MASTER'S THESIS

Study of a UAV with autonomous LIDAR navigation

Master Degree in Aeronautics

MUEA

Tutors: Bernardo Morcego & Ramon Pérez Magrané

Delivery day: 22 of June 2021

MARCEL MARÍN DE YZAGUIRRE

ESEIAAT
UPC

Abstract

The recent growth of Unmanned Aerial Vehicles has brought them to a wide variety of industries, from construction to film production or emergency services. These vehicles still depend on some part of human intervention and can only operate comfortably in very wide and open spaces. This is mainly due to the limited surrounding awareness that the UAVs have as they are equipped with very simple sensing equipment. In this research work, this issue will be tackled.

This project focuses on the use of a solid-state LIDAR sensor module to perform obstacle avoidance. The integration is done in a Hummingbird multirotor platform where a path following algorithm has already been tested. The objective will be to integrate the system into the existing platform and validated it in real flight conditions. The Hummingbird UAV at the end of the project will be capable of autonomously navigating while at the same time being capable of performing obstacle avoidance manoeuvres around non-planned and static objects.

Resum

El recent creixement dels vehicles aeris no tripulats els ha portat a una àmplia varietat d'indústries, des de la construcció fins a la producció de pel·lícules o serveis d'emergència. Aquests vehicles encara depenen en part de la intervenció humana i només poden funcionar còmodament en espais oberts i molt amplis. Això es deu principalment a la percepció limitada d'entorn que tenen els UAV, ja que estan equipats amb equips de detecció molt senzills. En aquest treball de recerca es tractarà aquest tema.

Aquest projecte es centra en la utilització d'un mòdul sensor LIDAR d'estat sòlid per evitar obstacles. La integració es fa en una plataforma multirotor Hummingbird on ja s'ha provat un algoritme de seguiment de ruta. L'objectiu serà integrar el sistema a la plataforma existent i validar-lo en condicions reals de vol. El UAV Hummingbird al final del projecte serà capaç de navegar de forma autònoma alhora que realitzar maniobres per evitar obstacles amb objectes estàtics i no planificats.

Resumen

El reciente crecimiento de los vehículos aéreos no tripulados los ha llevado a una amplia variedad de industrias, desde la construcción hasta la producción de películas o servicios de emergencia. Estos vehículos todavía dependen en parte de la intervención humana y solo pueden operar cómodamente en espacios muy amplios y abiertos. Esto se debe principalmente a la limitada percepción de los alrededores que tienen los UAV, ya que están equipados con equipos de detección muy simples. En este trabajo de investigación se abordará este tema.

Este proyecto se centra en el uso de un módulo sensor LIDAR de estado sólido para evitar obstáculos. La integración se realiza en una plataforma multirotor Hummingbird donde ya se ha probado un algoritmo de seguimiento de ruta. El objetivo será integrar el sistema en la plataforma existente y validarlo en condiciones de vuelo reales. El UAV Hummingbird al final del proyecto será capaz de navegar de forma autónoma y, al mismo tiempo, podrá realizar maniobras de evasión de obstáculos alrededor de objetos no planificados y estáticos.

Contents

List of Figures

List of Tables

Acronyms

DDPG	Deep Deterministic Policy Gradient.
DDPG-OA	Deep Deterministic Policy Gradient with Obstacle Avoidance.
FCU	Flight Controller Unit.
GNCS	Guidance, Navigation and Control System.
GNSS	Global Navigation Satellite System.
IMU	Inertial Measure Unit.
LIDAR	Laser Imaging Detection and Ranging.
NLGL	NonLinear Guidance Law.
OA	Obstacle Avoidance.
PF	Path Following.
ROS	Robot Operating System.
UAS	Unmanned Aerial System.
UAV	Unmanned Aerial Vehicle.
VTP	Virtual Target Point.

1. Introduction

In this thesis, an autonomous navigation system for UAV will be brought from simulation to real-life tests. The autonomous navigation will be GPS and LIDAR based. The navigation software used will be based on separated guidance and control structure with agents based on Q-learning that includes obstacle avoidance.

1.1. Objectives

The main objectives of this Master's thesis are:

- Assemble and program the LIDAR sensors for object detection
- Validate the correct functioning of the system.
- Implement the algorithm in ROS on the Hummingbird UAV platform
- Perform flight test with all the system functioning.

1.2. Scope

The scope of this project is to implement an autonomous navigation system using LIDAR in a quadcopter. The UAV will be programmed using ROS and will be based of Rubs's PhD Thesis[11]. The project will focus on creating a functional UAV that could flight autonomously using LIDAR as a piece of additional sensing equipment (apart from the standard navigation instruments). The quadcopter will be capable of performing a preprogrammed path while avoiding unexpected static objects.

1.3. Requirements

At the completion of this project, the Hummingbird system equipped with the LIDAR will be able to fly while it:

- Follows a pre-defined path.
- Detects the readings from the LIDAR installed onboard.
- Processes the measurements from the LIDAR.
- Reacts and avoid objects according to the sensing equipment.
- Performs autonomous flight object avoidance.

1.4. Background and Justifications

Autonomous navigation for UAVs has been on the rise for several years as technology and components got more complex, capable and available. Nowadays a standard quadcopter with the capability to perform autonomous flights using GPS, Inertial Measure Unit (IMU) and the inboard computer can be easily and cheaply assembled. The codes for basic flight capabilities are also wildly available as open software such as Ardupilot runs thousands of drones around the world.
The next frontier for autonomous aerial vehicles is to incorporate newer technologies and sensors, as those become more available, to achieve higher degrees of autonomous navigation. This new level of autonomy will allow the multirotor to fly on object dense environments such as forests, warehouses, streets, etc. With these new capabilities, of the UAVs will expand into newer possibilities where more complex tasks could be done in more complex environments. The UAVs will grow to become an essential tool in the industrialization 4.0.

1.5. State of the Art

Currently, there are few market-ready UAV solutions that use LIDAR technology. Those are used for various applications from navigation to terrain mapping. LIDAR (or similar sensors) are used in a very simplistic way where they avoid crashing the UAV when the pilot is on control. More advanced systems are capable of following subjects while performing some kind of obstacle avoidance (instead of just stopping in front of the object). This study will be dealing with state of the art technology based on research projects.

The current state of the art in guidance for consumer drones can be found in products such as Skydio 2 or DJI Ari 2S. The Skydio 2 uses an array of 6 wide-angle cameras to gain 360 degrees surrounding awareness. This drone is capable of following a target through dense and complex environments such as forests. The UAV does not have an autonomous path following functionality but is capable of maintaining a line of sight with the target despite the interference of objects. In this case, all the object avoidance is done using cameras and computer vision due to the small size factor of the drone [16]. The DJI drone has similar characteristics but the obstacle avoidance system it is simpler. This drone is not capable of surpassing the obstacle it just avoids crashing into it[3]. In both cases the technologies used are cameras. The main advantage of this approach is the reduction in the size of the hardware required. On the other hand, the precision and reliability of the system can't be compared with more complex systems such as LIDARs, also this approach is not capable of measuring distances, something essential for complex navigation. The camera implementations are a good starting point for gaining general surrounding awareness and performing some degree of obstacle avoidance but in the actual state of technology, LIDAR equipment presents more accurate measurements as well as precise range.

The other starting point for this study is the control algorithm for the UAV. During the past decade, the autonomous navigation industry has made a big leap in technology. Nowadays the more advanced navigation algorithms are based on Reinforcement Learning Approach. This Master thesis will depart from the algorithms and simulations developed in a PhD thesis devoted to Guidance, navigation and Control of Multirotors [11]. The objective of this technology is to improve the current methods of path following where the trajectories were imprecise and inefficient. This new approach implements the Deep Deterministic Policy Gradient algorithm and is trained in a realistic multirotor simulator. The algorithm is able to accurately follow a path while adapting the vehicle's velocity depending on the path's shape.

2. General UAV Considerations

In this section, the necessities and possible configurations of the UAV will be briefly explored. The quadcopter used is a commercial product and has already been assembled with a specific set of characteristics. The main crucial steps will be regarding the installation of the LIDAR. The following table presents the general parameters of the Hummingbird.

Description:	**Value**
Dimensions:	54 x 54 x 5.5 cm
Propeller size:	8"
Propeller Type:	Flexible (default) or AscTec (optional)
Motors:	4 x 80W
Max. thrust:	20N
Max. payload:	200g
Max. total weight:	710g
Max. airspeed:	15m/s
Max. flight time:	20mins (without payload)
Battery:	2100mAh (LiPo)

Table 1: Hummingbird Characteristics

As mentioned all the UAV flight requirements are self-contained in the AscTec product. The main consideration will need to address the payload capacity of the system. The onboard LIDAR must not surpass the 200g of maximum payload. If other accessories such as propeller guards are instaled those need to be taken into consideration.

2.1. UAV system Necessities

The necessities are focused on obtaining a UAV capable of performing autonomous navigation and avoiding objects in test flights. The main necessities are listed below.

Hardware necessities

- Optimal LIDAR anchoring points
- Power and data connections
- Max payload and manoeuvrability equilibrium
- Long endurance flight
- Landing gear protection and some resistance to collision
- GPS connection

Software necessities

- Sensing and processing of the LedarTech LIDAR
- Post-processing of the data to be readable for the algorithm
- Ground and Onboard computer configuration with Ubuntu 16.04 and all the ROS packages

These necessities will be essential for the correct testing and development of the overall project.

2.2. Main Hummingbird Characteristics

In this section, the main characteristics of the Hummingbird platform will be briefly described[6]. The AscTec Hummingbird is a quadcopter equipped with the AscTec Autopilot. The frame is made of rigid carbon fibre-balsa wood sandwich material, being lightweight and strong. The AscTec Hummingbird PowerBoard is used to distribute power and communication lines to all motor controllers. It comprises a switching power regulator to generate a stable 6V supply for the AscTec AutoPilot board. The AutoPilot is the sensing and flight control unit of the Hummingbird. The UAV is also equipped with an AscTec 3D-MAG (compass magnetometer) and a GPS module. The Hummingbird works with a 3S LiPo battery (11.1V). Figure 1 shows the UAV in its plain form.

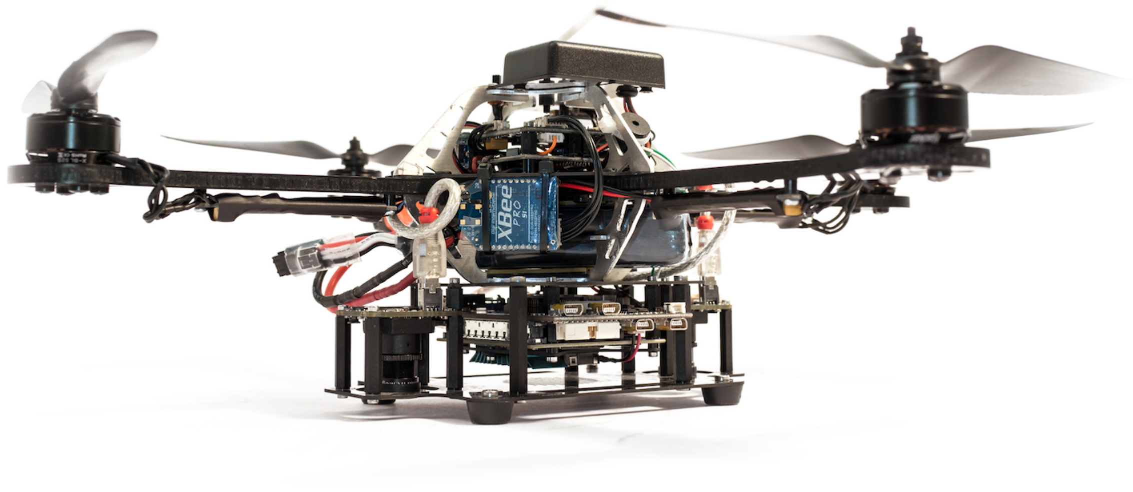


Figure 1: Hummingbird standard configuration

As it can be seen in figure 1, the drone has a special rig installed above and below that will allow a correct installation of the onboard computer, the Laser Imaging Detection and Ranging (LIDAR) as well as the battery.

2.3. Safety and protection considerations

During all the test navigation it is possible that the drone will experience some rough landings and sometimes even some collisions. For this reason, two protection strategies have been implemented. The first one is a flexible landing gear. It will provide some damping during landing and avoid the drone to tip over. this landing gear is composed of 10 flexible fibre rods disposed of in a radial pattern with a foam balls a the tip.
The second protection mechanisms is a small cage where the drone and its propellers are guarded. This lightweight structure will provide some level of protection to the propellers in case of a frontal or lateral collision. This system will avoid braking propellers as well as the drone falling due to propeller interference.
It's important to mention that the onboard computer is also caged inside a carbon frame under the drone to protect its electronics.

2.4. LIDAR Considerations

In this section, the requirements from the LIDAR will be reviewed. Those will be essential for the correct functioning of the overall obstacle avoidance system.
The LIDAR needs to be connected to the inboard power source as well as the inboard computer. The power supply must be a DC 12V source which could come from a 12V supply BEC that must be equipped on the Hummingbird. The connection with the inboard computer will be accomplished via USB or SPI connections.

The other important configuration will be the attaching point. It must not disturb the stability of the UAV and not be obstructed by any object. The LIDAR also needs to have the correct inclination to achieve the optimum view angle of the floor and the objects in front.
Table 2 presents the basic LIDAR VU8 characteristics:

Description	**Value**
Weight.	107.6 g
Dimensions.	70.0 x 35.9 x 49.6 mm
Power supply.	12 V
Horizontal FOV.	48°
Vertical FOV.	3°
Range (Retro-reflector).	85 m
Range (white target).	19 m
Range (grey target).	13 m
Accuracy.	5 cm
Distance precision.	1 cm
Refresh rate.	100 Hz

Table 2: Characteristics of the Leddar VU8 Medium FOV

Figure 2 shows the sensing capabilities of the LIDAR and how they are structured. The LIDAR has 8 channels that go from left to right when the sensor is positioned vertically. Each channel can detect an independent distance measurement. An object crossing the beam of the module is detected and measured. It is qualified by its distance, segment position, and amplitude. The quantity of light reflected back to the module by the object generates the amplitude. The bigger the reflection, the higher the amplitude will be. The amplitude is expressed in counts. A count is the unit value of the used ADC in the receiver. The fractional of counts is caused by the accumulation to get more precision.

Field	Description
Segment (Seg)	Beam segment in which the object is detected.
Distance	Position of the detected object.
Amplitude	Quantity of light reflected by the object and measured by the module.
Flags	8-bit status (bit field)

Table 3: Raw detection table description [9]

The Flag parameter provides the status information that indicates the measurement type. If the bit position is zero (0) the measurement is declared invalid. If the bit position is one (1) the measurement is accepted as normal. If the bit position is three (3) the received signal is above the saturation level. Measurements are valid (VALID is set) but have lower accuracy and precision. It is recommended to consider decreasing the light source intensity.

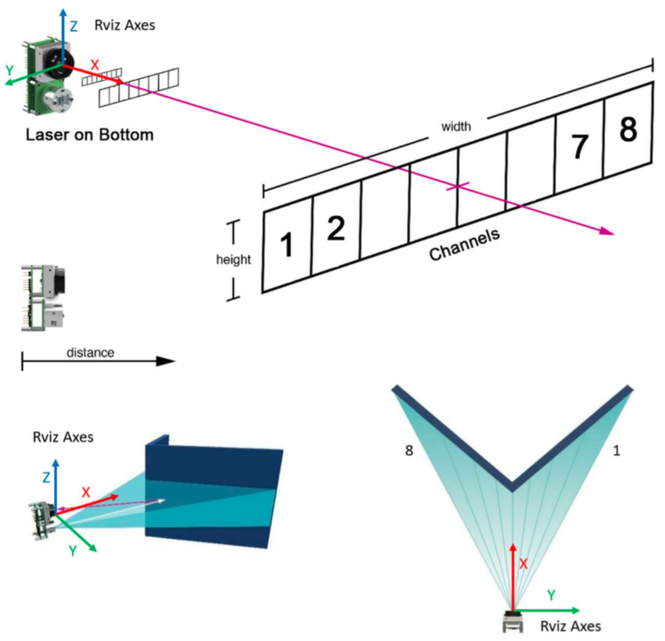


Figure 2: Rviz Leddar ROS orientation and Channels

Figure 3 presents an image of the sensor. As it can be seen it has an emitting part and a receiving sensor. It does not have any housing and all the electronics are exposed at the rear of the senor. The USB connection is located on the top of the system and the power connection at the back.

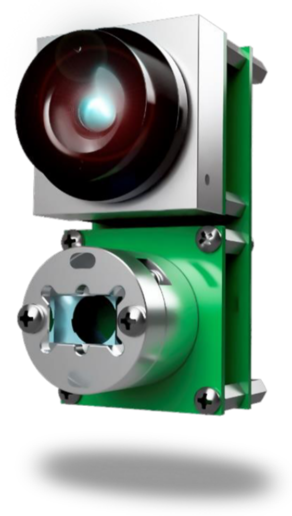


Figure 3: LIDAR LedarTech sensor

3. Software

In this section, the software used for this project and its progressive configuration will be described.

3.1. Ubuntu

All of this project will be based on ROS. This software has been developed with Linux in mind. For this project, Ubuntu 16.04 LTS will be used as the base operative system. All the simulation and programming will be done on this Operative System. A 2010 Sony Vaio will be restored and equipped with Ubuntu.

3.2. Robot Operating System - ROS

Robot Operating System (ROS) is an open-source robotics middleware suite. ROS is not an operating system but a software frameworks for robot software development, it provides services designed for a heterogeneous computer cluster such as hardware abstraction, low-level device control, implementation of commonly-used functionality, message-passing between processes, and package management.
The version used for this development will be the Kinetic Kame (last tenth ROS distribution release), published on May 23rd, 2016
Complementary ROS programs such as Gazebo will also be used as they will provide additional simulation tools.

3.2.1. ROS Structuring

In this section, the main file and peer-to-peer networked organization of ROS will be explained [10].
ROS Filesystem Level

- **Packages:** Packages are the main unit for organizing software in ROS. A package may contain ROS runtime processes (nodes), a ROS-dependent library, datasets, configuration files, or anything else that is usefully organized together. Packages are the most atomic build item and release item in ROS. Meaning that the most granular thing you can build and release is a package.
- **Repositories:** Are a collection of packages that share a common VCS system (version control system (VCS) tool, designed to make working with multiple repositories easier). Packages that share a VCS share the same version and can be released together using the catkin release automation tool bloom. Often these repositories will map to converted rosbuild Stacks. Repositories can also contain only one package.

Computation Graph level

- **Nodes:** Nodes are processes that perform computation. A robot control system usually comprises many nodes. As an example, one node controls a laser range-finder, one node controls the wheel motors, one node performs localization, one node performs path planning, and so on. A ROS node is written with the use of a ROS client library, such as roscpp or rospy.
- **Master:** The ROS Master provides name registration and lookup to the rest of the Computation Graph. Without the Master, nodes would not be able to find each other, exchange messages, or invoke services.
- **Parameter Server:** The Parameter Server allows data to be stored by key in a central location. It is currently part of the Master.
- **Messages:** Nodes communicate with each other by passing messages. A message is simply a data structure, comprising typed fields. Standard primitive types (integer, floating-point, boolean, etc.) are supported, as are arrays of primitive types. Messages can include arbitrarily nested structures and arrays.

- **Topics:** Messages are routed via a transport system with publish/subscribe semantics. A node sends out a message by publishing it to a given topic. The topic is a name that is used to identify the content of the message. A node that is interested in a certain kind of data will subscribe to the appropriate topic. There may be multiple concurrent publishers and subscribers for a single topic, and a single node may publish and/or subscribe to multiple topics. In general, publishers and subscribers are not aware of each others' existence. The idea is to decouple the production of information from its consumption.
- **Services:** The publish/subscribe model is a very flexible communication paradigm, but its many-to-many, one-way transport is not appropriate for request/reply interactions, which are often required in a distributed system. Request/reply is done via services, which are defined by a pair of message structures: one for the request and one for the reply.
- **Bags:** Bags are a format for saving and playing back ROS message data. Bags are an important mechanism for storing data, such as sensor data, that can be difficult to collect but is necessary for developing and testing algorithms.

3.2.2. Gazebo

"Robot simulation is an essential tool in every roboticist's toolbox. A well-designed simulator makes it possible to rapidly test algorithms, design robots, perform regression testing, and train AI system using realistic scenarios. Gazebo offers the ability to accurately and efficiently simulate populations of robots in complex indoor and outdoor environments. At your fingertips is a robust physics engine, high-quality graphics, and convenient programmatic and graphical interfaces. Best of all, Gazebo is free with a vibrant community." [14]

3.2.3. RotorS

"RotorS is a Micro Aerial Vehicle (MAV) gazebo simulator developed by the Autonomous Systems Lab at ETH Zurich. It provides some multirotor models such as the AscTec Hummingbird, the AscTec Pelican, or the AscTec Firefly, but the simulator is not limited for the use with these multirotors. There are simulated sensors coming with the simulator such as an IMU, a generic odometry sensor, and the VI-Sensor, which can be mounted on the multirotor. This packages also contains some example controllers, basic worlds, a joystick interface, and example launch files." [1]

3.3. Ubuntu - ROS environment configuration

ROS for Linux works very differently from the usual Windows/Macintosh interface. During our academic progression through the Bachelor's Degree and Master's different programming methods such as Matlab, C++, and Python were learned in some form or another. All of them were performed in a soft programming environment. In this case, Linux as will be seen requires a different compilation approach. This project will require a considerable learning curve regarding complex Linux environments. As described in the installation guide ROS works using the Linux terminal. This will be the principal interface for launching different nodes. ROS works with a workspace folder where all the different programs are stored and compiled. In order to run a ROS "program" (from now on called nodes) a new terminal window will be used. Normally to run a complex ROS program different set of nodes will need to be launched. Those nodes will perform different tasks which will vary from the background process to the main simulations.

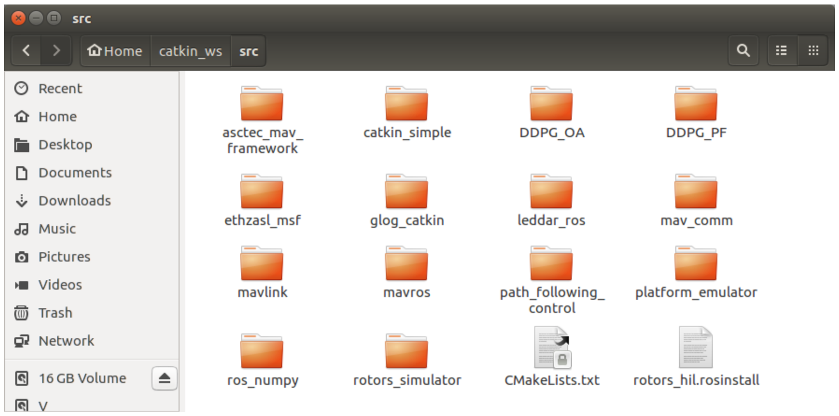
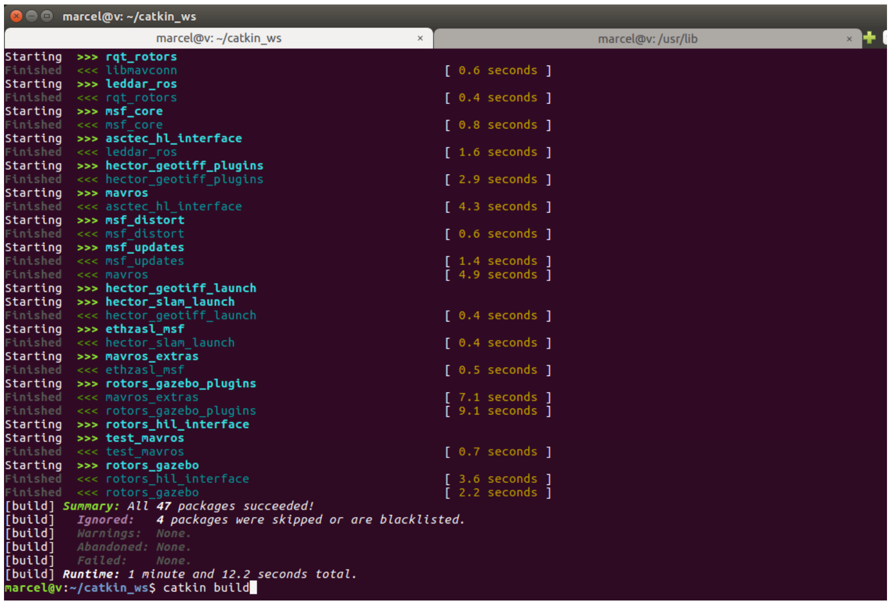


Figure 4: ROS Source folder inside the Catkin Workspace created for this project

3.3.1. Catkin Build

During the configuration of the source ROS folder, a compiler must be used in order to process all the packages with which ROS would have access to work wit. This compiling process can be done using different tools. In this thesis, the Catkin Build tool has been found the most reliable option. It presents a clear interface (see figure 5) and also easier to troubleshoot when compiling errors appear. Compiling all packages for the first time can be time-consuming as it can last up to 1h. But once this process has complied ROS will store all the data in the adjacent folders (logs, devel, build) and it will just compile again the packages if some modifications are done.



```
Starting  >>> rqt_rotors
Finished  <<< libmavconn                                     [ 0.6 seconds ]
Starting  >>> leddar_ros
Finished  <<< rqt_rotors                                     [ 0.4 seconds ]
Starting  >>> msf_core
Finished  <<< msf_core                                       [ 0.8 seconds ]
Starting  >>> asctec_hl_interface
Finished  <<< leddar_ros                                     [ 1.6 seconds ]
Starting  >>> hector_geotiff_plugins
Finished  <<< hector_geotiff_plugins                         [ 2.9 seconds ]
Starting  >>> mavros
Finished  <<< asctec_hl_interface                            [ 4.3 seconds ]
Starting  >>> msf_distort
Finished  <<< msf_distort                                    [ 0.6 seconds ]
Starting  >>> msf_updates
Finished  <<< msf_updates                                    [ 1.4 seconds ]
Finished  <<< mavros                                         [ 4.9 seconds ]
Starting  >>> hector_geotiff_launch
Starting  >>> hector_slam_launch
Finished  <<< hector_geotiff_launch                          [ 0.4 seconds ]
Starting  >>> ethzasl_msf
Finished  <<< hector_slam_launch                             [ 0.4 seconds ]
Starting  >>> mavros_extras
Finished  <<< ethzasl_msf                                    [ 0.5 seconds ]
Starting  >>> rotors_gazebo_plugins
Finished  <<< mavros_extras                                  [ 7.1 seconds ]
Finished  <<< rotors_gazebo_plugins                          [ 9.1 seconds ]
Starting  >>> rotors_hil_interface
Starting  >>> test_mavros
Finished  <<< test_mavros                                    [ 0.7 seconds ]
Starting  >>> rotors_gazebo
Finished  <<< rotors_hil_interface                           [ 3.6 seconds ]
Finished  <<< rotors_gazebo                                  [ 2.2 seconds ]
[build] Summary: All 47 packages succeeded!
[build]   Ignored:   4 packages were skipped or are blacklisted.
[build]   Warnings:  None.
[build]   Abandoned: None.
[build]   Failed:    None.
[build] Runtime: 1 minute and 12.2 seconds total.
marcel@v:~/catkin_ws$ catkin build
```

Figure 5: Catkin Build compilation process of the workspace

3.3.2. Preliminary configuration and Troubleshooting

Through this preliminary configuration (detailed in Annex A) different environment configuration has been tested. The first environment was configured with Ubuntu 20.04 LTS and ROS Noetic. Both versions are the latest release. The initial configuration of ROS Noetic and Ubuntu 20.04 was performed without any complications. The compatibility issues appeared during the compilation process of the ROS environment and the installed packages. After extensive exploration of the errors, no viable solution was found and a downgrade of the system was required. It's not uncommon that new releases have not yet solved some compatibility issues with older releases. For this reason, Ubuntu 16.04 LTS and ROS Kinetic were installed. With this set up all the initial configuration problems were solved[5].

3.4. LeddarTech

LeddarTech is a company leader in environmental sensing solutions for autonomous vehicles and advanced driver assistance systems. Their product Vu8 will be the Leddar used for this project. [9]
The LIDAR Leddar Vu8 is an affordable, versatile solid-state LiDAR sensor module that delivers exceptional detection and ranging performance in a small, robust package. Leddar Vu8 modules provide the ability to detect and track multiple objects simultaneously over eight distinct segments with superior lateral discrimination capabilities.

3.4.1. LeddarTech Windows OS Test

LeddarTech has a software exclusive for windows that after an easy installation allows the user to do quick tests. The software is called LeddarTM Configurator and can be download here: (`https://leddartech.com/resources/`). The installation is straight forward and after following the steps of the setup Wizard the program is installed. For connection to the LeedarVu module the following steps must be followed.

1. Connect the power cable to the module and to a power source, check that the polarity is correct.
2. Connect the USB cable to the module and to the computer
3. On the computer, double-click the LeddarTM Configurator icon.
4. In LeddarTM Configurator, click the connect button.
5. In the Connection dialogue box, in the Select a connection type list, select either LeddarVu SPI for a standard board or LeddarVu Serial for a USB, CAN and SERIAL and click connect.

After these steps, a simple visualization of the LIDAR field of view will be presented. Figure 6 shows the test visualization of the sensor in my room.

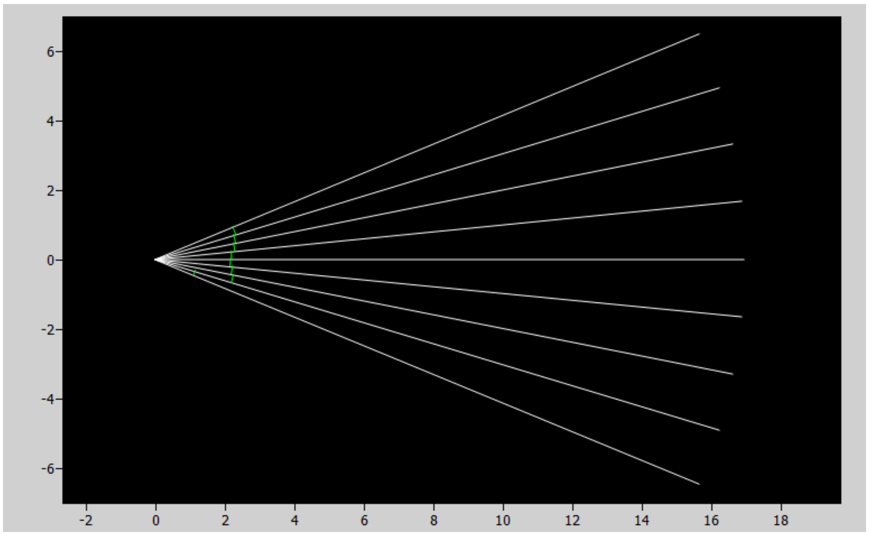


Figure 6: Leddar Test using the Windows configuration Software

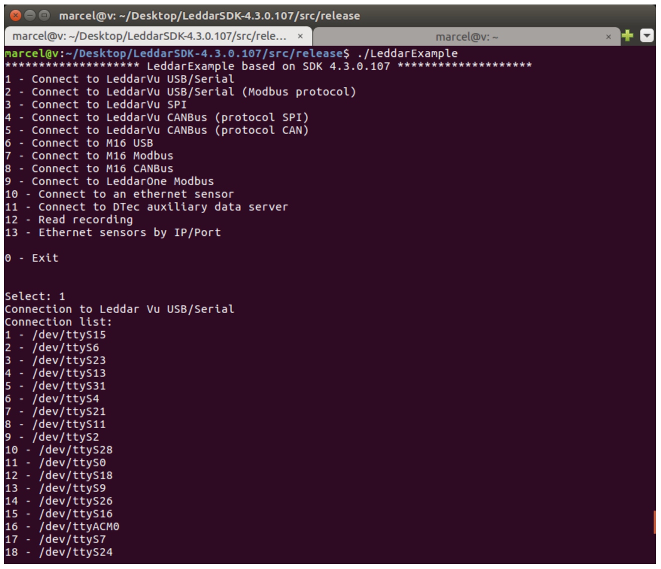
It can be seen that as the LIDAR is calibrated for measuring long distances the walls of my room appear very close to the sensor.
This is a simple test that allows knowing if the sensor is working properly. In case of need, additional configurations can be done with this software to easily tune or upgrade the sensor.

3.4.2. LeddarSDK

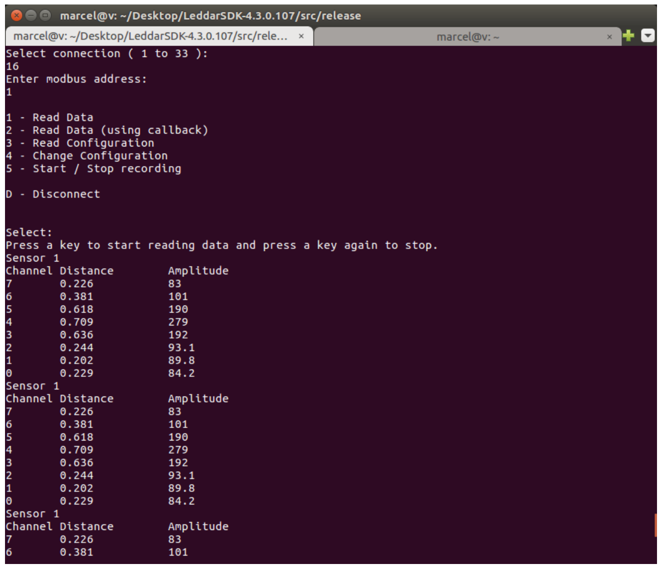
LeddarTech SDK is a C++ cross-platform SDK that allows you to connect to LeddarTech sensors. [8] This SDK will be essential for the cross-communication of the Leddar node and the actual hardware.

Leddar C++

The first test that can be done with this package in order to test the Leddar in Linux is to run the LeddarExample in C++. This code will provide the following interface (Figure 7). This code needs some use imputes for setting up. For the Leddar Vu8 with USB the connection mode will be 1. Then in the connection list, the port 16 */dev/ttyACM0* will be selected. The Modbus address is by default 1. If everything is properly set up the sensor can start to take a measurement. The measurements will read first the channel, then the distance from that channel and finally the amplitude. In this mode, the Flags won't be displayed.



```
marcel@v:~/Desktop/LeddarSDK-4.3.0.107/src/release$ ./LeddarExample
******************** LeddarExample based on SDK 4.3.0.107 ********************
1 - Connect to LeddarVu USB/Serial
2 - Connect to LeddarVu USB/Serial (Modbus protocol)
3 - Connect to LeddarVu SPI
4 - Connect to LeddarVu CANBus (protocol SPI)
5 - Connect to LeddarVu CANBus (protocol CAN)
6 - Connect to M16 USB
7 - Connect to M16 Modbus
8 - Connect to M16 CANBus
9 - Connect to LeddarOne Modbus
10 - Connect to an ethernet sensor
11 - Connect to DTec auxiliary data server
12 - Read recording
13 - Ethernet sensors by IP/Port

0 - Exit


Select: 1
Connection to Leddar Vu USB/Serial
Connection list:
1 - /dev/ttyS15
2 - /dev/ttyS6
3 - /dev/ttyS23
4 - /dev/ttyS13
5 - /dev/ttyS31
6 - /dev/ttyS4
7 - /dev/ttyS21
8 - /dev/ttyS11
9 - /dev/ttyS2
10 - /dev/ttyS28
11 - /dev/ttyS0
12 - /dev/ttyS18
13 - /dev/ttyS9
14 - /dev/ttyS26
15 - /dev/ttyS16
16 - /dev/ttyACM0
17 - /dev/ttyS7
18 - /dev/ttyS24
```

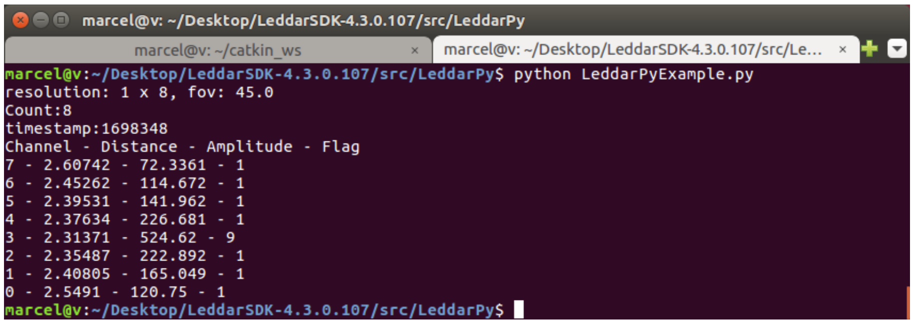


```
Select connection ( 1 to 33 ):
16
Enter modbus address:
1

1 - Read Data
2 - Read Data (using callback)
3 - Read Configuration
4 - Change Configuration
5 - Start / Stop recording

D - Disconnect


Select:
Press a key to start reading data and press a key again to stop.
Sensor 1
Channel Distance        Amplitude
7       0.226           83
6       0.381           101
5       0.618           190
4       0.709           279
3       0.636           192
2       0.244           93.1
1       0.202           89.8
0       0.229           84.2
Sensor 1
Channel Distance        Amplitude
7       0.226           83
6       0.381           101
5       0.618           190
4       0.709           279
3       0.636           192
2       0.244           93.1
1       0.202           89.8
0       0.229           84.2
Sensor 1
Channel Distance        Amplitude
7       0.226           83
6       0.381           101
```

Figure 7: Leddar Example tested with the C++ and the LIDAR Vu8

Leddar Python

The code designed for Python works in a similar fashion as the one with C++. This code needs to be modified in order to be able to read a specific sensor. The modifications required are described in the Annex. This code when executed just takes one measure (this can also be modified if necessary). The measure as in the other case presents the Chanel, Distance, Amplitude and the Flag. This code will automatically hide the channels that read erroneous information.



```
marcel@v: ~/Desktop/LeddarSDK-4.3.0.107/src/LeddarPy
marcel@v: ~/catkin_ws        marcel@v: ~/Desktop/LeddarSDK-4.3.0.107/src/Le...
marcel@v:~/Desktop/LeddarSDK-4.3.0.107/src/LeddarPy$ python LeddarPyExample.py
resolution: 1 x 8, fov: 45.0
Count:8
timestamp:1698348
Channel - Distance - Amplitude - Flag
7 - 2.60742 - 72.3361 - 1
6 - 2.45262 - 114.672 - 1
5 - 2.39531 - 141.962 - 1
4 - 2.37634 - 226.681 - 1
3 - 2.31371 - 524.62 - 9
2 - 2.35487 - 222.892 - 1
1 - 2.40805 - 165.049 - 1
0 - 2.5491 - 120.75 - 1
marcel@v:~/Desktop/LeddarSDK-4.3.0.107/src/LeddarPy$
```

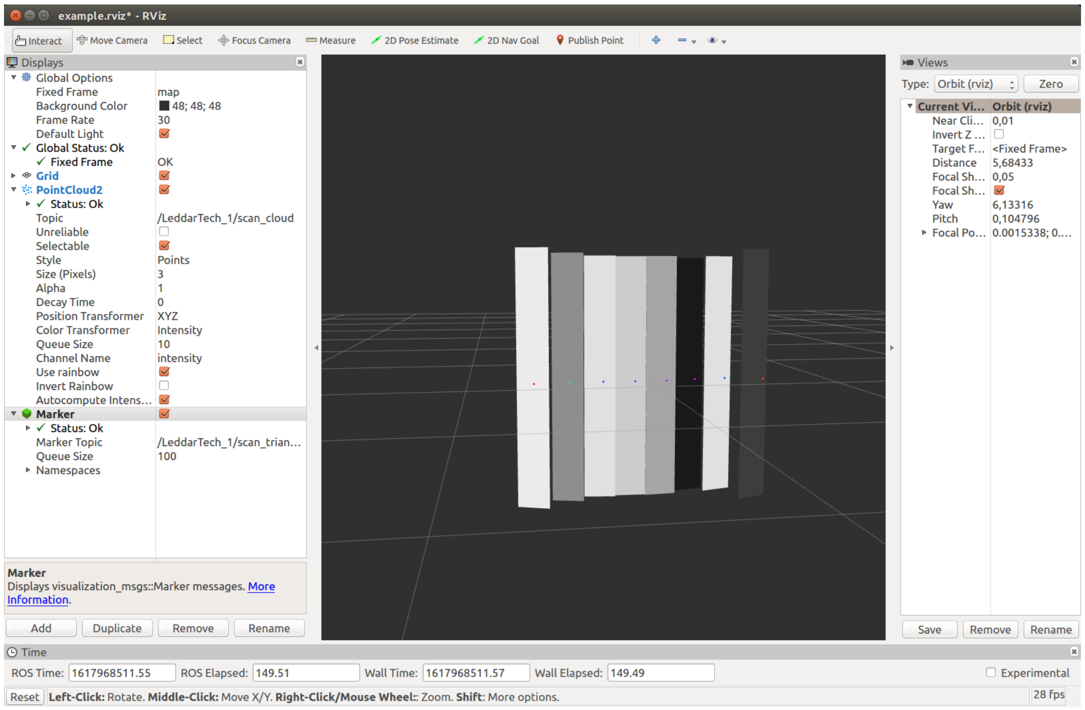
Figure 8: Leddar Test using the Python code

3.4.3. Leddar ROS Package

This ROS package configures and communicates with LeddarTech devices using their SDK. [13]. This package has to be installed carefully but if everything is done properly (see Annex) it allows Rviz (ROS - Ubuntu graphical interface) to display the measurements in real-time.
The ROS node can be launch using the following command.

```
roslaunch leddar_ros example.launch param1:=/dev/ttyACM0 device_type:=Vu8
```

The graphical interface will present a dot for each measured distance. The vertical bars also represent the surface that the sensor is measuring. The Leddar is located at the centre of the virtual room that will be used as a point of reference. The vertical bars will increase in size and move in relation to the sensed distance. The colour of the bars indicates the measured amplitude. The information obtained is the same as in the other codes but with a nicer visual representation.



```
Global Options
  Fixed Frame                map
  Background Color           48; 48; 48
  Frame Rate                 30
  Default Light              ✓
Global Status: Ok
  Fixed Frame                OK
Grid                         ✓
PointCloud2                  ✓
  Status: Ok
  Topic                      /LeddarTech_1/scan_cloud
  Unreliable
  Selectable                 ✓
  Style                      Points
  Size (Pixels)              3
  Alpha                      1
  Decay Time                 0
  Position Transformer       XYZ
  Color Transformer          Intensity
  Queue Size                 10
  Channel Name               intensity
  Use rainbow                ✓
  Invert Rainbow
  Autocompute Intens...      ✓
Marker                       ✓
  Status: Ok
  Marker Topic               /LeddarTech_1/scan_trian...
  Queue Size                 100
  Namespaces

Marker
Displays visualization_msgs::Marker messages. More Information.

Type: Orbit (rviz)   Zero
Current Vi...   Orbit (rviz)
  Near Cli...   0,01
  Invert Z ...
  Target F...   <Fixed Frame>
  Distance      5,68433
  Focal Sh...   0,05
  Focal Sh...   ✓
  Yaw           6,13316
  Pitch         0,104796
  Focal Po...   0.0015338; 0....

ROS Time: 1617968511.55   ROS Elapsed: 149.51   Wall Time: 1617968511.57   Wall Elapsed: 149.49
Left-Click: Rotate. Middle-Click: Move X/Y. Right-Click/Mouse Wheel:: Zoom. Shift: More options.   28 fps
```

Figure 9: Leddar Test using Leddar in ROS with Rviz

4. Guidance, Navigation and Control System: General Considerations

In this section, the Navigation, Guidance, Navigation and Control System (GNCS) in which this thesis is based will be explained. Figure 10 presents the main control blocks of the GNCS of the UAV.
The first block is the Unmanned Aerial System (UAS). This block contains the main frame of the drone with the motors (actuators) and the sensors (IMU and LIDAR). It will be the most outer part of the system where the actuators will output all the commands from the control block and the sensors will be fed to the navigation block. Its maintenance and well being is essential for the correct functioning of the aircraft.
The control block is responsible for the stabilization of the vehicle (autopilot) and for making the vehicle follow a desired given trajectory (Path Following). These two elements receive the state information from the state estimation block and they end up controlling the UAV by sending commands to the actuators.
The Navigation block is responsible for estimating the state of the drone. The state estimator will receive inputs from the sensors and outputs them to the autopilot algorithm. Inside the navigation block, there is the perception block that is in charge of detecting the targets and avoiding obstacles. Their inputs come also from the sensors.
Finally, the Guidance block is in charge of the path planning and correction generating a final trajectory that will be sent to the control block.

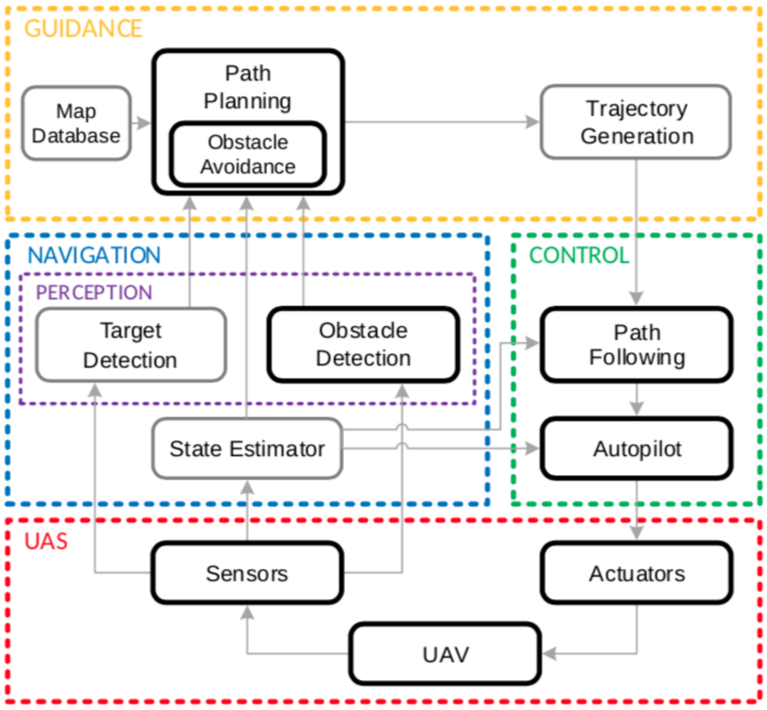


Figure 10: Guidance, Navigation and Control Structure [11]

4.1. Control

The control block includes the stabilization of the system and the trajectory control. In this thesis, a control stabilization Autopilot algorithm will not be implemented as is not the objective of this study and the stabilization of a quadcopter has been studied extensively. Instead, the same model used in the previous studies will be adopted. In this section, the most important elements and process used in the control block will be briefly explained.
Figure 12 shows a schematic of the control and path-following structures. The Autopilot will act as a link between the path following algorithm indications and the actual flight performance and stability of the quadrotor.

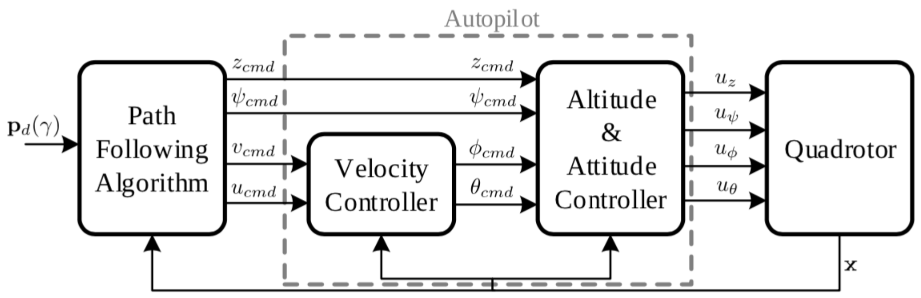


Figure 11: Structure representation for control and path following algorithms. [11]

4.2. Navigation

Navigation can be defined as the process of data acquisition, data analysis and extraction of information about the vehicle's state and its surrounding environment with the objective of accomplishing assigned missions successfully and safely [7]. In this study, the navigation algorithm will be in charge of guiding the UAV throughout a real test environment. It will be equipped with sensors such as GPS, accelerometers, magnetometers and the LIDAR. These sensors will be involved in the state estimation and perception algorithm.

4.3. LIDAR Considerations

Selection of the sensor:
This LIDAR was selected in Rubí's thesis taking different aspects into consideration. The first one is the size and weight. It evident that the sensor must be small and light. The second consideration must be regarding its power consummation. Some 360 degrees LIDARs despite having a very powerful 3D field of view present a high power consummation (and normally are heavier than simpler LIDARs). Other LIDAR presents a 2D reading in 360 degrees field of view. This solution despite its appearance it's not a good implementation for a flying platform as its orientation it's constantly changing. The chose LIDAR is a frontal sensor that is lighter with a good enough field of view, low power consummation and has no moving parts. The LeddarTech sensor Vu8 is a frontal LIDAR sensor with a field of view of 48° and a range of 85m.

Ground Defections
The main problem that has to be considered while working with LIDAR is that as the drone pitches forward the LIDAR will start sensing the ground. The same phenomenon will occur when the drone rolls. In the algorithm used these detections are taken into consideration. A parallel algorithm calculates the orientation and position of the drone in order to indicate the Odroid if it's sensing the ground or an object.

4.4. Guidance

Guidance is the part of the system that is in charge of carrying out the planning and decision-making functions to achieve assigned missions or goals [7]. This block takes all the inputs from navigation and creates a trajectory for the control system. The Guidance block in the UAVs performs the functions that otherwise will be assigned to a human pilot. In this study, the guidance control will follow a pre-established path and will perform obstacle avoidance tasks.

5. Algorithm adaptation and development

In this section, the main navigation algorithms used in this study will be described. The algorithms that will be tested are taken from the PhD [11]. The main algorithms are NonLinear Guidance Law (NLGL) and DDPG.

5.1. NonLinear Guidance Law

This algorithm is based on a Virtual Target Point (VTP). This point is created by intercepting a generated line tangent to the planned point and the circle of radius L around the aircraft. The VTP is constantly updated as the vehicle progresses, the vehicles tries to reach this point. The resulting vector will indicate the direction that the aircraft must follow. The speed will be constant and the yaw axis will target the VTP. The altitude will be fixed and determined by the set altitude of each point of the planned path.

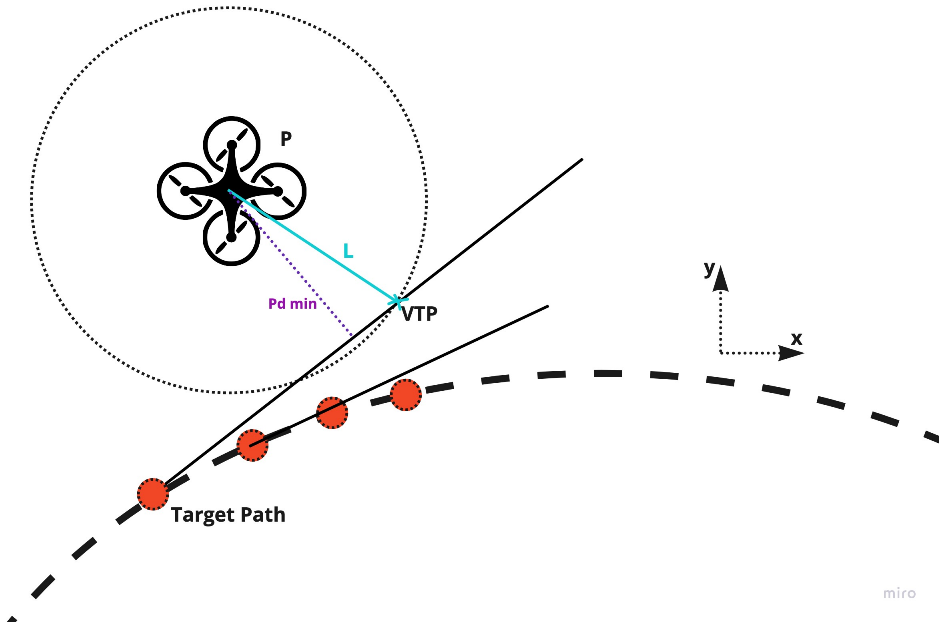


Figure 12: Obtaining of a VTP for a curve path with NLGL

This algorithm has two main variables that define the behaviour of the path following algorithm. These are L (radius of the circle) and δ (linear distance between the VTP and Pd_{min} point).

5.1.1. Adaptive 3D "NLGL"

The NLGL algorithm used for this study will be able to modify these parameters in order to adapt for a better trajectory in relation to its velocity and path shape. A Neural Network will be in charge of calculating L in relation to these two parameters. In fact this algorithm works by using an agent trained with DDPG that substitutes the traditional NLGL. This will be explained in the next section.

5.2. Deep Deterministic Policy Gradient

DDPG is an actor-critic reinforcement deep learning algorithm. This type of algorithms present some advantages such as good performance in continuous and large state-action space environments.
Figure 13 presents a representation for the structure of an actor-critic agent. The actor (policy) is presented independently

from the value function (critic). In relation to the learned policy function, the actor outputs the optimal action depending on the state of the environment. The critic will then estimate the value function in relation to the state and the action. This value function will give information on the expected accumulated future reward. The critic will also be in charge of calculating the difference error that will be feedback to the system for the learning process of the critic and the actor. Two neural networks are used in this algorithm for functioning as the actor and the critic.

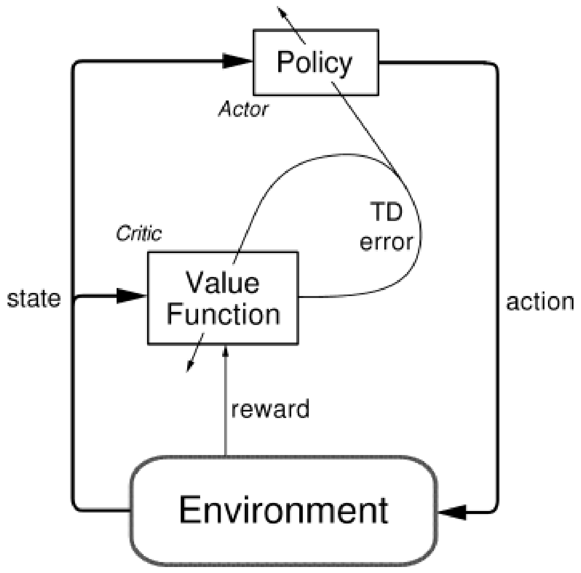


Figure 13: DDPG Actor-Critic-Environment relationship [15]

The DDPG algorithm used in this thesis has been trained in different agents. Those correspond to different mathematical parametrizations of the DDPG.

5.2.1. Agent 1:

The test with Agent 1 has the particularity that the deep reinforcement learning agent is only in charge of computing the reference of the yaw angle, it outputs a correction over the current angle that the vehicle must apply. The angle correction avoids using sudden fast angle changes. The other commands will be defined by the path following controller will only depend on the path following controller. The altitude is given by the altitude of the path at the closest point. The velocity is also predefined in relation to the path.
Agent 1 was trained following a half lemniscate (8-shaped) path at a constant velocity of 1 m/s. The path is discretized with a precision of 0.01m between each point.

5.2.2. Agent 2:

This approach deals with the anticipation that the system appears to have using the first agent. In this scenario, all the same parameters from the previous approach are maintained but another state variable is included. This state is an angle error between the drone yaw angle and the path tangential angle. This angle error is calculated not with the current referent path point that the vehicle is following but with one some steps forward (this distance can also be tuned for better performance). This approach allows the agent to know the current state and also anticipate complex path changes such as curves. Agent 2 is trained using the same considerations that in Agent 1.

5.2.3. Agent 3:

This final agent includes the characteristics of the previous but it also gives control of velocity corrections to the agent. The velocity will then adapt to the path shape. The training for Agent 3 was a little bit different as it needs a richer environment with different curves in order the learn the optimal velocities for each yaw angle. The radius of the training path was changed for every test.

Figure 15presents the results of each test. The first column of the table present the cross-track error $(\bar{d})$. The second one the total flight time of each test. The last column presents the average velocity of the drone during the test. As it can be appreciated, as the complexity of the agents increase the results get better. This behaviour is expected to behave equally in the real test flight.

	$\bar{\mathbf{d}}$ (m)	time (s)	$\overline{\|\mathbf{v}\|}$ (m/s)
Agent 1	0.1123	54.79	0.9476
Agent 2	0.0895	51.70	0.9484
Agent 3	0.0968	40.00	1.2338
Agent 3 $(v_{max} = 1)$	0.0816	54.41	0.9111

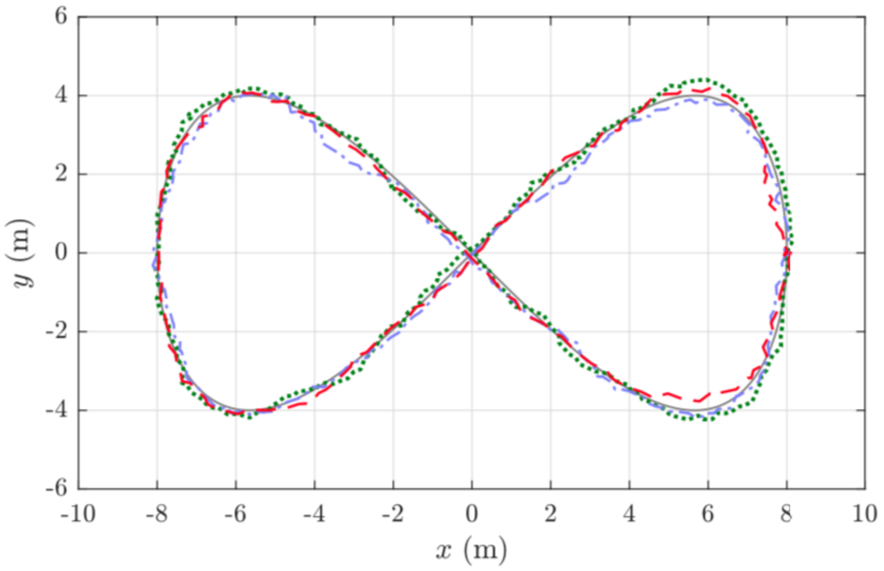


Figure 14: Training Results for each Agent [11] with sensor models: Agent 1 in green (dotted line), Agent 2 in blue (dash-dotted line) and Agent 3 (dashed line) in red.

5.3. DDPG with LIDAR Obstacle Avoidance

The algorithm used in conjunction with the third agent also includes a reactive Obstacle Avoidance with Deep Reinforcement Learning. This allows using the onboard LIDAR for detecting and avoiding static objects that may appear in the drone path. This algorithm uses the same Path Following (PF) agent presented in the previous sections and incorporates a new agent in charge of providing the reference path to the PF in such a way that the main functionalities of the PF agent are maintained while gaining the ability to surpass obstacles. The solution is divided then into two agents to facilitate the training process and reduce the size of the network

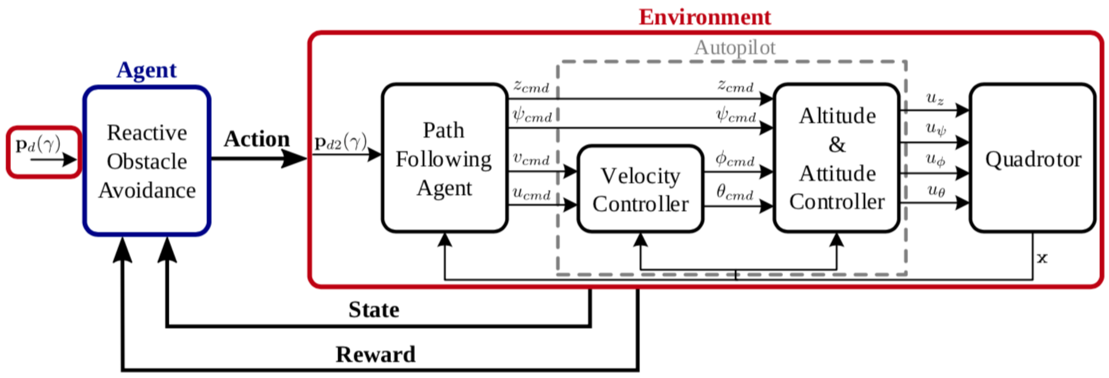


Figure 15: Reinforcement learning structure [11]

Reinforcement learning is structured with the agent being the Reactive Obstacle Avoidance and the environment being the path following agent, the autopilot, the quadrotor reference path and the quadrotor's environment. It's important to note that the Obstacle Avoidance (OA) agent receives the original reference path and is in charge of computing the reference path that is commanded to the path following agent
The code works with simulated objects and simulated LIDAR which will be replaced by the Leddartech sensor. The elements that conform the DDPG-OA algorithm can be described as such:

Action
In this structure, the obstacle avoidance agent receives the reference path $(p_d(\gamma))$ and sends it, possibly modified $(p_{d2}(\gamma))$, to the path following agent. The path is send using a state vector, the OA agent computes certain variation or increments over this state before sending it to the PF agent. The state vector of the PF agent is formed by four elements: the cross-track error, the angle error, the angle error in a point forward on the path and the velocity of the vehicle. The OA agent will only act on the cross-track error as if an offset to the reference path was being applied.

State
The first state that must be included in the path offset for the agent to know the real value of the path offset. Then the processed LIDAR measures are included. The cross-track error is also present in the state vector, it will allow the agent to localize the vehicle in the space helping to interpret the LIDAR data.
One of the problems with this approach is that the small field of view of the LIDAR combined with that the agent policy that only considers the current information provided by the state vector can crash the drone once the object is not visible. The UAV will try to move back to its original path and laterally collide with the obstacle. This problem has been solved by supplying historical information of the obstacles to the agent. In this case, the integral of the LIDAR measures were used as an indicator that recently an object was detected. Avoiding an immediate path correction waiting for low integral state values.

Reward
The reward function is defined using virtual zones around the obstacle objects. The outer perimeter area is the safety zone, the UAV can trespass this area but will receive negative rewards. This area is not encouraged. Concentrically inside there is the banned zone where the drone can not enter and a collision is detected if the UAV trespasses the area. Finally, there is the main obstacle which is also totally forbidden. The agent has been trained using these considerations.

5.3.1. Simulated LIDAR

The simulated LIDAR is a plugin that allows to train and test the different algorithms in the simulated environment as if the real Vu8 sensor was installed. This plugin aims to recreate the behaviour of the real LeddarTech equipment. The plugging works together with Gazebo in order to sense the placed objects in the simulation; normally just the cylinder. The virtual sensor publishes the information described below. From all that data the main important vector is the one called ranges. In this particular scenario is detecting an object appearing in the right field of view of the sensor.

```
header:
seq: 7978
stamp:
secs: 80
nsecs: 400000000
frame_id: "laz_base"
angle_min: -0.418879032135
angle_max: 0.418879032135
angle_increment: 0.104719758034
time_increment: 0.0
scan_time: 0.0
range_min: 0.0500000007451
range_max: 15.0
ranges: [inf, inf, inf, inf, 3.1328916549682617,
2.9748170375823975, 3.0170302391052246, 3.327003002166748]
intensities: []
```

The other information displays the base properties of the virtual sensor such as the max range, field of view and timestamp of each measure.

5.3.2. Leddartech LIDAR

The LIDAR from Leddartech will be explored fully in the next sections. The main objective will be to create an interface to process all the data to replace the virtual LIDAR. It will be necessary that all the data is processed and presented in the same manner as the simulated LIDAR. All the DDPG-OA models have been trained with the simulated LIDAR and the same behaviour will be expected.

5.4. ROS Path following and Algorithms Nodes

Al the navigation and obstacle avoidance algorithms are coded in different ROS nodes. The main nodes to launch (which a the same time have other secondary nodes associated with them) are described below.

5.4.1. Main nodes used for simulations with Gazebo

The nodes used for the simulation with Gazebo and RotorS are described below. As the actual AscTec platform is not used some additional nodes are needed in order to stimulate those elements.

- **hummingbird_hovering_example_with_gps_sensor.launch & ..._gps_LIDAR.launch:** These two nodes are groups of nodes that will launch the RotorS simulation with Gazebo where the Hummingbird will take off and stay in a hovering state until newer instructions are commanded. The first variant just includes the GNSS receiver and the second one also adds the LIDAR.
- **platform_emulator_ground_truth.launch & ..._sensors.launch:** This package simulates the topics of the real Hummingbird platform. It generates the same messages that the flight control board and radio send to the Odroid.

It allows the autopilot to communicate with RotorS and the navigation nodes. It has two modes; one with ground truth values and the other with sensors. The first one reads the real values from the estates and in the other the state values come from the sensory model.

- **platform_emulator start_exp:** This node shut down the autopilot integrated into the hovering RotorS nodes and starts the navigation algorithm nodes. It allows for the experiment to start. The Hummingbird will change from a hovering state to start following the planned path. This node will also change the state of the switch of the virtual RC controller from manual mode to automatic.
- **path_following_control ddpg_control.launch & nlgl_control.launch:** These nodes are the main navigation algorithm. Those will provide the path for the drone to follow as well as the method of trajectory navigation. In the case of the DDPG additional python scrips for the trained agents will need to be launched.

5.4.2. Main nodes used for Flight

The nodes for launching the actual AscTec Hummingbird into flight are presented below. The first list corresponds to the nodes launched in the onboard Odroid computer. The second list are the ones which run in the ground station computer.

- **asctec_hl_interface fcu.launch:** This node launches all the elements from the AscTec packages. It will start the communication between the Odroid and the flight controller. This node will also provide some topics with information regarding the state of the drone, link communication, battery status,...
- **asctec_hl_gps set_gps_reference_node:** This node fixes the origin of coordinates for the transformation from reference Global Navigation Satellite System (GNSS) coordinates to a usable X and Y system. It read the data from the Onboard GNSS receiver
- **asctec_hl_gps gps_conversion_node:** This node performed the conversion from a global system of reference to the local system of reference for the GNSS coordinated. It goes in tandem with the previous node.
- **path_following_control ddpg_control.launch:** This is the exact same nodes as in the simulation environment. In this case the DDPG agents will be launched separately.

Processes launched from the Ground station computer that works together with ROS.

- **python3 ddpg.py:** This are the DDPG algorithm that will allow the drone to navigate. These algorithms are not launched in the Odroid as the machine learning performed by Tensorflow requires a 64-bit computer and new generation CPUs.
- **python3 ddpg_OA.py:** This are the obstacle abidance algorithm that will allow the drone to process the obstacles and perform the avoidance manoeuvre. As before these algorithms are not launched in the Odroid as the machine learning performed by Tensorflow requires a 64-bit computer and new generation CPUs.

6. Gazebo Simulations

In this section, the Rotors simulations will be tested and reviewed. The correct operation of this part will be essential for advancing to the real field test. In this section, the algorithms and nodes created in the PhD will be tested and recreated before being integrated into the real quadcopter.
The two-path following control algorithms that will be tested are NLGL and DDPG. In these simulations, different test variants will also be done in order to study the different ROS agents.

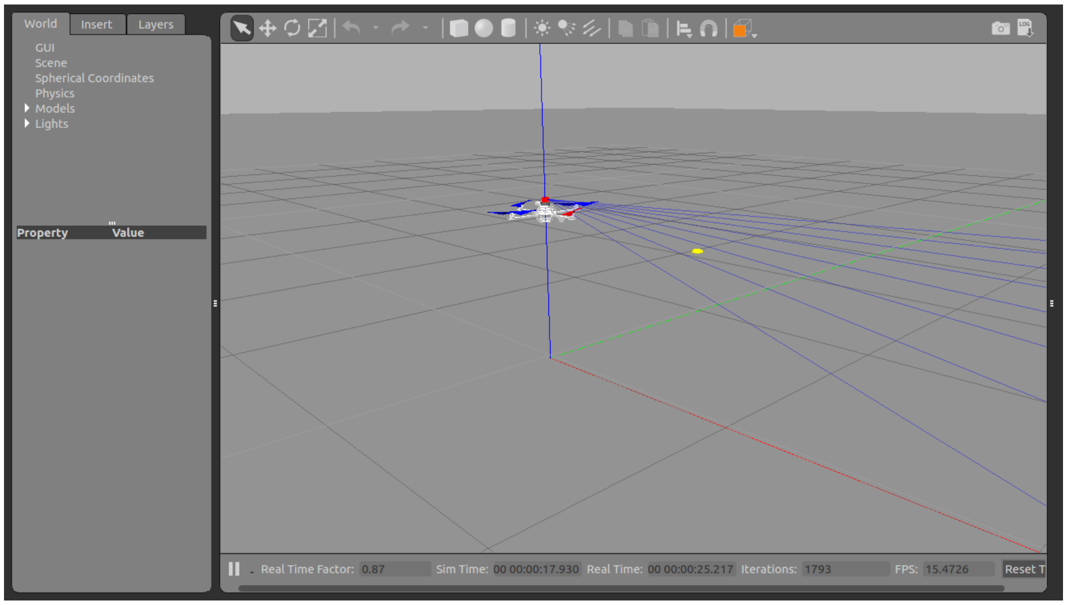


Figure 16: Gazebo Hummingbird Simulation with Leddar

Figure 16 presents a screenshot of the Gazebo interface with the simulation running. The Hummingbird model used in this scenario includes the LIDAR. The sensor is represented with its eight sensing areas. As the sensor is installed vertically the detecting surfaces extend horizontally.

Recording all the ROS flight data can be done using the .bag system. Bags are typically created by a tool like rosbag, which subscribe to one or more ROS topics, and store the serialized message data in a file as it is received. These bag files can also be played back in ROS to the same topics they were recorded from, or even remapped to new topics. Using bag files within a ROS Computation Graph is generally no different from having ROS nodes send the same data, though you can run into issues with timestamped data stored inside of message data.
The bag file format is very efficient for both recording and playback, as messages are stored in the same representation used in the network transport layer of ROS [10].

For using the rosbag tool the following commands can be done.

```
rosbag record -o "NAME" --duration=="TIME(s)" -a
```

The generated file will be stored in the current directory.

6.0.1. Process for launching the simulations

The simulations can be launched using the next sequence of commands. The nodes are launched in separated terminal tabs. Each type of simulations will have its particular codes and launchers. The presented one is the most complex sequence but as a general overview this could be followed:

```
roslaunch rotors_gazebo hummingbird_hovering_example_gps_LIDAR.launch
roslaunch platform_emulator platform_emulator_ground_truth.launch
roslaunch path_following_control ddpg_control.launch
python3 ddpg_PF.py #On the file folder
python3 ddpg_OA.py #On the file folder
rosrun lidar listenerLIDARfastV3.py
rosrun platform_emulator start_exp
```

6.1. NLGL - Ground Truth

This algorithm is the simplest one and allows for testing the system. In this test, a lemniscate path is followed. Once the hovering examples are launched the drone starts to hover at a fixed height. When the NLGL algorithm is launched the UAV starts to follow the path. Figure 18 shows a three-dimensional representation of the followed path.
Figure 17 present the rqt_graph for the NLGL simulation. The first box called $/hummingbird$ is a topic where it has differed topic such as $joint_{s}tate_{p}ublisher$ or $robot_{s}tate_{p}ublisher$ the first subscribed to the other. Then there is also the $Gazebo$ node which also is subscribed to $robot_{s}tate_{p}ublisher$. At the right part of the diagram, there are all the nodes related to the attitude determination and control and path-following control. These nodes are subscribed to the other reading the published topics. The flight controller topics $/fcu$ are subscribed to the control and navigation topics and will be the ones in charge of translating that information back to the simulation environment. The general workflow of the nodes can be summed up as follows. The $/emulated_{r}cdata$ trigger the simulation with the virtual RC controller. The simulation executes the $(nlgl_{c}ontroller_{n}ode$ which will start to publish and receive information from the flight controller nodes. Then this information will be read by $/translate_{t}opic$ node which will send it to Gazebo. The $translate_{t}opic$ node will also read data from the simulated Gazebo environment that will be sent back to the flight controller through the topic $/fcu/imu$.
All the topic involved in this simulation can be found in the Annex section D.1.

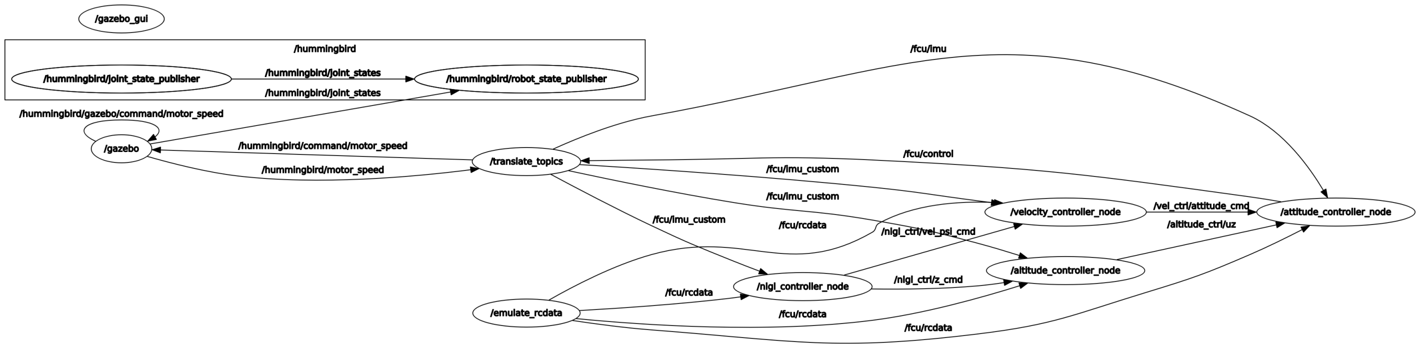


Figure 17: NLGL path following Nodes and topics visualization

As it can be seen in figure 18 the UAV is capable of following a complex path without heavy oscillations, misalignment with the path or any other perturbations. The circuit is followed uninterruptedly. This result was expected as the NLGL algorithm was already tuned in the previous PhD Thesis.

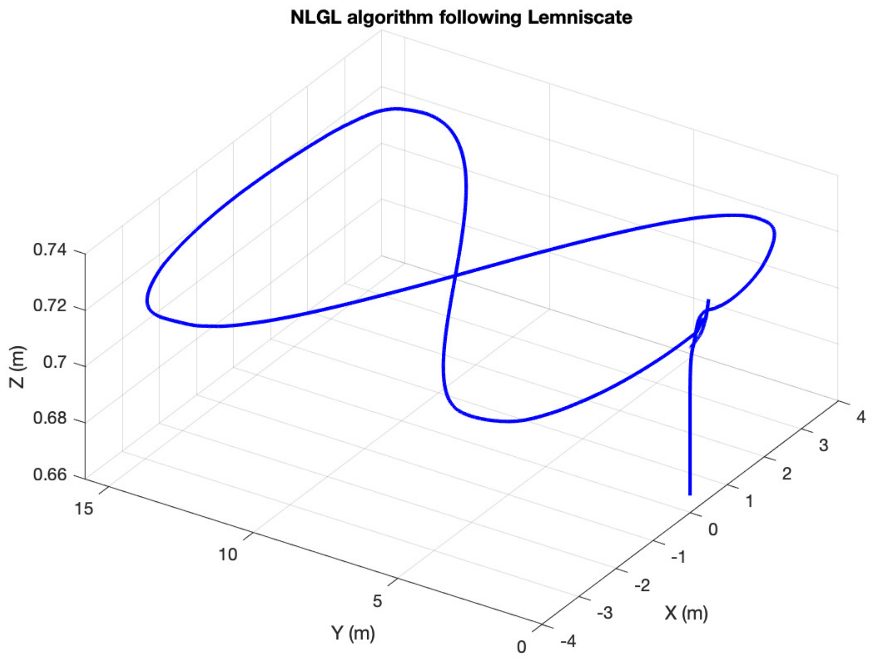


Figure 18: Lemniscate path following with NLGL side view

The assuring part of the previous test is that despite the iterations and variation in the start condition the drone is capable of always following the same path. Figure 19 presents the same simulation with a top view; the drone followed the planned lemniscate without any problem.

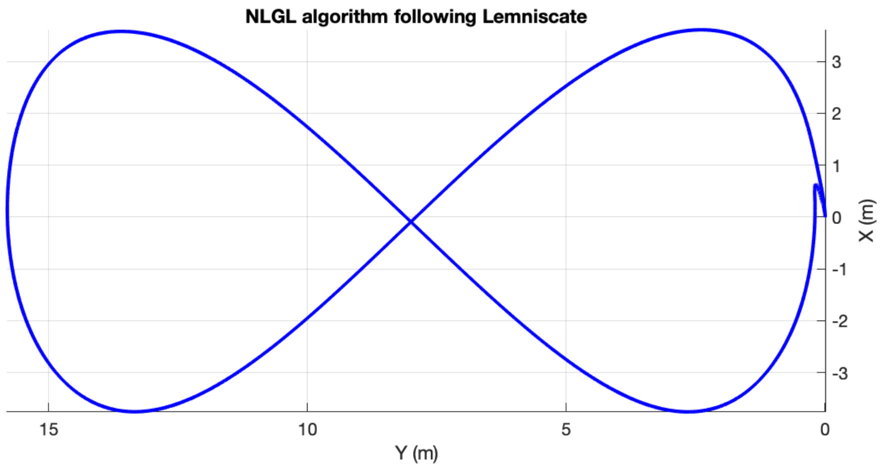


Figure 19: Lemniscate path following with NLGL top view

6.2. DDPG - Ground Truth

In this section, the DDPG algorithm has been tested in the simulated environment. The planned path is a Lemniscate. As it can be seen in figure 20 the simulated UAV is capable of following the planned path without any inconveniences. The path is traced fluidly without missteps.

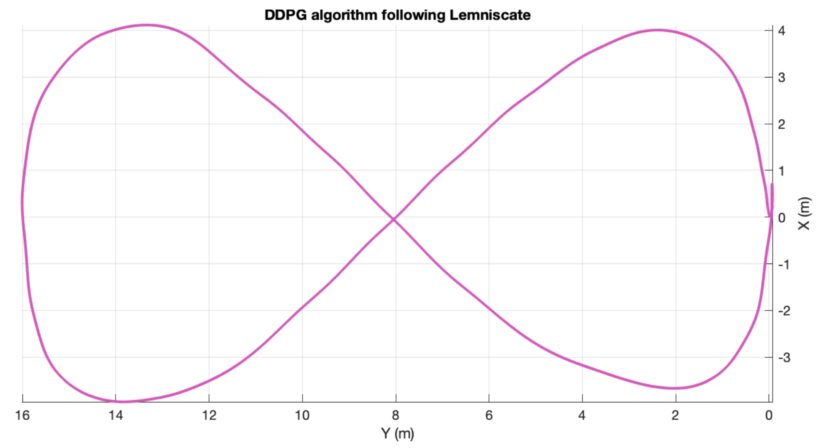


Figure 20: DDPG Lemniscate simulation in Gazebo top view

Figure 21 present the same simulated run from a side perspective. It can be seen that the drone is capable of maintaining a vertical precision of 0.05m. It has to be taken into consideration that the scale is distorted to show the ripples in the vertical axis but in fact, those are very small and almost not noticeable when looking at the drone. Nevertheless the DDPG algorithm despite completing the simulation with faster results it struggles a little bit more to maintain a constant altitude.
These oscillations will be evident when the UAV is equipped with the virtual LIDAR as the raw measures will hit the ground in regular oscillations.

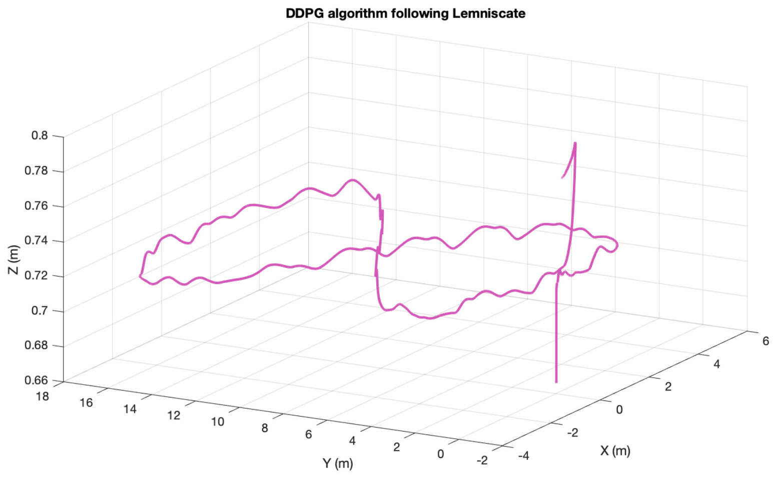


Figure 21: DDPG simulation in Gazebo lateral view

6.2.1. DDPG Spiral Path Simulation

In this section the DDPG algorithm was executed with the trained agent 3. This path was tested in order to verify the correct functioning of the agent as well as for having a reference path for comparison when working with the DDPG-OA. Figure 22 presents the tridimensional path. The simulated Hummingbird is able to perform the programmed geometry without any problems. It is interesting to notice that in this simulation the UAV is flying very close to the ground.

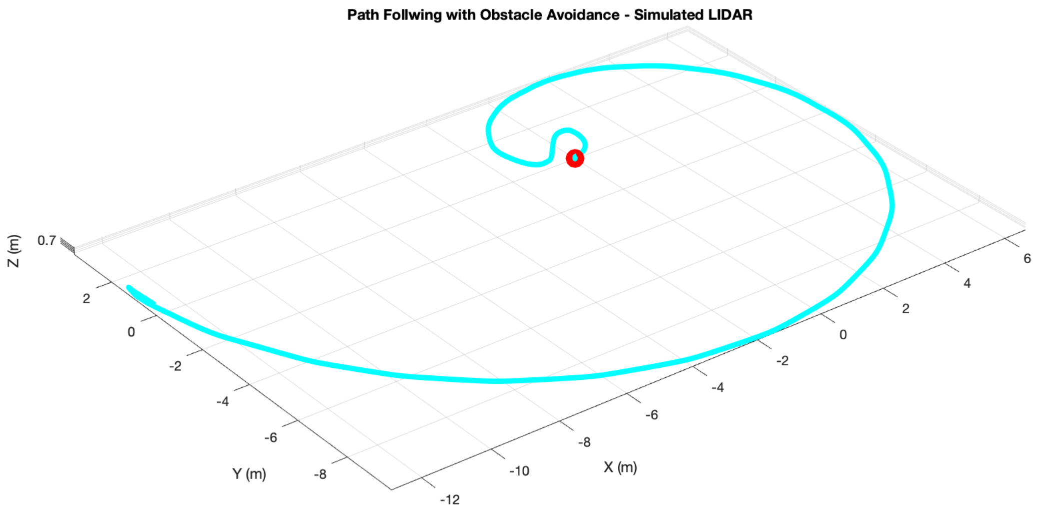


Figure 22: DDPG Spiral Path simulation in Gazebo

In this simulation, the Hummingbird was also equipped with the virtual LIDAR despite not being used for any of the algorithm (it will be used in the next section for the obstacle avoidance). The simulated LIDAR was mounted on the drone for having some reference measures when no object is present. Of all the different test this was chosen to be the one with the LIDAR as the drone is flying very close to the ground. The artefacts present in Figure 23 are caused by ground detections. As the drone is flying very close to the ground (less than one meter) it's detecting the ground as it's pitching forward. The LIDAR measures resemble pulses due to the drone forward oscillations. This reference footage will be very useful for identifying the real detection from the ground detections.

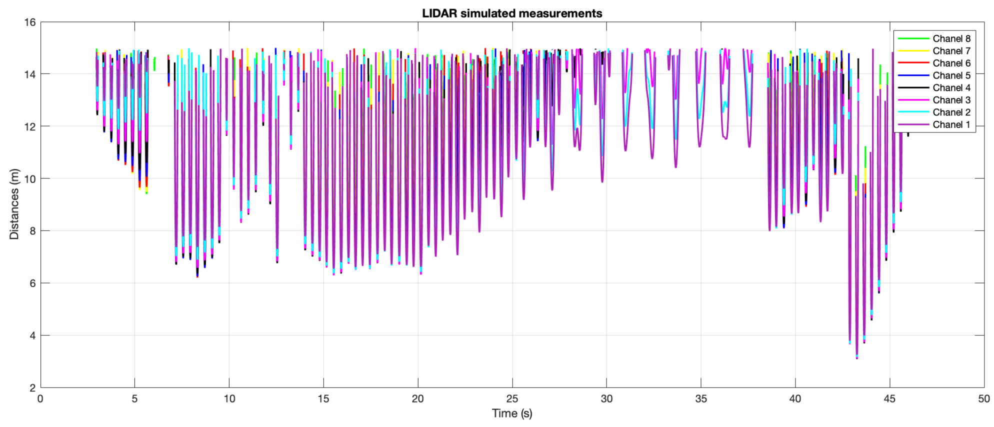


Figure 23: DDPG simulated LIDAR With no objects in filed

6.3. DDPG - OA

In this simulations, the DDPG-OA algorithm was tested in a simulated environment. The LIDAR sensing is also being simulated by the same nodes. The programmed path is a straight line that collides with the cylindrical obstacle.
Figure 24presents the trajectory followed by the drone. As it can be clearly appreciated the UAV moves laterally and surrounds the object in order to avoid it.

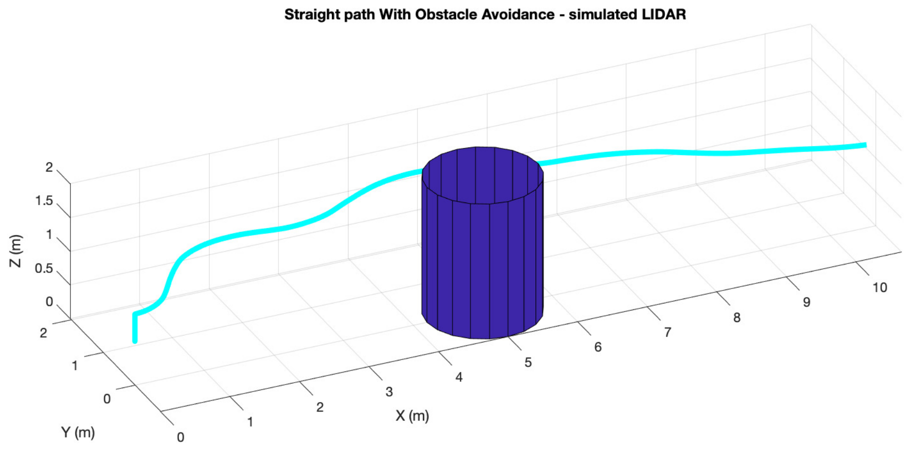


Figure 24: DDPG-OA simulation in Gazebo side view

Figure 25 presents the top view of the same experiment. Again it can be seen that the drone avoids the object maintaining the safe zones and bane zones distances. Once the object has been surpassed it comes back to the original path.

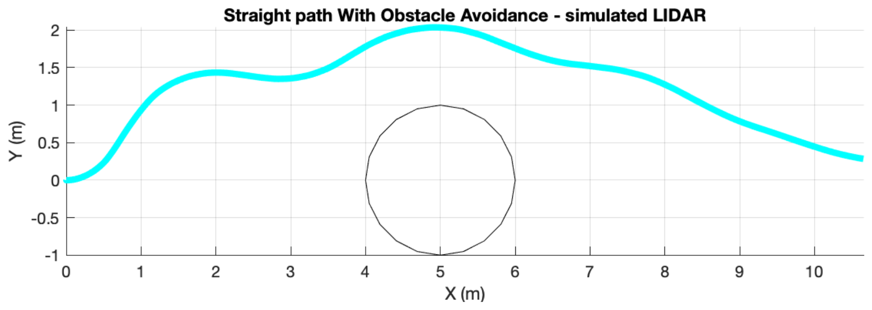


Figure 25: DDPG-OA simulation in Gazebo top view

Figure 26 and 27 presents the data extracted from the virtual LIDAR. The right plot presents all the eight channels from the LIDAR and the left plot present the lateral and centre channels. These plots are interesting as they allow us to understand what the LIDAR is sensing and the decisions that the DDPG algorithm is taking. Chanel 8 of the LIDAR does not detect the object unless the drone rotates. On the other hand channel, 1 detects the object as the drone is surrounding it. Channel 4 as its located in the centre of the array and detect the object from the initial moment. It's to one to prompt the movement of the drone. It's interesting to notice that channel 1 detect the object as the drone is turning counterclockwise at Nº of measure 550. The ripples present in the other measures are due to the drone rolling to keep sensing the object while it's surpassing it.

This data will be very useful because it will be a baseline for comparing with the real test.

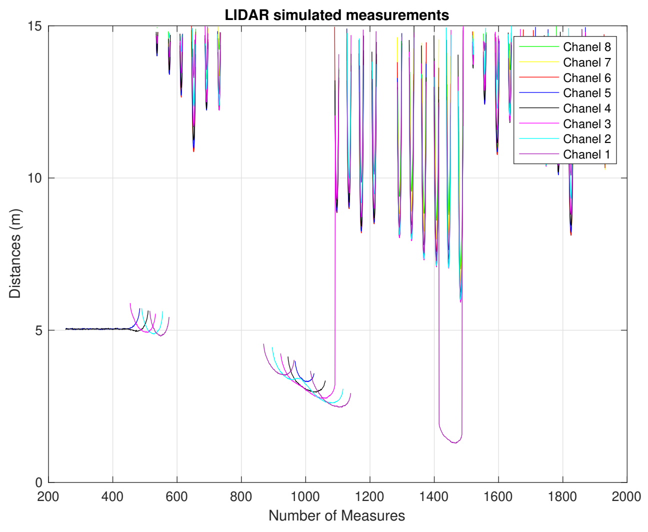


Figure 26: Virtual sensed LIDAR for DDPG-OA all measures

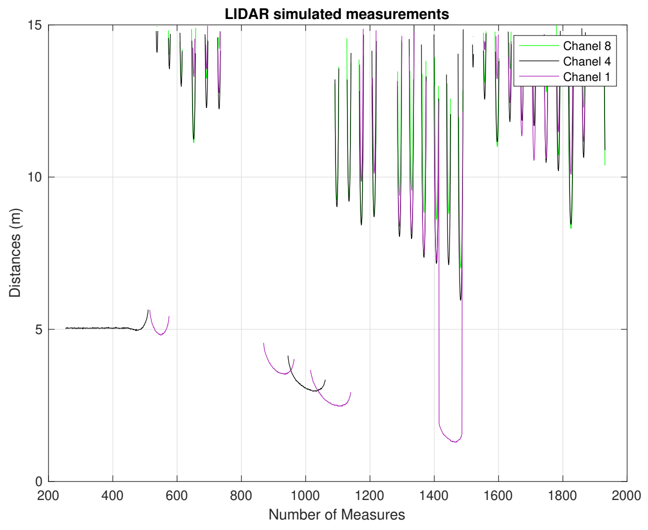


Figure 27: Virtual sensed LIDAR for DDPG-OA all measures separated measures

It's interesting to remark the ground detections artefacts present in the virtual LIDAR. Despite that, the real detections are clearly identifiable as the LIDAR measured distances changes drastically.
Figure 28 presents the same LIDAR data but separated by channels. This plot has been done in order to facilitate the understanding of the LIDAR data as sometimes the previous plots are not very intuitive. The advantage of this representation is that it can be easily understood ho the object is being seen. In this plot, the object is first detected in the central channels of the LIDAR. Then as the drone rotates the detections appears in the right channels. The left

channels never see the objects. As the drone rotates feeling the cylinder the detection alternate between the central and right channels. As before the other pulses are irrelevant as they represent the ground detections.

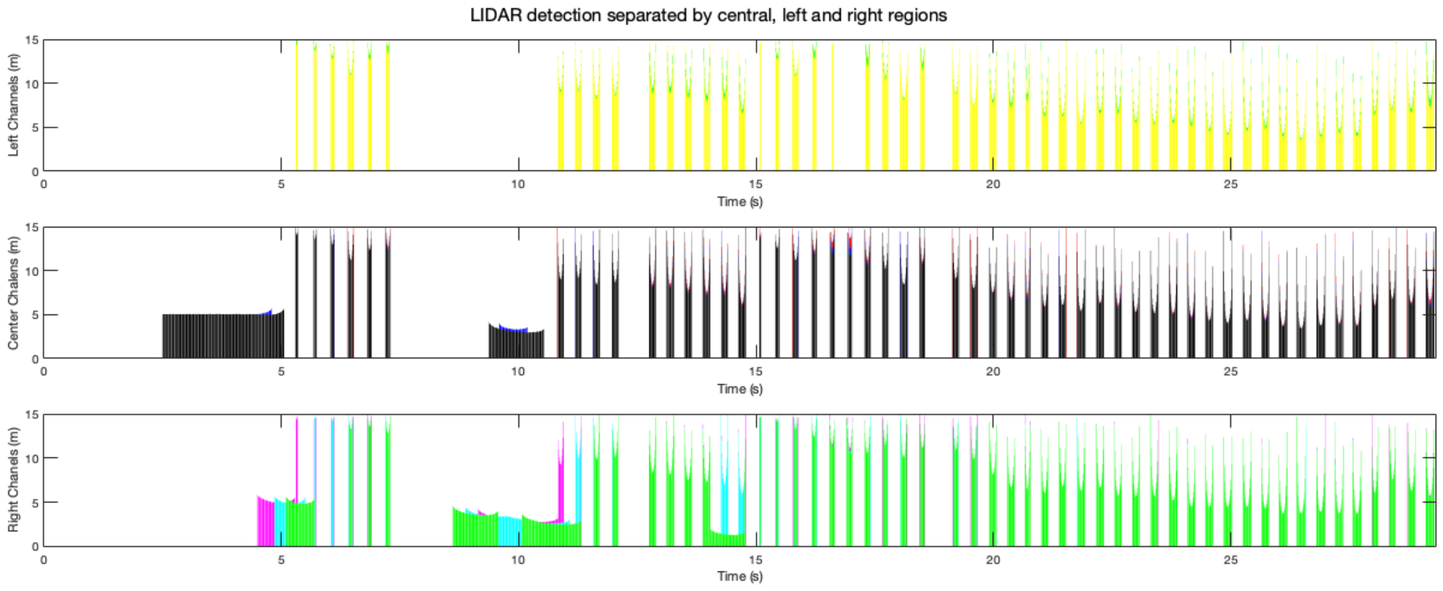


Figure 28: Virtual LIDAR for DDPG-OA separated measures per sections

6.3.1. DDPG-OA Test 2

In this section, the results of the simulation for the DDPG-OA in a lemniscate geometry are presented. Despite being a more complex geometry the simulated Hummingbird is capable of navigating and also following the programmed path. Figure 29 presents the tridimensional view of the simulated environment with the track of the UAV marked in blue. The red dot identified the starting point of the UAV.

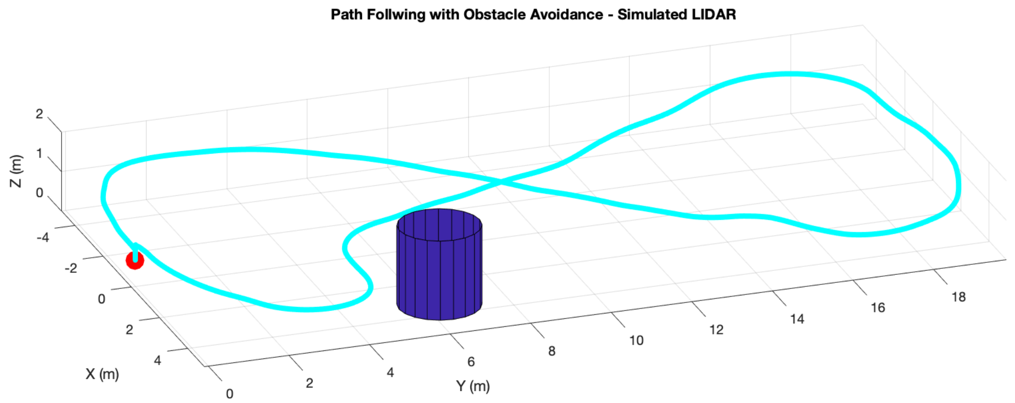


Figure 29: DDPG-OA simulation in Gazebo side view

Figure 30 shows the top view of the simulated experience. It can be clearly noted how the hummingbird is able to detect the cylinder and avoid it being able to return to its objective path.

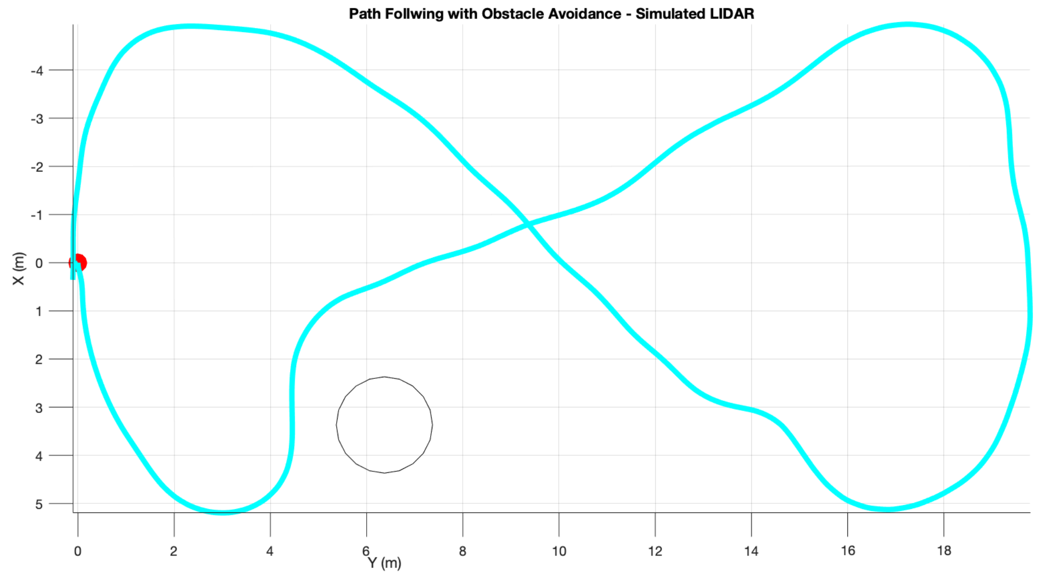


Figure 30: DDPG-OA simulation in Gazebo side view

Figure 31 presents the complete path of the Hummingbird where the X and Y coordinates are presented in relation to the LIDAR measurements. If all the detection from the ground are not taken into consideration it can be seen that at second 12 the cylinder is clearly detected by the three channels. At second 42 is detected again but from afar while the Hummingbird is performing the return turn.

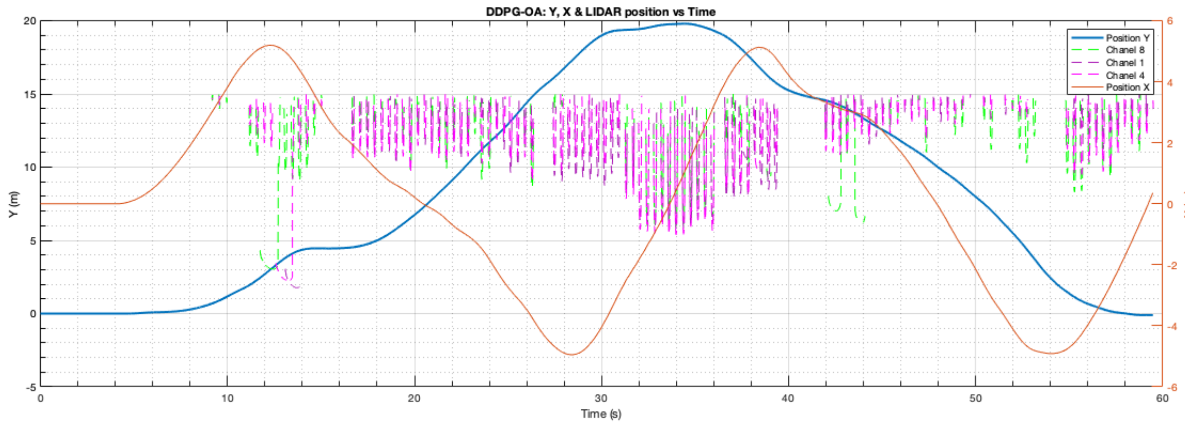


Figure 31: DDPG-OA simulation in Gazebo side view

Figure 32 present all the raw data from the virtual LIDAR. The two mentioned instants are very notable as they differ from the trend. All the channels are detecting the cylinder in those instants.

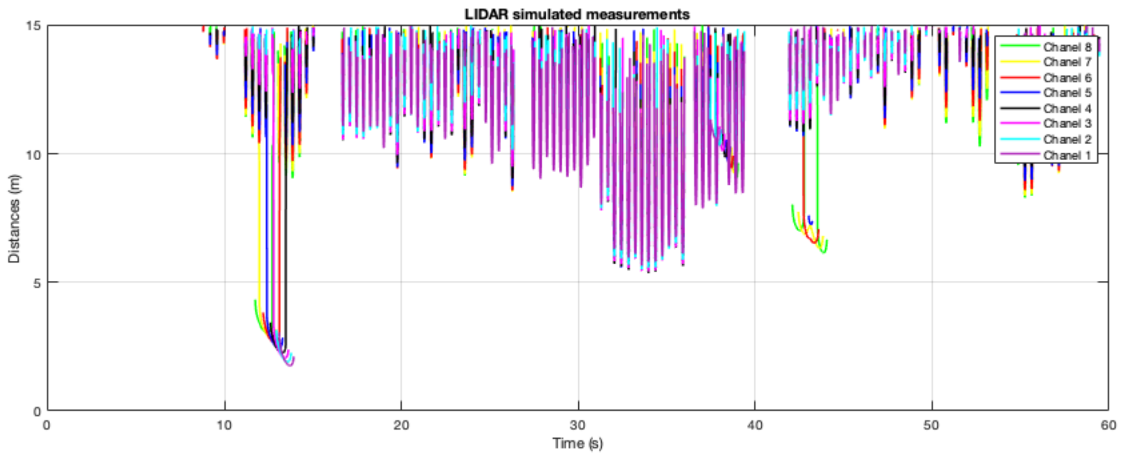


Figure 32: DDPG-OA simulation in Gazebo side view

The following plot seen in figure 33 presents the data of the virtual LIDAR detections separated by central, left and right sections. This plot operates as in the previous test. The solid colour regions show the detections and the spikes the ground detections. It can be seen that the object is simultaneously detected by all the channels in the first encounter (instant 14s). In the second encounter at second 37, the object is seen by the central and right sections of the sensor. In the third encounter (second 44) the object is seen by the central and left channels.

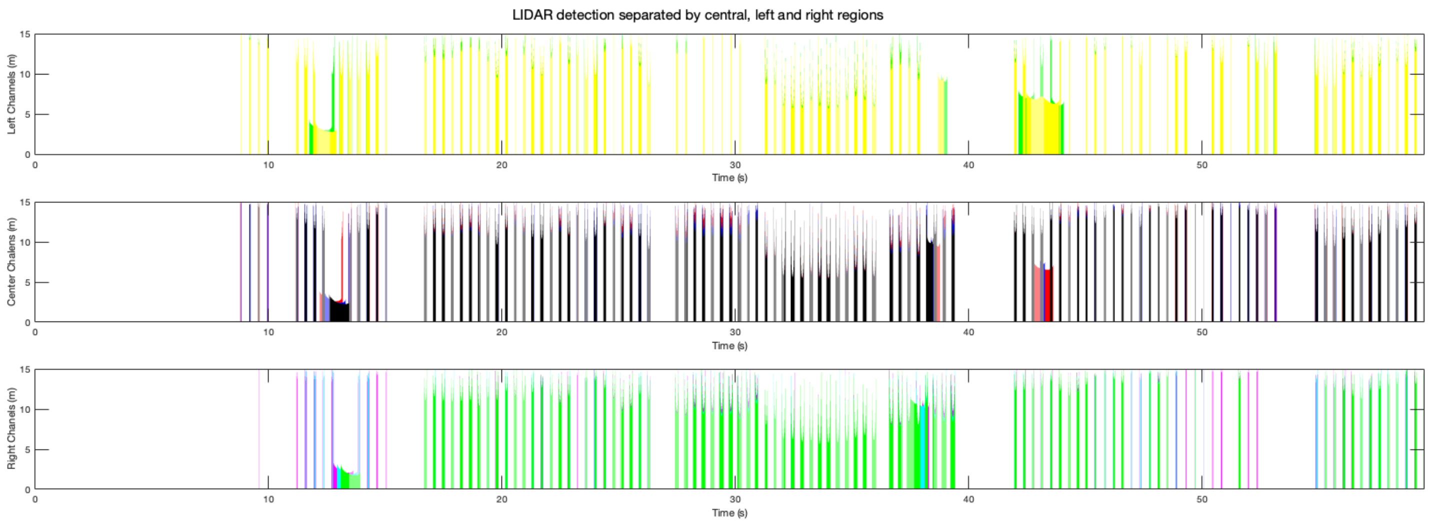


Figure 33: Virtual sensed LIDAR for DDPG-OA separated measures for sections

6.3.2. DDPG-OA Test 3

In this section, the DDPG-OA algorithm has been tested again in a fully virtual environment with a spiral path. The object was placed near the begging of the track. As it can be observed in figure 34 the UAV was able to navigate around the object and follow the path without complications.

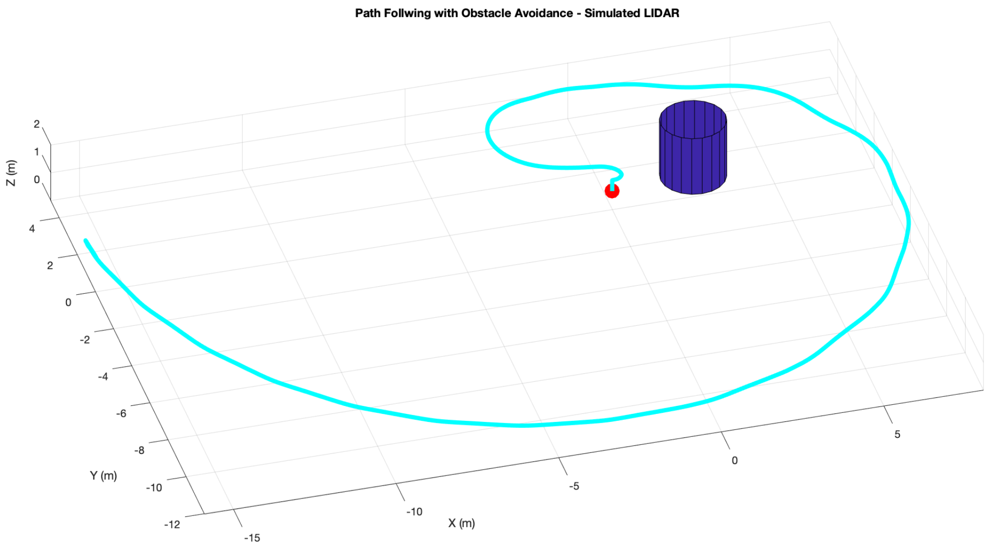


Figure 34: DDPG-OA spiral path simulation in Gazebo side view

Once the UAV starts following the programmed path it performs a hard turn and imitates the spiral in a slightly different path from if there were no objects. Once the object is left behind the drone progresses without any problem.

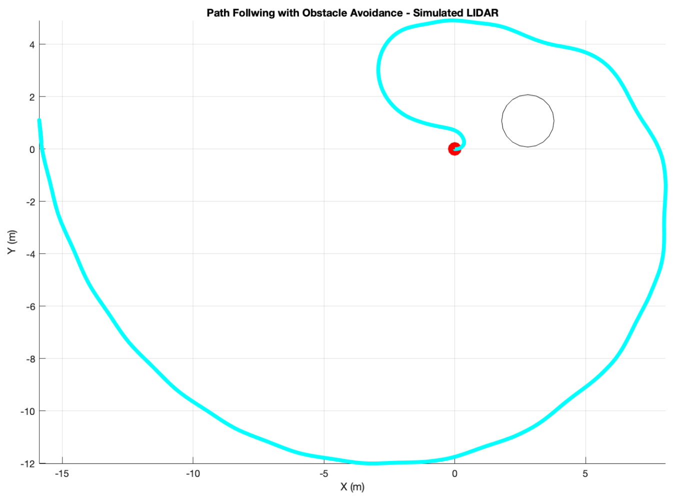


Figure 35: DDPG-OA spiral path simulation in Gazebo top view

Figure 36 presents the combined X and Y position of the UAV with the LIDAR measures. The only detection of the object occurs a the beginning of the path. It can be seen at second 8. The Hummingbird model slightly changes its trajectory to avoid it.

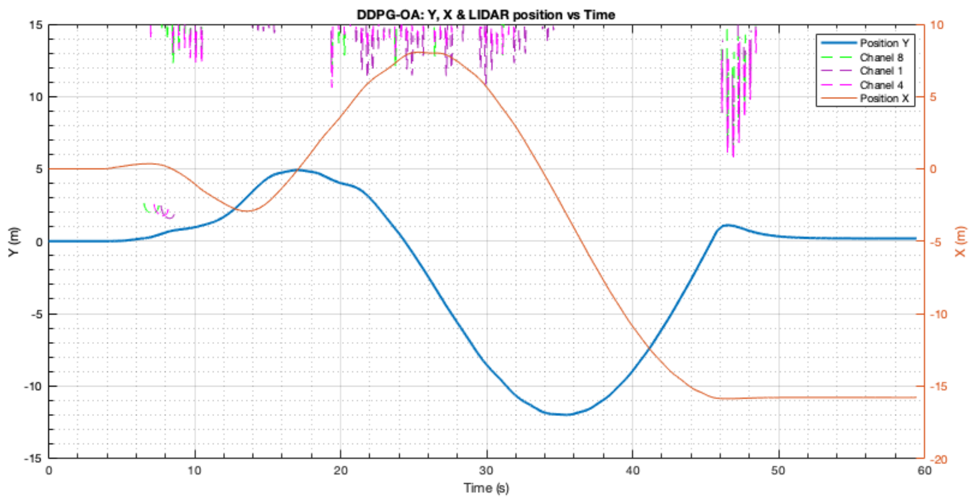


Figure 36: DDPG-OA X and Y position with LIDAR

This lats figure (plot 37) present the raw measures from the virtual LIDAR the noise from the ground detections are present but quite insignificant. The peak at 8s corresponds to the LIDAR detecting the cylinder. After this point as expected no more detections of the body were done.

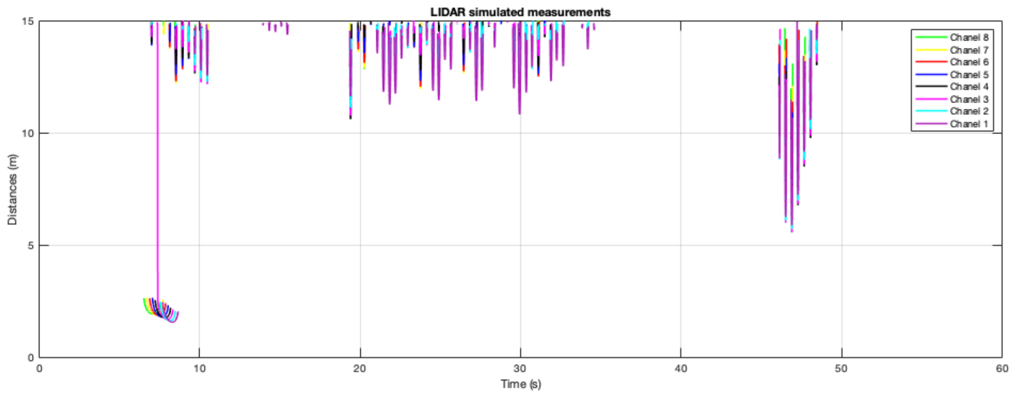


Figure 37: DDPG-OA simulated LIDAR

6.3.3. DDPG-OA Test 4: Multiple Obstacles

In this test, the DDPG-OA algorithm was modified in order to create multiples obstacles. The objective was to further validate the capacity of the system to perform complex avoidance manoeuvres. As it can be seen in figure 38 the drone was capable of avoiding the cylindrical objects and returning to the planned path.

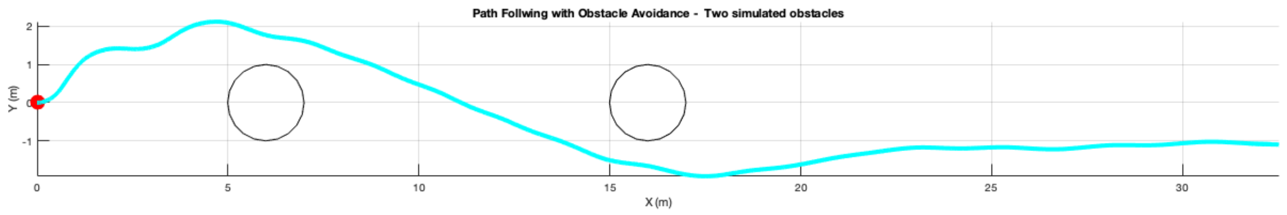


Figure 38: DDPG-OA with two obstacles top view

The Hummingbird also performed the most efficient avoidance manoeuvre. It used the initial inertia and direction after passing the first object to surpass the second object by the opposite side. Once the objects were left behind the drone returned to the initial path. It had some difficulties returning to the precise initial trajectory. Nevertheless, the trajectory was very robust.

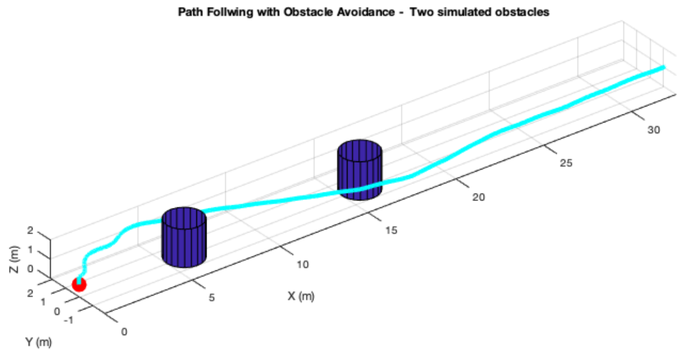


Figure 39: DDPG-OA with two obstacles frontal view

Figure 40 presents the combination of the X and Y coordinates and the virtual LIDAR. As it can be seen the ground detections are still present but the obstacles are clearly visible. The two detections can be distinguished from the ground detections are they are at a distance of 4 to 2 meters.

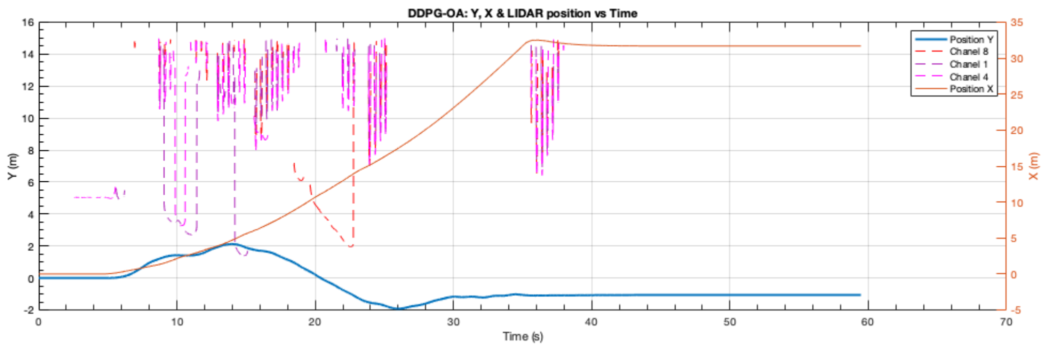


Figure 40: X, Y coordinates with LIDAR - Multiple obstacles

6.3.4. DDPG-OA Test 5: Multiple Obstacles

This test is similar to the previous one but a third obstacle is incorporated in order to for the UAV to perform very close to its operational limits. The drone has to pass between the two obstacles. As is presented in figure 41 the drone was capable of performing the manoeuvre.

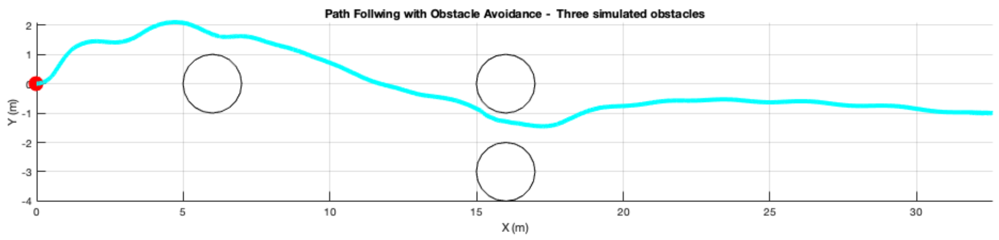


Figure 41: DDPG-OA with three obstacles top view

Figure 42 presents the tridimensional view of the path. The drone with some difficulties manages to pass between the two obstacles. The gap was 1m wide.

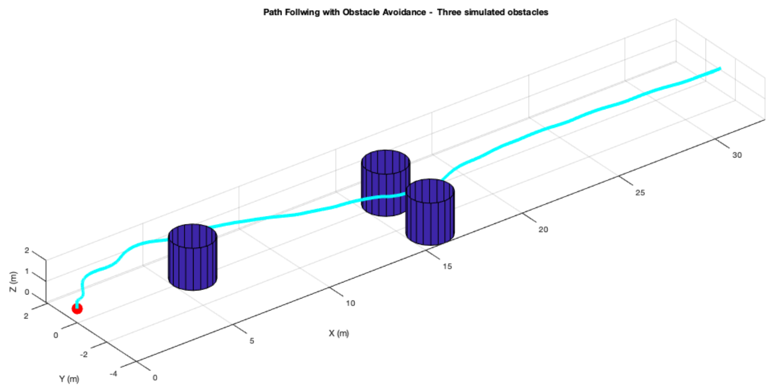


Figure 42: DDPG-OA with three obstacles frontal view

In figure 43 the X and Y coordinates with the LIDAR are presented. This plot can be read as the previous one. The main difference is that the LIDAR detections of the objects appear more often as there is a third object.

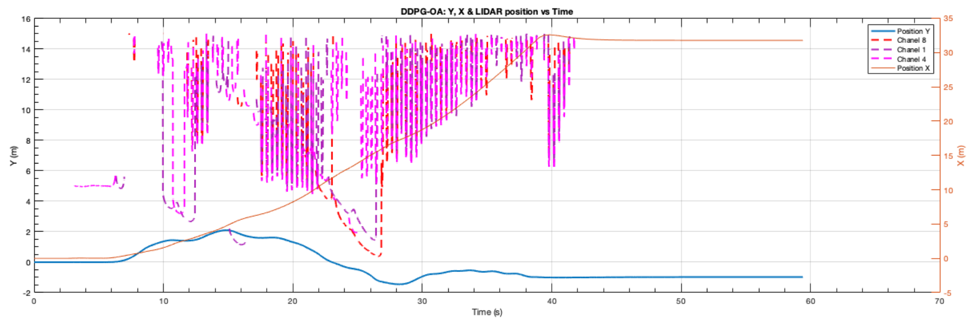


Figure 43: X, Y coordinates with LIDAR - Multiple obstacles

6.3.5. DDPG-OA Test 6: Multiple Obstacles

This final test was done again with three obstacles. The obstacles are lined up on the programmed straight path of the drone. As it can be seen similar result from test 4 are obtained but with an additional manoeuvre. The UAV successfully avoided the tree objects while performing the path following algorithm.

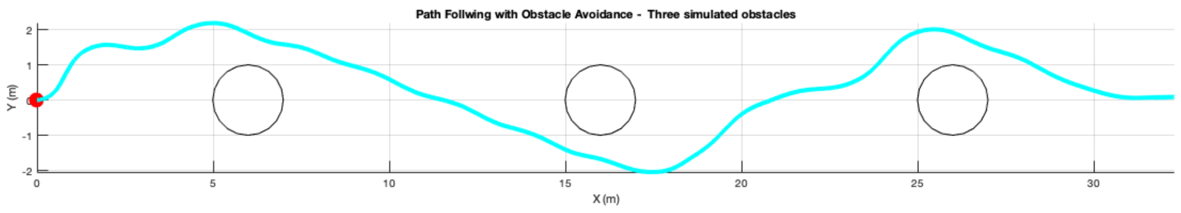


Figure 44: DDPG-OA with three obstacles top view

Figure 44 and 44 present the plot for the manoeuvre with the obstacles represented. The UAV avoided successfully and efficiently the tree objects.

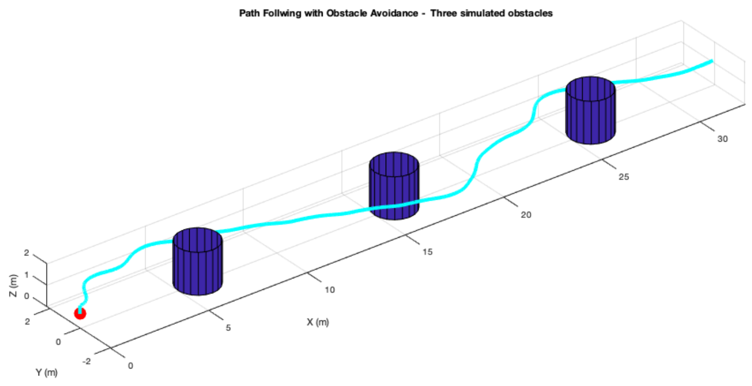


Figure 45: DDPG-OA with thee obstacles tridimensional view

Figure 46 presents the LIDAR detections with the X and Y coordinates. The objects can be clearly distinguished from the other detections as they are only detected partially by some channels. As in the other test, the avoidance manoeuvres performed in the Y-axis correlate perfectly with the virtual sensor detections.

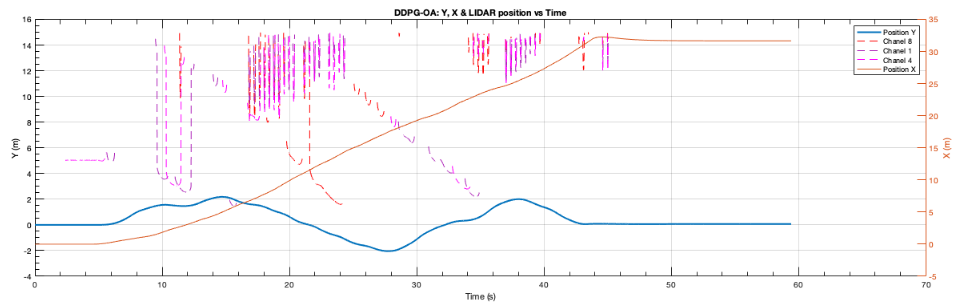


Figure 46: X, Y coordinates with LIDAR - Multiple obstacles

7. LIDAR Integration

In this section, a custom code with several nodes and topics have been created in other to read the data from the LIDAR and send it to the DDPG with Obstacle Avoidance algorithm in order to replace the simulated sensor. Different approaches have been tested in order to process and transform all the data into usable measures for the DDPG-OA algorithm. The two communication methods worked fine but one was easier to launch and more robust. In order to create all the codes, a new ROS package was created under the name $lidar$. In there all the scrips are coded.

7.1. LIDAR integration to ROS

This LeddarTech sensor can be programmed and integrated into a huge variety of platforms using languages such as Python or C++. In the following section, the system will be integrated into the created ROS environment for this thesis.

7.1.1. Method 1: Python + ROS

This method was implemented in two parts. The first one was a launcher that establishes communication with the LIDAR and published all its data under the standard $PointCloud2$. This standard is designed for communicating LIDAR sensor with programs such as Rviz. As it will be apparent later this raw format is not meant for working directly with the data. The second part was a python code that created a new node to read the published $PointCloud2$ data and process it.
The first ROS launch file was modified from the $Example.launhc$ provided by LeedarTech. This file was modified in order to just start all the nodes related to the data reading of the sensor but skip all the Rviz related topics. This was essential as the Odroid Rviz is not available and would have crashed the node. The launcher publishes the following topics.

```
/LeddarTech_1/scan_cloud
/LeddarTech_1/scan_raw
/LeddarTech_1/scan_triangles
/LeddarTech_1/specs
```

The relevant topics were the Scan Cloud and the Scan Triangles. The $scan\_triangles$ format is a topic that publishes a triangular mesh of the area that the sensor is measuring. The $scan\_cloud$ topic publishes a raw vector of distances and other information of the measurements. This one was an interesting topic as the measured distances are the inputs needed for the DDPG-OA algorithm.
As mentioned the Scan Cloud format is not user friendly, this topic publishes a 128 vector with seemingly random numbers.

```
header:
  seq: 25
  stamp:
    secs: 1619374797
    nsecs: 427628040\
  frame_id: "sensor_frame"
height: 1
width: 8
fields:
  -
    name: "x"
    offset: 0
    datatype: 7
    count: 1
  -
```

```
        name: "y"
        offset: 4
        datatype: 7
        count: 1
      -
        name: "z"
        offset: 8
        datatype: 7
        count: 1
      -
        name: "intensity"
        offset: 12
        datatype: 7
        count: 1
    is_bigendian: False
    point_step: 16
    row_step: 128
    data: [98, 72, 78, 63, 72, 158, 147, 190, 180, 171, 32, 179, 102, 210, 135, 67, 18,....]
    is_dense: True
```

The second node created was in charge of processing the data using the library *point_cloud2.py* [2]. This second code acted as a listener and publisher for the processed data. Once the data were processed it gained a more coherent format.

```
[(0.7354807257652283, -0.2631591856479645, -3.414485405528467e-08, 289.94854736328125)
 (0.4187788963317871, -0.10489865392446518, -1.8870945694970942e-08, 387.22052001953125)
 (0.7495801448822021, -0.11118970811367035, -3.312370111530072e-08, 522.7318115234375)
 (0.8493937253952026, -0.04172803834080696, -3.717295626870509e-08, 540.1593627929688)
 (1.403473973274231, 0.06894826143980026, -6.14217867678235e-08, 315.4338684082031)
 (1.791629672050476, 0.26576313376426697, -7.917153510561548e-08, 225.85000610351562)
 (1.7832554578781128, 0.4466822147369385, -8.035676302142747e-08, 197.9754180908203)
 (1.8506622314453125, 0.6621775031089783, -8.591739941721244e-08, 151.67967224121094)]
```

These data could be finally processed to.

```
    ('Ch8: ', 0.73548073)
    ('Ch7: ', 0.4187789)
    ('Ch6: ', 0.74958014)
    ('Ch5: ', 0.84939373)
    ('Ch4: ', 1.403474)
    ('Ch3: ', 1.7916297)
    ('Ch2: ', 1.7832555)
    ('Ch1: ', 1.8506622)
```

This approach was abandoned as a simpler and cleaner way to send data from one node to the other was found in method 2. The conversion from point cloud to vector using the *point_cloud2.py* was unreliable and sometimes the calculated measures were incorrect. The code from this approach can be found in the Annex .
Theses nodes can be launched as follows:

```
    roslaunch leddar_ros LIDAR_simple.launch
    rosrun lidar M1_listenerLIDAR_ros.launch
```

7.1.2. Method 2: Python

Method two was designed with simplicity and versatility in mind. For this reason, the code was fully implemented with Python. The code consisted of two nodes. The main node is in charge of reading the data directly from the sensor, processing it and publishing it. The secondary code was a test node to verify that all the information is being published correctly. The main advantage of this code was that all the reading, processing and publishing was being done in the same code. Also as the measures are being directly read from the sensor the point cloud format is not needed.
This method had several iterations in order to speed up the sampling rate. The method connects directly to the serial port where the LIDAR is connected and listen to the echoes of the pulses. Then the measures can be extracted from there. It creates a $LIDAR/talker$ topic that publishes the processed data. In order to fit the data format of the DDPG-OA the data is published under an array of 8 positions.
The raw data is read as follows.

```
[(7L, 0.8074188232421875, 285.79302978515625, 0L, 1)
 (6L, 0.4388885498046875, 378.6453552246094, 0L, 1)
 (5L, 0.7815399169921875, 523.3114013671875, 0L, 9)
 (4L, 0.8585205078125, 539.9019775390625, 0L, 9)
 (3L, 1.4276275634765625, 313.94097900390625, 0L, 1)
 (2L, 1.8220977783203125, 227.9708709716797, 0L, 1)
 (1L, 1.8505859375, 199.4412078857422, 0L, 1)
 (0L, 1.979827880859375, 152.3479766845703, 0L, 1)]
```

This data format despite presenting a similar structure with the Point Cloud is much more treatable as each row corresponds to the standard channel parameters of a LIDAR reading. The first column indicates the channel, the second one the measured distance, the third the amplitude and columns four and five are flags information.
Once the data is processed and published it presents the following format.

```
[INFO] [1619427051.126773]: layout:
  dim: []
  data_offset: 0
data:
  - 0.789352
  - 0.427505
  - 0.772247
  - 0.848328
  - 1.40912
  - 1.80414
  - 1.83302
  - 1.96361
```

The measures recorded during a period of 20s can be seen in figure 47. Each colour represents a channel of the LIDAR Vu8. Those ranges are a simple test sweep over the objects and walls of a room. This is the data that is feed to the DDPG-OA algorithm.

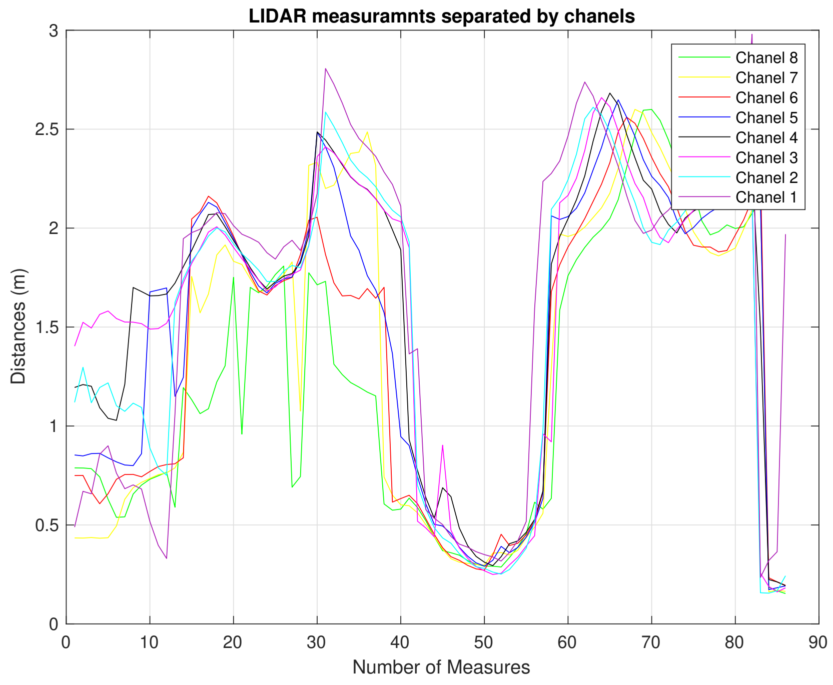


Figure 47: LIDAR measures separated by channels

In this approach, an additional node was also coded in order to verify that any other node subscribed to the topic $LIDAR/talker$ was able to read and correctly interpret the data. This listener node acts as a test bench to verify that the DDPG-OA node will be able of reading and understanding the data. It's important to notice that this code does not initiate $roscore$ on its own and need to be initiated previously to its launch. This node can be launched as follows:

```
roscore
rosrun lidar .................
rosrun lidar ReciverTestLidar.py
```

7.1.3. Considerations for particular cases

The main challenges when programming a decoder for the LIDAR are presented in the boundary cases. During the standard behaviour for the sensor the equipment is emitting the eight pulses of light and it's receiving back the eight echoes. In this scenario, the sensor behaves as expected publishing all the measures.
The main complexity appears when the LIDAR operates in an open field; then the following situations can occur:

- The LIDAR it's not receiving any echos
- The LIDAR has an obstructed field of view
- The LIDAR it's receiving just a fraction of the echos.

In this scenarios, the sensor does not publish the channels with the null measures. This means that the matrix with the raw data will change in shape and size. This must be corrected in order to feed a steady vector of measures to the DDPG-OA algorithm. The created algorithm analyzes each of the echoes matrices and determines which of them are missing some channels. The missing channels will be replaced with an infinite value, meaning that the beam of light is

not getting back.
As an example:

```
# In case of detecting a central object
LIDAR = []
# The null measure will  become
LIDAR = [inf, inf, 1.80414, 1.83302, 1.96361, inf, inf ]
#
# In case of flying without any obstacles insight
LIDAR = []
# The null measure will  become
LIDAR = [inf, inf, inf, inf, inf, inf, inf ]
```

The algorithm also ensures that all the measures are always in the same order and that all the channels in the LIDAR are properly organized in order to maintain consistency for the algorithm.

7.1.4. Modification of the DDPG-OA algorithm

The DDPG-OA has been modified in order to accommodate the new topic and its LIDAR measures. The main LIDAR related modifications in the code are presented below.
In this part of the code the data recollected from the subscribed topic is copied in the working vector for the main algorithm.

```
def LidarCallback(msg):
        global ranges, pitch, pitch_LIDAR
        ranges = []
        for i in range(np.size(msg.data)):
                ranges.append(msg.data[i])
        pitch_LIDAR = pitch
```

In this section, the subscribed topic and type of variable have been modified for the new LIDAR module.

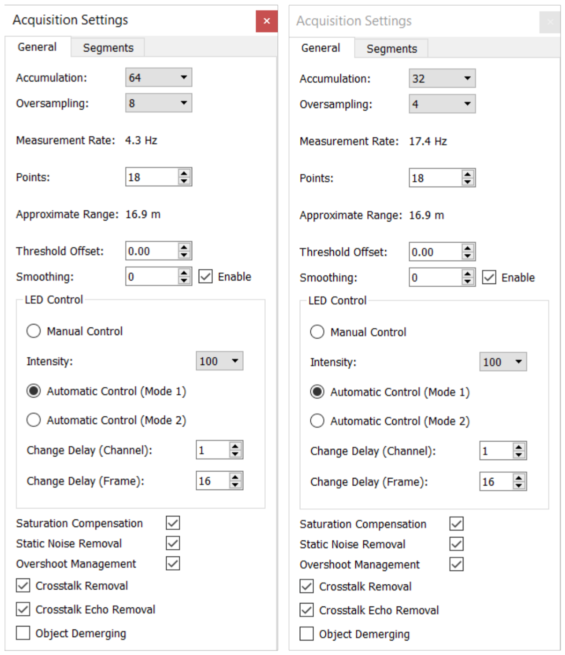
```
rospy.Subscriber('LIDAR/talker', Float64MultiArray, LidarCallback)
```

7.1.5. Configuration of the Leddartech Sensor

The sensor can have different configurations for more precise measurements, more field of view or faster measuring rates. For the current application, it has been concluded that a faster measuring rate was primordial over higher resolution. The LIDAR can be configured using the LeddarTM Configurator. There are two main parameters that will define the sampling rate of the sensor. Those are:

- Accumulations: Number of accumulations, higher values enhance the range and reduce the measurement rate and noise. Depending on the application, a reduction of the noise might be more important than a high measurement rate. Values can be 1, 2, 4, 8, 16, 32, 64, 128, 256, 512 & 1024.
- Oversampling: Number of oversampling cycles, higher values enhance the accuracy/precision/resolution and reduce the measurement rate. Depending on the application, a higher resolution might be more important than a high measurement rate. Values can be 1, 2, 4, 8, 16 & 32.
- Points: Number of base sample points. Determines the maximum detection range. The more points there are, the more they have an impact on the processing load since it impacts the number of sample points to process for each segment. Values range from 2 to 128

Figure 48 presents the default (right) and modified (left) configuration. The initial configuration has a Measurement rate of 4.3Hz; despite presenting high accuracy this measurement rate is quite low for a UAV LIDAR navigation system. For this reason, the Accumulation and Oversampling have been lowered in order to gain a higher rate. The final parameter was set to 17.4 Hz which is a good enough value for a real-time obstacle avoidance system on a drone. This modification was also done in order to match the Gazebo virtual LIDAR refresh rate with which the drone was trained.



Acquisition Settings
General | Segments

Accumulation: 64
Oversampling: 8
Measurement Rate: 4.3 Hz
Points: 18
Approximate Range: 16.9 m
Threshold Offset: 0.00
Smoothing: 0 ☑ Enable

LED Control
○ Manual Control
Intensity: 100
● Automatic Control (Mode 1)
○ Automatic Control (Mode 2)
Change Delay (Channel): 1
Change Delay (Frame): 16

Saturation Compensation ☑
Static Noise Removal ☑
Overshoot Management ☑
☑ Crosstalk Removal
☑ Crosstalk Echo Removal
☐ Object Demerging

Acquisition Settings
General | Segments

Accumulation: 32
Oversampling: 4
Measurement Rate: 17.4 Hz
Points: 18
Approximate Range: 16.9 m
Threshold Offset: 0.00
Smoothing: 0 ☑ Enable

LED Control
○ Manual Control
Intensity: 100
● Automatic Control (Mode 1)
○ Automatic Control (Mode 2)
Change Delay (Channel): 1
Change Delay (Frame): 16

Saturation Compensation ☑
Static Noise Removal ☑
Overshoot Management ☑
☑ Crosstalk Removal
☑ Crosstalk Echo Removal
☐ Object Demerging

Figure 48: Configuration of the Leddartech Sensor Using the proprietary Leddar Vu8 Software

7.2. LIDAR - Gazebo test 1

The LIDAR can be tested in the simulated environment. The DDPG-OA has been modified in order to read from the real LIDAR and not the simulated one. Different tests were conducted in order to verify that the behaviour was the expected one. This initial test was done in a balcony where infinite measures, ranges of 8m and very close measures (less than 1m) could be read orienting the sensor to different buildings.
The first test is marked in red. In this test, the LIDAR pointed upwards reading infinite/null measurements. As expected the drone followed a straight path until it collided with the simulated object.
The other tests (marked in blue) present different behaviours when the drone was subjected to various LIDAR non-null measurements. In test one, it can be seen that as the LIDAR is oriented to a surface at the last seconds the UAV starts to turn. It also collided as the LIDAR reading were not correlated with the simulated environment and were manually generated.
In test two and three the simulation started with the LIDAR pointing to an object. It can be seen that instead of following the straight path constantly moves to the side in order to avoid the object. As the LIDAR remained fixed pointing to an object the simulated UAV kept trying to avoid it.

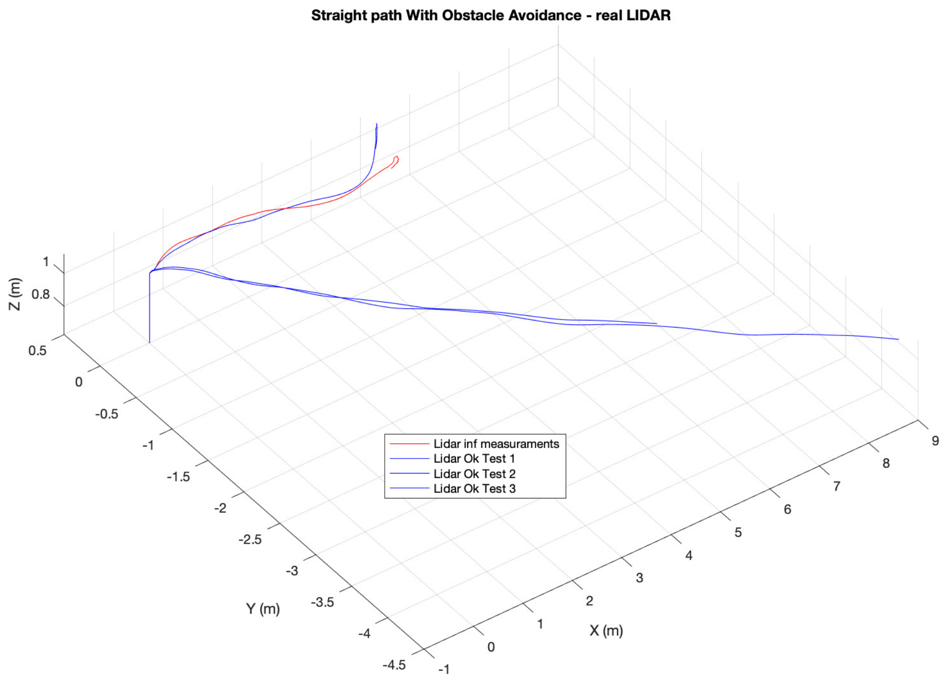


Figure 49: Test Run with Gazebo and the LIDAR

7.3. LIDAR - Gazebo test 2

In this test, the drone was tested in a simulated environment with feed from the real LIDAR again. The objective was again to simulate the detection of objects with the sensor in order to visualize in Gazebo how the Hummingbird will behave. The LIDAR was manually moved and pointed to different distant objects to simulate the apparition of objects in the path. The original trajectory is again a straight path. Figure 50 shows the dimensional trajectory of the UAV.

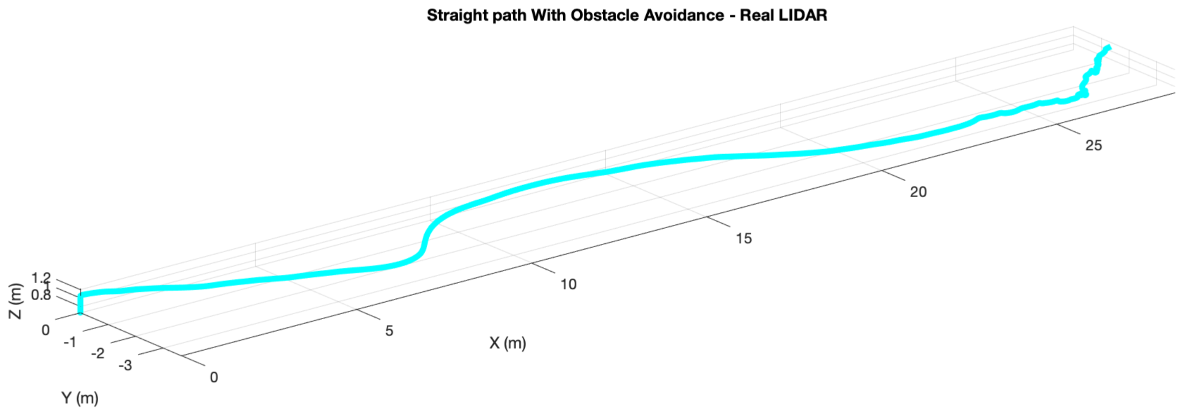


Figure 50: Vizualitzation of the LIDAR Channels

Some interesting conclusions can be drawn when the data is filtered a bit and compared. The plot in figure 50 presents X and Y position in relation to the LIDAR measurements. All the data has been time-correlated in order to be compared properly. This is necessary for more complex analyses as each system works and collects data at different frequencies. In this plot, the LIDAR measurements have also been simplified and just three channels are displayed. The first major conclusion that can be drawn is that once the UAV starts moving as it's detecting a near object it moves sideways. At around second 4 as channels 1 and 4 of the LIDAR are detecting an object the UAV starts its avoiding trajectory. At second 9 the UAV stops sensing any object and starts correcting its path back to the original y=0 straight path. Again at second 16, a new perturbation is detected by the LIDAR and the UAV begins another avoidance trajectory. Once the detected object is no longer visible the drone starts correcting back to its path at second 24. At that point, the UAV seems to struggle to correct its trajectory. This is mainly because the DDPG algorithm has reached its final programmed path position.

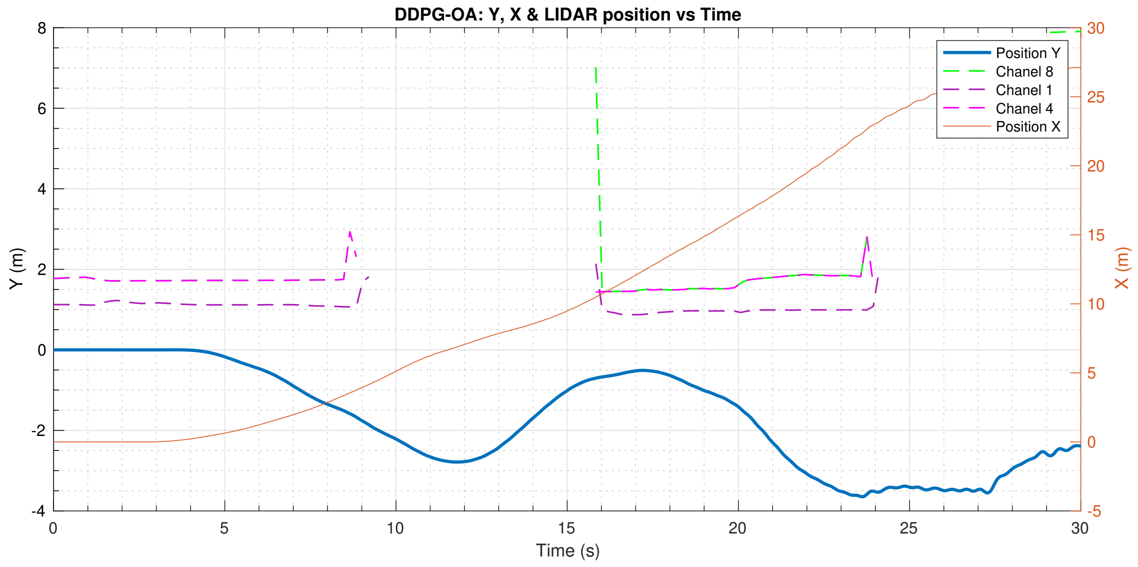


Figure 51: Visualization of the Position vs LIDAR Response

Figure 51 presents all the data from the real LIDAR, in this case, all the channels are visible. As it can be seen the overall data from all the channels are similar to the ones selected in the previous plot. The data has two big gaps that correspond to infinite measures. As the LIDAR was moved manually and those gaps correspond to the LIDAR being pointed to the sky. It's also interesting to notice that not all the channels present the same data. This is due to the fact that the reference surface used to simulate an object was not a flat plane and had some slopes.

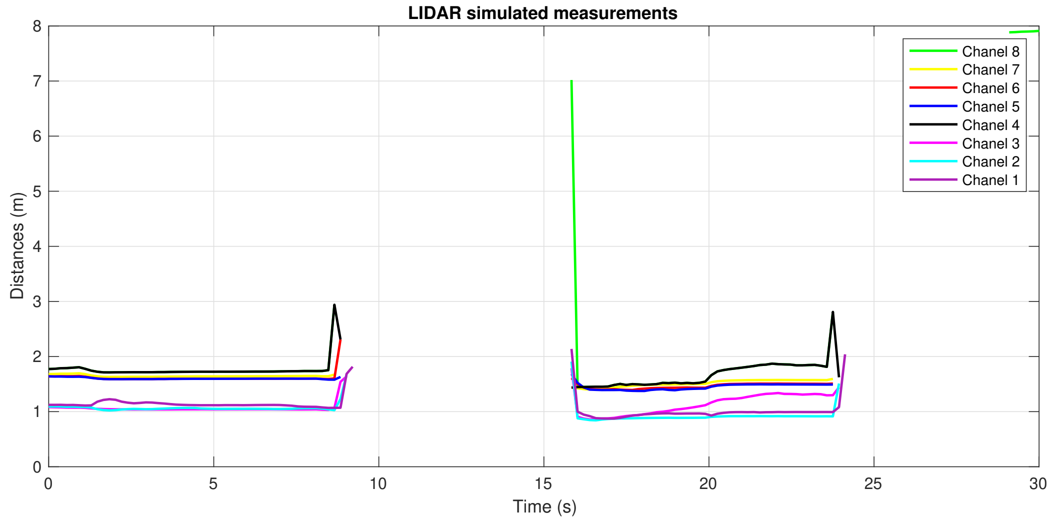


Figure 52: Vizualitzation of the LIDAR Channels

Finally the plots in Figure 80 presents the position data in the X and Y axis separately. The Y position plot is the same presented before in combination with the LIDAR measurements. The oscillations made for avoiding the object are clearly visible. The X position plot presents a steady slope meaning that the drone, despite actively modifying its path, was capable of maintaining a constant speed.

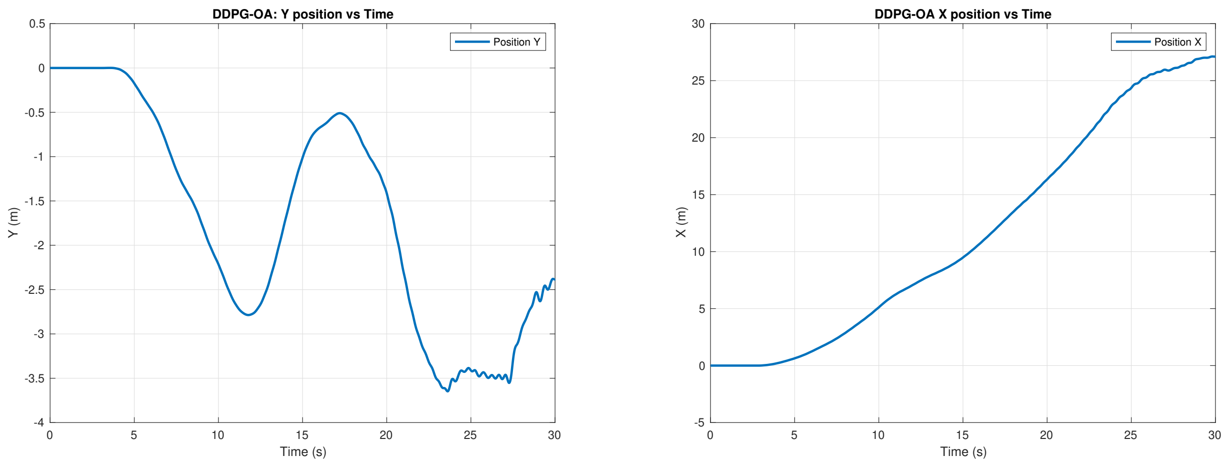


Figure 53: X and Y Position of the UAV during the Simulation

7.4. LIDAR - Gazebo test 3

In this test, the Leddartech sensor Vu8 has again been tested in tandem with the simulated drone environment in Gazebo. The objective was to perform a lemniscate path while the readings of the sensor were null and perform an avoidance manoeuvre when some surfaces were detected. Figure 54 presents the bidimensional path of the Hummingbird with a top view. As it can be appreciated the UAV has performed the proposed geometry without problems. It has also performed a small avoidance manoeuvre after crossing the mid-section point when returning to the origin.

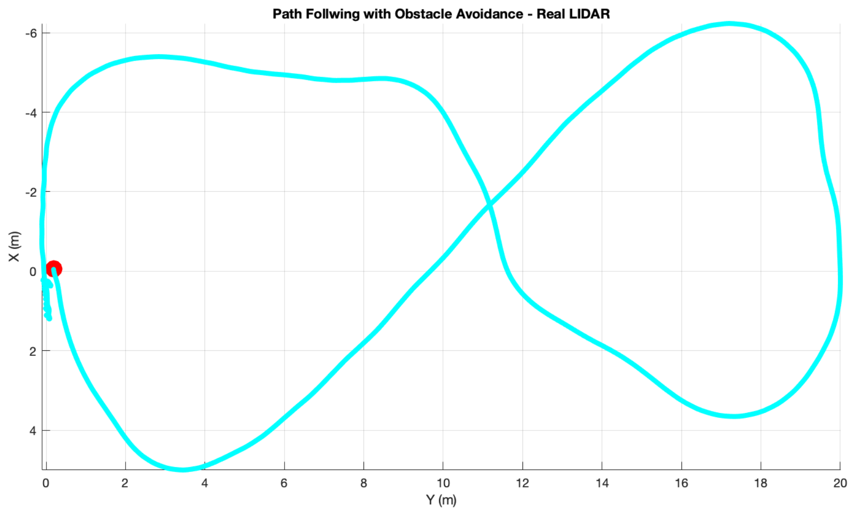


Figure 54: Lemniscate path with Obstacle Avoidance

Figure 55 shows the LIDAR measurements during the test. As in can be seen all the data is null as the LIDAR was being pointed to an "infinite surface" (from the sensor's point of view) except at second 40 where an object is introduced in the sensor's field of view for a few seconds.

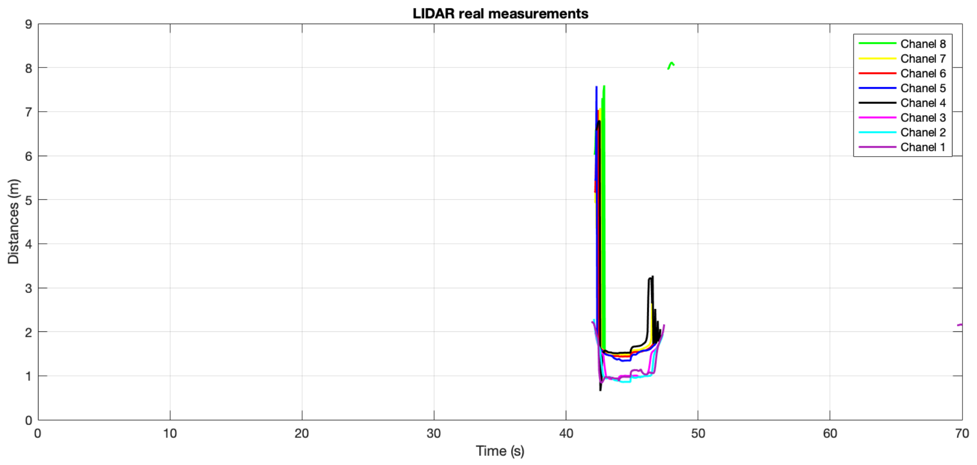


Figure 55: Visualization of the LIDAR Channels

Figure 56 presents the combination of LIDAR information with the X and Y coordinates. As it can be seen when the LIDAR detects the object the Hummingbird performs a small avoidance manoeuvre. It's more difficult to visualize due to the nature of the path but from seconds 42 to 48 the drone performs the obstacle avoidance. In the 3D path, the manoeuvre corresponds pat the midpoint when its returning to the upper left curve.

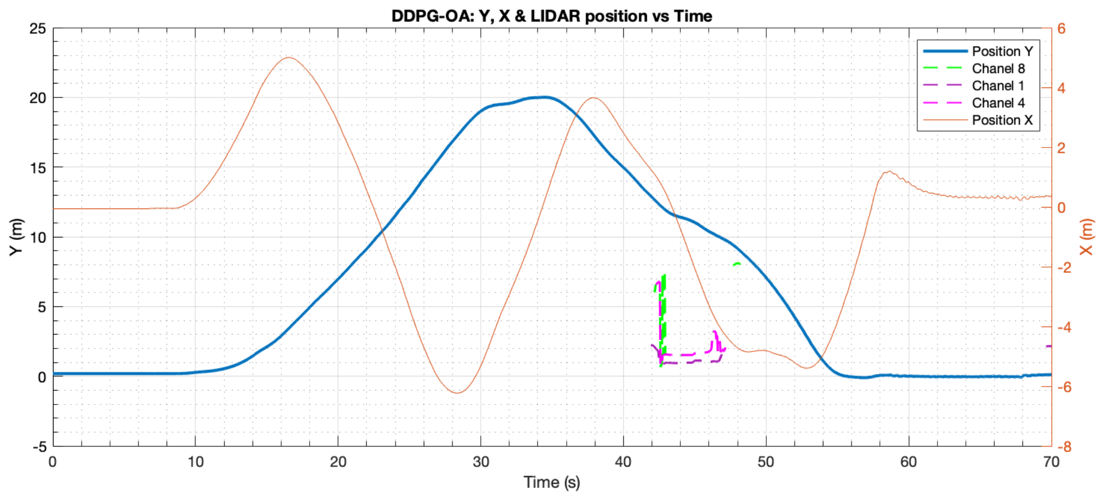


Figure 56: Position coordinates and LIDAR comparison

Figure 57 presents a new simulation performed with the virtual LIDAR and no objects in the field. This test can be used as a comparison for reference with the previous simulation. The Hummingbird struggled to gain control when it reaches the opposite end of the lemniscate but it was able to finally gain control and perform the path back to the starting point.

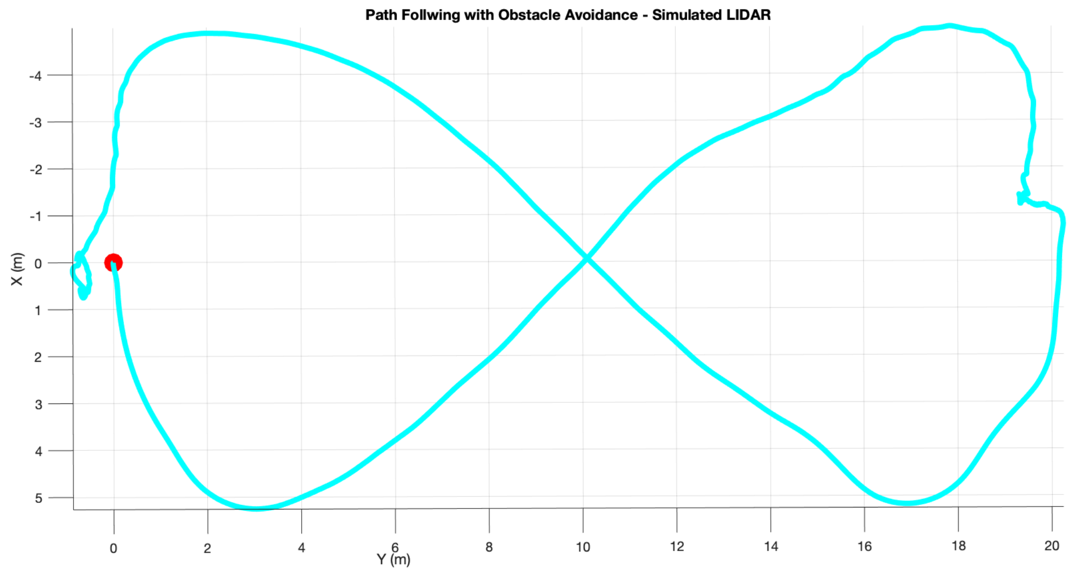


Figure 57: Lemniscate path with Obstacle avoidance, simulated LIDAR and no obstacles

Finally figure 58 presents the data from the virtual LIDAR. Again all these raw measures are from the ground detections. Apart from those measurements, the LIDAR does not detect any object as expected. This plot despite being very noisy will also serve as a reference for future comparisons.

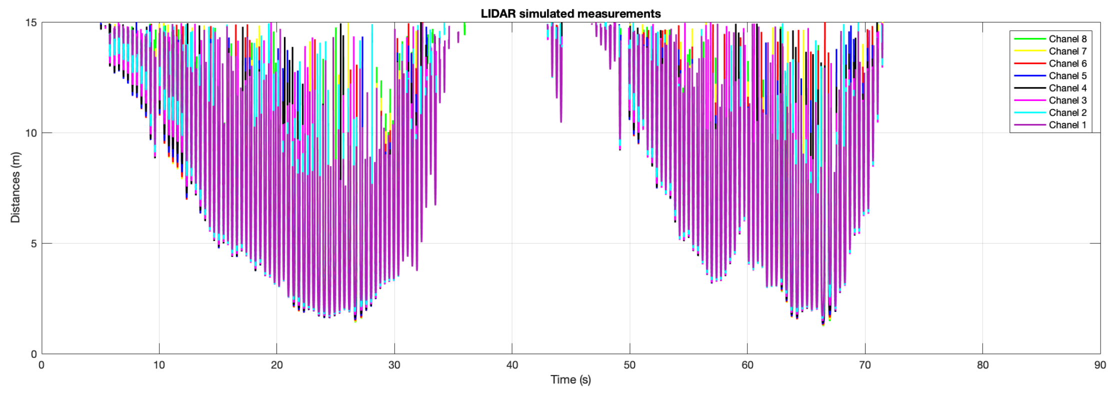


Figure 58: Virtual LIDAR for Obstacle avoidance and no obstacles

Figure 59 presents the vibrations recorded by the IMU in the Y-axis during the simulation. As it can be seen the correlation between figure 59 and 58 is very apparent. As the drone pitches up and down oscillating the ground detections become stronger. This in theory will not become a problem as the algorithm can filter those types of false detections.

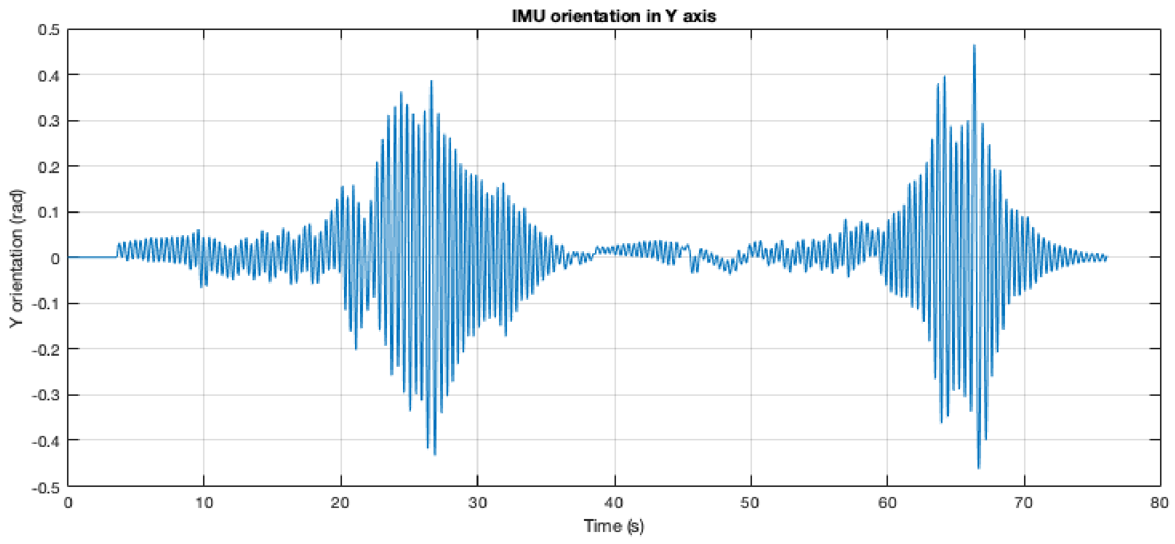


Figure 59: Vibration of the IMU in the Y axis

7.5. Gazebo Test Real LIDAR - SIMULATED

Unfortunately, the LeddarTech sensor stopped working just when the final experiments were ready to start. Just some simple flight were done with the LIDAR equipped to the drone. In order to be able to perform those without the LIDAR, a code was developed that, using all the previous build interfaces, simulates the detection of objects by the real sensor in real flight simulations. In order to test the implementation of the additional code, some more simulations were performed in Gazebo. The Hummingbird was programmed to follow a straight path and when the sensor "detected" an object it performs an avoiding manoeuvre.
Figures 60, 61, 62 and 63 present the bidimensional path from a top view of the different test. As it can be appreciated once the LIDAR detection is introduced to the system the Hummingbird deviates sideways to avoid the object. This method does not allow for a complete test of the system using the Leddartech sensor but now that the sensor is not available it provides a method to verify that when the sensor is installed it will also work on a real test scenario.

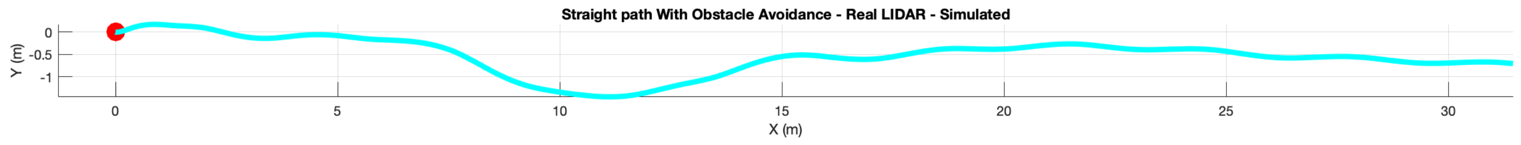


Figure 60: Tridimensional Path in Test 1 using the Real Simulated Sensor

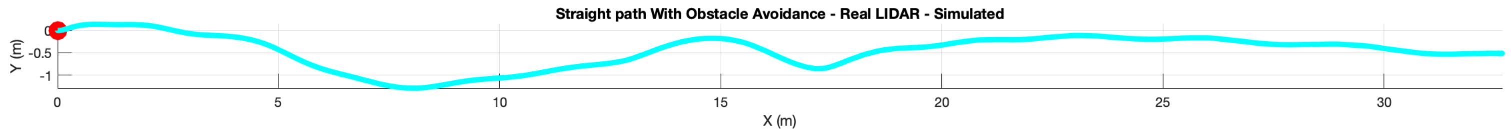


Figure 61: Tridimensional Path in Test 2 using the Real Simulated Sensor

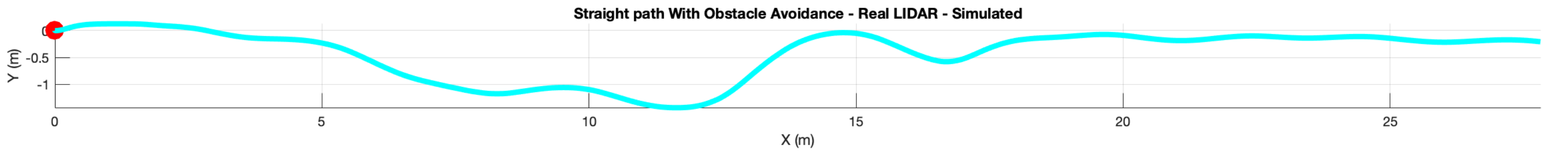


Figure 62: Tridimensional Path in Test 3 using the Real Simulated Sensor

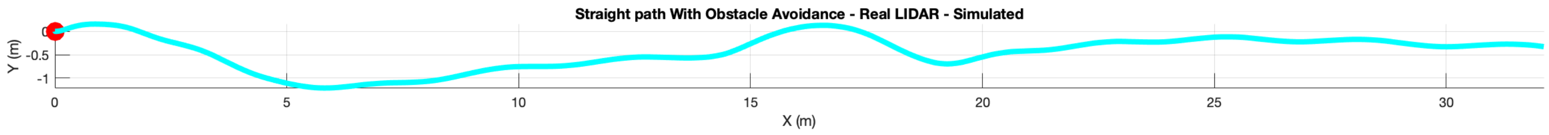


Figure 63: Tridimensional Path in Test 4 using the Real Simulated Sensor

The following Figures 64, 65, 66 and 67 present a correlation between the X and Y movement coordinates and the feed LIDAR detections to the algorithm. As it can be appreciated the algorithm is capable of maintaining consistency through all the tests. Once the system detects the new input from the LIDAR is moves sideways. There is a small delay between the detection and the movement but this was expected. In this test different simulated LIDAR pulses were tested to see which one resemble more the actual detections made with the Leddartech sensor and also to see which one better worked with the Obstacle avoidance algorithm. All the pulses worked as expected but the longer and continuous pulses seem to produce a more visible avoidance manoeuvre by the algorithm. In these tests, the pulses were time triggered. In the version of this algorithm that will run on the Hummingbird the pulses fed to the algorithm are activated using a switch of the RC controlled. This allows for more rigorous testing and the simulated detection can be introduced to the system when the flight conditions are adequate or the drone is flying over a particular region. This emulates more purely the real tests that would have been performed placing an object in the flight arena for the Hummingbird to detect. As the sensor is no longer functioning the detection will be introduced by the user to test that all the system works as expected.

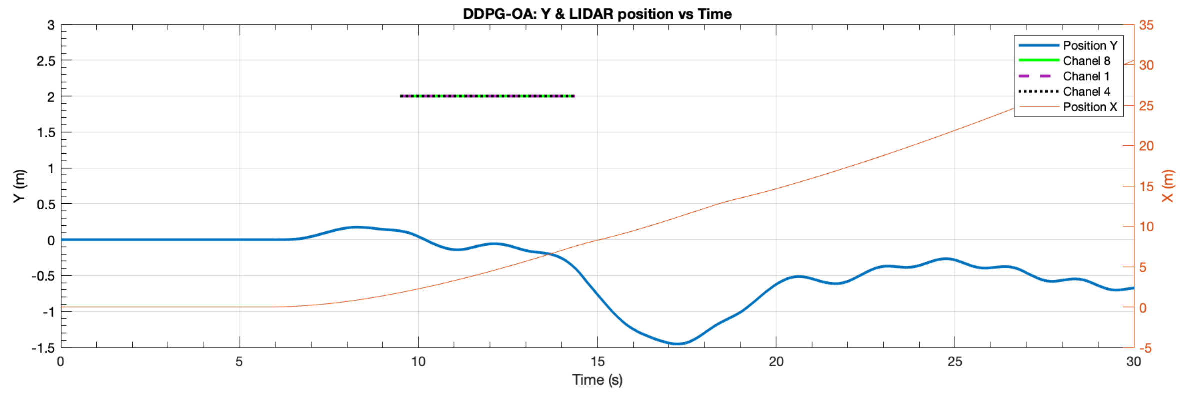


Figure 64: Position and LIDAR detections correlation in Test 1

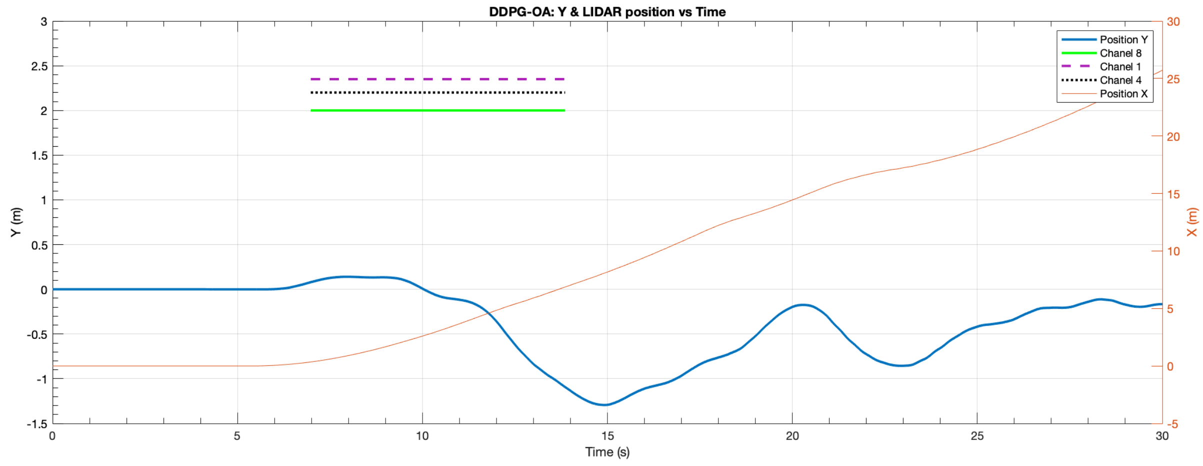


Figure 65: Position and LIDAR detections correlation in Test 2

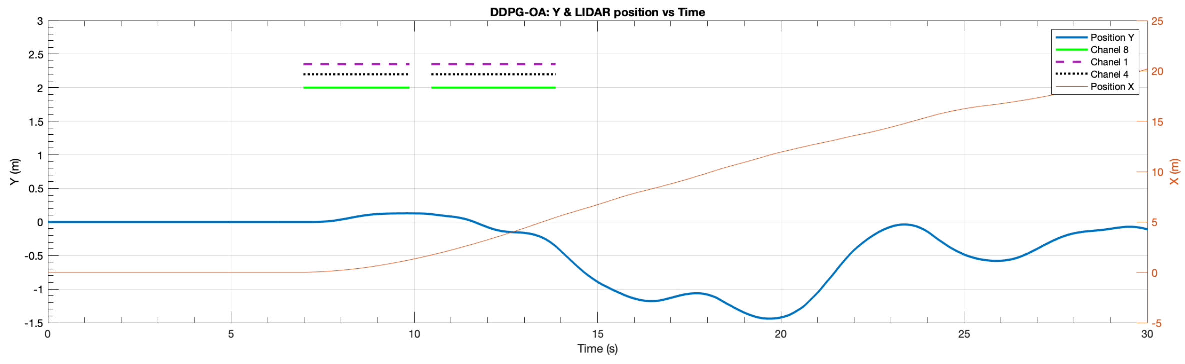


Figure 66: Position and LIDAR detections correlation in Test 3

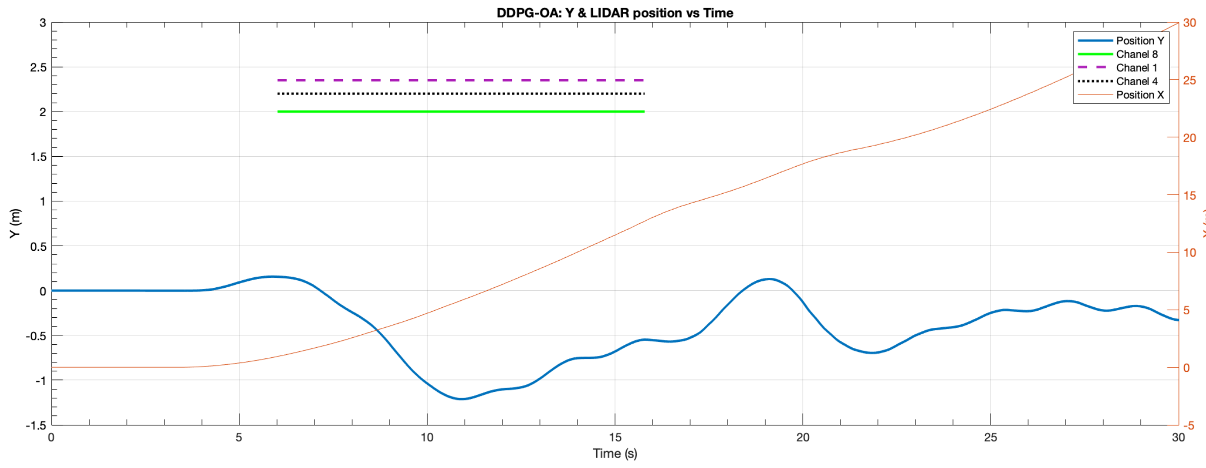


Figure 67: Position and LIDAR detections correlation in Test 4

Summing up this methodology will allow for the verification of the overall system during the test phase. This method ensures that once the main sensor is ready again the UAV will work perfectly with it and the same test could be recreate with even more precision as the detections will be done properly by the real sensor.

7.5.1. Modified Central channels detections with simulated real LIDAR

This tests were done using the same simulated real LIDAR. The main difference was that instead of all the channels receiving a measure just the central channels of the sensor detect the "object". The obtained manoeuvre as can be seen in figure 68is much more accentuated as the algorithm sees the detection much closer. If the wide field of view of the sensor detects an object in all the channels means that using the training data that the object is still far away. A more narrow detection using only the central channels means that the abject (cylindrical obstacle) is closer.

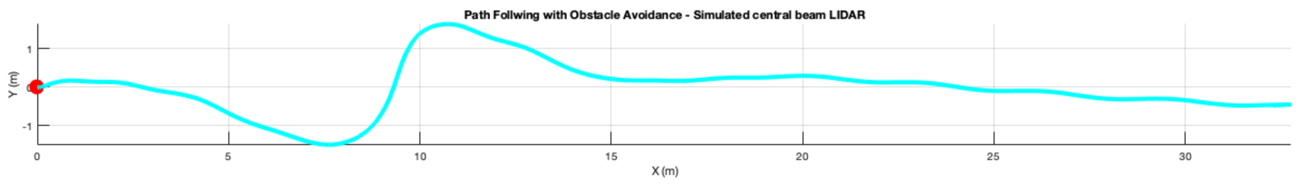


Figure 68: Tridimensional Path in Test 5 using the modified Real simulated Sensor

Figure 69 presents the X and Y position with the central LIDAR detections. Once the programmed detections are seen by the algorithm the UAV turns strongly. The recuperation of the initial track once the detections are no longer present is also much more accentuated. The curvature for recuperating the initial path is more notable than the previous test.

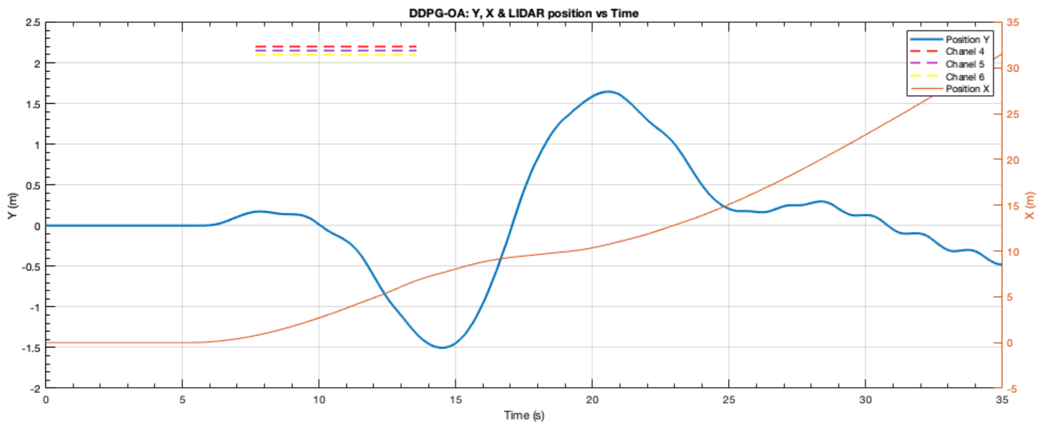


Figure 69: Position and LIDAR detections correlation in Test 5

In conclusion, the developed method could be used as a workaround to test that all the implemented code and the obstacle avoidance modified algorithm could also work in real test flight conditions despite not having the sensor for testing with it.

8. Hummingbird Platform

8.1. General Configuration

The Hummingbird quadcopter is a complex platform with several electronics on board. The main elements are described below.

Sensors: The AscTec 3D-MAG is a triple-axial compass module used to determine the vehicles heading by measuring the Earth's magnetic field. The GNSS receiver is a taoglas CGGP which is GPS-GLONASS-GALILEO compatible and can perform Dual-Band reading with its patch antenna.

Actuators: The X-BL-52s motors by HACKER Motors Germany are custom-built for the AscTec Hummingbird. They are controlled by X-BLDC brushless motor controllers highly optimized for the X-BL-52s motors.

Asctec AutoPilot Board: The AutoPilot Board has two processors low and high-level processor. The low-level processor receives signals from the sensors (except the GPS) and sends commands to the motor controllers. It provides an estimation of the attitude angles of the drone. This part of the control board it's not accessible by the user but some of its functionality can be accessed through the high-level processor. The high-level processor is in charge of the GNSS receiver and its connection with the low-level processor. For this configuration in use, it will only be used as a bridge between the attitude control board and the Odroid computer.

Odroid-XU4 The Odroid-XU4 is a new generation of computing device with more powerful, more energy-efficient hardware and a smaller form factor (83 x 58 x 20 mm approx, excluding cooler). Offering open source support, the board can run various types of Linux, including Ubuntu 16.04 and Android 4.4 KitKat, 5.0 Lollipop and 7.1 Nougat. It features an octa-core Exynos 5422 big.LITTLE processor, advanced Mali GPU (ARM Mali-T628 6 Core), 3GB of RAM and Gigabit ethernet [12].
This computer is equipped with Ubuntu 16.04 LTS and with ROS Kinetic installer together with the AscTec packages it can communicate with the autopilot of the quadcopter.

8.1.1. Connections

Power distribution
The system is powered using the interchangeable Li-Po battery installed in the Hummingbird. This battery powers the ESC and Motor directly with 11.1V. The onboard flight controller has its own voltage regulator that transforms the oscillating battery voltage (from 12.6 to 10.8). Another independent voltage regulator also outputs 5V to the Odroid from the battery. Finally, the third independent voltage regulator outputs 12V to the LeddarTech sensor on top of the drone (see figure 70). This voltage regulator was a custom build and installed for this project. This model was chosen for it's lightweight and easy to install. Its small factor also makes it easy to hide it inside the electronics compartment where it's protected from the propellers.

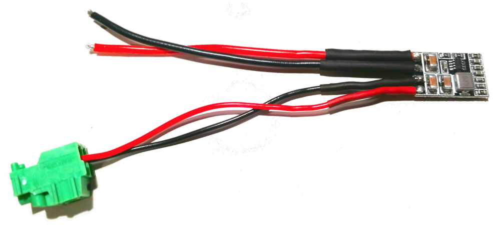


Figure 70: Voltage Regulator set up for the LIDAR

Power Testing

Before doing any flight tests the voltage regulator combined with the LIDAR were tested in order to verify that the power distribution and voltage were functioning properly. The main inconvenience with the selected approach is that the battery fully charged provide 12.6 V and fully discharged 10.6 V. Once the battery voltage droops below 12V the voltage regulator can not do anything. A small test was performed to see how the LIDAR performed underpowered. The battery started fully charged and was progressively discharged until the LIDAR stopped working. Unexpectedly the LIDAR was able to work well beyond the normal operational margin. It worked fine until a voltage of 11.0 was reached.
This information is essential as when flying with the LIDAR the experiment must be stopped before reaching that lower limit. Once the LIDAR stops working the algorithm detects infinite measures and can not detect any objects.

8.1.2. Connections Diagram

The power distribution from the UAV is quite direct as everything is powered by the battery. There are three main power rails. One for the ESC, motors and flight controllers, one for the Odroid and one for the LIDAR. If needed the LIDAR can be disconnected from its 12V bus and powered externally as the power port is accessible. There is also a switch that can turn on and off the flight controller power bus in order to only power the Odroid.

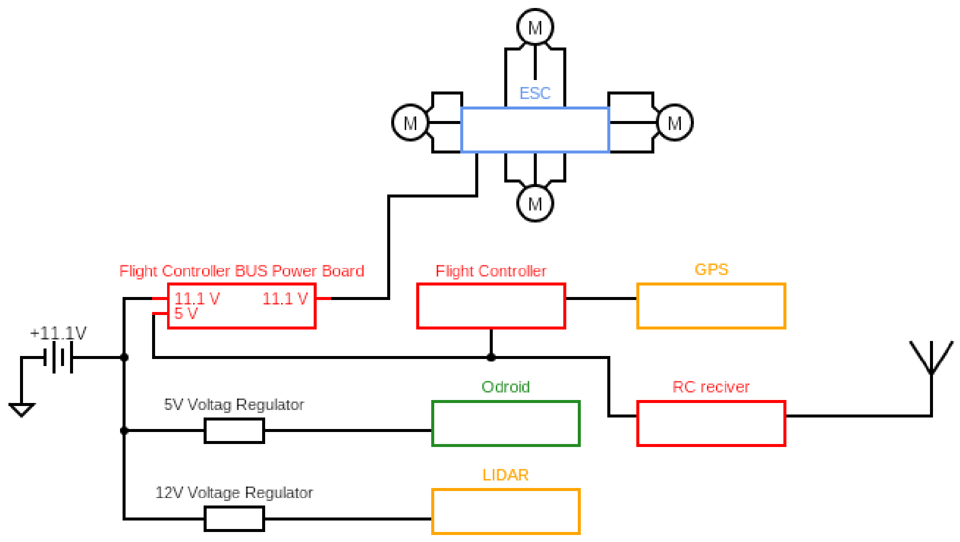


Figure 71: Diagram connections of the different power busses along the UAV

8.2. Ubunto - ROS - Hummingbird Configuration

In this section, the organization with the different platforms and programs will be explained as well as the configuration of the main relevant parts and their relation.

8.2.1. Software distribution

Figure 72 presents how the software elements are related to each other. In order to fly the drone has three computers working at the same time: The ground computer, the flight controller and the Odroid. On the presented diagram the organization of these computers and cross-communication is presented.
The flight controller is divided into two processor boards the high level and the low level. The low-level processor is in

charge of talking with all the peripherals of the system such as ESC, radios, sensors... The high-level processor (HLP) talks with this board in order to gain access to that information and perform navigation tasks. The HLP also is in contact with the Odroid where all the main algorithms are being run. The Odroid is also connected via WIFI with the external computer in order to process the DDPG algorithms with TensorFlow.

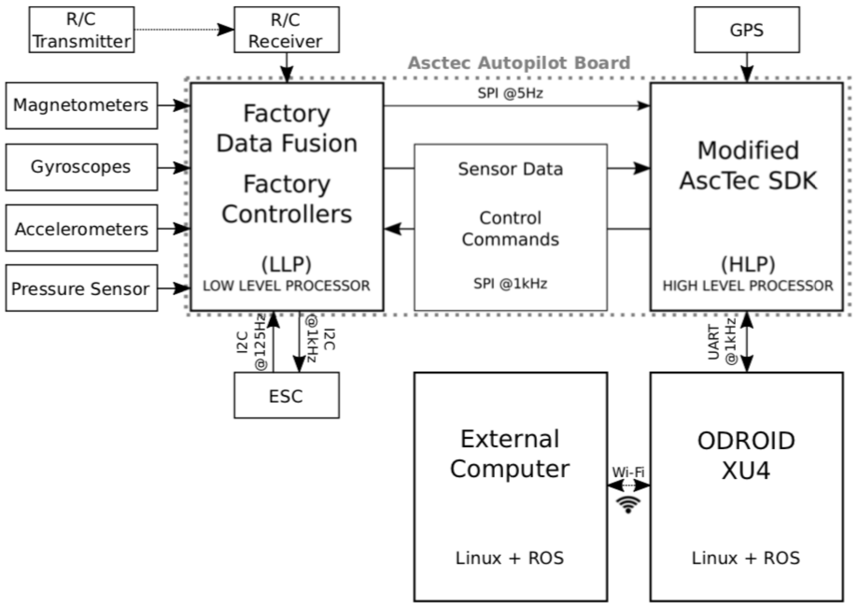


Figure 72: Scheme of the elements of the experimental platform [11]

8.2.2. Nodes and Topic Organisation

The following figure 73 presents all the active nodes during the flight. All the main nodes are organized around the Flight Controller Unit that publishes all the information about the state of the drone. There are also "peripheral" nodes, those are the GPS processing nodes and LIDAR. This image is very useful to understand the relation between the simultaneous running process. As an example the DDPG Path Following on the left it can be seen that feeds information to the attitude controller node and velocity controller node and its receiving information from the GPS node, DDPG Obstacle Avoidance (that itself is receiving data from the LIDAR),...
The DDPG-OA is receiving data directly from the GPS node, the LIDAR, and the Flight controller. It processes the information and publishes it. This information is received by the DDPG Path Following algorithm that as mentioned will modify the attitude and control of the drone.
This diagram allows the understanding and correlation of the different process that the drone is calculating.

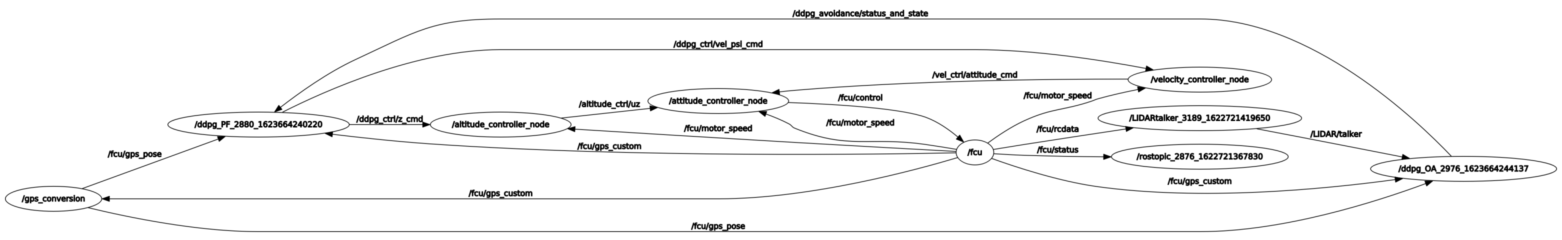


Figure 73: Scheme of the elements of the experimental platform

8.2.3. Connecting to the Onboard computer

For configuration and working with the onboard computer, the next procedure must be followed.
Once the Odroid has power it will automatically generate a WiFi network which will be used for the communication between the ground station and the Odroid. In order to connect to the Odroid, the following procedure must be done. This establishes an SSH protocol with the computer and allows the user to directly control the remote computer via a terminal

```
ssh odroid@10.0.0.200
password:
>>>> odroid@odroid:~\$
# For shutting down
sudo shutdown -P now
```

Each new terminal must run these commands in order to properly identify the ROS master and it's working directories. Those lines must also be executed in the ground station working terminals.

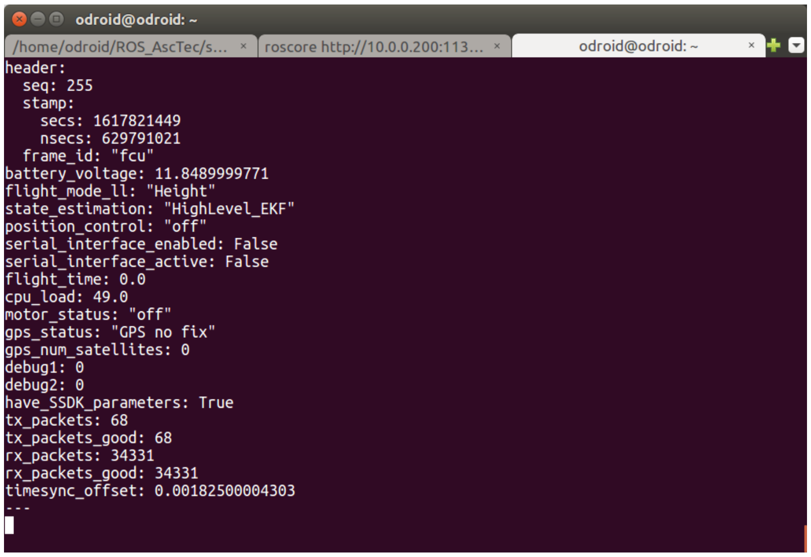
```
source /opt/ros/kinetic/setup.bash
source devel/setup.bash
```

8.2.4. Connection to the Flight Controller

Once a successful connection with the Odroid has been established the next step is to connect to the flight controller. It's important to notice that the board must be powered at the same time as the Odroid; otherwise, the computer will struggle to find it.
The communication link will be established using ROS and the AscTec packages.

```
roscore
roslaunch asctec_hl_interface fcu.launch
rostopic echo -c /fcu/status
```

This set-up will display the status and basic information of the flight controller. It will be essential for monitoring its well being. The topic fcu-status will display the following information (see figure 74). The parameters to keep an eye on are the battery voltage and the GPS status and the number of satellites. The battery must not reach 10.2 and the satellites must be in GPS FIX condition (above 5 satellites).


odroid@odroid: ~
/home/odroid/ROS_AscTec/s...
roscore http://10.0.0.200:113...
odroid@odroid: ~

```
header:
  seq: 255
  stamp:
    secs: 1617821449
    nsecs: 629791021
  frame_id: "fcu"
battery_voltage: 11.8489999771
flight_mode_ll: "Height"
state_estimation: "HighLevel_EKF"
position_control: "off"
serial_interface_enabled: False
serial_interface_active: False
flight_time: 0.0
cpu_load: 49.0
motor_status: "off"
gps_status: "GPS no fix"
gps_num_satellites: 0
debug1: 0
debug2: 0
have_SSDK_parameters: True
tx_packets: 68
tx_packets_good: 68
rx_packets: 34331
rx_packets_good: 34331
timesync_offset: 0.00182500004303
---
```

Figure 74: Elements displayed for the Flight Controller's topic

8.3. Hummingbird Pre-flight Preparation

In order to properly and safely fly the hummingbird different steps have been performed. The first ones are regarding the preparation of all the UAV hardware and batteries. The batteries must be fully charged in order to provide voltage margin for the LIDAR to work properly as mentioned in the previous section.
The landing dampening system must be also installed as it will be essential for avoiding hard landings or tipping the UAV. Even more, when the Odroid is under the UAV and the LIDAR is on top. The legs and balls must be fully extended in order to provide proper mitigation.
The motors and propellers must be checked before the flight to ensure that there are no obstruction and the motors can spin freely. Once the Hummingbird has taken off it can be flown in three modes. Knowing how to operate each of them will help to safely transition from manual to autonomous navigation mode. Those are:

- Manual Mode:
  This flight mode allows the user to have direct control of the 3 rotation axis of the drone throttle. Each movement of the joystick will correspond to a direct rotation of the drone.

- Height Control Mode:
  In this mode, the drone will use the GPS and barometer to maintain altitude. The user will modify the altitude using the throttle stick. The pitch, roll and yaw movements will also be controlled by the flight controller and a proportional correlation of moment between the sticks and the drone will be established.

- GPS Mode:
  In this mode, the drone will maintain height and also position. The joysticks will move the drone on each axis. As in the previous mode they don't control rotation.

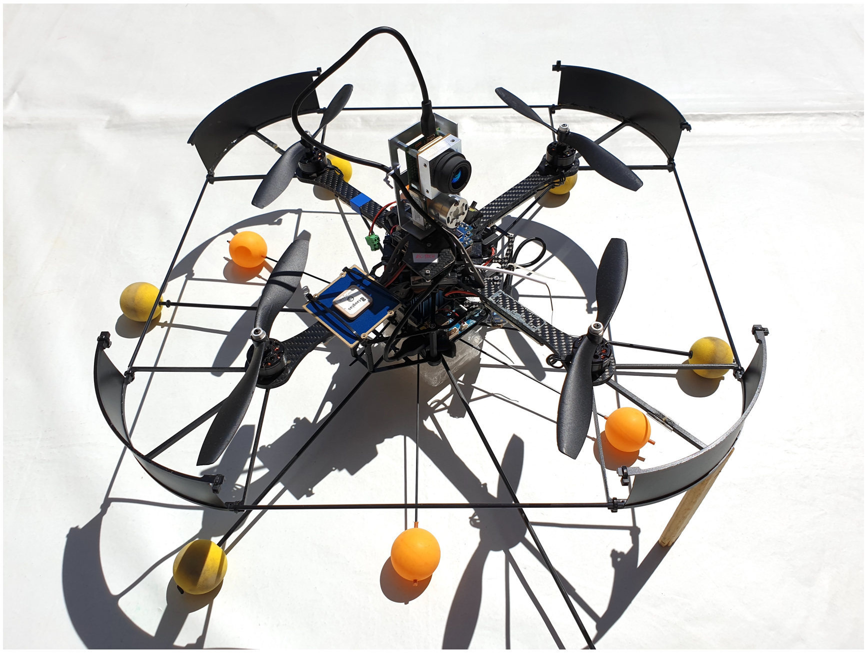


Figure 75: Hummingbird ready for take-off

In order to launch any program, the Hummingbird will need to be airborne and stable. The pilot will activate from the RC controller the automatic navigation mode. In case of abort, it can switch back to manual mode.

8.3.1. ROS Flight Preparation

Once the UAV is ready, the procedures for launching the Hummingbird has to follow the next steps:

- Give power to the Odroid (main computer), connect the flight controller and connect its USB to the Odroid. A WIFI will be created and the ground computer must be connected to it.
- Using new terminals connect to the $SSHodroid@10.0.0.200$.
- Once all the needed terminals are ready the following ROS nodes must be initiated on the Hummingbird's Odroid.
  - `roslaunch asctec_hl_interface fcu.launch`

    This will initiate the ROS node that will make up and monitor the flight controller. It will also publish vital information regarding its status. They must be run in the same terminal and sequentially.
  - `rostopic echo -c /fcu/status`

    This will subscribe and present the information published by the flight controller node
  - `rosrun asctec_hl_gps set_gps_reference_node`
    `rosrun asctec_hl_gps gps_conversion_node`

    These nodes will determine the initial flight position of the Hummingbird and converted it to X,Y and Z coordinates.
  - `roslaunch path_following_control ddpg_control.launch`

    This node launches the autopilot which will talk with the DDPG nodes running in the ground station.
  - `rosrun lidarLISTENERfastV3.py`

This node launches the LIDAR and prompts the sensor to start measuring and publishing data.

- On the ground computer using separated terminals the following commands must be executed
  - `python3 ddpg_PF.py`
    `python3 ddpg_OA.py`

    This will initiate DDPG-OA nodes that use TensorFlow and have to be processed in the ground station remotely.
- Once all those commands are properly introduced, the system displays no error and has GPS FIX the hummingbird can be manually taken off. Once the drone is hovering over the initial point using the RC controller the flight modes can be switched from manual to autonomous using the potentiometer. This will initiate the DDPG algorithms and the UAV will start following the programmed path.
- Once the Hummingbird has finished the desired path the pilot must switch back to manual and land the aircraft.

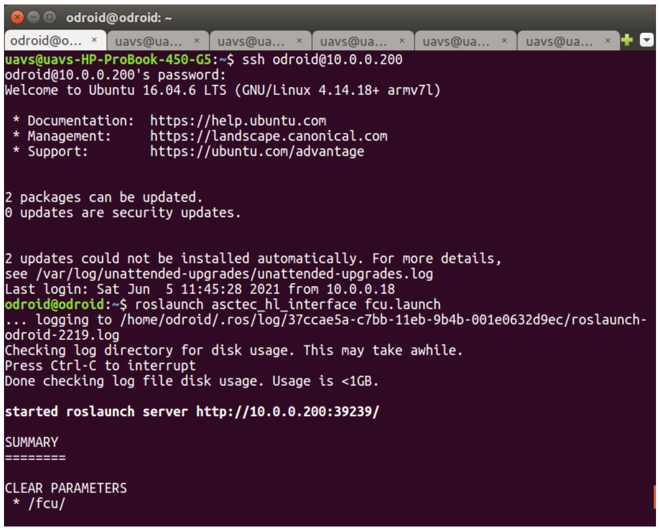
8.3.2. Check-List For Flight

This section presents a checklist for all required elements before performing a flight.

- Verify that the Flight controller is Turned Off and the USB is disconnected from the Odroid.
- Verify that the LIDAR is properly connected and the polarity of the cables from voltage regulators are connected in their respective terminals on the back of the LIDAR.
- Verify that all the propellers spin and there are no obstructions.
- Put the battery on the compartment under the Hummingbird and connect the power cables. The light from the Odroid should turn on.
- With the UAV stable turn on the Flight Controller and wait for the beep. This means that the IMU has been calibrated.
- Connect the Flight controller to the Odroid using the USB cable.
- Connect the ground satiation computer to the WiFi created by the Odroid and using several new terminals connect to the onboard computer following the steps described in the previous sections.
- Launch the Flight Controller Unit (FCU) nodes and subscribe to the respective topics. Verify that the battery level is not higher than 12.6V and no lower than 11.1V. Check also for the GPS indicator, the Unmanned Aerial Vehicle (UAV) must have GPS fix with at least 6 satellites.
- Turn on the remote controller, it must be in model 2. Verify that it has an adequate voltage (11.4V to 10.1 V).
- The potentiometer on the RC controller must be turned clockwise fully. The top left vertical two-position switch must be also turned up. The red left three positions vertical switch must be completely down for manual mode.
- Position the Hummingbird on the take-off pad.
- Launch the corresponding nodes for the Odroid and for the ROS on the Ground Station PC described in the previous section.
- For arming the vehicle position the throttle joystick down and yaw right. The motors will start spinning.
- Take off and hover over the initial position and turn counterclockwise the potentiometer on the RC controller in order to start the experiment.

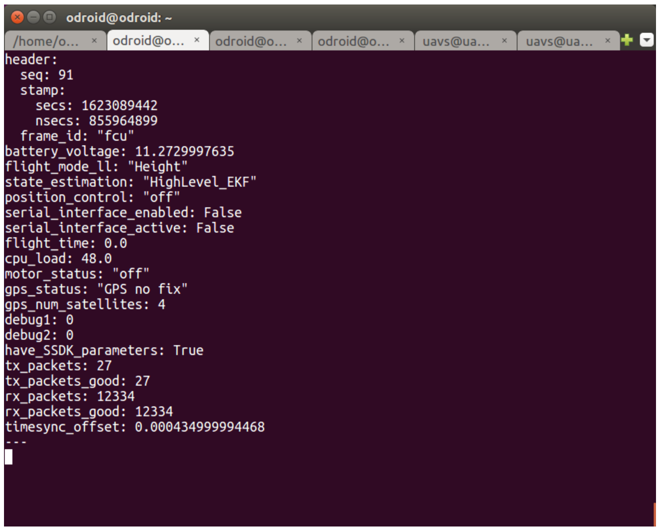
- Once finished, turn the potentiometer fully clockwise to land in manual mode
- Before disconnecting the battery shut down the Odroid from any terminal on the ground computer.

The following images detail the mentioned steps performed on the ground computer. Each image represents the different terminals that have to be launched in order to execute the programs on the Hummingbird and in the ground computer. The followed steps are the ones presented before where the ground computer is connected to the Odroid and the different ROS onboard nodes are launched.



```
uavs@uavs-HP-ProBook-450-G5:~$ ssh odroid@10.0.0.200
odroid@10.0.0.200's password:
Welcome to Ubuntu 16.04.6 LTS (GNU/Linux 4.14.18+ armv7l)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage


2 packages can be updated.
0 updates are security updates.


2 updates could not be installed automatically. For more details,
see /var/log/unattended-upgrades/unattended-upgrades.log
Last login: Sat Jun  5 11:45:28 2021 from 10.0.0.18
odroid@odroid:~$ roslaunch asctec_hl_interface fcu.launch
... logging to /home/odroid/.ros/log/37ccae5a-c7bb-11eb-9b4b-001e0632d9ec/roslaunch-
odroid-2219.log
Checking log directory for disk usage. This may take awhile.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://10.0.0.200:39239/

SUMMARY
========

CLEAR PARAMETERS
 * /fcu/
```

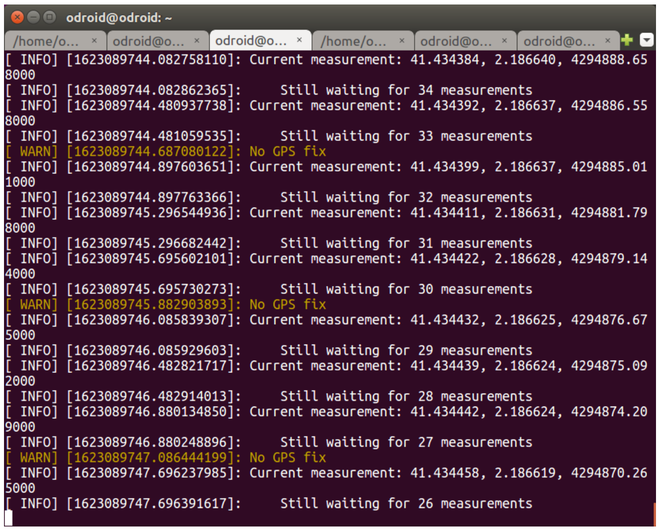
(a) AscTec Flight Controller launcher



```
header:
  seq: 91
  stamp:
    secs: 1623089442
    nsecs: 855964899
  frame_id: "fcu"
battery_voltage: 11.2729997635
flight_mode_ll: "Height"
state_estimation: "HighLevel_EKF"
position_control: "off"
serial_interface_enabled: False
serial_interface_active: False
flight_time: 0.0
cpu_load: 48.0
motor_status: "off"
gps_status: "GPS no fix"
gps_num_satellites: 4
debug1: 0
debug2: 0
have_SSDK_parameters: True
tx_packets: 27
tx_packets_good: 27
rx_packets: 12334
rx_packets_good: 12334
timesync_offset: 0.000434999994468
---
```

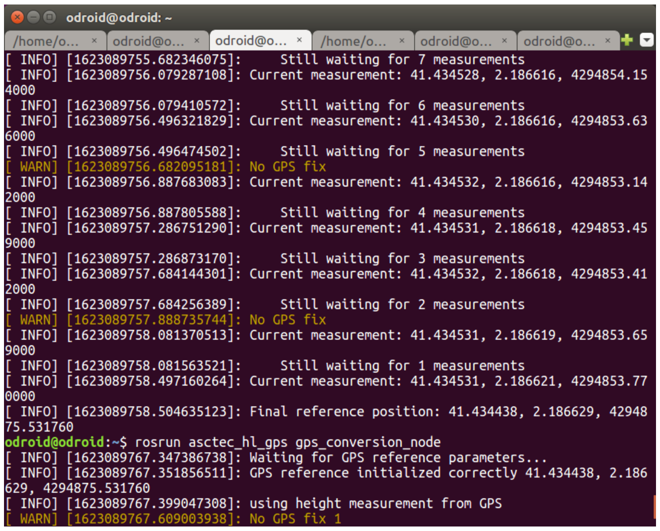
(b) Flight Controller Status Topic

Figure 76: Steps 1 & 2 of the ground station Set-Up



```
[ INFO] [1623089744.082758110]: Current measurement: 41.434384, 2.186640, 4294888.65
8000
[ INFO] [1623089744.082862365]:     Still waiting for 34 measurements
[ INFO] [1623089744.480937738]: Current measurement: 41.434392, 2.186637, 4294886.55
8000
[ INFO] [1623089744.481059535]:     Still waiting for 33 measurements
[ WARN] [1623089744.687080122]: No GPS fix
[ INFO] [1623089744.897603651]: Current measurement: 41.434399, 2.186637, 4294885.01
1000
[ INFO] [1623089744.897763366]:     Still waiting for 32 measurements
[ INFO] [1623089745.296544936]: Current measurement: 41.434411, 2.186631, 4294881.79
8000
[ INFO] [1623089745.296682442]:     Still waiting for 31 measurements
[ INFO] [1623089745.695602101]: Current measurement: 41.434422, 2.186628, 4294879.14
4000
[ INFO] [1623089745.695730273]:     Still waiting for 30 measurements
[ WARN] [1623089745.882903893]: No GPS fix
[ INFO] [1623089746.085839307]: Current measurement: 41.434432, 2.186625, 4294876.67
5000
[ INFO] [1623089746.085929603]:     Still waiting for 29 measurements
[ INFO] [1623089746.482821717]: Current measurement: 41.434439, 2.186624, 4294875.09
2000
[ INFO] [1623089746.482914013]:     Still waiting for 28 measurements
[ INFO] [1623089746.880134850]: Current measurement: 41.434442, 2.186624, 4294874.20
9000
[ INFO] [1623089746.880248896]:     Still waiting for 27 measurements
[ WARN] [1623089747.086444199]: No GPS fix
[ INFO] [1623089747.696237985]: Current measurement: 41.434458, 2.186619, 4294870.26
5000
[ INFO] [1623089747.696391617]:     Still waiting for 26 measurements
```

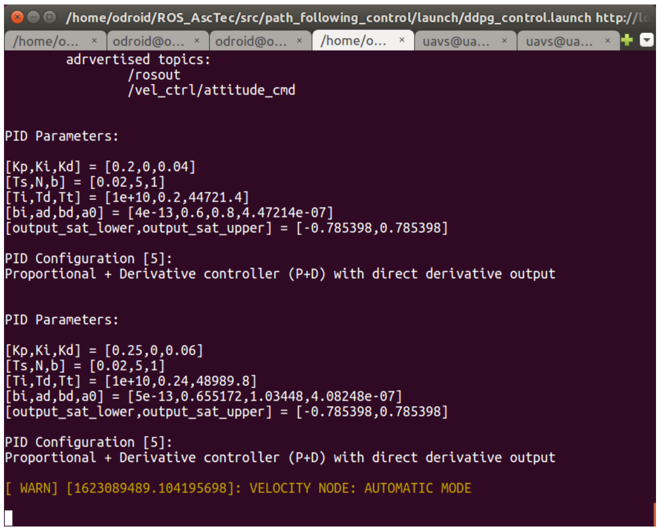
(a) GPS set reference coordinates node



```
[ INFO] [1623089755.682346075]:     Still waiting for 7 measurements
[ INFO] [1623089756.079287108]: Current measurement: 41.434528, 2.186616, 4294854.15
4000
[ INFO] [1623089756.079410572]:     Still waiting for 6 measurements
[ INFO] [1623089756.496321829]: Current measurement: 41.434530, 2.186616, 4294853.63
6000
[ INFO] [1623089756.496474502]:     Still waiting for 5 measurements
[ WARN] [1623089756.682095181]: No GPS fix
[ INFO] [1623089756.887683083]: Current measurement: 41.434532, 2.186616, 4294853.14
2000
[ INFO] [1623089756.887805588]:     Still waiting for 4 measurements
[ INFO] [1623089757.286751290]: Current measurement: 41.434531, 2.186618, 4294853.45
9000
[ INFO] [1623089757.286873170]:     Still waiting for 3 measurements
[ INFO] [1623089757.684144301]: Current measurement: 41.434532, 2.186618, 4294853.41
2000
[ INFO] [1623089757.684256389]:     Still waiting for 2 measurements
[ WARN] [1623089757.888735744]: No GPS fix
[ INFO] [1623089758.081370513]: Current measurement: 41.434531, 2.186619, 4294853.65
9000
[ INFO] [1623089758.081563521]:     Still waiting for 1 measurements
[ INFO] [1623089758.497160264]: Current measurement: 41.434531, 2.186621, 4294853.77
0000
[ INFO] [1623089758.504635123]: Final reference position: 41.434438, 2.186629, 42948
75.531760
odroid@odroid:~$ rosrun asctec_hl_gps gps_conversion_node
[ INFO] [1623089767.347386738]: Waiting for GPS reference parameters...
[ INFO] [1623089767.351856511]: GPS reference initialized correctly 41.434438, 2.186
629, 4294875.531760
[ INFO] [1623089767.399047308]: using height measurement from GPS
[ WARN] [1623089767.609003938]: No GPS fix 1
```

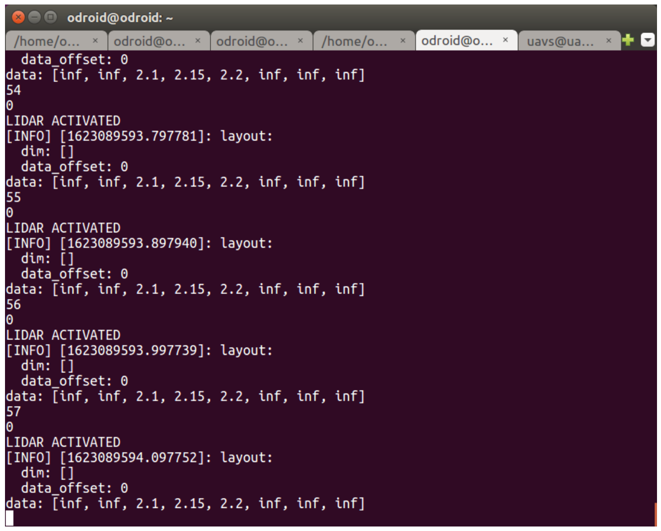
(b) GPS set conversion node

Figure 77: Steps 3 of the ground station Set-Up


/home/odroid/ROS_AscTec/src/path_following_control/launch/ddpg_control.launch http://l
/home/o... odroid@o... odroid@o... /home/o... uavs@ua... uavs@ua...

```
        adrvertised topics:
                /rosout
                /vel_ctrl/attitude_cmd


PID Parameters:

[Kp,Ki,Kd] = [0.2,0,0.04]
[Ts,N,b] = [0.02,5,1]
[Ti,Td,Tt] = [1e+10,0.2,44721.4]
[bi,ad,bd,a0] = [4e-13,0.6,0.8,4.47214e-07]
[output_sat_lower,output_sat_upper] = [-0.785398,0.785398]

PID Configuration [5]:
Proportional + Derivative controller (P+D) with direct derivative output


PID Parameters:

[Kp,Ki,Kd] = [0.25,0,0.06]
[Ts,N,b] = [0.02,5,1]
[Ti,Td,Tt] = [1e+10,0.24,48989.8]
[bi,ad,bd,a0] = [5e-13,0.655172,1.03448,4.08248e-07]
[output_sat_lower,output_sat_upper] = [-0.785398,0.785398]

PID Configuration [5]:
Proportional + Derivative controller (P+D) with direct derivative output

[ WARN] [1623089489.104195698]: VELOCITY NODE: AUTOMATIC MODE
```

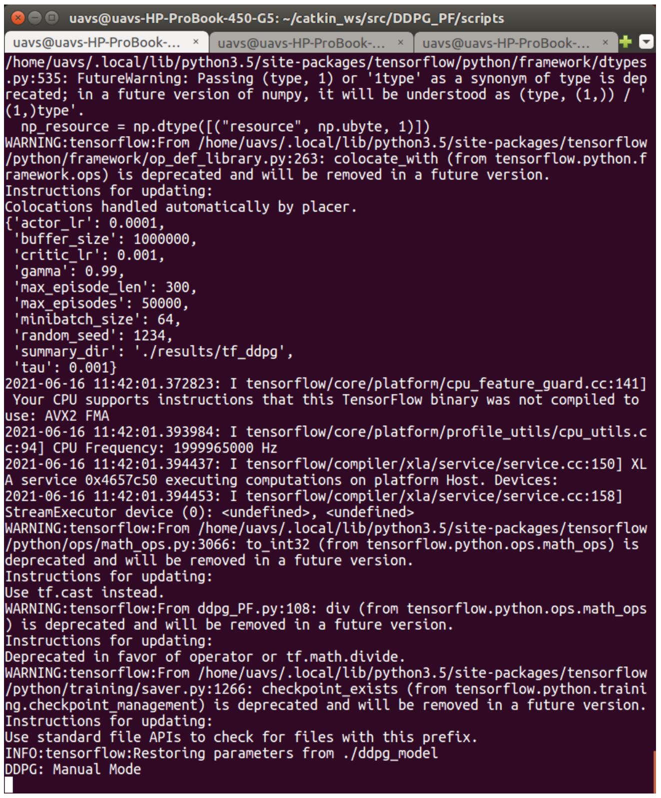
(a) DDPG launcher on for the Onboard Computer


odroid@odroid: ~
/home/o... odroid@o... odroid@o... /home/o... odroid@o... uavs@ua...

```
  data_offset: 0
data: [inf, inf, 2.1, 2.15, 2.2, inf, inf, inf]
54
0
LIDAR ACTIVATED
[INFO] [1623089593.797781]: layout:
  dim: []
  data_offset: 0
data: [inf, inf, 2.1, 2.15, 2.2, inf, inf, inf]
55
0
LIDAR ACTIVATED
[INFO] [1623089593.897940]: layout:
  dim: []
  data_offset: 0
data: [inf, inf, 2.1, 2.15, 2.2, inf, inf, inf]
56
0
LIDAR ACTIVATED
[INFO] [1623089593.997739]: layout:
  dim: []
  data_offset: 0
data: [inf, inf, 2.1, 2.15, 2.2, inf, inf, inf]
57
0
LIDAR ACTIVATED
[INFO] [1623089594.097752]: layout:
  dim: []
  data_offset: 0
data: [inf, inf, 2.1, 2.15, 2.2, inf, inf, inf]
```

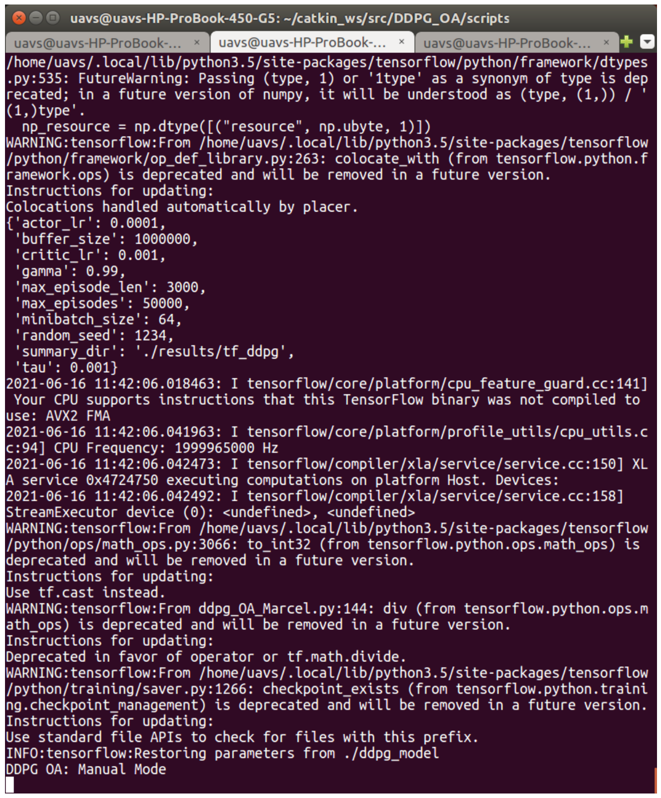
(b) LIDAR launcher for the onboard computer

Figure 78: Steps 4 & 5 of the ground station Set-Up


uavs@uavs-HP-ProBook-450-G5: ~/catkin_ws/src/DDPG_PF/scripts
uavs@uavs-HP-ProBook-... uavs@uavs-HP-ProBook-... uavs@uavs-HP-ProBook-...

```
/home/uavs/.local/lib/python3.5/site-packages/tensorflow/python/framework/dtypes
.py:535: FutureWarning: Passing (type, 1) or '1type' as a synonym of type is dep
recated; in a future version of numpy, it will be understood as (type, (1,)) / '
(1,)type'.
  np_resource = np.dtype([("resource", np.ubyte, 1)])
WARNING:tensorflow:From /home/uavs/.local/lib/python3.5/site-packages/tensorflow
/python/framework/op_def_library.py:263: colocate_with (from tensorflow.python.f
ramework.ops) is deprecated and will be removed in a future version.
Instructions for updating:
Colocations handled automatically by placer.
{'actor_lr': 0.0001,
 'buffer_size': 1000000,
 'critic_lr': 0.001,
 'gamma': 0.99,
 'max_episode_len': 300,
 'max_episodes': 50000,
 'minibatch_size': 64,
 'random_seed': 1234,
 'summary_dir': './results/tf_ddpg',
 'tau': 0.001}
2021-06-16 11:42:01.372823: I tensorflow/core/platform/cpu_feature_guard.cc:141]
 Your CPU supports instructions that this TensorFlow binary was not compiled to
use: AVX2 FMA
2021-06-16 11:42:01.393984: I tensorflow/core/platform/profile_utils/cpu_utils.c
c:94] CPU Frequency: 1999965000 Hz
2021-06-16 11:42:01.394437: I tensorflow/compiler/xla/service/service.cc:150] XL
A service 0x4657c50 executing computations on platform Host. Devices:
2021-06-16 11:42:01.394453: I tensorflow/compiler/xla/service/service.cc:158]
StreamExecutor device (0): <undefined>, <undefined>
WARNING:tensorflow:From /home/uavs/.local/lib/python3.5/site-packages/tensorflow
/python/ops/math_ops.py:3066: to_int32 (from tensorflow.python.ops.math_ops) is
deprecated and will be removed in a future version.
Instructions for updating:
Use tf.cast instead.
WARNING:tensorflow:From ddpg_PF.py:108: div (from tensorflow.python.ops.math_ops
) is deprecated and will be removed in a future version.
Instructions for updating:
Deprecated in favor of operator or tf.math.divide.
WARNING:tensorflow:From /home/uavs/.local/lib/python3.5/site-packages/tensorflow
/python/training/saver.py:1266: checkpoint_exists (from tensorflow.python.traini
ng.checkpoint_management) is deprecated and will be removed in a future version.
Instructions for updating:
Use standard file APIs to check for files with this prefix.
INFO:tensorflow:Restoring parameters from ./ddpg_model
DDPG: Manual Mode
```

(a) DDPG Path Following Ground Computer


uavs@uavs-HP-ProBook-450-G5: ~/catkin_ws/src/DDPG_OA/scripts
uavs@uavs-HP-ProBook-... uavs@uavs-HP-ProBook-... uavs@uavs-HP-ProBook-...

```
/home/uavs/.local/lib/python3.5/site-packages/tensorflow/python/framework/dtypes
.py:535: FutureWarning: Passing (type, 1) or '1type' as a synonym of type is dep
recated; in a future version of numpy, it will be understood as (type, (1,)) / '
(1,)type'.
  np_resource = np.dtype([("resource", np.ubyte, 1)])
WARNING:tensorflow:From /home/uavs/.local/lib/python3.5/site-packages/tensorflow
/python/framework/op_def_library.py:263: colocate_with (from tensorflow.python.f
ramework.ops) is deprecated and will be removed in a future version.
Instructions for updating:
Colocations handled automatically by placer.
{'actor_lr': 0.0001,
 'buffer_size': 1000000,
 'critic_lr': 0.001,
 'gamma': 0.99,
 'max_episode_len': 3000,
 'max_episodes': 50000,
 'minibatch_size': 64,
 'random_seed': 1234,
 'summary_dir': './results/tf_ddpg',
 'tau': 0.001}
2021-06-16 11:42:06.018463: I tensorflow/core/platform/cpu_feature_guard.cc:141]
 Your CPU supports instructions that this TensorFlow binary was not compiled to
use: AVX2 FMA
2021-06-16 11:42:06.041963: I tensorflow/core/platform/profile_utils/cpu_utils.c
c:94] CPU Frequency: 1999965000 Hz
2021-06-16 11:42:06.042473: I tensorflow/compiler/xla/service/service.cc:150] XL
A service 0x4724750 executing computations on platform Host. Devices:
2021-06-16 11:42:06.042492: I tensorflow/compiler/xla/service/service.cc:158]
StreamExecutor device (0): <undefined>, <undefined>
WARNING:tensorflow:From /home/uavs/.local/lib/python3.5/site-packages/tensorflow
/python/ops/math_ops.py:3066: to_int32 (from tensorflow.python.ops.math_ops) is
deprecated and will be removed in a future version.
Instructions for updating:
Use tf.cast instead.
WARNING:tensorflow:From ddpg_OA_Marcel.py:144: div (from tensorflow.python.ops.m
ath_ops) is deprecated and will be removed in a future version.
Instructions for updating:
Deprecated in favor of operator or tf.math.divide.
WARNING:tensorflow:From /home/uavs/.local/lib/python3.5/site-packages/tensorflow
/python/training/saver.py:1266: checkpoint_exists (from tensorflow.python.traini
ng.checkpoint_management) is deprecated and will be removed in a future version.
Instructions for updating:
Use standard file APIs to check for files with this prefix.
INFO:tensorflow:Restoring parameters from ./ddpg_model
DDPG OA: Manual Mode
```

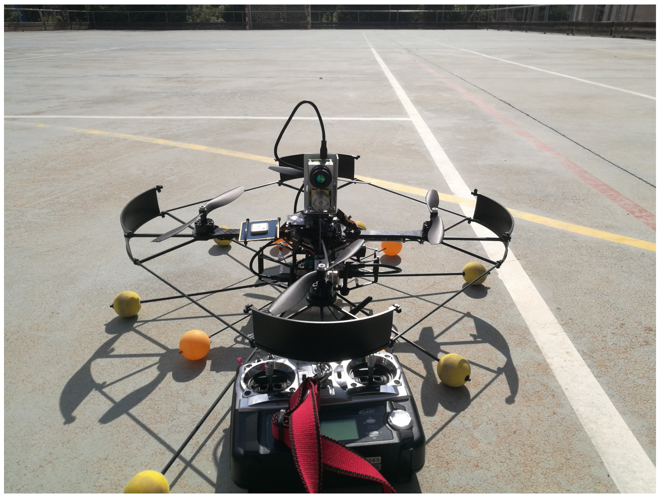
(b) DDPG Obstacle Avoidance Ground Computer

Figure 79: Steps 6 & 7 of the ground station Set-Up

9. Flight Tests

In the following section the different flight test performed in order to test and validate all the software improvement and modifications. During these tests, the flight performance and results have improved progressively as all the software parameters and flight conditions were improved as well.

Figure 80a presents the drone ready and assembled on the flight field. The Odroid is under the drone and the LIDAR sensor is mounted on top of the frame. The GPS is located on the left arm and is equipped with ten foam ball to protect the UAV from hard landings. The lateral protection system consisting of a simple frame can also be seen. It will protect the propellers in case of flipping over or tilting strongly near the ground.
The radio is a standard RC radio with seven channels. Four for direction control, one for flight modes and the other two as additional control parameters. The potentiometer is used for activating the autonomous mode.


(a) Drone ready for flight in the test area.

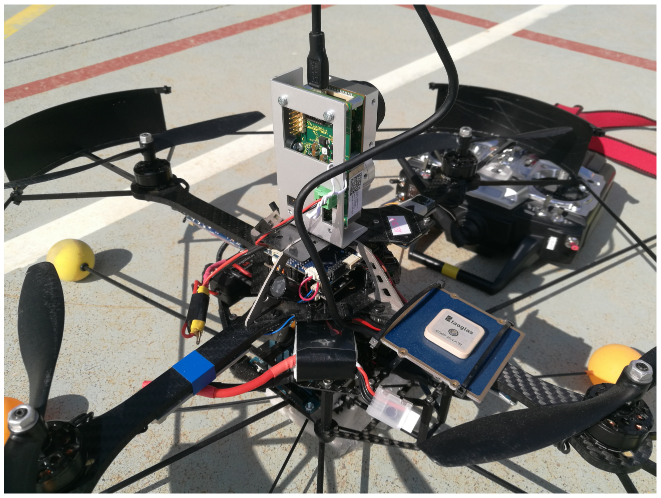

(b) Rear view of the sensor attached to the drone

Figure 80: Preparation of the Drone previous to flight

Figure 80bpresents a rear view of the drone. The LIDAR sensor can be seen mounted using an aluminium frame bolted on the top of the drone. The sensor is powered using a voltage regulator connected to the battery (seen in the middle of the image). The sensor is connected via USB to the Odroid under the drone. In the image, the USB cable can be seen protruding over the sensor. A more detailed view can also be seen from the GPS. The PCB contains the standard GPS patch antenna and has some electronic under to properly communicate with the Odroid. The battery is located at the central body of the UAV between the top flight control boards and the Odroid on the bottom. The battery is accessible for easy replacement, this battery can last for about 7 test of 1.5 minutes each.

9.1. Flight Test 0

This initial flight has been done in order to test the accuracy of a fixpoint hovering flight. The flight took place at an altitude of 1.5m and the drone was manually maintained in a fixed tridimensional position. The duration of the exercise was 50s and the accuracy of the manually hover flight was 15cm on each axis. This test will be used to know the precision of the positioning sensors such as the GPS and the barometer.
Figure 81 presents the X and Y coordinates seen by the drone onboard computer following the GPS system. As it can be seen the flight described by the sensor is very different from the one that the UAV actually flew. The position has an enormous uncertainty. Just to compare the flight was a vertical takeoff with a hover for 45 seconds and a landing at the same point. It can be seen that the uncertainty of the GPS is around 2 meters. This will be an added complexity as it will be more difficult to review the actual flown path of the drone.

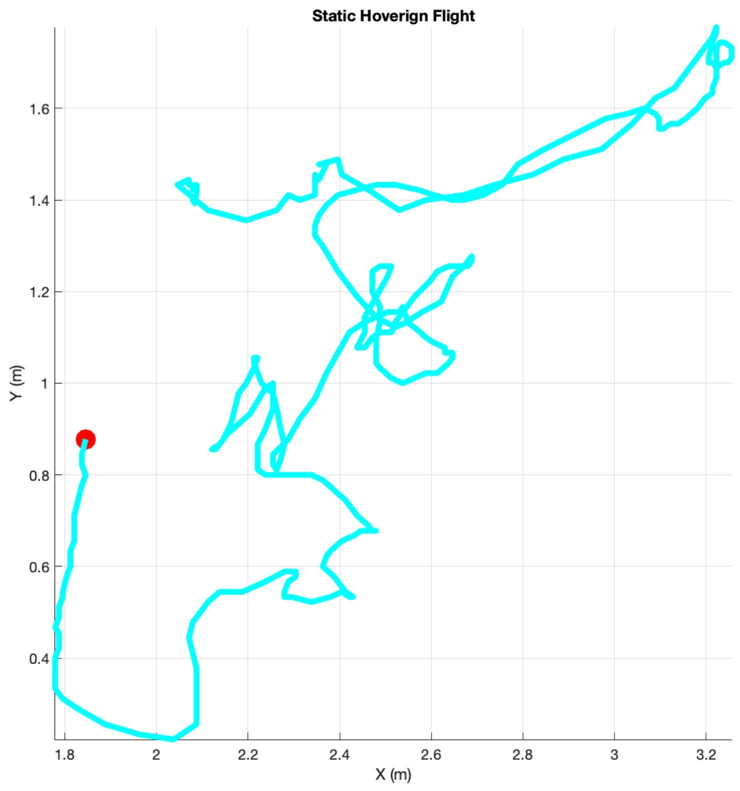


Figure 81: Hovering Flight X and Y error displacement

Figure 82 presents the vertical error of the drone. Again the uncertainty is very high as the position of the UAV oscillates over time and has an even higher vertical uncertainty of around 5 meters. This as before is a problem as adds difficulty in interpreting the data.

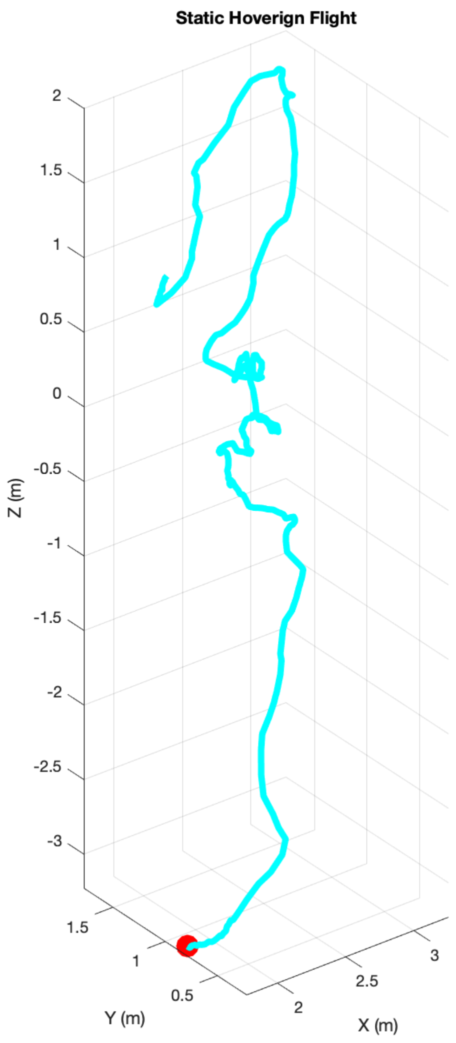


Figure 82: Hovering flight Z error displacement

As a side note the day that the test was conducted the system was seen a total of 10 GPS and the sky conditions were cloudy. Clouds can reduce the accuracy of a GPS measure but not at the presented level. This GPS must be replaced for more precise hardware in future projects. This difficulty was already present in Rubí's thesis as it was difficult to archive good GPS readings. A lot of test are needed in order to obtain a valid and coherent path following.

Summing up this test has been useful for analysing the sensors that will be used for testing the actual avoidance capabilities of the drone. Again the behaviour could be validated visually when flying but plotting all the log data will be equally important. This GPS uncertainty will need to be taken into consideration when tracing straight path or hovering into fixed positions. Once the drone is moving the uncertainty seems to be reduced as the path in the dynamic test seems a little bit more accurate.

9.2. Flight Test 1

This first flight test was done in the drone flight area of ESEIAAT. The main objective was to familiarize myself with the flight behaviour and protocols of the Hummingbird. The modified DDPG-OA with the LIDAR mounted were also tested. The programmed objective was to perform a straight line using the DDPG-OA without any visible lines. The flight conditions were good but with some wind and wind gust of 27Km/h. The results are presented below. As it will be evident the tests were a little bit inconclusive but in line for a first flight. The main objective of this first flight was to record and process some data to see that ROS and all its systems work properly.

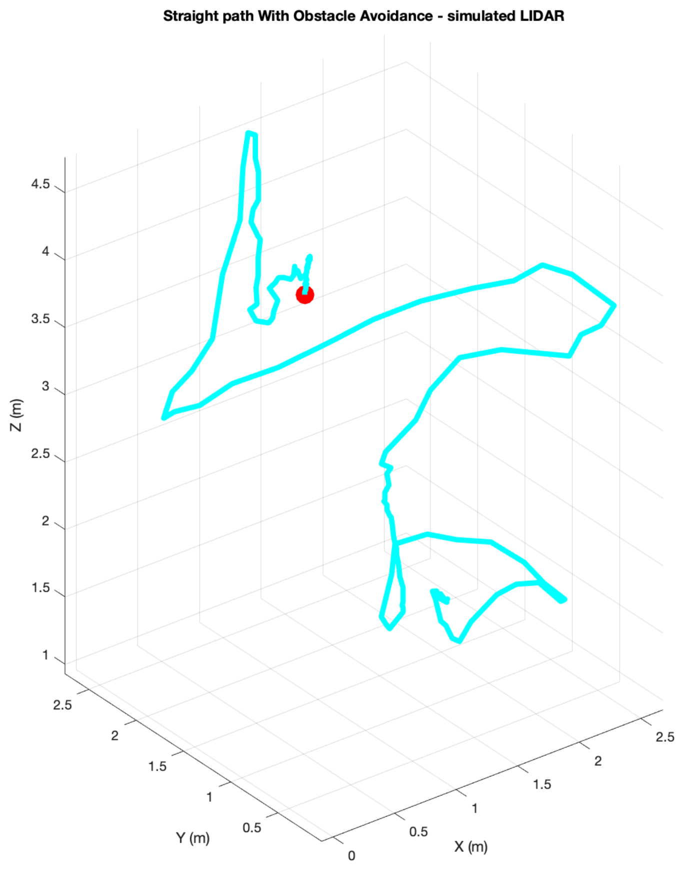


Figure 83: Tridimensional path of Flight Test 1

Figure 83 presents the dimensional path of the Hummingbird during the test. The red dot identifies the first position recorded. As it can be appreciated the UAV struggled to maintain a precise attitude control and follow a straight path, probably due to the wind and other accuracy and perturbation factors. In this scenario, the Autonomous navigation was only performed for a few seconds (between time 10s to 17s). Again this preliminary test was to familiarize with the platform and develop a code to post-process all the recorded data.

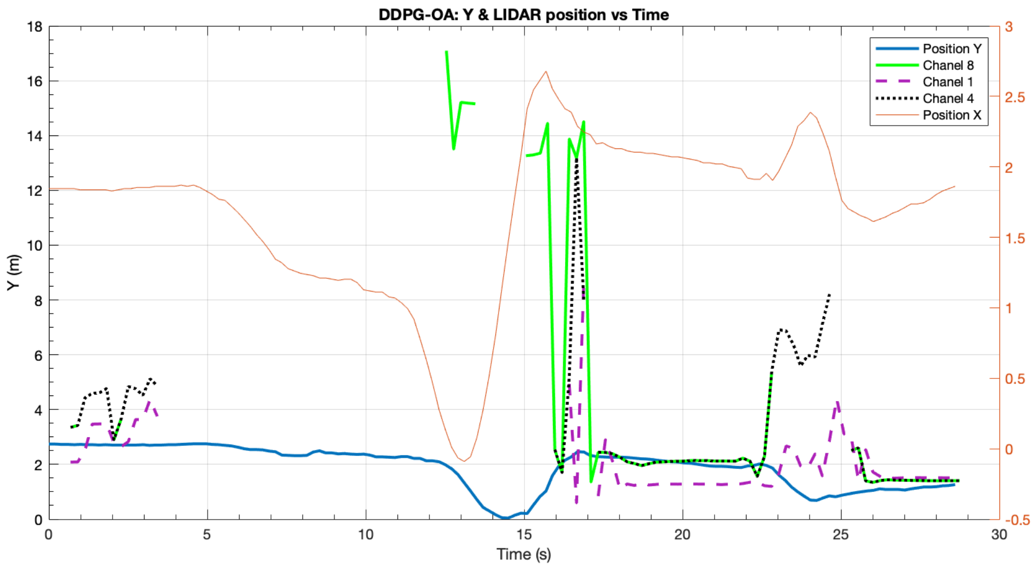


Figure 84: Leddar Test using the Windows configuration Software

Figure 84 presents the combined LIDAR and position data for that particular flight. It can be appreciated that the data present the same overall behaviour that during the simulations. The LIDAR performs as expected and the obtained data is the same as on the simulations (figure 55). Figure 85 presents all the raw data of the LIDAR. The interesting part of that plot is that between seconds 5 to 15 as the drone is hovering at a fixed altitude and attitude the LIDAR does not detect anything as it was far away from any of the flight field boundaries.

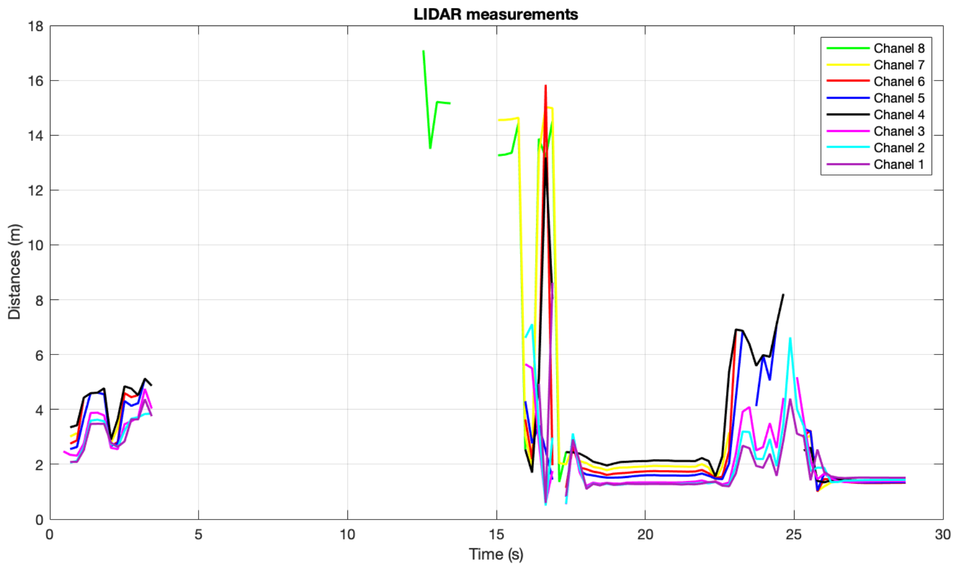
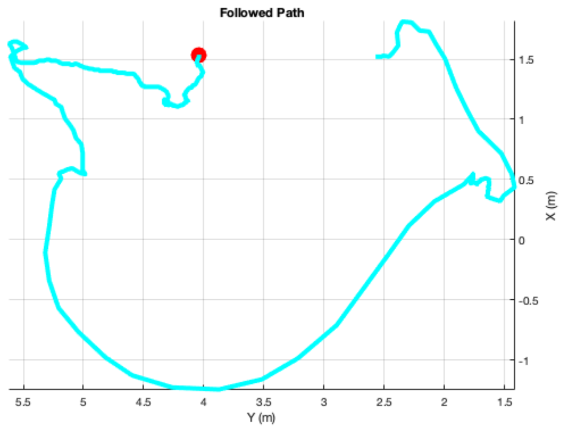


Figure 85: Leddar sensor data captured during Flight Test 1

9.2.1. Flight Test 1: Additional LIDAR Flight Data

This section present some additional flight log data. This flight, as the previous one, where done with the LIDAR onboard. The same conclusions than the previous section can be drawn. The LIDAR works perfectly in the ROS environment an the data has the expected behaviour.



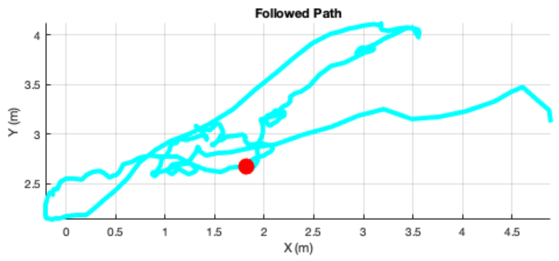


Figure 86: Bidimensional path of the flight trajectory of test 1.2 (right) and 1.3 (left)

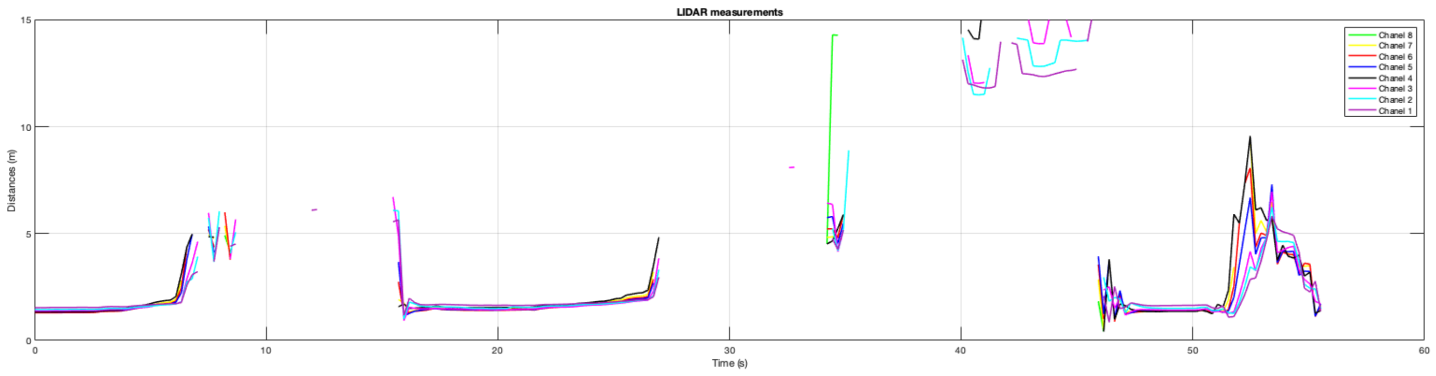


Figure 87: Leddar sensor data captured during Flight Test 1.2

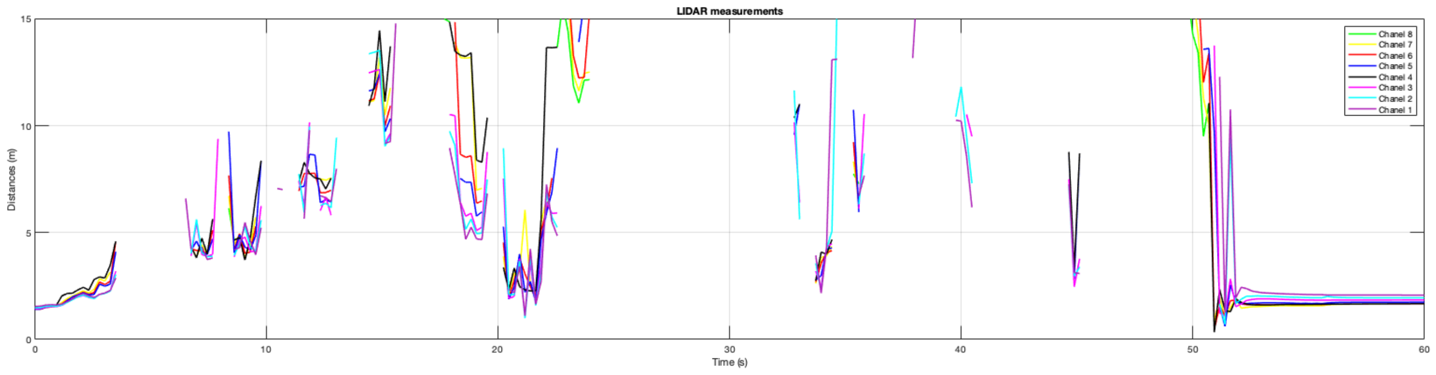


Figure 88: Leddar sensor data captured during Flight Test 1.3

9.3. Flight Test 2

In this test, the Hummingbird was tested just using the Path following algorithm. The script was programmed to perform a straight line. In this test, a notable wind was present during the flights. This wind came from the left side of the drone. In the frame of reference for the 3D plots, it came in the direction of the Y-axis pointing down (the wind vector could be from (2.5,-1) to (0.5,-1)). The wind intensity was around 15 Km/h. As it can be seen in the plots due to the wind conditions the straight line was altered to a diagonal path.

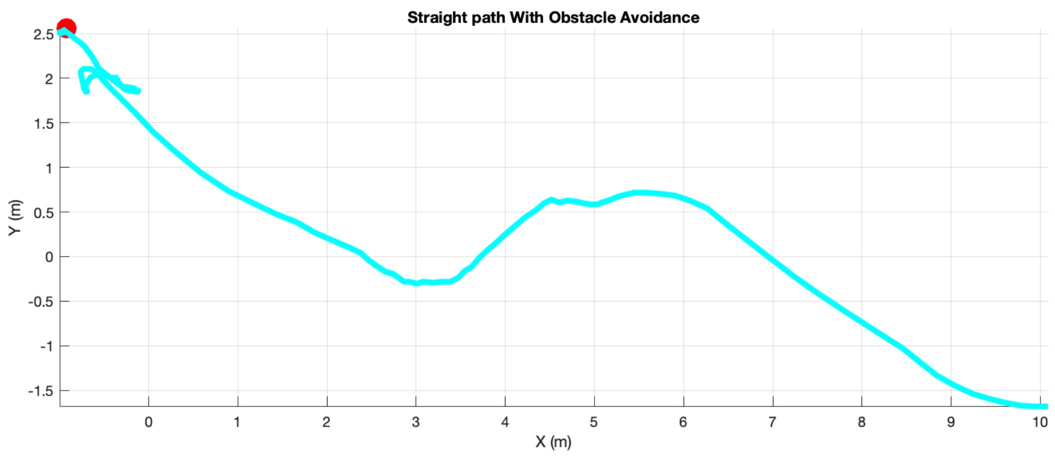


Figure 89: Leddar sensor data captured during Flight Test 1

In some test (figure 89) the Hummingbird tried to correct the external perturbations but its programmed correction response was not intense enough to oppose the wind, this was already path of the Path Following Algorithm. The green dot indicates the moment that the DDPG-OA algorithm is activated. In the other plots, the activation was almost at the initial of the test.

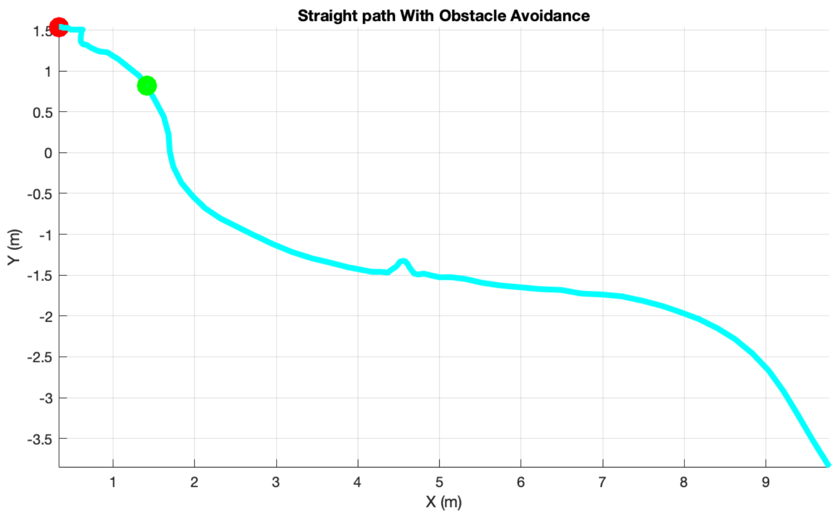


Figure 90: Leddar sensor data captured during Flight Test 2

Figure 91 is the one that is the most strongly affected by the wind while, in figure 89 the UAV flown in good enough conditions to slightly correct the wind perturbations but in this case, the wind completely deformed the programmed path in a diagonal line.
In conclusion, this test flight was considered successful as the Hummingbird was capable of, more or less, following a straight path. This test will be very useful to compare these results with the ones where LIDAR detections are introduced to the system.

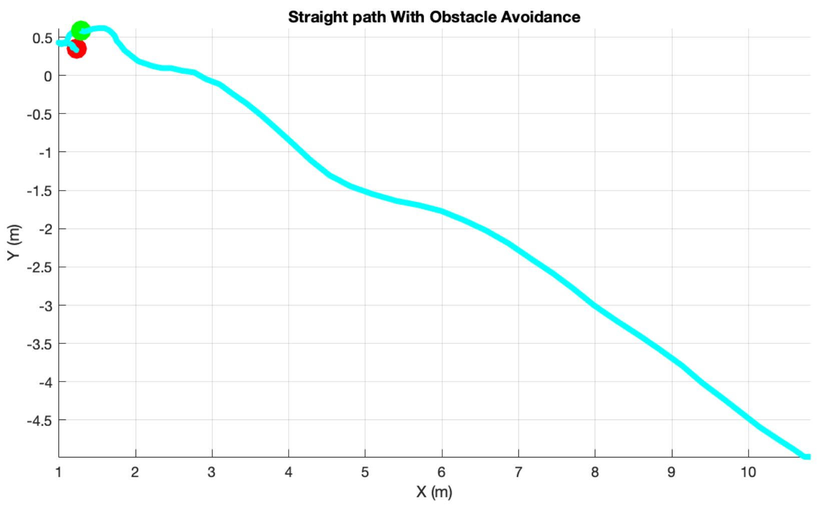


Figure 91: Leddar sensor data captured during Flight Test 3

9.4. Flight Test 3

In this flight test, the LIDAR sensor was not available as it was already sent for reparation and the simulating LIDAR code was used in order to provide some method of testing that all the developed system worked even if the main sensor was not in place. The objective of this test flight is to improve the rectilinear paths and also see the response of the sensor to the code generated sensor detections. The results that are expected are the same ones as seen in the Gazebo straight path simulations using the same code to feed the algorithm with external LIDAR measurements (figures 60 and 64). During this test, the wind conditions were good enough with stable wind velocities of 5 Km/s. The results are presented below.

In this first test, the drone performed a straight path while performing an avoiding manoeuvre. As it can be seen the feed measure to the algorithm are a simple LIDAR pulse. The drone successfully performed a straight path with an even better result than in the previous test.

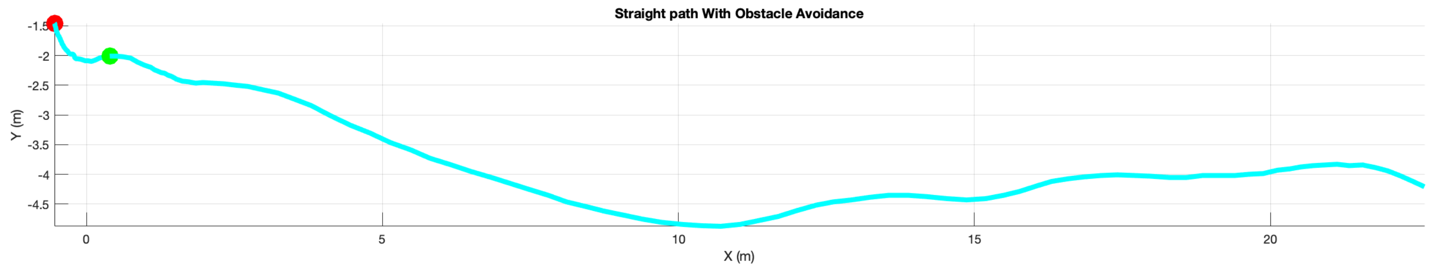


Figure 92: Top view of the performed path. The green dots marks the start of the algorithm. Test 3

Figure 93 presents the correlation between the X and Y coordinates and the LIDAR readings. It can be seen that the drone responds to the LIDAR detections and starts and avoiding manoeuvre but struggles to recuperate the initial path, mainly due to the wind perturbations. The plotted segment is equivalent to the path followed from the green point forward.

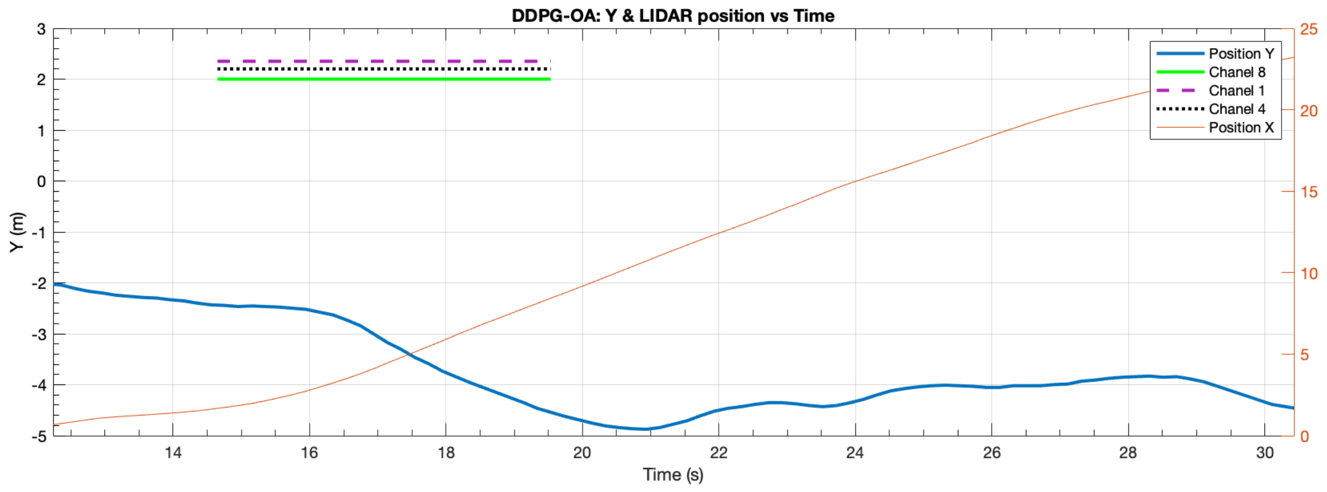


Figure 93: Combination view of the LIDAR with X and Y position Test 3

Test 3.2 presented in figure 94 shows a similar result where the drone successfully follows a straight path. In this test, the external perturbations make it difficult to differentiate the LIDAR detections from the main path corrections of the algorithm. Despite that, the overall path is a notable good line. The drone flies in path following mode from the green dot forward.

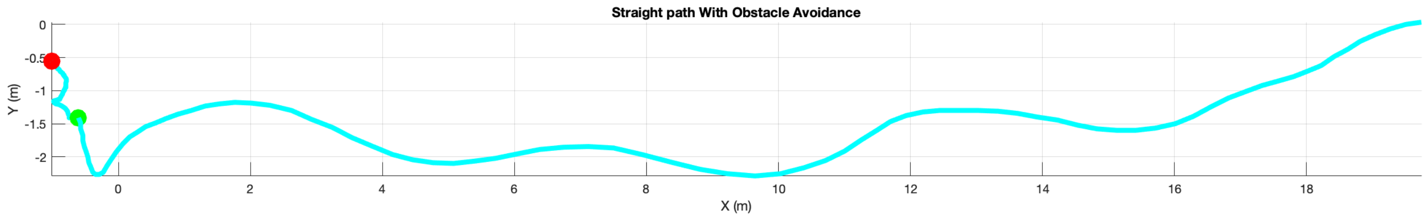


Figure 94: Top view of the performed path. The green dots marks the start of the algorithm. Test 3.2

The X and Y positions combined with the LIDAR does not present more information as the wind perturbations generate an unsolid path that can not be distinguished from the imposed LIDAR detections.

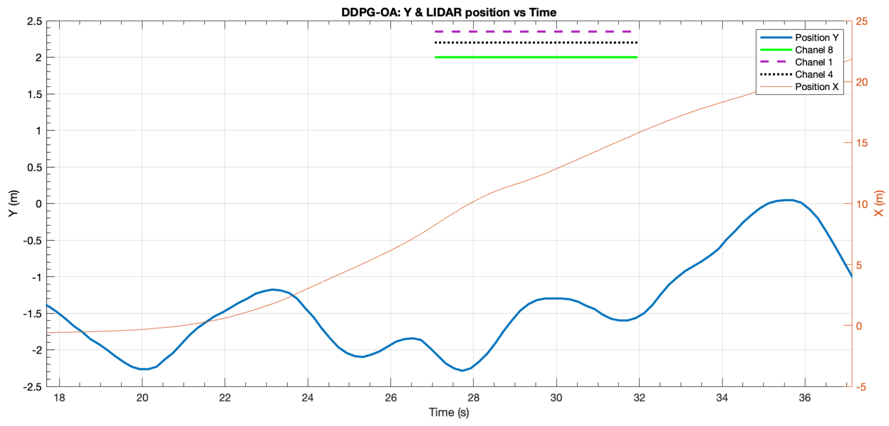


Figure 95: Combination view of the LIDAR with X and Y position Test 3.2

In the next test presented (figure 96) the obtained results are very similar to the previous one. Once the drone has been stabilized and the autonomous mode is activated (green dot) the UAV starts a rectilinear path with good success. Once the virtual LIDAR detection is activated and feed to the system the drone starts to turn in order to avoid the object. Despite the manoeuvre being correct, the turn has been accentuated by some wind which causes the drone to come near the perimeter of the flight arena and the test was aborted. Nevertheless, the overall path was successfully traced.

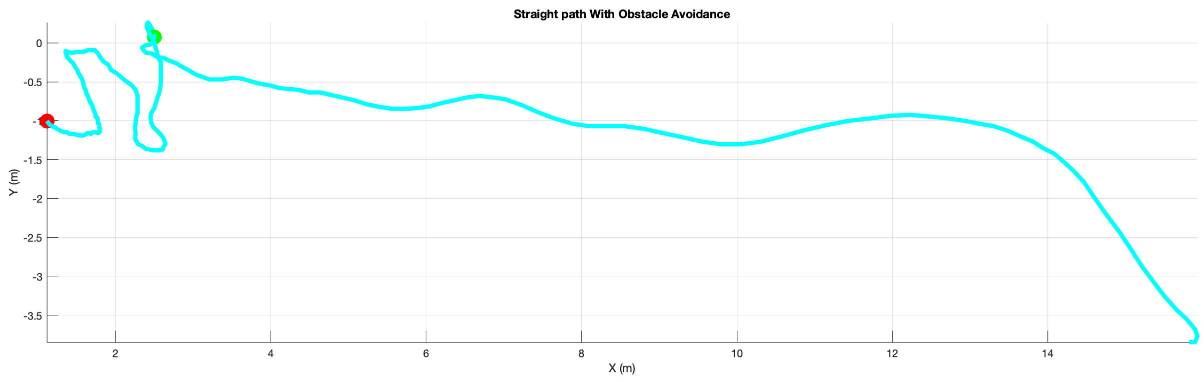


Figure 96: Top view of the performed path. The green dots marks the start of the algorithm. Test 3.3

Figure 97 presents the separated flight coordinates and the LIDAR during the autonomous flight mode. It can be appreciated that the Y coordinate is quite straight and once the LIDAR is activated the drone starts to turn. As mentioned the test was aborted before the manoeuvre could be finished as the drone was surpassing the safe flight perimeter of the test flight area.

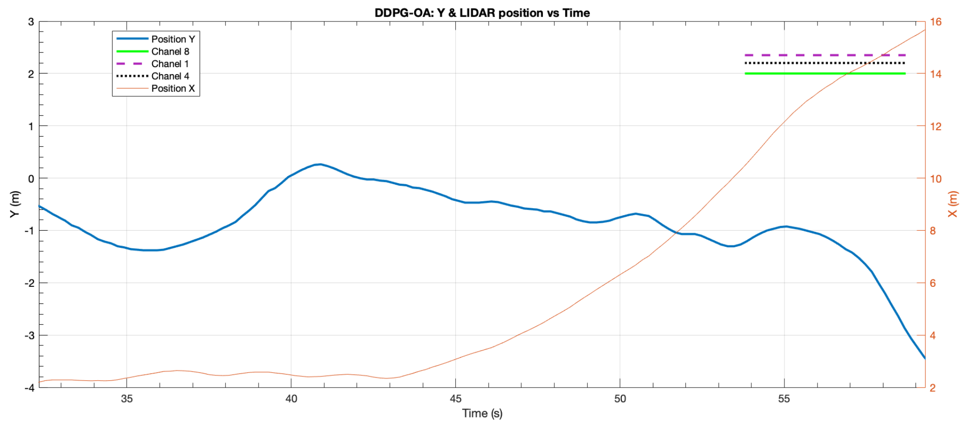


Figure 97: Combination view of the LIDAR with X and Y position Test 3.3

Summing Up the test were performed successfully and the LIDAR mockup readings were correctly tested. The flight conditions were not ideal as many external wind perturbations gave trouble to the path following the algorithm. In the next flight, more stable flight conditions will be needed in order to correctly identify and distinguish the avoidance trajectory.

9.5. Flight Test 4

These flight tests were performed early morning to avoid any wind. As it has been seen the system is very sensitive to external perturbations. The objective of this flight was to obtain cleaner straight paths from which the drone could perform a clear evasive manoeuvre when the LIDAR was fed with the virtual detection in front of them. As in the previous test the red dot indicates the initial position of the UAV and the green dot indicates the initialization of the autonomous mode.
In this first test seen in Figure 98, the drone was capable of performing a straight line with good accuracy. Once the

sensor detected the virtual target the drone performed an avoiding manoeuvre. In these plots, the first detection of the virtual target has been represented with red dots. These dots are what the sensor sees and performs the manoeuvre based on. This virtual wall that the sensor sees appears during a fixed period of time and always at the same distance from the drone.

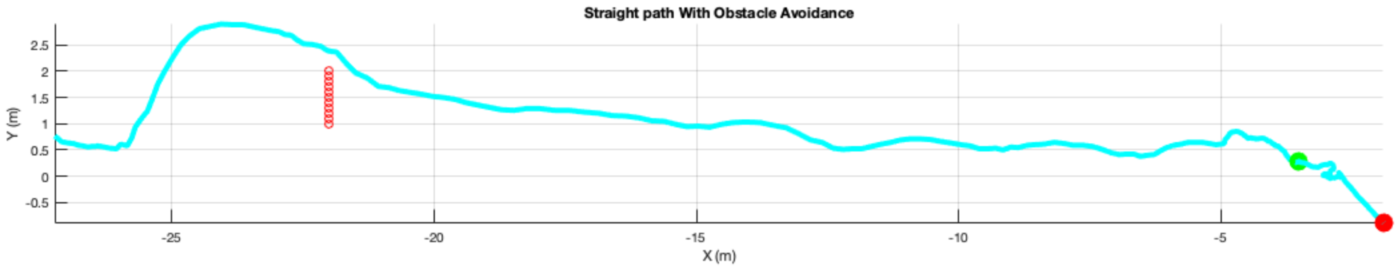


Figure 98: Top view of the performed path. The green dots marks the start of the algorithm. Test 4

Figure 99 presents the separated X and Y coordinates in correlation with the LIDAR detections. These show as constantly fixed objects always at the same distance of the drone. They simulate an omnipresent object that the drone sees for a limited time and performs the avoiding manoeuvre based on them. It can be observed that the trajectory is very rectilinear and once the detections are seen by the system the UAV stars avoiding the "wall". As in the simulation tests once the object is no longer visible the drone returns to the original path with an additional oscillation trying to reach the initial path.

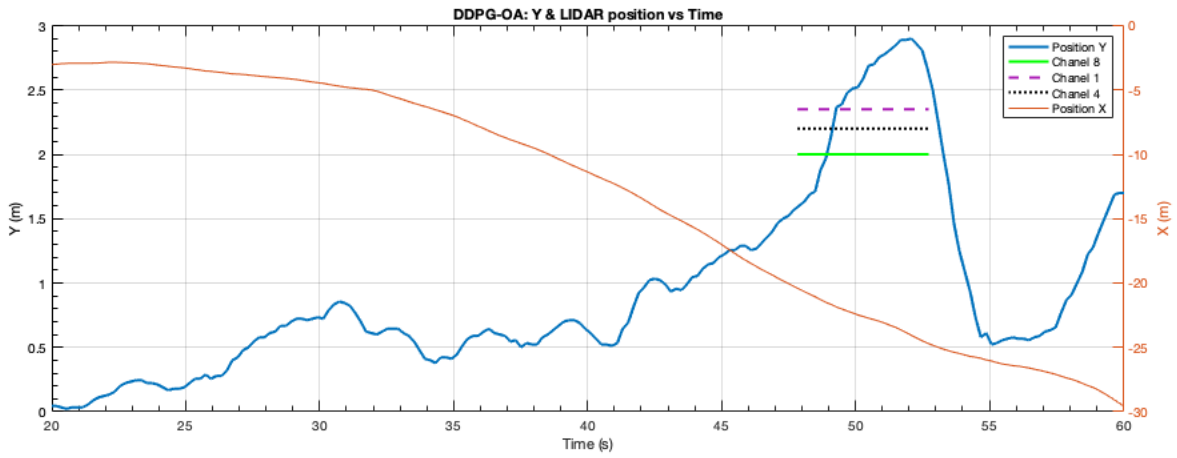


Figure 99: Combination view of the LIDAR with X and Y position Test 4

The second flight presented in figures 100 and 101 present again a very stable path where the drone has performed an avoidance manoeuvre successfully. The red dot indicates the starting point of the flight and the green one at the start of the autonomous flight. As the LIDAR detection was triggered at the same time as the autonomous mode the drone performed an avoidance manoeuvre at the very beginning of the path and then regained the rectilinear trajectory. The red dots indicate the moment where the attached virtual object was first seen by the sensor. As in the other test and

Gazebo simulations, the drone had some troubles regaining the linearity of the path. After a few zigzags, it settled again on the straight path.

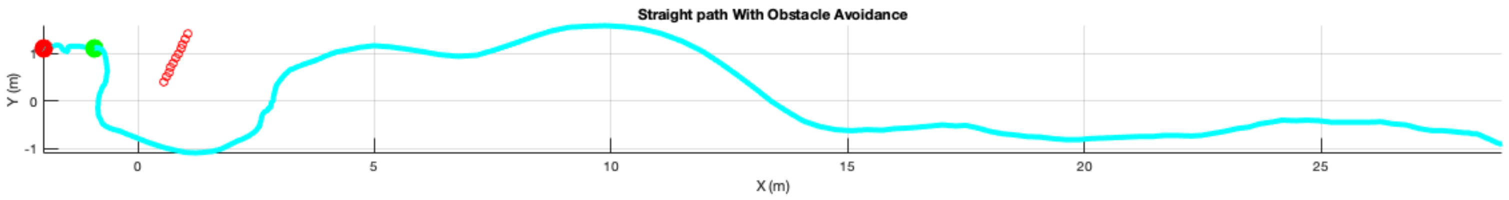


Figure 100: Top view of the performed path. The green dots marks the start of the algorithm. Test 4.2

It can be observed that the detections of the LIDAR are correlated with the Y displacement of the drone. As the "wall" is seen the drone starts moving and once it disappears the vehicle turns to regain the initial straight path. Again the response using the DDPG is not immediate as the drone waited for a few instants to evaluate the position and distance to the target.

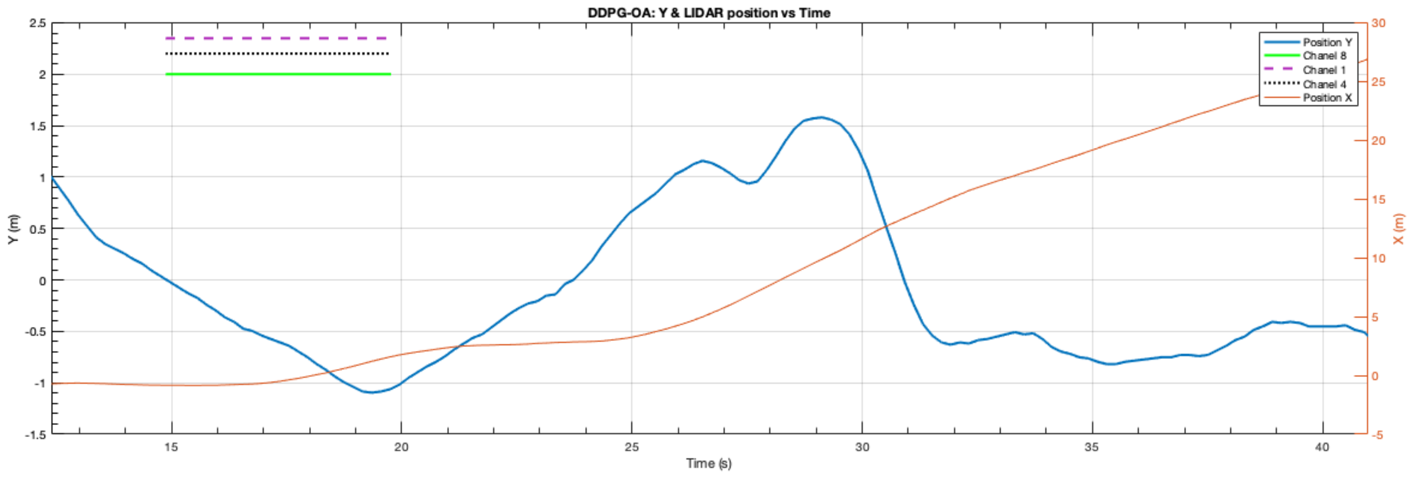


Figure 101: Combination view of the LIDAR with X and Y position Test 4.2

In general, these tests gave better results as the flight conditions were carefully chosen and the drone propellers were replaced with newer ones. All the possible external perturbations were reduced as much as possible. The results are good as the drone presented a strong correlation between the detections of the virtual objects and the avoidance trajectories.

9.6. Final Test flights

These final flights were done in order to further validate the results seen in the previous test. The two tests were done early in the morning with no wind and clear sky. The objective was to obtain a simple avoidance manoeuvre and a complex one with two avoidance procedures.
Figure 102 presents the bidimensional path of the drone. As in the previous test, the red dot indicates the starting position and the green dot the activation of the DDPG algorithm. As it can be seen the drone followed a very straight path. Once the detection was triggered the hummingbird responded accordingly and proceeded to avoid the fictional wall. As in the previous test, the line of red dots indicates the first instant that the algorithm was feed with the fixed detection in front of the UAV.

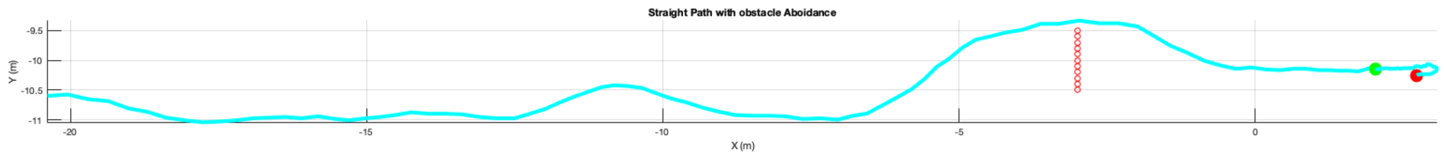


Figure 102: Top view of the performed path. The green dot marks the start of the algorithm. Test 5.1

Figure 103 presents the correlation between X and Y coordinates and the LIDAR detections. It can be seen that once the detection is seen by the algorithm it waits two seconds before moving sideways. Once the detection is no longer visible it returns to the original path. This small delay at the beginning of the manoeuvre was executed as the algorithm is trained to try to come as closer as possible to the detection before changing paths.

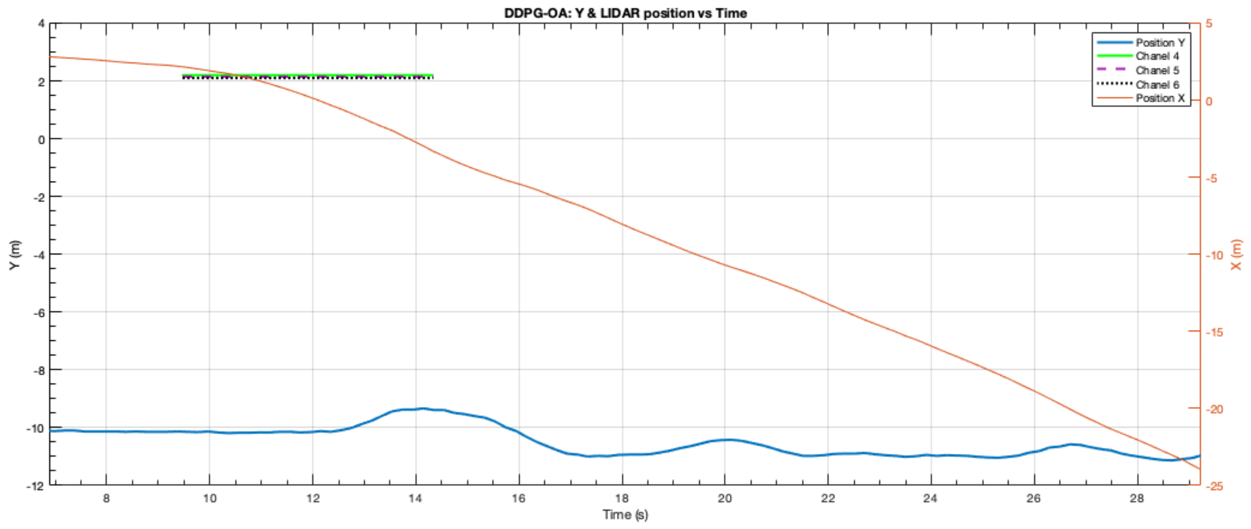


Figure 103: Combination view of the LIDAR with X and Y position Test 5.1

Figure 104 presents the second test. In this run, two detections were launched during the flight. This path is not so linear as the one before. Nevertheless, the two manoeuvres can be distinguished and were performed when expected. The line of red dots shows the first activation programmed fix detection; in the same way as in the previous test.

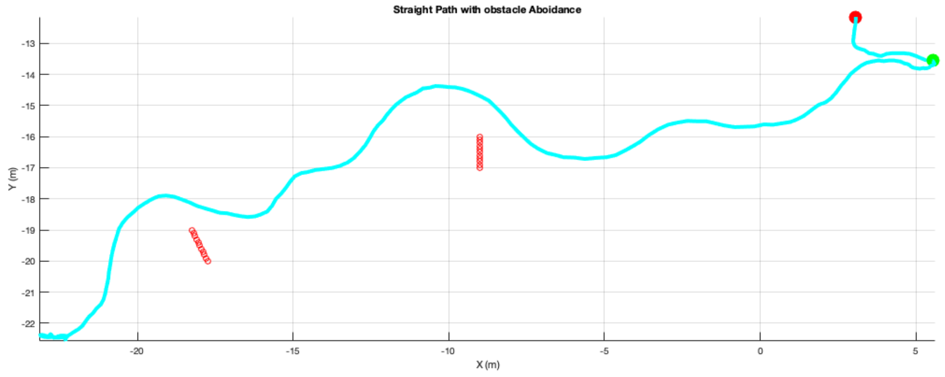


Figure 104: Top view of the performed path. The green dot marks the start of the algorithm. Test 5.2

Figure 105 presents the combination of position and LIDAR detections. The two detections are clearly visible. The geometry of the path makes it more difficult than the previous test to distinguish the obstacle avoidance path. Despite that, the test can be considered successful. As it has been seen in simulations the algorithm is capable of avoiding multiple objects along its path in real flight conditions.

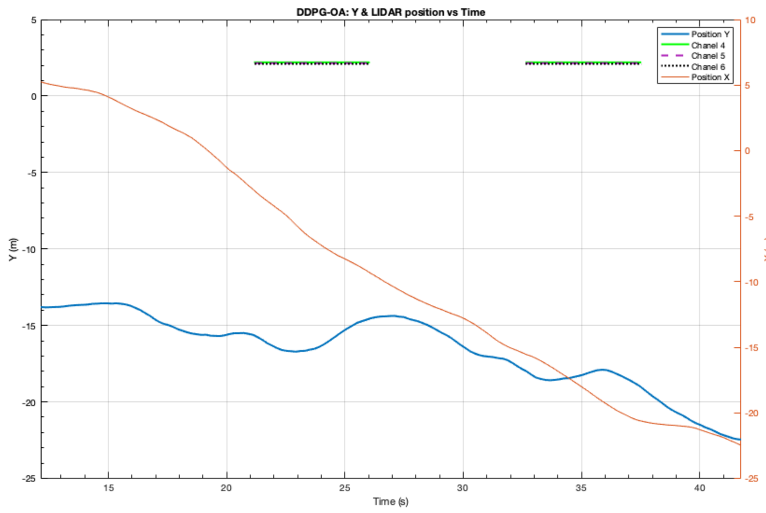


Figure 105: Combination view of the LIDAR with X and Y position Test 5.2

9.7. Flight Tests Conclusions

These flight tests have been a methodical task that has required a lot of time and effort. The first flight were inconclusive but after some iterations and the identification of some variables such as wind and GPS the test started to take shape. During the testing period, no accidental crashes occurred even when the algorithm lost control of the UAV. The total number of flights (successful and unsuccessful) done during this period is around 150 flights. The total airborne time of the drone is approximately 130 minutes. The ratio of success vs failure flights is 3/5. Overall the test provided a successful validation platform for the LIDAR implementation in obstacle avoidance. Once the proposed future improvements are implemented the drone will be a very reliable autonomous vehicle.

10. Conclusions and Discussion

In this section, the conclusions from all the conducted studies will be presented and discussed. In addition, some future developments and state of the art technology implementations will be discussed.

This Master thesis despite presenting some unexpected difficulties along the way such as the LIDAR sensor failing it concluded positively and all the objectives were properly achieved. The project proved challenging as all the programming languages were unknown and working with an open-source computer such as Ubuntu has added complexity.

10.1. Conclusions

This project proved complex and challenging but the obtained results were equally valuable. This project can be divided into three phases where each of them had its own difficulties and goals. The conclusions will follow this structure to better remark the evolution of the different steps along the way.

10.1.1. Ubuntu and ROS environment configuration

The first steps of this project involved preparing a computer from the ground up to support Ubuntu, ROS and all the additional pluggings needed. This didn't prove an easy task as Linux is not a robust environment for some things and it was very difficult to create an environment where all the pieces worked properly. The first build was done using Ubuntu 20.04 version which proved too new to be fully compatible with some of the ROS packages to work properly.
The second iteration was done with Ubuntu 16.04 and proved more successful. Nevertheless, a third iteration was needed with a completely new computer as the old CPU was not compatible with the Tensorflow module. Building the ROS environment also proved complex as a lot of packages were needed and sometimes the compatibility between them was not immediate.
The process of building a ROS environment was completely new and a lot of effort was put into assembling all the software pieces. The available information of how to assemble a particular ROS environment was also limited and some time the active participation on some GitHub forums was needed in order to troubleshoot the problems with the developer community. Some modifications were needed to be done in some packages such as RotorS in order to install modified versions of the simulator to properly communicate with all the newer packages.
This part required more time and effort than expected as it was a constant back and forth iterating to accomplish a successful build. Nevertheless, it proved very enriching as it gave me very rapidly a solid base in Linux and Ubuntu. At the end a very detailed guide was developed that indicates step by step all the actions needed to configure the environment for the next projects based on the work done (presented on Annex A).

10.1.2. LIDAR integration and configuration

The second part of the thesis consisted in developing a way of integrating the LedarTech sensor into the ROS environment and linking it into the existing virtual obstacle avoidance code. This task again proved very challenging and required a lot of time and effort.
The first problem was setting up and configuring Ubuntu and ROS to detect and recognise the sensor. Again no existing documentation was available or coherent and the process was done recuperating pieces from different GitHub repositories and web pages. In the end, a communication channel was established with the Company LedarTech in Canada which provided two modified STK packages that properly work with both the ground and onboard computer. Once the basic code was working on both the Ground Computer and the Odroid, and the LIDAR was taking measures, a custom ROS package was developed. This part again proved complex as nor programming ROS or Python was done before. A learning process was successfully accomplished and a ROS package capable of reading the LIDAR information and publishing it for the Obstacle Avoidance algorithm was implemented. The obstacle avoidance was also modified to properly process the new LIDAR data. The main challenges in this part (apart from learning the new language) was pre-processing the data of the LIDAR in order to feed to the algorithm the expected sensory data. This data has to be similar to the one in which the model had been virtually trained. The sensor was tested in the virtual environment and it was capable of

altering the drone trajectory in relation to the detections seen by the real sensor.
At this point, the code was modified and loaded to the Odroid for testing the system independently. The Hummingbird was fitted with a voltage regulator and was capable of flying with the sensor and processing all the data onboard.
All the steps were accomplished and the sensor was properly integrated into the existing environment. The simulation with the sensor went as expected. The hummingbird performed the avoidance manoeuvres in the virtual scenario of the real objects detected by the external sensor.

10.2. Test flights and validation

This last part consisted on performing tests flights with the developed codes and validating all the results seen in the simulation. The first test flight went properly and the UAV was capable of following the planned path. The test performed with the LIDAR also were successful in demonstrating that the created environment worked properly and that the sensor was correctly implemented into the system. Unfortunately once the full test flight were going to be done the LIDAR stopped working. To fully validate in a real path following and obstacle avoidance scenario a workaround was designed. This new method simulated the sensor using the existing ROS package environment created. The new program simulated with code the detection that the Leddartech sensor was seeing. The new code was linked to the RC radio and the pilot could manually trigger detections by feeding detections to the algorithm as if the LIDAR was present.
This method, despite not being perfectly accurate, gave valid result to validate the tests. The drone was capable of performing an avoidance manoeuvre once a detection was seen by the sensor. Once the object was not visible the drone returned to its original path.
Despite the added difficulties with the unexpected failure of the LIDAR sensor the tests were successfully performed and the created codes and algorithm were tested. The Hummingbird was capable of correctly navigating and avoiding the obstacles in its path.

10.3. Future Improvements

To improve this research work, several key points could be improved. Those are listed below.

- Perform test with the real sensor once this has been repaired. The results should be the same but more extensive conclusions could be drawn.
- The algorithm must be improved to better tolerate outdoor flight conditions. The drone was very sensitive to wind perturbations and only in pristine flight conditions the expected results were obtained.
- The PID of the drone control loop must also be adjusted for future operations as pitch and roll are responsive but the drone has a hard time when rotating in the yaw axis.
- The GNSS system has to be improved as the accuracy of the navigation equipment was very low and presented a high degree of uncertainty. A more robust and precise GPS system could provide the drone with a much better path following performance in the real world.
- Incorporate a barometer to the control system as the altitude of the UAV determined by the GNSS system is not enough for precise flights.
- The structure of the drone must also be improved once all the tests are done. The training legs and propeller guard while offering some degree of protection they decrease significantly the performance of the UAV. These elements increase the total weight of the vehicle and increase notably the coefficient of drag. The wind perturbations are also extenuated by those surfaces.
- Test the drone with a more complex geometry path and generate an interface to give the drone target points for easy design of a path.

10.4. Viability and Future Applications for the developed technology

As mentioned this technology has yet to be refined in order to be completely functional and reliable. Once this has been achieved the developed platform could be used in a variety of new scenarios. The advantage of flying platforms is their ability to rapidly cover ground without the complexity of terrain mobility and navigation. Nonetheless, this advantage becomes and inconvenient as once the vehicle do collide with an obstacle the damages can be catastrophic. Is for that reason that navigation and guidance system will be essential for future applications in flying vehicles. The developed technology could provide the ability to autonomously navigate in complex environments to a variety of flying drone platforms. As demonstrated the sensor and onboard computer are lightweight and modular which make them easy to install. The applications once this is achieved are very wide. Nowadays the majority of drones that fly in complex environments such as forests, cities, warehouses are being controlled with some degree of human interaction as the onboard sensor have some limitations. This new generation of path following and obstacle avoidance could be the next step for allowing fully autonomous operation of UAVs in complex scenarios.

As an example drones are being used for assessing damages when a natural catastrophe occurs such as fires, folding, earthquakes... but they are used for obtaining broad aerial information. With this new line of work, the drones could autonomously enter inside the close areas to gain detailed information. Path planning and obstacle avoidance in those scenarios are key for success but as it has been presented it can be achieved.
Drones are also being used in large construction operations or monitoring of large facilities. As before UAVs have a passive role when interacting with the environment as very restrictive safe distances are needed. With the proposed technology the vehicles would be able to go inside those constructions, monitor close areas or move easily inside warehouses.
This technology could also be used to make more accessible human - UAV integration. The high velocity moving parts makes it difficult to operate near a flying vehicle as it's not aware of its surroundings. With these sensors equipped onboard, the drone could operate near humans or fragile equipment without the danger of colliding and making harm. This will be really useful as both entities could share workspace and cooperate together. Also as the algorithms are based on Artificial Intelligence they have the capability to be easily trained in the new scenarios and be able to outperform traditional algorithms based on sequential programming. Their capability to adapt to the new environment together with their obstacle avoidance capabilities makes them the perfect quadrotor to operate in new more complex and challenging situations.

With the rise of these new technologies, the operation of UAVs will become more available and complex as well as safer. This technology will be the key for the future where drones could one day operate autonomously around humans bringing new opportunities to light.

A. Annex: User Installation Guide of the Software

In this section a user guide for the installation and use of ROS and RotorS will be presented. This section will highlight the most important steps during the installation and some of the possible errors. This guide is made with those new to the Linux environment as well as ROS (In this case Kinetic).
The first steps are the installation and configuration of ROS. The Official Web-page offers a detailed explanation [1]. The essential steps are described below.

Basic ROS installation:

```
sudo apt-get update
sudo apt-get install ros-kinetic-desktop-full
echo "source /opt/ros/kinetic/setup.bash" >> ~/.bashrc
source ~/.bashrc
```

Some additional packages must be installed. This packages provide a more user friendly compiling interface. Additional ROS packages installation:

```
sudo apt-get install python_catkin_tools
```

Once ROS has been installed a basic configuration of the workspace must be done. This workspace will be where all the plugins and programs will be installed.

Basic ROS catkin workspace setup:

```
mkdir -p ~/catkin_ws/src
cd ~/catkin_ws/
source devel/setup.bash
catkin build
```

If the installation of ROS and the creation of the catkin workspace has been done properly the following folder will be on the directory. The src (source) directory will be where all the ROS packages will be installed.

```
cd ~/catkin_ws/
ls
\$  build  devel   logs   src
```

The next step will be to install RotorS. The official RotorS version wont be used as it can generate compatibility problems. The slightly improved BebopS version will be used. This version is the same as the original RotorS with some additions. The standard RotorS version can also be installed installed in this step.

```
cd ~/catkin_ws/src
git clone -b med18_gazebo9 https://github.com/gsilano/rotors_simulator.git
cd ~/catkin_ws
rosdep install --from-paths src -i
catkin build
```

At this point the catkin compiler wont be able to process the other and some error will be displayed. This error indicates that some packages are required for compiling RotorS.

```
cd ~/catkin_ws/
sudo apt-get install ros-kinetic-geographic-msgs
sudo apt-get install ros-kinetic-octomap-msgs
sudo apt-get install ros-kinetic-octomap-ros
```

[1] https://wiki.ros.org/kinetic/Installation/Ubuntu

```
sudo apt-get install ros-kinetic-control-toolbox
sudo apt-get install ros-kinetic-mav-msgs
sudo apt-get install ros-kinetic-mavlink
sudo apt-get install ros-kinetic-mavros-msgs
```

Additional libraries must also be installed.

```
sudo apt-get install libgoogle-glog-dev
sudo apt-get install libgeographic-dev
sudo apt-get install autoconf
```

Once ad the additional packages have been installed on the system the catkin compiler should be able to process the RotorS package. The system can now run a simple example.

```
cd ~/catkin_ws
catkin build
roslaunch rotors_gazebo mav_hovering_example.launch mav_name:=firefly world_name:=basic
```

The installation until this point allows for basic RotorS usage. To work with the software used in this thesis additional ROS have to be installed. This packages are related with the communication of the different ROS nodes.

```
cd ~/catkin_ws/src
git clone https://github.com/ethz-asl/ethzasl_msf.git
git clone https://github.com/ethz-asl/asctec_mav_framework.git
git clone https://github.com/ethz-asl/glog_catkin.git
git clone https://github.com/catkin/catkin_simple.git
git clone https://github.com/ethz-asl/mav_comm.git
git clone https://github.com/mavlink/mavlink.git
git clone https://github.com/mavlink/mavros.git
catkin build
```

After all this extensions have been installed all the source codes can be merged inside the source (src) folder in ROS workspace. Another compilation (*catkin build*) of the worksapce must be done.

Some Python packages will be needed for running the DDPG simulations. The steps needed are described below.

```
sudo add-apt-repository ppa:deadsnakes/ppa
sudo apt update
sudo apt install python3.6 #Just in case of need, is not the first option with Ubuntu 16.04
```

After this the pip packages can be installed.

```
sudo apt update
sudo apt install python3-dev python3-pip python3-venv
pip install --upgrade pip
#
pip install tflearn
pip install rospy
pip install gym
pip install argparse
pip install netifaces
```

Possible troubleshooting at this point: If some problems appear while upgrading pip and the installation fails the following command forces the system to upgrade.

```
    python3 -m pip install --user --upgrade pip
```

The Tensorflow packages is quite tricky to install as it has some requirements that the CPU or GPU of the system must fulfill. The first thing is to check if the system CPU supports AVX. This simple comand can be typed in and in the "Flags" section avx will be dysplayed if the system suports it [17, 4].

```
  lscpu
    #Architecture        x86_64
    #CPU(s)              2
    #....                ....
    #Flags:              fpu vme de psc tsc msr pae mce cx8 ....
```

The secon thing to chech is if the python version istalled is the adequated one and if its 64 bits (essential for Tensorflow to work).

```
    python
    >>> import platform
    >>> platform.architecture()
        # Somthing like ('64bit', 'ELF') will appear
```

The installation of Tensorflow can be done following the next steps. There are newer versions of tensorflow available. This version will work properly for older computers and it has been tested with this setup.

```
    pip3 install --user --upgrade tensorflow==1.13.1  # install in \£HOME
```

To test if Tensorflow has been correctly installed the following can be done.

```
    python3
    >> import tensorflow as tf; print(tf.reduce_sum(tf.random.normal([1000, 1000])))
    # A sample tensor will be returned
```

Possible troubleshooting at this point: If different versions of python are required to correctly install the different packages. In case that python 3.7 is needed for this installation of Tensroflow the following commands can be implemented.

```
    sudo apt update
    sudo apt install python3.7
    sudo update-alternatives --install /usr/bin/python python /usr/bin/python3.5 1
    sudo update-alternatives --install /usr/bin/python python /usr/bin/python3.7 2
    sudo update-alternatives --config python #Select desired version
```

Its importan to take into consideration that having to many python versions installed directly onto the system can unset pip. It is recommended to stick with the default Ubuntu 16.04 Python version.
With this packages all the simulations can be played.

A.0.1. LIDAR setup and configuration

Different steps must be followed in order to install and set up all the packages required for using the LIDAR. This process can be complex as a lot of dependencies are needed along the way. This instructions worked for the set up in use.

The first step is to download the LeddarSDK. For this build a previous version (V 4.3.0.107) will be used to avoid compatibility issued. It can be download here `https://github.com/leddartech/LeddarSDK/releases/tag/4.3.0.107`.
Once the SDK has been download it can be unzipped (for this example) in the Desktop folder.
The Leddar Configurator 4 must be build.

```
cd Desktop/LeddarSDK/src
chmod +x build.sh
./build.sh
        #If the proces finishes succesfully it will show:
        #Building for architecture x64 in folder release
        #Done
```

A this point all the necessary dependencies have been installed into the system. The next step is to check is to configure the rules to allow ubuntu to detect the LIDAR via USB. The rules can be added as follows.

```
cd /etc/udev/rules.d
nano 90-LeddarTech.rules
#Add the Following line in the file-> ATTRS{idVendor}=="28f1",MODE="0666"
# Pres ctr + x to save the file
# Restart the udev subsystem to make sure the rule take effect
sudo udevadm trigger --action=change
```

To check that the computer is properly detecting the new USB.

```
lsubs
#....
# BUS 006 Device 007: ID 28f1:500
#....
```

To see in which port is connected the following command can be run connecting and disconnecting the Leddar. The permission for the connections can also be set up.

```
ls -l /dev/tty*
#....
# crw-rw-rw- 1 root dialout 166,  0 abr  6 21:04 /dev/ttyACM0
#....
```

Once everything is set up the first test example can be executed. This is a C++ scrip that does not require additional configuration.

```
cd Desktop/LeddarSDK/src/release
./LeddarExample
```

Leddar Python
The next step is to test the LIDAR using the Python code. This code allows for single measurements of the sensor. The first steps are installing the Python LIDAR extensions.

```
cd src/LeddarPy/
python setup.py install --user
#The following libary is mandatory for using with ROS
cd src/Py_leddar_utils/
python setup.py install --user
```

In this section some packages will be download and installed, internet connection is needed.
In order to run the code some modifications in the file *LeddarPyExample.py* have to be done. All the other type of sensor and reading methods have to be commented.

```
## Vu8 Sensor
    sensor_list = leddar.get_devices("Serial")
    dev.connect("/dev/ttyACM0", leddar.device_types["Serial"])
```

Once the file has been set up in can be execute as follows:

```
cd Desktop/LeddarSDK/src/LeddarPy
python LeddarPyExample.Py
```

Leddar ROS
To run the Leddar ROS packages the following instructions must be followed.
First the folder called *Leddar_ROS* in the SDK folder has to be moved into the Catkin Workspace under the name of *leddar_ros*. Then in a terminal:

```
cd ~/catkin_ws/
chmod +x src/leddar_ros/scripts/device.py
source /opt/ros/kinetic/setup.bash
source ./devel/setup.bash
    #Its not needed if it has been already implemented in the bash file.
catkin build
```

For launching the Rviz with the leddar module (assuming that all the previous Leddar steps have been followed correctly) it can be done as:

```
roslaunch leddar_ros example.launch param1:=/dev/ttyACM0 device_type:=Vu8 param3:=1 param4:=0
```

Possible LIDAR troubleshooting up to this point
When executing the different python scripts some error can appear. If the error mentions *"This support is currently experimental, and must be enabled with the -std=c++11 or -std=gnu++11 compiler options"* the following command solves the issue.

```
export CFLAGS='-std=c++11'
    #Execute befor runing "python setup.py ...."
```

This error can appear when configuration the setup.py of the LIDAR

A.0.2. Odroid Configuration

The configuration for the Odroid is very similar as the one for the ground station. The Ubuntu and ROS versions are the same. The compilation process for the ROS in the Odroid has historically been done using:

```
catkin_make # The previous compilation where done using catkin build
```

All the required ROS packages are already installed in the Odroid. An additionally installation of the Leddar ROS packages and Numpy ROS packages will need to be done.
The installation of the Leddar ROS SDK is the same as the one described for the ground station. But as the onboard CPU is an AMD chip the following modification to the bash.sh file must be done.

```
elif [[ \${MACHINE_TYPE} =~ arm.* ]]; then
    echo "Building for architecture ARM in folder release";
    make -j4 default CXXFLAGS="-std=c++11" LDFLAGS="-Wl,-rpath,'\$\$ORIGIN'
    /../../libs/FTDI/linux/ARM,-rpath,'\$\$ORIGIN'/../../libs/MPSSE/linux/ARM"
    builddir=release > compil_out.txt 2>&1
 else
    echo "Building for architecture x86 in folder release";
    make -j4 default CXXFLAGS="-m32 -std=c++11"
    CFLAGS=-m32 LDFLAGS="-m32 -Wl,-rpath,'\$\$ORIGIN'
    /../../libs/FTDI/linux/x86,-rpath,'\$\$ORIGIN'/../../libs/MPSSE/linux/x86"
    builddir=release > compil_out.txt 2>&1
```

This modification and the compiler option CXXFLAGS="-std=c++11" for the build in that system.
All the remaining steps from configuration the USB ports to installing the python dependencies are the same.
The LeedarTech installation in the Odroid will required internet connection. In this build the packages that normally will be download and installed by the *setup.py* where downloaded in another computer and installed separately. The LeddarTech installers will in due time tell the needed software and download location. As there are a lot of releases available for each software the full versions from around 2017 were used.

A.0.3. Possible overall troubleshooting up to point

The most common source of errors will be discussed in the following paragraphs.

ERROR 1:
If the system is unable to connect to its own server.

```
gedit ~/.bashrc
export ROS_HOSTNAME=localhost
export ROS_MASTER_URI=http://localhost:11311
```

ERROR 2:
If the gnome terminal wont open after installing different python versions the following can be done.
First one must move to one of the failsafe TTY terminals by pressing ctr + alt + f3 (exit the TTY by pressing ctr + alt + f7). Once there:

```
sudo nano /usr/bin/gnome-terminal
#Change -> !/usr/bin/python3
#To     -> !/usr/bin/python3.5
```

This will reestablish the default version of python (in this case python 3.5) for the terminal.

A.1. How to launch the different Nodes and Programs

The series of commands will launch the different nodes needed for the simulation. Each node has to be launched from a different terminal as it's an independent ROS node.

The simplest ones it the NLGL with ground truth. It requires the least amount of nodes and it is essays to execute.

```
roslaunch rotors_gazebo hummingbird_hovering_example_with_gps_sensor.launch
roslaunch platform_emulator platform_emulator_ground_truth.launch
roslaunch path_following_control nlgl_control.launch
rosrun platform_emulator start_exp
```

The DDPG algorithms can be launched as:

```
roslaunch rotors_gazebo hummingbird_hovering_example_with_gps_sensor.launch
roslaunch platform_emulator platform_emulator_ground_truth.launch
roslaunch path_following_control ddpg_control.launch
python DDPG/scrips/1/ddpg.py
rosrun platform_emulator start_exp
```

The DDPG with obstacle avoidance can be launch as

```
roslaunch rotors_gazebo hummingbird_hovering_example_with_gps_sensor.launch
roslaunch platform_emulator platform_emulator_ground_truth.launch
roslaunch path_following_control ddpg_control.launch
```

```
python DDPG/scrips/ddpg.py
python DDPG_OA ddpg_OA.py
rosrun platform_emulator start_exp
```

Its important that the same trained metadata (ddpg_model.meta, ddpg_model.index, ddpg_model.data i checkpoint) for the DDPG is also copied in the DDPG_OA folder. Using this methodology all the different launchers for each part of the code can be executed. As mentioned is important that the launch is sequential as some codes are related to previously launched programs.

B. Annex: Budget

In this section the cost of the project will be separated in its different parts.

B.1. Theoretical Costs

Table (table 4) presents the costs of the different components and materials to control and operate the UAV as it has been done in this study.

Item	Quantity	Unitary Cost (€)	Cost (€)
Hummingbird Kit	1	4000	4000
Hummingbird landing protection	1	15	15
Voltage Regulator	1	3	3
Batteries	4	31.5	126
Cables and accessories	1	20	20
Leddartech Sensor	1	733	733
Ground Station Computer	1	780	780
TOTAL	10	657.7	6577 €

Table 4: Cost of building a UAV

The total price for building the UAV is 6577 € per model. The aircraft and additional system are composed of 10 sub units with an average price of 657 € per element.

B.2. Additional Costs:

This segment presents the additional cost as transport or the external equipment. The resources used will mainly be broken down in hours employed, hourly cost or amortisation by hour and a total cost.

Item	Duration (hours)	Price per hour €/h	Price (€)
Car (amort of 0.0005%)	20	11	220
Transport by car or other means	20	4	80
Desktop computer (amort of 0.001%)	200	1.5	600
Total	-	-	900

Table 5: Additional costs table

The additional costs for this project (presented in table 5) taking into consideration just the most essential items make a sum of 900 €.

B.3. Technical Costs:

This section presents a brief broken down of all the hours employed for this project will be presented as well as all the additional material used and its amortisation.

Activity	Duration (hours)	Cost per hour (€/h)	Total Cost (€)
Project Charter	5	20	100
Previous Research	10	20	200
ROS and Ubuntu Set UP	50	20	1000
Gazebo Simulations I	20	20	400
LIDAR set Up and Configuration	25	20	500
LIDAR Integration	20	20	400
Hummingbird Modification	5	35	175
Gazebo Simulations II	50	20	1000
Flight Tests	25	25	625
Virtual LIDAR development	10	20	200
Code Development and Data Processing	15	20	400
Report elaboration and Analysis	65	20	1500
TOTAL	300	21.6	6500 €

Table 6: Table of technical costs

In table 6 the major parts of the project are presented. The price per hour varies due to the fact that the test conduced with the UAV are done outside with a progressive increase complexity as the hours done in the laboratory. Tot total average cost per hour is 21.6 € and the total technical cost of developing the project is 6500 €.

C. Annex: Python and Matlab codes

C.1. Python LIDAR Codes

In this project different Python ROS codes have been created in order to modify the existing DDPG_OA algorithms or to integrate the LIDAR system. In this section all the created codes will be briefly presented and explained. All the LIDAR programs were created inside a new ROS package called LIDAR, all the scrips needed to reed and publish the data form the sensor are there.

C.2. LIDAR CODES

In this section some of the final developed LIDAR codes will be presented. Code Method 1 was the first developed and despite being operational was abandoned. Code with method 2 present the second approach on reading the data from the sensor and publishing it. This was the code that worked better with the application.

C.2.1. Code Method 1:

```python
#!/usr/bin/env python


import rospy
import ros_numpy
import leddar
from std_msgs.msg import String
from sensor_msgs.msg import PointCloud2
import sensor_msgs.point_cloud2

def callback(data):
    rospy.loginfo(rospy.get_caller_id() + 'I heard %s', data.data)
    print(data) #Aqui ens imprimeix els valors originals del Pointcloud

    #xyz_array = ros_numpy.point_cloud2.get_xyz_points(data)
    #print(xyz_array)

    for point in sensor_msgs.point_cloud2.read_points(data, skip_nans=False):
            pt_x = point[0]
            pt_y = point[1]
            pt_z = point[2]

    print('X')
    print(pt_x)
    print('Y')
    print(pt_y)

    #xyz_array=sensor_msgs.point_cloud2.get_xyz_points(data, skip_nans=False)

    lidar_array=ros_numpy.point_cloud2.pointcloud2_to_array(data, squeeze=True)

    print('--------------')
    print(lidar_array)
    Ch8=lidar_array[0]
    Ch7=lidar_array[1]
    Ch6=lidar_array[2]
    Ch5=lidar_array[3]
    Ch4=lidar_array[4]
    Ch3=lidar_array[5]
    Ch2=lidar_array[6]
    Ch1=lidar_array[7]

    print('--------------')
    print('Ch8: ', Ch8[0])
    print('Ch7: ', Ch7[0])
    print('Ch6: ', Ch6[0])
    print('Ch5: ', Ch5[0])
    print('Ch4: ', Ch4[0])
    print('Ch3: ', Ch3[0])
    print('Ch2: ', Ch2[0])
    print('Ch1: ', Ch1[0])


def LIDARlistener():

    rospy.init_node('listenerLIDAR','LIDARtalker', anonymous=True)

    rospy.Subscriber('/LeddarTech_1/scan_cloud', PointCloud2 , callback)


    #spin() #simply keeps python from exiting until this node is stopped

if __name__ == '__main__':
    LIDARlistener()
    #LIDARtalker()
    rospy.spin()
```

C.2.2. Validation Publisher Code:

```
#!/usr/bin/env python
import rospy
from std_msgs.msg import String
from std_msgs.msg import Float64MultiArray
import numpy as np


def callback(data):
    print('---------------')
    print(data.data[0])
    print('Numbre of Measures')
    print(np.size(data.data))

def listener():


    rospy.init_node('listener', anonymous=True)

    rospy.Subscriber('LIDAR/talker', Float64MultiArray, callback)

    # spin() simply keeps python from exiting until this node is stopped
    rospy.spin()

if __name__ == '__main__':
    listener()
```

C.2.3. Code Method 2:

```python
#!/usr/bin/env python


import rospy
import ros_numpy
import leddar
import time
from std_msgs.msg import String
from std_msgs.msg import Float32
from std_msgs.msg import Float64
from sensor_msgs.msg import PointCloud2
import sensor_msgs.point_cloud2
import numpy as np
from sensor_msgs.msg import LaserScan
from std_msgs.msg import Int32MultiArray
from std_msgs.msg import Float64MultiArray


def echoes_callback(echoes):
    #data=[]
    data = echoes["data"]


    pub = rospy.Publisher('LIDAR/talker', Float64MultiArray, queue_size=100)
    #rate = rospy.Rate(10) # 10hz

    #print(data)
    #print(len(data))


    data0=float('inf')
    data1=float('inf')
    data2=float('inf')
    data3=float('inf')
    data4=float('inf')
    data5=float('inf')
    data6=float('inf')
    data7=float('inf')

    if len(data)==0:
        print('No estic llegint RES')

    else:
        for i in range (0,len(data)):
            #print(i)
            if data[i]["indices"] ==  7:
                data0=data[i]["distances"]
            elif data[i]["indices"] ==  6:
                data1=data[i]["distances"]
            elif data[i]["indices"] ==  5:
                data2=data[i]["distances"]
            elif data[i]["indices"] ==  4:
                data3=data[i]["distances"]
            elif data[i]["indices"] ==  3:
                data4=data[0]["distances"]
            elif data[i]["indices"] ==  2:
                data5=data[i]["distances"]
            elif data[i]["indices"] ==  1:
                data6=data[i]["distances"]
            elif data[i]["indices"] ==  0:
                data7=data[i]["distances"]



#    measures = ...
    [measure[0],measure[1],measure[2],measure[3],measure[4],measure[5],measure[6],measure[7]]
    measure = [data0,data1,data2,data3,data4,data5,data6,data7]

    #print(measure)

    data_to_sent=Float64MultiArray()
    data_to_sent.data=measure

    rospy.loginfo(data_to_sent)
    pub.publish(data_to_sent)
    rate.sleep()

def LIDARtalker():

    rospy.init_node('talker', anonymous=True)
    #rate = rospy.Rate(100) # 10hz

    while not rospy.is_shutdown():

        dev = leddar.Device()
        #Vu8 Sensor (Marcel)
        sensor_list = leddar.get_devices("Serial")
        dev.connect("/dev/ttyACM0", leddar.device_types["Serial"])

        dev.set_callback_echo(echoes_callback)
        dev.start_data_thread()
        time.sleep( 100 )
        #dev.stop_data_thread()



if __name__ == '__main__':
    LIDARtalker()
    rospy.spin()
```

C.2.4. Virtual LIDAR Codes

```
#!/usr/bin/env python


import rospy
import ros_numpy
import leddar
import time
from std_msgs.msg import String
from std_msgs.msg import Float32
from std_msgs.msg import Float64
from sensor_msgs.msg import PointCloud2
import sensor_msgs.point_cloud2
import numpy as np
from sensor_msgs.msg import LaserScan
from std_msgs.msg import Int32MultiArray
from std_msgs.msg import Float64MultiArray
from asctec_hl_comm.msg import mav_rcdata
```

```

potentiometer = 0

def RcDataCallback(msg):
    global joystick, potentiometer
    joystick = msg.channel[4]
    potentiometer = msg.channel[6]


def LIDARtalker():

    global time # Esta en decimes de segon
    time=0
    init=1
    pub = rospy.Publisher('LIDAR/talker', Float64MultiArray, queue_size=10)
    rospy.init_node('LIDARtalker', anonymous=True)
    rate = rospy.Rate(10) # 10hz
    rospy.Subscriber('fcu/rcdata', mav_rcdata, RcDataCallback)



    while not rospy.is_shutdown():

        if potentiometer≤3800 and potentiometer≥3100 and init==0:
            init=1

        if potentiometer≤3080 and init==1:
            time=0
            init=0


        if time≤20 and potentiometer≤3080:
            data0=float('inf')
            data1=float('inf')
            data2=float('inf')
            data3=float('inf')
            data4=float('inf')
            data5=float('inf')
            data6=float('inf')
            data7=float('inf')
        elif time>20 and time ≤70 and potentiometer≤3080:
            data0=float('inf')
            data1=float('inf')
            data2=float(2.1)
            data3=float(2.15)
            data4=float(2.2)
            data5=float('inf')
            data6=float('inf')
            data7=float('inf')
        else :
            data0=float('inf')
            data1=float('inf')
            data2=float('inf')
            data3=float('inf')
            data4=float('inf')
            data5=float('inf')
            data6=float('inf')
            data7=float('inf')



        print(time)
        print(potentiometer)


        if potentiometer≤3080:
            print('LIDAR ACTIVATED')
        elif potentiometer≤3800:
            print('LIDAR ARM')

        time=time+1
        measure = [data0,data1,data2,data3,data4,data5,data6,data7]
        data_to_sent=Float64MultiArray()
        data_to_sent.data=measure
        rospy.loginfo(data_to_sent)
        pub.publish(data_to_sent)
        rate.sleep()



if __name__ == '__main__':
    LIDARtalker()
    RcDataCallback()
    rospy.spin()
```

C.3. Matlab Codes

In order to visualize all the log data from the ROS flights a custom code was created to visualize all the data and review the validity of the flight. Different versions of the next code were created to plot different variables in different scenarios. The following code is one of the crated to plot the position data, LIDAR data and RC transmitter data.

```

%% CODI PER LLEGIR PROVES DE VOL

close all
clear all
clc

file='Exp_Marcel_3_2021-04-24-18-09-45.bag'; %VOLS AMB EL LIDAR REAL


bagInfo = rosbag('info',file)
bag=rosbag(file);

LIDAR=1; % Si Hi ha el LIDAR muntat

pose=select(bag,'Topic','/fcu/gps_position');
lidar_data=select(bag,'Topic','/LIDAR/talker');
fcu_status=select(bag,'Topic','/fcu/status');
RC_data=select(bag,'Topic','/fcu/rcdata');

pose_struc = readMessages(pose,'DataFormat','struct');
lidar_struc = readMessages(lidar_data,'DataFormat','struct');
fcu_status_struc = readMessages(fcu_status,'DataFormat','struct');
RC_struc = readMessages(RC_data,'DataFormat','struct');




k_pos=000; %MODIFIAR EL TOTAL DE PUNTS EN EL 3D
k_start=50; % MODIFICAR LINICI DEL DDPG
% TIME LIMIT
```

```
time_lim=60; %segons
time_init=0; %segons

length_pose=pose.NumMessages-k_pos;

%Per els grafics 3D que no es poden limitar i shan de borrar dades
for j=1:length_pose
    xPoints_tot(j) = pose_struc{j}.Position.X;
    yPoints_tot(j) = pose_struc{j}.Position.Y;
    zPoints_tot(j) =pose_struc{j}.Position.Z;

end

% Per els grafics respecte el temps que es poden limitar amb el temps
for j=1:pose.NumMessages
    xPoints(j) = pose_struc{j}.Position.X;
    yPoints(j) = pose_struc{j}.Position.Y;
    zPoints(j) =pose_struc{j}.Position.Z;

end

    if LIDAR == true

    for j=1:lidar_data.NumMessages

        Ch8(j) = lidar_struc{j}.Data(1);
        Ch7(j) = lidar_struc{j}.Data(2);
        Ch6(j) = lidar_struc{j}.Data(3);
        Ch5(j) = lidar_struc{j}.Data(4);
        Ch4(j) = lidar_struc{j}.Data(5);
        Ch3(j) = lidar_struc{j}.Data(6);
        Ch2(j) = lidar_struc{j}.Data(7);
        Ch1(j) = lidar_struc{j}.Data(8);
        %Ch1(j) = lidar_struc{j}.Ranges(8);
    end
         %Create time axis lidar
        t_i=lidar_data.StartTime;
        t_f=lidar_data.EndTime;
        num_mesg=lidar_data.NumMessages;
        Delta_t=(t_f-t_i)/num_mesg;
        Total_TIME=t_f-t_i
        time_lidar(1)=0;
        for i=2:num_mesg
           time_lidar(i)= time_lidar(i-1)+Delta_t;
        end


    end



for j=1:RC_data.NumMessages
    RC_channel(j) = RC_struc{j}.Channel(7);
end

%Calcular a quina posicio esta el switch que ens indica si es drone esta
%treballant amb un mode o amb un altre


for j=1:RC_data.NumMessages
    if (RC_channel(j)<4000)
    RC_channel_Auto(j) = true;
    else
    RC_channel_Auto(j) = false;
    end
end


    %Create time axis RC
    t_i=RC_data.StartTime;
    t_f=RC_data.EndTime;
    num_mesg=RC_data.NumMessages;
    Delta_t=(t_f-t_i)/num_mesg;
    Total_TIME=t_f-t_i;
    time_RC(1)=0;
    for i=2:num_mesg
       time_RC(i)=time_RC(i-1)+Delta_t;
    end




    %create time axis drone
    t_i=pose.StartTime;
    t_f=pose.EndTime;
    num_mesg=pose.NumMessages;
    Delta_t=(t_f-t_i)/num_mesg;
    time_pose(1)=0;
    for i=2:num_mesg
       time_pose(i)= time_pose(i-1)+Delta_t;
    end


      %creat cylinder
    [X,Y,Z]=cylinder(1);
    x=5;    %Location of the cylinder
    y=0;
    h=2;


    figure(1)
    plot3(xPoints_tot,yPoints_tot,zPoints_tot,'c','LineWidth',4)
    hold on
    scatter3(xPoints_tot(1),yPoints_tot(1),zPoints_tot(1),200,'r','filled')
    hold on
    scatter3(xPoints_tot(k_start),yPoints_tot(k_start),zPoints_tot(k_start),200,'g','filled')
    hold on
    % PER SI ES VOL PLOTEJAR LA DETECCIO VIRTUAL

    n=-0.25;
    for m=1:0.1:2
       scatter3(-9,-18+m,0,'r')
        n=n+0.05;
    end
    hold on

     n=-0.25;
    for m=1:0.1:2
       scatter3(-18-n,-21+m,0,'r')
        n=n+0.05;
    end
    hold on


    %surf(X+x,Y+y,Z*h) % SI ES VOL MOSTRAR CILINDRE
    axis equal
```

```
        title('Straight Path with obstacle Aboidance')
        xlabel('X (m)')
        ylabel('Y (m)')
        zlabel('Z (m)')
        grid


         %Position X vs Time
         figure(2)
         plot(time_pose,xPoints)
         xlim([time_init time_lim])
         title('DDPG-OA X position vs Time', 'DisplayName', 'Position x')
         xlabel('Time (s)')
         ylabel('X (m)')
         legend()
         grid on

         %Position Y vs Time
         figure(3)
         plot(time_pose,yPoints, 'DisplayName', 'Position Y')
         xlim([time_init time_lim])
         title('DDPG-OA: Y position vs Time')
         xlabel('Time (s)')
         ylabel('Y (m)')
         legend()
         grid on

        if LIDAR == true


             %Position Y and X vs Lidar vs Time
        figure(4)
        plot(time_pose,yPoints,'DisplayName', 'Position Y','LineWidth',2);
        yyaxis right
        ylabel('X (m)')
        hold on
        plot(time_pose,xPoints, 'DisplayName', 'Position X');
        yyaxis left
        ylabel('Y (m)')
        hold on
        plot(time_lidar,Ch4,'DisplayName', 'Chanel 4','color','g','LineWidth',2)
        hold on
        plot(time_lidar,Ch5,'DisplayName', 'Chanel 5','color','#ad20b8','LineWidth',2)
        hold on
        plot(time_lidar,Ch6,'DisplayName', 'Chanel 6','color','k','LineWidth',2)
        xlim([time_init time_lim]) %Per Limitar Aixi si ens Interessa veure mes o menys temps
        title('DDPG-OA: Y & LIDAR position vs Time')
        set(gca,'XMinorTick','on','YMinorTick','on')
        legend()
        xlabel('Time (s)')
        grid on


         %Position Y and X vs Lidar vs Time DETALLAT A ZONA CONTROL AUTO

         T_i = find(RC_channel_Auto, 1, 'first');
         T_f = find(RC_channel_Auto, 1, 'last');

        figure(6)
        plot(time_lidar,Ch8,'DisplayName', 'Chanel 8','color','g','LineWidth',1.5)
        hold on
        plot(time_lidar,Ch7,'DisplayName', 'Chanel 7','color','y','LineWidth',1.5)
        hold on
```

```
    plot(time_lidar,Ch6,'DisplayName', 'Chanel 6','color','r','LineWidth',1.5)
    hold on
    plot(time_lidar,Ch5,'DisplayName', 'Chanel 5','color','b','LineWidth',1.5)
    hold on
    plot(time_lidar,Ch4,'DisplayName', 'Chanel 4','color','k','LineWidth',1.5)
    hold on
    plot(time_lidar,Ch3,'DisplayName', 'Chanel 3','color','m','LineWidth',1.5)
    hold on
    plot(time_lidar,Ch2,'DisplayName', 'Chanel 2','color','c','LineWidth',1.5)
    hold on
    plot(time_lidar,Ch1,'DisplayName', 'Chanel 1','color','#ad20b8','LineWidth',1.5)
    hold on
    xlim([time_init time_lim])
    ylim([0 15]) %LIMITACI DELS EIXOS PELS PLOTS DEL LIDAR
    grid on
    legend()
    title('LIDAR measurements')
    xlabel('Time (s)')
    ylabel('Distances (m)')

    figure(5)
    plot(time_pose,yPoints,'DisplayName', 'Position Y','LineWidth',2);
    yyaxis right
    ylabel('X (m)')
    hold on
    plot(time_pose,xPoints, 'DisplayName', 'Position X');
    yyaxis left
    ylabel('Y (m)')
    hold on
    plot(time_lidar,Ch4,'DisplayName', 'Chanel 4','color','g','LineWidth',2)
    hold on
    plot(time_lidar,Ch5,'DisplayName', 'Chanel 5','color','#ad20b8','LineWidth',2)
    hold on
    plot(time_lidar,Ch6,'DisplayName', 'Chanel 6','color','k','LineWidth',2)
    xlim([time_RC(T_i) time_RC(T_f)]) %Per Limitar Aix  si ens Interessa veure mes o menys temps
    title('DDPG-OA: Y & LIDAR position vs Time')
    set(gca,'XMinorTick','on','YMinorTick','on')
    legend()
    xlabel('Time (s)')
    grid on

    end

    figure(7)
    plot(time_RC,RC_channel_Auto)
    title('Autonomus Mode')
    xlabel('Time (s)')
    ylabel('Autonomus Mode (m)')


    figure(8)
    plot(time_RC,RC_channel)
    title('RC Switch Imput')
    xlabel('Time (s)')
    ylabel('Autonomus Mode (m)')


    figure(9)
    subplot(3,1,1)
    bar(time_lidar,Ch8,'g','LineWidth',1.5)
    hold on
    bar(time_lidar,Ch7,'y','LineWidth',1.5)
    xlabel('Time (s)')
```

```
ylabel('Right Channels (m)')

subplot(3,1,2)
bar(time_lidar,Ch6,'r','LineWidth',1.5)
hold on
bar(time_lidar,Ch5,'b','LineWidth',1.5)
hold on
bar(time_lidar,Ch4,'k','LineWidth',1.5)
xlabel('Time (s)')
ylabel('Center Chalens (m)')

subplot(3,1,3)
bar(time_lidar,Ch3,'m','LineWidth',1.5)
hold on
bar(time_lidar,Ch2,'c','LineWidth',1.5)
hold on
bar(time_lidar,Ch1,'g','LineWidth',1.5)
xlabel('Time (s)')
ylabel('Left Chanels (m)')

sgtitle('LIDAR detection separated by central, left and right regions')
```

D. Annex: ROS Topics

D.1. NLGL Simulation Topics

The topics involved in the NLGL Hummingbird simulation are listed below.

```
/altitude_ctrl/uz
/attitude_ctrl/euler
/clock
/fcu/control
/fcu/gps_custom
/fcu/gps_pose
/fcu/imu
/fcu/imu_custom
/fcu/motor_speed
/fcu/rcdata
/gazebo/link_states
/gazebo/model_states
/gazebo/parameter_descriptions
/gazebo/parameter_updates
/gazebo/set_link_state
/gazebo/set_model_state
/gazebo_gui/parameter_descriptions
/gazebo_gui/parameter_updates
/hummingbird/command/motor_speed
/hummingbird/gazebo/command/motor_speed
/hummingbird/gps_sensor/gps
/hummingbird/gps_sensor/ground_speed
/hummingbird/ground_truth/imu
/hummingbird/ground_truth/odometry
/hummingbird/ground_truth/pose
/hummingbird/ground_truth/pose_with_covariance
/hummingbird/ground_truth/position
/hummingbird/ground_truth/transform
/hummingbird/imu
/hummingbird/joint_states
/hummingbird/motor_speed
/hummingbird/motor_speed/0
/hummingbird/motor_speed/1
/hummingbird/motor_speed/2
/hummingbird/motor_speed/3
/hummingbird/odometry_sensor1/odometry
/hummingbird/odometry_sensor1/pose
/hummingbird/odometry_sensor1/pose_with_covariance
/hummingbird/odometry_sensor1/position
/hummingbird/odometry_sensor1/transform
/hummingbird/wind_speed
/nlgl_ctrl/behaviour
/nlgl_ctrl/gamma_info
/nlgl_ctrl/gamma_time
/nlgl_ctrl/vel_psi_cmd
/nlgl_ctrl/z_cmd
/rosout
/rosout_agg
/statistics
/tf
/tf_static
/vel_ctrl/attitude_cmd
```

D.2. DDPG OA Flight test Topics

The topics used during the DDPG OA test are presented below.

```
/LIDAR/talker
/altitude_ctrl/uz
/attitude_ctrl/euler
/ddpg_avoidance/status_and_state
/ddpg_ctrl/vel_psi_cmd
/ddpg_ctrl/z_cmd
/diagnostics
/fcu/control
/fcu/current_pose
/fcu/debug
/fcu/ekf_state_in
/fcu/ekf_state_out
/fcu/fcu/parameter_descriptions
/fcu/fcu/parameter_updates
/fcu/gps
/fcu/gps_custom
/fcu/gps_pose
/fcu/imu
/fcu/imu_custom
/fcu/mag
/fcu/motor_speed
/fcu/odometry
/fcu/pose
/fcu/rcdata
/fcu/ssdk/parameter_descriptions
/fcu/ssdk/parameter_updates
/fcu/state
/fcu/status
/rosout
/rosout_agg
/tf
/tf_static
/vel_ctrl/attitude_cmd
```

E. Bibliography

References

[1] Dr. Kostas Alexis. Rotors simulator. http://www.kostasalexis.com/rotors-simulator.html.

[2] Jon Binney. ros_numpy: point_cloud2.py Source File, 2008.

[3] DJI. DJI Air 2S - Todo en uno - DJI.

[4] Easy-tensorflow.com. Easy TensorFlow - Install TensorFlow.

[5] GitHub. Various GitHub Repositories and Forums, url = https://github.com/, urldate = 2021-04-20.

[6] Ascending Technologies Gmbh. AscTec Hummingbird with AutoPilot User's Manual - Ascending Technologies GmbH. pages 1–28.

[7] Farid Kendoul. Survey of advances in guidance, navigation, and control of unmanned rotorcraft systems. *JOURNAL OF FIELD ROBOTICS*, page 29(2):315–378, 2012.

[8] LeddarTech. Leddartech sdk 4. https://sdk.leddartech.com/v4.3/#/.

[9] LeddarTech. LeddarVu 8-Segment Solid-State LiDAR Module - User Guide.

[10] Multiples Authors. ROS/Concepts - ROS Wiki, 2014.

[11] Bartomeu Perelló Rubí. *GUIDANCE, NAVIGATION AND CONTROL OF MULTIROTORS*. PhD thesis, Universitat Politècnica de Catalunya, 2020.

[12] Roy Rob and Bommakanti Venkat. ODROID-XU4 Manual.

[13] McGill Robotics. Ros leddar. https://github.com/mcgill-robotics/ros-leddar.

[14] Robot simulation. Gazebo. http://gazebosim.org/.

[15] Mark Sinton. Deep deterministic policy gradients (ddpg). https://github.com/msinto93/DDPG.

[16] Skydio. Skydio 2 Autonomous Drone - Skydio Inc. — Skydio.

[17] Tensorflow.org. Install TensorFlow with pip.