Universidad
Rey Juan Carlos

AEROSPACE ENGINEERING IN AIR NAVIGATION

Academic Course 2015/2016

Final Degree Project

Building of an UAV: from the hardware to the driver and autonomous applications

Author: Jorge Cano Martínez

Tutor: Jose María Cañas

Declaration of Authorship

I, Jorge Cano Martínez, declare that this end of dregree project titled, 'Autonomus behaviour of Unmanned Air Vehicless' and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed: _____

Date: _____

Acknowledgements

I would like to express thanks to my advisor, Jose María Cañas, not only for giving me the opportunity to work with him in so interesting field, but also for his hard working, kindness, patience, dedication and wisdom.

I would like to thank all the JdeRobot mates and the people who have collaborated in that project, specially to Alberto Martín and Aitor Martínez.

Of course, I thank all my family, for their support, sacrifice and affection in every aspect of my live, giving me all I need to be completely happy and that have made me the person I am today.

Finally, I absolutely must thank my girlfriend, Ana, her love, understanding and compromise has given me forces for the goals achieved. Without her this project would has never been possible.

I feel indebt with all of you. Thank you so much.

Jorge Cano Martínez

Abstract

The first “autonomous” aerial vehicles born on the middle 40’s, where the Nazis developed the first ballistic missile capable to reach its objective correcting its trajectory using its own sensors. The role of this type of technology on the military world has been crucial since that moment, and countries have spent tons of resources on the development of UAV’s (Unmanned aerial vehicle). Although the industry of UAV’s appeared many years ago, it has been getting popularity between the general public as “Drones”.

This project intents to be part on that concept, it aims to reach a fully autonomous behavior of a UAS (Unmanned Aerial System) in order to fulfill a certain mission robustly and efficiently. The design, construction and programming of an UAV from scrach is the main objective, undertanding these vehicles indeed from the conception to the validation.

The solution proposed in this project aims to provide a reliable platform and sofware support for all MAVLink protocol based UAV’s, very popular nowadays mainly in research projects.

The chapters 1, 2 and 3 of this document are focused on the stablishment of the enviroment of the project, compound of an introduction, the working methodology followed and the infrastructure used. The chapters 4,5 and 6 are the description of the different achieved tasks: design and construction of a hardware platform, the implementation of a sofware driver to communicate with the vehicle and the programming of an application for the fulfilment of the mission. Finally, chapter 7 collects the achievements reached on the development of the project and the future steps recommended that would use this project as a benchmark.

There is a long way to go with the Drones, and lot of aspects to develop, but they will become extremely important to our lives, more than we today imagine.

Contents

List of Figures

List of Tables

Abbreviations

ALBA	**A**vion **L**igero **B**lanco **A**ereo
ALO	**A**vion **L**igero de **O**bservación
API	**A**pplication **P**rogramming **I**nterface
APM	**A**rdu**P**ilot**M**ega
ARCAS	**A**erial **R**obotics **C**ooperative **A**ssembly **S**ystem
CACGS	**C**omputer **A**ssisted **C**arrier **G**uidance **S**ystem
CNC	**C**ontrol **N**umeric **C**utter
CSP	**C**entral **S**piral **P**oint
CTOL	**C**onventional **T**ake-**O**ff and **L**anding
DIY	**D**o **I**t-**Y**ourself
DOF	**D**egrees **O**f **F**reedom
ESC	**E**lectronic **S**peed **C**ontroller
FMA	**F**lying **M**achine **A**rena
GCS	**G**roup **C**omunication **S**ystem
GUI	**G**raphical **U**ser **I**nterface
IARC	**I**nternational **A**erial **R**obotic **C**ompetition
ICE	**I**nternet **C**omunications **E**ngine
ICS	**I**ntel **C**ompute **S**tick
IMU	**I**nertial **M**easurement **U**nit
INTA	**I**nstituto **N**acional de **T**écnicas **A**eroespaciales
LiPo	**Li**thium **Po**limer
LSB	**L**east **S**ignificant **B**it
MAV	**M**icro **A**erial **V**ehicle
MSB	**M**ost **S**ignificant **B**it
MTOW	**M**aximum **T**ake-**O**ff **W**eight

NED	**N**orth **E**ast **D**own
PDB	**P**ower **D**istribution **B**oard
PEO	**P**oli**e**thilene **O**xide
PWM	**P**ulse **W**idth **M**odulation
RC	**R**adio **C**ontrol
RPAS	**R**emotely **P**iloted **A**ircraft **S**ystem
RPAV	**R**emotely **P**iloted **A**erial **V**ehicle
SIVA	**S**istema **I**ntegrado de **V**igilancia **A**erea
TOW	**T**ake-**O**ff **W**eight
SPE	**S**olid **P**olimer **E**lectrolite
SLAM	**S**imultaneous **L**ocaton **A**nd **M**apping
SUAS	**S**mall **U**nmanned **A**ircraft **S**ystems
UAS	**U**nmanned **A**ircraft **S**ystems
UAV	**U**nmaned **A**ir **V**ehicle
UAVS	**U**nmanned-**A**ircraft **V**ehicle **S**ystem
UDP	**U**ser **D**atagram **P**rotocol
URJC	**U**niversidad **R**ey **J**uan **C**arlos
VTOL	**V**ertical **T**ake-**O**ff & **L**anding

Chapter 1

Introduction

This chapter outlines the robotics concept and a brief history, from its beginnings until nowadays. Finally, the development of the aerial robots and the research, commercial and defence use of this type of technology are explained.

1.1 Robotics

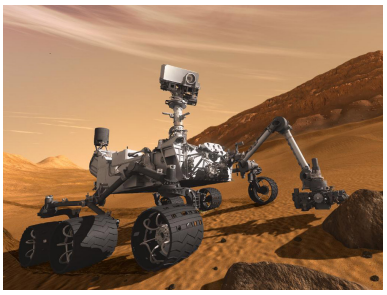
Robotics is a branch of engineering that involves the conception, design, manufacture, and operation of robots, as well as the different systems for their control and information processing. This field overlaps with mechanical engineering, electronics, computer science, artificial intelligence, nanotechnology and lately bioengineering.

A robot is an artificial, mechanical or virtual, entity able to complete an activity autonomously, established beforehand, that substitutes humans in dangerous, difficult or special tasks. The word "robot" derives from the Czech world "robota" that means forced labour and it was firstly used in the Karel Capek novel named Rossum's Universal Robots in 1923.

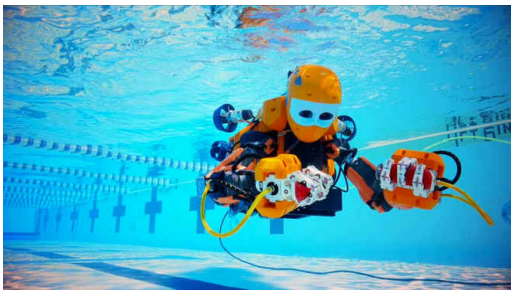
Since 2000 until today, robotics has starred huge breakthroughs compared to the advantages achieved in the 20^{th} century, mainly because of the advancement of technology and the reduction of it cost. Robotics has been able to conquer land, sea, and air in our planet and in other ones. The figure 1.1 shows some examples of last technology robots, as autonomous cars, highly sophisticated humanoids, space rovers, underwater robots, etc...

(a) T8 hexapod

(b) Asimo



(c) Curiosity



(d) Robo-Mermaid

FIGURE 1.1: Mobile Robots

One of the greatest impacts of robotics on everyday life is the Roomba vacuum cleaner robot of the iRobot company, which can be seen in the figure 1.2. It was first launched to market in 2002 and now has a family of robots capable of mopping the floor, cleaning pools, polishing floors or cleaning gutters. Its last version is probably one of the most sophisticated robots affordable by general population.

FIGURE 1.2: Roomba Robot

Camera sensor has adquired a great impact due mainly to its low cost (compared to other sensors) and the large amount of information that can be extracted from it. In the figure 1.3 Nao robots playing a football game can be seen. They use SLAM (Simultaneous

Location and Mapping) technique of artificial vision to locate themselves on the field, their teammates, their opponents, the ball, field boundaries and the opposing goal.

FIGURE 1.3: Nao robots autopositioning based on vision

Also robotic cars make use of artificial vision algorithm to navigate throught the traffic, these robots are able to detect the lines of the road, and identify other cars arround them. In the figute 1.4 the Tesla Model 3 is shown; it is the last advance of a long research line. Google, other serious pioneer of autonomous driving, has based its research on very complex laser-based radar calle "lidar". Tesla competes with a different approach, it has instead opted for cheap hardware sensors: cameras, ultrasound and advanced software.

(a) Tesla Model 3 (b) Road lines recognition

FIGURE 1.4: Artificial vision in autonomous cars

In the industrial environment robotics has reached a level of efficiency and precision never before known in the world. From automated assembly lines with robotic arms for automobile assembly, robots for packaging tasks, classifier robots and many others, make up a growing robotic industrial sector. From the first automatons to the last robots sent to Mars, robotics has helped man to extend and enhance human qualities. The human

being has delegated robots performing the most dangerous tasks, most repetitive ones or those that require a high level of accuracy to improve our current way-of-living.

1.1.1 Robot Hardware

From an engineering point of view, a robot is a complex system equipped with sensors, actuators and processors; combined, give basic skills such as perception, action, processing and memory to interact with the environment. In a simpler way, a robot is a computer with external peripherals that allow it to obtain information about the environment and to interact with it.

- The sensors allow the robot to get information from the environment or from itself. This capability permits a robot to work on dynamic scenarios by changing its behaviour depending on the environment conditions. There are a variety of sensors, from the most basic ones as light sensors, temperature or proximity; to the most complex and rich as cameras, depth sensors, lasers, GPS locators, etc.
- Actuators are devices that allow the robot to interact with the environment and/or to perform movements. They allow the robot to perform its tasks in a real environments, such as navigation avoiding obstacles or assembling parts in an assembly line. The actuators can be of various types such as motors used for moving the wheels of the robot, the motors of articulations or engines spinning propellers to provide lift.

Computers are responsible for processing the data collected by the sensors and materialising the results of the decision algorithms in actions carried out by actuators. Processors give the robot reasoning ability that allows the treatment of large amount of data from the sensors and the ability to execute complex algorithms in a short period of time.

1.1.2 Robot Software

Software is organised information in the form of operating systems, utilities, programs, and applications that enable computers to work. The software is a very important element because without it, a robot would be a machine incapable of generating intelligent behaviour. Like any other software, software for robots must meet certain requirements:

- Adaptability: a robot is situated in a dynamic and unpredictable scenario and in constant interaction with the environment. Because of this feature the software must be flexible to provide reliable answers fast and continuously.

- Multitasking: the nature of the software for robots must be multitasking, having the ability to collect sensor data, to process them, to decide and to take action simultaneously.

- Distributed: sensors with increasingly more complex data and larger computational load is difficult to be handled by a single component, that is why distributed computing has almost become a standard in the development of robotic applications. In addition, coordinating multiple robots or use of other data sources external to the robot itself, such as systems cameras to detect 3D robot in a controlled environment, often is necessary.

- Dealing with heterogeneous hardware: often same type of sensor or actuator device would require special software characteristics depending on its manufacturer. So it is necessary the existence of an homogeneous software interface to unify access to sensors and actuators regardless of manufacturer or model of the device. This allows development of algorithms for a given robot, that can be exported to a different one, enabling code reuse.

To help the developer responsible for making applications for robots, there are several software platforms available (open or closed source code). These platforms typically are installed on an operating system. They are the bridge between the robot and the developer, and takes advantage of the characteristics of modern operating systems such as hardware abstraction, multitasking capabilities and ease to create distributed applications with external libraries. Usually, if the robot model is widespread, the developer would also have simulators available where code could be tested in a controlled environment.

1.2 Aerial Robotics - Unmanned Aerial Vehicles

Aerial Robotics is the branch of robotics that deals with the study of the behaviour of autonomous unmanned aerial vehicles (UAV's). An UAV is a powered aerial vehicle

that does not carry any human operator, uses aerodynamic forces to provide lift, can fly autonomously or be piloted remotely, can be expendable or recoverable, and can carry a lethal or nonlethal payload. Therefore, missiles are not considered UAVs because the vehicle itself is a weapon that is not reused, though it is also unmanned and in some cases remotely guided. The flight of UAV's may operate with various degrees of autonomy: either under remote control by a human operator, or fully or intermittently autonomous, by on-board computers.

The term "*drone*", more widely used by the public, has encountered strong opposition from aviation professionals and government regulators, which make use of other terms:*unmanned aircraft system* (UAS), *unmanned-aircraft vehicle system* (UAVS) *remotely piloted aerial vehicle* (RPAV) and *remotely piloted aircraft system* (RPAS).

In recent years, a new scientific and technological challenge in this area has been to get these vehicles completely autonomous, that is, they can fly without driving, neither supervision of a person.

UAV's typically fall into one of six functional categories. However, there are differnt classifications regarding other aspects, although multi-role airframe platforms are becoming more prevalent:

- Target and decoy: providing ground and aerial gunnery a target, that simulates an enemy aircraft or missile.
- Reconnaissance: providing battlefield intelligence.
- Combat: providing attack capability for high-risk missions.
- Logistics: cargo delivering.
- Research and development: improve UAV technologies.
- Civil and commercial UAVs: agriculture, aerial photography, data collection.

As technology advances and costs are lowered, new sensors have been integrated into these vehicles such as information adquisition (cameras, lasers...), batteries with greater autonomy, most powerful computing systems, etc.. The cheapening of UAVs technology has produced an increasing use of it in research centers throughout the world to have

these vehicles and getting achievements that were unthinkable a few years ago. This fact is increasing the scope of use of UAV's. They are used not only in military environments, but for forest services, agriculture, environment, hydrology, cartography, natural disasters or geology.

1.2.1 Applications

UAV's have been traditionally used for military purposes, it is the field where they are more extended. Among other governments and organisations, in Spain we have INTA (Instituto Nacional de Técnica Aeroespacial) [1]. For years INTA has been working in a research program to develop the necessary technologies to design and build a range of unmanned aircrafts. As a result of these activities, the institute has developed the following products:

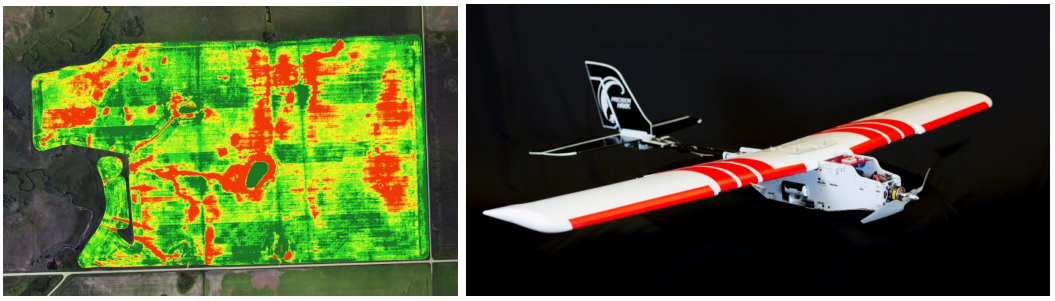
- SIVA: A sophisticated unmanned aerial surveillance multiple applications vehicle for the civil and military field, which can be used as a vehicle for real-time observation.
- ALO: A low cost observation and high reliability system suitable for the acquisition of aerial images in civilian and military short range missions.
- ALBA: A complete system of aerial guided target suitable for improving the operation of anti-aircraft artillery units through its training under real fire.
- MILANO [2]: An strategic system for monitoring and observing composed of UAV's linked via satellite with a ground control station. The station plans monitors and controls both the aircraft and the shipped payloads. The aircraft has an endurance of more than 20 hours and can operate at altitudes up to 26,000 feet.
- DIANA [3]: A system of high-performance aerial target developed to simulate real threats, It can reach 200m/s and has a flight mode call "wave-skim" that allows the vehicle to fly under 15m height in the sea. Because of its versatility, the system can be used as air training system for large numbers of current and future weapons. The INTA DIANA 1A design is represented in the figure 1.5

FIGURE 1.5: Diana UAV 1A

A commercial and popular prototype is Google's Project Wing, the code name for the company's delivery service drone. It hopes to be launched in 2017. The company first announced Project Wing in a YouTube video back in 2014, but since then, Google has been keept in secret how this service will work. Now a new patent filing from the company reveals how part of Project Wing deliveries packages. Delivery phase of the wing is shown in the image 1.6.

FIGURE 1.6: Googlewing Wing

A wide extended use of UAV's is the precision agriculture, which consists on the adquisition of information of the terrain for the optimizing the resources expent on the labor (figure 1.7 a). There are a wide variety of information sources useful for this sector, as water resources, vegetation analysis, 3D mapping or desease detection. This advance on agriculture would saves a great amount of resources and would help also to the sustainability of the agriculture; optimizing the use of pesticides and water. There exist a wide variety of enterprises and projects dedicated to precision agriculure. One great example of it is PrecisionHawk shown in the figure 1.7 b.



(a) Terrain analysis performed by UAV's (b) precisionhawk

FIGURE 1.7: Precision agriculture

However, and despite the great progress made in recent years, there are still major obstacles to overcome. Not only technological, but ethical and regulatory developments for security reasons. One of the most promissing areas for UAV testing and development are the competitions for UAV's, that have been growing worldwide. These competitions, where teams have to compete in creating an application to overcome a particular challenge, are thought to encourage research into these technology.

The UAV Outback Challenge [4] is an annual competition for the development of unmanned aerial vehicles. The competition was first held in 2007 and features an open challenge for adults, and a high-school challenge. The event is aimed at promoting the civilian use of unmanned aerial vehicles and the development of low-cost systems that could be used for search and rescue missions. The event is one of the largest robotics and UAV challenges in the world, with $50,000 on offer to the winner of the Open segment of the Challenge. The events involve a thorough scoring system with an emphasis on safety, capability and technical excellence. In particular there is a strong trend towards autonomous flight. Notably, teams share technical details of their entries, allowing successful innovations to proliferate and increasing the speed of technological development.

The format of the Challenge changes as technological improvements make the tasks more achievable. In 2011 it changed to a search and rescue challenge. No teams successfully completed the challenge until 2014, when several teams were successful. In 2016 the open challenge will change to an automated medical retrieval task.

IARC (International Aerial Robotic Competition) [5] is an aerial robotics university competition, founded in 1991 at the Georgia Institute of Technolog. The main purpose of the competition is to advance in aerial robotics behavior. From 1991 through 2013

they have proposed a total of 7 missions. Each of them is to develop a fully autonomous behaviours that is able to pass the test. New missions are created every time that the previous is overcome, so, a mission may be years unaddressed until one team achieves the goal.

1.2.2 Research

ETH Zürich University, the Institute for Dynamic Systems and Control, is possibly the most important research centre in quadcopters. Its facilities include the Flying Machine Arena (FMA) [6] where a group of researchers conducted their experiments on the control of UAV's. This laboratory has a high precision motion capture system, a wireless network and software developed by themselves that, using very sophisticated algorithms, allows estimation and control of UAV's. The motion capture system is able to locate multiple objects at a rate that exceeds 200 frame per second. UAVs in space system can move at a speed of 10 m/s, which is about 5 centimetres between two successive catches. The information in this system intersects with other data and dynamic system models to predict the state of an object in the future. The system uses this data to determine the following command, which must be executed in the vehicle for a certain movement; and then, the wireless network transmits it to the vehicle performing the action.

FMA group carries on several research projects, such as:

- "Quadcopter Pole Acrobatics" [7] system that allows quadcopters to balance an inverted pendulum, to throw it into the air, and catch and balance it again on a second vehicle. Based on models, a launch condition for the pole is derived and used to design an optimal trajectory to throw the pole towards a second quadcopter. An optimal catching instant is stimated and the corresponding position is predicted using the current position and velocity. An algorithm generates a trajectory for moving the catching vehicle to the predicted catching point in real time (instant shown of the figure 1.8). By evaluating the pole state after the impact, an adaptation strategy adapts the catch maneuver such that the pole rotates into the upright equilibrium by itself.

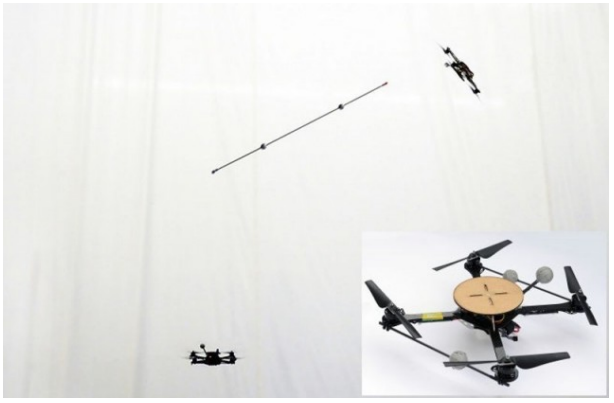


FIGURE 1.8: Poleacro quadcopter

- "Omni-Directional Aerial Vehicle" [8] design and control of a novel six degrees-of-freedom aerial vehicle. Based on a static force and torque analysis for generic actuator configurations, it derives an eight-rotor configuration that maximises the vehicle's agility in any direction. The proposed vehicle design possesses full force and torque authority in all three dimensions. A control strategy that allows for exploiting the vehicle's decoupled translational and rotational dynamics is introduced. A prototype of the proposed vehicle design is built using reversible motor-propeller actuators, making it capable of flying at any orientation (figure 1.9).

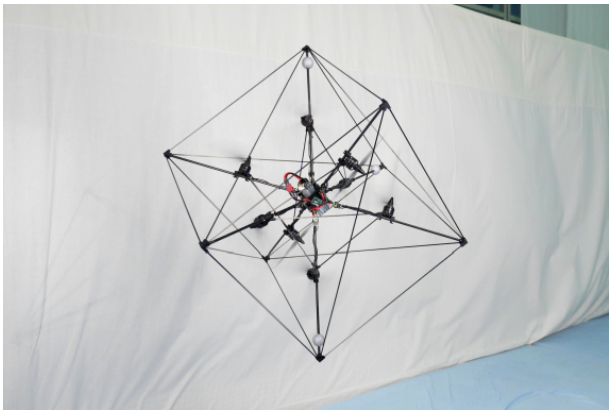


FIGURE 1.9: Omni-Directional Aerial Vehicle

- "Tailsitter" an hybrid vehicle that takes off vertically like a multirotor but is also able to fly horizontally like a fixed-wing airplane. A new algorithm for robustly controlling a tailsitter flying machine in hover position has been developed. Using the algorithm, the tailsitter is able to recover from any orientation, including upside down.

- "Distributed Flight Array" is a flying platform consisting of multiple autonomous single propeller vehicles that are able to drive, dock with their peers, and fly in a coordinated mode. Once in flight the array hovers for a few minutes, then falls back to the ground, only to repeat the cycle again.

The SHERPA project [9], found by European Union, aims to develop a mixed ground and aerial robotic platform to support search and rescue activities in a real-world hostile environment like the alpine scenario. The prototype design can be seen in the figure 1.10.

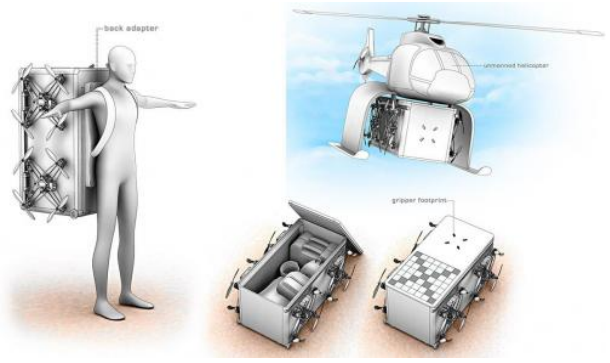


FIGURE 1.10: Sherpa Box

The technological platform and the alpine rescuing mission are the scenario to address a number of research topics about cognition and control pertinent to the call. What makes the project potentially very rich from a scientific viewpoint is the heterogeneity and the capabilities to be owned by the different actors of the SHERPA system. They all interact and collaborate with each other, with their own features and capabilities, toward the achievement of a common goal.

The ARCAS (Aerial Robotics Cooperative Assembly System) [10] project proposes the development and experimental validation of the first cooperative free-flying robot system for assembly and structure construction. The project will pave the way for a large number of applications, including the building of platforms for evacuation of people or landing aircraft, the inspection and maintenance of facilities and the construction of structures in inaccessible sites and in the space.

ARCAS is funded by the European Union and involves two Spanish universities, "Universidad de Sevilla" and the "Universidad Politécnica de Cataluña".

1.3 Multi-Rotor Aircraft

The UAVs type of VTOL (Vertical Take-Off and Landing) are vehicles with the ability to hover, vertical take-off and landing. These vehicles are able to maintain the hover height and direction, thus remain static at a point at a certain height. A clear example of VTOL vehicle type are helicopters, although there are aircrafts with VTOL capabilities as the Harrier or the Boeing V-22 Osprey, shown in the figure 1.11.

FIGURE 1.11: Boeing V-22 Osprey

A multirotor or multicopter is a VTOL rotorcraft with more than two rotors. An advantage of multirotor aircraft is the simpler rotor mechanics required for flight control. Unlike single-rotor and double-rotor helicopters, which use complex variable pitch rotors, whose pitch vary as the blade rotates for flight stability and control, multirotors often use fixed-pitch blades. Control of vehicle motion is achieved by varying the relative speed of each rotor to change the thrust and torque produced by each.

Due to their ease of both construction and control, multirotor aircrafts are frequently used in radio controlled aircrafts and UAV projects in which the names tricopter, quadcopter, hexacopter and octocopter are frequently used to refer to 3, 4, 6 and 8 rotor vehicle, respectively.

In order to allow more power, more stability and reduced weight coaxial rotors can be employed, in which each arm has two motors, running in opposite directions (one facing up and one facing down).

1.3.1 Flight Principles

As airplanes or helicopters, multirotors can fly due to movement of a wing (rotor blades) through the air. In this case the blades describe a circular movement unlike planes that need horizontal movement of the vehicle to generate lift. Because the air passes through a wing, a pressure differential is produced. The pressure in the upper surface (extrados) is less than the pressure at the bottom surface (intrados). This pressure difference results in the lift force.

When in an aircraft the lift is greater than the weight, it begins to fly. However, the ability to generate lift is not the only problem facing the VTOL vehicles. If we have a helicopter with a single motor, with the ability to generate sufficient lift to raise the vehicle off the ground, the vehicle would turn around the motor shaft axis in the opposite direction to the blades rotation. This is because the aerodynamic force resultant of the propeller has a horizontal component opposite the sense of rotation of the blades. This phenomenon induces a momentum parallel to propeller rotor axis. For maintaining a stable flight, this momentum has to be compensated; in a conventional helicopter, this is done by adding a tail rotor, that spinning perpendicular to the main one, compensating that momentum. The figure 1.12 shows the forces and momentums involved.

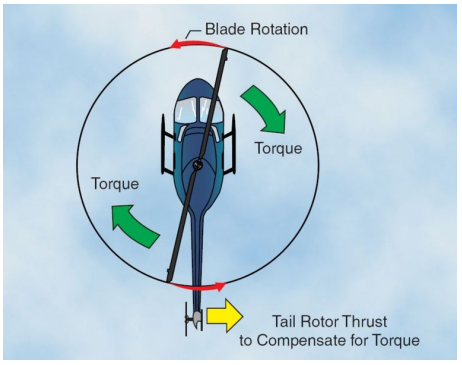
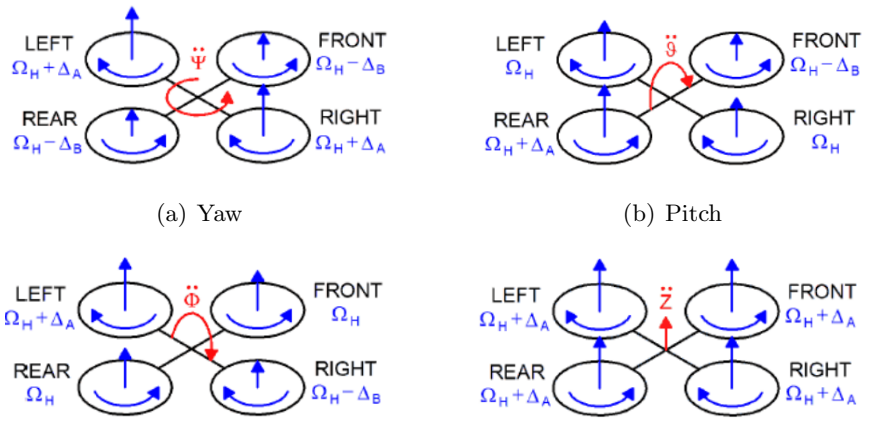


FIGURE 1.12: Torque compensation

Multirotors take advantage of these momentum generated by its rotor to stabilise itself and manoeuver. These vehicles typically use pairs of rotor propellers to compensate each other momentum and varies their spinning velocity to achieve and control the desired momentum. The simplest way to explain this effect is with a quadcopter (4 rotors vehicle).

With this configuration, based on the typical aeronautic body reference frame (x-front, y-right, z-down), if the rotors rotating in the counter-clockwise are fed with more power, the vehicle would turn to the right. This shift in the horizontal plane, which makes the vehicle turning on itself, is called yaw.

In the case of extend the power to the rotors in one end, for example the rear, the quadcopter would shift causing the vehicle turn head towards the floor because the raise of lift vector in that pair of rotors. Notice that the momentum in each rotor is also increased, but as they do it the same amount and have opposite directions, the rise in yaw momentum is compensated. This movement is known as pitch, and produces the horizontal linear movement of the aircraft due to the inclination of the lift vector. The same effect is applied in different pairs of rotors to produce the lateral inclination movement, or roll, and its corresponding lateral movement. The figure 1.13 represents examples of different states of the rotors to generate desire movements.



(a) Yaw (b) Pitch

(c) Roll (d) Throttle

FIGURE 1.13: Rotors configurations

With this setup, the vehicle reaches the 6 DOF [11](figure 1.14) for a complete control of the flight: linear velocities (x,y,z) and angular velocities (pitch, roll and yaw).

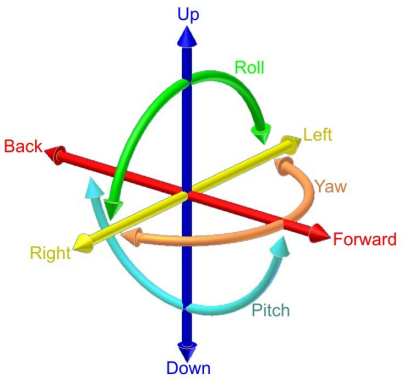


FIGURE 1.14: Roll, Pitch and Yaw

1.4 UAV Background at the URJC Robotis Laboratory

In 2013, within the robotics laboratory at the Rey Juan Carlos University. a new line of research on UAV's was launched, which aims to develop new applications in this sector implementing computer vision techniques to control vehicles and perform autonomous complex missions. The goal is that applications developed in these projects are easily exported to other vehicles. Thus, if a member of the group is developing a navigation algorithm for a certain vehicle, the same algorithm can be used for other similar ones.

Currently several projects have been developed in the URJC robotics laboratory, such as vision, self-localization and navigation, which are the direct antecedents of this project. Numerous robotic components have been designed as drivers, algorithm applications and support for behavioral simulations for robots as NAO, ArDrone, FX-61 Phantom, Pioneer and Kobuki, among others. The work of Alberto Martin Florido and Daniel Yagüe are highlighted

The project of Alberto Martin Florido involves the creation of `ardrons_server` component, which gave JdeRobot support for Parrot ArDrone version 1.0 , allowing access to its sensors and actuators. This work also includes a tool for remotely controlling the vehicle and reading information from its sensors through a graphic interface, the `uav_viewer` component. In addition, an algorithm for vision and autonomous navigation has been developed within the component. This algorithm allows the quadcopter to track objects in three dimensions using as source of information from the front and ventral cameras.

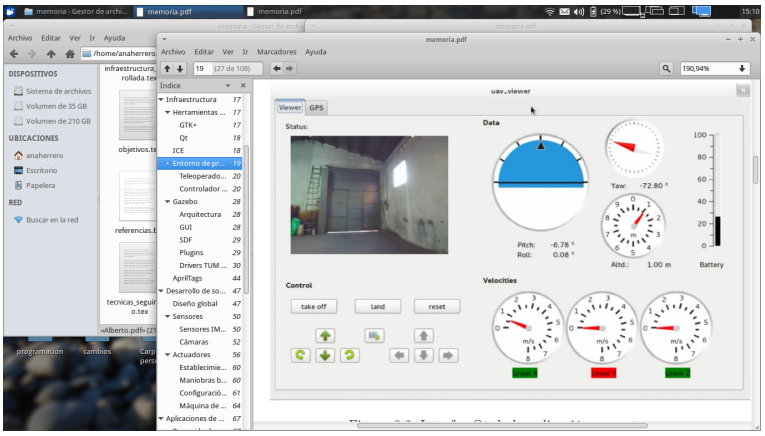


FIGURE 1.15: uav_viewer component intercace

Daniel Yagüe's project aim is to provide support for aerial robots in the JdeRobot environment within the Gazebo simulator, so that there is a realistic response of the vehicle in order for future people to program autonomous behaviour for any application. Along with this support, various navigation applications were designed and validated experimentally; for example, beacon tracking applications using the position information, road tracking and following through the ventral camera, locating objects in a determined area and tracking of other aerial vehicle.

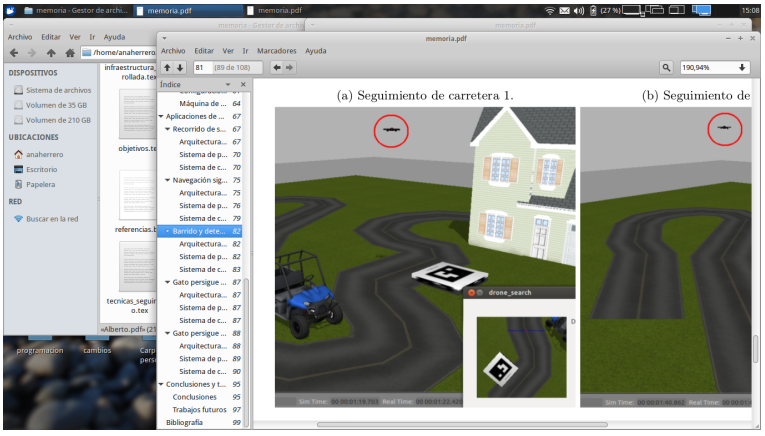


FIGURE 1.16: Road tracking Daniel Yagüe´s project

Chapter 2

Objectives

In this chapter the purpose, goals and requirements of this project described, as well as the methodology and work plan to achieve them successfully.

2.1 Project Objective

The general aim of the project is to design and build an UAV vehicle, provide it with autonomous intelligence and make it capable to fulfil an outdoor mission, in the most robust and efficient possible way. For meeting the objective, the problem has been divided into three simpler subgoals:

- Design and construction of an open-source aerial hardware platform able to offer to this project and future ones a reliable and effective vehicle for a long research.
- Program the software driver to provide the vehicle connection with JdeRobot [12] infrastructure and supply all the communication channels and information offered by the vehicle sensors and motors.
- Plan and execute an example autonomous mission, amenable to be used in real problems.

2.2 Requirements

The requirements of the project have to be defined in order to ensure its consistency, its correct development and to avoid losing focus in any step of the project.

- The developed infrastructure should be integrated in JdeRobot platform.
- The infrastructure would perform its mission having concern of its level of abstraction, interfering as less as possible with higher or lower levels of abstraction, as the vehicle stabilisation process for example.
- The interfaces between components would make use, as much as possible, of the existing ones; defined in JdeRobot and Autopilot source code.
- All the supplied and resulting code should be open-source, for future projects to take advantages from it.
- The communication channels should stream with low latency and be reliable.
- The infrastructure should make an efficient use of the computational resources.

2.3 Working Methodology

Given the nature of the project, and like any other engineering project, using a model that defines the life cycle of the application is necessary. For the development of this project we have decided to adopt the spiral development model. This model was defined by Barry Boehm in 1986 and it is based on a spiral in which each iteration represents a set of activities. Eeach iteration is divided into the following activities:

- Determine targets: this activity is performed in order to define the current iteration. Following this model the final goal of the project is divided into sub-goals, which are stablished in section 2.1.
- Risk Analysis: This activity is carried out by several studies in order to meet potential threats or unwanted events that may occur in the current target.
- Development and test: at this point verify the correct operation performed in iteration to correct the errors and that they do not reach in the following iterations.

- Planning: In this activity the previous phases will be reviewed to determine whether they should continue with future activities.

FIGURE 2.1: Spiral development model

To perform these activities and iterations we held weekly meetings throughout the development work. Thus, every week a new subgoal was established. If the previous had been completed, we then planned the next. If they had not been completed, made the current target deeper to correct errors or re-plan it. During the course of the project a web logbook [13] was fed and the different goals of the project where published. In addition, all the source code generated is kept in a repository with version control system (GitHub [14]) accessible from the Internet. Thus the tutor and any other interested person has access at any time to the code.

2.4 Work Plan

The execution phase of the project has followed this plan:

1. Design and construction of the vehicle platform

   a) Conception: goals evaluation, design requirements establishment for the correct fulfilment of the achievements and platform type choice between all the available ones (fix wing, helicopter, multirotor, etc...).

b) Design: market study, preliminary design and final components choice regarding established requirements, resulting on a theoretical platform design with estimated specifications

c) Integration: construction of the platform itself and connection of all the component for making them work properly. Also integrating the components in an space efficient way.

d) Initial configuration: auto pilot initialisation and configuration for our platform in concrete.

e) Flight test: verification of the correct performance of the platform as a system.

2. Development of the Driver software

a) Learn software background: JdeRobot infrastructure, robotics, computer vision and communication protocol; theory necessary for the correct development of the driver.

b) Develop the driver software: programing of the server that supply to the infrastructure all the necessary communication channels, provide information captured by sensors and send commands to the vehicle

c) Test driver software: verify the correct performance of the driver.

3. Client software and Autonomous mission

a) Client software development: software that interprets the data acquired and enables decision making.

b) Mission concept: mission plan, risk analysis and validation

c) Client-mission integration: Program the mission, carry out the decision making and send actuation commands.

d) Final Flight test: verify the correct performance of the whole project.

Chapter 3

Infrastructure

This chapter describes the infrastructure that this project is based on; mainly software which the project uses as starting point. The code developed in this project trusts on the performance of the software following described.

3.1 Ardupilot

ArduPilot [15] (also ArduPilotMega - APM) is an open source UAV's platform, able to control autonomous multicopters, fixed-wing aircraft, traditional helicopters and ground rovers. It was created in 2007 by the DIY Drones community. It is based on the Arduino open-source electronics prototyping platform. The first Ardupilot version was based on a `thermopile`, which relies on determining the location of the horizon relative to the aircraft by measuring the difference in temperature between the sky and the ground. Later, the system was improved to replace `thermopile` with an Inertial Measurement Unit (IMU) using a combination of accelerometers, gyroscopes and magnetometers. Ardupilot is an award winning platform that won the 2012 and 2014 UAV Outback Challenge competitions.

ArduPilot contains a control system on board making use of sensors, it is capable of automatic manoeuvres such as take-off, stabilisation, landing and the possibility to hover at a point at a certain height. This system allows configuration of certain parameters and reception values for the movement of the drone and allows to plug a great variety of sensors to configure the vehicle for the desired purpose.

Today, the ArduPilot project has evolved to a range of hardware and software products, including the APM and Pixhawk/PX4 [16] line of autopilots, and the ArduCopter, ArduPlane and ArduRover software projects.

The free software approach from Ardupilot is similar to that of the PX4/Pixhawk and Paparazzi Project [17], in all of them low cost and their availability enable its use in small remotely piloted aircrafts, such as micro air vehicles and miniature UAV's.

ArduPilot is the flight controller sofware operating in PixHawk board and this project makes use of its main functions to low level control, stabilising the vehicle and performing high level point to point navigation. It also helps the project with sensor drivers and data fusion. The version used is the ArduCopter 3.3.2

Features:

- C++ open source based code.
- High quality auto-level and auto-altitude control.
- Offers both enhanced remote control flight (via a number of intelligent flight modes) and execution of autonomous missions.
- Worldwide spread and tested code with a large community support.
- Multiple sensor options.
- Allows different communication channels (MAVLink protocol).
- Endless options for customisation and expanded mission capabilities.

3.2 APM Mission Planner

Mission Planner version 2.0 [18] is a full-featured ground station application for the ArduPilot open source autopilot project. Developed by Michael Oborne, it is part of the ArduPilot project. It can be used as a configuration utility or as a dynamic control supplement for autonomous vehicles. The mission interface of APM is shown in the figure 3.1.

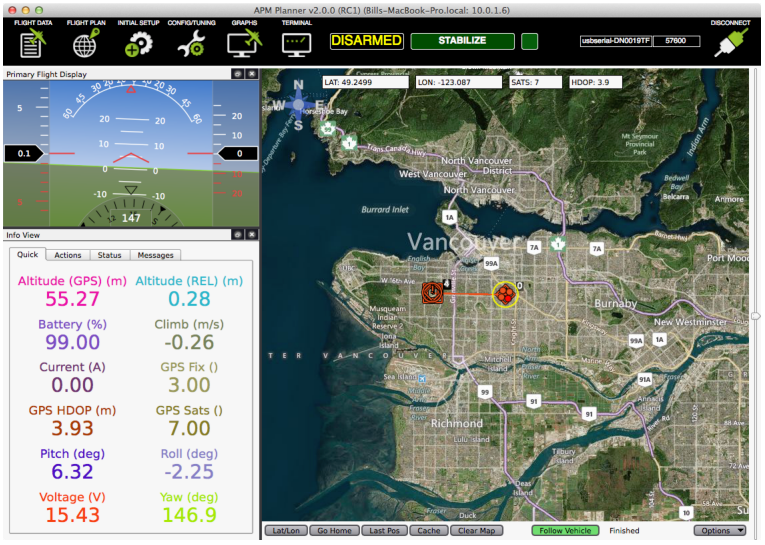


FIGURE 3.1: Mission Planner

The main uses to which APM mission planner is used are:

- Load the firmware (the software) into the autopilot (APM, PX4...) that controls te vehicle.
- Setup, configure, and tune the vehicle for optimum performance.
- Plan, save and load autonomous missions into the autopilot with simple point-and-click way-point entry on Google maps or other.
- Download and analyse mission logs created by the autopilot.
- Interface with a PC flight simulator to create a full hardware-in-the-loop UAV simulator.
- With appropriate telemetry hardware, monitor the vehicle's status while in operation.

It is the platform through which the code and initial configuration of the vehicle are uploaded to the PixHawk.

3.3 MAVLink Protocol

MAVLink (Micro Air Vehicle Communication Protocol) version 1.0 [19] is a very lightweight, header-only message marshalling library for micro air vehicles.

It can pack C-structs over serial channels with high efficiency and send these packets to the ground control station. It has been extensively tested on the PX4, Pixhawk, APM and Parrot AR.Drone platforms and serves the project as a communication backbone for the MCU/IMU communication as well as for Linux interprocess and ground link communication.

MAVLink was redeveloped by Lorenz Meier [20]. MAVLink messages are defined in XML and then converted to C/C++, C# or Python code, every message is identifiable by the ID field on the packet, and the payload contains the data from the message. The general message structure is divided in 8 fields:

- Start-of-frame: Denotes the start of frame transmission
- Pay-load-length: length of payload (n)
- Packet sequence: Each component counts up its sending sequence. Allows to detect packet loss.
- System ID: Identification of the sending system. Allows to differentiate several systems on the same network.
- Component ID: Identification of the sending component. Allows to differentiate several components of the same system
- Message ID: Identification of the message defines what the payload "means" and how it should be correctly decoded.
- Payload: The data into the message, depends on the message id.
- Cyclic redundancy check: Check-sum of the entire packet, excluding the packet start sign (LSB to MSB)

The message detailed fields could be found here [21].

However, this protocol does not limit to a message marshalling library, MAVLink ecosystem encompasses a great variety of programs and applications for the use of this protocol. This is one of the reasons why it has become the most popular protocol between MAV (micro aerial vehicle) developers, many projects are based on this protocol, such as ArduPilot, ETH Flying Machine Arena or Sky-Drones [22] among others.

3.4 MAVProxy

MAVProxy is a fully-functioning GCS (group communication system) for UAV's, version 1.4.38 is the one used in this project. It is a minimalist, portable and extendable GCS for any UAV supporting the MAVLink protocol. It has a number of key features, including the ability to forward messages from UAV's over the network via UDP (User Datagram Protocol) to multiple other ground station software on other devices.

- It is a command-line, console based app. There are plugins included in MAVProxy to provide a basic GUI.
- It is written in Python.
- It is open source.
- It is portable; it should run on any POSIX OS with python, pyserial, and function calls, which means Linux, OS X, Windows, and others.
- It supports loadable modules, and has modules to support console/s, moving maps, joysticks, antenna trackers, etc...

3.5 JdeRobot

JdeRobot [12] is a software development suite for robotics and computer vision applications. These domains include sensors (for instance, cameras), actuators, and intelligent software in between. It acts as robotic sofware enviroment for the development of the components of this project, making use of its interfaces, driver and tools of the version 5.3.2

It has been designed to help in programming such intelligent software. It is mostly written in C++ language and provides a distributed component-based programming environment where the application program is made up of a collection of several concurrent asynchronous components. Components may run in different computers and they are connected using ICE communication middleware. Components may be written in C++, Python, Java... and all of them interoperate through explicit ICE interfaces.

JdeRobot simplifies the access to hardware devices from the control program. Getting sensor measurements is as simple as calling a local function, and ordering motor commands as easy as calling another local function. The platform attaches those calls to remote invocations on the components connected to the sensor or the actuator devices. They can be connected to real sensors and actuators or simulated ones, both locally or remotely using the network. Those functions build the API for the Hardware Abstraction Layer. The robotic application get the sensor readings and order the actuator commands using that API to unfold its behaviour.

Several driver components have been developed to support different physical sensors, actuators and simulators. Currently supported robots and devices:

- RGBD sensors: Kinect and Kinect2 from Microsoft, Asus Xtion.
- Wheeled robots: Kobuki (TurtleBot) from Yujin Robot and Pioneer from MobileRobotics Inc.
- ArDrone quadrotor from Parrot.
- Gazebo simulator.
- Firewire cameras, USB cameras, video files (mpeg, avi...), IP cameras (like Axis).
- Pantilt unit PTU-D46 from Directed Perception Inc.
- Laser Scanners: LMS from SICK and URG from Hokuyo.
- Nao humanoid from Aldebaran.
- EVI PTZ camera from Sony.
- Wiimote.
- X10 home automation devices.

JdeRobot includes several robot programming tools and libraries. First, viewers and teleoperators for several robots, its sensors and motors. Second, a camera calibration component and a tuning tool for colour filters. Third, VisualHFSM tool for programming robot behaviour using hierarchical Finite State Machines. And several more. In addition,

it also provides a library to develop fuzzy controllers, a library for projective geometry and computer vision processing.

Each component may have its own independent Graphical User Interface or none at all. Currently, GTK and Qt libraries are supported, and several examples of OpenGL for 3D graphics with both libraries are included.

JdeRobot is open-source software, licensed as GPL and LGPL. It also uses third-party software like Gazebo simulator, ROS, OpenGL, GTK, Qt, Player, Stage, GSL, OpenCV, PCL, Eigen, Ogre

3.6 ICE

ICE (Internet Communications Engine) [23] is an object-oriented middleware that provides remote procedure calls, grid computing and client / server functionality developed by ZeroC under a dual GNU GPL and a proprietary license. It is available for C ++, Java, .Net languages, Objective-C, Python, PHP and Ruby, in most operating systems. There is also a version for mobile phones called Ice-e. ICE allows to develop distributed applications with minimal effort, abstracting the programmer to interact with low network interfaces. The process of application development should focus only on the logic and not in the peculiarities of the network. It is a multilanguage middleware platform and thus, we can implement clients and servers in different programming languages and on different platforms. ICE works with distributed objects, so that two objects in our application need not be running on the same machine. Objects can be on different machines and communicate across network through the sending of messages between them.

JdeRobot uses ICE for communication between its nodes, therefore, the task of reading values from a sensor or command orders to a robot is as simple as running a method of an object in the application. A significant advantage is the possibility to develop applications independent from context. A programmer may develop a driver in C ++ for a particular robot that is embedded in the robot, on the other hand, another developer can develop an application for image processing in Python that runs on a PC. Through ICE we can use these two applications, which originally were independent, as a single

application without having to worry about low-level communications. With this we can develop modular applications of great complexity without additional effort.

3.7 OpenCV

OpenCV (Open Source Computer Vision Library) [24] is a library of artificial vision developed by Intel. It is under a BSD license, which allows free use for commercial purposes. This means that is commonly used for all kinds of projects, from surveillance systems, motion detection and image processing, as in the case of this project.

It is written in C ++, multiplatform and contains interfaces for C, C++, Java, Python and MATLAB. Having been developed by Intel it provides system integrated performance primitives, which are a set of routines under specific level for Intel (IPP) processors.

Chapter 4

Hardware Platform

In this chapter the systems and subsystems that compose the vehicle platform are going to be described. First, the conception phase, design and integration process. Second, the description and technical specifications of the components and of the complete vehicle.

4.1 Design

Based on the established objectives, the aerial vehicle type selection is the first thing to be done. A quadcopter configuration is the most effective configuration for our purposes, due to its manoeuvrability, ability to hover, simplicity compared with other multirotors or helicopters and availability of pieces offered in the market.

The vehicle we would like to design and build from scratch should try to comply as much as possible with the following features:

- Medium size, between 2-5kg weight and less than 1.5m size
- Open source software
- Reliable
- Several communication channels
- Usable at outdoor scenarios
- Modular design

- Function extendable
- Cost efficient

In the design of a quadcopter, or any multirotor vehicle, the correct choices in propeller design (size and pitch), electric motor (size and power) and power source characteristics (based mainly on the vehicle weight and desired vehicle response) are crucial. These parameters define the effectiveness and efficiency of the vehicle that are directly reflected on manoeuvrability and endurance of the vehicle.

Many facts take part in the vehicle design and there is not an unique way to achieve certain vehicle characteristics. Only trial-failure method, combined with the experience, produces successful results. In engineering, and more in this aspect, theory and reality are parallel lines, not convergent ones. Table 4.1 shows the variables of the configuration to be decided and their estimated effects:

	Efficiency / Flight time	Manoeuvrability / Response	Maximum thrust	Weight	Cost
Propeller size	↑↑	↓↓	↑↑↑	∼	∼
Propeller pitch	↓↓	↑	↑↑	∼	∼
Motor power	↓	↑↑	∼	↑	↑↑
Motor revolutions	↓↓	↑↑↑	↑↑	∼	∼
Battery size	↑↑↑	↓	∼	↑↑↑	↑↑↑
Battery voltage	↓↓	↑↑	↑↑	↑↑↑	↑↑↑

TABLE 4.1: Effects of the configuration variables to be considered

After several sketches, the final vehicle design makes use of the components shown in the figure 4.1:

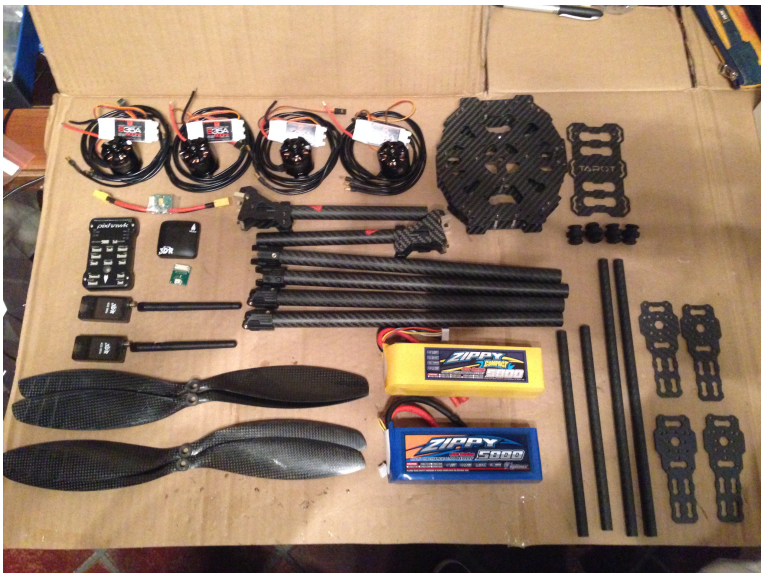


FIGURE 4.1: Pieces and componets of the vehicle

Integration process is the task of placing, connecting and securing the different components, making a smart use of the available space, not forgetting the weight balance. Although it seems an easy task, in reality integration is a complex and long process in which it is necessary to deal with electronics, electromagnetics, chemistry, materials theory and soldering.

As a result, the final vehicle set up is reached and now the components work together as an unique and complex system: the UAV was named as "HoverWasp" (figure 4.2).

FIGURE 4.2: HoverWasp vehicle

Vehicle specifications:

- Rotor axis distance: 650mm diagonal
- Vehicle span: 950mm diagonal
- TOW: 2000g
- MTOW: 3000g
- Flight time: 25min
- Range: undefined
- Maximum combined thrust: 6000g
- Maximum combined power: 1760W

The vehicle could be divided into different subsystems connected together into the same frame (Figure 4.3), each one designed and built for the fullfilment of the tasks required. They are described in detail in the following sections of this chapter.

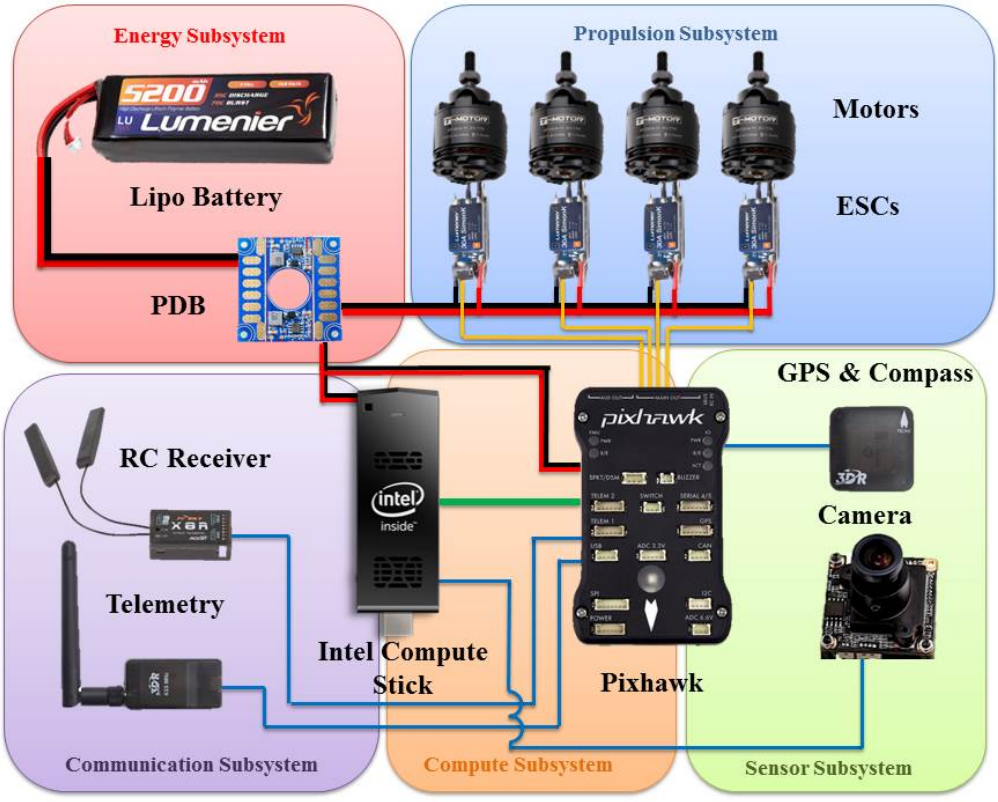


FIGURE 4.3: Hardware design

4.2 Frame subsystem

The skeleton is basically the frame in wich all the components are installed. The Tarot Iron Man 650 is built by Toray 3K carbon fiber woven board, with 3K hollow twill pure carbon fiber tube and a full CNC machining plate designed with higher standards. All the carbon and CNC of the full frame weighs only 476 grams. The arms folding design makes it highly portable and is highly adjustable to meet the needs of most utilitarian applications.

Features:

- Fold-ability for transportation and storage
- Toray 3K carbon fibre
- Lightweight
- Folding landing gear
- Motor-to-motor size: 650mm
- Weight: 476g
- Height from ground to lower rods: 180mm
- Height from ground to top: 220mm

4.3 Energy subsystem

The energy subsystem is responsible for providing the different componets sufficient power for their correct performance. The power source of the vehicle is an unique DC source. It is a LiPo (Lithium Polimer) battery, a class of batteries that make use of a solid polymer electrolyte (SPE) such as polyethylene oxide (PEO) and characterized for their high performance and their good power-to-weight ratio.

In HoverWasp vehicle a four cells battery is used, that delivers a mean of 14.5 V. However, not all the componets work with that amount of voltage. This is why the vehicle makes use of a PDB (Power Distribution Board) and a voltage step-down, that supply different voltages to feed different components.

4.4 Propulsion subsystem

The propulsion subsystem is responsible for the generation of the forces and momentums for making the vehicle fly. It is composed for four electric motors, propellers and ESC's (Electric Speed Controller).

The ESC's are conected to the PDB for energy feeding and to the Pixhawk board for the input PWM signal, and are in charge of transforming the DC current from the energy subsystem into AC to control the brushless motors. Their correct performance is crucial for the manoeuvrability and stability of the vehicle.

The motors and propellers transmit the electric force into movement and into aerodynamic forces and momentums.

4.4.1 T-motor MT2814 710KV 440W antigravity series

T-motor is a worldwide leading supplier of components and systems for the aerial photography, industry uses and commercial applications in the precision drive technology sector.

Specifications of the T-motor MT2814:

- KV: 710
- Max Continuous Power: 440W
- Max Continuous current: 27A
- Max efficiency current(6-18A): more than 83
- Internal resistance: 125mΩ
- Configuration: 12N14P
- Stator Diameter: 28mm
- Stator Length: 14mm
- Shaft Diameter: 4mm
- Motor Dimensions: 35mm x 36mm

- Weight: 120g
- ldle current (10v): 0.4A
- N°of Cells (Lipo): 3S - 4S

Figure 4.4 stores the data of the motor testing performed by the manufacturer.

Item No.	Volts (V)	Prop	Throttle	Amps (A)	Watts (W)	Thrust (G)	RPM	Efficiency (G/W)	Operating temperature(℃)
Antigravity MT2814 KV710	11.1	T-MOTOR 11*3.7CF	50%	3	33.30	310	4730	9.31	37
			65%	4.1	45.51	360	5300	7.91	
			75%	5	55.50	400	5750	7.21	
			85%	6.9	76.59	530	6450	6.92	
			100%	8.4	93.24	630	6900	6.76	
		T-MOTOR 12*4CF	50%	3.3	36.63	430	4300	11.74	38
			65%	4.7	52.17	500	4900	9.58	
			75%	6.6	73.26	660	5550	9.01	
			85%	8.8	97.68	800	6150	8.19	
			100%	10.8	119.88	920	6580	7.67	
		T-MOTOR 13*4.4CF	50%	3.6	39.96	510	4100	12.76	43
			65%	5.5	61.05	600	4800	9.83	
			75%	7.7	85.47	760	5400	8.89	
			85%	10.3	114.33	900	6000	7.87	
			100%	12.6	139.86	1060	6400	7.58	
		T-MOTOR 14*4.8CF	50%	4.2	46.62	560	3600	12.01	48
			65%	7.5	83.25	800	4450	9.61	
			75%	10.7	118.77	1000	5000	8.42	
			85%	14	155.40	1200	5500	7.72	
			100%	16.9	187.59	1360	5800	7.25	
		T-MOTOR 15*5CF	50%	5.1	56.61	700	3400	12.37	55
			65%	9.1	101.01	920	4100	9.11	
			75%	13.1	145.41	1140	4600	7.84	
			85%	17.3	192.03	1380	5050	7.19	
			100%	20.3	225.33	1530	5300	6.79	
	14.8	T-MOTOR 11*3.7CF	50%	4.4	65.12	540	6000	8.29	45
			65%	6.1	90.28	600	6700	6.65	
			75%	8.2	121.36	760	7400	6.26	
			85%	11	162.80	940	8300	5.77	
			100%	13.3	196.84	1100	8800	5.59	
		T-MOTOR 12*4CF	50%	4.9	72.52	680	5500	9.38	50
			65%	7.7	113.96	860	6400	7.55	
			75%	10.5	155.40	1040	7100	6.69	
			85%	14.1	208.68	1280	7800	6.13	
			100%	16.8	248.64	1460	8300	5.87	
		T-MOTOR 13*4.4CF	50%	5.2	76.96	700	5100	9.10	54
			65%	8.8	130.24	980	6100	7.52	
			75%	12	177.60	1200	6840	6.76	
			85%	15.7	232.36	1480	7500	6.37	
			100%	19.2	284.16	1600	7950	5.63	
Notes: The test condition of temperature is motor surface temperature in 100% throttle while the motor run 10 min.									

FIGURE 4.4: Characteristic and efficiency table of the T-motor MT2814 710kV 440W

4.5 Sensor subsystem

This subsystem is responsible of obtaining the enviroment data and measurements about vehicle state (attitude, position...). It is composed by different components spread over the vehicle:

- GPS: Ublox LEA-6H.
- Gyros: ST Micro L3GD20 3-axis 16-bit and MPU 6000 3-axis both installed inside Pixhawk.
- Accelerometers: ST Micro LSM303D 3-axis 14-bit and MPU 6000 3-axis both installed inside Pixhawk.
- Compass: HMC5883L digital compass.
- Barometer: MEAS MS5611 precission barometer.
- Camera: ELP 1080 OV2710 sensor.

Summing up, they offer information about vehicle position and attitude and images of the surrounding and gives the information to the Pixhawk board and the on-board computer.

4.6 Communication subsystem

There are several communication channels in the vehicle, each one designed for a different purpose:

- Radio control (2.4GHz): for manual RC control and recovery, it is stablished between FrSky X8R module and Taranis transmitter controller.
- Wifi: installed inside the on-board computer, it provides communication channel, via ssh synchronization, for running software components inside the on-board computer.
- Bluethooth: installed also inside the on-board compute, it is not used in this project

- Telemetry (433MHz): MAVLink Protocol based channel for monitoring vehicle state.

4.7 Computing subsystem

The computational resources of the vehicle are distributed in two boards, each of them in charge of a different level of control of the vehicle:

- Pixhawk: responsible for the low level of control, it acts as intermediary between the high level of control, and the sensors and actuators. It stabilises the vehicle with the attitude information, compute point-to-point navigation and includes an autopilot.
- Intel Compute Stick (ICS): it is the on-board computer in charged of high level control, ICS processes the images, obtains information given by Pixhawk and makes decisions which are delivered to the low level in order to be executed.

4.7.1 Pixhawk board from 3Drobotics

Pixhawk is an advanced autopilot system designed by the PX4 open-hardware project and manufactured by 3D Robotics. It features advanced processor and sensor technology from ST Microelectronics® and a NuttX real-time operating system, delivering incredible performance, flexibility and reliability for controlling many autonomous vehicles.

The benefits of the Pixhawk system include integrated multithreading, a Unix/Linux-like programming environment, completely new autopilot functions such as sophisticated scripting of missions and flight behaviour, and a custom PX4 driver layer ensuring tight timing across all processes. Pixhawk allows existing APM and PX4 operators to seamlessly transition to this system and lowers the barriers to entry for new users to participate in the world of autonomous aerial vehicles.

Features:

- Advanced 32 bit ARM Cortex® M4 Processor running NuttX RTOS

- 14 PWM/servo outputs (8 with failsafe and manual override, 6 auxiliary, high-power compatible)
- Abundant connectivity options for additional peripherals (UART, I2C, CAN)
- Integrated backup system for in-flight recovery and manual override with dedicated processor and stand-alone power supply
- Backup system integrates mixing, providing consistent autopilot and manual override mixing modes
- Redundant power supply inputs and automatic failover
- External safety button for easy motor activation
- Multicolor LED indicator High-power, multi-tone piezo audio indicator
- MicroSD card for long-time high-rate logging

Specifications:

- Microprocessor:
  - 32-bit STM32F427 Cortex M4 core with FPU
  - 168 MHz/256 KB RAM/2 MB Flash
  - 32 bit STM32F103 failsafe co-processor
- Sensors:
  - ST Micro L3GD20 3-axis 16-bit gyroscope
  - ST Micro LSM303D 3-axis 14-bit accelerometer / magnetometer
  - Invensense MPU 6000 3-axis accelerometer/gyroscope
  - MEAS MS5611 barometer
- Interfaces:
  - 5x UART (serial ports), one high-power capable, 2x with HW flow control
  - 2x CAN
  - Spektrum DSM / DSM2 / DSM-X® Satellite compatible input up to DX8 (DX9 and above not supported)

  - Futaba S.BUS® compatible input and output
  - PPM sum signal
  - RSSI (PWM or voltage) input
  - I2C®
  - SPI
  - 3.3 and 6.6V ADC inputs
  - External micro USB port

- Power System:
  - Ideal diode controller with automatic failover
  - Servo rail high-power (7 V) and high-current ready
  - All peripheral outputs over-current protected, all inputs ESD protected

- Weight and Dimensions:
  - Weight: 38g
  - Width: 50mm
  - Thickness: 15.5mm
  - Length: 81.5mm

4.7.2 Intel Compute Stick STCK1 A8 LFC

Intel Compute Stick (ICS) is a cost-efficient computer for portable uses. It is a Mini PC with full-size performance, reliability, and ease of use. Intel Compute Stick has all the performance needed for running thin client, embedded or cloud applications.

ICS computer makes use of Linux Ubuntu 14.04 as operating system and the high level software components used in the project are intalled in it, as JdeRobot. This computer performs the computational effort required by the driver and the applications described on the chapters 5 and 6.

- Processor: Intel®Atom Processor Z3735F
- Graphics: Intel®HD graphics via HDMI

- Audio: Intel®HD audio via HDMI
- System memory: 1GB soldered, single channel, DDR3L memory
- Storage: 8GB eMMC
- Connectivity: Integrated 802.11 bgn wireless, Bluetooth
- 4.0, USB2.2, microSD slot
- Power requirements: 5V 2A DC
- Size: 103mm x 37mm x 12mm

4.8 Configuration and testing

To configure the Pixhawk firmware and update it, APM mission planner software has been used. This program offers a lot of tools that help the user to access to the software without fighting with the code lines. With this program some configurations were performed: update of firmware, calibration of some sensors (compass, gyros and accelerometers), calibration of the transmitter and setting of the different flight modes on the transmitter.

Prior to the vehicle first flight, it is necessary to make some tests on the ground, verifying the correct operation of the motors and the radiolink for safety reasons. Then, the flight test were performed:

- Fully manual: the main objective was to verify that all the components were connected and working correctly, almost without software intervention.
- Autonomous stabilisation and landing: in this case, we verified the correct working of the positioning sensors (gyros, accelerometers, compass, barometer and GPS) and tested the low level software performance.
- Simple waypoint mission: verification of the correct use of the point to point navigation that is previously configured and validation of the GPS and IMU data fusion inside the Pixhawk.

FIGURE 4.5: HoverWasp flight test

The different flight tests have been crucial for the development of the vehicle and its correct validation. Even with a complete success in the attemps, some changes were done in the vehicle in order to improve its performance, given as a result the vehicle described in this chapter:

- Change the power source from 3S to 4S, rising the velocity of the propeller and motor and improving substantially the response and the behaviour with harder weather conditions.
- Change the propeller from 14 inches to 12 inches of size, to maintain general amount of thrust with the power source change, reducing also the motor amperage drainage.
- Elimination of analogic video communication channel.
- Change of the RC channel to improve robustness against electromagnetic noise.

Once the flight tests of the vehicle platform were performed, the quadcopter is ready to start the software development. The next objective for the system was to be connected with JdeRobot, it is described in the chapter 5.

Chapter 5

Software Driver

This chapter describes the software component developed to autonomously interact with the drone from the applications. It is responsible for accessing the sensors and actuators of the vehicle using MAVLink protocol communication messages, and for translating them into JdeRobot ICE interfaces. The drone applications will access to the vehicle sensors and actuators talking to this driver.

5.1 Design

`MAVLinkServer` is a driver based on MAVProxy software (presented in section 3.4) and developed to act as a translator middleware. It has been designed as a JdeRobot driver in Python language. This commponent is also responsible for maintaining communication channels open and updated, both upstream and downstream.

`MAVLinkServer` relies on the MAVProxy parser, the part of the program that is in charge of the management of the MAVLink messages to and from the Pixhawk board. It establishes connection with Pixhawk autopilot, maintains the communication channel operative, acquires, interprets messages, creates and sends new ones with the information requested or ordered by the application.

The developed code is mainly in charge of the management of the JdeRobot interfaces. It is able to handle the information given by the MAVProxy side. It regulates the creation and modification of the classes where information is temporally stored and opens ICE communication channels to make the component usable for JdeRobot applications.

These two sides of the component provide a reliable and muli-compatible driver thanks to the ICE features; `MAVLinkServer` may connect with applications written in other different languages, like C++, Python or Java, through JdeRobot interfaces. The figure 5.1 represents an scheme of the blocks inside `MAVLinkServer` and its connections to other components.

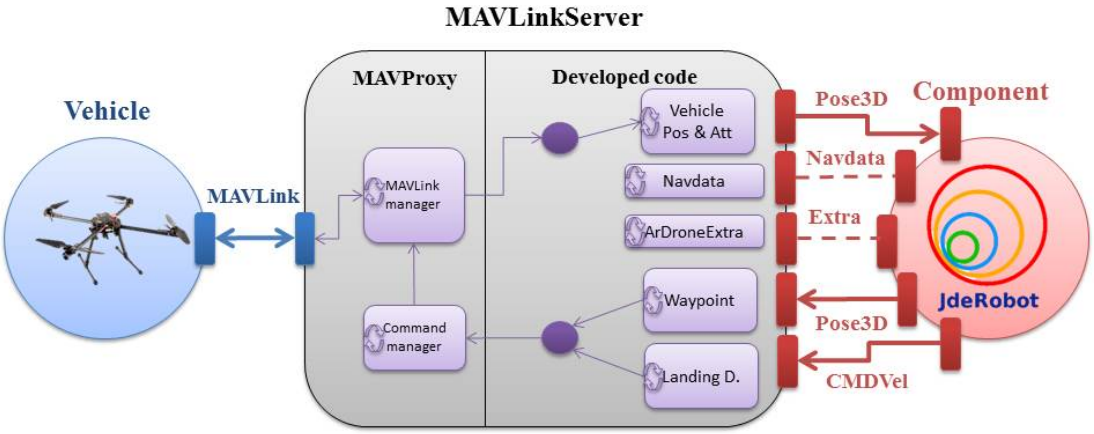


FIGURE 5.1: server Scheme

To accomplish the objectives of the project, `MAVLinkServer` needs to provide a certain level of control of the vehicle. At first attempt, this component was thought to provide "intermediate" control based on velocity commands, but this version of MAVProxy does not implement them. Facing this problem, `MAVLinkServer` was redesigned to make use of a higher level of control: waypoints for drone navigation using Pose3D JdeRobot interface.

The used JdeRobot interfaces in this project are Pose3D (one providing the measured vehicle position and attitude and other one for ordering a waypoint position) and CMDVel (for sending the landing decision). Note that Pose3D makes use of quaternions instead of Euler angles. This is done to avoid angle singularities and overlaps and computationally they are simpler than other attitude formalisms such as Euler angles or a direction cosine matrix.

Pose3D

```
Pose3DData{
    float x;  /* x coord */
    float y;  /* y coord */
    float z;  /* z coord */
    float h;  /* */
    float q0; /* qw */
    float q1; /* qx */
    float q2; /* qy */
    float q3; /* qz */
    };
```

CMDVel

```
class CMDVelData{
    float linearX;
    float linearY;
    float linearZ;
    float angularX;
    float angularY;
    float angularZ;
    };
```

5.2 Implementation

First step is to define the corresponding interfaces (explained in section 5.1). `MAVLinkServer` supplies all the JdeRobot interfaces for the drone access from any external component. Here it is an example of Pose3D interface.

```
import jderobot, time, threading
lock = threading.Lock()

class Pose3DI(jderobot.Pose3D):

    def __init__(self,_x,_y,_z,_h,_q0,_q1,_q2,_q3):

        self.x = _x
        self.y = _y
```

```
10          self.z = _z
11          self.h = _h
12          self.q0 = _q0
13          self.q1 = _q1
14          self.q2 = _q2
15          self.q3 = _q3
16
17          print "Pose3D start"
18
19      def setPose3DData(self, data, current=None):
20
21          lock.acquire()
22          self.x = data.x
23          self.y = data.y
24          self.z = data.z
25          self.h = data.h
26          self.q0 = data.q0
27          self.q1 = data.q1
28          self.q2 = data.q2
29          self.q3 = data.q3
30          lock.release()
31
32          return 0
33
34      def getPose3DData(self, current=None):
35
36          time.sleep(0.05) # 20Hz (50ms) rate to tx Pose3D
37
38          lock.acquire()
39          data = jderobot.Pose3DData()
40          data.x = self.x
41          data.y = self.y
42          data.z = self.z
43          data.h = self.h
44          data.q0 = self.q0
45          data.q1 = self.q1
46          data.q2 = self.q2
47          data.q3 = self.q3
48          lock.release()
49
50          return data
```

This class inherits "jderobot.Pose3D" and the corresponding functions are defined. It can be noticed the use of programming "locks" for protecting the data stored in the class. This is because `MAVLinkServer` is constantly refreshing the information provided by the sensors and also constantly publishing it through ICE. This could cause race conditions. With the use of the locks in the classes (figure 5.2), the information could not be read while other task is writing on it and the other way around, ensuring the correct management of the information in mutual exclusion.

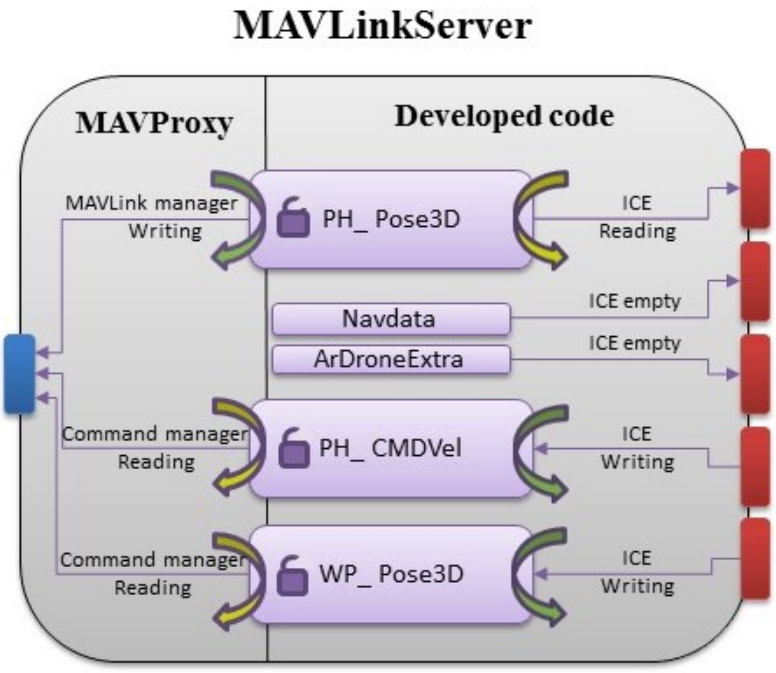


FIGURE 5.2: Class memory sharing

CMDVel interface has the same structure as Pose3D, also with the use of locks.

```
1  import jderobot, time, threading
2  lock = threading.Lock()
3
4  class CMDVelI(jderobot.CMDVel):
5
6      def __init__(self,lx,ly,lz,ax,ay,az):
7
8          self.linearX = lx
9          self.linearY = ly
10         self.linearZ = lz
11         self.angularX = ax
```

```
12          self.angularY = ay
13          self.angularZ = az
14
15          print "cmdvel start"
16
17      def __del__(self):
18
19          print "cmdvel end"
20
21      def setCMDVelData(self, data, current=None):
22
23          lock.acquire()
24
25          self.linearX = data.linearX
26          self.linearY = data.linearY
27          self.linearZ = data.linearZ
28          self.angularX = data.angularX
29          self.angularY = data.angularY
30          self.angularZ = data.angularZ
31
32          lock.release()
33
34          return 0
35
36      def getCMDVelData(self, current=None):
37
38          time.sleep(0.05)  # 20Hz (50ms) rate to rx CMDVel
39
40          lock.acquire()
41
42          data = jderobot.CMDVelData()
43          data.linearX = self.linearX
44          data.linearY = self.linearY
45          data.linearZ = self.linearZ
46          data.angularX = self.angularX
47          data.angularY = self.angularY
48          data.angularZ = self.angularZ
49
50          lock.release()
51
52          return data
```

`MAVLinkServer` launches several threads for ICE communication channels, one for each type of information. Despite only Pose3D and CMDVel are to be really used, the driver offers the remaining interfaces (NavData and ArDroneExtra) for compatibility and future uses.

```
1  PH_Pose3D = Pose3DI(0,0,0,0,0,0,0,0) #1 to avoid indeterminations
2  PH_CMDVel = CMDVelI(0,0,0,0,0,0) #1 to avoid indeterminations
3  PH_Extra = ExtraI()
4  WP_Pose3D = Pose3DI(0,0,0,0,0,0,0,0)
5
6  #Open an ICE TX communication and leave it open in a parallel threat
7  PoseTheading = threading.Thread(target=openPose3DChannel, args=(PH_Pose3D,)
       , name='Pose_Theading')
8  PoseTheading.daemon = True
9  PoseTheading.start()
10
11 # Open an ICE RX communication and leave it open in a parallel threat
12 CMDVelTheading = threading.Thread(target=openCMDVelChannel, args=(PH_CMDVel
       ,), name='CMDVel_Theading')
13 CMDVelTheading.daemon = True
14 CMDVelTheading.start()
15
16 # Open an ICE TX communication and leave it open in a parallel threat
17 CMDVelTheading = threading.Thread(target=openExtraChannel, args=(PH_Extra,)
       , name='Extra_Theading')
18 CMDVelTheading.daemon = True
19 CMDVelTheading.start()
20
21 # Open an ICE channel empty
22 CMDVelTheading = threading.Thread(target=openNavdataChannel, args=(), name=
       'Navdata_Theading')
23 CMDVelTheading.daemon = True
24 CMDVelTheading.start()
25
26 # Open an ICE TX communication and leave it open in a parallel threat
27 PoseTheading = threading.Thread(target=openPose3DChannelWP, args=(WP_Pose3D
       ,), name='WayPoint_Theading')
28 PoseTheading.daemon = True
29 PoseTheading.start()
30
31 # Open an MAVLink TX communication and leave it open in a parallel threat
```

```
32 PoseTheading = threading.Thread(target=sendWayPoint2Vehicle, args=(
       WP_Pose3D,), name='WayPoint2Vehicle_Theading')    PoseTheading.daemon =
       True
33 PoseTheading.start()
34
35 # Open an MAVLink TX communication and leave it open in a parallel threat
36 PoseTheading = threading.Thread(target=landDecision, args=(PH_CMDVel,),
       name='LandDecision2Vehicle_Theading')
37 PoseTheading.daemon = True
38 PoseTheading.start()
```

Each thread makes use of its own function where the ICE configuration is performed. There the ICE publication is done and it is important to ensure which data is sent, in order for other applications to get the information correctly and ensure compatibility. The same procedure is performed for all the interfaces. The following code represents the Pose3D function as an example.

```
1  def openPose3DChannelWP(Pose3D):
2
3      status = 0
4      ic = None
5      Pose2Rx = Pose3D #Pose3D.getPose3DData()
6
7      try:
8          ic = Ice.initialize(sys.argv)
9          adapter = ic.createObjectAdapterWithEndpoints("Pose3DAdapter", "
       default -p 9994")
10         object = Pose2Rx
11         #print object.getPose3DData()
12         adapter.add(object, ic.stringToIdentity("Pose3D"))
13         adapter.activate()
14         ic.waitForShutdown()
15
16     except:
17         traceback.print_exc()
18       status = 1
19
20     if ic:
21       # Clean up
22       try:
```

```
23            ic.destroy()
24        except:
25            traceback.print_exc()
26    status = 1
27
28 sys.exit(status)
```

MAVproxy is constantly handling MAVLink messages in a parallel thread in a "while true" loop. This component takes advantage of it and refreshes of the sensor information required. As a high level driver, this program does not interfere with the data fusion performed by Pixhawk and it trusts on its performance, which reliability has been widely tested. The following code is programmed after the MavProxy tasks inside the mentioned loop:

```
1  Rollvalue = mpstate.status.msgs['ATTITUDE'].roll      #rad
2  Pitchvalue = mpstate.status.msgs['ATTITUDE'].pitch    #rad
3  Yawvalue = mpstate.status.msgs['ATTITUDE'].yaw        #rad
4
5  # ESTIMATED: fused GPS and accelerometers
6  PoseLatLonHei = {}
7  PoseLatLonHei['lat'] = math.radians((mpstate.status.msgs['
       GLOBAL_POSITION_INT'].lat)/1E7)      #rad
8  PoseLatLonHei['lon'] = math.radians((mpstate.status.msgs['
       GLOBAL_POSITION_INT'].lon)/1E7)      #rad
9  PoseLatLonHei['hei'] = (mpstate.status.msgs['GLOBAL_POSITION_INT'].
       relative_alt)/1000    #meters
10
11 PH_quat = quaternion.Quaternion([Rollvalue, Pitchvalue, Yawvalue])
12 PH_xyz = global2cartesian(PoseLatLonHei)
13
14 #print PH_quat
15 #print PH_xyz
16
17 data = jderobot.Pose3DData()
18 data.x = PH_xyz['x']
19 data.y = PH_xyz['y']
20 data.z = PH_xyz['z']
21 data.h = 1
22 data.q0 = PH_quat.__getitem__(0)
23 data.q1 = PH_quat.__getitem__(1)
```

```
data.q2 = PH_quat.__getitem__(2)
data.q3 = PH_quat.__getitem__(3)

PH_Pose3D.setPose3DData(data)
```

For the vehicle waypoint delivery other thread is launched, as the ICE channel ones, which its corresponding associated functions. It makes use of the MAVProxy order manager, in this case, with the command "guided lat lon hei", which sends to the vehicle the waypoint in GPS coordinates to be reached.

```
def sendWayPoint2Vehicle(Pose3D):

    while True:
        time.sleep(1)
        wayPointPoseXYZ = Pose3D.getPose3DData()
        wayPointXYZ = {}
        wayPointXYZ['x'] = wayPointPoseXYZ.x
        wayPointXYZ['y'] = wayPointPoseXYZ.y
        wayPointXYZ['z'] = wayPointPoseXYZ.z
        wayPointLatLonHei =  cartesian2global(wayPointXYZ)

        latittude = str(wayPointLatLonHei['lat'])
        longitude = str(wayPointLatLonHei['lon'])
        altittude = str(int(wayPointLatLonHei['hei']))

        WPstring = 'guided ' + latittude + ' ' + longitude + ' ' +
    altittude
        process_stdin(WPstring)

        #print wayPoint
```

At the final step of the mission, the vehicle is ordered to land. This driver has an special function dedicated to this task and the management of the possible situations:

```
def landDecision(CMDVel):

    while True:
        time.sleep(1)
        command = CMDVel.getCMDVelData()
```

```
6
7       if (command.linearZ == -1):
8           print 'Lading decision: True'
9           process_stdin('mode land')
10
11          while (command.linearZ == -1):
12              time.sleep(1)
13              command = CMDVel.getCMDVelData()
14
15          print 'Target Lost, recovering trajectory'
16          process_stdin('mode guided')
```

In addition, several functions have been developed inside `MAVLinkServer` in order to complete the driver functionality . Among others, a function to change from GPS coordinates (lat, long, alt) to global Cartesian coordinates (x,y,z) or some simple functions have been defined for the correct usage of quaternions [25].

```
1
2 def global2cartesian(poseLatLonHei):
3
4       wgs84_radius = 6378137 #meters
5       wgs84_flattening = 1 - 1 / 298.257223563
6       eartPerim = wgs84_radius * 2 * math.pi
7
8       earthRadiusLon = wgs84_radius * math.cos(poseLatLonHei['lat'])/
        wgs84_flattening
9       eartPerimLon = earthRadiusLon * 2 * math.pi
10
11      poseXYZ = {}
12      poseXYZ['x'] = poseLatLonHei['lon'] * eartPerimLon / (2*math.pi)
13      poseXYZ['y'] = poseLatLonHei['lat'] * eartPerim / (2*math.pi)
14      poseXYZ['z'] = poseLatLonHei['hei']
15
16      return poseXYZ
17
18 def cartesian2global(poseXYZ):
19
20      wgs84_radius = 6378137  # meters
21      wgs84_flattening = 1 - 1 / 298.257223563
22      eartPerim = wgs84_radius * 2 * math.pi
```

```
23      referenceLat = 40.1912 ## Suposed to be Vehicle lattitude
24
25      radLat = math.radians(referenceLat)
26      earthRadiusLon = wgs84_radius * math.cos(radLat)/wgs84_flattening
27      eartPerimLon = earthRadiusLon * 2 * math.pi
28
29      poseLatLonHei = {}
30      poseLatLonHei['lat'] = poseXYZ['y'] * 360 / eartPerim
31      poseLatLonHei['lon'] = poseXYZ['x'] * 360 / eartPerimLon
32      poseLatLonHei['hei'] = poseXYZ['z']
33
34      return poseLatLonHei
35
36  def body2NED(CMDVel, Pose3D):
37
38      q1 = [0, CMDVel.linearX, CMDVel.linearY, CMDVel.linearZ]
39      q2 = [Pose3D.q0, Pose3D.q1, Pose3D.q2, Pose3D.q3]
40
41      q1 = qNormal(q1)
42      q2 = qNormal(q2)
43
44      q2inverse = qInverse(q2)
45      qtempotal = qMultiply(q1,q2inverse)
46      q = qMultiply(q2,qtempotal)
47
48      rotatedVector = q[1:len(q)] #obtain [q1,q2,q3]
49
50      return rotatedVector
51
52
53  def qMultiply (q1,q2):
54
55      q1 = qNormal(q1)
56      q2 = qNormal(q2)
57
58      # quaternion1
59      w1 = q1[0]
60      x1 = q1[1]
61      y1 = q1[2]
62      z1 = q1[3]
63
```

```
64      #quaternion2
65      w2 = q2[0]
66      x2 = q2[1]
67      y2 = q2[2]
68      z2 = q2[3]
69
70      w = w1*w2 - x1*x2 - y1*y2 - z1*z2
71      x = w1*x2 + x1*w2 + y1*z2 - z1*y2
72      y = w1*y2 + y1*w2 + z1*x2 - x1*z2
73      z = w1*z2 + z1*w2 + x1*y2 - y1*x2
74
75      q = [w,x,y,z]
76
77      q = qNormal(q)
78      return q
79
80  def qNormal(q1):
81
82      qmodule = math.sqrt(q1[0]*q1[0] + q1[1]*q1[1] + q1[2]*q1[2] + q1[3]*q1
        [3])
83      q = [0,0,0,0]
84
85      if (qmodule == 0):
86          qmodule = 0.000000000001
87
88      q[0] = q1[0] / qmodule
89      q[1] = q1[1] / qmodule
90      q[2] = q1[2] / qmodule
91      q[3] = q1[3] / qmodule
92
93      return q
94
95  def qConjugate(q1):
96
97      q1 = qNormal(q1)
98      q = [0,0,0,0]
99      q[0] = q1[0]
100     q[1] = -q1[1]
101     q[2] = -q1[2]
102     q[3] = -q1[3]
103
```

```
104     q = qNormal(q)
105     return q
106
107 def qInverse(q1):
108
109     q1 = qNormal(q1)
110     qconjugate = qConjugate(q1)
111     qmodule = math.sqrt(q1[0] * q1[0] + q1[1] * q1[1] + q1[2] * q1[2] + q1
        [3] * q1[3])
112
113     if (qmodule == 0):
114         qmodule = 0.000000000001
115
116     q = [0,0,0,0]
117     q[0] = qconjugate[0] / qmodule
118     q[1] = qconjugate[1] / qmodule
119     q[2] = qconjugate[2] / qmodule
120     q[3] = qconjugate[3] / qmodule
121
122     q = qNormal(q)
123     return q
```

5.3 Information pipeline

`MAVLinkServer` is a multithreaded component. The flow of the information and the operation of the driver pursue the following task path:

- The driver starts running all the different threads, opening its communication channels and defining the information that would be in each one. It makes use of two communication channels based on Pose3D, one for publishing vehicle position and attitude and another one for receiving waypoints orders; and a CMDVel channel for landing commands.
- MAVLink messages from Pixhawk come into the driver, and they are interpreted in order to get the position and attitude information of the vehicle. Acquired information is treated to transform it into JdeRobot standards (Pose3D). Latitude

and longitude are transformed into global xyz coordinates, using WGS84 as Earth model, and Euler angles attitude are transformed into quaternions.

- Pose3D is written in the corresponding local classes in a controlled way, making use of the lock.
- Classes are published in the corresponding ICE channels as the threads are running, to let other JdeRobot components to access to them.
- Landing commands are received through CMDVel and waypoints through Pose3D channels. Information is extracted from the corresponding classes with the mentioned locks and a reference frame change is performed in order to be synchronised with Pixhawk. Body referenced waypoints are transformed into GPS coordinate system making use of the designed quaternion functions. Waypoints and landing decision are finally delivered to MAVProxy command manager.
- Commands are translated into a MAVLink messages and sent to the Pixhawk board.

It can be seen that each thread has a task but they do not have all the same workload. For this reason the timing of the control loop of each thread is different. Despite the threads have different rhythms, the component is successfully working in all its different tasks.

5.4 Testing

For the verification and validation of the performance of this driver component, some experimental tests have been carried out.

First, the correct adquisition and trasformation of the sensors data, the correct performance of the different threads launched and right publication of the data throught ICE communication channels were verified. To do so,the well-tested JdeRobot tool, Uav-viewer developed by Martin Florido, shown in the figure 5.3, was used and connected to the real HoverWasp running the developed driver. It can be seen that the data collected by the sensors of the vehicle is correctly sent throught the driver ICE communication channels and displayed on the Uav-viewer GUI.

FIGURE 5.3: Uav-viewer connected to Hoverwasp trought MAVLinkServer

The second experiment was to verify the correct adquisition of the commands provided by an external component and the right delivery of them to the Pixhawk board via MAVProxy and MAVLink. The figure 5.4 shows how a waypoint to be reached is stablished by an external component (right window) and the comand is correctly sent to the pixhawk obtaining a call back message (left window).

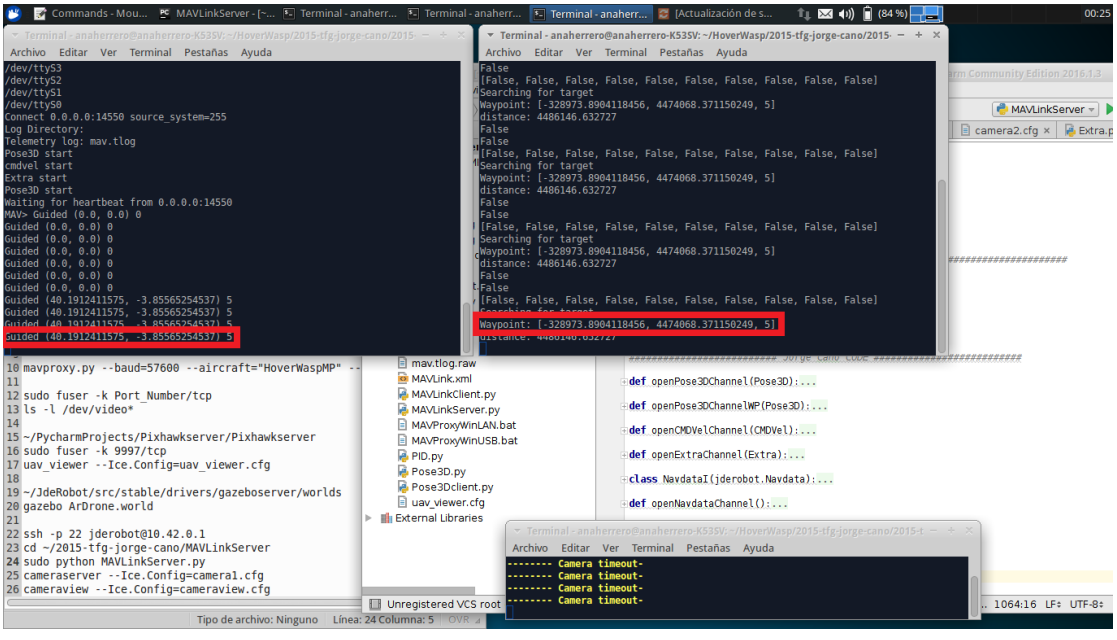


FIGURE 5.4: MAVLinkServer sending commands to Hoverwasp

During these experiment some limitations were detected, related to the implementation of the velocity commands on MAVProxy. The MAVLink messages corresponding with velocity commands are not well built by MAVProxy, and the Pixhawk board is not able to interpret the orders. As they were completely out of the scope of this project, MAVLinkServer was redesigned to accept waypoint orders instead of velocity commands, as mentioned before.

Chapter 6

Application component

Once the software infrastructure to access the sensors and actuators of the UAV have been presented, in this chapter the developed application that implements an example of autonomous navigation will be described. The visual control is a control system in which the feedback obtained from a video camera is used to decide vehicle behavior.

6.1 Behaviour design

This application aims to perform a target search mission around a designated localisation; once the target has been spotted, loiter over that position and land in the nearby area. This mission uses the information supplied by the attitude sensors and the on-board camera. One possible direct real implementation of this mission would be a "man overboard" search in the sea.

The searching method is based on a "last-known-position" method which assumes that the target is more likely to be found around its last known position, if an established trajectory or movement is not recognised. The scan method chosen in this project is the spiral, in which the vehicle describes a squared spiral trajectory starting on that last known position and covering further distance on each loop.

The behavior has been programmed in the `MAVLinkClient` component, which has four different functions: First one is to design and manage the mission and to establish the search trajectory, the second one is to calculate the next waypoint and to send it to

the vehicle, the third function is to analyse the images and identify the target, and the fourth one is to loiter over the target and land.

`MAVLinkClient` is a JdeRobot component completely developed in this project using Python as programming language. The objective of the component is to generate waypoints (Pose3D) based on the processing of the information available, that is, to make its own decisions and to perform the desired autonomous task. The figure 6.2 represents the block diagram of the component and its connections.

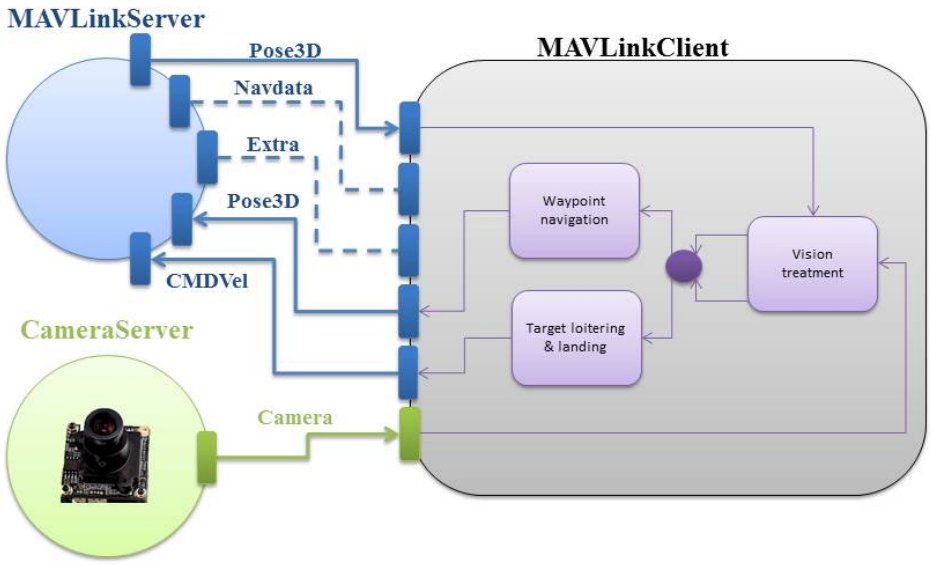


FIGURE 6.1: Client Scheme

6.2 Implementation

`MAVLinkClient` is a multithreaded component that makes use of the available ICE channels and sends differrent commands depending on the information provided by the drone driver and the camera driver. This component will be launched appart from `MAVLinkServer` and `cameraserver`, the JdeRobot driver for camera. All the threads have to be run before starting any computation.

```
PH_Pose3D = Pose3DI(0,0,0,0,0,0,0,0)

PoseTheading = threading.Thread(target=rxPose3D, args=(PH_Pose3D,), name='
    ClientPHPose_Theading')
PoseTheading.daemon = True
PoseTheading.start()

WP_Pose3D = Pose3DI(0,0,0,0,0,0,0,0)

PoseTheading = threading.Thread(target=txPose3DWP2Server, args=(WP_Pose3D,)
    , name='ClientWPPose_Theading')
PoseTheading.daemon = True
PoseTheading.start()


PH_CMDVel = CMDVelI(0,0,0,0,0,0)

CMDVelTheading = threading.Thread(target=txCMDVel2Server, args=(PH_CMDVel,)
    , name='ClientCMDVel_Theading')
CMDVelTheading.daemon = True
CMDVelTheading.start()

CameraTheading = threading.Thread(target=rxCamera, args=(), name='
    ClientCamera_Theading')
CameraTheading.daemon = True
CameraTheading.start()
```

As in the driver component, each threas makes use of its own function where the ICE configuration is performed and the handle of the information in the cannel is done.

```
def rxPose3D(Pose3D):

    status = 0
    ic = None
    try:
        ic = Ice.initialize(sys.argv)
        base = ic.stringToProxy("Pose3D:default -p 9998")
        datos = jderobot.Pose3DPrx.checkedCast(base)
        #print datos

```

```
11          if not datos:
12              raise RuntimeError("Invalid proxy")
13
14
15          while True:
16              time.sleep(0.02)
17              data = datos.getPose3DData()
18              Pose3D.setPose3DData(data)
19              #print Pose3D
20
21      except:
22          traceback.print_exc()
23          status = 1
24
25      if ic:
26          # Clean up
27          try:
28              ic.destroy()
29          except:
30              traceback.print_exc()
31              status = 1
32
33  def txPose3DWP2Server(Pose3D):
34
35      status = 0
36      ic = None
37      try:
38          ic = Ice.initialize(sys.argv)
39          base = ic.stringToProxy("Pose3D:default -p 9994")
40          datos = jderobot.Pose3DPrx.checkedCast(base)
41          #print datos
42          if not datos:
43              raise RuntimeError("Invalid proxy")
44
45          while True:
46              time.sleep(0.02)
47
48              Pose3D2send = Pose3D.getPose3DData()
49              datos.setPose3DData(Pose3D2send)
50
51              # print Pose3D
```

```
52
53      except:
54          traceback.print_exc()
55          status = 1
56
57      if ic:
58          # Clean up
59          try:
60              ic.destroy()
61          except:
62              traceback.print_exc()
63              status = 1
64
65  def txCMDVel2Server(CMDVel):
66
67      status = 0
68      ic = None
69
70      try:
71          ic = Ice.initialize(sys.argv)
72          base = ic.stringToProxy("CMDVel:default -p 9997")
73          datos = jderobot.CMDVelPrx.checkedCast(base)
74          #print datos
75
76          if not datos:
77              raise RuntimeError("Invalid proxy")
78
79          while True:
80              time.sleep(0.05)
81              CMDVel2send = CMDVel.getCMDVelData()
82              datos.setCMDVelData(CMDVel2send)
83              #print CMDVel2send
84
85
86      except:
87          traceback.print_exc()
88          status = 1
89
90      if ic:
91          # Clean up
92          try:
```

```
93              ic.destroy()
94          except:
95              traceback.print_exc()
96              status = 1
97
98
99  def rxCamera():
100
101     status = 0
102     ic = None
103
104     try:
105         ic = Ice.initialize(sys.argv)
106         base = ic.stringToProxy("Camera:default -p 9999")
107         datos = jderobot.CameraPrx.checkedCast(base)
108         # print datos
109         if not datos:
110             raise RuntimeError("Invalid proxy")
111
112         while True:
113             time.sleep(0.5)
114             global PH_Camera
115             global CameraHeight
116             global CameraWidth
117             cameraImage = datos.getImageData("RGB8")
118             CameraHeight = cameraImage.description.height
119             CameraWidth = cameraImage.description.width
120
121             lock.acquire()
122             PH_Camera = np.frombuffer(cameraImage.pixelData, dtype=np.uint8
    )
123             PH_Camera.shape = CameraHeight, CameraWidth, 3
124             lock.release()
125             # cv2.imshow('PH_Camera', PH_Camera)
126
127     except:
128             traceback.print_exc()
129             status = 1
130
131     if ic:
132         # Clean up
```

```
133         try:
134             ic.destroy()
135         except:
136             traceback.print_exc()
137             status = 1
```

6.3 Search navigation following a spiral pattern

The vehicle should scan the selected area, so the first step is to define the search trajectory shown in the figure 2.1. The spiral desired is defined by four values:

- Central spiral point (CSP): (x,y) in global coordinates in metres
- Mission height in metres.
- Scan distance (S): distance between parallel sides of the spiral in metres.
- Number of spins: number of revolutions of the spiral before restarting trajectory.

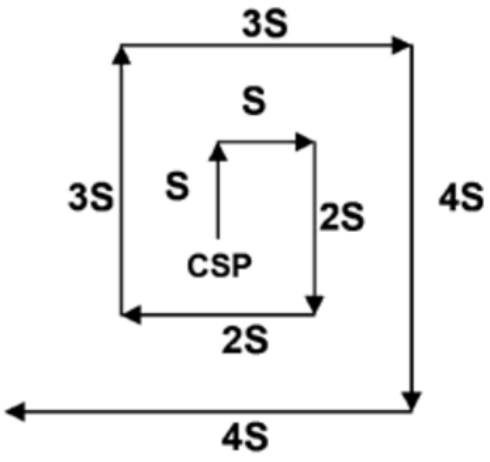


FIGURE 6.2: Squared spiral trajectory

The next function designs the trajectory based on the information given and returns a list of waypoints of the path:

```
def spiralTrajectory(startWP, scanDistance, spinsNumber):

    trajectory = []

    x = startWP['x']
    y = startWP['y']
    z = startWP['z']

    firstWP = [x, y, z]
    trajectory.append(firstWP)

    scanDistanceLatLon = scanDistance

    for i in range(spinsNumber):

        x = x + scanDistanceLatLon*(2*i+1)
        trajectory.append([x,y,z])
        y = y + scanDistanceLatLon*(2*i+1)
        trajectory.append([x,y,z])

        x = x - scanDistanceLatLon*(2*i+2)
        trajectory.append([x,y,z])
        y = y - scanDistanceLatLon*(2*i+2)
        trajectory.append([x,y,z])

    return trajectory
```

The following function is in charge of the management of the trayectory. It uses the current position and attitude of the vehicle (Pose3D) and calculates the relative position of the waypoint from the vehicle. A waypoint is considered caught up when the distance between vehicle and waypoint is less than a pre-determined distance ("ReachedDist").

```
def nextWaypointPose3D(position, trajectory):

    waypoint = trajectory[0]
    #vector = velVector(position, waypoint)
    print "Waypoint: %s" %waypoint
    dist = distance(waypoint, position)
    print "distance: %f" %dist

```

```
9       if (dist <= ReachedDist):
10          trajectory = trajectory[1:len(trajectory)] #pop
11          print "Waypoint reached"
12
13      return trajectory
```

While the target is not found, the component sends to the driver the position of the next waypoint to be reached.

```
1       print 'Searching for target'
2
3       # command, updatedTrajectory = nextWaypointCMDVel(xyz, trajectory)
4       updatedTrajectory = nextWaypointPose3D(vehicleXYZ, trajectory)
5       trajectory = updatedTrajectory
6
7       #restart trajectory when finish
8       if (len(trajectory) == 0):
9           trajectory = DefTrajectory
10          print "Trajectory restarted:"
11          print trajectory
12
13      #send waypoint to server
14      wayPoint = jderobot.Pose3DData()
15      wayPoint.x = trajectory[0][0]
16      wayPoint.y = trajectory[0][1]
17      wayPoint.z = trajectory[0][2]
18      WP_Pose3D.setPose3DData(wayPoint)
19      # print wayPoint
```

If during the trajectory following mode, the target is found, the vehicle changes its mission and tries to loiter over it; using the camera and image processing to identify the relative position between the vehicle and the target.

6.4 Visual perception of the target

The visual detection of an object, shown in the figure 6.3, and the decision of landing nearby are the core funtions of this component. This decision changes the mode and

the corresponding order is sent to the driver. This code is inserted inside the execution loop of the component.

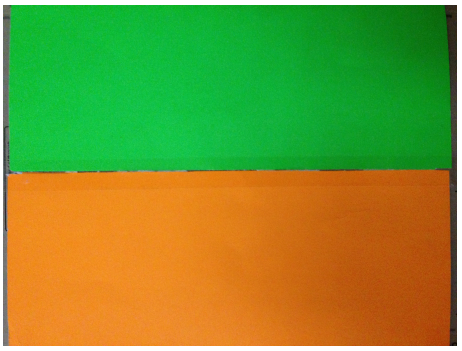


FIGURE 6.3: Target to be found by MAVLinkClient

```
time.sleep(1)#0.05) #20Hz
vehiclePose = PH_Pose3D.getPose3DData()
vehicleXYZ = [vehiclePose.x, vehiclePose.y, vehiclePose.z]
vehicleYaw = yawFromQuaternion(vehiclePose)

lock.acquire()
Image2Analyze = PH_Camera
ImageShape = PH_Camera.shape
lock.release()

GreenCenter, GreenArea, GreenFound = detection(Image2Analyze, hminG, hmaxG,
    sminG, smaxG, vminG, vmaxG)
OrangeCenter, OrangeArea, OrangeFound = detection(Image2Analyze, hminO,
    hmaxO, sminO, smaxO, vminO, vmaxO)
targetArea = GreenArea + OrangeArea

twoAreas = GreenFound and OrangeFound
nearAreas = distance(GreenCenter,OrangeCenter) <= math.sqrt(targetArea)

if twoAreas and nearAreas:
    targetCentrePixels = (((GreenCenter[0] + OrangeCenter[0]) / 2), ((
    GreenCenter[1] + OrangeCenter[1]) / 2))
    targetBuffer.append(True)
else:
    targetBuffer.append(False)

targetBufferData = targetBuffer.get()

```

```
26 targetFound = targetBuffer.decicion()
27 print targetBufferData
```

For target detection in the image this project has chosen to identify objects from its colour. This colour detection has some disadvantages compared to other detection methods. The biggest drawback is that light changes in the environment where the robot moves and the shadows can project into the target. The images obtained from the camera are encoded in RGB. This colour space is not very robust to changes in ambient light. For this reason a change from the RGB colour space to HSV was decided, which although does not completely solve the problem, it is robust enough for detecting an object in a changing environment, as long as the light changes are not very high.

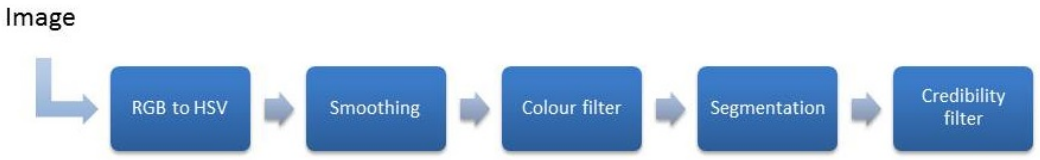


FIGURE 6.4: *Pipeline* of the perception system of `object_tracking`

The pipeline of the perception system 6.4 is computationally simple and lightweight. The following sections will explain in more detail each of the stages.

```
1  def detection(image, hmin, hmax, smin, smax, vmin, vmax):
2      #--------- Thresold Image ---------#
3
4      rawImage = image
5
6      ksize = (5, 5)
7      sigma = 9
8      gaussImage = cv2.GaussianBlur(rawImage, ksize, sigma)
9
10     hsvImage = cv2.cvtColor(gaussImage, cv2.COLOR_RGB2HSV)
11
12     matrixMin = np.array([hmin, smin, vmin])
13     matrixMax = np.array([hmax, smax, vmax])
14
15     filteredImage = cv2.inRange(hsvImage, matrixMin, matrixMax)
```

```
16
17      #------- Detect Object -------#
18
19      center = []
20      area = 0
21      center = (0,0)
22      colorFound = False
23
24
25      detImage = rawImage
26
27      contour, hierarchy = cv2.findContours(filteredImage, cv2.RETR_EXTERNAL,
         cv2.CHAIN_APPROX_NONE)
28
29      if len(contour) > 0:
30          contourdx = -1
31          cv2.drawContours(detImage, contour, contourdx, (255, 255, 0))
32
33      for i in range(len(contour)):
34          contarray = contour[i]
35
36          epsilon = 5
37          closed = True
38          approxCurve = cv2.approxPolyDP(contarray, epsilon, closed)
39
40          rectangle = cv2.boundingRect(approxCurve)
41          rectX, rectY, rectW, rectH = rectangle
42
43          if rectW < maxTargetLado and rectW > minTargetLado and rectH <
        maxTargetLado and rectH > minTargetLado:
44              myRectangle = rectangle
45          else:
46              myRectangle = 0, 0, 0, 0
47
48          myRectX, myRectY, myRectW, myRectH = myRectangle
49          myPoint1 = myRectX, myRectY
50          myPoint2 = myRectX + myRectW, myRectY + myRectH
51          center = myRectX + (myRectW / 2), myRectY + (myRectH / 2)
52          area = myRectW * myRectH
53
54          # cv2.rectangle(detImage,myPoint1,myPoint2,(255,0,0),2)
```

```
55          cv2.circle(detImage, center, 1, (255, 0, 0), 2)
56
57          minTargetArea = minTargetLado * minTargetLado
58
59          if (area > minTargetArea)and(center != (0,0)):
60              colorFound = True
61          else:
62              colorFound = False
63
64      return center, area, colorFound
```

As a result, the component obtains a positive or negative identification of the object, and in the case, the horizontal relative position between the vehicle and the target is estimated. The detection of the target requires a lot of computational effort. It is the heaviest part of the component. The detection is performed twice, one for each colour of the target.

6.4.1 RGB to HSV

If a colour image is about to be treated, the most common encoded values are in the RGB colour space (Red, Green, Blue) that determines the colour composition in terms of the intensity of the three primary colours of light. The RGB model is commonly used to display colours on screens, TVs, etc., but has a big disadvantage when trying to identify a colour within a real image. The RGB colour space is not very robust against light changes. In the field of robotics, where robots are in changing environments, it is convenient to use another colour space more robust to light changes, for example the HSV space, represented in the figure 6.5.

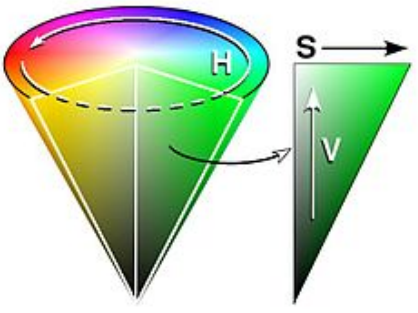


FIGURE 6.5: Cone colors space HSV

HSV (Hue, Saturation, Value) is a nonlinear transformation of RGB model in cylindrical coordinates. Thus, the colour is defined by:

- Hue: is the angle between 0 and 360 or representing colour tone.
- Saturation: is the offset of black-white brightness values ranging from 0 to 100.
- Value: is the height in the white-black axis, which has values ranging from 0 to 100.

The changes in brightness are usually reflected in V, leaving the S and H more or less unchanged. Due to the possibility to define a colour based on a hue (H) and a saturation(S), the HSV model is an excellent candidate for robust identification of objects through their colour in an image within a changing environment.

6.4.2 Image Smoothing

Within the image processing stage, there are some techniques that improve some aspect of the image, as the smoothing techniques. Noise, signal degradation, errors in the adquisition process or image transmission, lack of uniform illumination can interfere the object analysis. The perception system in the component implements the smoothing technique Gaussian Blur, which is the result of convoluting an image with the Gaussian function (figure 6.6):

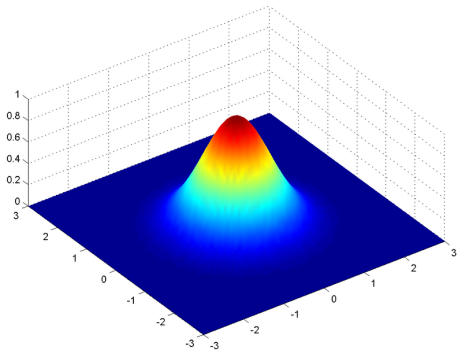


FIGURE 6.6: Gauss function in 2D

Distribution values are used to construct a convolution matrix that is applied to the original image. The central pixel value is given greater weight (higher value of the Gaussian) and adjacent pixels receive a smaller weight depending on the distance from

the original pixel. The result of this process is a blur retaining borders and edges of the object. This task is performed by the function of OpenCV GassianBlur(), and the result can be seen on the figure 6.7.

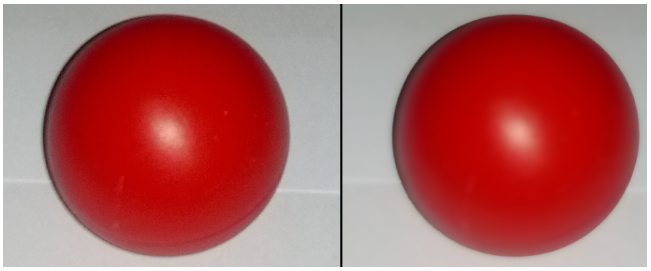


FIGURE 6.7: *Gassuian Blur* (left: raw image; right: smoothed image)

6.4.3 Colour Filtering

The choice of colour is an important aspect of object detection. A vivid colour will be easily detectable, but a colour with more muted tones will be more difficult to detect. The tool in JdeRobot, named ColourTunner, allows us to select the colours that we want to identify, defining their HSV values. By applying thresholds to a colour image it is transformed into a binary image, where objects of interest stand out with a different background pixel value. If the pixel value passes the threshold, the resulting pixel will be white, while if it fails, it will be black; as can be noticed on the figure 6.8. The result is the binary image. `inRange()` function, belonging to OpenCV, applies thresholds to the image.

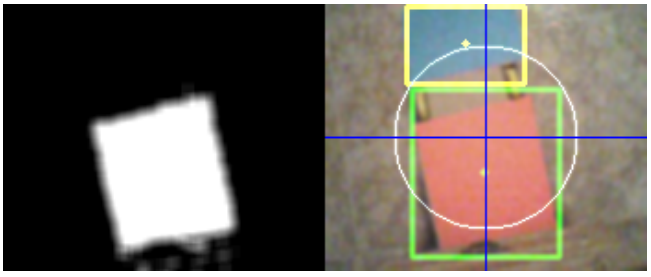


FIGURE 6.8: Image thresholding

6.4.4 Segmentation

The purpose of object detection is to obtain the coordinates of the centre of the object in the image. Thus, we can distinguish whether the object has moved in different iterations

of the algorithm and distinguish where. To accomplish this task the perception system of the `MAVLinkClient` component performs a series of steps:

- Once the binary image with the desired colour pixel detection has been obtained, a segmentation technique based on edge detection is used. The idea of this method is to detect the borders of objects.
- With the outlines of objects located, the next step is to approach them to polygons. From a list of points that determine the outline of an object, the `approxPolyDP()` function OpenCV is used. It is able to return another list of points with a small number of them that define a particular poligon.

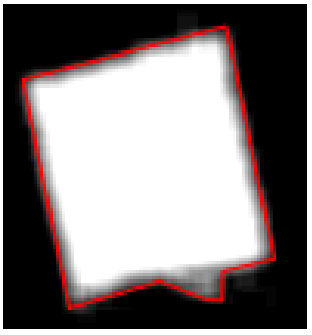


FIGURE 6.9: Contours detected with the function *findContours()*

- Knowing some point in the rectangle on the image, it can find the centre of the box following a simple formula. The `boundingRect()` function OpenCV returns a list of rectangles from a list of points. This function attempts to create the minimum rectangle that contains all the points in the list indicated as parameter to the function.
- After calculating the rectangle that contains the target, the centre coordinates of the poligon are obtained in order to stimate the position of the target with respect to the camera. The coordinates will be in the reference system of the image itself, so it is necesary to make a change to the used reference system.

6.4.5 Noise filtering

Due to the ambient light conditions, the proper motion of the drone or the object we are detecting would leave the image, the rectangle approximated by the perception system fluctuates. These fluctuations cause the estimated coordinates of the centre of the object vary and, in many cases, the object has not actually moved. For example, a change in ambient light causes the sensing system detecting an object of size different from the size of the same object previously detected. To combat against this error the application has implemented a "Buffer" for making a "credibility filter". This filter takes into account the previously captured images (a certain number samples) and storse a positive or negative variable in order to determine whether the target has really been identified or not.

When the number of positive identifications are higher than an established threshold, a confirmation is generated and the code changes the drones's task from following trajectory to loitering control.

The number of samples and the threshold give the filter its reliability and change the response and the consistency of the `MAVLinkClient` component

6.5 Loitering Control and Landing

In this control the only aspect to take into account is to position of the drone above the colour that marks the position of the object on the ground. Only two dimensions are controlled, the X axis and Y axis; height control is done by the pixhawk itself, with the established altitude.

Loitering control will take as reference the 2D position of the point corresponding to the centre of the object in the image, estimated by the perception system, and calculates the difference of it with respect the image centre point. It will give us the error of the position of the drone relative to the target in both X and Y. These perception errors are transformed into real scale errors using a function which depends on the height of the vehicle.

```
1 def global2cartesian(poseLatLonHei):
```

```
2
3       wgs84_radius = 6378137 #meters
4       wgs84_flattening = 1 - 1 / 298.257223563
5       eartPerim = wgs84_radius * 2 * math.pi
6
7       earthRadiusLon = wgs84_radius * math.cos(poseLatLonHei['lat'])/
        wgs84_flattening
8       eartPerimLon = earthRadiusLon * 2 * math.pi
9
10      poseXYZ = {}
11      poseXYZ['x'] = poseLatLonHei['lon'] * eartPerimLon / (2*math.pi)
12      poseXYZ['y'] = poseLatLonHei['lat'] * eartPerim / (2*math.pi)
13      poseXYZ['z'] = poseLatLonHei['hei']
14      return poseXYZ
15
16  def pixel2metres(pixelXY, pixelNum, height, fov):
17
18      height = MissionHeight #only for testing
19      vehicleAngle = fov/2
20      groundAngle = (math.pi/2) - vehicleAngle
21      hipoten = height / math.sin(groundAngle)
22      groundSide = math.sqrt((hipoten*hipoten) - (height*height))
23
24      metersXpixel = 2*groundSide / pixelNum
25
26      metresXY = [metersXpixel*pixelXY[0] , metersXpixel*pixelXY[1]]
27
28      return metresXY
29
30  def frameChange2D(vector, angle):
31
32      rotatedX = vector[0]*math.cos(angle) + vector[1]*math.sin(angle)
33      rotatedY = vector[1]*math.cos(angle) - vector[0]*math.sin(angle)
34
35      rotatedVector = (rotatedX,rotatedY)
36
37      return rotatedVector
```

With this information, the component estimates the position of the target and generates and commands to the `MAVLinkServer` a new waypoint to be reached. The estimation of the next waypoint is iterative in order to improve consistency of the component.

As a high level control and with the intrinsic errors of the system, a landing accurancy is necessary to be defined. The drone would try to reach the estimated position of the target and when the relative horizontal distance between target an vehicle is less than the landing accurancy, the component sends a landing order to the server via CMDVel ICE channel.

The landing decision is irrevocable from the program point of view; it is, when the component confirms the landing decision, whatever it happens, the landing decision is not changed from the program. However, `MAVLinkClient` still process images and calculates the next waypoint to reach, for recovering searching mission in the case of landing decision manually is changed.

6.6 Information pipeline

`MAVLinkClient` is a multithreaded component at the communication level. Nevertheless, the information and main task are lineal and follow this path:

- MAVLinkClient starts running all the different threads, receiving information of the communication channels and defining the information that would be sent to the driver.
- The component defines the trajectory and returns a list of waypoints.
- The images are updated and analized to detect the target
- Buffer stores the positive or negative detections and makes a decision.
- In the case of not finding the target, next trajectory waypoint is calculated and sent to the driver
- In the case of target found, the position of the target is calculated and the drone will try to reach it.
- When the target position is reached, landing command is sent to the server.

6.7 Testing

The testing of the `MAVLinkClient` is the final step of the project, the correct performance of the component defines the success of the project and this validation as a whole autonomous system. This experiment validates not only the built drone platform, but also the developed software driver and application component.

The first test validates mainly the target detection algorithm in simulation. The figure 6.10 shows an example of the test carried out, where it can be noticed that the function perfectly detects the colours of the target and calculates its central point.

The risk of testing the component directly on the vehicle is high, that is why this experiment makes use of Gazebo simulator for initial testing of the code. In the figure 6.10 the simulated vehicle is receiving the information provided by the component with the artificial vision algorithm being executed at the same time.

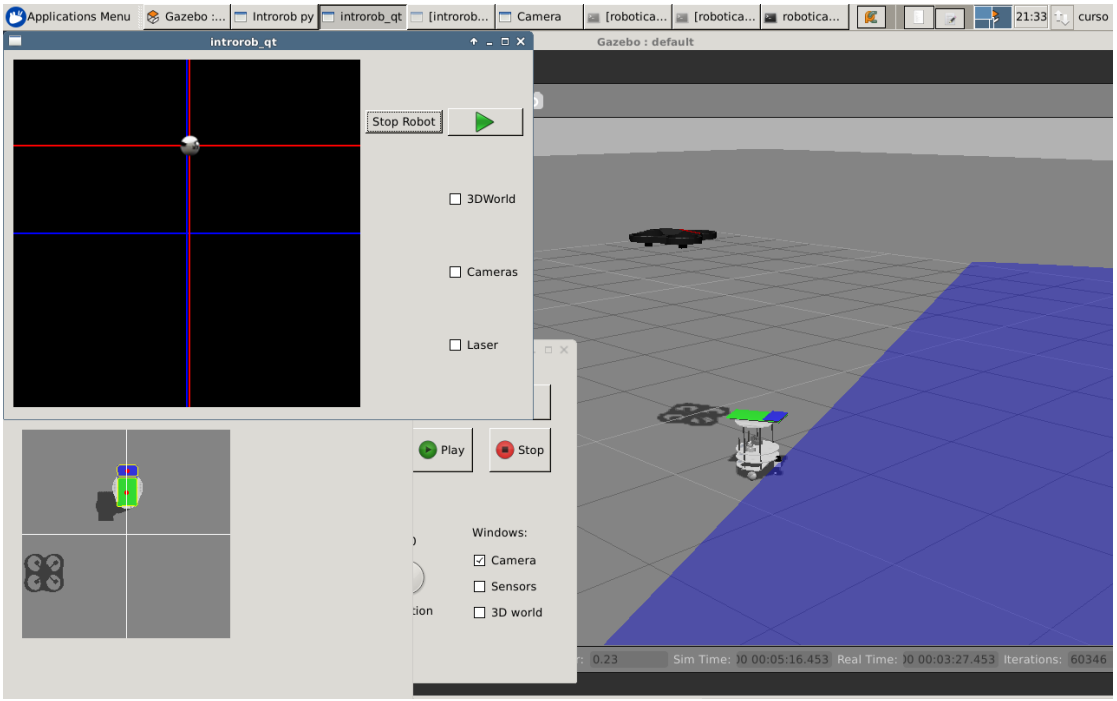


FIGURE 6.10: MAVLinkClient simulation test

With the success of the previous test, it was time to perform the final experiment with the real drone. `MAVLinkServer`, `cameraserver` and `MAVLinkClient` were launched in the on-board computer via ssh with and external computer, to execute the mission.

The mission starts and the vehicle starts scanning the area and following the calculated spiral search trajectory. `MAVLinkClient` analyzes the images provided by `cameraserver`, trying to identify the target. Once the target is found, new waypoints are calculated in order to get closer to the target and to make the decision of landing in the surroundings, commanding the decision to the drone through `MAVLinkServer`. The mission is considered successfully complete when the vehicle lands smooth and safely. The process of the mission is shown in the figure 6.11

(a) Mission start

(b) Start scanning

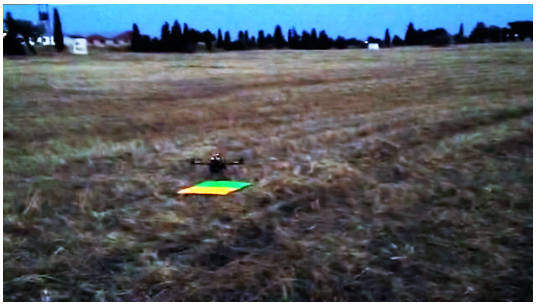
(c) Waypoint A

(d) Waypoint B

(e) Approaching to target

(f) Landing

(g) Landed A



(h) Landed B

FIGURE 6.11: Project mission

Chapter 7

Conclusions

In this chapter, a critical analysis of the outcomes described in the previous chapters is done to obtain conclusions, about the vehicle design and construction, the development of the driver and of the application. Finally, several future lines or next steps that may extend the work done on this project are suggested.

7.1 Conclusions

The primary objective of this proyect was to design, build and program an aerial vehicle for the fulfilment of an autonomous complex mission using artificial vision as source of information.

To acomplish the objective, the project has been divided into three main subgoals: design and construction of an aerial vehicle, development of the JdeRobot driver for such vehicle and the programming of a component for the management of the mission. Each subgoal and its developments has been described in their corresponding chapter.

- The HoverWasp vehicle has been desgined and built from scratch complying the stablished requirements. It is a reliable and useful platform in which new developments and missions could be carried on. Multirotor dynamics and electronics are the technological areas involved in this part, as well as the engineering effort done for its design and integration.

- `MAVLinkServer` component is a sofware driver that can be used to connect any MAVLink based vehicle to the JdeRobot software. The communication channels and interfaces have been specially selected following an intuitive criteria for its future use. The API's supported by this components are: Pose3D for the access to the sensors position infomation, and for providing high level control using waypoints; CMDVel for landing commands, and other extra JdeRobot interfaces for future applications. Programming capabilities, the undertanding and use of quaternions and cartography have been the learnt topics in this section.

- `MAVLinkClient` component makes use of high level point-to-point control of the UAV to complete an autonomous mission: search and detection of an objective using image processing scanning an outdoor area and landing in the surroundings. Computer vision has been the determining factor on the creation of this component.

Given the result of the previous chapters, it can be said that the global objective has been successfully achieved. All the phases of the project have been covered, from requirements analysis to final validation. The progess has been documented in the project webpage [13] and the code updated to the repository [14]. The driver software developed in this project has been integrated into the open source robotics middleware JdeRobot, being proud contributor to the development of the URJC robotics laboratory.

Along the development of the project, there has been much knowledge adquired. The field of UAV's gathers information that comes from different areas: programming, telecomunications, electronics, mechanics or aerospace are some of them.

As mentioned before in this document, theory and reality are parallel lines, not convergent ones. It means that it is completely necessary to bring to the real world the knowledge adquired in the engineering theory. It is a crucial step for a prepared engineer to have the opportunity to experiment and face real world problems, that at the end of the day, is the assingment which we have been trained for.

7.2 Future Lines

The value of this project is not only the achieved goals or the effort made for meet them; there is a greater value on the high number of applications that can be done using this project as a benchmark and starting point.

Possibly the main future development is to give support for middle level control, the one based on velocity commands. For the versions used in this project, this type of control has not been implemented yet in MAVProxy. Velocity command control would offer the complete control of the vehicle and the possibilities that would offer are uncountable.

Also, the vehicle taking off is too agresive for a safe use of it. A smoother control would benefit the platform and would almost cancel the human intervention in the missions.

Moreover, the change of the camera, from visible spectrum to a thermal detection cam, would improve the search and rescue application developed on this project.

Bibliography

[1] INTA. Programas de alta tecnologia - aviones no tripulados. http://www.inta.es/programasAltaTecnologia.aspx?Id=1&SubId=3, . [Online].

[2] INTA. Sistema diana. *Ministerio de defensa*, .

[3] INTA. Sistema milano. *Ministerio de defensa*, .

[4] UAV challenge. Competition website. https://uavchallenge.org/. [Online].

[5] Aerial Robotic Competition. Competition website. http://www.aerialroboticscompetition.org/. [Online].

[6] Flying Machine Arena. Web from the investigation group. http://www.flyingmachinearena.org/. [Online].

[7] Dario Brescianini, Markus Hehn, and Raffaello D'Andrea. Quadrocopter pole acrobatic. Flying Machine Arena project contribution. Swiss National Science Foundation (SNSF).

[8] Dario Brescianini and Raffaello D'Andrea. Design, modeling and control of an omni-directional aerial vehicle. Flying Machine Arena project contribution. Swiss National Science Foundation (SNSF).

[9] Sherpa Proyect. http://www.sherpa-project.eu/sherpa/, . [Online].

[10] Arcas Proyect. Aerial robotic cooperative assembly system. http://www.arcas-project.eu/, . [Online].

[11] Teppo Luukkonen. Modelling and control of quadcopter. *School of Science*, August 2011. Independent research project in applied mathematics.

[12] JdeRobots. Software framework for developing applications in robotics. http://jderobot.org. [Online].

[13] JdeRobot. Jorge cano end of degree project website. http://jderobot.org/J.canoma-tfg. [Online].

[14] GitHub. Jorge cano end of degree project code repository. https://github.com/RoboticsURJC-students/2015-tfg-jorge-cano. [Online].

[15] ArduPilot. Ardupilot project development documentation website. http://ardupilot.org/ardupilot/, . [Online].

[16] Pixhawk. Px4 autopilot project development documentation website. https://pixhawk.org. [Online].

[17] Paparazzi-UAV. Open source dron hardware and software project. https://wiki.paparazziuav.org/. [Online].

[18] ArduPilot. Mission planner documentation website. http://ardupilot.org/planner/, . [Online].

[19] Qgroundcontrol-MAVLink. Message marshalling library for mav. http://qgroundcontrol.org/mavlink/. [Online].

[20] Lorenz Meier. Professional website. http://people.inf.ethz.ch/lomeier/. [Online].

[21] MAVLink. Mavlink protocol message set description. https://pixhawk.ethz.ch/mavlink/. [Online].

[22] Sky-Drones. Smart autopilot company website. http://www.sky-drones.com/. [Online].

[23] ZeroC-ICE. Comprehensive rpc framework. https://zeroc.com/. [Online].

[24] OpenCV. Artificial vision library documentation webpage. http://opencv.willowgarage.com/wiki/. [Online].

[25] Karsten Groÿekatthöfer and Zizung Yoon. Introduction into quaternions for spacecraft attitude representation. *Department of Astronautics and Aeronautics*, May 2012. Technical University of Berlin.