UNIVERSIDADE DA BEIRA INTERIOR
Engenharia

Semi-autonomous Wind Blade Inspection by a RPAS

Paulo Jorge Cunha Antunes

Dissertação para obtenção do Grau de Mestre em
Engenharia Aeronáutica
(Ciclo de estudos integrado)

Orientador: Prof. Doutor Kouamana Bousson
Coorientador: Prof. Doutor Juan Antonio García Manrique

Covilhã, outubro de 2017

Dedication

Dedico este trabalho às duas pessoas que nunca me deixaram sozinho, dando sempre o seu melhor para que nada me faltasse, que me tentaram transmitir sempre o seu conhecimento, sem perder a fé em mim,

Aos meus pais.

Acknowledgments

First, I would like to thank Juan Antonio García Manrique, for granting me the opportunity of incorporating this ambitious project plus all the hours of mentorship and guidance provided in order to accomplish the goals.

I would like to thank all the help from my laboratory partner, Omar Navarro over aspects I was not familiarized with, and also Cezara Rebeca for the help regarding bureaucratic aspects, which could not be handled by me, due to long distance, plus the friendship and partnership, from both of them, that made this project more enjoyable.

To my friends, inside of the University, that made me so ambitious and helped me grow personally and professionally, for all the projects joined together and created, that enabled me to tackle many obstacles and difficulties, providing me many tools to finish this projects in indirect ways. Also, all the friends outside the University, to all the people that I meet in Valencia, that helped me see things on different perspectives, bringing me a smile, whenever I needed it.

To the institution UBI, that provided me the opportunity to develop a project outside the university and to my tutor Professor Koamana Bousson.

To end, also would like to thank my parents, for providing me the unconditional love, stability and financial help, that did not doubt of my capacities for a minute, being my keystone.

Resumo

Entre muitas necessidades que o crescimento populacional acartou, energia elétrica é uma delas. Esta é um bem presente em todas as necessidades associadas com o nosso dia-à-dia.

Cada vez é mais comum a maquinaria presente na nossa vida ser alimentada através de eletricidade. Este é um aspeto positivo relativamente ao ambiente, uma vez que esta é uma energia ecológica. De forma a manter a quantidade necessária, é preciso aproveitar o maximo de cada fonte de energia disponível, uma vez que, atualmente, uma só não consegue responder a todas as necessidades existentes. Uma das formas mais viáveis de obter energia, é através das energias renováveis. Entre as quais, esta presente a energia eólica, sendo manifestada através do ar, fornecendo uma incalculável quantidade de energia à nossa disposição. A forma mais eficiente de transferir esta energia para eletricidade, é através dos Geradores Eólicos. Estas maquinas, tirando proveito da sua dimensão e configuração, permitem utilizar o vento para girar as suas pás, de forma a converter energia cinética em mecânica.

Esta tecnologia permite obter uma quantidade significante de energia, sendo uma industria em ascensão nos últimos anos. No entanto, estando estas estruturas exposta à fadiga, devido às condições extremas suportadas diariamente, devido ao sol, chuva, entre outras, limita drasticamente a duração de estes materiais, bem como a sua eficiência. Sendo este um grande número de fatores a ter em conta, a industria que as produz não consegue prever com precisão a data em que as mesmas devem ser reparadas, sendo necessário executar manutenção. No entanto, o tamanho destas estruturas constitui um entrave quando requisitada uma supervisão sem que seja necessário grande financiamento associado.

Quadcopteros são um tipo de RPAS que possuem uma fantástica manobrabilidade, estabilidade e simplicidade. Devido a isto, é possível efetuar manutenção em diferentes tipos de estruturas, de uma forma comoda, mesmo em espaços perigosos e por vezes de difícil acesso.

O trabalho atual apresenta uma solução face a esta adversidade, combinando as vantagens de um RPAS, Phantom 4 da DJI, com a linguagem de programação Java, fornecendo assim uma aplicação. Com esta, é permitido a todos os usuários produzir um percurso sobre Waypoints estabelecidos, através de localização geográfica, baseando-se em parâmetros definidos, de forma a adaptar-se às necessidades de cada utilizador.

Ao melhorar o trabalho aqui apresentado, será possível obter uma trajetória autónoma, que ao não depender do técnico que a executa, aporta uma maior fiabilidade e repetibilidade ao processo, aumentando assim a eficiência dos Geradores Eólicos.

Palavras-chave

Energias renováveis, Geradores Eólicos, RPAS, manutenção, Java, Waypoints.

Abstract

Over many requirements that population growth brought, electric energy is one of them. Energy is present on our everyday life needs. It is becoming common for machinery to be powered by electric energy. This is a positive aspect regarding environment, once this is a green energy. In order to keep the demand required, it is a must to share the most of every electric source available, once that it is impossible for one to fulfill all the supply needs.

One of the most suitable solutions to obtain this energy, is through renewable energy. Between them, wind provides great benefits, being expressed through the air, providing uncalculated power at our disposition. The most efficient way known to transfer this energy into electricity, is by Wind Turbines. These machines, due to their height and layout, are able to use the wind to spin their blades, converting kinetic energy into mechanical power.

This technology, granting notable incomes, being this an industry rising in the recent years. Although, being these structures exposed to fatigue, by the extreme conditions such as sun, rain, between others, drastically limits the material life endurance and their efficiency. Being this a wide combination of factors, the industry that produces these components, lacks the capacity of predicting accurately the time that the parts should be replaced. Thus, it is a must to provide maintenance over them. Furthermore, due to their dimensions, these structures constitute a hindrance to provide proper inspection without much pecuniary aspects involved.

Quadrotors are a type of RPAS which possess great maneuverability, stability and simplicity. Owing to this, becomes possible to perform surveillance on different types of structures, presenting a comfortable way to execute it on dangerous and difficult access spaces.

The current work presents a solution to this problem statement, combining the RPAS, Phantom 4 of DJI, with the programming language Java, providing an application. With this, every user is allowed to produce a trajectory where the requirements can adapt to his needs, incorporating essential parameters to define a path over the Waypoints established, relying on geographic localization, producing a path through them.

Improving this work to a completely autonomous inspection would bring total independency from the technician executing the maintenance, providing a viable and repeatable process, increasing the income of energy from Wind Turbines.

Keywords

Renewable energies, Wind Turbines, RPAS, maintenance, Java, Waypoints.

Table of Contents

List of Figures

List of Tables

List of Acronyms

API	Application Program Interface
APK	Android Package
APP	Application
BRC	Blade Reliability Collaborative
DJI	Dà-Jiāng Innovations Science and Technology Co., Ltd
HDPI	High Density Pixel Image
IDE	Integrated Development Environment
LDPI	Low Density Pixel Image
MDPI	Medium Density Pixel Image
MTOW	Maximum Take-Off Weight
ROI	Regions of Interest
UBI	Universidade da Beira Interior
UPV	Universitat Politècnica de València
XHDPI	eXtra High Density Pixel Image
XXHDPI	eXtra eXtra High Density Pixel Image
XXXHDPI	eXtra eXtra eXtra High Density Pixel Image

Nomenclature

m	Mass	[kg]
g	Acceleration due to gravity	[m/s^2]
L	Arm length	[m]
F_i	Vertical force produced by each rotor	[N]
${}^{a}R_i$	Rotation matrix	[-]
r_i	Position vector related to each direction	[m]
$\dot{r}_i$	Velocity vector related to each direction	[m/s]
$\ddot{r}_i$	Acceleration vector related to each direction	[m/s^2]
r_{des}	Desired position vector	[m]
$\dot{r}_{des}$	Desired velocity vector	[m/s]
$\ddot{r}_{des}$	Desired acceleration vector	[m/s^2]
$\ddot{r}_{i,c}$	Commanded acceleration vector	[m/s^2]
a	Inertial system	[-]
i	Vehicle body system	[-]
p	Angular velocity related to longitudinal direction	[rad/s]
q	Angular velocity related to transversal direction	[rad/s]
r	Angular velocity related to vertical direction	[rad/s]
k_F	Experimental constant of force produced by a motor	[N/rpm^2]
k_M	Experimental constant of moment produced by a motor	[N/rpm^2]
q'	State of the rigid body	[-]
$\dot{q}$'	Rate of change of the state of the rigid body	[-]
X	Longitudinal direction	[m]
y	Transversal direction	[m]
Z	Vertical direction	[m]
a_{min}	Minimum acceleration produced by the motors	[m/s^2]
a_{max}	Maximum acceleration produced by the motors	[m/s^2]
M_i	Moment produced by each motor	[Nm]
I_i	Moment of Inertia related to the rigid body over the different directions	[kgm^2]
$\dot{x}$	Longitudinal velocity	[m/s]
$\dot{y}$	Transversal velocity	[m/s]
$\dot{z}$	Vertical velocity	[m/s]
$\ddot{x}$	Longitudinal acceleration	[m/s^2]
$\ddot{y}$	Transversal acceleration	[m/s^2]
$\ddot{z}$	Vertical acceleration	[m/s^2]
u_1	Net body force	[kgm/s^2]
u_2	Body moment	[Nm]
u_3	Body moment	[Nm]
u_4	Body moment	[Nm]
e_p	Error related to the proportional gain	[-]
e_v	Error related to the derivative gain	[-]
k_d	Derivative gain	[-]
k_p	Proportional gain	[-]
T	Time related to the trajectory	[s]
S_i	Total amount of time over the waypoints	[s]

p_i	Polynomial equation that generated the trajectory	[-]
T_i	Time between waypoints	[s]
T_u	Ultimate period related to the Ziegler-Nichols method	[s]
k_u	Ultimate gain related to Ziegler-Nichols method	[-]
$\hat{n}$	Normal vector of the trajectory described by the vehicle	[-]
$\hat{t}$	Tangential vector of the trajectory described by the vehicle	[-]
$\hat{b}$	Binomial vector of the trajectory described by the vehicle	[-]

Greek letters

ϕ	Roll angle	[rad]
θ	Pitch angle	[rad]
ψ	Yaw angle	[rad]
ϕ_{des}	Desired roll angle	[rad]
θ_{des}	Desired pitch angle	[rad]
ψ_{des}	Desired yaw angle	[rad]
$\dot{\phi}$	Derivative of roll angle	[rad/s]
$\dot{\theta}$	Derivative of pitch angle	[rad/s]
$\dot{\psi}$	Derivative of yaw angle	[rad/s]
ω_i	Angular speed related to each motor	[rpm]
α_{ij}	Coefficients related to the restrictions imposed on the polynomial	[-]

Chapter 1: Introduction

1.1 Motivation

On the recent years, Earth has experienced a huge population growth. Over many supply issues it has brought, energy is one of them. The demand for this core asset, is a must to sustain people's needs. Thus, it is a must to share the best of each energy source, once it is very difficult for a single one to support the whole world needs'. Moreover, this process should also be sustainable, since the most efficient fuels, like fossil, possess limited resources, not to mention the pollution associated with the process. With this, urges the need to empower renewable energies, aiming for a long-term prosperity.

Nevertheless, actions can be taken regarding processes that are doubtlessly hazard. Choosing renewable energies over the fossil ones, is surely a greater step towards sustainability. Among renewable energy sources, wind is used in large-scale electricity generation. This technology, has been rising in the recent years. [1] Aspects like providing a mature technique, plus commercial prospects have enhanced the large-scale implementation. This can be proved by Global Wind Statistics, which shows that just in the year of 2016, 13.926 wind turbines were installed in Europe. [3]

Based on Global Wind Statistics, it is known that 486.749 wind turbines were installed worldwide by the end of 2016, that corresponds to a 54.600 MW generated by the whole world, just during the year of 2016. [3]

Although, this also comes with some issues. The big structures maintenance represents a difficult task for human action to achieve since the height requires numerous support, effort and time to accomplish. Allied with composed materials, big structures can have higher efficiency than with conventional materials, but also harder to conduct inspections though devices, such as cameras.

Wind turbines blades, due to their dimensions, constitute a difficult structure to conduct maintenance on. Since these devices are power by wind, taller structures enable to access stronger winds, that are intrinsically related with power generation. [4] Moreover, their energy harvesting is cubicle related with their height [5], being this the main reason for their dimensions.

To make renewable energies competitive with the traditional energies sources, it is a must to run them as efficient as possible. A key aspect that compromises their efficiency, is the state of the blades, once their return of power generation can be compromised up to 30% regarding their efficiency. [2] This implies a continuous monitoring and repairing. As it can be seen, it is important to make their maintenance more accessible as possible, giving repeatability to the process and being independent of the wind turbine height.

Nowadays, most of the inspection methods existent for this end are either inadequate, outdated or expensive. The traditional methods are: Rope Inspection, Service Platform, ground based inspection and Hydraulic crane. Most of these are either safety hazard, rope inspection, unviable for reasons of low quality image inspection, ground based inspection or even too much expensive, hydraulic crane.

With this crisis, RPAS appear as an innovative vehicle, with great potential, providing the desired tools for the development of solutions, that can be extended on a wide range of civil structures, presenting a comfortable way to execute it on dangerous and difficult access spaces. In particular, provides fast and accurate inspections, enabling to improve the viability, with the plus of lower expenses. [6] [81]

With the use of RPAS, to execute visual inspections, becomes possible to provide repeatability to the process. By obtaining pictures from the same position, a correlation can be established between previous photos, helping to determine the state of the blade. It is believed that this process can be highly advantageous, once wind farms with 100 wind turbines, can significant increase revenue. Related to this issue, the maintenance strategies, to maximize his lifecycle, should be executed continuously, otherwise, the cost of repair will increase significantly, as it can be seen on Figure 1.

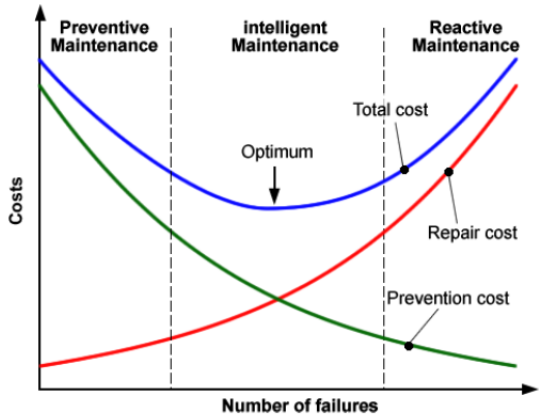

Figure 1: Maintenance cost of Wind Turbines. (Obtained from [7])

Additionally, when all the inspection data is collected, the question of obtaining a report can be expedited, providing better scheduled tasks to minimize overall operations and maintenance costs.

1.2 Introduction to the project

In cooperation with the Universitat Politècnica de València (UPV), it was proposed to develop a creative solution, facing the traditional ways of Wind Turbine's inspection. This would provide an efficient and safer inspection by bringing technology and qualified technicians together.

On this project, it is generated an application, over which are defined waypoints on a 2D map, that represent the localization of Wind Turbines. Over these, is defined a trajectory that incorporates the blades, from root to tip. With this, is possible to capture photos producing a visual inspection. The flight proposed is design for a velocity of 0,75 m/s on which would start and rest at the center of the Wind Turbine. The model selected to provide the measures of the blades and the tower was the GE 1.5sle, shown of Figure 2. [8]

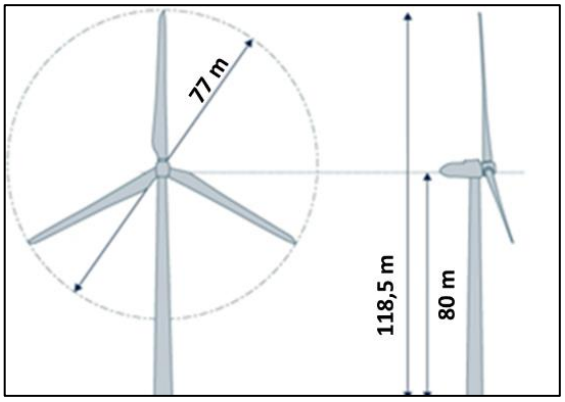

Figure 2: Wind Turbine model GE 1.5sle.

Over a wide range of possible vehicles, for this research was used the RPAS Phantom 4. It contains numerous advantages, that will be mentioned below, such as complex sensors and widgets that can be adapted to singular purposes by implementing simple methods. These parameters were included on the code, to provide a better approximation for the reality.

The Phantom 4 flight controller, has the capacity of store critical data from each flight on the on-board storage device. This vehicle has a Vision Positioning System when flying indoors or environments without GPS, enhancing the hovering state. Regarding precision, it has incorporated dual IMUs and compasses, granting redundancy.

The gimbal provides a 3-axis steady platform for the camera, which can be tilted within 120º range. It also includes safety modes as Failsafe, air braking features and Return-to-Home. This minimizes the risk of accident associated with the RPAS, even if the control signal is lost. The flight data is automatically recorded into the internal storage of the aircraft. This includes telemetry, aircraft status information and other parameters.

This RPAS is equipped with Obstacle Sensing System that constantly scans for obstacles. Thus, collisions can be avoided, once an object is detected, the RPAS will maintain a defined security distance, generating an alternative path. His range of detention system is from 0.7m to 15m.

The DJI Vision Positioning System uses ultrasound and image data to help the aircraft to maintain its current position. This Vision Positioning System is constituted by two ultrasonic sensors and four monocular sensors. This Vision Position is activated automatically when the aircraft is turned on. The GPS Positioning has a vertical hover accuracy of 0.5m and horizontal hover accuracy of 1.5m. The visual position system has a vertical hover accuracy of 0.1m and horizontal hover accuracy of 0.3m. [9]

Parameters of Phantom 4	
MTOW RPAS	1380 g
Propellers	240 x 127 mm
Diagonal Size (Propellers excluded)	350 mm
Maximum ascendant speed	6 m/s
Maximum descendant speed	4 m/s
Maximum speed	20 m/s
Maximum Service Ceiling Above Sea Level	6000 m
Maximum Wind Speed Resistance	10 m/s
Maximum flight time	28 min
Photo quality	12 MP
Satellite Positioning Systems	GPS/GLONASS
Video quality	4k at 120 fps (frames per second)

Table 1: Specifications of Phantom 4. [9]

1.3 Project aim and document structure

The main objectives of this projects are:

- The trajectory must obey to the restrictions established by the dynamics and input of the RPAS.
- The control law applied to this system must be able to re-plan the trajectory at each control update, and apply the new inputs.
- The application developed has an aim for semi-autonomous inspection over a diversified number of structures. Although, once it is on the first stage of development, that with the right development, will enable to improve inspections to a whole new level of quality. [10]
- The application must be general, enabling to be used for other types of structures.

On the Chapter 1, will be presented the motivation regarding this project, the description of the RPAS used, as well the basics of each state that took part while developing it.

On the Chapter 2, will be presented a state of art referring the actual techniques used for wind turbines inspection, the means to achieve it and also the basic aspects regarding Java language.

On the Chapter 3, will be described all the steps along the creation of the Android application, as well the requirements associated with it.

On Chapter 4, will be presented the MATLAB experiment, defining the model used to simulate a trajectory on this software, and how it is planned to associate it, with the Android application.

On Chapter 5, will be shown the results regarding the Android application and the MATLAB simulation.

On Chapter 6, will be presented the conclusions retained from this project, some future works which can be developed regarding further investigation.

Chapter 2: Framework for Wind Blade inspection

Wind turbines are exposed to extreme conditions, as strong wind, sun, rain and erosion. Even though, composite materials manufacture is taken in highly consideration, they possess a life endurance associated. Thus, to avoid the replacement of these structures, several inspections are taken, regarding the detection of initial damaged areas, enabling his reparation, saving huge amounts of money and avoiding energy harvesting losses.

Since these problems are produced over a wide range of environments factors, there was created in 2010 the Blade Reliability Collaborative (BRC). [11]

With this, was possible to obtain essential data from the supervision of these surfaces, relating it to the non-destructive inspection techniques, either while their manufacturing or in the field. With the creation of this organization, limitations regarding the current inspections techniques and how to develop improvements were possible. Simulating their defects propagation, information related to the inspections and even establish cooperation between different agencies and industries was possible. This result in a maximization of the lifecycle of wind turbines. [11]

2.1 Inspection Strategies

Wind turbines are manly composed by fibre-reinforced composite. Due to their variable thickness and multilayers, it requires high qualified technicians to provide good control over these materials. In order to maximize their lifecycle, some non-destructive techniques were defined, enabling to execute them on site monitoring. [12]

Most of the times, the initial damaged areas, are due to defected manufacturing. As so as, the first inspection should be done is on the factory.

Factory inspections [13], are a fundamental part of the production, once there are some processes that should be supervised, to guarantee that their execution was flawless. The blade has two shells, to which should be bonded correctly. Thus, being this a critical aspect to detect, precise methods are required, such as ultrasonic scanner and thermography. These demand precise machinery, that can only be used properly off-line. This inspection allows to detect fibre wrinkles, dry and weak areas, porosity, among other defects. Other materials, like the receptors, should be supervised in order to his correct installation.

Another aspect that should be taken in consideration is the rotor balancing. [13] Every structure must be in balance, otherwise it works his way until it reaches this stage. In practical terms, by assembly a rotor on these structures producing uneven distribution of his mass or blades with an angle deviation, might cause sever disturbances. To avoid it, methods like reflectorless lase distance measurement shall be used.

Thermography [14]

This is a quite common method among inspections that are only applied when the structure is off-line. It takes use of infrared cameras in order to detect variations on the blades temperature, which is directly related with the cracks origin.

Ultrasonic testing techniques [14]

This technique allows to detect defects and guarantee the quality of the surface and the under-surface layers' present on the blade. By beaming ultrasonic waves, their propagation over the material allow to determine quite precisely the place where the material suffers from a discrepancy. With this is possible to indicate anomalies, either related to bad adherence between layers, or even defects of delamination. The ultrasonic signals are based on time algorithms.

Vibration Analysis [13]

This technique is most used for the crack detection and growth. Making use of sensors operating between 0,01 – 100 kHz. Usually this range is divided in three, so the position transducers work at low-frequency, velocity sensors work at middle-frequency and the accelerometers at the high-frequency.

2.2 Inspection Methods

After the assembly, to examine this type of structures some methods are considered, such as helicopter, RPAS, platform or even ground inspection.

Helicopter Inspection [13]

Regarding to this type of inspection, it requires a significant number of wind turbines, such in the case of wind farms. This is due to the huge cost associated to the helicopter consumption. By taking advantage of this inspection high resolution cameras and thermography can be combined to provide a more accurate idea of the wind blade state.

Drone Inspection [15] [13]

This type of vehicles can execute the same type of supervision as the helicopters, with the advantage of being a cost-efficient solution. Low-cost services, that incorporate high-performance vision sensors that take advantage of aerial sensing platforms are some of the highlights that this type of vehicles offer, which allow to overcome certain boundaries, in order to achieving autonomous visual inspection. They can execute two types of inspection, static and dynamic.

The static inspection includes all the methods mentioned on the Helicopter Inspection with the plus side of allowing to define Regions of Interest (ROI), establishing a more detailed comparting with future inspections. Although, it does not provide a drainage system inspection.

The dynamic inspection is done while the wind turbine is working, enabling to detect abnormal deformation or even signs of fatigue material.

Blade Inspection Platform [13]

This inspection requires a platform on the wind turbine, allowing to execute a full inspection. The installation of the platform is quite expensive, and requires considerable stopping time. This assures the same goals as the rope inspection. Although, it is quite dangerous.

Blade Inspection Rope [13] [14]

This type of inspection is relatively fast, allowing the technicians to perform "Tap test", which help to determine the damage extension of hidden imperfections. It also allows to perform thermography inspections, lightning protection system and drainage system assessed. This is the method which provides the most complete inspection over these structures. Although, the conditions of work can be quite dangerous, promoting accidents among workers.

Optical Blade inspection from ground [13] [14]

This type of inspections has the advantage of reduced cost, since it is executed from a long-distance lens. Being executed from ground, has a brief duration, which reflects on a short stopping time of the wind turbine. With the photos, it is possible to create a database, establishing comparison between others in a timeline.

Although, this method has a wide range of limitations. Only noticeable cracks present on the blades are noticed, once the distance of the camera is quite far. Thus, small defects and small cracks will not be detected. Among other aspects, to determine the real state of the blade, this is not a viable method.

Blade Internal Inspection [13]

By taking advantage of solar light, UV lamp, thermography and others at a close distance, allows to detect core defects, delamination's, structural defects, between other defects. This grants great accuracy to determine the state of the blade, although has cost associated and time to execute the maintenance.

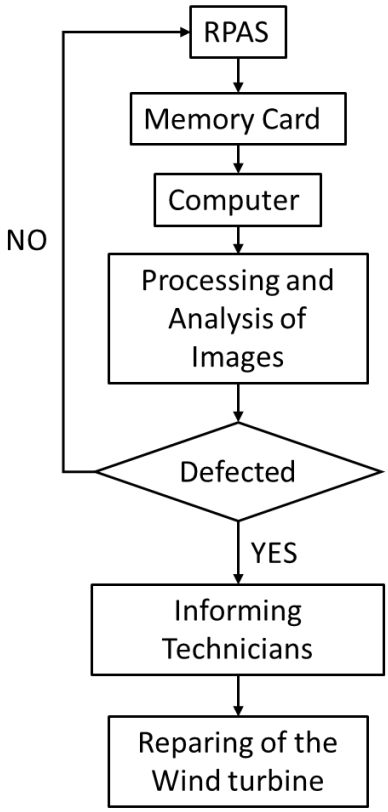

Figure 3: Information path from the inspection with a RPAS, to the point where the inspection is required.

2.3 Java Basics

Once the product obtained was based on Java, it is essential to provide some basic understanding and clarify some fundamental aspects on which this project relies.

2.4 What is Java?

Java is a programming language. Based on the C++, C and Objective-C, arouses object-oriented programming model, that instead of being based in “actions” and logic, focus on objects and data. [16] Among other features, this language became so popular because of the portability that it offers in network, requiring only a Java virtual machine. Being an object-oriented language, provides a class of objects that can inherit the code common to the class related. It also offers flexibility, by executing on each client needs, not in a general server. [17]

An object, can be defined as a software bundle characterized by his state and behavior. The state includes the name, version among others. The behavior is shown by the methods defined. [18]

The objects are the first priority to define when designing a program. They must be identified and established relations between them. Since this software system is complex, diagrams can simplify their understanding, illustrating how the data flows and relates itself in the practical

way. This process is usually called “data modeling” [19]. Not only help’s the analysis of the project but also ensures that the requirements are fulfilled. This process can be divided in four stages: conceptual, enterprise, logical and physical. The conceptual data modeling is related to identification of high level relationships between entities. Enterprise data modeling is when the unique requirements are established. Logical data modeling is the base for the physical stage, being here defined entities, attributes and relationships involved. The physical data modeling is the implementation of the logical stage. [20].

Objects can also be described as units of code. They are categorized in classes or subclasses enabling to share models, methods and definitions simplifying the whole code. Classes can be defined as the blueprints from which objects are created, defining his state and behavior. [18] These objects are what is ran on the computer [21]. The classes on which they are divided are templated definitions that contain real values instead of variables. [22] The difference between the class and subclass is that the deeper it is, the more specific it becomes. Thus, subclasses can inherit some characteristics of the class, although they are different between them. A practical example of an Object is the Camera, that will be mention with more detail on the Methods section. This has associated with it a set of Methods, as an example, to start recording a video, as it can be seen on Figure 4.

```
private void startRecord(){
    final Camera camera = DJIDemoApplication.getCameraInstance();
    if (camera != null) {
        camera.startRecordVideo(new CommonCallbacks.CompletionCallback(){
            @Override
            public void onResult(DJIError djiError)
            {
                if (djiError == null) {
                    Toast.makeText(MainActivity.this, "@string/recvid", Toast.LENGTH_SHORT).show();

                }else {
                    Toast.makeText(MainActivity.this, djiError.getDescription(), Toast.LENGTH_SHORT).show();

                }
            }
        }); // Execute the startRecordVideo API
    }
}
```

Figure 4: Methods for Camera, in order to record a video.

Java is also a platform that contains a set of tolls, components and elements that allow the development of an application. An application program is the use of a technology combined with an operative system, designed to simplify his use. Applications can also transfer data between platforms. To this end, must be established an application program interface (API) which provides the requests and the means to obtain it. [23]

In Java, the classes are defined only the data in needs to be deal with. Thus, when an object is run, it is not allowed for the code to access the whole program data, improving the standard security and avoiding data corruption. [24]

2.5 What is a Software Development Kit?

With the exponential growth of applications, comes the need of providing tools like libraries that help a program to take shape in diverse platforms. By providing these tools, comes also the need to establish some procedures for the application to communicate with the operative system, setting aside the need to code the common functions. Since it is more common having ideas for an application, than possess the programming skills, most of these kits already include a saved sequence of commands, called Macro's. A Macro can be defined as a statement that has been already implemented, that can be saved and called afterwards. Most of times, includes a variable parameter. This is quite useful, owing to defining a series of instructions that will be used multiple times along the project, not only saves time, but also makes the code more concise. This also plays as an advantage for another user's that are working on the same project, or in a similar one. [25]

In addition, it is common for the SDK provide a sample code, that can help user's providing a base idea that can serve as a guide. These SDK, also require an IDE. This will provide essential items, as an editor for writing source code, debuggers and compiler. [26]

2.6 Android Overview

Android takes part of most concepts of Linux, such as open source and the operating system. His main purpose was for smartphones and tables, devices controlled by a touchscreen. Nevertheless, it is also applied nowadays in others, such as TV's, watches and other common electronic devices. Android started in the year of 2003 with Android Inc. being incorporated in Google in 2005. The first mobile device operating with the system of Android was sold in October 2008. [27]

After the first version was released, the need to implement new functionalities and fix some issues created new versions. Each latest version is identified by an API Level. This is very important, since it helps to determine whether apps are compatible or not. [27]

Thus, when creating an application, there are some fundamental points to consider, in order to ensure compatibility. Some of these points are: the API Level, which is directly related with the Android versions used, and the dimensions of screen that it will be used. Elements like statistics of the most common screen sizes and API Levels can be found online released by Google. [27]

2.7 What is Android Studio?

Android studio was the software used to process all the information developed allowing to produce the application. This is an IDE, that provides useful tools for application development. [28]

This software is composed by a flexible Gradle base building system, an emulator which enables to run the changes made on the application without a real device or even building a new APK. [28]

Since it is a program used worldwide by a big range of people, comes the need to establish some security procedures such as the principle of minimal privilege. In simple words, it is descried as access only to the components that it will work with. By establishing this, empowers a secure environment since an application will not be allowed to access other parts without the user's permission. [30]

Lately, Android has become a very popular operating system. In order to maintain this vantage, Google is constantly developing tools for new programmers. This will assure less difficulties putting their ideas into practice. All these advantages are available in Android Software Development Kit, that enables people to code, test and debug their application. [27]

These code files are compiled by Android SDK into APK, represented by the suffix .apk in the archives. These represent android packages that contain all the content of the application, which is also used to install it and are a variant of Java's JAR format. [31]

The application presented in operative systems run isolated from each other. This is possible by generating a VM for each process. [32]

This software is constituted mainly by two bars. One that allows you to run and launch the applications, and a navigations bar, providing tools to edit the project. There is also an Editor Window, allowing modifications on the code.

There are some default module files in Android applications that are essential, such as:

Android Manifest: This is the main file of the projects in terms of being that essential information is defined, such as: minimal API Level, files and which of them will take part in main activity, software and hardware components that are requested by the application, and the API libraries that the application requires to be bound to. Components not defined here will be considered inexistent in the application. [33] [34]

APP Resources: When it is required additional resources such as XML layout, UE stream, animations, images or even colors, these are placed in a file named "res". Establishing this procedure, it is possible for the user to request different configurations. They will provide an interactive experience with the user when requested different configurations. [33]

2.8 What is an App?

App corresponds to the abbreviation of the word application. It can be defined as a software programs for computer's, smartphone or another device. They are constituted by Java code and resources files from each app that takes part of the Android's architecture. [31]

As mentioned above, applications come as ***.apk*** files. These files are produced by using the tool AAPT (Android Asset Packaging Tool). This tool is part of Android SDK which compiles the code,

images and libraries. With this, they are optimized, enabling to execute in Android RunTime. [31]

2.9 Android App permissions

Android implements the principle of minimal privileges. This means that applications by default only have access to components they need to execute inside the application. With this is created a safe execution environment. However, sometimes applications need to access system services or access data from other applications. To achieve this, there are some alternatives. The most commonly used, is to ask the user's permission to access data or use parts of the device. For example, the user must grant permissions when the application desires to access text messages, camera, or others when working in a part which is not granted previous permission to access it. [35]

2.10 What is an API?

An Application Program Interface is the series of subroutine definitions and protocols for building application software. [36]

This is very important once it allows applications to access to functionalities that are already exist in others, saving time and simplifying the process. The API documentation is usually provided to facilitate usage. By incorporating the commands already defined, is possible to send them to programs running on the server that the application is working on. With this is possible to access to resources only available on the server, like logins and passwords. [37]

To establish a contact between a class and the outside word, an interface must be created. This contact is built based on methods. An example can be given as the camera created for the RPAS application, shown on the Figure 5, more precise, the action to stop recording.

```
private void stopRecord(){
    Camera camera = DJIDemoApplication.getCameraInstance();
    if (camera != null) {
        camera.stopRecordVideo(new CommonCallbacks.CompletionCallback(){
            @Override
            public void onResult(DJIError djiError)
            {
                if(djiError == null) {
                    Toast.makeText(MainActivity.this, "@string/recstop", Toast.LENGTH_SHORT).show();

                }else {
                    Toast.makeText(MainActivity.this, djiError.getDescription(), Toast.LENGTH_SHORT).show();

                }
            }
        });


    }
```

Figure 5: API implementation to stop recording, regarding the camera.

2.11 Application Components

Every Android application is constituted by components. These components can be described as essential building blocks which have entry points, allowing the system or users to access it. These can be divided in: Activities, Services, Broadcast receivers and Content providers. Each one has a distinct lifecycle that defines how they are created and destroyed. [38]

Activities

Activities are actions that allow the user to interact with the application, being the entry point for it. During activities transaction, they also store the data. Thus, this is a very important aspect of an application.

Each activity is assigned a window. An application is constituted by large number of activities, that working together avoid inconsistences, being each one independent from the others. [39] As the application is initiated, the main activity shows up. From the main activity is it possible to launch others, although, as mentioned above, all independent from another's. This part of the application can be seen as a state machine, once launched, stays pendent for user actions. Even though the programmer has now the capacity of controlling the way this starts, the steps executed on each stage can be defined.

The stages take part of an application are: Creation, Execution, Redundancy, Pause, Stop and Destroy. To the whole stages is called Life Cycle, as represented on Figure 6, and it is controlled by the Activity Manager.

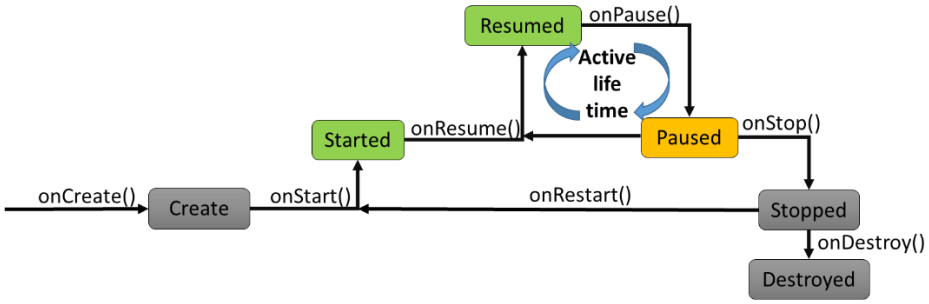

Figure 6: Activity life cycle, where grey color represents invisible states, green color the active states and the yellow one, partial visible states. (Based on [40])

Each transaction between states represents a "callback" on the activity. Some of these methods are overwritten automatically in the projects of Android Studio, a good example is the onCreate. This method happens when there is a transaction starting when the user clicks on the icon of the activity, till his execution. An example can be seen on Figure 7, although the whole code related to this method will not be displayed due to his length.

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
```

Figure 7: Example of a onCreate method.

The states of the life cycle illustrated on Figure 6 are described as: [39]

onCreate(): This callback fires when the system creates the activity. The onCreate() method sets up some fundamental parts of the activity. Some of those are the declaration of the users' interface and configure some of the User Interface (UI). Once this is done, the activity enters the Started state, meaning that has been created, although is not visible yet.

onStart(): This callback appears when the Activity enters the Started state. This callback makes the activity visible and becomes interactive. Sometimes, the broadcast receiver is registered on this stage. After this, the activity enters in Resume stage.

onResume(): On this stage, the application is enabled to interact with the user. The application will stay on this stage until another action is trigged. When this interruption occurs, the application enters in the on the Paused stage.

onPause(): This callback is considered a signal that the user is leaving the activity. This method is used so activities like music of animation associated with the application being executed are not displayed anymore.

onStop(): This callback is triggered when the activity is no longer visible to the user. On this method, usually the resources associated with it are released, such as unregister from the broadcast receiver.

onDestroy(): This callback is used right before the activity is destroyed. [41]

Even though the application can contain different activities on his structure, it must be defined a main activity, specifying it on the Android Manifest file.

Services

Services enable a task to be extended in the background of the system. These components of the application lack of graphical interface. They are a very important, once the user is allowed to perform long-running task on the background while using a different application.

These services can be defined by their life cycle: [42]

- Foreground: This type of service is set to run even if the user is not interacting with it. They also display a status bar icon. A common example is reproducing music on the background for the user.
- Background: This type of service allows to execute an action, even though it is not being noticed by the user. An example can be completing a download.

- Bound: This type of service happens when one app requests another. Thus, it provides an API to another process. With this, is possible for the system to know where are the dependencies established between two processes.

Nevertheless, a service can be defined as private, on the manifest file. This will prevent the access from another application. [42]

Broadcast receivers

Broadcast receivers, like services, lack of a graphic interface. Although, they might display an icon on the status bar. These are the components that alert the user of actions which the priority equals or overlaps the current activity. They usually display alerts or notifications of events. Many of these occur from the system, such as an income call or system notifications. [41]

The Broadcast receivers differ from services in different ways, but most for the reason that these wait for an event, in order to run, which does not happen on services. [43]

Content providers

Content providers manages the access of components to the data. This data can be stored in persistent locations from more than one application. With this, content provider manages the share of data between the applications. This is done with previously defined security mechanisms. [42]

Moreover, this component also provides the implementation of a standard set of APIs. With these, is possible to perform transactions between app's. This aspect provides a unique functionality to the Android system, which any app can start another app's component. For example, if the user desire to take a photo from a distinct app from the camera, instead of developing an activity to capture a photo, it can simply start the activity in the camera app, allowing to capture a photo. Afterwards, by managing data, the photo is returned to the app which it was requested. By implementing this subroutine, it might give the sense as if the camera is a part of the app. This functionality of sharing files, simplify the code and eliminates the overlapping of the same activities. Although, they might not allow the application to modify the data. [44] [34]

Intent

An intent can be described as a general operation to be executed, that can be used by activities, services and broadcast receivers. Intents can be used to display a wide range of utilities, from transferring information or even objects, to send the order to launch applications. On this last, intents represent the passive data in order to display abstract actions to be performed. [45] Moreover, regarding broadcast receivers, intent will display useful information related to the state of an action, in order to maintain the user informed. Although, it must be mentioned that

for content providers, unlike the other application components, they are not triggered by intents, instead they require a ContentResolver. [46]

Furthermore, detailed information will be provided on Methods' section, Chapter 3.

2.12 What is Gradle?

Every application needs to rely on a building system in order to declare the project configuration. This is the part where Gradle is required. By defining a directed cycle graph (DAG), it is possible to prioritize actions, once multi-projects, many times required in application, can be quite extensive. By providing a multi-project build, it is possible to define which parts are up-to-date, assuring that the system will not become overloaded. This is an open source system, that took the basis from Apache Ant, although it uses Groovy-based domain-specific language (DSL) to declare the project configurations. [47] [29]

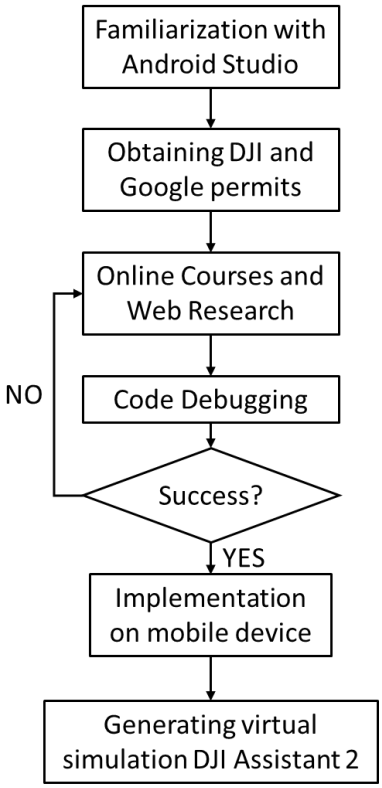

Figure 8: Path followed in order to developed the present application.

Chapter 3: Drone Mission Maker

In this section, will be described with detail the application created. The main reason for his creation is to take advantage of the capacities regarding the RPAS and connect them with wind turbines. This is an emergent subject which grants great advantages allowing also the implementation in an industrial scale.

The difference between other projects, is that this one, takes advantage from low cost development, to the point of enabling independent people to generate it, requiring only a phone with Android operative system, the Phantom 4 and a computer.

Below, will be defined the standards that must be fulfilled in order to grant sustainability and expand this application to the market. Technical aspects of the application previously studied and their inherent improved aspects of the functionalities currently offered within this application will also be explained.

Another aspect that should be mentioned, is the flexibility and improvements that can be obtained in the near future, subject that will take part on the Chapter 6, part of future works.

3.1 Previous licences required to the development

To the accomplishment of this project it was required two licences, one from *Google* and another one from *DJI*.

The first licence allows the use of Geolocalization of our application on the mobile device, once that Phantom 4 has already incorporated his own. To obtain it, users must enter the developer's website of Google, [48] on which a registration is required. There is also a pricing table available to consult, once that for successful applications, there are certain taxes involved with the service provided by Google. [49]

On the second licence, from the company on which the RPAS had been produced and developed, *DJI*, is required. Thus, a registration as developer on the *DJI* website is required, in order to obtain a DJI key. [50] This, is generated based on the software platform, type of Software Development Kit and category of application being used.

This key is completely free, although there is a possibility to make a registration as premium developer, providing additional tools and tutorials bringing possibilities of wider functionalities of the application, as well clarifying tutorials.

Although this includes costs, that would provide the possibility to save hours of work and enable to be in touch with the latest advances on this area. Taken in consideration, subscribing the "premium developer" is a small investment to what might be a greater good.

Another point that should be taken in consideration is that, this application is on its alfa stage. This means that every licence was obtained allowing the basic concepts to work with no

restrictions. Nevertheless, before uploading this application to *Google Play Store*, official application system for *Android*, some technical aspects should be taken in consideration. It must be assured that this is compatible for all clients' devices.

Thus, some aspects should be assured, more specifically, the ones that *Google* imposes, in order to avoid this application getting declined. [51]

3.2 Previous knowledge

In order to obtain a correct start to the creation of this applications, some previous knowledge is required. More specifically, being a user of *Java* language is strongly advices. Some basic aspects were already described earlier on the introduction section.

To possess knowledge about *Java* is fundamental, regarding basic and complex structures. Thus, for some of these, will be explained in a very synthesized way.

Nevertheless, to maintain an efficient work, it is required a team familiarized with this kind of tools. This is important, so it is possible to start from the idea that basic concepts are already acquired, such as functions hierarchy, methods, operators or even matrixial structures. These are basic elements necessary to grant coherency and cohesion between all the sub functions presented on this code.

Having said that, the concepts more taken in account, on this code were:

Code files

These are the text files that assign value and structure the functions to project the application, that can be found on the left side of *Android Studio* on the *Progress Bar*, illustrated by a blue circle with the letter *C* by the middle of it, which represents a *Class file, as it can be seen on* Figure 9.

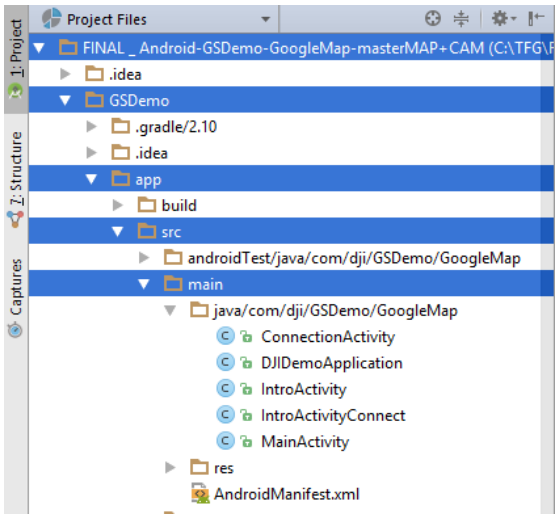

Figure 9: Localization of the files on Android Studio.

Commonly, when it is obtained a correct initiation in the background of Android Studio, this is known. Nevertheless, it is useful to differentiate it from the *Layouts*.

Layouts:

These are also text files. Although, the contained code, is defined in a different way compared to *Class* files. This is due to their function on the program.

Layout files are related to the graphic interface of the program, the one seen by the user, that have as base class the *ViewGroup*. This graphic interface is called on the main code of *Java* to be visualized later, according to the parameters defined on the *Layout* file, displayed on the screen.

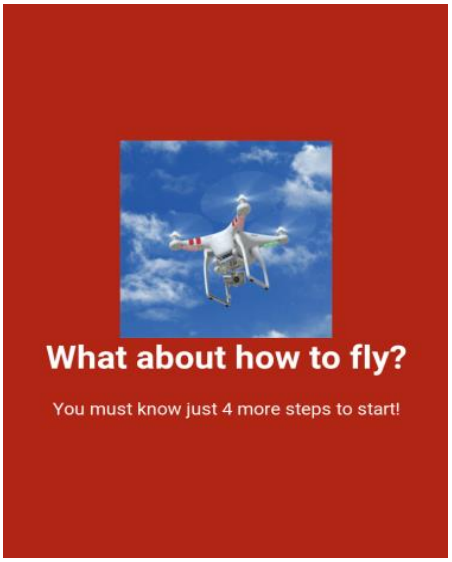

Figure 10: Visual Interface.

```
<?xml version="1.0" encoding="utf-8"?>

<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical" android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="@color/bg_screen1">

    <LinearLayout
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerInParent="true"
        android:gravity="center_horizontal"
        android:orientation="vertical"
        android:weightSum="1">

        <pl.droidsonroids.gif.GifTextView
            android:layout_width="181dp"
            android:layout_height="181dp"
            android:background="@drawable/flyintro" />

        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="What about how to fly?"
            android:textColor="@android:color/white"
            android:textSize="30sp"
            android:textStyle="bold" />

        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_marginTop="20dp"
            android:paddingLeft="40dp"
            android:paddingRight="40dp"
            android:text="You must know just 4 more steps to start!"
            android:textAlignment="center"
            android:textColor="@android:color/white"
            android:textSize="16sp" />

    </LinearLayout>

</RelativeLayout>
```

Figure 11: Code part in order to generate a visual interface.

This analogy helps to understand the load of code Figure 11 that requires to generate the Figure 10. Thus, to present this simple graphical interface, it is necessary to include different concepts like the location of the RPAS' Gif, the text and as well use the different types of *Layouts*.

The common types of *Layouts* used on this kind of applications are:

- *LinearLayout:* On this type of Layout, it is given an automatic ordination of the graphical elements in a linear direction, that can be either horizontal or vertical. This is the simplest type of Layout. Although it is not recommended if the objects need to be at a certain location, once it already includes a predefined disposition for the elements. [52]

- RelativeLayout: On this type of layout, it is allowed to locate an element at a define position aligned either with the border of the screen or with a relative position based on a previously defined element. It is commonly used which enables to eliminate nested view groups and also maintain the layout hierarchy equal, improving the application's performance. [53] [54]

- FrameLayout: This is the most complex type of Layout, from the three mentioned here. Most of the times, it is only used when it is required to block a part of the screen, enabling to display a single item or a specified activity. In this case, was used to the map. [55]

For these reasons, to create graphical interfaces containing a diversified number of buttons, reserve a place to display the map and the option to be switched for the camera of the RPAS,

it is needed to establish a diversified number of combinations and mix the different types of Layout provided inside of a ".xml" file, that are the once which contain this type of code. It is also required to use the options of visibility that will be explained later.

To the initial of the tutorial it is a must to separate the *Layout* in two sections:

- The top section, which will have a sequence of slides with information, that can be switched by sliding the screen to the side.
- The lower section that will contain the progress state, to let know the current slide the user is, providing an idea of the total number of existent informative slides.

This introduction tutorial requires another additional code file, that comes with the need to relate different kinds of *Layouts*, the ones moving on the top section with the ones settled on the base of the screen, as it can be seen on Figure 12.

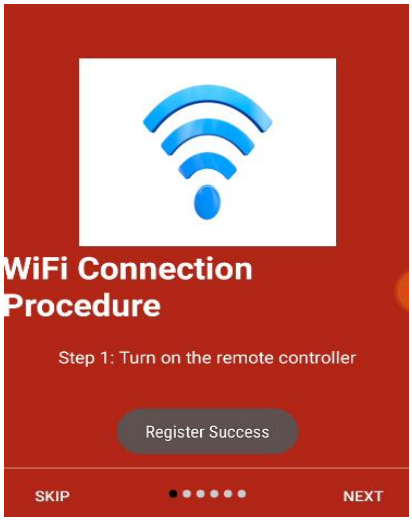

Figure 12: Interface separated from the tutorial.

3.3 Visibility options

This option will not be illustrated, for the simple fact that his complexity does not require it. Nonetheless, it is necessary to keep it in mind, once it has a huge set of advantages. On this case, it was used to add up the cameras and map interface to a single one, like has been said before, just by the usage of one button.

This, will include or exclude the visibility of some buttons in function of the part of the interface that is requested.

This is possible with the command $AnyObject.setVisibility(View.VISIBLE/INVISIBLE)$. From this is possible to activate or disactivate.

Figure 13: Visible interface regarding the Map part.

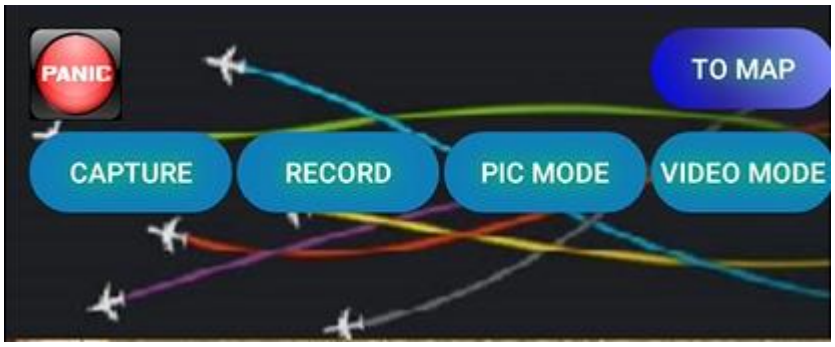

Figure 14: Visible interface regarding the Camera part.

As a result of this functionality it has been possible to combine the two interfaces with a very simple procedure, avoiding the exchange of files within the code between interfaces and methods, also known as *Intents,* which will be explained on the next section.

3.4 Exchange between Class code files: Intent

The command *Intent*, is based on a simple structure as it is possible to obverse on Figure 15, that enables to exchange information between the file that is being executed, to the one that is desired to execute, with secondary tasks, until they are required. [45]

```
private void launchHomeScreen() {
    startActivity(new Intent(this, IntroActivity.class));
    finish();
}
```

Figure 15: Intent structure.

This action shall not be mistaken by the one that is received when it is established an interaction with an element, like a button. On this particular analogy, it is obtained a feedback from the function assigned to the button inside of the code. Although the information accessed is located in the general Class file. With the *Intent*, it is possible to access to other *Class* file between the ones available on the Java folder, that can be seen on Figure 15, as well the basic divisions and *Class* files needed on a project of medium scale.

3.5 Exchange between Class code files: Intent

The application icon is a fundamental aspect of every project, that most of the times, is not given the deserved importance once coders focus only on the code.

This small detail is the one which will provide the first impression to the users. Given the state of this project, the icon was projected for different sizes, once different devices may require it. Thus, resorting to the *Launcher Icon Generator* [56], it was possible to create an icon based on a photo provided. The result was the Figure 16.

Figure 16: Application icon.

The existent sizes for website icons are: [57]

- Low Density Pixel Image (LDPI): This format is not provided from the website once it does not have a common use, contains 36 x 36 pixels
- Medium Density Pixel Image (MDPI): This format is provided by the website, contains 48 x 48 pixels.
- High Density Pixel Image (HDPI): This format is provided by the website, contains 72 x 72 pixels.
- Extra High Density Pixel Image (XHDPI): This format is provided by the website, contains 96 x 96 pixels
- Extra Extra High Density Pixel Image (XXHDPI): This format is provided by the website, contains 144 x 144 pixels.
- Extra Extra Extra High Density Pixel Image (XXXHDPI): This format is provided by the website, contains 192 x 192 pixels.

3.6 Launcher: Manifest file

As mentioned on the introduction of this document, there is a file named *Android Manifest* which contains the essential information that this application will require before being

launched, such as permissions, package identifiers, activities, passwords, that include the ones provided by *DJI* and *Google Maps*.

Nevertheless, on this part will be focus the Activities. These, must be mentioned on the *Manifest* file and possess the following layout as shown on Figure 17.

```
<activity
    android:name=".IntroActivityConnect"
    android:configChanges="orientation|screenSize|keyboardHidden|keyboard"
    android:label="Drone Mission Maker"
    android:screenOrientation="portrait">
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />

        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>
```

Figure 17: Launcher activity of the application.

The declaration of an Activity as *Launcher*, may seem as obvious but should only be defined on the activity being launched first, although not being this linear.

On the application, for reasons opposed by the producer and the small macros that are allowed to use, it is required to connect first before being launched. This can be solved by opening the code with the software *Android Studio*, and declare the activity that it is desired to open and examine the mobile device.

As it can be seen, *Manifest* provides us a quick overlook on all the essential parts of the code that are restricted by the development stage, other than their usual function. [58]

3.7 Click Events

The method *onClick* [59] is directed to the buttons. Can be executed directly from the main loop, in order to avoid delays, since one of the main priorities of every app should be instant feedback when a button in clicked. This method will be mentioned later on, to a better understanding of the *Switch-Case* command.

```
public void onClick(View v) {
    switch (v.getId()) {

        case R.id.btn_capture:{
            captureAction();
            break;
        }
        case R.id.btn_shoot_photo_mode:{
            switchCameraMode(SettingsDefinitions.CameraMode.SHOOT_PHOTO);
            break;
        }
        case R.id.btn_record_video_mode:{
            switchCameraMode(SettingsDefinitions.CameraMode.RECORD_VIDEO);
            break;
        }
```

Figure 18: Code related to the onClick method.

On the other hand, this method only cares about the functionalities activated when a click is made on the buttons, although these need to be defined and enable *Listener* on them, so that the action can be saved and transfer this to the method *onClick*.

Defining a button is a very simple process, that can be found in innumerous webpages for all the formats and needs. Although will be shown how to provide the *listener*, as it can be seen on Figure 19.

```
mRecordBtn.setOnClickListener(this);
```

Figure 19: Enabling the functionality Listener on a button.

The button mRecordBtn corresponds to the btnrecord identifier.

This simple method can be implemented on regular button, and also more complex ones, such as *Toggle Button*. Toggle Button is a two states button, which possess his own code outside this method, despite being a *click* event. Nevertheless, this button has been developed with relatively low effort, and is based on the definition of the *Listener*, although it possesses many similar aspects with the *onClick* method.

Once the method *onCheckedChanged* [60] is quite similar with the one mentioned above. A typical sample of an example is provided on Figure 20.

```
SwapButton.setOnCheckedChangeListener(new CompoundButton.OnCheckedChangeListener() {
    @Override
    public void onCheckedChanged(CompoundButton buttonView, boolean isChecked) {

    }
}) ;

SwapButton.setOnCheckedChangeListener(new CompoundButton.OnCheckedChangeListener() {
    @Override
    public void onCheckedChanged(CompoundButton buttonView, boolean isChecked) {
        if (isChecked) {
```

Figure 20: Toggle button.

By having a close look to the Figure 18, can be noticed that the *Else* statement is missing, on which should be included the functionalities when a button is deactivated. However, it can not be shown once the code which includes the button is far extensive to be displayed on a figure.

Finally, with all the basic concepts being mentioned above on this section, becomes easier to explain the more complex aspect when it comes to “click” events, that is the *onMapClick*. [61]

The difficulty associated with this method is not related with the load of code required, but with the different type of specific commands that are needed.

To start, it is a must to define the map as a $private$ $GoogleMap$ OurMap. This is not a compatible format file with *Android Studio*, to which should be imported some tools, like

$import$ $com.google.android.gms.maps.GoogleMaps;$ between another ones. Fortunately, *Google* tries to minimize the effort to this task by providing additional messages that advice to click “Alt + Enter” on the issued part of the code to incorporate any additional library.

In addition, it was created a delay method in order to give extra time, enabling the map to load completely, in case of requiring any additional files than the usual ones. This will provide the *listener* of the map, as it can be seen on Figure 21.

```
private void setUpMap() {
    gMap.setOnMapClickListener(this);

}
```

Figure 21: Map listener.

Finally, with the Listener mode activated, it is possible to load the method *onMapClick*, on which, by clicking on the Map button, becomes possible to define Waypoints. In cooperation with *GoogleMaps*, it is possible to convert these to geographic coordinates.

As it can be seen on the Figure 21, Altitude is defined manually, once it can not be obtained with $point.altitude$, as it is made for the latitude and longitudinal coordinates. This is due to the Waypoints being defined in 2D. Thus, in order to obtain different altitudes over the Waypoint’s, was implemented the *Setting Dialog*, defined especially for this application.

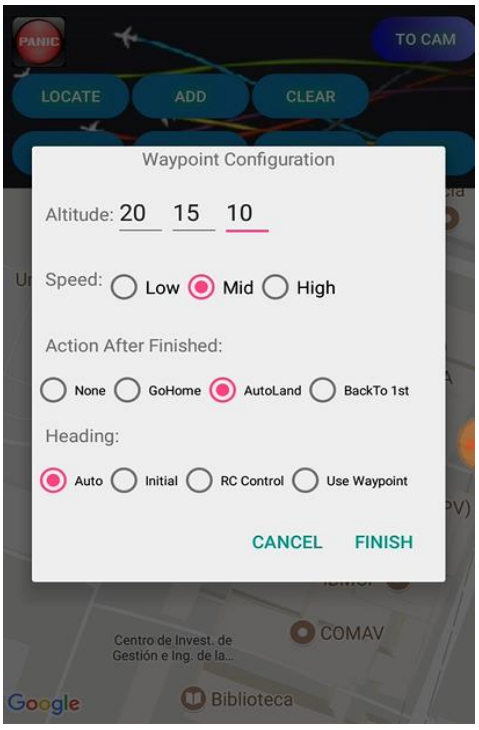

Figure 22: Menu that establishes the required the Altitude over each Waypoint created.

3.8 Command Switch-Case

This is a very well-known statement, which exists in other languages than *Java*. Being is a very simple statement, becomes useful while defining the actions attributed to each button. This will be explained based on Figure 23, in a brief way be taking on segments of the code.

```
switch (v.getId()) {
    ////
    case R.id.btn_capture:{
        captureAction();
        break;
    }
    case R.id.btn_shoot_photo_mode:{
        switchCameraMode(SettingsDefinitions.CameraMode.SHOOT_PHOTO);
        break;
    }
```

Figure 23: Structure of Switch-Case.

As it can be seen next to switch, there is the command $v.getId()$, which will select among all button's identifiers present on the Class file, the one requested. Once identified, it passes to the case part, on which for every identifier there is an action defined. As an example, there is Figure 23, which represents the Capture button, that when pressed, has assigned the action of taking a photo, next to the break statement, which assured that once this action is concluded, the onClick method is stopped in order to avoid internal issues.

3.9 Setting Dialog

This section will mention the process of incorporating the data related to the *Waypoints*. This do not possess a specific code, but a combination of known concepts that will help to better understand how this section was projected minding of the general application.

Although, the functionalities of this application were not mentioned, first should be introduced the *Waypoints* related to a defined task with the *onMapClick*, previously explained. With this, should be assigned restrictions on which Waypoint that the RPAS should complete. This is possible with *Setting Dialog*, shown on Figure 22, which includes all the variables which can be defined. These allow to define a specific altitude over different point velocity over the path, action executed when reached the final point and the pitch mode. With more detail, will be mentioned again on the section of Functionalities, thus will be given more attention to the creation of this menu.

To start, it is required to define a method to be executed when pulsing the "CONFIG" button.

```
case R.id.config:{
    showSettingDialog();
    break;
}
```

Figure 24: Action of the configuration button.

Furthermore, the code related to $showSettingDialog()$ will be initiated as it is shown on Figure 24.

```
private void showSettingDialog(){
    LinearLayout wayPointSettings = (LinearLayout)getLayoutInflater().inflate(R.layout.dialog_waypointsetting, null);
```

Figure 25: Initiation of Setting Dialog.

To start, it is used the Layout *inflate* which allow to display the menu over the principal layout of the application, on which will be introduced the data. [62]

As done before, the *Listeners* and *EditText* statements will be explained in a brief way. The *EditText* are text's that can be modified with the keyboard by the user. These statements are the parameters required from the user in order to produce actions based on them.

The following step will enable to process this data over the variables used, on which most of them are in a direct mode. An example is when it is desired to return home, the function "*Go Home*" should be called as in Figure 26.

```
mFinishedAction = WaypointMissionFinishedAction.GO_HOME;
```

Figure 26: Example of the option related to Waypoints.

These libraries provided by *DJI*, include a large quantity of methods, such as the one shown on Figure 26, like $WaypointsMissionFinishedAction$. This makes possible for developers to save time on the creation of new applications without having to implement basic procedures to exchange information with the RPAS. Although a well understanding and experience is required, once the information lack detailed explanation, being this a quite closed surrounding. This requires the user to search for different sources of knowledge in order to play this defined libraries and methods in his favor.

To conclude, all this saved and processed information by the external methods is managed by the loop $configWayPointMission()$. On this, is called the list of WayPoints previous requested on the event onMapClick, on which is established this data, be taking on waypointMissionBuilder.

In order to provide a better understanding over this, Figure 27 shows the direct functionality:

```
if (waypointMissionBuilder == null){

    waypointMissionBuilder = new WaypointMission.Builder().finishedAction(mFinishedAction)
            .headingMode(mHeadingMode)
            .autoFlightSpeed(mSpeed)
            .maxFlightSpeed(mSpeed)
            .flightPathMode(WaypointMissionFlightPathMode.NORMAL);

}else
{
    waypointMissionBuilder.finishedAction(mFinishedAction)
            .headingMode(mHeadingMode)
            .autoFlightSpeed(mSpeed)
            .maxFlightSpeed(mSpeed)
            .flightPathMode(WaypointMissionFlightPathMode.NORMAL);

}

if (waypointMissionBuilder.getWaypointList().size() > 0){

    for (int i=0; i< waypointMissionBuilder.getWaypointList().size(); i++){
        //waypointMissionBuilder.getWaypointList().get(i).altitude = altitude;
        final float [] setalt =new float[]{altitude,altitude2,altitude3,altitude4,altitude5,altitude6};
        waypointMissionBuilder.getWaypointList().get(i).altitude = setalt[i];
    }

    setResultToToast("@string/setwpok");
}
```

Figure 27: Final stage of the acquisition of data to start the flight.

It is possible to understand where the data is set by looking to the variables defined previously: $mFinishedAction$, $mHeadingMode$, $mSpeed$, $altitude$.

The Builder, will be responsible for the usage of these variables defined on the list of Waypoints at his time.

When analyzing the part of the altitude, it is quite different, once that this parameter is constant on the macro provided by *DJI*. Thus, it has been modified so it can be adjusted over different purposes for the application usage, more specifically, on this surveillance of Wind Turbine blades, height is variable. [63]

3.10 External Methods

On the application that has been developed, a diversified number of functionalities were implemented. Although a valuable aspect that should be mentioned is the complex macro provided by the developers, which was a huge help to this achieve. Thus, much of this information presents itself as a hard aspect to understand by just a quick overview. Although, some aspects will be exposed of higher interest and convenience.

Flight Controller

Through basic commands, it is possible to access the functionalities of localization of our RPAS, since it is already implemented the functionalities of localization and stabilization of flight parameters.

Waypoint Class

These are the options that allow to establish communication with the RPAS. To be more precise, *onMapClick*, is in fact, the point where a certain type of proprieties will be applied, not the point that it should pass.

Camera

With the $getInstanceCamera()$ and the respective callback, which is a method that will be executed after a certain action, between a whole diversity of internal methods, like $startShootPhoto$, will establish essential internal procedures. These are required to maintain communication and activate the camera of the RPAS, enabling to capture images and store them on the SD card, that the RPAS has the capacity to possess.

In a similar way, with other methods, is possible to execute a video with the camera by implementing the respective code, that enable to manage these functionalities.

Although being an effort to gather different macro's and resolve compatibility issues between them, *DJI* developers page provide good topics about the MacroCam. [64]

Connection

Previously has been shown a tutorial to the creation of this application, that will be launched through the *Class* ConnectionActivity.

This will block the contain of the application for a security reason, in order to assure aspects like compatible internal methods, official RPAS and version used are subjects that should be validated so it can be assured that the product is projected correctly providing regular performance while working.

This aspect should not be taken in deep consideration, once it is explained in detail on almost every tutorial of the page, such as the one mentioned above regarding the camera application. [64]

3.11 Functionalities

This application was developed regarding the inspection of aerogenerator blades, to which must include a whole range of functionalities that can be adapt to different clients.

On this alfa stage of the application, it is provided the services of the camera to take photos and to record videos, which can be processed afterwards, and also a movement system of the RPAS that can be followed on the map displayed on the mobile screen, with the manually defined altitude on each Waypoint, allowing a whole range of trajectories. Furthermore, it also includes a "Panic" button, that can be activated, blocking every previous defined action, allowing the return of the RPAS to the "home" position in case of something wrong or just a need to redefine the trajectory assigned.

3.12 Accessibility

This application was created aiming to keep the user's interest on short tutorials, providing an fundamental concepts, in order to make effortless to establish the connection between the mobile device and the RPAS enabling it to fly. Once these steps are completed, when connected correctly, the file *ConnectionActivity*, will assure it. The Map will appear on the interface displayed on the mobile device which contains some buttons on the top of the screen that allow the execution of defined actions.

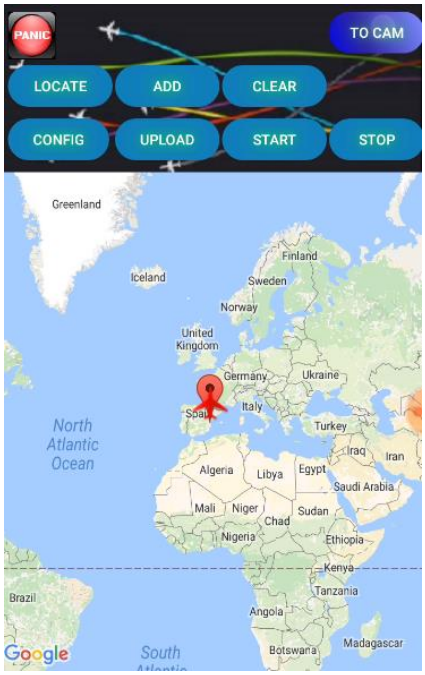

Figure 28: Interface of the Map.

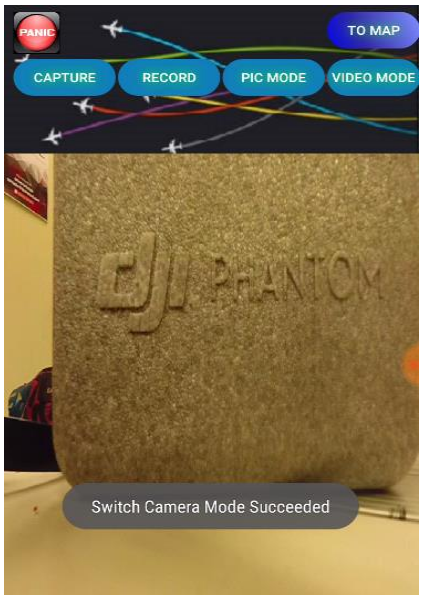

Figure 29: Interface of the camera.

It is shown a simple interface, on which the Map buttons are in different displacements. The switch between interfaces can be obtain as mentioned before with the *Toggle Button* that is shown on the top right of the screen. The "Panic" button will never turn invisible for security reasons, that will be described later on.

This interface will also return informative *Toasts*, that are short duration messages displayed near the bottom of the screen, as it can be seen on Figure 29 in a grey balloon. This provide useful information that let us know if the action has been completed successful or not.

```
private void setResultToToast(final String string){
    MainActivity.this.runOnUiThread(new Runnable() {
        @Override
        public void run() {
            Toast.makeText(MainActivity.this, string, Toast.LENGTH_SHORT).show();
        }
    });
}
```

Figure 30: Toast code example.

Furthermore, since this application has an industrial purpose, it is important to mention that it is provided three different languages, so that messages on this application can be executed in English, Spanish and Portuguese. [65] This is possible modify them or even incorporate more by accessing the strings.xml file, that can be accessed as shown in Figure 31.

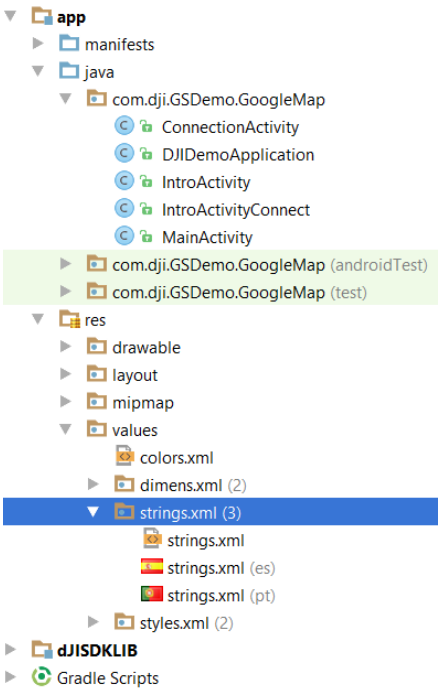

Figure 31: Location of different languages provided to this application.

Then, by opening the file strings.xml, will be clicked the option "Open editor".

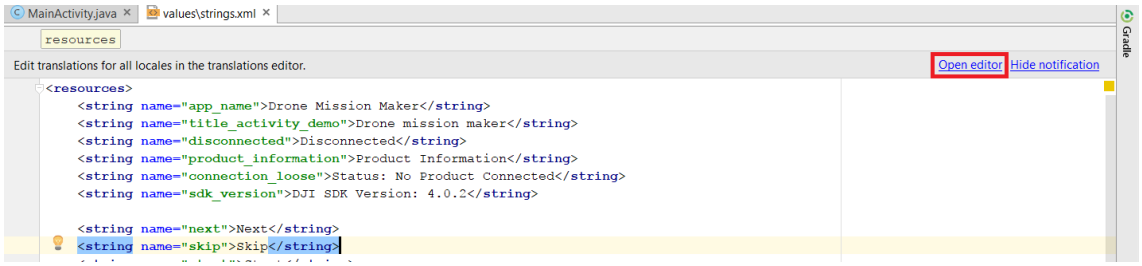

Figure 32: Option that allows to edit the languages displayed on the application.

Once inside of the editor, a language must be selected to translate all the defined texts to the language desired. It is of higher importance to locate all the text's that will be used on the application on the string.xml file, as well to call them on the code by their identification, either being Class files or layout.xml files, so that the change between languages is possible.

3.13 Camera utilities

On the beginning, for the functionalities section, was introduced the functionality of the camera, as well the explication of the code based on previous knowledge. Thus, remains to mention his behavior. As seen on the Accessibility section, the camera interface contains the buttons presented on the Figure 29.

The relevant buttons are: camera mode, video mode, recording and capture. It is required to establish on the application, which mode is being used, the camera or the video mode. Furthermore, to start it is clicked the mode that is desired to use, so when the informative messages, Toast, appears it is ready to press the button "capture" or "record", depending on the mode that is being used.

Moreover, it was introduced the functionality over the video mode, that show's bellow the "record" button, a timer so that the user can have an idea of the duration of this action. In the end, all these files will be saved on the SD card of the RPAS.

3.14 Mission Maker and GPS

On this section, will be explained how does the mission maker works, in other terms, the steps that the user should complete, once that the code has been explained already.

The explication shown below was previously included in a brief way over the tutorial displayed at beginning of the application. Furthermore, it is shown the buttons of the interface that take part on the Map, as it can be seen on Figure 28.

The procedure, is that once the RPAS is connected with the application, the use must take place where GPS signal is strong, once that all the trajectory generated with this application is based on GPS signal, if this parameter is not fulfilled, a message will be displayed by an internal library which will not allow to continue the execution. Once the GPS signal is strong, the button "Locate" should be pressed, that will establish the RPAS position over the map. This position is

manually defined by the latitude and longitudinal parameters on the *DJI Assistant 2*, available for free on the DJI website. [66]

Moreover, after having a defined location of the RPAS, to define Waypoints should be pressed the button "ADD" that will provide the *onMapClick* method, the button can be seen on Figure 33. These way, it is possible to define the Waypoints by pressing the Map. After the points over which it is desired to complete a circuit are established, the same button should be pressed, although it will have the Exit word on it while being onMapClick mode, so this action is finished successfully.

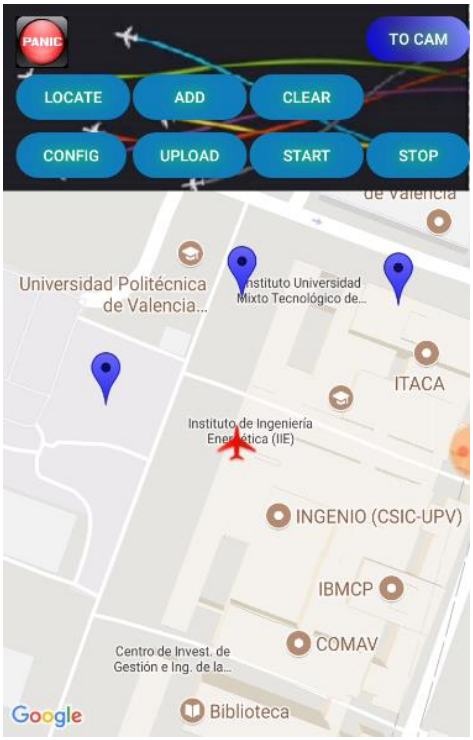

Figure 33: Assignation of the waypoints manually.

Secondly, the button "Config" should be pressed, which will request the menu to introduce data, *Setting Dialog*, that enables to defined different altitudes over each Waypoint, velocity, pitch and final action desired. This process was explained with higher detail in the section of "Previous knowledge", that for more information should be consulted above.

The last steps are to press the "Upload" button, which will load the data into a list of Waypoints, that after this, it is ready to press the "Start" button, in order to initiate the mission.

During this, the button "Stop" operates to pause the mission, and also use the camera mode while the Map mode is executed on background, without any issue.

It is also possible to see the "Clear" button, that allows to clear the mission established. This is useful once if any Waypoint was misplaced, it can be changed.

3.15 Security regarding adverse conditions

During the automated mission of the RPAS, it will execute all the parameters that the user defines. Nevertheless, errors might have been placed, and the fact of pressing the "Stop" button will not be enough to return the RPAS to a safe place, once after this another mission to return to the initial position, also called "Home" is required. Since the action "Go Home" will not have defined "Home" position as the start position of the mission, but the position where the "Stop" button was used. These small details generated by the *Macro* provided by *DJI MacroMap*, must be taken in concern when evaluation the safeness of the RPAS regarding the human error. [67] With the goal of avoiding these issues, a "Panic" button was developed, located on the top left corner of the screen, visible whether in Camera or Video mode.

Figure 34: Panic button.

Moreover, will be explained the procedure in order to bring the RPAS to the initial position.

First, the functionality over the "Locate" button was added, that in addiction of his normal function, also allows to store the Latitude and Longitude of the initial position, but only for the first time it is pressed, that it is saved as a global variable. In other words, this initial position is saved until the application is exit. With this, even by pressing the "Stop" or "Clear" buttons, necessary to define the mission of returning home, it is possible to store the initial position, no matter what.

Once this has been clarified, the mission can follow his normal course. First, it is pressed the "Stop" button, that instantly are included some Delays in order to prevent the system from collapsing and enabling itself to manage all the orders. This is required since when the data is introduced manually, it is given a certain time for the application between point to point. Nevertheless, when everything is instantly, and every data is inserted immediately, the application might collapse.

The next command applied should be "Clean" so the list of Waypoints is cleared. After this, the "Locate" button should be pressed so the application can have the position where the drone is currently. Next, it can be executed a new mission, that contains as initial point and final point, the current location, the geographic coordinates saved previously as global variable.

Once that every data was erased, it is advised to defined "AutoLand" as the final action when reach to the initial position, that commonly the user might want to land if anything went wrong during the trajectory. Moreover, should be defined "Low Velocity" to return and a automatic

pitch mode so that no actions from the user are required. All this trajectory will be executed at a 30 meters altitude, in order to avoid possible obstacles while returning.

To summarize, this “Panic” button provided an additional security to the user that in case of fail, preventing the loss of the RPAS’ control, enabling to retrieve him safely. On the Results section, will be shown the functionality of this button.

Chapter 4: MATLAB Simulation Model

On this chapter, will be presented the methods used to obtain the algorithm so it would be possible to generate a trajectory in MATLAB. The model used had incorporated the parameters of Phantom 4, such as the mass and arm length. Moreover, will be presented how it is planned to adapt this code to the Application produced in Java.

4.1 Dynamic Model

The robot's movements can be defined by six degrees of freedom. In order to characterize them, the Inertial frame (*a*) and Body frame (*i*) frame were created. The Inertial frame is composed by a_1, a_2 and a_3, being the positive part of a_3 pointing upward. In addiction, was defined the body frame (*i*), which has the origin latch onto the Quadrotors centre of mass, as it is possible to see on the Figure 35.

Figure 35: Referential systems.

To characterize the system, will be used Z – X – Y Euler angles, modelling the rotation of the Quadrotor.

In addition, to characterize the vehicles attitude, it is defined a rotation matrix, ${}^{a}R_{i}$, presented on the equation (1). It enables to describe a vector from the Inertial System, *a*, in the vehicles body system, *i*, multiplying the vector from the inertial by the rotation matrix, or vice versa.

$$ {}^{a}R_{i} = \begin{bmatrix} c\psi c\theta - s\phi s\psi s\theta & -c\phi s\psi & c\psi s\theta + c\theta s\phi s\psi \\ c\theta s\psi + c\psi s\phi s\theta & c\phi c\psi & s\psi s\theta - c\psi c\theta s\phi \\ -c\phi s\theta & s\phi & c\phi c\theta \end{bmatrix} \quad (1) $$

The sθ and cθ, represent sin(θ) and cos(θ), respectively. This also applies for ϕ and ψ *angles*.

Defining *r*, as the position vector with respect to the center of mass (*i*) from the inertial frame (*a*).

The primary forces on the system are the ones produced by gravity, on the – i_3 direction, and the ones produced by the motors F_i, on the i_3 direction. To summarize this, is provided equation (2) that represents the acceleration with respect to the center of mass.

$$m\ddot{r} = \begin{bmatrix} 0 \\ 0 \\ -mg \end{bmatrix} + R \begin{bmatrix} 0 \\ 0 \\ F_1 + F_2 + F_3 + F_4 \end{bmatrix} \quad (2)$$

The angular velocities are defined as *p*, *q* and *r*, on equation (3). These are related with the derivatives of roll, pitch and yaw angles.

$$\begin{bmatrix} p \\ q \\ r \end{bmatrix} = \begin{bmatrix} c\theta & 0 & -c\phi s\theta \\ 0 & 1 & s\phi \\ s\theta & 0 & c\phi c\theta \end{bmatrix} \begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} \quad (3)$$

4.2 Control Inputs

Over the referential system established, the Quadrotors state is defined based on the longitudinal, transversal and vertical positions, first tree elements on equation (4). Nevertheless, his orientation over the tree axis should also be considered, last tree elements of equation (4). With this, is defined the quadrotors state. Regarding equation (5), is represented the state of the rigid body, which is composed by his six-dimension vector (q) and its rate of change ($\dot{q}$).

$$q' = [x \quad y \quad z \quad \phi \quad \theta \quad \psi]^T \quad (4)$$

$$x = [q' \quad \dot{q}']^T \quad (5)$$

On this model, a series of simplifications were made. It is assumed that the thrust produced by the rotors, can be swapped immediately. This is not completely accurate, once that a dynamical system is characterized for being a system on which the effects of actions do not occur immediately. It is also limited in terms of the vehicle's body rate, by the gyroscopes and tracking controllers. In addition, the force produced by each motor, a_i was limited, being the maximum value produced by all the rotors, 1,5 times of the Quadrotors weight.

Being *i* = 1,2,3,4.

$$a_{min} \leq a_i \leq a_{max} \quad (6)$$

On which *a* represents linear acceleration. The value of a_{min} is positive or zero, since these Quadrotors do not possess the ability to stop or reverse the direction of rotation of the propellers. [68] The maximum thrust is limited by peak motor torque. The total of vertical forces can be obtained by the sum of the forces produced by the motors plus the force of the gravity applied on the vehicle.

$$\vec{F}_{Total} = \vec{F}_{motor} + \vec{F}_{gravity} \qquad (7)$$

Discretizing equation (7), it is possible to obtain equation (8), on which *m* represents the mass of the vehicle, *g* is the force of gravity, ω the angular speed of the motors and k_F the proportionality constant of the force produced by the motors. [68]

$$ma = \sum_{i=1}^{4} k_F \omega_i^2 + mg \qquad (8)$$

4.3 Equations of motion

Related to the equations of motion, the ones regarding to the motors, creates moments over the three-referential axis established. Since motors 1 and 3 spin on the – i_3 direction they will produce a positive moment. Motors 2 and 4 spin on the i_3 direction, producing negative moments. The variable *L* represents the distance between the vehicle' center of mass and the position of the motor. With this defined, the inertia matrix related to the moments produced by the components of the vehicle can be written as: [68]

$$\mathrm{I}\begin{bmatrix}\dot{p}\\ \dot{q}\\ \dot{r}\end{bmatrix} = \begin{bmatrix}L(F_2 - F_4)\\ L(F_3 - F_1)\\ M_1 - M_2 + M_3 - M_4\end{bmatrix} - \begin{bmatrix}p\\ q\\ r\end{bmatrix} \times I \begin{bmatrix}p\\ q\\ r\end{bmatrix} \qquad (9)$$

4.4 Motor model

The motors were modeled by a test of steady-state fixed rotor in order to determine a constant of proportionality that multiplied by the angular speed would allow to estimate the force produced:

$$F_i = k_F {\omega_i}^2 \qquad (10)$$

Experimentation with a fixed rotor at steady-state show $k_F \approx 6.11 \times 10^{-3}\, N/rpm$. [69]
With the same ideology, it is possible to determine a constant of proportionality, to estimate the moment produced:

$$M_i = k_M {\omega_i}^2 \qquad (11)$$

The constant, was determined to be about $k_M \approx 1.5 \times 10^{-9}\, N/rpm$. [69]

The components of angular velocity, relative to the body frame, are defined as:

$${}^{a}\omega_i = p i_1 + q i_2 + r i_3 \qquad (12)$$

4.5 Control Model

The rigid body dynamics have slower response compared to the motor dynamics. Incorporating them leads to a fifth order dynamic model that implies additional complexity without significant improvement in performance. [68] Using Euler angles, it is possible to parameterize the orientation, position and velocity of the center of mass.

$$x = [x \quad y \quad z \quad \phi \quad \theta \quad \psi \quad \dot{x} \quad \dot{y} \quad \dot{z} \quad p \quad q \quad r]^T \qquad (13)$$

Neglecting the parameterization by the position and velocity of the center of mass, rotation matrix and angular velocity, can be simplified to: [68]

$$u = [u_1 \quad u_2 \quad u_3 \quad u_4]^T \qquad (14)$$

Where u_1 is the force from all the propellers and u_2, u_3, and u_4 are the moments related to the body frame axis. [68]

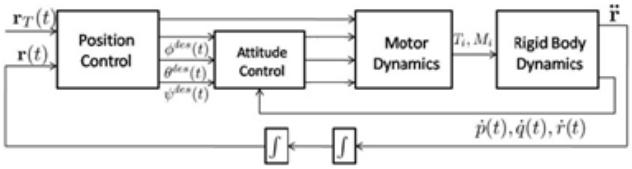

Figure 36: Nested control loop for the position and attitude control. (Obtained from [69])

This nested loop, has an inner loop that corresponds to the attitude control and an outer loop that corresponds to the position control. On the inner loop is specified the orientation either with the rotation matrix or a series of roll, pitch and yaw angles. From the angular rates, will be calculated u_2. This is a function of the thrust and the moments that provide from the motor speeds. It will be calculated based on the desired attitude. On the outer loop, its aim is to obtain the position vector. This, will be compared with the actual position and actual velocity that enables us to obtain u_1. [68]

4.6 Attitude Control

Since this is an initial approach, the waypoints are defined so the Quadrotor completes an itinerary that contains all the blades in order complete a visual survey with photos on them. To obtain a quality inspection, it is needed a high-quality photo, which requires a stable flight, no aggressive maneuvers, and mostly considered on near hover state. Thus, it is presented a control based on small angles. This controller is based on a linearization of the equations of motion and motor models, at an operating point that corresponds to the nominal hover state: [68]

$$\theta \approx \phi \approx 0 \qquad (15)$$

$$\psi \approx \psi_0 \qquad (16)$$

Considering small angles of pitch and roll:

$$c\phi \approx 1,\ c\theta \approx 1,\ s\phi \approx 0\ ,\ s\theta \approx 0 \qquad (17)$$

So, it is possible to deduce the nominal values at hover state for the first two inputs defined, the thrust of the motors, u1, and the Quadrotor attitude, u_2:

$$u_{1,0} = mg \qquad (18)$$

$$u_{2,0} = 0 \qquad (19)$$

By this, it is possible to assume that each rotor produces a quarter of the total thrust needed to maintain hover, so:

$$F_{i,0} = \frac{mg}{4} \qquad (20)$$

And so, it is possible to deduce that the nominal angular velocity produced by the motors are: [68]

$$\omega_{i,0} = \omega_h = \sqrt{\frac{mg}{4k_F}} \qquad (21)$$

Thus, it is possible to determine the desired rotor speeds using the desired ($u_{1,des}$) and the moments ($u_{2,des}$, $u_{3,des}$ and $u_{4,des}$):

$$u_{des} = \begin{bmatrix} k_F & k_F & k_F & k_F \\ 0 & k_F L & 0 & -k_F L \\ -k_F L & 0 & k_F L & 0 \\ k_M & -k_M & k_M & -k_M \end{bmatrix} \begin{bmatrix} {\omega^2}_{1,des} \\ {\omega^2}_{2,des} \\ {\omega^2}_{3,des} \\ {\omega^2}_{4,des} \end{bmatrix} \qquad (22)$$

4.7 Trajectory Tracking

For the position control, will be presented two position methods, based on roll and pitch angles as inputs. Both methods are based on the linearization of the equations of motion, as shown above. The first method is for the nominal state, hover, used for maintaining the position. The second is designed to track and follow a desired trajectory, *r(t)*, over the three directions. On these two methods, the position control algorithm will determine the desired roll and pitch angles, ϕ_{des} and θ_{des}, that can be used to compute the commanded

accelerations. [68] This yaw angles, can be constant, ψ_0 , or variable with time, $\psi_c(t)$. It is a similar method to a backstepping approach. [70]

On this case, the position and orientation are fixed, which implies that all velocities and acceleration are zero, as well the roll and pitch angles. The input u_2, must be close to zero, as well the rates *p*, *q* and *r*. The yaw angles can be non-zero if it is fixed, $\psi_c(t) = \psi_0$. [69] It is assumed that u_1 is very close to the weight.

With these assumptions, is now possible to define expressions to determine the accelerations over the three axes from Equation (2):

$$\ddot{r}_1 = g(\Delta\theta cos\psi_0 + \Delta\phi sin\psi_0) \quad (23)$$

$$\ddot{r}_2 = g(\Delta\theta cos\psi_0 + \Delta\phi sin\psi_0) \quad (24)$$

$$\ddot{r}_3 = \frac{1}{m}u_1 - g \quad (25)$$

Related to the products of inertia, once the axes are close to the principal ones, it is possible to assume that:

$$I_{xx} \approx I_{yy} \quad (26)$$

Since the Quadrotor is symmetric, it is possible to write the rates of change based on the linearization of the equations (2) and (3): [68]

$$\dot{p} = \frac{u_2}{I_{xx}} = \frac{L}{I_{xx}}(F_2 - F_4) \quad (27)$$

$$\dot{q} = \frac{u_3}{I_{yy}} = \frac{L}{I_{xx}}(F_3 - F_1) \quad (28)$$

$$\dot{r} = \frac{u_4}{I_{zz}} = \frac{\gamma}{I_{zz}}(F_1 - F_2 + F_3 - F_4) \quad (29)$$

The error related to the position, is defined as the different between the desired state and the actual state:

$$e_i = (r_{i,des} - r_i) \quad (30)$$

Furthermore, will be incorporated a PD controller. This contains a proportional gain that produces oscillations when the system is far from the desired state plus the derivative gain, which predicts system behaviour and helps to stabilize it. Thus, can be written the following equation, in order that the error goes exponentially to zero:

$$(\ddot{r}_{i,des} - \ddot{r}_{i,c}) + k_{d,i}(\dot{r}_{i,des} - \dot{r}_i) + k_{p,i}(r_{i,des} - r_i) = 0 \quad (31)$$

4.8 Hover stage

On this state, it is desired for the Quadrotor to maintain a defined position, which means that the acceleration and velocity are zero:

$$\ddot{r}_{i,c} = \dot{r}_{i,c} = 0 \qquad (32)$$

Moreover, it must be established that this condition is guaranteed. Thus, applying equation (31), lead to: [68]

$$\ddot{r}_{i,des} + k_{d,i}\dot{r}_{i,des} + k_{p,i}(r_i - r_{i,des}) = 0 \qquad (33)$$

Based on equation (23-25), and reminding the fact that pitch and roll angles must be maintained at a constant value, in order the achieve this state, the equation can be simplified to:

$$\ddot{r}_{1,des} = g(\theta_{des}cos\psi_T + \phi_{des}sin\psi_T) \qquad (34)$$

$$\ddot{r}_{2,des} = g(\theta_{des}sin\psi_T + \phi_{des}cos\psi_T) \qquad (35)$$

$$\ddot{r}_{3,des} = \frac{1}{m}u_1 - g \qquad (36)$$

Moreover, with a PD controller, as written in the equation (31), the error shall go exponentially to zero and also satisfy a second order differential equation (33). Related to this equation, there is a set of terms related to the specified trajectory, $\ddot{r}_{i,des}$, $\dot{r}_{i,des}$ and $r_{i,des}$, and a set of terms related with the actual trajectory, $\dot{r}_i$ and r_i. Thus, it is from our interest to calculate the commanded sum of the thrusts from the motors, that will define the force that is desired to produce on the Quadrotor, in order to maintain his position:

$$u_1 = mg + m\ddot{r}_{3,des} = mg - m(k_{d,3}\dot{r}_3 + k_{p,3}(r_{3,des} - r_3) \qquad (37)$$

The commanded roll and pitch angles can be obtained once known the desired acceleration in the *x* and *y* directions, as shown below:

$$\phi_c = \frac{1}{g}(\ddot{r}_{1,des}sin\psi_{des} - \ddot{r}_{2,des}cos\psi_{des}) \qquad (38)$$

$$\theta_c = \frac{1}{g}(\ddot{r}_{1,des}cos\psi_{des} + \ddot{r}_{2,des}sin\psi_{des}) \qquad (39)$$

The desired roll and pitch angles are defined as shown on the equation (40) and (41):

$$\phi_{des} = \frac{1}{g}(\ddot{r}_{1,c} sin\psi_{des} - \ddot{r}_{2,c} cos\psi_{des}) \quad (40)$$

$$\theta_{des} = \frac{1}{g}(\ddot{r}_{1,c} cos\psi_{des} + \ddot{r}_{2,c} sin\psi_{des}) \quad (41)$$

Furthermore, the pitch and roll velocities are defined as zero since these angles are constant through all the hover state:

$$p_{des} = 0 \quad (42)$$

$$q_{des} = 0 \quad (43)$$

$$r_{des} = \dot{\psi}_T(t) \quad (44)$$

For the feedback loop, it is assumed that the commanded roll, pitch and yaw angles are known, as well their derivatives. Thus, all it is required for attitude feedback control are the actual values of roll, pitch and yaw angles plus their derivatives to determine u_2:

$$u_2 = \begin{bmatrix} k_{p,\phi}(\phi_{des} - \phi) + k_{d,\phi}(p_{des} - p) \\ k_{p,\theta}(\theta_{des} - \theta) + k_{d,\theta}(q_{des} - q) \\ k_{p,\psi}(\psi_{des} - \psi) + k_{d,\psi}(r_{des} - r) \end{bmatrix} \quad (45)$$

This model was projected to have a faster feedback by the attitude controller, than the position controller loop, as it can be seen in the Figure 36.

4.9 3D Trajectory

Since nominal state has already been defined, now it is going to be defined a controller that will enable to follow a three-dimensional trajectory, with the acceleration being close to near-hover state. This model follows the same assumptions made above, with the difference that $\ddot{r}_{i,c}$ and $\dot{r}_i$ are no longer zero.

It is considered near-hover state, assumptions of linear equations hold. To generate the desired acceleration, it is used equation (31), so the error goes exponentially to zero.

From the equation (23), it is possible to determine the commanded state over the longitudinal and transversal directions, respectively:

$$\ddot{r}_{1,c} = \ddot{r}_{1,des} - k_{d,1}(\dot{r}_{1,des} - \dot{r}_1) - k_{p,1}(r_{1,des} - r_1) \quad (46)$$

$$\ddot{r}_{2,c} = \ddot{r}_{2,des} - k_{d,2}(\dot{r}_{2,des} - \dot{r}_2) - k_{p,2}(r_{2,des} - r_2) \quad (47)$$

Yet, since this model has some errors and limitations relating to the input thrusts, commanded trajectory may present some oscillations when sharp turns are encountered. With this restriction, comes the need to provide a modification to better describe the trajectories. Defining $\hat{t}$ as the trajectory's unit tangent vector and $\hat{n}$ as the normal unit vector, the binormal vector, $\hat{b}$, is obtained by cross product of the trajectory's unit tangent vector and the normal unit vector, $\hat{b} = \hat{t} \times \hat{n}$. With this, is possible to define position and velocity errors: [69]

$$e_p = \left(\left(r_{i,des} - r_i\right) \cdot \hat{n}\right)\hat{n} + \left(\left(r_{i,des} - r_i\right) \cdot \hat{b}\right)\hat{b} \qquad (48)$$

$$e_v = \dot{r}_{i,des} - \dot{r}_i \qquad (49)$$

Only the position error, over the normal plane to the curve's closest point is considered. The position error in tangential direction is ignored. With the expression defined in equation (31), it is possible to calculate the commanded acceleration, $\ddot{r}_{i,des}$:

$$\left(\ddot{r}_{i,des} - \ddot{r}_i\right) + k_d e_v + k_p e_p = 0 \qquad (50)$$

All the other assumptions made for hover state hold, plus with the exceptions clarified on this section, it becomes possible to generate a trajectory on 3D. [68]

4.10 Trajectory generation

To the generation of trajectory, will be presented some equations depending on the precision required, their order will increase. Below is presented an equation for minimum snap trajectory, that requires a 7th order polynomials. Each polynomial piece travels between two waypoints, taking a stipulated amount of time, T_i. [68]

Being p_i the polynomial between positions, ω_i and ω_{i+1}, that take,T_i, time to complete. The total amount of time is represented by:

$$S_i = \sum_{k=0}^{i-1} T_k \qquad (51)$$

The polynomial that represents the restrictions through the waypoints is: [68]

$$p_i(t) = \alpha_{i0} + \alpha_{i1}\frac{t - S_i}{T_i} + \alpha_{i2}\left(\frac{t - S_i}{T_i}\right)^2 + \cdots + \alpha_{i7}\left(\frac{t - S_i}{T_i}\right)^7 \qquad (52)$$

To complete this equation, it is required to determine all the coefficients α_{ij}. For this end, it is needed to establish some constraints. First, it must go through all the waypoints: [68]

$$p_0(0) = \omega_0 = \alpha_{00} \tag{53}$$

$$p_i(S_i) = \omega_{i+1} \text{ for all } i = 0 \dots n \tag{54}$$

Second, must be defined that the Quadrotor will start and stop at rest: [68]

$$p_n^{(k)}(0) = 0 \text{ for all } 0 \leq k \ \leq 3 \tag{55}$$

$$p_n^{(k)}(S_n) = 0 \text{ for all } 0 \leq k \ \leq 3 \tag{56}$$

It must also be defined that the four derivatives are continuous between the waypoints: [68]

$$p_{i-1}^{(k)}(S_i) = p_i^{(k)}(S_i) \text{ for all } i = 1 \dots n \text{ and } k = 1 \dots 7 \tag{57}$$

With this, is possible to convert the constraints and unknown coefficients, α_{ij}, into an equation, being *A* matrix build based by the restrictions and *b*, matrix composed by the waypoints: [68]

$$A\alpha = b \tag{58}$$

Where α contain all the unknown coefficients, and A and b represent all constraints.

4.11 Gain tuning: Ziegler-Nichols method

Since there are twelve gains to tune, half for the position, the other half for the attitude. A trial and error method approach becomes inefficient. Thus, urges the need to define a more systematic approach to gain tuning, the Ziegler-Nichols method. [68] [71]

The Ziegler-Nichols method is used to tune Proportional, Proportional Integrative, Proportional Derivative and Proportional Integrative and Derivative controllers. On this case, it will be applied for a Proportional Derivative control.

This method has a set of steps that should be followed:

1 Set the proportional and derivative gains to zero.

2 Increase the proportional gain until the system outputs sustained oscillations that will be denominated as Ultimate Gain, K_u.

3 Measure the period, T_u, described by the oscillations.

4 Define the proportional gain as $K_p = 0{,}8 \times K_u$

5 Define the derivative gain as $K_v = T_u/8$

Sometimes, even after this procedure, the controller response is too slow. To increase it, it is advised to scale the proportional and derivative gains by small increments until it is obtained a good response. [68] [72]

4.12 MATLAB to Java

After developing this algorithm, a thought urged in order to adapt it for the Java application, provide a more complete inspection.

Among some solutions founded, here will be presented the most viable regarding to make the files compatible with Java language.

MATLAB have incorporated many features, on which provides small applications inside of the software, very useful for the common needs, in order to minimize the effort of user. Among them, there is Compiler SDK. Sharing the advantages that this software provides to the users in areas such as Control Design, Image processing and others, incorporating Application Deployment, becomes possible to share ideas outside MATLAB, with the advantage of being able to pack it into another language, without the need to recode the algorithm.

The MATLAB Compiler SDK [73] can be found by accessing APPS > Library Compiler > Java Package.

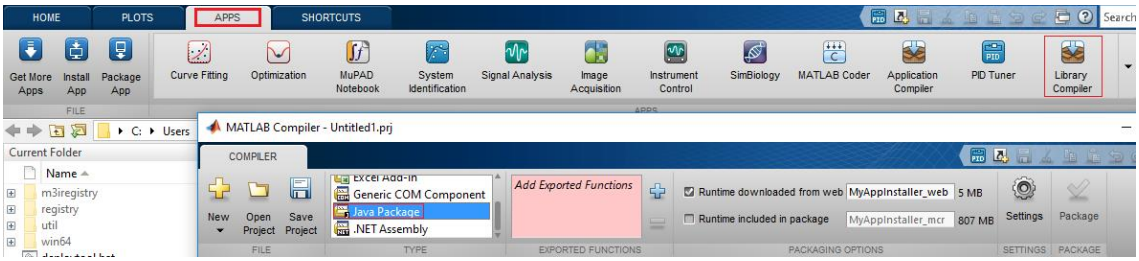

Figure 37: Localization of Java Package, on the MATLAB environment.

It is also important to say, changes on this code after converted to Java are not possible to execute, once it is only available for use.

Another solution to convert these files into compatible ones with Java language, there is MATLABcontrol [74], that is a Java API. [75] This software allows to interact with a MATLAB code by calling it with Java.

Chapter 5: Simulation Results

On this section, will be displayed the results obtained by the methods described earlier regarding the simulation.

First will be shown the part related to the Android Application, taking advantage of the DJI equipment, in order to validate all the methods established. This simulation will have the base of three points established. On these waypoints, the RPAS will generate a linear trajectory between them. Nearby the end, will be triggered the Panic button, proving its functionality defined earlier. Moreover, will be displayer the MATLAB simulation, on which was defined a PD controller to stabilize the flight, providing a good trajectory over the Wind turbine blades.

5.1 DJIl Application

Due to the flight restrictions established in Spain [76] [77], to fly an RPAS became very restrictive. Nevertheless, DJI provides a realistic simulator that allows people to learn how to manage this vehicle, and test applications, without the risk of damaging the equipment. Thus, this became the most viable solution to produce an autonomous trajectory based on the parameters requested on the application. The downside of this simulator is that is only available when connected with the RPAS.

Among the 8 possible waypoints, there were defined 3, as it can be seen on Figure 33. Based on the waypoints defined, some parameters shall be established, in order to produce an adequate trajectory for the user needs. These parameters can be seen on Figure 22.

After these parameters are established, a trajectory will be generated. The main parameters to describe the state of the RPAS are shown in the bottom left of the DJI Simulator, as it can be seen on Figure 38. A blue line was incorporated on the simulation, in order to have a clear view over the direction that the vehicle is heading, once the environment provided by the simulator does not possess particular reference points to orientate the viewers.

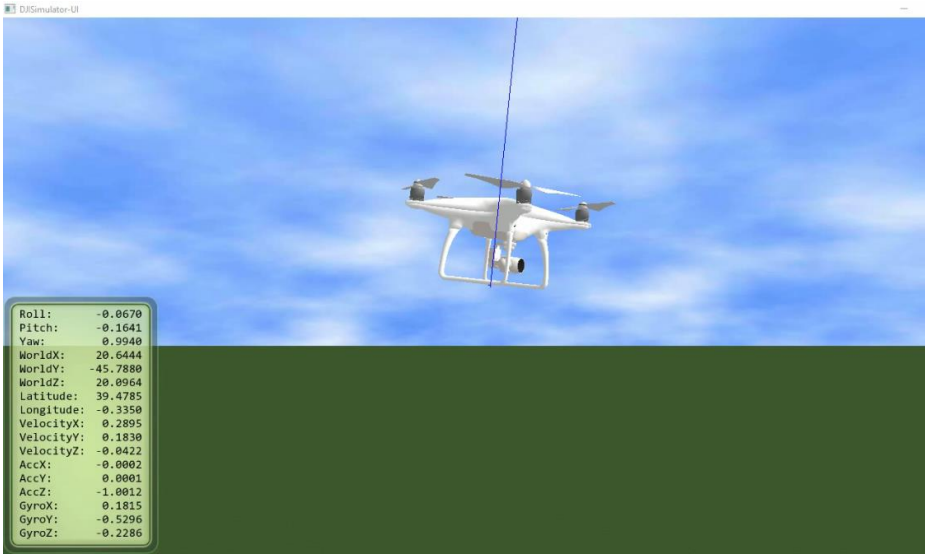

Figure 38: Simulation with the DJI Assistant 2 based on the parameters of the Phantom 4.

Moreover, the Panic button, shown on Figure 34 is activated successfully, which allows the user to return to the "Home" position, where stands the user with the remote control. This can be noticed because the RPAS ascends to 30 meters, as it can be seen on Figure 39. This was defined as a safe height that would allow the vehicle to avoid obstacles on the generated trajectory to return home. This safety height can be modified on the code.

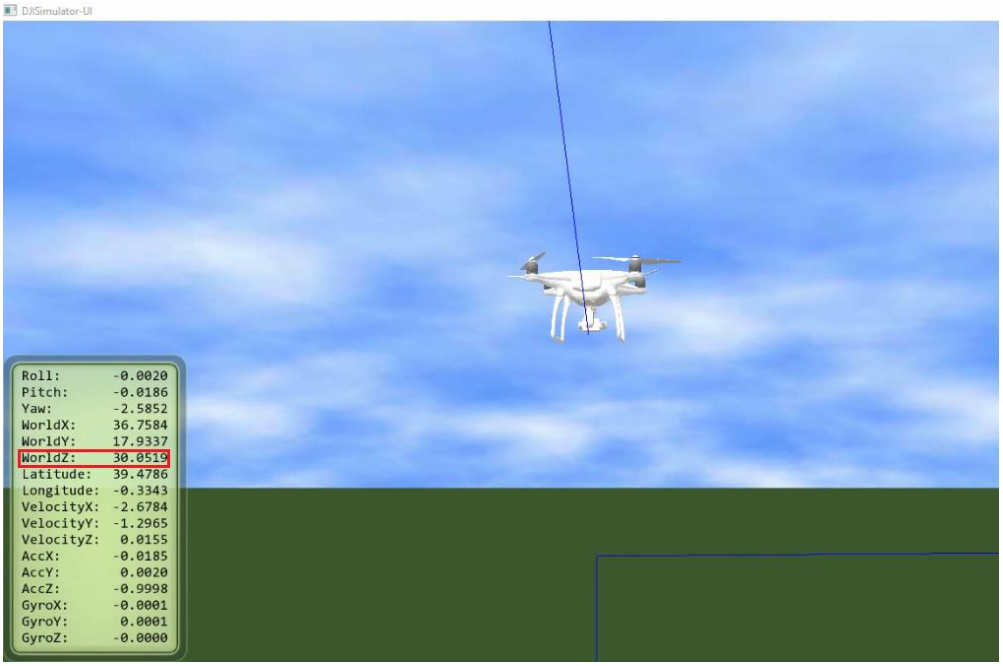

Figure 39: Service ceiling of Phantom 4 established for the "Panic" Mission.

5.2 MATLAB Simulations

5.2.1 MATLAB Simulation: Direct inspection

On this first simulation, will be presented the results obtained from the simulation idealized to be described on each waypoint established on the Application presented on Section 5.1.

The points defined to represent the Wind Generator blades, were based on the model GE 1.5sle, presented on the Introduction section. Moreover, this path was established as the waypoint defined would contain the blades from the wind turbine, from root to the tip. Thus, the waypoints defined are presented below, which were calculated based on trigonometric rules. The values shown on the matrix are in meters. The velocity of this trajectory is 0,75 m/s.

$$Waypoints = \begin{bmatrix} 0 & 0 & 0 & -33{,}34 & 0 & 33{,}34 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 80 & 118{,}5 & 80 & 60{,}75 & 80 & 60{,}75 & 80 \end{bmatrix}$$

On Figure 40 is possible to see the actual trajectory, represented by the red colour, and the desired trajectory, represented by the blue colour. This contains the trajectory described by the RPAS. Since there is not a clear distinction between the desired and actual trajectory, plus it is able to provide a clear overview of the whole structure, it is possible to conclude that these are good results.

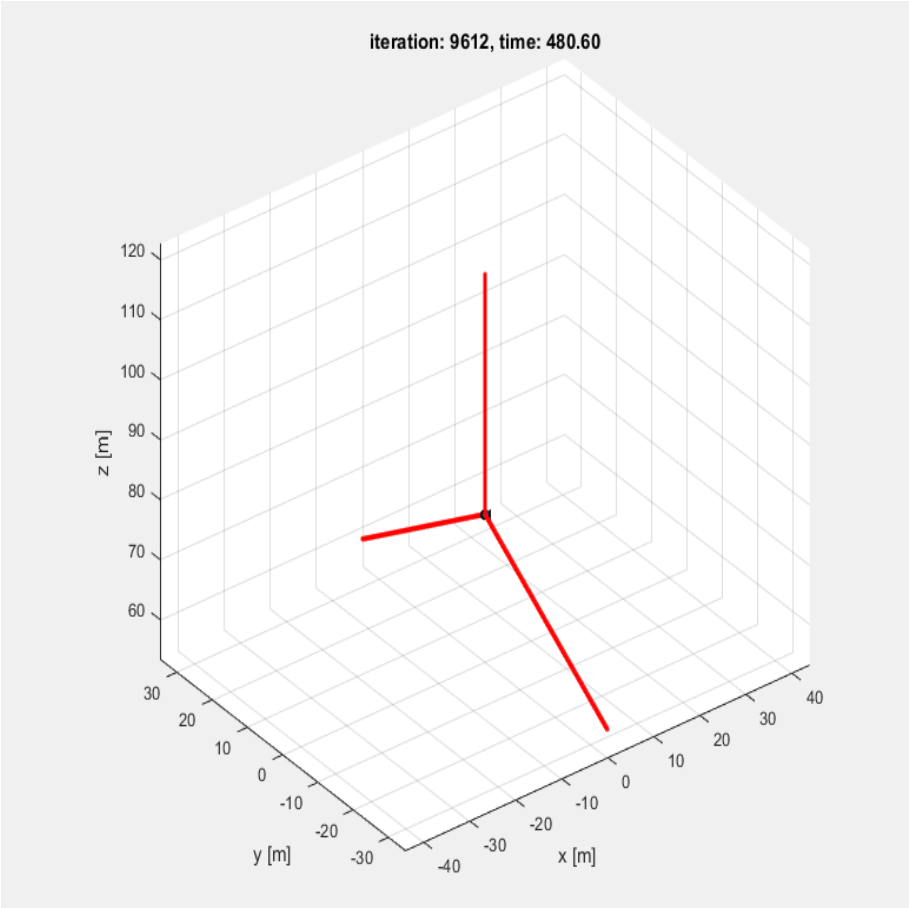

Figure 40: 3D trajectory over the Wind Turbine blades.

On the Figure 41, the colour red represents the actual described values and the blue colour, represents the desired ones. The first row represents the trajectory over the *x* axis, the second row represents the trajectory over the *y* axis and the third one represents the trajectory over the *z* axis. Since the type of polynomial equation used to describe this trajectory was of first order, each change of coordinates is represented as a corner on the graph. A small oscillation can be seen over the longitudinal axis, x. Although, being on the 10^{-6} order, this can be despised. Over the other axis, an accurate trajectory is described, over the one desired.

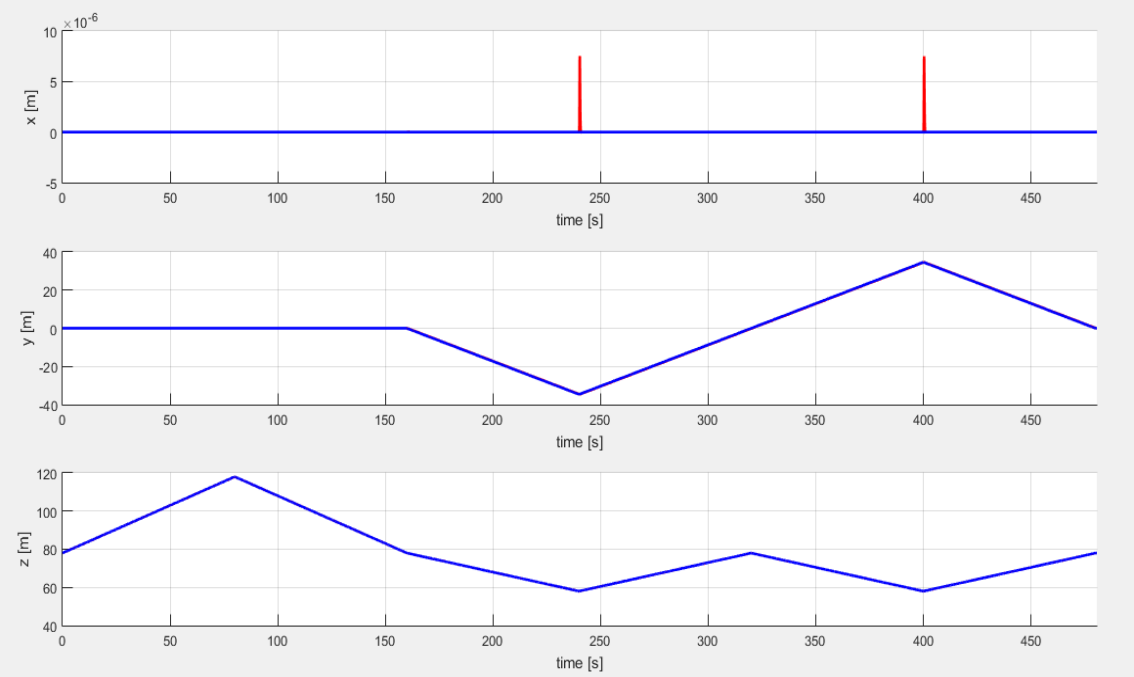

Figure 41: Oscillations between the desired and actual position described.

On Figure 42, the red colour represent the actual values described and the blue colour the desired values. On the velocities, it is possible to see that some values are with higher errors compared with the ones presented on the Figure 41. The most notorious oscillations are presented on the third column, which represents the vertical axis, since it is the one which has more variations associated. Although, it is considered that these are good values to be described by the RPAS.

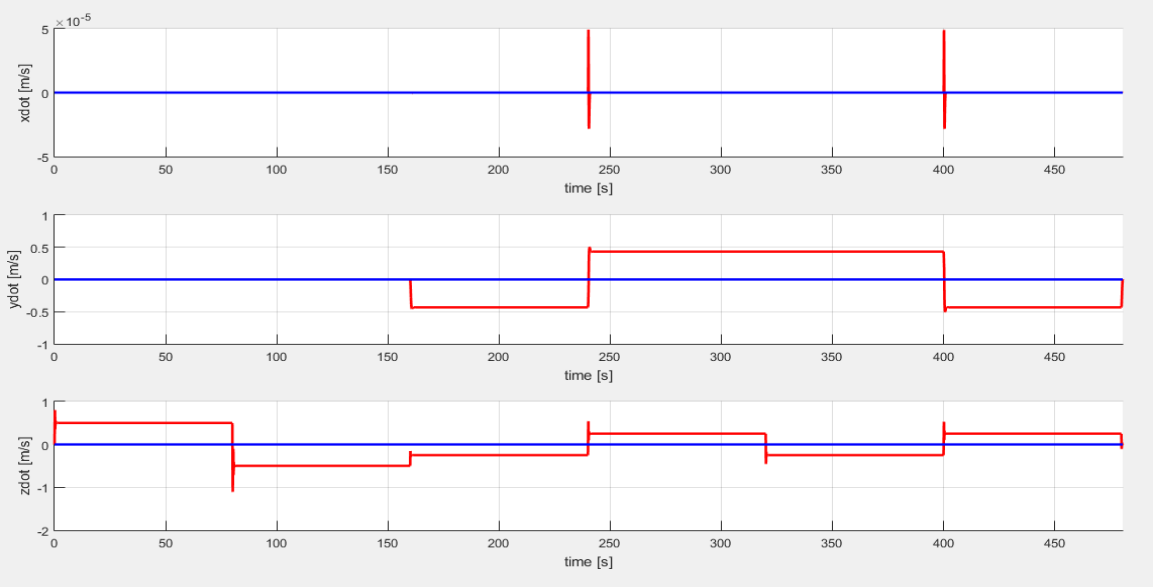

Figure 42: Oscillations between the desired and actual velocity described.

5.2.2 MATLAB Simulation: Random starting point

On this second simulation, will be presented the results obtained from the one idealized to be described from a random waypoint established, which would not require defining the height which the waypoint require the RPAS go through, as shown on the Application. This was also developed based on the idea that the waypoint was separated from the wind turbine, that in this case was considered the distance of 20m.

The points defined represent the Wind Generator blades, were based on the model GE 1.5sle, presented on the Introduction section. Moreover, this idealization was established as the waypoint defined would contain the blades of the wind generator, from the root to the tip. Thus, the waypoints defined are presented below, which were calculated based on trigonometric rules. The velocity of this trajectory is 0,75 m/s.

$$Waypoints = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & -33{,}34 & 0 & 33{,}34 & 0 & 0 & 0 \\ 0 & 0 & 20 & 20 & 20 & 20 & 20 & 20 & 20 & 0 & 0 \\ 0 & 30 & 80 & 118{,}5 & 80 & 60{,}75 & 80 & 60{,}75 & 80 & 30 & 0 \end{bmatrix}$$

On Figure 43 it is possible to see the desired trajectory, represented by the blue color, and the actual trajectory, represented by the red color. This contains the trajectory described by the quadcopter. Since there is not a clear distinction between the desired and actual trajectory, plus it is able to provide a clear overview of the whole structure, can be concluded that these are accurate results facing the desired inspection.

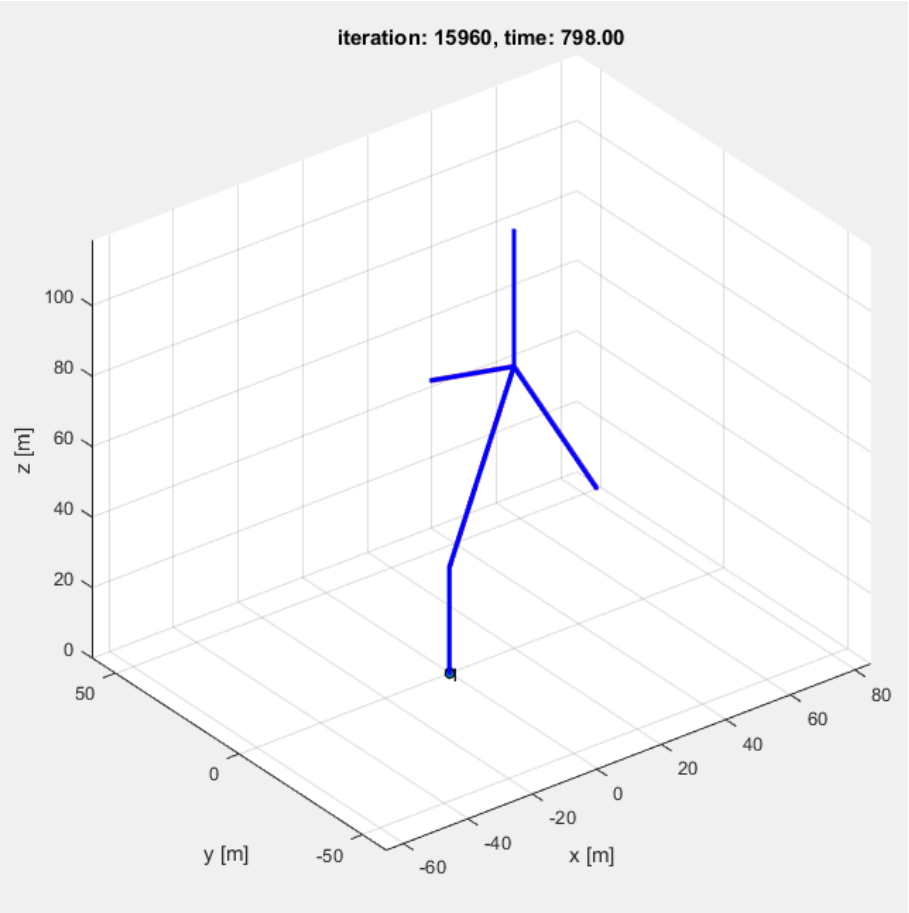

Figure 43: 3D trajectory from a random point, over the wind blades.

On Figure 44, it is represented by the red color the actual trajectory and in blue the desired trajectory. The first row represents the trajectory over the *x* axis, the second row represents the trajectory over the *y* axis and the third one represents the trajectory over the *z* axis. Since the type of polynomial equation used to describe this trajectory was of first order, each change of coordinates is represented as a corner on the graph. A small oscillation can be seen over the longitudinal axis, x. Although, the deviation is despised. Over the other axis, an accurate trajectory is described, over the one desired.

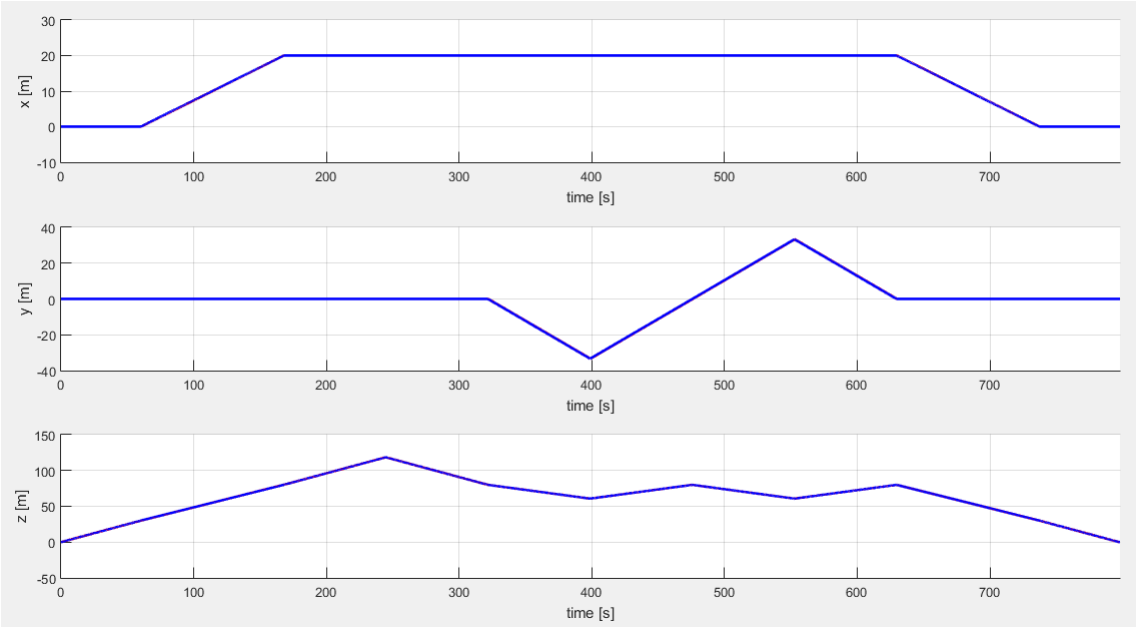

Figure 44: Desired and actual values related to the position of the trajectory.

On Figure 45 can be seen the red and blue color, which represent the actual and desired values, respectively. On the velocities, it is possible to see that some values are with higher errors compared with the ones presented on the position described, Figure 44. The more notorious oscillations are presented on the third column, which represents the vertical axis, since it is the one which has more variations associated. Although, it is considered that these are good values to be described by the RPAS.

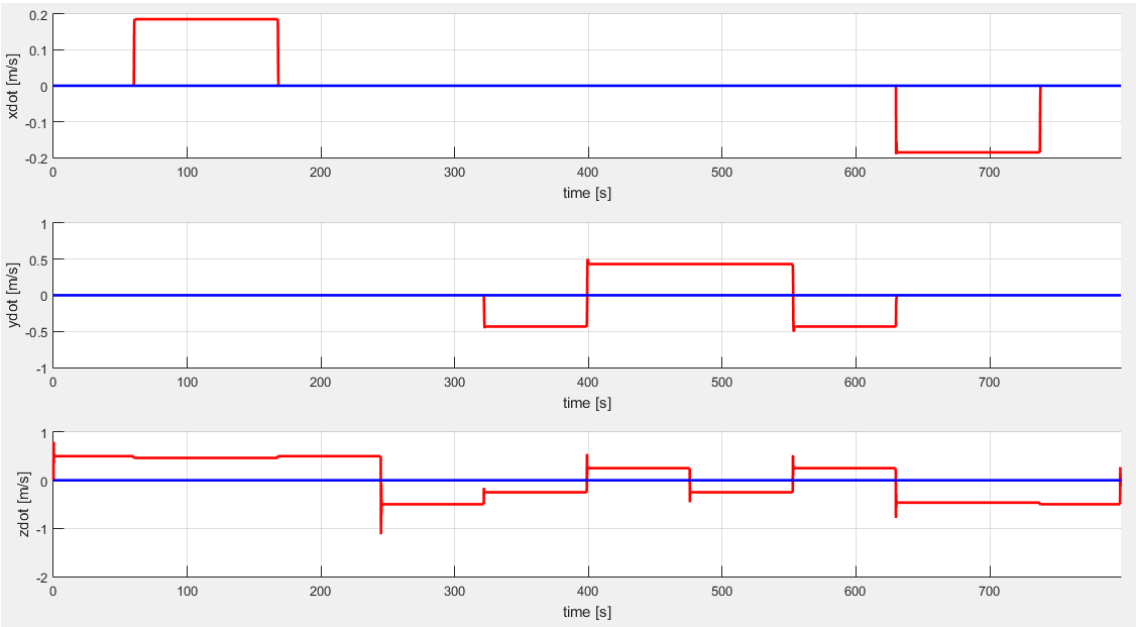

Figure 45: Desired and Actual values of velocity during the trajectory.

5.2.3 MATLAB Simulation: Round trajectory over the Blades

On this simulation case, the trajectory is generated based on the equations of Sin and Cosine to describe the longitudinal and transversal position, respectively, as shown below, where the *r* represents the radio of the circumference described in the trajectory.

$$x = Cos(2\pi) \times r$$

$$y = Sin(2\pi) \times r$$

With this, is possible to define the starter point as a corner of the Wind Turbine, creating a circular trajectory, with a radio of 40m, which in this case, the initial longitudinal coordinate is 40. The initial height was defined as 60m, once with the layout presented on Figure 2, this is the minimum height that a blade from this model of Wind Turbine can be displaced. This path was developed so the RPAS could execute three laps, starting at 60m and finishing at 118,5m which is the maximum height of the Wind Turbined.

On Figure 46 it is possible to see the desired trajectory, represented by the blue color, and the actual trajectory, represented by the red color. This contains the trajectory described by the quadcopter. Since there is not a clear distinction between the desired and actual trajectory it is possible to conclude that these are accurate results facing the desired values.

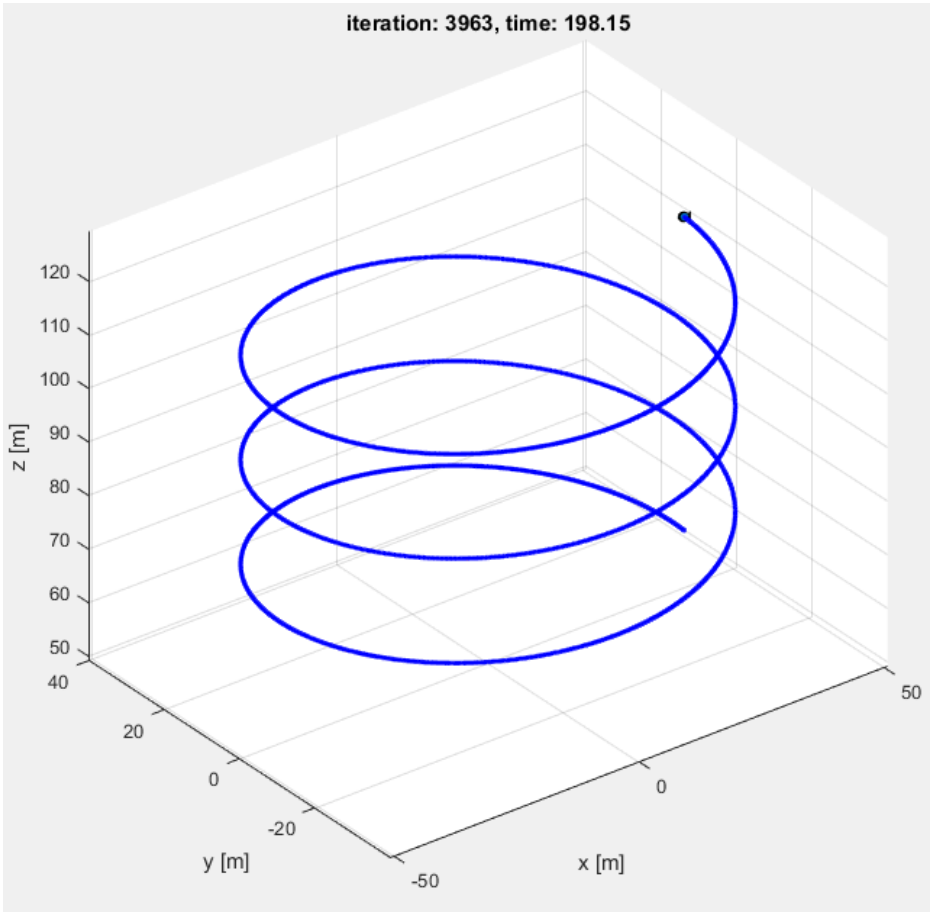

Figure 46: Round trajectory developed with three laps over the Wind Turbine.

On Figure 47, it is represented by the red color the actual trajectory and in blue the desired trajectory. The first row represents the trajectory over the *x* axis, the second row represents the trajectory over the *y* axis and the third one represents the trajectory over the *z* axis. On this case, no oscillations are seen. Thus, it is possible to say that the gains are properly developed in order to provide an accurate trajectory.

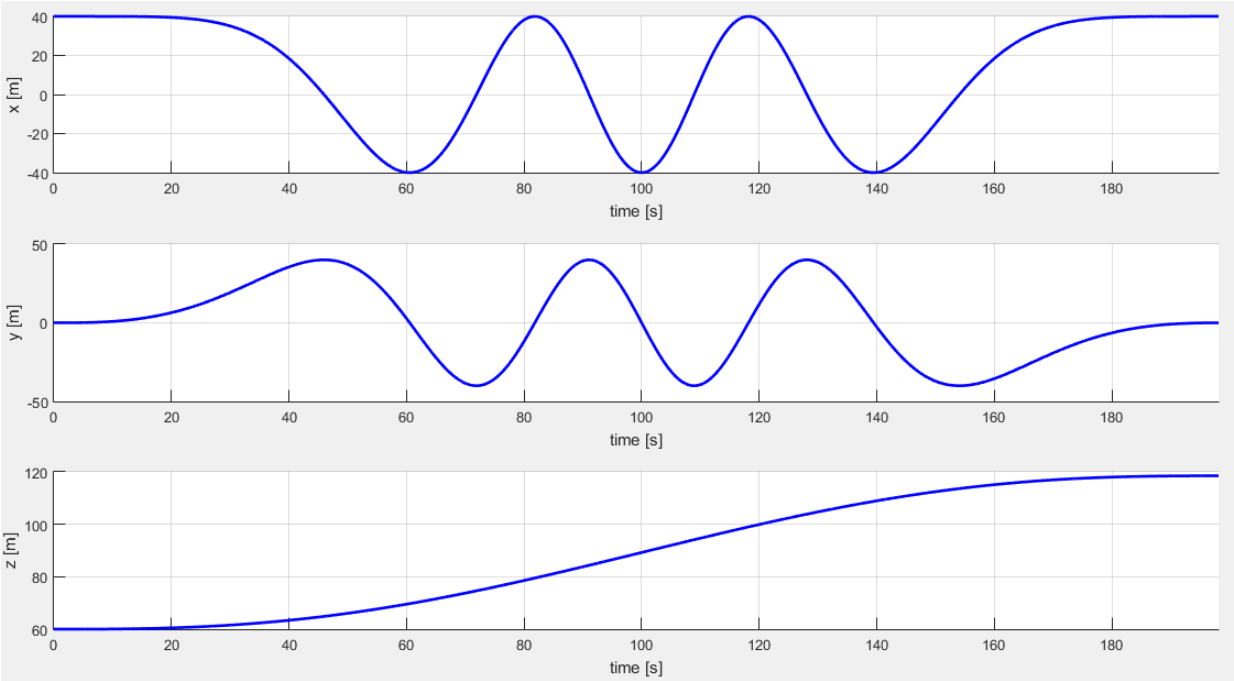

Figure 47: Position over time related to the path described.

On Figure 48 can be seen the red and blue color, which represent the actual and desired values, respectively, regarding the velocities described by the RPAS. The first row represents the velocity over the *x* axis, the second row represents the velocity over the *y* axis and the third one represents the velocity over the *z* axis. On this case, no oscillations are shown. Thus, it is possible to say that the gains are properly developed in order to provide an accurate trajectory.

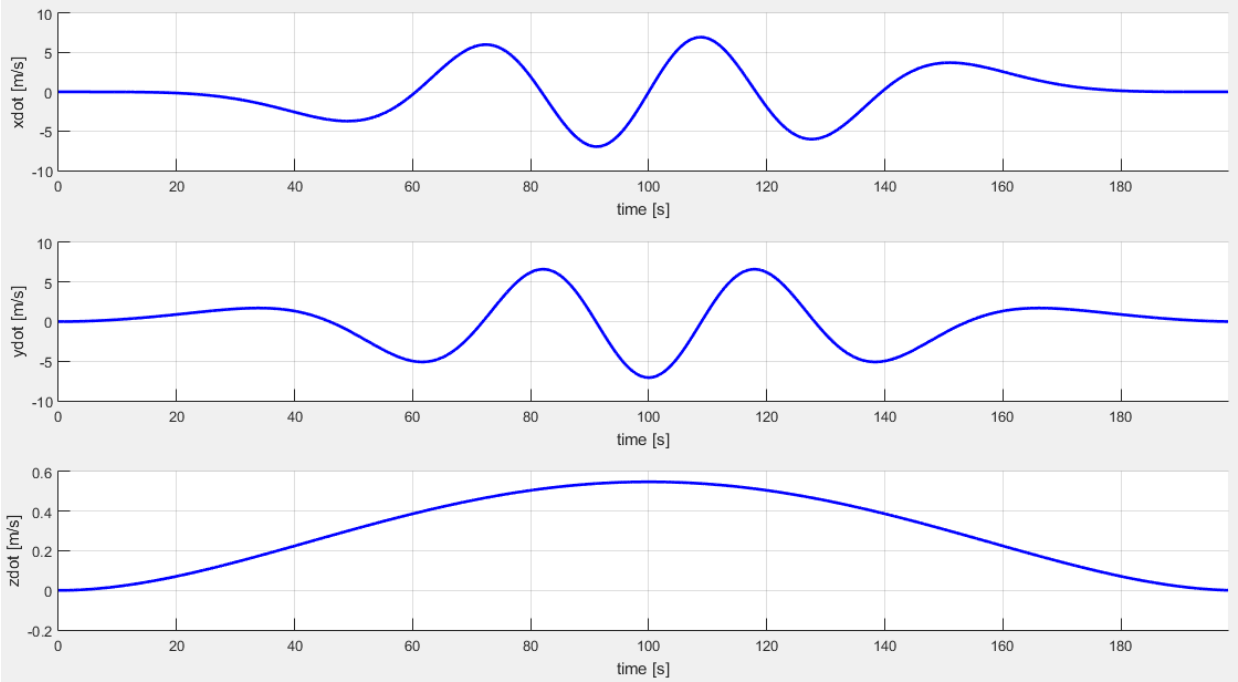

Figure 48: Velocity over time related to the trajectory described.

Chapter 6: Conclusion

The main purpose of this project was to create a semi-autonomous model for wind turbine blades resorting to visual inspection, by an Android application, taking advantage of DJI equipment.

On the first stage, it was defined some basic goals to achieve, related to the mission development and to the restrictions of the RPAS.

To define goals, is a fundamental part of every project, once it restricts general activities and crave for priorities on which the whole process will be developed. Based on these, among all the options available, was selected the Phantom 4. This RPAS possesses a wide range of sensors that enable to determine distance to objects, with a plus of a high stable camera, that provides high quality photos, minimizing the error associated with the visual inspections.

Furthermore, was used Java commands to execute diversified options, enabling the user, to generate a trajectory based on his singular mission preferences, providing an accurate solution. Among these preferences, it was allowed to define the height over each waypoint, the velocity that the mission would be executed and also the end action.

Moreover, was developed a MATLAB code, which was adapted from previous works. [68] This code has incorporated some of the data related to the Phantom 4, such his mass and arm length, fundamental parameters for the code developed on MATLAB.

Being this a composite material surface, the data obtained can be misunderstood and lead to false damage detection, or even miss a defected part. Thus, this technique must be quite precise, in order to minimize errors. This was a primary requirement defined, establishing the control methods, only for small angles oscillations, near hover state.

By making use of Java Package, it is known that the MATLAB code can complement the Android application, in order to provide a more complete surveillance. With it, was reached the main objective, by providing innovative solutions for wind turbine blades inspection, empowering renewable energies.

Providing a semi-autonomous inspection, on which the user could control the gimbal, enables to capture photos from a desired angle, establishing a more accurate comparison, between previous inspections when overlapping the photos.

Being this a project on the edge of the present technology, regarding wind turbine inspections, it is needed to say that, all the investment made, should be recovered within 2 to 3 years, due to the continuous improvements made on RPAS, becoming outrun by others. This means, projects that are taken in consideration for long term, as 10 years, are in risk of losing the profit of it.

6.1 Difficulties

This project was a great challenge. From the start, it was known that would be based on a completely different language from the ones studied before. This means, a high importance and additional time had to be attributed to the learning of Java language.

To the creation of applications, web research and online courses were made, in order to improve the knowledge relatively to object oriented programming, fundamental parts of the code and even familiarization with the Android environment.

The request for permissions in order to use concepts created from different companies, such as Google and DJI, and incorporating it on the code was also a challenge. Although, some tutorials are provided in the DJI page. The fundamental information is quite restricted, on which only basic concepts are explained, requiring a self-learning, critical thinking and problem-solving skills to overcome a complex approach to the solution.

The usage of the whole system is also not self-explanatory, to which several hours around the programing environment, web searching, professors specialized in Java plus a very self-indulgent spirit to achieve this project were required.

Nevertheless, Java is a well-defined language, enabling everyone that desires, to learn it. After taking the initial steps, it is possible to understand the main commands, which save several code lines, and avoid repetitively between objects.

6.2 Results

After overcoming the familiarization with Java, creating an application using Android Studio was very challenging and bracing. Being this an operative system used for most of the people, understanding how the basic concepts work behind the users' interface can work as a great advantage to structure the code.

By combining Java and DJI equipment, was possible to obtain a very efficient and interactive environment, diversifying his uses, requiring only several hours from the user to learn how to play in his advantage their functionalities, with a relatively low investment material.

As it was mentioned, by having a proportional controller, the system has the capability to stabilize itself when its values oscillate from the desired ones. The derivative gain helps to predict the systems behavior.

Even though it was generated a spline defined by a 7^{th} order polynomial, the polynomial used to describe these waypoints was from first order, once that instead of a smooth trajectory it is preferred to maintain an accurate path that allows the quadcopter to pass by the middle of the blades, enabling the user to obtain more detailed information.

It is possible to see that, the trajectory that resulted more accurate is the one which describes a round path across the blades. Although, due to the dimensions of the errors presented on this

type of structures, it might not be the most adequate when searching for small defects. Nevertheless, it gives a good overview of the whole structure.

Although it was not possible to provide a simulation which incorporates the two codes, the results show great advantages comparted with the current inspection methods.

6.3 Future works

It is possible to see that this solution requires several manual data, such as geographic position of the wind turbines, length of the blades and so on. A proposal that is made, is to incorporate on the current code, a TapFly and active tracking missions. [78] With this, would be possible to obtain a visual inspection based on instructions given by the technician that is executing the inspection, which would not be limited by the coordinates previous defined in the code.

Another proposal of future work would be a manual collection of photos from similar angles of a blade, in order to overlap them. With it, would be possible to create a study of this technique, and evaluating his precision regarding the photos obtained, before implementing it on a RPAS. This would give a better understanding of the angles that the photos should be obtained, providing extra information regarding the trajectory required for the inspection.

A fundamental part of this inspection would be the creation of a database, that would allow the flowing of the data, from the RPAS to a storage. This would make possible for technicians to overlap the pictures and create a timeline between them in a more efficient way. This would allow to see the deterioration of the blades over time, obtaining a better idea where they occur and how to prevent them. A virtual model of the displacements of the Wind Turbines would provide a good extra on the software, in order to assignee, after the inspection of the photos, the ones that require maintenance.

Associated with the detection of defects on wind turbines, a possible future work that could be developed is while the inspection is in progress, if a defect is detected, the RPAS would approach the respective region, in order to capture more photos, providing a better analysis over the failure present on the blade. With this, would be possible to determine with higher precision the time within which, the blade would have to be repaired.

Regarding the Wind Farms, where numerous wind turbines are located, would be also a good project to implement the work of Paulo Neves. [82] By incorporating a Docking system, would allow the RPAS to perform a battery exchange when needed, allowing the inspection of various wind turbines without the need of the direct human intervention. In this case, it is supposed that the inspection is completely autonomous.

On the MATLAB code, the velocity between waypoints should be weighted in order to obtain accurate data, allowing the capture of high quality photos, decreasing the inspection error. The trajectory should also be more complex, in terms that it should include the two sides of the blade, leading edge and trailing edge, instead of straight line between the tip and the root.

Upgrading this algorithm to a PID controller would make the system more accurate. The integrative gain would be able to have control over the magnitude and the duration of the error. Although it is a very limited method.

The ultimate goal of this project, would be to provide a completely autonomous inspection. This would give total independency from the technician executing the maintenance. With it, would be attributed more viability to this action, plus repeatability, in order to prevent the deterioration of the blade and consecutively, his lower efficiency. Therefore, embracing semi-autonomous inspection using RPAS is a way to empower renewable energies, equilibrating the competition with fossil fuels.

Solutions like this are already available on the market from companies like Sky Specs [79] and Pro-Drone [80] which provide a completely autonomous and robust inspection. Projects like PITCH ME [83] promote and provide valuable prizes to the best projects presented in order to empower the inspection of wind turbines.

Bibliography

[1] N. W. C. Collaborative, “Wind Turbine Interactions with Birds,Batsm and their Habitats: A Summary of Research Results and Priority Questions,” Natl. Wind Coord. Colla., 2010.

[2] R. G. Iain Staffell, “How does wind farm performance decline with age?,” Imperial College Business School, London, Volume 66, 2014, pages 775-786.

[3] L. FRIED, “Global Wind Energy Council,” 2017. [Online]. Available: http://www.gwec.net/wp-content/uploads/vip/GWEC_PRstats2016_EN_WEB.pdf. [Accessed 23 8 2017].

[4] A. Fritz, “Wind Energy: An Emerging Energy Resource,” NDGS Newsletter Volume 26, 1999, pages 1-5.

[5] S. F. University, “Lecture notes,” [Online]. Available: http://www.sfu.ca/phys/346/121/lecture_notes/lecture29.pdf. [Accessed 23 8 2017].

[6] “Sky Specs,” [Online]. Available: http://www.skyspecs.com/skyspecs-solution/value-proposition/. [Accessed 24 8 2017].

[7] R. W. Pierre Tchakoua, “Wind Turbine Condition Monitoring: State-of-the-Art Review, New Trends, and Future Challenges,” energies, Canada, volume 7, 2014, pages 2595-2630.

[8] “Global Wind Statics,” 2 10 2017. [Online]. Available: http://www.aweo.org/windmodels.html. [Accessed 5 9 2017].

[9] DJI, “User Manual” [Online]. Available: https://dl.djicdn.com/downloads/phantom_4/en/Phantom_4_User_Manual_en_v1.0.pdf. [Accessed 24 8 2017].

[10] G. M. a. N. Hallermann, “Quality Assement of Unamanned Aerial Vehicle (UAV) Based Visual Inspection of structures,” Germany, 2014, pages 3-18.

[11] A. O. a. J. P. Thomas D. Ashwill, “Blade Reliability Collaborative: Collection of Defect, Damage and Repair Data,” SANDIA National Laboratories, California, 2013, pages 5-29.

[12] E. J. R. S. A. V. R. Raisutis, “The review of non-destructive testing techniques suitable for inspection of the wind turbine blades,” ULTRAGARSAS Journal, Lithuania, volume 63, No. 1., 2008.

[13] A. Pinho, “Turbine Blades Inspection Methods,” 8.2 Especialistas em Energias Renovaveis, Lda., Curitiba, 2016.

[14] A. M. T. Fausto Pedro Garcia Marquez, “Condition monitoring of wind turbines: Techniques and methods,” ELSEVIER, Spain, volume 46, 2012,Pages 169-178.

[15] J. C. a. S. J. D. C M Yeum, “Autonomous image localization for visual inspection of civil infrastructure,” IOPScience, volume 26, No. 3, 2017, Smart Materials and Structures.

[16] Margaret Rouse, SearchMicroservices “object-oriented programming (OOP),” [Online]. Available: searchmicroservices.techtarget.com/definition/object-oriented-programming-OOP. [Accessed 5 9 2017].

[17] Margaret Rouse, Search Microservices “Java,” [Online]. Available: http://searchmicroservices.techtarget.com/definition/Java. [Accessed 5 9 2017].

[18] Oracle “Lesson: Object-Oriented Programming Concepts,” [Online]. Available: https://docs.oracle.com/javase/tutorial/java/concepts/index.html. [Accessed 5 9 2017].

[19] Margaret Rouse, SearchDataManagement “data modeling,” [Online]. Available: http://searchdatamanagement.techtarget.com/definition/data-modeling. [Accessed 5 9 2017].

[20] Margaret Rouse, SearchDataManagement “data,” [Online]. Available: http://searchdatamanagement.techtarget.com/definition/data. [Accessed 5 9 2017].

[21] Margaret Rouse, SearchMicroservices “object,” [Online]. Available: http://searchmicroservices.techtarget.com/definition/object. [Accessed 5 9 2017].

[22] Margaret Rouse, WhatIs “class,” [Online]. Available: http://whatis.techtarget.com/definition/class. [Accessed 5 9 2017].

[23] Don Rose, SearchSoftwareQuality “application,” [Online]. Available: http://searchsoftwarequality.techtarget.com/definition/application. [Accessed 5 9 2017].

[24] Margaret Rouse, SeachMicroservices "object-oriented programming (OOP),". [Online]. Available: http://searchmicroservices.techtarget.com/definition/object-oriented-programming-OOP. [Accessed 5 9 2017].

[25] Margaret Rouse, WhatIs "macro" [Online]. Available: http://whatis.techtarget.com/definition/macro. [Accessed 24 8 2017].

[26] P. Christensson, TechTerms "SDK," 15 April 2010. [Online]. Available: https://techterms.com/definition/sdk. [Accessed 24 8 2017].

[27] “Android App Development for Beginners: Android Overview,” edX [Online]. Avaliable: https://courses.edx.org/courses/course-v1:GalileoX+CAAD002X+1T2017/course/ [Accessed 5 9 2017].

[28] Developers “Meet Android Studio,” [Online]. Available: https://developer.android.com/studio/intro/index.html. [Accessed 25 8 2017].

[29] Android Studio “Android Plugin for Gradle Release Notes,” [Online]. Available: https://developer.android.com/studio/releases/gradle-plugin.html. [Accessed 5 9 2017].

[30] Developers “Application Fundamentals,” [Online]. Available: https://developer.android.com/guide/components/fundamentals.html. [Accessed 25 8 2017].

[31] edX “Android App Development for Beginners: What in an APP?,” [Online]. Avaliable: https://www.edx.org/course/android-app-development-beginners-galileox-caad002x-0 [Accessed 5 9 2017].

[32] Developers “Activating components,” [Online]. Available: https://developer.android.com/guide/components/fundamentals.html#ActivatingComponents. [Accessed 5 9 2017].

[33] edX “Android App Development for Beginners: APP COMPONENTS,” [Online]. Avaliable: https://www.edx.org/course/android-app-development-beginners-galileox-caad002x-0 [Accessed 5 9 2017].

[34] Developers “The manifest file,” [Online]. Available: https://developer.android.com/guide/components/fundamentals.html#Manifest. [Accessed 25 8 2017].

[35] Developers “Application Fundamentals,” [Online]. Available: https://developer.android.com/guide/components/fundamentals.html. [Accessed 24 8 2017].

[36] Wikipedia “Application programming interface,” [Online]. Available: https://en.wikipedia.org/wiki/Application_programming_interface. [Accessed 5 9 2017].

[37] Oracle “What is an Interface?,” [Online]. Available: https://docs.oracle.com/javase/tutorial/java/concepts/interface.html. [Accessed 5 9 2017].

[38] Developers “App components,” [Online]. Available: https://developer.android.com/guide/components/fundamentals.html#Components. [Accessed 25 8 2017].

[39] Developers “Activity,” [Online]. Available: https://developer.android.com/reference/android/app/Activity.html. [Accessed 24 8 2017].

[40] Udacity “O ciclo de vida da Activity no Android,” [Online]. Available: https://www.youtube.com/watch?v=85MppyLJHz0&feature=youtu.be. [Accessed 25 8 2017].

[41] Developers “The Activity Lifecycle,” [Online]. Available: https://developer.android.com/guide/components/activities/activity-lifecycle.html. [Accessed 24 8 2017].

[42] Developers “Services,” [Online]. Available: https://developer.android.com/guide/components/services.html. [Accessed 24 8 2017].

[43] Developers “BroadcastReceiver,” [Online]. Available: https://developer.android.com/reference/android/content/BroadcastReceiver.html. [Accessed 24 8 2017].

[44] Developers “Content Providers,” [Online]. Available: https://developer.android.com/guide/topics/providers/content-providers.html. [Accessed 24 8 2017].

[45] Developers “Intent,” [Online]. Available: https://developer.android.com/reference/android/content/Intent.html. [Accessed 25 8 2017].

[46] Developers “Intents and Intent Filters,” [Online]. Available: https://developer.android.com/guide/components/intents-filters.html. [Accessed 25 8 2017].

[47] Wikipedia “Gradle,” [Online]. Available: https://en.wikipedia.org/wiki/Gradle. [Accessed 5 9 2017].

[48] G. M. API, Developers Google “Google Maps APIs,” [Online]. Available: https://developers.google.com/maps/android/. [Accessed 5 9 2017].

[49] G. M. APIs, Developers Google “Pricing and Plans,” [Online]. Available: https://developers.google.com/maps/pricing-and-plans/. [Accessed 5 9 2017].

[50] DJI, Developers DJI “Run Sample Application,” [Online]. Available: https://developer.dji.com/mobile-sdk/documentation/quick-start/index.html. [Accessed 5 9 2017].

[51] Google, “Upload an app,” [Online]. Available: https://support.google.com/googleplay/android-developer/answer/113469?hl=en. [Accessed 5 9 2017].

[52] Developers, “LinearLayout,” [Online]. Available: https://developer.android.com/reference/android/widget/LinearLayout.html. [Accessed 5 9 2017].

[53] Developers, “Relative Layout,” [Online]. Available: https://developer.android.com/guide/topics/ui/layout/relative.html . [Accessed 5 9 2017].

[54] tutorialspoint, “Android Relative Layout,” [Online]. Available: https://www.tutorialspoint.com/android/android_relative_layout.htm. [Accessed 5 9 2017].

[55] Developers, “FrameLayout,” [Online]. Available: https://developer.android.com/reference/android/widget/FrameLayout.html. [Accessed 5 9 2017].

[56] Android Asset Studio, "Launcher icon generator" [Online]. Available: https://romannurik.github.io/AndroidAssetStudio/icons-launcher.html. [Accessed 8 8 2017].

[57] Wikipedia, “Pixel density,” [Online]. Available: https://en.wikipedia.org/wiki/Pixel_density. [Accessed 5 9 2017].

[58] Developers, “App Manifest,” [Online]. Available: https://developer.android.com/guide/topics/manifest/manifest-intro.html. [Accessed 5 9 2017].

[59] Developers, “Button,” [Online]. Available: https://developer.android.com/reference/android/widget/Button.html. [Accessed 5 9 2017].

[60] Developers, “CompoundButton.OnCheckedChangeList,” [Online]. Available: https://developer.android.com/reference/android/widget/CompoundButton.OnCheckedChangeListener.html. [Accessed 5 9 2017].

[61] Developers Google, “Google APIs for Android,” [Online]. Available: https://developers.google.com/android/reference/com/google/android/gms/maps/GoogleMap.OnMapClickListener. [Accessed 5 9 2017].

[62] Developers, “LayoutInflater,” [Online]. Available: https://developer.android.com/reference/android/view/LayoutInflater.html. [Accessed 5 9 2017].

[63] DJI Developers, “Creating a MapView and Waypoint Application,” [Online]. Available: https://developer.dji.com/mobile-sdk/documentation/android-tutorials/GSDemo-GoogleMap.html. [Accessed 5 9 2017].

[64] DJI Developer, “Creating a Camera Application,” [Online]. Available: https://developer.dji.com/mobile-sdk/documentation/android-tutorials/index.html. [Accessed 5 9 2017].

[65] Developers, “Supporting Different Languages and Cultures,” [Online]. Available: https://developer.android.com/training/basics/supporting-devices/languages.html. [Accessed 5 9 2017].

[66] DJI, “Phantom 4 Downloads,” [Online]. Available: http://www.dji.com/es/phantom-4/info#downloads. [Accessed 5 9 2017].

[67] DJI, “Implementing the UI of Application,” [Online]. Available: https://developer.dji.com/mobile-sdk/documentation/android-tutorials/GSDemo-GoogleMap.html#implementing-the-ui-of-application. [Accessed 5 9 2017].

[68] V. Kumar, “Robotics: Aerial Robotics - Lecture Notes,” University of Pensilvania, Pensilvania, 2014.

[69] D. W. Mellinger, “Trajectory Generation and Control for Quadrotors,” University of Pennsylvania, Pennsylvania, 2012, pages 10-98.

[70] A. D. F. L. K. Subbarao, “Backstepping Approach for Controlling a Quadrotor Using Lagrange Form Dynamics,” Texas, volume 56, Issue 1-2, 2009, pp 127-151.

[71] N. N. J.G. Ziegler, “Optimum Settings for Automatic Controllers,” N.Y., J. Dyn. Sys, Meas., Control 115(2B), 1993, 220-222.

[72] H. Khalil, “Adaptive output feedback control of nonlinear systems represented by input-output models,” IEEE, MI, volume 41, Issue 2, 1996, pages 177-188.

[73] C. Rawal, MathWorks “MATLAB Compiler SDK,” [Online]. Available: https://www.mathworks.com/products/matlab-compiler-sdk.html. [Accessed 5 9 2017].

[74] Google Code, “matlabcontrol - Walkthrough.wiki,” [Online]. Available: https://code.google.com/archive/p/matlabcontrol/wikis/Walkthrough.wiki. [Accessed 5 9 2017].

[75] Ned Twigg, Github, "matconsolectr,” [Online]. Available: https://github.com/diffplug/matconsolectl. [Accessed 5 9 2017].

[76] F. VI, “Boletin Oficial de Estado,” 17 10 2014. [Online]. Available: http://www.boe.es/boe/dias/2014/10/17/pdfs/BOE-A-2014-10517.pdf. [Accessed 6 9 2017].

[77] «Ministerio de Fomento,» Aeronavegabilidad, [Online]. Available: http://www.fomento.es/MFOM/LANG_CASTELLANO/DIRECCIONES_GENERALES/AVIACION_CIVIL/INFORMACION/NORMATIVA/NORMATIVA_BASICA/AERONAVEGABILIDAD/. [Accessed: 6 9 2017].

[78] DJI Developers, “Creating a TapFly and ActiveTrack Missions Application,” [Online]. Available: https://developer.dji.com/mobile-sdk/documentation/android-tutorials/P4MissionsDemo.html. [Accessed 6 9 2017].

[79] Inspections “Sky Specs,” [Online]. Available: http://www.skyspecs.com/. [Accessed 8 9 2017].

[80] Inspections, “Pro-Drone,” [Online]. Available: http://www.pro-drone.eu/. [Accessed 8 9 2017].

[81] W. F. Deal, “Wind Power: An Emerging Energy Resource,” International Technology and Engineering Educators Association, volume 70, n.1, 2010, p 9-15.

[82] P. T. d. S. Neves, "A Docking System for Battery Exchange," Universidade da Beira Interior, Covilhã, 2014.

[83] Inspections, "Wind Power Montly," [Online]. Available: http://www.windpowermonthly.com/event/blade-inspection-damage-and-repair/pitch-me. [Accessed 18 9 2017].

Appendix A

Paper related to control and modelation of the trajectory in MATLAB

TRAJECTORY CONTROL AND MODELATION FOR MAINTENANCE ON TURBINE BLADES BY A RPAS

Paulo ANTUNES[1], Kouamana BOUSSON[2], J.A. Garcia-Manrique[3]

[1] Aerospace Sciences Department, Universidade da Beira Interior, Address: Calçada Fonte do Lameiro 6201-001 Covilhã, Portugal
[2] Aerospace Sciences Department, Universidade da Beira Interior, Address: Calçada Fonte do Lameiro 6201-001 Covilhã, Portugal
[3]Department of Mechanical Engineering and Materials, Universitat Politécnica deValencia, Camino Vera, s/n, 5E, 46022, Valencia, Spain

E-mails: [1]paulo9o@email.com
[2]bousson@ubi.pt; [3]jugarcia@mcm.upv.es

Abstract

The big structures maintenance represents a difficult task for human action to achieve since the height requires numerous support, effort and time to accomplish. Allied with composed materials, big structures can have higher efficiency than with conventional materials, but also harder to conduct inspections though devices such as cameras. Quadcopter are a type of RPAS (Remotely Piloted Aircraft System) that take advantage of the exponential growth on technology and nanotechnology. Owing to this, has become possible to perform surveillance on different types of structures, presenting a comfortable way to execute this action on dangerous and difficult access spaces by providing safer methods, bringing experience, of qualified workers, and technology together. This paper will provide the methods for a trajectory relying on the specification of a Phantom 4, providing a code that might be adapted for a more complex algorithm. Therefore, the main objective will be to establish a path over previously defined waypoints, incorporating the whole blade. The results were satisfying once this was achieved. Nevertheless, some subjects might be improved, like the velocity between waypoints and the polynomial selected to defined the trajectory.

KEYWORDS: QUADCOPTERS, MAINTENANCE, RPAS, WAYPOINTS, NANOTECHNOLOGY, SIMULATION, COMPOSED MATERIALS.

Introduction

The quadcopter is a type of Remotely Piloted Aircraft System (RPAS) that can perform Vertical Take Off and Landing (VTOL). This provides the maneuverability that most of RPAS lack off, not only for VTOL but also for his dimensions that allow to incorporate onboard sensors. It is an under-actuated system with four inputs (roll, pitch, yaw and throttle) and six outputs. [1]

With this, is possible to operate in constrained and difficult access spaces. With the technologic exponential increasing for this kind of machines, comes the interest of projecting them into our everyday life needs. Achieving big structure maintenance represent a difficult task for human action since the height requires numerous support, effort and time to accomplish. Thus, machines come to give the help that is demanded. This type of task not only are expensive, but dangerous since most of these techniques are either outdated or inadequate, like the rope descendent technique.

Figure 1: Inspection on Wind Turbine conducted by rope descendent technique. [6]

With the use of remotely controlled or semi-autonomously operations, the monitoring and inspection of big structures like wind turbine blades, telecommunication antennas, buildings and others can be brought to a new level of quality and economy. [2] Thus, this article presents an efficient approximation, to what might be a solution for the next years.

There is no doubt that removable energies are the road that humankind must pursue, in order to obtain sustainability on the near future. Wind blades are a particular structure. Not only by the airfoil used but also the irregular shape which is presented on the tip. So as to maintain these structures, presents a complex subject. More common than ever, these blades are produced in composite materials, that sometimes are difficult to predict his lifetime. [9]

Adding the access problem, can minimize the power obtained.

Figure 2: Damaged leading edge from a Wind Turbine blade. [12]

This paper summarizes a model created to describe the movements of a RPAS. This kind of task is possible by incorporating a GPS (Global Positioning System), IMU (inertial measurement units) allowing to travel between previously defined waypoints, to obtain a live inspection with a high-definition camera. Based on this model was created a simulation on Matlab. Since the cameras need to be steady to capture high quality photos, decreasing the error associated to the surveillance is a challenge, so as to only small angle were considered a model for small angles control.

Quadcopters can have diverse applications such as: Precision Farming, Archeology, Photography and Robot First Response. [7]

In this paper, it is adapter a trajectory generation methodology, developed by [5] on which is possible to plan a trajectory from an initial point to a desired one. The main objectives of the methodology of this paper are:

- This trajectory must present an approach to a real-world surveillance.
- The trajectory must obey to the restrictions established by the dynamics and input of the quadcopter.
- The control law applied to this system must be able to re-plan the trajectory at each control update, and apply the new inputs to the first section of the loop.
- Approximate a trajectory to a desired one to obtain good visual tracking for the captures of the photos.

Therefore, will be presented a dynamic model used, which will provide an idea of the control inputs and their function when controlling the system.

Moreover, will be defined control laws, the nested loop which allows us to control the position an attitude of the quadrotor and some assumptions made in order to optimize the trajectory for the end it was created.

Based on these principles, will be defined the equations used to obtain a trajectory tracking based on a desired position. To end, will be presented a problem proposal definition and the results obtained, which will be commented with some future improvements.

1. Dynamic Model

The robot's movements can be defined by six degrees of freedom. In order to characterize them, the Inertial frame (*a*) and Body frame (*i*) frame were created. The Inertial frame is composed by a_1, a_2 and a_3, being the positive part of a_3 pointing upward. In addiction, was defined the body frame (*i*), which has the origin latch onto the quadcopters center of mass, as it is possible to see on the Figure 3.

Figure 3: Coordinate systems of the quadcopter.

To characterize the system, will be used Z – X – Y Euler angles, modeling the rotation of the quadcopter.

In addition, to characterize the vehicles attitude, it is defined a rotation matrix, ${}^{a}R_{i}$, presented on the equation (1). It enables to describe a vector from the Inertial System, *a*, in the vehicles body system, *i*, multiplying the vector from the inertial by the rotation matrix, or vice versa.

$$ {}^{a}R_{i} = \begin{bmatrix} c\psi c\theta - s\phi s\psi s\theta & -c\phi s\psi & c\psi s\theta + c\theta s\phi s\psi \\ c\theta s\psi + c\psi s\phi s\theta & c\phi c\psi & s\psi s\theta - c\psi c\theta s\phi \\ -c\phi s\theta & s\phi & c\phi c\theta \end{bmatrix} \quad (1) $$

The sθ and cθ, represent sin(θ) and cos(θ), respectively. This also applies for ϕ and ψ *angles*.

Defining *r*, as the position vector with respect to the center of mass (*i*) from the inertial frame (*a*).

The primary forces on the system are the ones produced by gravity, on the $-i_3$ direction, and the ones produced by the motors F_i, on the i_3 direction. To summarize this, is provided equation (2) that represents the acceleration with respect to the center of mass. [1]

$$ m\ddot{r} = \begin{bmatrix} 0 \\ 0 \\ -mg \end{bmatrix} + R \begin{bmatrix} 0 \\ 0 \\ F_1 + F_2 + F_3 + F_4 \end{bmatrix} \quad (2) $$

The angular velocities are defined as *p*, *q* and *r*, on equation (3). These are related with the derivatives of roll, pitch and yaw angles.

$$ \begin{bmatrix} p \\ q \\ r \end{bmatrix} = \begin{bmatrix} c\theta & 0 & -c\phi s\theta \\ 0 & 1 & s\phi \\ s\theta & 0 & c\phi c\theta \end{bmatrix} \begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} \quad (3) $$

1.1 Control Inputs

Over the referential system established, the quadcopters state is defined based on the longitudinal, transversal and vertical positions, first tree elements on equation (4). Nevertheless, his orientation over the tree axis should also be considered, last tree elements of equation (4). With this, is defined the quadrotors state. [1]

$$q' = [x \quad y \quad z \quad \phi \quad \theta \quad \psi]^T \tag{4}$$

$$x' = [q' \quad \dot{q}']^T \tag{5}$$

Regarding equation (5), is represented the state of the rigid body, which is composed by his six-dimension vector (q') and its rate of change ($\dot{q}'$).

On this model, a series of simplifications were made. It is assumed that the thrust produced by the rotors, can be swapped immediately. This is not completely accurate, once that a dynamical system is characterized for being a system on which the effects of actions do not occur immediately. It is also limited in terms of the vehicle's body rate, by the gyroscopes and tracking controllers. In addition, the force produced by each motor, a_i was limited, being the maximum value produced by all the rotors, 1.5 times of the quadcopters weight.

Being i = 1,2,3,4.

$$a_{min} \leq a_i \leq a_{max} \tag{6}$$

On which a represents linear acceleration. The value of a_{min} is positive or zero, since these quadcopters do not possess the ability to stop or reverse the direction of rotation of the propellers. [8]

The maximum thrust is limited by peak motor torque. The total of vertical forces can be obtained by the sum of the forces produced by the motors plus the force of the gravity applied on the vehicle.

$$\vec{F}_{Total} = \vec{F}_{motor} + \vec{F}_{gravity} \tag{7}$$

Discretizing equation (7), it is possible to obtain equation (8), on which m represents the mass of the vehicle, g is the force of gravity, ω the angular speed of the motors and k_F the proportionality constant of the force produced by the motors. [5]

$$ma = \sum_{i=1}^{4} k_F \omega_i^2 + mg \tag{8}$$

1.1 Equations of motion

Related to these forces, the one regarding to the motors, creates moments over the three-referential axis established. Since motors 1 and 3 spin on the $-\,i_3$ direction they will produce a positive moment. Motors 2 and 4 spin on the i_3 direction, producing negative moments. The variable L represents the distance between the vehicle' center of mass and the position of the motor. With this defined, the inertia matrix related to the moments produced by the components of the vehicle can be written as: [5]

$$\mathrm{I}\begin{bmatrix}\dot{p}\\ \dot{q}\\ \dot{r}\end{bmatrix} = \begin{bmatrix}L(F_2 - F_4)\\ L(F_3 - F_1)\\ M_1 - M_2 + M_3 - M_4\end{bmatrix} - \begin{bmatrix}p\\ q\\ r\end{bmatrix} \times I \begin{bmatrix}p\\ q\\ r\end{bmatrix} \tag{9}$$

1.2 Motor model

The motors were modeled by a test of steady-state fixed rotor in order to determine a constant of proportionality that multiplied by the angular speed would allow to estimate the force produced:

$$F_i = k_F {\omega_i}^2 \tag{10}$$

Experimentation with a fixed rotor at steady-state show $k_F \approx 6.11 \times 10^{-3}\ N/rpm$. [5]

With the same ideology, it is possible to determine a constant of proportionality, to estimate the moment produced:

$$M_i = k_M {\omega_i}^2 \tag{11}$$

The constant k_M, was determined to be about $1.5 \times 10^{-9}\ N/rpm$. [4]

The components of angular velocity, relative to the body frame, are defined as:

$${}^a\omega_i = pi_1 + qi_2 + ri_3 \tag{12}$$

2. Control

2.1 Control Model

The rigid body dynamics have slower response compared to the motor dynamics. Incorporating them leads to a fifth order dynamic model that implies additional complexity without significant improvement in performance. [5]

Using Euler angles, it is possible to parameterize the orientation, position and velocity of the center of mass.

$$x = [x \quad y \quad z \quad \phi \quad \theta \quad \psi \quad \dot{x} \quad \dot{y} \quad \dot{z} \quad p \quad q \quad r]^T \tag{13}$$

Neglecting the parameterization by the position and velocity of the center of mass, rotation matrix and angular velocity, can be simplified to: [5]

$$u = [u_1 \quad u_2 \quad u_3 \quad u_4]^T \tag{14}$$

Where u_1 is the force from all the propellers and u_2, u_3, and u_4 are the moments related to the body frame axis. [5]

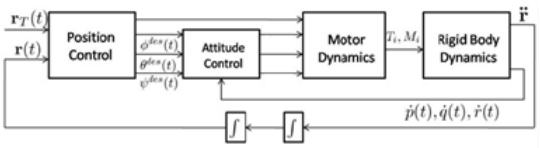

Figure 3: Nested control loops for position and attitude control. [7]

This nested loop, has an inner loop that corresponds to the attitude control and an outer loop that corresponds to the position control. On the inner loop is specified the orientation either with the rotation matrix or a series of roll, pitch and yaw angles. From the angular rates, will be calculated u_2. This is a function of the thrust and the moments that provide from the motor speeds. It will be calculated based on the desired attitude. On the outer loop, its aim is to obtain the position vector. This, will be compared with the actual position and actual velocity that enables us to obtain u_1. [5].

1.1 Attitude Control

Since this is an initial approach, the waypoints are defined so the quadcopter completes an itinerary that contains all the blades in order complete a visual survey with photos on them. To obtain a quality inspection, it is needed a high-quality photo, which requires a stable flight, no aggressive maneuvers, and mostly considered on near hover state. Thus, it is presented a control based on small angles. This controller is based on a linearization of the equations of motion and motor models, at an operating point that corresponds to the nominal hover state: [5]

$$\theta \approx \phi \approx 0 \quad (15)$$

$$\psi \approx \psi_0 \quad (16)$$

Considering small angles of pitch and roll:

$$c\phi \approx 1, c\theta \approx 1, s\phi \approx 0, s\theta \approx 0 \quad (17)$$

So, it is possible to deduce the nominal values at hover state for the first two inputs defined, the thrust of the motors, u_1, and the quadcopter attitude, u_2:

$$u_{1,0} = mg \quad (18)$$

$$u_{2,0} = 0 \quad (19)$$

By this, it is possible to assume that each rotor produces a quarter of the total thrust needed to maintain hover, so:

$$F_{i,0} = \frac{mg}{4} \quad (20)$$

And so, it is possible to deduce that the nominal angular velocity produced by the motors are: [5]

$$\omega_{i,0} = \omega_h = \sqrt{\frac{mg}{4k_F}} \quad (21)$$

Thus, it is possible to determine the desired rotor speeds using the desired ($u_{1,des}$) and the moments ($u_{2,des}$, $u_{3,des}$ and $u_{4,des}$):

$$u_{des} = \begin{bmatrix} k_F & k_F & k_F & k_F \\ 0 & k_F L & 0 & -k_F L \\ -k_F L & 0 & k_F L & 0 \\ k_M & -k_M & k_M & -k_M \end{bmatrix} \begin{bmatrix} \omega^2_{1,des} \\ \omega^2_{2,des} \\ \omega^2_{3,des} \\ \omega^2_{4,des} \end{bmatrix} \quad (22)$$

2. Trajectory Tracking

For the position control, will be presented two position methods, based on roll and pitch angles as inputs. Both methods are based on the linearization of the equations of motion, as shown above. The first method is for the nominal state, hover, used for maintaining the position. The second one is designed to track and follow a desired trajectory, *r(t)*, over the three directions. On these two methods, the position control algorithm will determine the desired roll and pitch angles, ϕ_{des} and θ_{des}, that can be used to compute the commanded accelerations [5]. This yaw angles, can be constant, ψ_0, or variable with time, $\psi_c(t)$. This method is similar to a backstepping approach. [10]

On this case, the position and orientation are fixed, which implies that all velocities and acceleration are zero as well the roll and pitch angles. The input u_2 must be close to zero, as well the rates p, q and r. The yaw angles can be non-zero if it's fixed, $\psi_c(t) = \psi_0$.[3] It is assumed that u_1 is very close to the weight.

With these assumptions, is now possible to define expressions to determine the accelerations over the three axes from Equation (2):

$$\ddot{r}_1 = g(\Delta\theta cos\psi_0 + \Delta\phi sin\psi_0) \quad (23)$$

$$\ddot{r}_2 = g(\Delta\theta cos\psi_0 + \Delta\phi sin\psi_0) \quad (24)$$

$$\ddot{r}_3 = \frac{1}{m}u_1 - g \quad (25)$$

Related to the products of inertia, once the axes are close the principal ones, it is possible to assume that:

$$I_{xx} \approx I_{yy} \quad (26)$$

Since the quadcopter is symmetric, it is possible to write the rates of change based on the linearization of the equations (2) and (3): [5]

$$\dot{p} = \frac{u_2}{I_{xx}} = \frac{L}{I_{xx}}(F_2 - F_4) \quad (27)$$

$$\dot{q} = \frac{u_3}{I_{yy}} = \frac{L}{I_{xx}}(F_3 - F_1) \quad (28)$$

$$\dot{r} = \frac{u_4}{I_{zz}} = \frac{\gamma}{I_{zz}}(F_1 - F_2 + F_3 - F_4) \quad (29)$$

The error related to the position, is defined as the different between the desired state and the actual state:

$$e_i = (r_{i,des} - r_i) \quad (30)$$

Furthermore, will be incorporated a PD controller. This contains a proportional gain that produces oscillations when the system is far from the desired state plus the derivative gain, which predicts system behavior and helps to stabilize it. Thus, can be written the following equation, in order that the error goes exponentially to zero:

$$(\ddot{r}_{i,des} - \ddot{r}_{i,c}) + k_{d,i}(\dot{r}_{i,des} - \dot{r}_i) + k_{p,i}(r_{i,des} - r_i) = 0 \quad (31)$$

1.1 Hover state

On this state, it is desired for the quadcopter to maintain a defined position, which means that the acceleration and velocity are zero:

$$\ddot{r}_{i,c} = \dot{r}_{i,c} = 0 \quad (32)$$

Moreover, it must be established that this condition is guaranteed. Thus, applying equation (31), leads to: [5]

$$\ddot{r}_{i,des} + k_{d,i}\dot{r}_{i,des} + k_{p,i}(r_i - r_{i,des}) = 0 \quad (33)$$

Based on equation (23-25), and reminding the fact that pitch and roll angles must be maintained at a constant value in order the achieve this state, the equation can be simplified to:

$$\ddot{r}_{1,des} = g(\theta_{des} cos\psi_T + \phi_{des} sin\psi_T) \quad (34)$$

$$\ddot{r}_{2,des} = g(\theta_{des} sin\psi_T + \phi_{des} cos\psi_T) \quad (35)$$

$$\ddot{r}_{3,des} = \frac{1}{m}u_1 - g \quad (36)$$

Moreover, with a PD controller, as written in the Equation (31), the error shall go exponentially to zero and also satisfy a second order differential equation (33). Related to this equation, there is a set of terms related to the specified trajectory, $\ddot{r}_{i,des}$, $\dot{r}_{i,des}$ and $r_{i,des}$, and a set of terms related with the actual trajectory $\dot{r}_i$ and r_i. Thus, it is from our interest to calculate the commanded sum of the thrusts from the motors, that will define the force that is desired to produce on the quadcopter, in order to maintain his position:

$$u_1 = mg + m\ddot{r}_{3,des} = mg - m(k_{d,3}\dot{r}_3 + k_{p,3}(r_{3,des} - r_3) \quad (37)$$

The commanded roll and pitch angles can be obtained once known the commanded acceleration in the x and y directions, as shown below:

$$\phi_c = \frac{1}{g}(\ddot{r}_{1,des} sin\psi_{des} - \ddot{r}_{2,des} cos\psi_{des}) \quad (38)$$

$$\theta_c = \frac{1}{g}(\ddot{r}_{1,des} cos\psi_{des} + \ddot{r}_{2,des} sin\psi_{des}) \quad (39)$$

The desired roll and pitch angles are defined as shown on the equation (40) and (41):

$$\phi_{des} = \frac{1}{g}(\ddot{r}_{1,c} sin\psi_{des} - \ddot{r}_{2,c} cos\psi_{des}) \quad (40)$$

$$\theta_{des} = \frac{1}{g}(\ddot{r}_{1,c} cos\psi_{des} + \ddot{r}_{2,c} sin\psi_{des}) \quad (41)$$

Furthermore, the pitch and roll velocities are defined as zero since these angles constant through all the hover state:

$$p_{des} = 0 \quad (42)$$

$$q_{des} = 0 \quad (43)$$

$$r_{des} = \dot{\psi}_T(t) \quad (44)$$

For the feedback loop, it is assumed that the commanded roll, pitch and yaw angles are known, as well their derivatives. Thus, all it is required for attitude feedback control are the actual values of roll, pitch and yaw angles plus their derivatives to determine u_2:

$$u_2 = \begin{bmatrix} k_{p,\phi}(\phi_{des} - \phi) + k_{d,\phi}(p_{des} - p) \\ k_{p,\theta}(\theta_{des} - \theta) + k_{d,\theta}(q_{des} - q) \\ k_{p,\psi}(\psi_{des} - \psi) + k_{d,\psi}(r_{des} - r) \end{bmatrix} \quad (45)$$

This model was projected to have a faster feedback by the attitude controller, than the position controller loop, as it can be seen in the Figure 3. [1]

1.2 3-D Trajectory

Since nominal state has already been defined, now it is going to be defined a controller that will enable to follow a three-dimensional trajectory, with the acceleration being close to near-hover state. This model follows the same assumptions made above, with the difference that $\ddot{r}_{i,c}$ and $\dot{r}_i$ are no longer zero.

It is considered near-hover state, assumptions of linear equations hold. To generate the desired acceleration, it is used equation (31), so the error goes exponentially to zero. From the equation (23), it is possible to determine the commanded state over the longitudinal and transversal directions, respectively:

$$\ddot{r}_{1,c} = \ddot{r}_{1,des} - k_{d,1}(\dot{r}_{1,des} - \dot{r}_1) - k_{p,1}(r_{1,des} - r_1) \quad (46)$$

$$\ddot{r}_{2,c} = \ddot{r}_{2,des} - k_{d,2}(\dot{r}_{2,des} - \dot{r}_2) - k_{p,2}(r_{2,des} - r_2) \quad (47)$$

Yet, since this model has some errors and limitations relating to the input thrusts, commanded trajectory may present some oscillations when sharp turns are encountered.

With this restriction, comes the need to provide a modification to better describe the trajectories. Defining $\hat{\boldsymbol{t}}$ as the trajectory's unit tangent vector and $\hat{n}$ as the normal unit vector, the binormal vector, $\hat{b}$, is obtained by cross product of the trajectory's unit tangent vector and the normal unit vector, $\hat{b} = \hat{t} \times \hat{n}$. With this, is possible to define position and velocity errors: [3]

$$e_p = \left((r_{i,des} - r_i) \cdot \hat{n}\right)\hat{n} + ((r_{i,des} - r_i) \cdot \hat{b})\hat{b} \quad (48)$$

$$e_v = \dot{r}_{i,des} - \dot{r}_i \quad (49)$$

Only the position error, over the normal plane to the curve's closest point is considered. The position error in tangential direction is ignored. With the expression defined in equation (31), it is possible to calculate the commanded acceleration, $\ddot{r}_{i,des}$:

$$(\ddot{r}_{i,des} - \ddot{r}_i) + k_d e_v + k_p e_p = 0 \quad (50)$$

All the other assumptions made for hover state hold, plus with the exceptions clarified on this section, it becomes possible to generate a trajectory on 3D. [5]

1. Trajectory generation

To the generation of trajectory, will be presented some equations depending on the precision required, their order will increase. Below is presented an equation for minimum snap trajectory, that requires a 7th order polynomials. Each polynomial piece travels between two waypoints, taking a stipulated amount of time, T_i.

Being p_i the polynomial between positions, ω_i and ω_{i+1}, that take,T_i, time to complete. The total amount of time is represented by:

$$S_i = \sum_{k=0}^{i-1} T_k \quad (51)$$

The polynomial that represents the restrictions through the waypoints is: [5]

$$p_i(t) = \alpha_{i0} + \alpha_{i1}\frac{t - S_i}{T_i} + \alpha_{i2}\left(\frac{t - S_i}{T_i}\right)^2 + \cdots + \alpha_{i7}\left(\frac{t - S_i}{T_i}\right)^7 \quad (52)$$

To complete this equation, it is required to determine all the coefficients α_{ij}. For this end, it is needed to establish constraints. First, it must go through all the waypoints: [5]

$$p_0(0) = \omega_0 = \alpha_{00} \quad (53)$$

$$p_i(S_i) = \omega_{i+1} \text{for all } i = 0 \dots n \quad (54)$$

Second, must be defined that the quadcopter will start and stop at rest: [5]

$$p_n^{(k)}(0) = 0 \text{ for all } 0 \leq k \leq 3 \quad (55)$$

$$p_n^{(k)}(S_n) = 0 \text{ for all } 0 \leq k \leq 3 \quad (56)$$

It must also be defined that the four derivatives are continuous between the waypoints: [5]

$$p_{i-1}^{(k)}(S_i) = p_i^{(k)}(S_i) \quad \text{for all } i = 1 \dots n \text{ and } k = 1 \dots 7 \quad (57)$$

With this, is possible to convert the constraints and unknown coefficients α_{ij}, into an equation matrix build based by the restrictions and the b matrix by the waypoints: [5]

$$A\alpha = b \quad (58)$$

Where α contain all the unknown coefficients, and A and b represent all constraints.

2. Gain tuning - Ziegler–Nichols method

Since there are twelve gains to tune, half for the position, the other half for the attitude. A trial and error method approach becomes inefficient. Thus, urges the need to define a more systematic approach to gain tuning, the Ziegler-Nichols method. [4][5]

The Ziegler-Nichols method is used to tune Proportional, Proportional Integrative, Proportional Derivative and Proportional Integrative and Derivative controllers. On this case, it will be applied for a Proportional Derivative control.

This method has a set of step that should be followed:

1. Set the proportional and derivative gains to zero.
2. Increase the proportional gain until the system outputs sustained oscillations that will be denominated as Ultimate Gain, K_u.
3. Measure the period, T_u, described by the oscillations.
4. Define the proportional gain as $K_p = 0{,}8 \times K_u$
5. Define the derivative gain as $K_v = T_u/8$

Sometimes, after this procedure the controller response is too slow. To increase it, it is advised to scale the proportional and derivative gains by small increments until it is obtained a good response. [4] [5]

6. Wind turbine blade modelling

With the model presented above, it is proposed to generate a trajectory, going through waypoints defined based on the dimensions of a

the code, in order to provide a better approximation for the problem. Furthermore, was idealized a trajectory that would incorporate the root and the tip of the blade, enabling the camera to capture it and have a better general idea of his state. The flight proposed is design for a velocity of 0.75 m/s on which would start and rest at the origin of the referential. This flight would take 80 seconds to complete. The Wind Turbine was considered at 10 meters from the initial position of the quadcopter. The model selected to provide the measures of the blades and the tower was the GE 1.5sle, shown of Figure 4. [11]

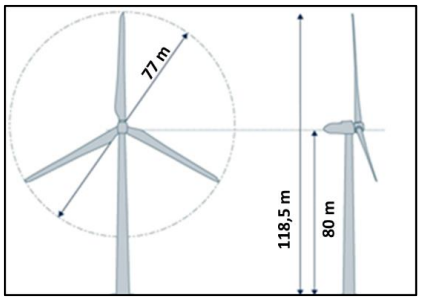

Figure 4: Model used as model to define the waypoints

These waypoints defined were obtain by concrete measurements based on trigonometric rules.

$$Waypoints = \begin{bmatrix} 0 & 0 & 10 & 10 & 10 & 10 & 10 & 10 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 33{,}34 & 0 & -33{,}34 & 0 & 0 & 0 \\ 0 & 30 & 80 & 118{,}5 & 80 & 60{,}75 & 80 & 60{,}75 & 80 & 30 & 0 \end{bmatrix} \quad (59)$$

On equation (59) the first row relates to the X direction, the second row related to the Y direction and the third row related to the Z direction.

1. Results

With the above problem statement, were obtained the results shown below, on which were tuned the gains, in order to improve the state estimation to his best.

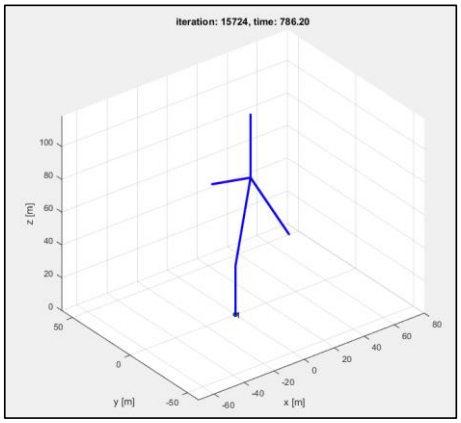

Figure 5: Trajectory described by the quadcopter

On Figure 5 is possible to see the desired trajectory, represented by the red color, and the actual trajectory, represented by the blue color. These described the quadcopter path. A small deviation from the desired state can be seen on Figure 6. Nevertheless, it is possible to conclude that these are good results once that the trajectory incorporated all the blade, from the tip to the root at the desired velocity established.

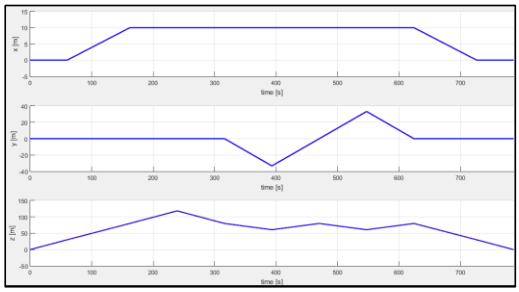

Figure 6: Desired trajectory (red) vs actual trajectory (blue)

On the Figure 6, it is possible to see the deviation from the desired trajectory, red line, from the actual trajectory, blue line. The first row represents the trajectory over the *x* axis, the second row represents the trajectory over the *y* axis and the third one represents the trajectory over the *z* axis. Since the type of polynomial equation used to describe this trajectory was of first order, each waypoint is represented as a corner on the graph.

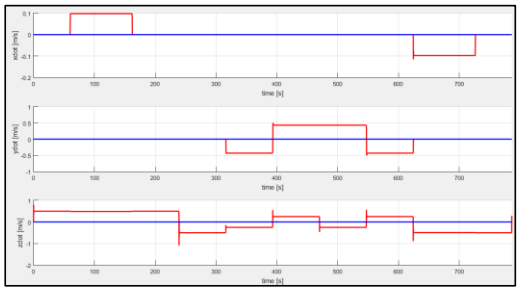

Figure 7: Oscillations of velocity with respect of time.

On the Figure 7 is possible to see the oscillation of velocity produced to move from one to another waypoint defined.

2. Conclusion

A model to define the trajectory of the RPAS to a wind blade maintenance has been proposed in this paper. Therefore, it is possible to conclude that Matlab can provide advantageous tools in order to generate precise trajectory's simulation. Although it is also possible to see that this work has various limitations in terms of providing a direct application for a software once that, as

velocity between waypoints of the Wind blades. This should be slower in order to capture pictures with more quality, decreasing the error associated with the inspection. The trajectory should also be more complex, in terms that it should include the two sides of the blade, leading edge and trailing edge, instead of straight line between the tip and the root. The gains should also be more stable when changing from the straight lines. The rates of angular and linear velocity be upgraded perhaps with toolboxes from Matlab, like PD tuner.

Even though it was generated a spline defined by a 7^{th} order polynomial, the polynomial used to describe these waypoints was from first order, once that instead of a smooth trajectory it is preferred to maintain an accurate path that allows the quadcopter to pass by the middle of the blades, enabling the user to obtain more detailed information.

As it was mentioned, by having a proportional controller, the system has the capability to stabilize itself when its values oscillate from the desired ones, being this a capacitance response. The derivative gain helps to predict the systems behavior, being this a resistance response. Upgrading to a PID controller would make the system more accurate, by stabilizing the system magnitude and time error. This is possible due to the associated with previous oscillations.

Moreover, the velocity between waypoints should be weighted. The trajectory to the approach of the Wind Generator can be done with a bit higher velocity once the only goal on this segment, is to approach to the object being inspected.

Although being a very limited method, it is known that current inspections solutions are either outdated, expensive or inadequate. Therefore, embracing automation using RPAS is a way to improve the renewable energies, equilibrating the competition with fossil fuels.

Bibliography

[1] S. Gupte, "A Survey of Quadrotor Unmanned Aerial Vehicles," IEEE, Orlando, USA, 2012.

[2] G. Morgenthal, "Quality Assessment of Unmanned Aerial Vehicle (UAV) Based Visual Inspection of Structures," Institute of Structural Engineering, Weimar, Germany, 2016.

[3] D. Mellinger, "Trajectory Generation and Control for Quadrotors," University of Pennsylvania, Pennsylvania, 2012.

[4] N. B. N. J.G. ZIEGLER, "Optimum Settings for Automatic Controllers," ASME, New York, 1942.

[5] V. Kumar, "Robotics: Aerial Robotics - Lecture Notes," University of Pensilvania, 2014.

[6] "SkyProff," [Online]. Available: http://skyproff.com/wind-turbine-operation-and-maintenance-services/. [Accessed 9 9 2017].

[7] G. A. G. D. P. A. C. A. I. Marius Kovacs, "Research on the potentiality of using aerial vehicles for monitoring the environment agent - air," ECOTERRA - Journal of Environmental Research and Protection, Romania, 2016.

[8] R. D. Markus Hehn, "Quadrocopter Trajectoy Generation and Control," in The International Federation of Automatic Control, Milano, 2011.

[9] S. L. J. F. S. W. O.O. Esu, "Feasibility of a fully autonomous wireless monitoring system for a," ELSEVIER, Leicestershire, 2016.

[10] Abhijit Das, "Backstepping Approach for Controlling a Quadrotor Using Lagrange Form Dynamics," Texas, 2009.

[11] http://www.aweo.org/windmodels.html

[12] "SPARES IN MOTION," [Online]. Available: https://d3icht40s6fxmd.cloudfront.net/sites/default/files/inspeccion-aerea-profunda-de-palas-informes-y-mapeos-de-alta-resolucion-en-espectro-termico-y-spares_danos_en_punta.jpg. [Accessed 9 9 2017].