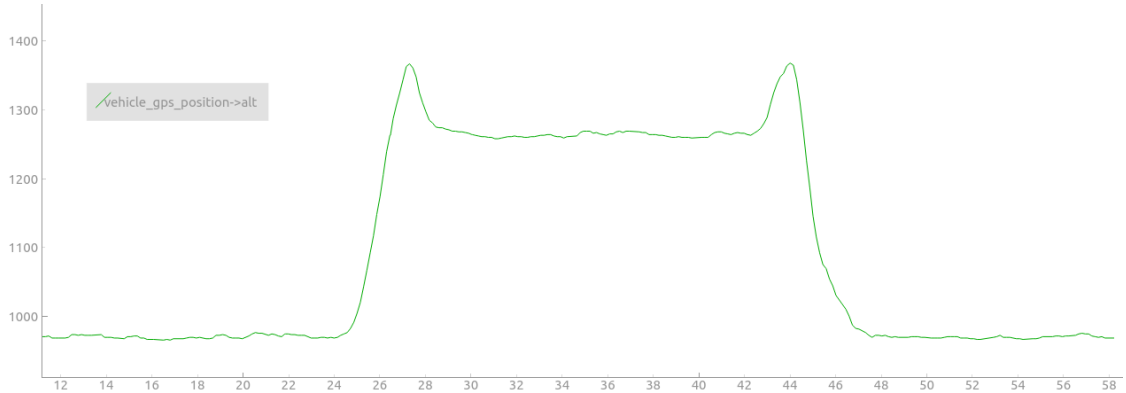
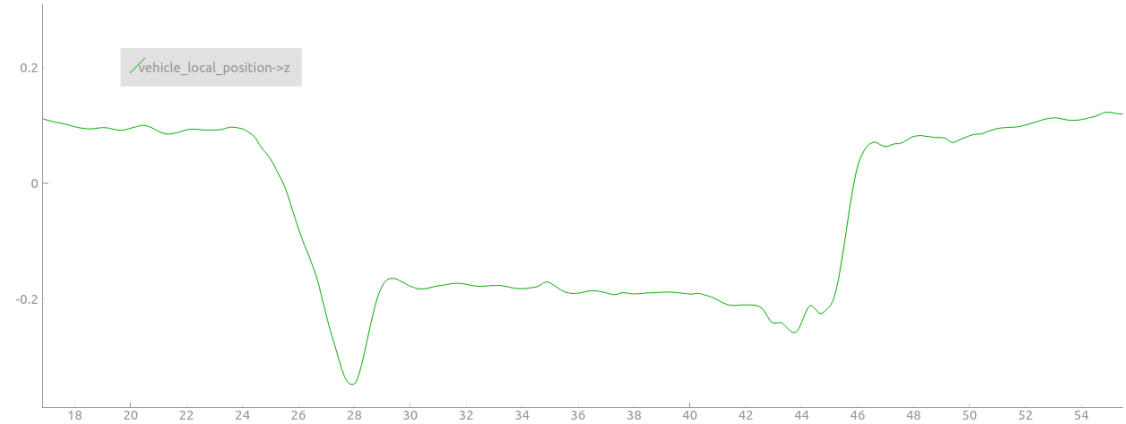



Figure 7.11: Vehicle Local Position - Movement along East/West axis - *Issue fixed*




(a) Vehicle GPS Position - Altitude




(b) Vehicle Local Position - Altitude

Figure 7.12: Movement in altitude

With this, the simulation and data analysis of the integration is complete. We conclude that the gathered data is correct and the indoor GPS data is arriving correctly to the flight controller.

To simplify the analysis of the chart from the logs, we created a simple script in python that reads the telemetry data directly from the drone in real time. The script consists in saving the first location of the position and calculate the distance to the next readings. First, we translate the coordinates of the position (lat,lon) to (x,y) with the equations (6.1) and (6.2), mentioned before. With references from the first reading, we determine the hypotenuse to the next readings and the difference of altitude. A small test of movement of the drone 50cm away from the first point was done, and the results were saved to a text file:

```
Time: 17:44:34.347
    Diference: 1 cm
    Altitude: 0 cm
Time: 17:44:34.555
    Diference: 1 cm
    Altitude: 0 cm
(...)
Time: 17:44:39.626
    Diference: 49 cm
    Altitude: 1 cm
Time: 17:44:39.930
    Diference: 53 cm
    Altitude: 0 cm
Time: 17:44:40.220
    Diference: 51 cm
    Altitude: 0 cm
(...)
Time: 17:44:52.078
    Diference: 0 cm
    Altitude: 0 cm
Time: 17:44:52.385
    Diference: 0 cm
    Altitude: -0 cm
```

This script allowed for a better understanding and acceptance of the gathering data when moving the drone. Also, this illustrates a proper communication done with the drone, as a consequence to Chapter 6, where we describe the drone control architecture.

Having done the simulations, it is time to test the real flight, witch is reported in the next section.

## 7.4 Test and Validation

With the localization developed, the remaining task is to perform flying tests with the real drone.

When arming the drone, that consists in start the motors with low speed, we noticed small disruptions of the measurements in the *Marvelmind* dashboard. The noise caused by the motors of the drone interfers with the mobile beacon sensors that were placed above the drone. The solution to this is to increase the altitude of the beacon, higher than the motors plane (Fig. 7.13).

Figure 7.13: Final disposal of the drone parts for final tests

Also, a control radio was acquired to allow commands like takeoff, land or even "kill", to instantly stop the motors.

After some unsuccessful takeoffs and crashes with the drone, we observed that the radio communication of the beacons had low update, caused by using the same base wave frequency of the telemetry radio (433MHz). This induced wrong and bad measurements of the position of the drone. The solution was to change the base frequency to 315MHz, the far distance allowed by the *Marvelmind* system. For window of averaging, the value remained the same, as explained in Section 4.2.

Subsequently, successful takeoffs of the drone were achieved. But a small drift occurred in the position of the drone when it is holding the position. Some multicopters require different tuning gains for optimal flight. Since the parameters that are tuned by default in PX4 are made to fly drones outdoor, we had to configure these parameters to suit our indoor environment. For that we used the multicopter tuning position manual [33].

The copter tended to drift in its diagonal, to the front and right, in positive roll and pitch definition.


Figure 7.14: Roll, Pitch and Yaw representation

The tuned parameters are disposed in the table 7.2. We started with the PID tuning of rate controllers - roll and pitch. The yaw did not need any improvement. The integral gain is set to compensate static thrust difference or gravity center offset, that occurred in our assembly and cable management.

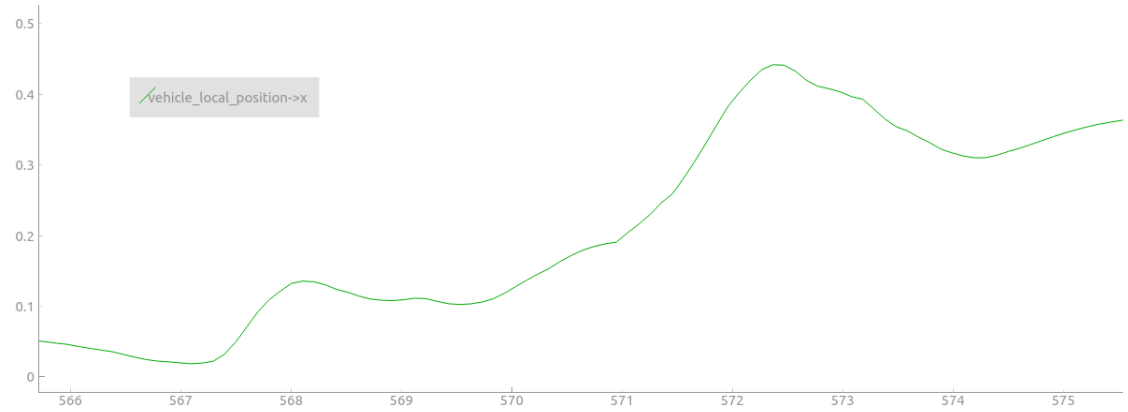
| Parameter | Value |
|---|---|
| MC_ROLLRATE_P | 0,205 |
| MC_ROLLRATE_I | 0,150 |
| MC_ROLLRATE_D | 0,005 |
| MC_PITCHRATE_P | 0,230 |
| MC_PITCHRATE_I | 0,170 |
| MC_PITCHRATE_D | 0,005 |

Table 7.2: Important PID parameters configuration

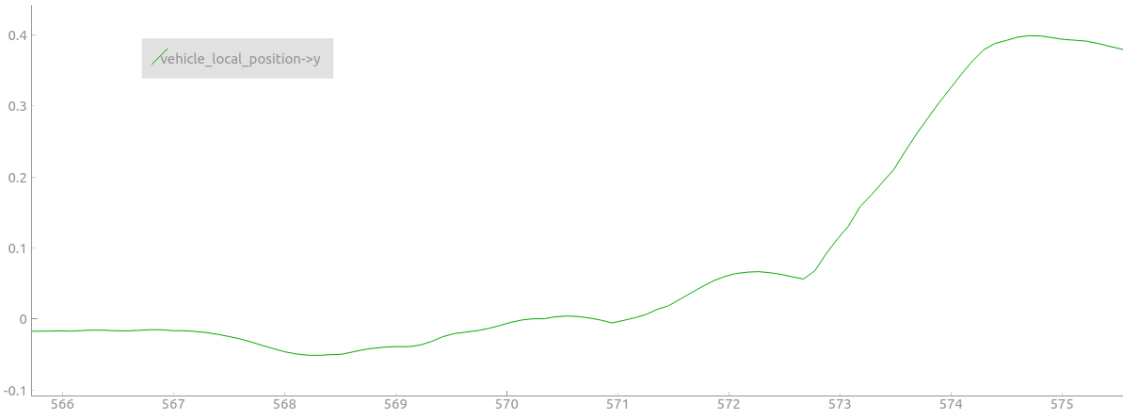
This tuning allowed a better autonomous takeoff and landing of the drone, even with the drift in altitude and lower position drift. A takeoff and landing test were done in order to test the flight of the drone. The programmed altitude for the takeoff is 50 centimeters. The results are in figure 7.15

As we can see, in the X axis (Fig. 7.15a) the error between the takeoff spot and the landing point is 30cm. On the other hand, the error in Y axis (Fig. 7.15b) between these two instances is 40cm. In the Z axis (Fig. 7.15c) we can also see that the altitude also exceeds the desired one, reaching 1 meter of altitude. The error is 50 cm in altitude.

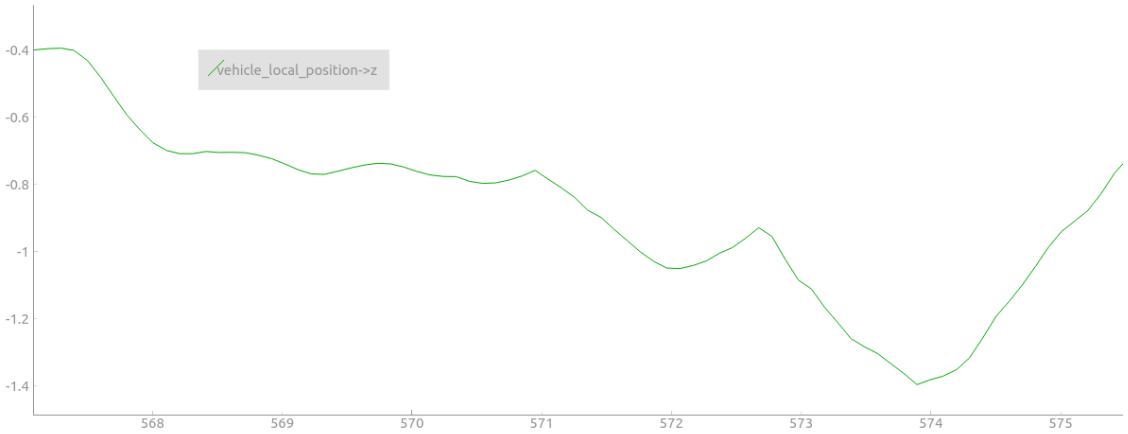
These errors could not be considered as acceptable errors, so we continued the tuning. Flying between waypoints could not be achieved with the default multicopter position controller parameters. This parameters affected the value of a desired setpoint (as contrasted with those parameters that affect how well the vehicle tracks the setpoint). So, we changed this parameters. These are displayed in the table 7.3. More information about the configured parameters can be found in [33].




(a) Local X




(b) Local Y




(c) Local Z

Figure 7.15: Vehicle Local Position - Takeoff and landing test (50cm altitude)

| Parameter | Value |
|---|---|
| MPC_TKO_RAMP_T | 0,375 s |
| MPC_XY_CRUISE | 3 m/s |
| MPC_XY_P | 1,10 |
| MPC_XY_VEL_P | 0,1 |
| MPC_XY_VEL_I | 0,021 |
| MPC_XY_VEL_P | 0,011 |

Table 7.3: Multicopter position controller parameters

The final flights were not perfect. The position error exceeded the requirements. However, we reached better results. Another tests were done and one result can be seen in figure 7.16. This demonstrates a takeoff and landing test result with the altitude programmed for 50 centimeters again.

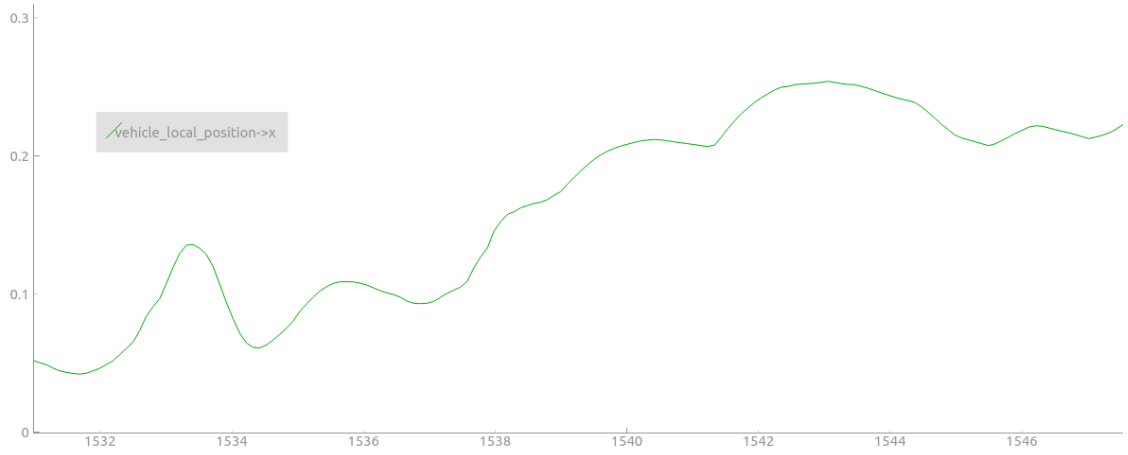
As we can see in figure 7.16c, the altitude was achieved with only approximately 5cm of error. In the X axis, the final error is approximately 12 centimeters and finally, in the Y axis, the error is 15 centimeters.

The results can validate the system performance, despite of the present errors.

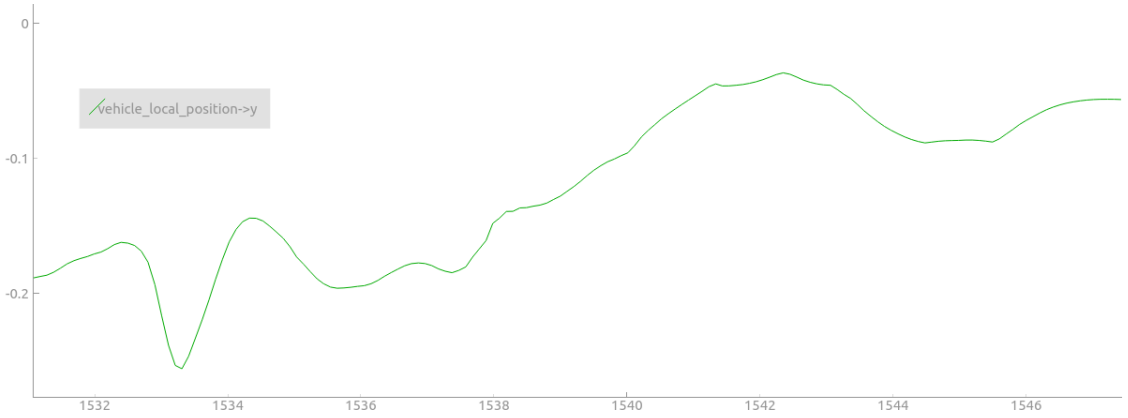
## 7.5 Discussion

The integration of the two parts can be considered successful. In Section 7.3, we can conclude that the location update and the measurements are in agreement with the reality and the position is adequately acquired by the drone. We have small error when doing this task, but this does not affect the performance of the integrated system.

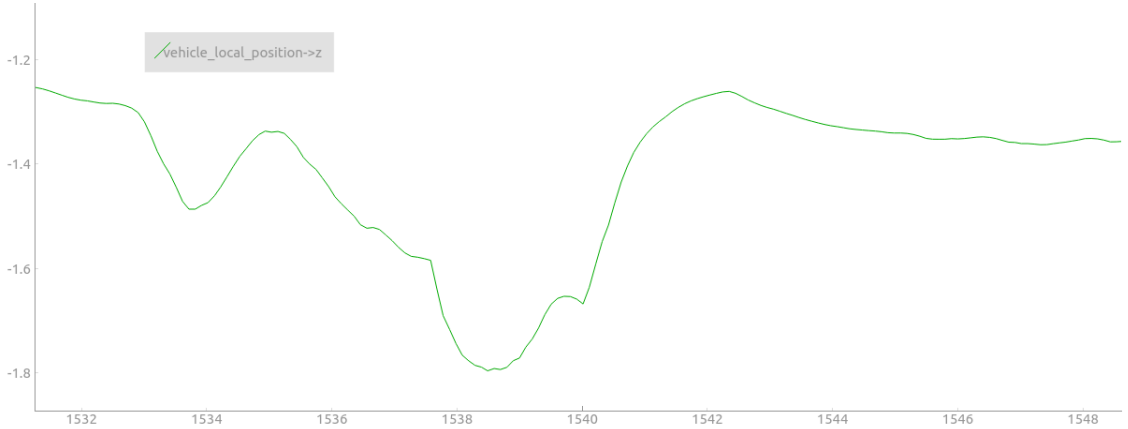
However, the flight performance illustrated in Section 7.4 is not accurate as the desired one. The drone is capable of correctly perform the actions intended to, but can not move well between waypoints. The drone can navigate throughout the room, but the error of the execution of the movements appear as the main reason of wrong movements of the drone.




(a) Local X




(b) Local Y




(c) Local Z

Figure 7.16: Vehicle Local Position - Takeoff and landing test (50cm altitude) - Final result

# Chapter 8

# Conclusions and Future Work

## 8.1 Conclusions

In this dissertation we addressed the integration of an indoor tracking solution for a drone. Indoor tracking is still an open subject since it is in an early state of development. However, it is with this recent expansion and improvement in indoor tracking technologies that we can now associate indoor localization with drones. The errors are lower and the results are better than the ones obtained using earlier technologies and procedures.

We introduced the *Marvelmind* system and its capabilities as a single localization system. We did some simulations with the mobile beacon itself. We conclude that the results using UWB beacons were really accurate, as announced by the company (+/-2cm).

Then, we described the drone solution for this problem. Also, we developed the drone control that is established by using Dronecode SDK. It is used to communicate directly to the drone and to send missions over Mavlink. Likewise, we refined a path planning algorithm that is capable of satisfying the customer requirements for a commercial product. The best solution to find the best path between points and corridors inside a warehouse can be reached with an algorithm that is based in A* search algorithm. The developed strategy was shown to be an effective way to handle missions in an indoor environment.

Finally, we proposed a new solution witch integrates the *Marvelmind* localization solution and the PX4 flight stack. This is an innovative approach that is not reported in literature and its integration is not a simple "plug & play" as it appeared to be at the beginning of the project. The main interface to communicate the beacon and the flight controller is accomplish by NMEA sentences, that found to be successfully arranged. We reported some preliminary results of the integration that were also satisfactory. Lastly, we did a flight performances analysis in order to evaluate the final integration. A maximum error of 15 centimeters is achieved using *Marvelmind* coordination, concluding that it is possible to fly drones in an indoor environment using the indoor tracking device proposed.

## 8.2 Future Work

There is a lot work to do regarding indoor tracking solutions for drones. These are not perfect and require a high controlled environment in order to get precise and accurate measurements. Also, the developed flight stack present in PX4 is not prepared to manage indoor flying using beacons. This is the entire reason that the flights were not accomplished with the desired accuracy and it requires further improvements before it can be used as a commercial product.

The error within the pose and position estimation of the drone also needs to be improved. More tests and tuning are required in exchange for correct position and holding of the drone in air. This could be also reached with more beacons placed in the room, as we could get more stable measurements.

Since this thesis was done in collaboration with a company, the main work would be to integrate with a real system in their projects. So, developing an interface can also be beneficial to their plans, since it has commands and simple buttons that help the final customer to properly work with the drone.

An ambitious project like this one involves many different topics and there will be always improvements to be made. This technology is still far from being completely dominated, and there is a window of opportunity in the development of new controllers and improvement of the existing methodologies. The final objective is to build real prototype drones and test them, firstly as a single pickup and place cargo drone system, and then as a warehouse management using drones in different altitudes.

# Appendix A

# Market Survey of Indoor Tracking Technologies

The market survey related to indoor tracking solutions is illustrated in this appendix.

| NAME / SPECS | Price [€] | Weight [g] | Accuracy [cm] | Range [m] | Battery time | Dimension [mm] | Update Rate [ms] | Notes |
|---|---|---|---|---|---|---|---|---|
| **MARVELMIND Robotics** | 405 | 90 | 2 | 30 | 72 h - beacon<br>12 h - mobile | 55x55x33 | 63 | Good option - allows IMU fusion |
| **Pozyx Accurate positioning** | 599 | 12 | 10 | 30 | ?? | 60x53 | 7 | Good option |
| **ACCUWARE DragonFly** | 7950 | Camera weight | 10-15 | 10 | ?? | Camera + PC | 33 | Good option |
| **UWINLOCK - indor location system** | ?? | ?? | Wireless - not good | 30 | battery less | ?? | 1000 | Lack of information |
| **BLINKSIGHT - accurate rlts indoor tracking** | ?? | ?? | 10 | 30 | 3 years | ?? | 1 | Lack of information |
| **TRACKTIO uRLTS** | ?? | ?? | 10 | 20 | 3 years | ?? | ?? | Discarded - 2D solution |
| **INFSOFT with UWB** | ?? | 112 | 30 | ?? | 1 year | 110x36x36 | ?? | Do not meet requirements (on site services) |
| **QUUPA inteligent locating system** | ?? | ?? | 50 | ?? | (big capacity) | 44x31x8 | 10 | Discarded - low accuracy |
| **NFER RTLS indoor location** | 3300 | ?? | 40 | ?? | ?? | (big) | 1000 | Discarded - low accuracy and big dimension |

# Appendix B

# Market Survey of Drones

The market survey related to drone solutions is detailed in this appendix.

| NAME / SPECS | Price [€] | Flight Time [min] | Velocity [m/s] | Payload [g] | Dimension [mm] | Programable SDK | Notes |
|---|---|---|---|---|---|---|---|
| TAROT 650 | 500 | 15 | 8 | 1500 | 658x658 | YES | Final price with shipment and VAT inc. |
| VulcanUAV Mini 8 | ?? | 15 | 22 | 1800 | 720 | YES | Waiting for more information |
| YUNEEC Typhoon H | 705 | 25 | 19 | 1950 | 520x457x310 | NO | Waiting for more information |
| YUNEEC H520 | 1765 | 28 | 17 | 1600 | 520x457x310 | YES | Waiting for more information |
| ACECORE Zoe | ?? | 40 | ?? | 6500 | 700x700x495 | ?? | Waiting for more information |
| DRONEVOLT Hercules 5 | ?? | ?? | ?? | 5000 | 700x700 | YES | Discarded - too expensive |
| DRONEVOLT Hercules 2 | ?? | 30 | 25 | 1000 | 300x300 | YES | Discarded - too expensive |
| SplashDrone 3 | 1060 | 20 | 20 | 1000 | 450 | YES | Discarded - small |
| DJI Matrice 100 | 3599 | 40 | 22 | 3600 | 650 | YES | Discarded - too expensive |
| DJI Inspire 2 | 3399 | 23 | 26 | 4250 | 605 | YES | Discarded - too expensive |
| PULSEAERO Vapor15 | ?? | 45 | ?? | 1100 | ?? | ?? | Discarded - helicopter |
| INTEL Aero Ready to Fly | 1040 | 20 | 15 | 1900 | 360x222x230 | YES | Discarded - small |
| STEADIDRONE FLARE | 1760 | 20 | 20 | 800 | 440X645X180 | ?? | Does not meet the requirements |
| STEADIDRONE MAVRIK | ?? | 25 | 18 | 2000 | 500x775x205 | ?? | Does not meet the requirements |
| PARROT Anafi | 699 | 25 | 15 | ? | 244x67x65 | NO | Small |
| HUBSAN H109S X4 Pro Standard | 448 | 21 | ?? | ?? | (normal) | NO | Fragile |
| HOLY STONE HS700 | 290 | 20 | ?? | ?? | 220x220x155 | NO | Fragile |
| HEXO + | 880 | 15 | ?? | ?? | 435x410x103 | NO | No longer produced |
| AIRBORNE DRONES Falcon x4 | 20000 | 50 | ?? | 500 | 450x530x100 | YES | Discarded - prototype |
| AIRBORNE DRONES Falcon x8 | 20000 | 32 | ?? | 2000 | 450x530x100 | YES | |
| ERLE-COPTER drone kit | 499 | 20 | ?? | 1000 | 360x360x95 | SIM | No longer produced |
| WALKERA Voyager 5 | 17500 | 41 | ?? | 3000 | (grande) | SIM | Expensive |
| PARROT Bebop 2 | 299 | 25 | ? | ? | 330x356x10 | NÃO | Too small |

# References


[1] Davide Dardari, Pau Closas, and Petar M. Djuric. Indoor Tracking: Theory, Methods, and Technologies. *IEEE Transactions on Vehicular Technology*, 64(4):1263–1278, April 2015.

[2] F. Carreira, J. Calado, C. Cardeira, and P. Oliveira. Navigation System for Mobile Robots Using PCA-Based Localization from Ceiling Depth Images: Experimental Validation. In *2018 13th APCA International Conference on Control and Soft Computing (CONTROLO)*, pages 159–164, June 2018.

[3] A. Chakrabarty, R. Morris, X. Bouyssounouse, and R. Hunt. Autonomous indoor object tracking with the Parrot AR.Drone. In *2016 International Conference on Unmanned Aircraft Systems (ICUAS)*, pages 25–30, June 2016.

[4] W. Jaworski, P. Wilk, P. Zborowski, W. Chmielowiec, A. Y. Lee, and A. Kumar. Real-time 3d indoor localization. In *2017 International Conference on Indoor Positioning and Indoor Navigation (IPIN)*, pages 1–8, September 2017.

[5] H. Plank, T. Egger, C. Steffan, C. Steger, G. Holweg, and N. Druml. High-performance indoor positioning and pose estimation with time-of-flight 3d imaging. In *2017 International Conference on Indoor Positioning and Indoor Navigation (IPIN)*, pages 1–8, September 2017.

[6] Luca Catarinucci, Riccardo Colella, Luca Mainetti, Luigi Patrono, Stefano Pieretti, Ilaria Sergi, and Luciano Tarricone. Smart RFID Antenna System for Indoor Tracking and Behavior Analysis of Small Animals in Colony Cages. *IEEE Sensors Journal*, 14(4):1198–1206, April 2014.

[7] M. J. Kuhn, M. R. Mahfouz, N. Rowe, E. Elkhouly, J. Turnmire, and A. E. Fathy. Ultra wideband 3-D tracking of multiple tags for indoor positioning in medical applications requiring millimeter accuracy. In *2012 IEEE Topical Conference on Biomedical Wireless Technologies, Networks, and Sensing Systems (BioWireleSS)*, pages 57–60, January 2012.

[8] F. O. Coelho, J. P. Carvalho, M. F. Pinto, and A. L. Marcato. EKF and Computer Vision for Mobile Robot Localization. In *2018 13th APCA International Conference on Control and Soft Computing CONTROLO*, pages 148–153, June 2018.

[9] Santosh Subedi and Jae-Young Pyun. Practical Fingerprinting Localization for Indoor Positioning System by Using Beacons, 2017.

[10] Yuxiang Wu, Xiaowei Liu, Weipeng Guan, Bangdong Chen, Xin Chen, and Canyu Xie. High-speed 3d indoor localization system based on visible light communication using differential evolution algorithm. *Optics Communications*, 424:177–189, October 2018.

[11] João Moutinho, Diamantino Freitas, and Rui Esteves Araújo. Indoor Global Localisation in Anchor-based Systems using Audio Signals. *The Journal of Navigation*, 69(5):1024–1040, September 2016.

[12] J. Moutinho, D. Freitas, and R. E. Araújo. Spread Spectrum Audio Indoor Localization. In *2015 IEEE 18th International Conference on Computational Science and Engineering*, pages 66–71, October 2015.

[13] Alejandro Correa, Marc Barcelo, Antoni Morell, and Jose Lopez Vicario. Enhanced Inertial-Aided Indoor Tracking System for Wireless Sensor Networks: A Review. *IEEE Sensors Journal*, 14(9):2921–2929, September 2014.

[14] Zafer Sahinoglu, Sinan Gezici, and Ismail Gvenc. *Ultra-wideband Positioning Systems: Theoretical Limits, Ranging Algorithms, and Protocols*. Cambridge University Press, New York, NY, USA, 2011.

[15] Ariel Gomez, Kai Shi, Crisanto Quintana, Grahame Faulkner, Benn C. Thomsen, and Dominic O'Brien. A 50 Gb/s Transparent Indoor Optical Wireless Communications Link With an Integrated Localization and Tracking System. *Journal of Lightwave Technology*, 34(10):2510–2517, May 2016.

[16] Ram Prasad Padhy, Sachin Verma, Shahzad Ahmad, Suman Kumar Choudhury, and Pankaj Kumar Sa. Deep Neural Network for Autonomous UAV Navigation in Indoor Corridor Environments. *Procedia Computer Science*, 133:643–650, January 2018.

[17] Drone Technology: Types, Payloads, Applications, Frequency Spectrum Issues and Future Developments.

[18] Rui Pinto, Filipe Neves, Filipe Neves Dos Santos, and Armando Sousa. Robot self-localization based on sensor fusion of gps and ibeacons measurements. In *Conference Paper*, February 2016.

[19] Marvelmind Manual. URL: https://marvelmind.com/pics/marvelmind_navigation_system_manual.pdf.

[20] Marvelmind Beacons Placement. URL: https://marvelmind.com/pics/indoor_navigation_system_ENG_copter_help_manual.pdf.

[21] Beacon Interfaces Manual. URL: https://marvelmind.com/pics/marvelmind_beacon_interfaces.pdf.

[22] Pixhawk Wiring. URL: https://docs.px4.io/en/assembly/quick_start_pixhawk.html.

[23] Dronecode SDK. URL: https://sdk.dronecode.org/en/.

[24] Marvelmind and Ardupilot Integration. URL: https://marvelmind.com/pics/PixHawk_Marvelmind_Integration_Manual.pdf.

[25] QGround Control. URL: http://qgroundcontrol.com/.

[26] A* Algorithm. URL: https://en.wikipedia.org/wiki/A*_search_algorithm.

[27] PX4 GPS Drivers. URL: https://github.com/PX4/GpsDrivers/tree/master/src.

[28] PX4 Development Guide. URL: https://dev.px4.io/v1.8.2/en/.

[29] PX4 Basic Configuration. URL: https://docs.px4.io/en/config/.

[30] Francois Caron, Emmanuel Duflos, Denis Pomorski, and Philippe Vanheeghe. GPS/IMU data fusion using multisensor Kalman filtering: introduction of contextual aspects. *Information Fusion*, 7(2):221–230, June 2006.

[31] PX4 Estimating and Control Library. URL: https://github.com/PX4/ecl/tree/0f49eb34a08e499a26d749ed981429047f2100ff/EKF/documentation.

[32] Simon D. Levy. The Extended Kalman Filter: An interactive tutorial for non-experts. URL: https://simondlevy.academic.wlu.edu/kalman-tutorial/.

[33] PX4 Multicopter Tuning. URL: https://docs.px4.io/en/config_mc/.