Escola d'Enginyeria de Telecomunicació i Aeroespacial de Castelldefels
UNIVERSITAT POLITÈCNICA DE CATALUNYA BARCELONATECH

TREBALL FINAL DE GRAU

TÍTOL DEL TFG: Sistema d'ajuda multisensor a la navegació per drons multiplataforma

TITULACIÓ: Grau en Enginyeria de Sistemes Aeroespacials

AUTORA: Laura Parga Gata

DIRECTOR: Sergi Tres Martínez

SUPERVISOR: Oscar Casas Piedrafita

DATA: 15/11/2020

Títol: Sistema d'ajuda multisensor a la navegació per drons multiplataforma

Autora: Laura Parga Gata

Director: Sergi Tres Martínez

Supervisor: Oscar Casas Piedrafita

Data: 15/11/2020

Resum

Actualment, i des de principis del mil·lenni, la tecnologia dron es troba en un continu creixement, a partir del qual s'han anat desenvolupant diferents tècniques que la fan més accessible i fàcil d'utilitzar en una gran varietat d'àmbits de treball, així com l'arqueologia. En aquest camp es desenvolupa una nova proposta de projecte per part d'Hemav Foundation, en la qual es pretén utilitzar un dron i tècniques de processat de imatge per substituir la tradicional tècnica de detecció de ceràmiques i l'anàlisi posterior. Per fer-ho, el vehicle ha de portar incorporada una càmera multiespectral i ha de volar l'àrea a estudiar de manera autònoma amb ajuda d'un GPS, és a dir, sense ser pilotat, per la qual cosa necessita ajudes a la navegació. L'altitud de vol ha de ser suficientment baixa com per tenir una bona resolució i ha de mantenir-se sempre constant per no afectar a la relació àrea-píxel de les imatges. Tenint en compte això, es necessiten dos sistemes d'ajuda a la navegació: un per mantenir l'altitud constant i seguir estrictament la forma que tingui el terreny sobre el que es vola i un altre per detectar i evitar obstacles, donat que a baixes altituds hi ha més probabilitat de col·lisió.

L'objectiu d'aquest projecte és el disseny i la implementació d'un sistema de manutenció d'altitud i un sistema de detecció i evitació d'obstacles.

Primer es duu a terme un estudi de mercat en què s'analitzen tant les diferents metodologies amb les que es pot mesurar distància com els sensors que ofereix el mercat actualment, per finalment triar un per cadascun dels sistemes que s'han de dissenyar. Donada la situació actual, no s'ha pogut utilitzar el sensor que s'ha considerat més apropiat per utilitzar d'altímetre, que era el LIDAR-Lite v3, però s'ha pogut aconseguir el TF mini. Per la detecció d'obstacles s'ha triat la càmera D435. Després d'això, es fa el disseny dels sistemes. Per una banda, en quant a l'arquitectura hardware: connexió de la càmera a l'ordinador a bord, que és una Raspberry Pi 4 model b, i del sensor a la controladora, la Pixhawk Cube. Per una altra banda, en quant al software: triar un llenguatge de programació, que serà Python, descarregar i instal·lar les llibreries necessàries per l'obtenció de dades (SDK de Intel RealSense per la càmera) i per la comunicació (MAVLink i DroneKit) i, finalment, programar el codi. Una vegada realitzat el disseny dels sistemes, s'ha de testejar tot: el sensor i la càmera per comprovar les especificacions i el codi per comprovar el funcionament.

Per una banda, el sensor té bona precisió però triga uns segons en estabilitzar-se i el rang és insuficient. Per una altra banda, la càmera funciona molt bé, ja que detecta obstacles de diferents formes i materials a diferents distàncies. Finalment, el codi funciona prou bé, encara que es podria millorar l’eficiència.

En resum, el disseny dels dos sistemes es pot implementar i utilitzar satisfactòriament en el projecte si es canvia el sensor que mesura altitud per un amb un rang màxim més alt, millor precisió i més estable en la mesura.

Title: Multisensory help system to multiplatform drone navigation

Author: Laura Parga Gata

Advisor: Sergi Tres Martínez

Supervisor: Oscar Casas Piedrafita

Date: 15/11/2020

Overview

Currently, and since the beginning of the millennium, drone technology is in a continuous growth, from which different techniques have been developed making it more accessible and easy to use in a variety of work areas, such as archaeology. In this field, a new project proposal is being developed by Hemav Foundation, in which is intended to used drones and image processing techniques to substitute the traditional pottery detection method and the subsequent analysis. In order to do it, the vehicle must carry on a multispectral camera and fly the study area in an autonomous way with the help of a GPS, which means, with no pilot, and so it needs some navigation helps. Flight altitude of the mission should be low enough to get high-resolution images and should keep it constant to not influence the area-pixel ratio. With this in mind, two navigation help systems are needed: one in order to keep altitude constant and follow strictly the terrain shape it is flying and another one in order to detect and avoid obstacles due to low altitudes imply more collision probability.

The aim of this work is the design and the implementation of an altitude maintenance system and an obstacle detection and avoidance system.

First a market study is done, in which both the methodologies that can be used and the devices that are available currently in the market are analysed, in order to finally choose which are the more suitable ones for the two systems in design. Given the actual situation, the sensor that was selected to measure altitude, the LIDAR-Lite v3, wasn’t able to use and so another one has been used: TF mini, which specifications are clearly worse than the other ones. For obstacle detection camera D435 has been used. After that, the systems design is done. On one hand, the hardware architecture: connection between camera and flight controller, Pixhawk Cube. On the other hand, the software: choose a programming language, which will be Python, download and install the required libraries for obtaining data (Intel RealSense SDK) and for communicating with the drone and between internal components (MAVLink and DroneKit) and, finally, to code the program. After all, some test has to be done: first the sensor and the camera to check the specifications and the code to check the behaviour.

On one hand, the sensor has good accuracy but it takes some seconds to stabilize and the maximum range is not enough. On the other hand, the camera works so well: it detects obstacles of different shapes and materials and situated in different locations. Finally, the code works, although it could be improved in terms of efficiency.

Summarizing, both systems design can be implemented and used in the project if the altitude sensor is changed by another one with higher maximum range, lower accuracy and a more stable behaviour.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

The technological development of the XXI century has allowed a renewable drone sector, both in design and manufacturing as in their possible applications. Hence, this sector has been, and still now is, through a continuous growth in a global level where the point is the innovation; just due to the fact of reinventing itself is the triggering of this expansion. In addition to the increase in the technical capabilities, UAS (Unmanned Aerial System) technology has become cheaper during this process.

Consequently, companies have seen a great chance to try a business in this sector so there have been a significant increase in the number of new business related with the manufacturing of drones, in a global market, as Fig. 1.1. shows.

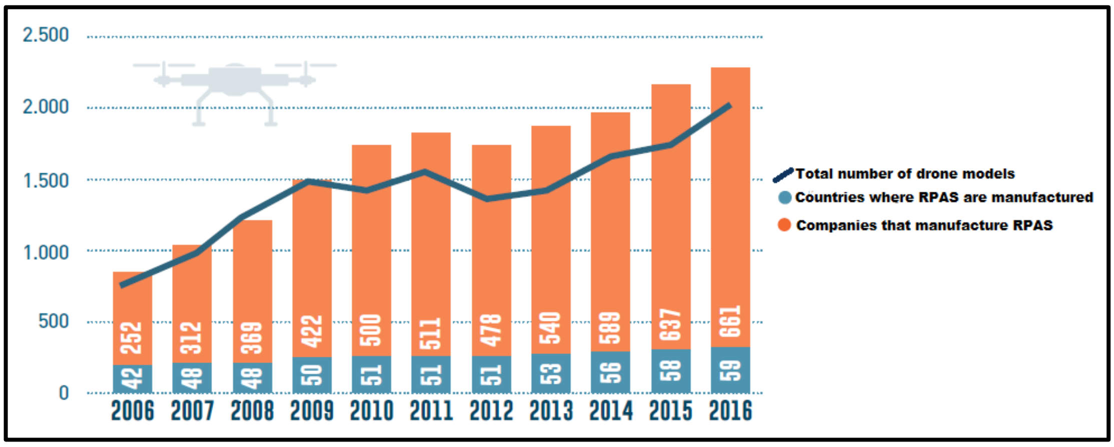


Fig. 1.1. Drone sector evolution worldwide [1]

In the case of Hemav Foundation [2], a non-profit private organisation, it uses this technology in order to boost projects with an environmental or a social impact, being this last one both helping those most in need and spreading the use of drone technology.

Currently Hemav Foundation is developing a new proposal for a project related with the visual pottery research using drones and machine learning, substituting the traditional archaeological method known as “fieldwalking”. This is not an innovating project, since it comes from a test of two researchers: Dr. Hector A. Orengo of the Catalan Institute of Classical Archaeology and Dr. Arnau Garcia-Molsosa of the McDonald Institute for Archaeological Research at the University of Cambridge, that tried this way of making the detection and the analysis of potsherds easier for archaeologists. Their idea was to detect pottery fragments using high-resolution drone imagery, photogrammetry and a combination of machine learning and geospatial analysis, in order to get better efficiency in the task. As a result of this test, they conclude it can be a reliable method.

Hemav Foundation new project intends to improve this idea using a drone following a preprogramed route by GPS navigation, which is, flying automatically without human help. Given the characteristics of the mission, the flight altitude is set according to the area needed in one pixel, and so it is important to maintain it constant during the entire path, which means the drone has always to follow the terrain it is flying. Furthermore, this flight altitude has to be lower enough to obtain high-resolution images, which makes probable the appearance of obstacles during the path, so the drone has to be able to detect and avoid them. As the drone will not be remotely piloted, this two mentioned points should be properly designed and implemented, which is the objective of this work.

1.1. Objectives

The aim of this work is to design, implement and proof two autonomous-drone navigation systems: one in order to maintain the flight altitude strictly constant (following the terrain) and the other in order to detect and avoid obstacles automatically, both in hardware and software. More specifically, the first objective is to find the most suitable distance measurement devices and so to detect obstacles and to measure height properly, which means distance to soil. The next target is to design the obstacle avoidance mission. Finally, the last mark is to be able to implement the designed code containing both subsystems (altitude maintenance and obstacle detection and avoidance) in the drone system.

1.2. Project distribution

This project distribution is done as the design process of any system, beginning with the state of art and ending with tests. In chapter two, the use of drones in archaeology situation is analysed and then a market study is done: first the different ways of measuring distance and then the market study about distance measurement devices, in order to choose the most suitable one. Henceforth the system design can be done, dividing it in the hardware architecture and the software implementation, which are the next two chapters. One on hand, in the hardware part it is important first to take into account the drone characteristics and elements that it carries, as the flight controller and the processors and then the previously chosen devices can be connected. On the other hand, in the software part it is needed to set a programming language, know how hardware elements communicate between them and look for the needed libraries in order to do so (compatibles with the chosen language). Finally, some experimental tests are done in order to determine the specifications of the systems.

2. STATE OF THE ART

During this chapter, the market situation is analysed. On one hand, regarding to the general project where the navigation systems will be implemented, the use of drones in archaeological surveys is studied, including benefits and drawbacks. On the other hand, taking into account that the objective of this work is to design and implement the altitude maintenance and the obstacle detection and avoidance navigation systems, a market study on distance measurement devices is done. First, analysing and comparing the main distance measurement methodologies, according to parameters like precision and range, and then, taking into account these different methods, a concrete distance measurement sensor is finally selected for each one of the systems designed, which means to select two: one to detect obstacles and another to measure the actual altitude.

2.1. Drones and archaeology

Since drone technology is relatively new, it is not widely used for archaeological purposes and so there is still a lack of market related with this combination: drones and archaeology. As it has been said during the introduction, the use of this technology is suffering a continuous growth in the last recent years, in which some fields are being studied, including archaeology, where drones are useful to increase the efficiency of the mission. This increase of efficiency has always been the main objective of the use of UAS by saving time and/or decreasing the number of workers in a mission and so doing more than a mission at the same time. The applications of this technology in this field began some years ago by monitoring the terrain, which means getting 3D models and digital cartography, in order to plan an action for the mission to do. It has been widely used to create digital images by 3D recording of excavations and historic monument. Actually, the main use of drones in archaeology is to do a photogrammetric survey of landscapes for the creation of digital surface and terrain. Digital photogrammetry software so is the most important technology on archaeological researches, regarding with drones, such as it has become much powerful, easy to use and accessible due to new developed techniques.

There is an article where the test of the Spanish and English researchers previously mentioned [3], in which is pretended to substitute the “fieldwalking” archaeological method by an innovating one using high-resolution drone imagery, photogrammetry and a combination of geospatial analysis and machine learning, is analysed in detail. Regarding to the traditional method, it consists of an archaeological researcher’s team walking a given area following parallel lines between them and searching pottery in a visual way. During this scanning, they count all ceramic fragments and collect only those that can provide chronological and/or typological information about human present. These potsherds then go through a process of measurement and analysis of the material, among other things, from which the data about human presence is extracted. Regarding to the automated method, it is divided into two parts. First, a drone scans a given area with a high-resolution multispectral camera, which obtains detailed images of the terrain to analyse. After that, geospatial analysis such as machine learning are done to identify, isolate and analyse ceramic fragments.

There are some points explained during the article that are important to take into account. Regarding to the image acquisition, the multispectral camera to use must be a high-resolution one, which could lead to detect smaller fragments, and the area to analyse should be ideally in good light conditions, with the presence of shadows, in ploughed soils and in sedimentary plains in order to obtain best detection rates. All these characteristics are the ideal ones, although it is known it does not depend on the archaeologists, but depending on it (soil, environment or light) different training data can be applied due to different conditions imply different processing. It also happens with the pottery colour. In order to detect different types of potsherds, it is convenient to have and apply different machine learning processes. Regarding to the image analysis, large computational resources, good knowledge of machine learning and good experience in training data engineering are required. Even then, important information such as pottery date or function cannot be extracted using this method, because in order to know it the specialist should go to where the potsherd was detected, pick it up and analyse herself/himself. There is a big trade-off, important in the design of the image processing: having false positives or achieving lower rates of detection. Depending on the point of view, it is recommended to detect a higher number of fragments but having also more false positives or to detect with more precision, which means less false positives, but discarding some pottery fragments.

This test was conducted both using the traditional and this innovating techniques and one of the conclusions was that the automated recording method is able to detect more ceramic fragments and faster than the standard one. The most difficult part of this technique is the analysis, where all images taken during the flight have to be processed one by one, so it is also the part that require most of the time, that will be more than 10h. It is also important to highlight that the main benefit is that it can be performed by a single person with minimal investment of time due to all the process is done automated.

Summarizing, in terms of time both methods can take more or less the same, while the increase of efficiency in the case of the automated technique is notable, by saving specialists working time and detecting a higher number of pottery fragments. The main drawback is the difficulty of the image processing. Although the technology is increasingly developed, a team of informatics specialists experienced in machine learning and training data engineering are needed. It is also important to highlight that this technology can lead to false positives, which means to detect different objects, even only shapes, as if it were potsherds.

2.2. Distance measurement technologies

This section is also divided according the main technologies can be found in the different distance measurement sectors that currently are available on the market [4], [5], [6]. All of them use the ToF (Time of Flight) or triangulation methods, but with different wave types: ultrasonic, infrared or LASER.

In the time of flight methodology case, the sensor emitter sends a signal and the time taken to come back to the receiver is measured, which means the time this signal has been travelling to the closer obstacle and return. Knowing all types of waves travel at speed of light ($\mathrm{c} = 3 \cdot 10^8\ \mathrm{m/s}$), the distance between the device and the closer object (d) can be easily compiled by using uniform rectilinear movement equation with the measured time ($\Delta\mathrm{t}$) and the speed of light.

$$\mathrm{d} = \frac{\Delta \mathrm{t}}{2}\,\mathrm{c} \qquad \textbf{(2.1.1.)}$$

In the triangulation case, the sensor consists of two separate modules, emitter and receiver, by a known distance ($\mathrm{d}_{\mathrm{modules}}$). This time the receiver measures the angle of incidence ($\alpha_{\mathrm{incidence}}$) of the transmitted signal reflection and so the distance to the closer obstacle (d) can be found by using trigonometry, particularly, the tangent definition.

$$\mathrm{d} = \frac{\mathrm{d}_{\mathrm{modules}}}{\tan(\alpha_{\mathrm{incidence}})} \qquad \textbf{(2.1.2.)}$$

It is important to take into account that the next distance measurement methodologies are the principal ones, but there are some sensors that use a combination of them.

2.2.1. Ultrasonic

An ultrasonic wave is a vibration transmitted through a medium at a frequency bigger than 20 kHz. Ultrasonic distance sensors use this type of sound waves, which can be inferred by acoustic signals, and time of flight principle.

The main benefits are the low current they consume (and so power), the multiple interface options and, as sound waves are used, the result is not affected by target colour or transparency. Although their advantages, these devices are not always useful because of their limited range, their low resolution, their slow refresh rate, which could not be appropriate to detect fast moving targets, and the incapability of detect objects with extreme textures or surfaces.

2.2.2. Infrared

Infrared radiation is a type of electromagnetic radiation whose wavelength is bigger than the visible but smaller than the microwave one. Infrared distance sensors are also known as LEDs (Light-Emitting Diodes) distance sensors and can use both time of flight and triangulation methodologies. Their main advantages are that can work in day-time and night-time usages, their lightweight and compact form.

In case of ToF technique, there are many benefits: can measure precisely due to the rapid refresh rates and long-ranges, are easy to use, facilitates multi-sensor integration and can generate 3D imaging. However, the depth (Z-axis) resolution is poor and is not recommended to outdoor missions because can be affected by sunlight. In case of triangulation principle, they are usually the cheapest ones but offer short-range and are not reliable with additional sensors.

2.2.3. LIDAR

LASER, that stands for Light Amplification by Stimulated Emission of Radiation, is an artificial type of light that produce a narrow beam made of different waves all lined up, in peak or phase, and that have similar wavelengths. LIDAR (Light Detection and Ranging) devices are distance sensors that use a LASER source with integrated optics that measure the time of flight of a transmitted signal.

This type of sensor can work a long range giving precise and stable measures with high accuracy and a fast update rate and can also be used both in night and day time, but are not safe to human eyes and are not reliable when integrated with other sensors. Their measures can also generate 3D structures. This type of distance sensors are considered the best detecting objects outdoors and are the ones with smaller wavelength so also the best detecting small targets, but are the ones with higher cost and higher current consumption and so power consumption.

2.2.4. Choice

Given the different distance measurement technologies and their advantages and disadvantages, it is time to analyse and compare them in order to choose the more suitable one for these objectives, considering the project characteristics in which will be implemented these two navigation systems. The following table summarizes what the previous sections have explained.

Table 2.1.4.1. Distance measurement technologies comparison

Technology	Principle	Advantages	Disadvantages
Ultrasonic	Time of flight	· Low current draw and so low power consumption · Not affected by object colour and transparency · More than one interface options	· Low resolution · Slow refresh rate · Limited range · Interfered by acoustic signals · Not detecting objects with extreme textures and surfaces
Infrared	Triangulation	· Compact · Lightweight · Low-cost · Day & night	· Short range · Low reliability with additional sensors
	Time of flight	· Compact · Lightweight · Easy to use · Reliability with additional sensors · Long range · Fast refresh rates · High precision · 3D imaging · Day & night	· Affected by sunlight · Z-axis poor resolution
LIDAR	Time of flight	· Long and high range · High accuracy · 3D imagery · Fast refresh range · Detection of small objects · Day & night · High precision · Stability in measures · Good detection outdoors	· High cost · Dangerous to human eyes · Low reliability with additional sensors · High current draw and so high power consumption

First, it is important to take into account that the drone will always fly outdoors. On one hand, in the altitude maintenance system it is no needed a wide and high range since the drone will fly always at relatively low altitudes in order to have good resolution in images but it should be a precise distance measurement sensor for the area-pixel rate to be constant. On the other hand, in the obstacle avoiding system the maximum range should be bigger in order to detect obstacles with anticipation and so avoid them, but neither a hundred of meters.

Precision is a one of the most important requirements and, as it can be seen in the comparison table, LIDAR sensors and the ones working with infrared waves and time of flight principle provide high precision. In order to choose between these two alternatives, is time to take into account that pottery research will be always outdoors, where LIDAR sensors provide good detection while IR + ToF sensors detection can be affected under the existence of sunlight. So, is easy to conclude that the most suitable for this work is to use a LIDAR distance measurement sensor.

2.3. Distance measurement sensors

The market research on LIDAR sensors starts on looking in the typical on-line stores specialized in selling products for electronic use, which includes a large variety of sensors, including distance measurement sensors.

One of the characteristics of LIDAR is that they can have a wide range, so this is the first filter to take into account: discarding the long-range ones (e.g. more than 50 m of maximum range) since as it has been previously justified it will never be needed to measure a large distance.

Digging deeper about different sensors, a line-up of cameras able to measure distances appeared, which are the Intel RealSense depth cameras. The main advantage of using this format instead of a sensor is that the distance is measured pixel by pixel, which will be helpful in the case of obstacle avoidance system, to detect objects.

During the following sections, this cameras line-up and some LIDAR sensors are analysed and compared in order to finally choose one distance measurement device for each system.

2.3.1. Obstacle avoidance

In order to detect obstacles, and then avoid them, one of the Intel RealSense depth cameras [7] is used. In this section, all of them are analysed and compared to choose the most suitable one.

The main difference of this line-up with respect to previously explained sensors is the presence of two receivers, located at a constant distance of the emitter. Concerning the depth calculation process, two data (from each one of the receivers) of a small area are received, which are correlated and then the depth value of a pixel is computed. Doing this for every pixel a 3D image is generated.

There is not a large variety regarding to the specifications of these cameras, which are summarized in the following table, so maybe some of them are useful in the project and so can be chosen by price instead of other properties, usually more important in any system’s design.

Table 2.2.1.1. Intel RealSense depth cameras comparison

	L515	D455	D435/D435i	D415	SR305
Technology	LIDAR	Active IR Stereo	Active IR Stereo	Active IR Stereo	Coded light
Range	0.25 – 9 m	0.4 – 20 m	0.105 – 10 m	0.16 – 10 m	0.2 - 1.5 m
Resolution (pixels)	Up to 1024 x 768	Up to 1280 x 720	Up to 1280 x 720	Up to 1280 x 720	Up to 640 x 480
Frame rate (frame/s)	30	Up to 90	Up to 90	Up to 90	Up to 60
Outdoor detection	No	Yes	Yes	Yes	No
Dimensions (mm)	61 diameter 26 height	124 x 26 x 29	90 x 25 x 25	99 x 20 x 23	139 x 26 x 12
Price	298€	204€	153€/170€	127€	67.5€

First to discard are the ones that cannot detect obstacles outdoor, because are not useful in this project. Regarding the other three, D455 highlights due to the maximum range but actually the minimum is a dangerous property. Although the system is created to detect obstacles at a distance of 2 m, if necessary, it should detect them at 30 or 40 cm, which could not be possible with D455. As a maximum range of 10 m is higher enough for the purpose, D455 is also discarded.

Between the other three, which characteristics are almost the same, it is not important which one to choose. Between D435 and D435i the only difference is the presence of an IMU (Inertial Measurement Unit), a combination of gyroscopes, magnetometers and accelerometers, which is not useful in this project, so it is discarded. Finally, the distinctive is the minimum range, so it is easy to choose the smaller one, but taking into account the price gap, if necessary, D415 is also valid.

Summarizing, the optimal option is to use the D435 [8] camera of Intel RealSense depth line-up, which provides depth frames of up to 1280 per 720 pixels with a frequency of up to 90 Hz in a range between 0.105 m and 10 m long.

2.3.2. Altitude maintenance

In order to measure flight altitude in a precise way a research on some distance sensors has been done looking for them in different specialized on electronic devices Internet webs. Some candidates of all possible sensors on market have been selected to compare between them and finally choose the most suitable one justifiably.

Is previously discussed the choice of LIDAR among the other technologies, but there were another valid distance measurement sensor methodology: the one that measures by time of flight principle using infrared waves, so one of this type is included in the comparison. Furthermore, one with a huge (for the purpose) maximum range is also included although it was a filter used in the research.

In next table are summarized the most important specifications and price, which would be also compared in case of similar properties. It is important to say that all prices have been extracted from a distributor that works in Spain [9].

Table 2.2.2.1. LIDAR sensors comparison

	TeraRanger One [10] TERABEE	**LIDAR-Lite v3** [11] GARMIN	**Leddar One** [12] LeddarTech	**TF mini** [13] seeed	**LW20/C** [14] LightWare
Technology	IR + ToF	LIDAR	LIDAR	LIDAR	LIDAR
Maximum range (m)	Indoor: 14 Outdoor: 6	40	40	Indoor: 12 Outdoor: 6	100
Resolution (cm)	0.5	1	0.3	1	1
Speed (Hz)	1000	500	140	1000	10000
Accuracy (cm)	2	2.5	5	6	10
Weight (g)	10	22	14	12	20
Dimensions (mm)	18 x 29 x 35	20 x 48 x 40	51 diameter 30.6 height	15 x 36 x 16	20 x 30 x 43
Price	64 €	132 €	128 €	31 €	316 €

First thing to focus on is the maximum range: in the case of LW20/C it is much bigger than needed, which is not a problem if the other specifications meet the requirements; but in TeraRanger One and TF mini cases this is too small taking into account that the drone will always fly outdoors in this project. Therefore, these two sensors are discarded and the other three are, for now, options.

Regarding to the resolution the best one is Leddar One, but this is not the most important factor to take into account. However, a precise and accurate measurement is needed so it is easy to select LIDAR-Lite v3 due to accuracy. The other parameter is speed, in which LW20/C highlights, but this is not a primordial point and, furthermore, this sensor is the less accurate and the most expensive.

In this case, the optimal option is to choose LIDAR-Lite v3 from GARMIN, that provides depth data with an accuracy of 2.5 cm, a resolution of 1 cm and a refresh rate of 500 Hz in a range up to 40 m. However, given the current situation, has been only possible to get TF mini of seeed, which most important lost regarding to the selected sensor is that the maximum range is not enough for the purpose, so it can cause real problems while flying. Furthermore, the accuracy, which is another of the important requirements, is also worse than the selected one. In contrast, it has higher refresh rate, but summarizing it is a bad change.

3. HARDWARE ARCHITECTURE

In this chapter is designed the hardware part. First is needed to know both the drone type and elements as controller or processor that make part of the drone used, ignoring the ones related to pottery acquirement as can be the potsherds detection camera. Once these characteristics are known, the design of the hardware architecture can be done by adding the two distance measurements sensors chosen in the previous section.

3.1. Drone characteristics and elements

The drone is intended to use in this project is a quadcopter, so that it could do any type of movement (change altitude or speed or address to a concrete location) during the avoidance mission in an easy way.

Regarding to hardware elements that already take part of this drone, it is needed to know it has a Pixhawk 2.1 as a flight controller with a GPS module incorporated and a Raspberry Pi 4 de 4 GB as a computer, which are connected between them.

3.1.1. Pixhawk

Pixhawk is an open standard that provides guidelines and hardware specification for drone’s systems development and currently offers six low-cost flight controllers. For this project, Pixhawk 2.1 [15], or Pixhawk Cube, is used in order to make the drone do any movement to follow the fly path or execute a mission. It is usually said that flight controller would be to UAS the equivalent of the brain to an animal, meaning that controls and monitors all the drone do. It is able to change flight direction, sense and speed by varying motor’s angular speed.

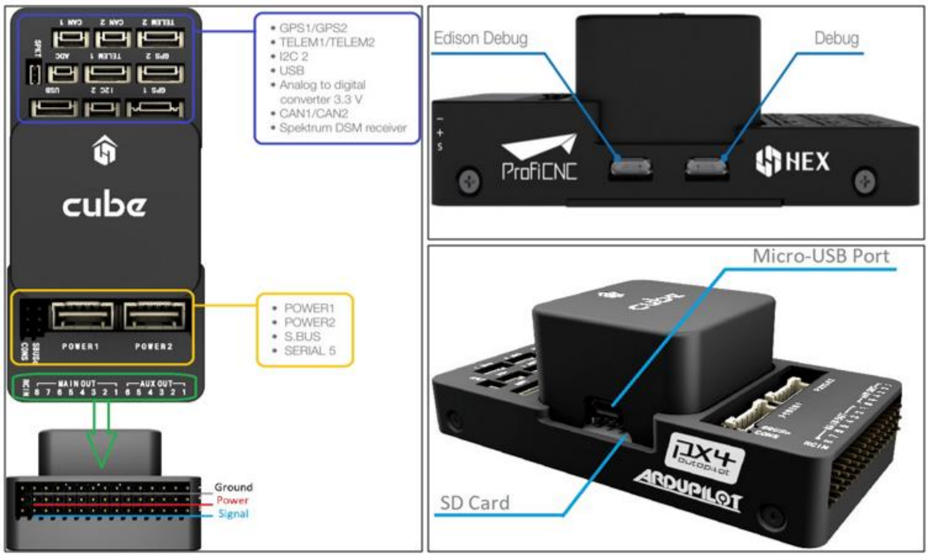


Fig. 3.1.1.1. Pixhawk 2.1 ports

The flight controller must always have a battery connected to power it, which is usually a Li-Po battery. In fact, in the case of quadcopter a power module is what is connected directly to the flight controller, since it must have also connected the ESC [Electronic Speed Controller], in order to power these elements that control the motors speed.

Furthermore, flight controller can have other modules as, for example, a buzzer for sound warnings or a telemetry module, that transmits data about the drone status to a receiver device, which is usually the pilot's computer.

In this case, it is important to take into account that the drone has a GPS module, which is also connected to the flight controller, providing position data in 3D from satellites.

3.1.2. Raspberry Pi

Raspberry Pi is a series of cheap and small single-board computers made of a processor and a graphic module, initially created to promote technological science. For this project, the newest one is used, Raspberry Pi 4 Model B [16] with 4 GB of RAM. This device is used to process data and is the one containing the code designed for the obstacle avoidance mission.

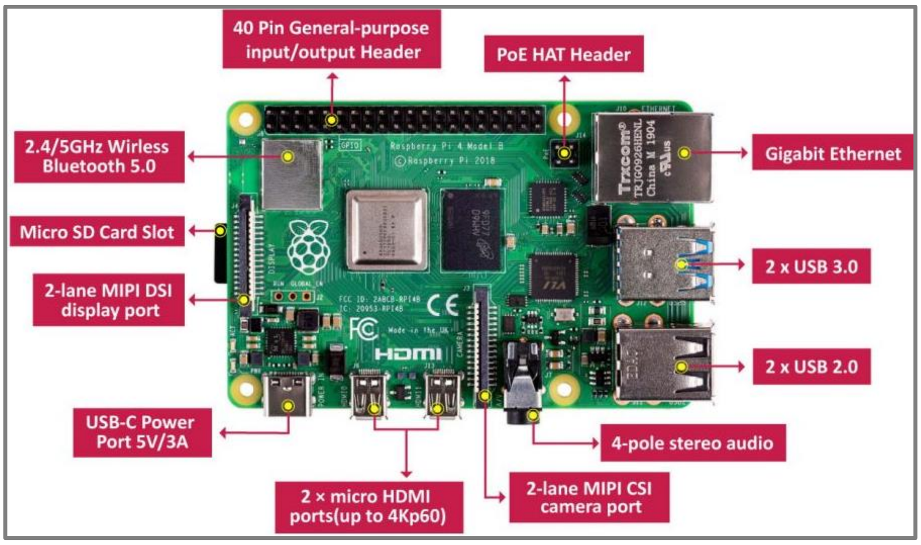

40 Pin General-purpose input/output Header
PoE HAT Header
2.4/5GHz Wirless Bluetooth 5.0
Gigabit Ethernet
Micro SD Card Slot
2 x USB 3.0
2-lane MIPI DSI display port
2 x USB 2.0
USB-C Power Port 5V/3A
4-pole stereo audio
2 × micro HDMI ports(up to 4Kp60)
2-lane MIPI CSI camera port

Fig. 3.1.2.1. Raspberry Pi 4 ports

3.2. Drone connection

Now the used drone is known and which hardware elements does it use, the connection of previously selected distance measurement devices has to be done.

Raspberry sends processed data to Pixhawk and so they have to be connected. Regarding to hardware, they are joined with cables from one TELEM port of Pixhawk to serial GPIO of Raspberry.

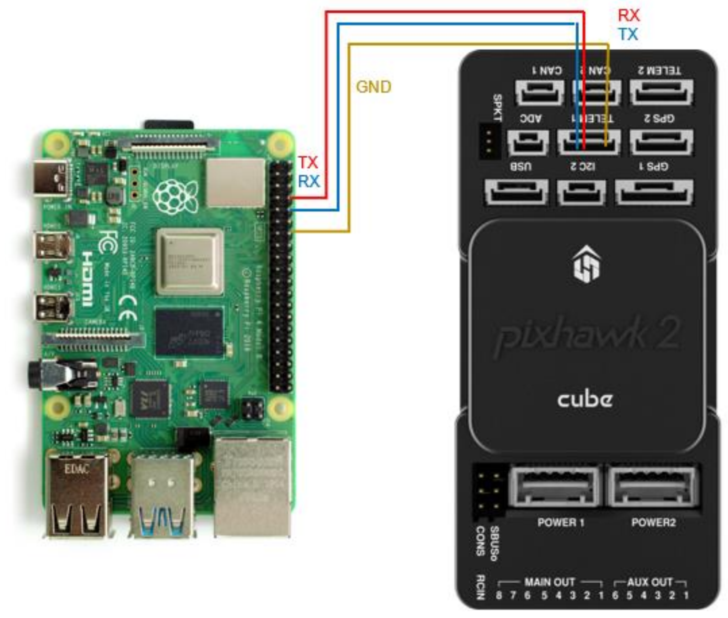


Fig. 3.2.1. Raspberry to Pixhawk connection

On one hand, the Intel camera transmits frames using a USB-C 3.1 Gen 1, which is not supported by Pixhawk and so it has to be connected to Raspberry USB 3.

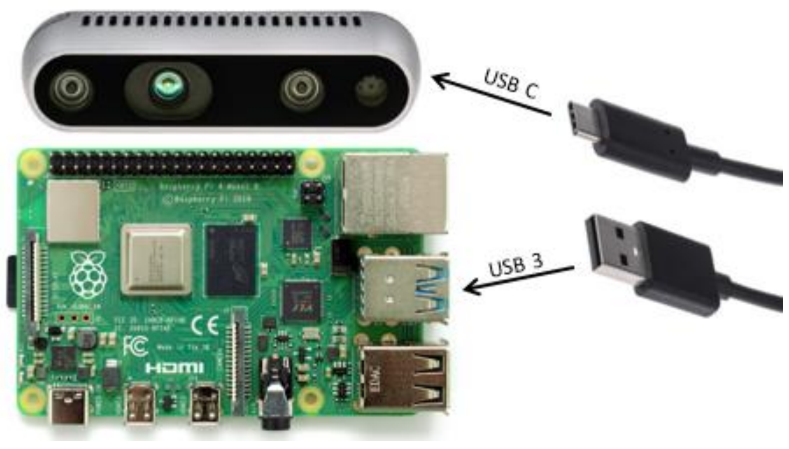


Fig. 3.2.2. D435 camera to Raspberry connection

On the other hand, the TF mini sends data by UART o I2C ports, so it can be connected to both Pixhawk TELEM ports and Raspberry GPIO serial ports. As these last ones are occupied by Raspberry to Pixhawk connection, the LIDAR is connected to Pixhawk, which can send the altitude data to Raspberry for the avoidance mission.

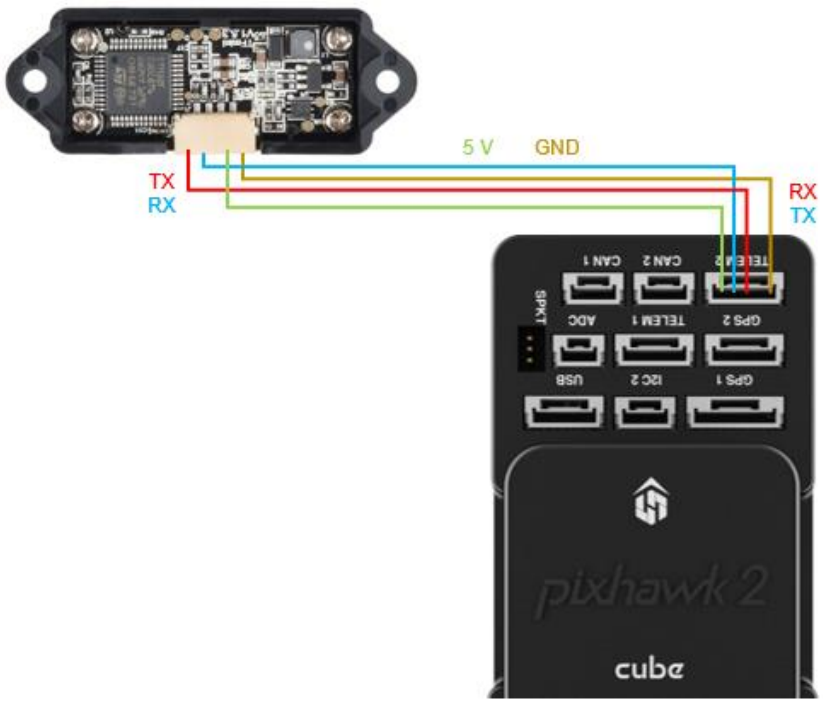


Fig. 3.2.3. TF mini LIDAR to Pixhawk connection

4. SOFTWARE IMPLEMENTATION

As it has been mentioned, the drone has a previously charged mission and flies the path in an autonomous way. In order to give the mission it has to run a code [17], which also contains the two navigation systems designed during this work: altitude maintenance and obstacle detection and avoidance.

This code is designed only to test it, it is not in the context of the real archaeological project. It creates a simple mission, which is moving forward 100 m in the heading direction, and, during this mission, flight altitude can be checked (altitude maintenance system) and an obstacle has to appear, in order to detect and avoid it (obstacle detection and avoidance system).

4.1. Programming language

Given his general nature and the facility of the language, Python has been chosen to code this work, which has been developed first in a laptop and then, after configuring it and installing his operating system, in the Raspberry Pi.

Python is a language created on latest years of 1980s and first released on 1990, currently developed under an open-source license with a philosophy of a readable syntax. It is highly used due to their general characteristics and applications and due to the fact it is free to use.

The IDE (Integrated Development Environment) chosen to code has been Visual Studio Code because of the ease to develop code remotely in Raspberry Pi using SSH (Secure Shell) protocol and also of using git.

Visual Studio Code is a free source-code editor first released in 2015 so popular because of it allows to code in a variety of programming languages and can so develop a variety of applications.

4.2. Communication between components

In order to get or transmit data to the drone or between his components, a communication protocol is used. In this case MAVLink [18], that stands for Micro Aerial Vehicle Link, is used. MAVLink is a lightweight protocol, in concrete to micro UAV (Unmanned Aerial Vehicle), that uses messages for the communication, which are defined within .xml files and so can be use in a large variety of languages.

For the two-way data transmission between the flight controller and the computer, DroneKit [19] is used. DroneKit is an API (Application Programming Interface) that uses MAVLink communication to easily create Python applications in order to develop autonomous flight drones, by controlling the drone movements directly. Furthermore it allows a direct access to vehicle data, meaning it telemetry, status and other parameters.

4.3. Required libraries

In order to get data from the Intel RealSense camera, it has to be used a SDK (*Software Development Kit)* that allows to work with the line-up of D400 cameras and whose name for Python wrapper is '*pyrealsense'* [20].

For the communication between the GCS (Ground Control Station) and the flight controller, MAVLink library has to be used. In the case of Python language it is named '*pymavlink*' [21] and can be easy installed using pip.

Finally for the flight controller to computer communication DroneKit library, called also '*dronekit*' [22], is used, which is directly written in Python and also can be easily installed using pip.

All these three used libraries used are free open-source libraries, which is a high advantage because of the existence of other users' code on-line, in addition to the official ones. It is important to install all the libraries in the Raspberry Pi, in order to work on the drone.

4.4. Systems performance

This last section explains finally the logic of the two systems, separately. As expected, the avoidance system is the difficult one and is made of a big loop, containing other ones inside. For the sake of simplicity, the way in which it avoids objects is increasing altitude.

First of all, it is important to take into account that when the D435 camera detects an obstacle, the initial mission of the drone is saved in a vector and then is stopped. After avoiding the obstacle, the one where avoidance has finished (which corresponds to the actual location) substitutes the point where initial mission was stopped and then the drone goes on with the mission since there.

it has been explained before, the fact of a pixel per pixel depth measure is a big advantage in order to detect obstacles, compared with sensors, which only do one measure. On the contrary, for reading and analysing data it is annoying, since the code has to get the distance of every pixel. As this data is used to determine the existence of an obstacle, the method used is to count how many pixels detect something at a distance less than the security one and then compare this sum with a previously set threshold, which state from which account of pixels is considered to have an obstacle.

Therefore, for the obstacle detection, two parameters have to be set: the security distance and the minimum account of pixels. Furthermore, for the obstacle avoidance there are also some parameters to be set: the increases (one on altitude and the other one on distance) and a bigger security distance, which are Δh, Δd and d, respectively in the avoidance loop image. In addition, a last parameter (it does not appear in the image) accounts for the number of times that altitude is increased.

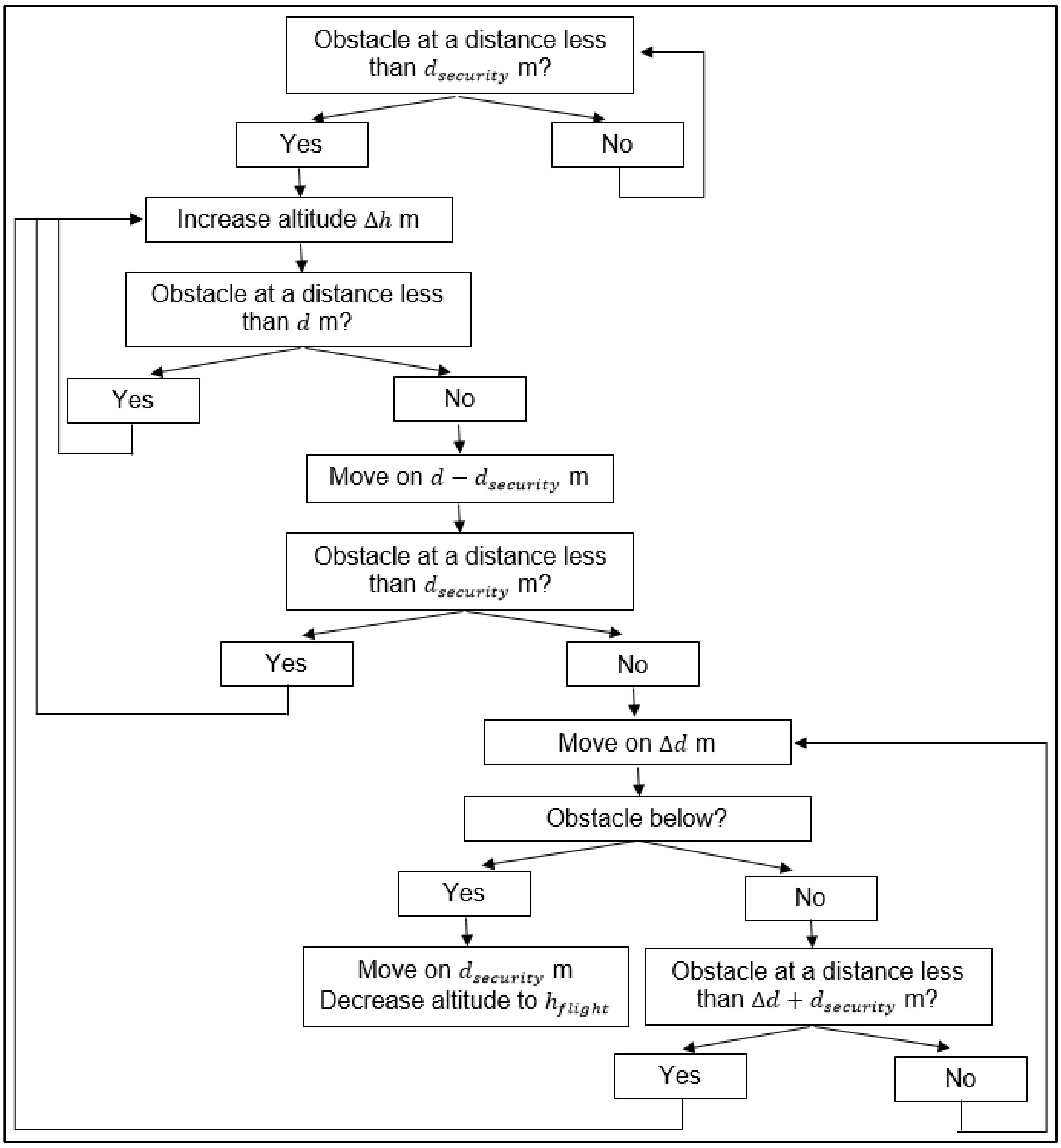

Obstacle at a distance less than $d_{security}$ m?
Yes
No
Increase altitude Δh m
Obstacle at a distance less than d m?
Yes
No
Move on $d - d_{security}$ m
Obstacle at a distance less than $d_{security}$ m?
Yes
No
Move on Δd m
Obstacle below?
Yes
No
Move on $d_{security}$ m Decrease altitude to h_{flight}
Obstacle at a distance less than $\Delta d + d_{security}$ m?
Yes
No

Fig. 4.4.1. Obstacle avoidance system loop

When an obstacle is detected, the height is increased and, in every increase it does, the code looks if there is an obstacle at a distance less than the large security one. When it gets an altitude where there is no obstacle, the drone moves on a distance equal to the subtraction of the two security distances and check again if there is an obstacles at a distance less than the smaller security one. If there is, it keeps in this loop of increasing altitude and checking the existence of an object on the flight direction. When finally there is not an obstacle in this direction it has to check if the obstacle is still under the drone and it is done with the LIDAR data. In order to do so it compares what should be the current altitude (flight altitude plus the multiplication of the counter with the altitude increment) with the distance it measures. In the case of smaller measured distance (obstacle still under the drone), it has to move on by increments of distance in order to pass the obstacle but before moving it has to check if there is obstacle behind.

Now the obstacle avoidance is explained, the height maintenance loop is much easier and only in one loop can be done. The code looks for the altitude data from the TF mini altimeter and compares it with what should be the flight altitude. If these two compared distances are not the same, a message is sent to Pixhawk in order to change flight altitude. It is important to mention that it is made to detect as equal differences that are below a 5% error, because it is considered the accuracy of the sensor.

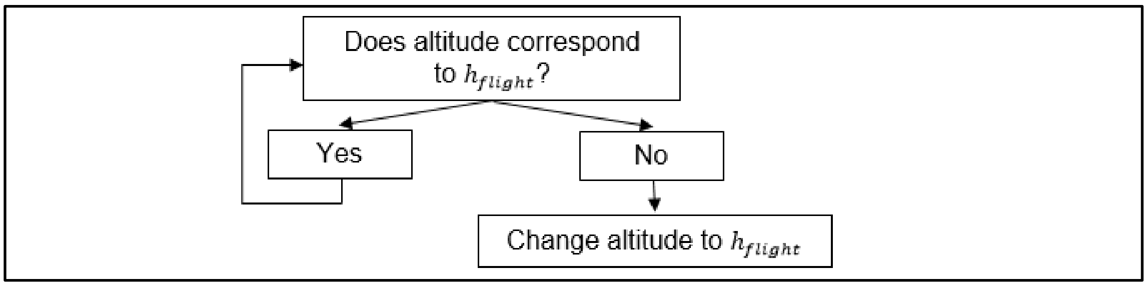

Does altitude correspond to h_{flight}?
Yes
No
Change altitude to h_{flight}

Fig. 4.4.2. Height maintenance system loop

5. EXPERIMENTAL TESTS

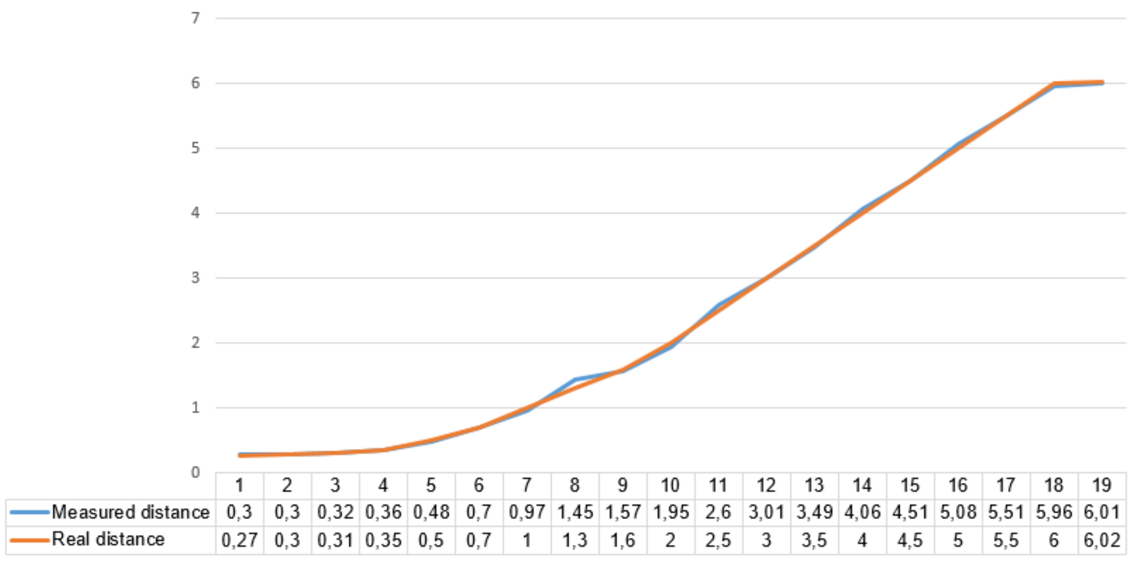
First both distance measurement devices are analysed in order to study the functioning. On one hand, it is needed to check the accuracy and the range of the sensor. On the other hand, the camera resolution has to be checked and the two parameters of obstacle detection loop should be set by testing it. Finally, the entire code will be tested by moving the drone as if it was doing it alone.

In a try of recreate possible flight conditions, all three tests has been performed outdoors.

5.1. TF mini LIDAR test

A small code test has been written in order to only read and print the measured distance of this sensor. As it has been seen in the market study (section 2.3.2.), this sensor works within a range from 30 cm to 6 m outdoors providing an accuracy of 6 cm with 1 cm resolution.

In order to check these specifications, some measures have been done, both with the TF mini sensor and manually with a measuring tape, which results are shown in Fig. 5.1.1. in meters.



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
Measured distance	0,3	0,3	0,32	0,36	0,48	0,7	0,97	1,45	1,57	1,95	2,6	3,01	3,49	4,06	4,51	5,08	5,51	5,96	6,01
Real distance	0,27	0,3	0,31	0,35	0,5	0,7	1	1,3	1,6	2	2,5	3	3,5	4	4,5	5	5,5	6	6,02

Fig. 5.1.1. Real distance VS measured distance

Generally, the previous figure shows that the sensor is accurate in measure, but it is needed to take into account that I had to stay some seconds holding it and waiting because it is so unstable when it is moved and, during first seconds, it returns oscillating values, which are so different between them.

The output distance value is given in centimetres without any decimals so, as the specifications indicate, the resolution is 1 cm. Computing the average error in the measured distance, an accuracy of 2 cm is get but, as it can be seen, the maximum one is 8 cm, which is more than the accuracy on the specifications datasheet.

Regarding to the range, when it detects something closer than 30 cm, automatically the output is 30 cm, which is the minimum range. According to the official seeed webpage data, the maximum range indoors is 12 m and outdoors 6 m, but actually it detected obstacles at distances up to 7 or 8 m (which data is not included in Fig. 5.1.1.). Although the LIDAR can measure it, the output value is even more unstable than usually and sometimes it goes further than 600 m, which indicates that has reached the maximum value, so the best option is to consider 6 m as the maximum range, as the datasheet indicates.

5.2. D435 camera test

A small code has been developed in order to do the test. It consists of the D435 functions file of the main code and a program that only start the connection and continuously prints if there is an obstacle or not, using the continuously sent depth frames of the camera.

Before doing the test and check the specifications and set the code loop parameters, Intel RealSense Viewer has been installed in order to see how the camera measures distance. An example of a photograph is in Fig. 5.2.1. (with no edit), which shows a depth image oh my desktop and is also attached a thumbnail of the colour stream image of the camera at the lower left corner.

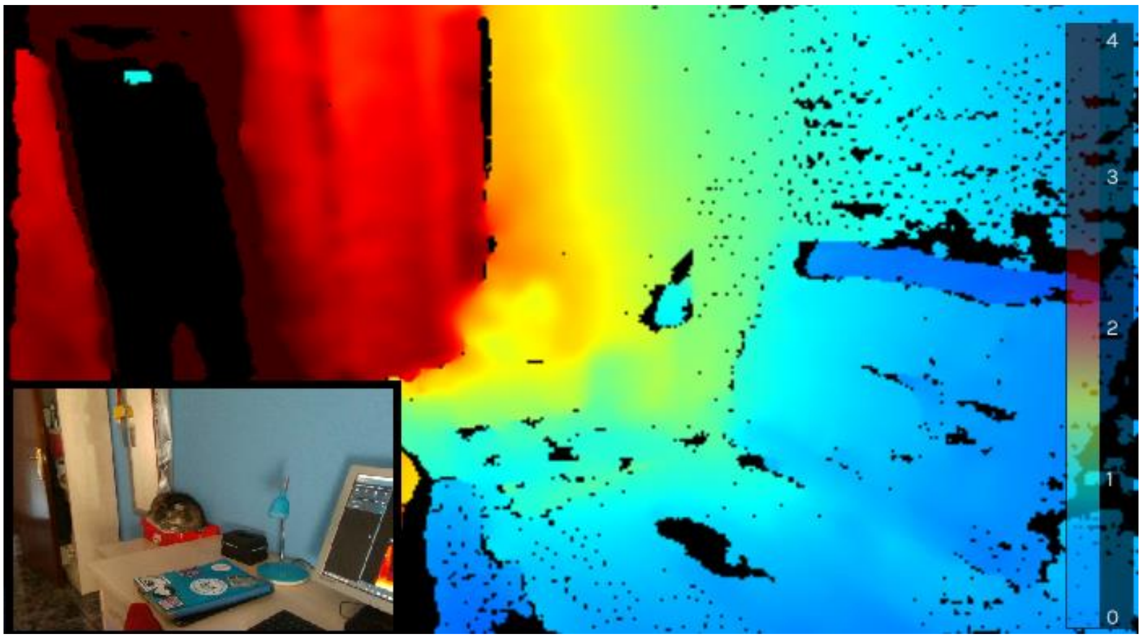


Fig. 5.2.1. D435 depth image example

As it can be seen in the previous figure, the ratio is not the same for the stereo module depth stream than for the colour stream of the camera. In the depth image, for example, all the computer display is seen while in the colour one only the left half of the screen is seen. So, the resolution of this two modes is different. For the code, it is important to know exactly the resolution of the depth image because it is needed while running the code in order to scan all positions of the matrix containing the depth data of each pixel. From the specifications, it is known that resolution is up to 1280x720 pixels. Therefore, the first test is to simply debug the code and see the depth frames dimensions, which were 640x480 pixels.

Then, the relation between altitude and the security distance parameter has to be known, taking into account that the vertical field of view of the camera is 58º. The following inequality, which comes from using basic trigonometry, is computed in order to not detect the ground as an obstacle while flying.

$$h \geq d_{seg} \cdot \tan(58º) \qquad \textbf{(5.2.1.)}$$

So from now on, the flight altitude is 1 m and the security distance as 50 cm in order to test it without having problems with terrane. When using the drone in the archaeological project, the flight altitude will be stablished depending on the images resolution and, using this equation, the security distance can be set.

The other parameter used during the obstacle avoidance mission is the threshold that stablish which is the minimum number of pixels detecting an obstacle that implies that an object is an obstacle. It is important to choose it with criteria because if it is too small can lead to a lack on efficiency on the code, which is not a big problem, but if it is too big can cause a collision with the obstacle. In the first case, the code thinks there is an obstacle and tries to avoid it but it maybe is not, which would not be efficient. In the last case, when the threshold is set bigger than needed, the code needs a big or a close object, which could imply to not decide to avoid or to take this decision when the drone has no longer time to avoid it. After some tests, the value is set as 1000, because it has not failed in the detection, testing with some different obstacles, varying the position (taking into account the field of view) and the distance (checking the range).

5.3. Code test

As the LIDAR is not useful in a drone, it has not been possible to test the code in a drone real flight, but a simulation has been done for testing the systems design and implementation. First, all hardware has been connected and subjected to a multiplatform drone frame in a realistic way. Then, the code has been adapted to use it without flying. The changes done were to comment all lines that execute functions in which DroneKit is used to make the drone move itself (e.g. changing altitude or moving forward a certain distance) and to replace all these lines by ‘print’ ones (e.g. “Initial mission saved” or “Increasing altitude 0.3 m”). This way, the movements done manually are what the printed lines indicate.

Regarding to the altitude maintenance system, in terms of code it worked perfectly since it is a really easy design. However, as it has been repeatedly said before, the LIDAR performance is not good enough to make it work in this use, although it was inside the working range. With respect to the obstacle detection and avoidance system, the detection part also worked perfectly, which indicates detection parameters and functions were correctly set and designed, but the avoidance part had some difficulties. On one hand, it is needed to take into account that during the avoidance mission, the code needs to use de altitude data coming from TF mini, so it is easy to think that the problem is again the sensor, since all fails always came from the moment in which is needed to measure height. On the other hand, the method used to test the code is not precise since my movements were estimating in distance because I was giving a premium to time.

One of these tests was recorded and it is available on YouTube [23], in which both the drone movement (done manually by me) and the code output can be seen. This video is uploaded without audio, but in the real situation I was helped by two people: one of them reading to me the output of the code, indicating the movement I had to do, and the other one recording the image.

6. CONCLUSIONS

The aim of this work is to design, implement and proof two navigation systems in order to help an autonomous-flight drone, in the context of an archaeological survey project. The research of pottery is done with a multispectral camera, so the altitude must be constant during all the flight, following the terrain it is flying. Because of this, one of the systems designed during this work, is to maintain strictly constant the height. Furthermore, it is needed to have good resolution in the images, so the flight altitude has to be lower enough to photograph a small area per pixel with high resolution. Due to the low flight altitude, it is highly possible to have some obstacles during the flight, so the other navigation help is an obstacle detection and avoidance system.

On one hand, in order to measure the height, which is the distance to soil, a LIDAR has been used. In the market study, GARMIN LIDAR Lite-v3 has been considered the most suitable one, but, given the current situation, the only one available was seeed TF mini. This sensor provides 1000 data per second with an accuracy of 6 cm and a resolution of 1 cm of objects from 30 cm to up to 6 m away outdoors. As it had been said while doing the market study, the specifications that this LIDAR provides are not appropriate for the use since the most important ones, accuracy and maximum range, are not good enough. After testing the device, the minimum range and the resolution are exactly the ones of the datasheet, but the accuracy is 8 cm and the range is 1 or 2 m bigger. Although the specifications are not the most appropriate, the bigger drawback is the instability of the measures. Every time the distance between the LIDAR and the closer object changes, the output distance value takes some second to stabilize itself and, during this time, it varies on a big range of values. So, as the use is to put in in a continuously moving object (drone), it is not useful since the measures read will be these ones of instability.

On the other hand, in order to detect obstacles, a depth camera has been chosen and used. The Intel RealSense depth camera provides up to 90 depth frames of up to 1280x720 pixels per second within a range between 10.5 cm and 10 m, with vertical, horizontal and diagonal fields of view of 58º, 87º and 95º, respectively. Given the specifications, this device seems useful for the detection mission, which has been checked after some tests. During the tests, the resolution get is 640x480 pixels, which is also enough for the use, and the vertical field of vied has been used to stablish the relation between the flight altitude and the security distance set, in order to not detect the ground as an obstacle. After that, an altitude of 1 m and a security distance of 50 cm have been used. Regarding to the parameter that stablish how many pixels detecting an obstacle are enough to establish that it is really an obstacle, it has been set as 1000, in which case the test code did not have any problem to detect some different objects in different positions and distances inside the range, from 10.5 cm to 10 m.

Summarizing, in respect of the distance measurement devices used, the altimeter is not useful in terms of range, accuracy and time and must be changed in order to use it in a drone and the obstacle detection camera has good specifications and results and so is a good choice.

After concluding that the LIDAR is not able to use in a drone, have been not possible to test the code in a flight, but a simulation has been done. In this simulation, the drone was moved manually as it was flying in order to approach to the real situation. The main problems during this test came from, on one hand, doing it manually with distance movements not measured (approximate) and, on the other hand, the instability of the LIDAR measurements.

Summarizing, in order to get a better design of these two navigation help systems, the most important thing is to change the altimeter sensor by one with better accuracy and, if it is possible, higher maximum range. The next step would be to do the LIDAR tests again to check the specifications and, if they are good enough, to try again the simulation. It is expected to get then better results, although not a perfect work because of the approximation of movements, but enough to know if it is ready to do a real flight test, which would approve (or not) the design of the avoidance mission.

REFERENCES

[1] Spain Government, “Strategic plan for civil sector drones development in Spain.” https://www.fomento.gob.es/NR/rdonlyres/7B974E30-2BD2-46E5-BEE5-26E00851A455/148411/PlanEstrategicoDrones.pdf (accessed Sep. 03, 2020).

[2] Hemav Foundation, “Hemav Foundation.” http://hemavfoundation.com/sobre-hemavfoundation/ (accessed Aug. 19, 2020).

[3] H. A. Orengo and A. Garcia-Molsosa, “A brave new world for archaeological survey: Automated machine learning based potsherd detection using high-resolution drone imagery,” *J. Archaeol. Sci.*, p. 12, 2019.

[4] TERABEE, “Choosing the right distance sensor for your application.” https://www.terabee.com/choosing-right-distance-sensor-your-application/#:~:text=Distance sensors equipped with IR,is reflected in all directions. (accessed Sep. 25, 2020).

[5] seeed studio, “Types of distance sensors.” https://www.seeedstudio.com/blog/2019/12/23/distance-sensors-types-and-selection-guide/ (accessed Sep. 25, 2020).

[6] Sparkfun, “Distance sensor comparison guide.” 25/9/2020, [Online]. Available: https://www.sparkfun.com/distance_sensor_comparison_guide.

[7] Intel RealSense, “Depth cameras comparison.” https://www.intelrealsense.com/compare-depth-cameras/ (accessed Sep. 26, 2020).

[8] I. RealSense, “D435 depth camera.” https://www.intelrealsense.com/depth-camera-d435/ (accessed Sep. 26, 2020).

[9] RobotShop, “RobotShop webpage.” https://www.robotshop.com/es/es/?___store=es_es&___from_store=eu_en (accessed Sep. 27, 2020).

[10] TERABEE, “TeraRanger One.” https://www.terabee.com/shop/lidar-tof-range-finders/teraranger-one/ (accessed Sep. 26, 2020).

[11] GARMIN, “LIDAR-Lite v3.” https://buy.garmin.com/es-ES/ES/p/557294 (accessed Sep. 26, 2020).

[12] LeddarTech, “Leddar One.” https://leddartech.com/lidar/leddarone/ (accessed Sep. 26, 2020).

[13] seeed studio, “TF mini.” https://www.seeedstudio.com/Seeedstudio-Grove-TF-Mini-LiDAR.html (accessed Sep. 27, 2020).

[14] LightWare, “LW20/C.” https://lightwarelidar.com/products/lw20-c-100-m (accessed Sep. 27, 2020).

[15] Pixhawk, “Pixhawk 2.1.” https://docs.px4.io/v1.9.0/en/flight_controller/pixhawk-2.html (accessed Oct. 12, 2020).

[16] Raspberry Pi, “Raspberry Pi 4 model B.” https://www.raspberrypi.org/products/raspberry-pi-4-model-b/ (accessed Oct. 12, 2020).

[17] L. Parga Gata, “Github repository,” *22/10/2020*, 2020. https://github.com/laurapg98/CollisionAvoidance_test.

[18] MAVLink, “MAVLink protocol.” https://mavlink.io/en/ (accessed Oct. 22, 2020).

[19] DroneKit, “DroneKit API.” https://dronekit.io/ (accessed Oct. 20, 2020).

[20] I. RealSense, “pyrealsense2 Python library.” https://github.com/IntelRealSense/librealsense/tree/master/wrappers/python (accessed Oct. 15, 2020).

[21] MAVLink, “mavlink Python library.” https://github.com/ArduPilot/pymavlink (accessed Oct. 20, 2020).

[22] DroneKit, “dronekit Python library.” https://github.com/dronekit/dronekit-python (accessed Oct. 20, 2020).

[23] L. Parga Gata, “Obstacle detection and avoidance test.” https://www.youtube.com/watch?v=YHlUhO3HOCQ&feature=youtu.be (accessed Oct. 23, 2020).

APPENDIX 1: Code

Main.py script

```
# Import functions
from D435functions import start_D435, exists_obstacle_ahead, stop_D435
from DRONEfunctions import start_connection, stop_connection, stop_mission,
save_mission, get_flight_altitude, change_altitude, move_forward,
add_current_waypoint, test_mission, exists_obstacle_under, upload_mission

# Flight parameters
flightaltitude = 1 # m *******************************************
speed = 5 # m/s *******************************************

# Camera parameters
serialnumber_D435 = "829212070982"
x_pixels = 640
y_pixels = 480

# Security parameters
securitydistance = 0.5 # m *******************************************
minpixels = 1000 # pixels *******************************************

# Connection with D435 camera
pipe_D435 = None
while pipe_D435 == None:
    pipe_D435 = start_D435(serialnumber_D435)

# Connection with drone
vehicle = None
while vehicle == None:
    vehicle = start_connection()

# Upload mission
test_mission(vehicle, flightaltitude, speed, 100)

try:
    while True:

        # Check altitude (maximum 5% error)
        while (altitude >= 0.95 * flightaltitude and altitude <= 1.05 *
flightaltitude):
            #get_flight_altitude(vehicle, flightaltitude)
            print("CHANGE TO FLIGHT ALTITUDE (" + str(flightaltitude) + "m)
| Current altitude: " + str(altitude) + " m")
            altitude = vehicle.rangefinder.distance
        print("Flying at " + str(altitude) + " m")
        print("Altitude OK\n")

        # Obstacle detected
        if (exists_obstacle_ahead(pipe_D435, securitydistance, x_pixels,
y_pixels, minpixels) == True):
            print("Obstacle detected")
            print("Starting avoidance mission\n")

            # Save initial mssion
            initialmission = save_mission(vehicle)
            print("Initial mission saved\n")

            # Stops the initial mission
            #stop_mission(vehicle)
            print("Initial mission stopped\n")

            # Obstacle avoidance parameters
            Ah = 0.3 # m ******************************************* IT MUST
BE SMALLER THAN SECURITY DISTANCE
```

```
            Ad = 0.3 # m ******************************************* IT MUST
BE SMALLER THAN SECURITY DISTANCE
            d = 1.5 # m ******************************************* IT MUST
BE HIGHER THAN SECURITY DISTANCE

            # Obstacle avoidance loop
            counter = 0
            while (exists_obstacle_ahead(pipe_D435, securitydistance,
x_pixels, y_pixels, minpixels) == True):
                print("[while 1] Obstacle at less than " +
str(securitydistance) + " m\n")
                while (exists_obstacle_ahead(pipe_D435, securitydistance,
x_pixels, y_pixels, minpixels) == True):
                    print("[while 2] Obstacle at less than " +
str(securitydistance) + " m\n")
                    while (exists_obstacle_ahead(pipe_D435, d, x_pixels,
y_pixels, minpixels) == True):
                        print("Obstacle at less than " + str(d) + " m")
                        #change_altitude(vehicle, Ah)
                        print("INCREASE ALTITUDE " + str(Ah) + " m\n")
                        counter += 1
                    #move_forward(vehicle, d - securitydistance, speed)
                    print("MOVE " + str(d - securitydistance) + " m\n")
                obstacle_under = True
                while (exists_obstacle_ahead(pipe_D435, securitydistance +
Ad, x_pixels, y_pixels, minpixels) == False and obstacle_under == True):
                    print("Obstacle under the drone")
                    print("No obstacle at less than " + str(Ad) + " m\n")
                    if (exists_obstacle_under(vehicle, flightaltitude,
counter*Ah) == False):
                        print("No obstacle under the drone")
                        #get_flight_altitude(vehicle, flightaltitude)
                        print("DECREASE ALTITUDE TO " + str(flightaltitude)
+ " m (flight altitude) \n")
                        obstacle_under = False
                    else:
                        print("Obstacle under the drone")
                        #move_forward(vehicle, Ad, speed)
                        print("MOVE " + str(Ad) + " m\n")

            # Return to initial mission
            print("Avoidance mission done. Going on with initial
mission.\n")
            initialmission_edit = add_current_waypoint(vehicle,
initialmission, flightaltitude)
            upload_mission(vehicle, initialmission_edit)

finally:

    # Connection with camera
    stop_D435(pipe_D435)

    # Connection with drone
    stop_connection(vehicle)
```

DRONEfunctions.py script

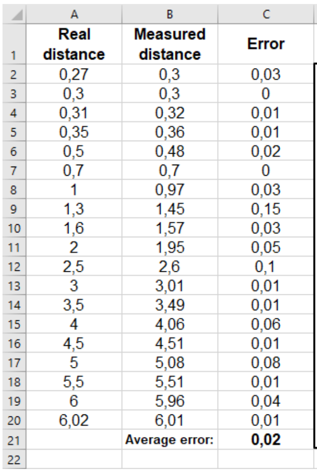
```
# Import libraries
from dronekit import connect, VehicleMode, Command, LocationGlobal
from pymavlink import mavutil

import serial
from math import asin,cos,pi,sin

# Starts connection with the drone (controller)
def start_connection():
    vehicle=connect('/dev/serial0', baud=921600, wait_ready=True)
    return vehicle

# Stops connection with the drone
def stop_connection(vehicle):
    vehicle.close()

# Stops the initial mission
def stop_mission(vehicle):
    vehicle.commands.clear()
    vehicle.commands.flush()

# Saves the actual mission & Returns it in a vector
def save_mission(vehicle):
    # Save mission
    vehicle.commands.download()
    vehicle.commands.wait_ready()
    # Store mission
    missionvector=[]
    for waypoint in vehicle.commands:
        missionvector.append(waypoint)
    return missionvector

# Changes to fligh altitude
def get_flight_altitude(vehicle, flightaltitude):
    # Current position
    latitude = vehicle.location.global_frame.lat
    longitude = vehicle.location.global_frame.lon
    # Final position
    newLocation = LocationGlobal(latitude, longitude, flightaltitude)
    # Move drone
    vehicle.gotoGPS(newLocation)

# Changes (+ increase, - decrease) the altitude Ah m
def change_altitude(vehicle, Ah):
    # Current position
    latitude = vehicle.location.global_frame.lat
    longitude = vehicle.location.global_frame.lon
    currentAlt = vehicle.location.alt
    # Final position
    newAlt = currentAlt + Ah
    newLocation = LocationGlobal(latitude, longitude, newAlt)
    # Move drone
    vehicle.gotoGPS(newLocation)

# Moves along Ad m
def move_forward(vehicle, Ad, speed):
    At = Ad / speed
    vehicle.send_global_velocity(speed, 0, 0, At)

# Adds current location as waypoint in a mission
def add_current_waypoint(vehicle, missionvector, flightaltitude):
    new_wp = Command(0, 0, 0, mavutil.mavlink.MAV_FRAME_GLOBAL_RELATIVE_ALT,
mavutil.mavlink.MAV_CMD_NAV_WAYPOINT, 0, 0, 0, 0, 0, 0,
vehicle.location.global_relative_frame.lat,
vehicle.location.global_relative_frame.lon, flightaltitude)
```

```
    missionvector.append(new_wp)
    return missionvector

# Uploads a vector of waypoints as a mission
def upload_mission(vehicle, missionvector):
    for waypoint in missionvector:
        vehicle.commands.add(waypoint)
    vehicle.commands.upload()

# Creates a mission (move along distance m) in order to test the code
def test_mission(vehicle, flightaltitude, speed, distance):
    # Current position
    lat = vehicle.location.global_frame.lat
    lon = vehicle.location.global_frame.lon
    heading = vehicle.heading
    # Final position
    finalPoint = pointRadialDistance(lat, lon, heading, distance/1000)
    # Create the mission
        # Take-Off:
    vehicle.commands.add(Command(0, 0, 0,
mavutil.mavlink.MAV_FRAME_GLOBAL_RELATIVE_ALT,mavutil.mavlink.MAV_CMD_DO_SET
_HOME, 0, 0, 1, 0, 0, 0, 0, 0, flightaltitude))
    vehicle.commands.add(Command(0, 0, 0,
mavutil.mavlink.MAV_FRAME_GLOBAL_RELATIVE_ALT,
mavutil.mavlink.MAV_CMD_DO_SET_HOME, 0, 0, 1, 0, 0, 0, 0, 0,flightaltitude))
    vehicle.commands.add(Command(0, 0, 0,
mavutil.mavlink.MAV_FRAME_GLOBAL_RELATIVE_ALT,
mavutil.mavlink.MAV_CMD_NAV_TAKEOFF, 0, 0, 20, 0, 0, 0, 0, 0,
flightaltitude))
    vehicle.commands.add(Command(0, 0, 0,
mavutil.mavlink.MAV_FRAME_GLOBAL_RELATIVE_ALT,mavutil.mavlink.MAV_CMD_DO_CHA
NGE_SPEED, 0, 0, 0, speed, 0, 0, 0, 0, flightaltitude))
        # Mission
    vehicle.commands.add(Command(0, 0, 0,
mavutil.mavlink.MAV_FRAME_GLOBAL_RELATIVE_ALT,mavutil.mavlink.MAV_CMD_NAV_WA
YPOINT, 0, 0, 0, 0, 0, 0, finalPoint.lat, finalPoint.lon, flightaltitude))
        # Landing
    vehicle.commands.add(Command(0, 0, 0,
mavutil.mavlink.MAV_FRAME_GLOBAL_RELATIVE_ALT,mavutil.mavlink.MAV_CMD_DO_LAN
D_START, 0, 0, 0, 0, 0, 0, finalPoint.lat, finalPoint.lon, flightaltitude))
    vehicle.commands.add(Command(0, 0, 0,
mavutil.mavlink.MAV_FRAME_GLOBAL_RELATIVE_ALT,mavutil.mavlink.MAV_CMD_NAV_LA
ND, 0, 0, 0, 0, 0, 0, finalPoint.lat, finalPoint.lon, flightaltitude))
    # Upload the mission
    vehicle.commands.upload()
    # Arm the drone
    vehicle.mode = VehicleMode("AUTO")
    vehicle.armed = True

# Computes the final point given the current position (lat, lon) [°],
bearing angle [°] and the distance [km] to move (straight)
def pointRadialDistance(lat1, lon1, bearing, distance):
    # Earth average radius
    rEarth = 6371.01 # km
    # Threshols for floating-point equality
    epsilon = 0.000001
    # Conversions
    rlat1 = lat1 * pi/180
    rlon1 = lon1 * pi/180
    degreeBearing = ((360-bearing)%360)
    rbearing = degreeBearing * pi/180
    rdistance = (distance)  / rEarth
    # Compute new latitude
    rlat = asin(sin(rlat1) * cos(rdistance) + cos(rlat1) * sin(rdistance) *
cos(rbearing) )
    # Compute new longitude
    if cos(rlat) == 0 or abs(cos(rlat)) < epsilon: # Endpoint a pole
        rlon=rlon1
    else:
        rlon = ( (rlon1 - asin( sin(rbearing)* sin(rdistance) / cos(rlat) )
+ pi ) % (2*pi) ) - pi
    # Conversions to degrees
    lat = rlat * 180/pi
    lon = rlon * 180/pi
    # New location (don't mind about altitude)
    return LocationGlobal(lat, lon, 0)

# Compares altitudes in order to know if there is an obstacle (True) or not
(False) under the drone
def exists_obstacle_under(vehicle, flightaltitude, Ah):
    if (vehicle.rangefinder.distance < 0.95 * (flightaltitude + Ah)): # With
a 5% of error
        return True
    else:
        return False
```

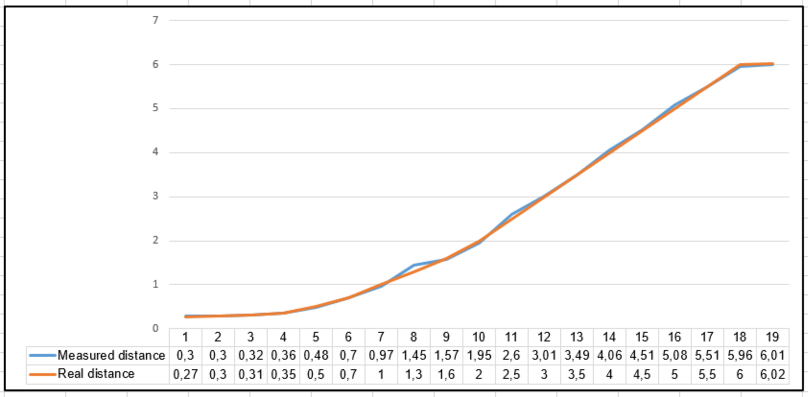
D435functions.py script

```
# Import libraries
import pyrealsense2 as libRS

# Starts connection with the camera
def start_D435(serialnumber):
    pipe_D435 = libRS.pipeline()
    cfg_D435 = libRS.config()
    cfg_D435.enable_device(serialnumber)
    cfg_D435.enable_stream(libRS.stream.depth) # depth
    pipe_D435.start(cfg_D435) # start recording
    return pipe_D435

# Establish (True, False) if there is an obstacle ahead
def exists_obstacle_ahead(pipe_D435, securitydistance, x_pixels, y_pixels,
minpixels):

    # Get depth data
    frames_D435 = pipe_D435.wait_for_frames() # frames
    depthframes = frames_D435.get_depth_frame() # filter --> only depth
frames

    # Number of pixels at a distance less than the security one
    counter = 0

    # Read depth data
    x = 0
    while x < x_pixels:
        y = 0
        while y < y_pixels:
            if(depthframes.get_distance(x,y) != 0 and
depthframes.get_distance(x,y) < securitydistance):
                counter+=1
            y+=1
        x+=1

    # Establish if there is (True) (False) or not an obstacle
    if counter > minpixels:
        return True
    else:
        return False

# Stop the connection with the camera
def stop_D435(pipe_D435):
    pipe_D435.stop()
```

APPENDIX 2: TF mini test data



Real distance	Measured distance	Error
0,27	0,3	0,03
0,3	0,3	0
0,31	0,32	0,01
0,35	0,36	0,01
0,5	0,48	0,02
0,7	0,7	0
1	0,97	0,03
1,3	1,45	0,15
1,6	1,57	0,03
2	1,95	0,05
2,5	2,6	0,1
3	3,01	0,01
3,5	3,49	0,01
4	4,06	0,06
4,5	4,51	0,01
5	5,08	0,08
5,5	5,51	0,01
6	5,96	0,04
6,02	6,01	0,01
	Average error:	**0,02**



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
Measured distance	0,3	0,3	0,32	0,36	0,48	0,7	0,97	1,45	1,57	1,95	2,6	3,01	3,49	4,06	4,51	5,08	5,51	5,96	6,01
Real distance	0,27	0,3	0,31	0,35	0,5	0,7	1	1,3	1,6	2	2,5	3	3,5	4	4,5	5	5,5	6	6,02