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: _____

Abstract

The firsts “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 these type of technology on the military word have been crucial since that moment, and countries has spent tons of resources on the development of UAV’s (Unmanned aerial vehicle).

Even the industry of UAV’s born many years ago, it has been getting popularity between the general public as “Drones”, through spying films, war videogames and TV; with sophisticated gadgets able to scan far territories looking for sensitive information of the enemy, executing complex tasks to help “the good people”. Back to the earth, there still a great deal of differences from the drones on the films and the real ones.

This project intent 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 determined mission robustly and efficiently. The design, construction and program an UAV from scrach is the main objective, undertanding these vehicles indeed from the conception to the validation.

The solution purposed in this project chase 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 by an introduction, the working methodology followed and the infrastructure used.

The chapters 4,5 and 6 are the description of the different task to achieve: design and construction of a hardware platform, the implementation of a sofware driver to communicate with the vehicle and the programming of an application component 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 lots of aspect to develop, but they will became extremely important to our lives, more than we today imagine.

“”

Jorge Cano Martínez

Contents

List of Figures

List of Tables

Abbreviations

APM	**A**rdu**P**ilot**M**ega
ARCAS	**A**erial **R**obotics **C**ooperative **A**ssembly **S**ystem
ATM	**A**ir **T**raffic **M**anagement
CAA	**C**ivil **A**viation **A**uthority
CACGS	**C**omputer **A**ssisted **C**arrier **G**uidance **S**ystem
CTOL	**C**onventional **T**ake-**O**ff and **L**anding
DIY	**D**o **I**t-**Y**ourself
DoD	**D**epartment **o**f **D**efence
DOF	**D**egrees **O**f **F**reedom
ESC	**E**lectronic **S**peed **C**ontroller
FAA	**F**ederal **A**viation **A**dministration
FMA	**F**lying **M**achine **A**rena
GCS	**G**roup **C**omunication **S**ystem
GUI	**G**raphical **U**ser **I**nterface
ICAO	**I**nternational **C**ivil **A**viation **O**rganisation
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
MAV	**M**icro **A**erial **V**ehicle
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
RC	**R**adio **C**ontrol

RPAS	**R**emotely **P**iloted **A**ircraft **S**ystem
RPAV	**R**emotely **P**iloted **A**erial **V**ehicle
TOW	**T**ake-**O**ff **W**eight
SPE	**S**olid **P**olimer **E**lectrolite
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 begins until nowadays. Finally it is explained the development of the aerial robots and the research, commercial and defence use of this type of technology.

1.1 Robotics

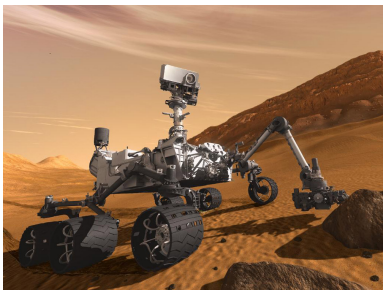
Robotics is a branch of engineering that involves the conception, design, manufacture, and operation of robots, as well as the different systems for their control and information processing. This field overlaps with, mechanical engineering, electronics, computer science, artificial intelligence, nanotechnology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

A robot is an artificial, mechanical or virtual, identity able to complete an activity autonomously, established beforehand, that substitutes humans in dangerous, difficult or special tasks. The world "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.

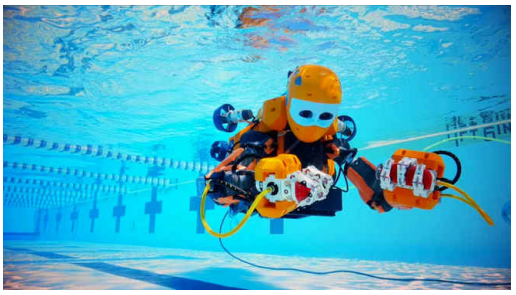
Since 2000 until today, robotics has starred huge breakthroughs compare to the advantages carried on in the 20^{th} century, mainly because of the advancement of technology and the reduction of it cost. Robotics has been able to conquer land, sea, and air in our planet and in other ones. The figure 1.1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

(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 company iRobot, 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 had 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. To locate themselves on

the field, his teammates, his opponents, the ball, field boundaries and the opposing goal, SLAM (Simultaneous Location and Mapping) technique of artificial visionis used.

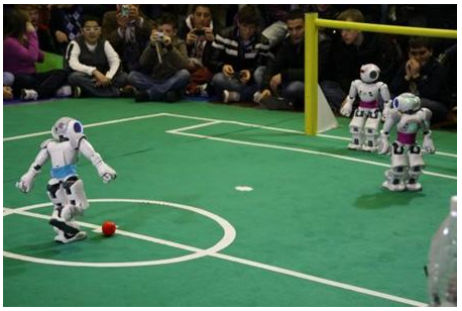
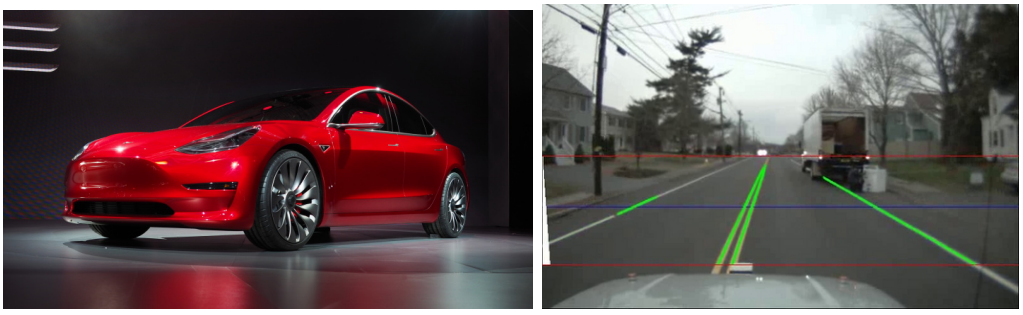


FIGURE 1.3: Nao robots autopositioning based on vision

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 it. In the figute 1.4 the Tesla Model 3 is shown, it is the last advantage of a long research line. Tesla compete with a differnet approach than the competence. Google, the other serious pioneer of autonomous driving, has based its research on very complex laser-based radar calle "lidar". Tesla has instead opted for cheap hardware sensors: cameras, ultrasound and advanced software.



(a) Tesla Model 3 (b) Curves 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, most repetitive 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 interact with it.

- The sensors allow the robot to get information from the environment or itself. This capability allows a robot to work on changing scenarios by changing its behaviour depending on the environment in which it is located. There are a variety of sensors, from the most basic as sensors light, temperature or proximity to the most complex and rich as cameras, depth sensors, lasers, GPS locators, etc. in aerial robots.
- Actuators are devices that allow the robot to interact with the environment and/or perform movements. Give the robot the ability to move or change the environment in which they work. They allow the robot to perform its tasks in a real environment, such as navigation avoiding obstacles or assembling parts in an assembly. The actuators can be of various types such as motors used for moving the wheels of the robot, the motors the movement 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 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

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 development software, software for robots must meet certain requirements:

- Adaptability: a robot is situated in a dynamic and unpredictable scenario 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 robots must be multitasking, having the ability to collect sensor data, process, decide and take action simultaneously.
- Distributed: sensors with increasingly more complex data processing algorithms and a large computational load, distributed computing have almost become a standard in the development of robotic applications. In addition, often coordinating multiple robots or use of other data sources external to the robot, such as systems cameras to detect 3D robot in a controlled environment, is necessary.
- Dealing with heterogeneous hardware: often same type sensor and actuator devices would be very different, and each manufacturer develops its own driver. So is necessary the existence of a homogeneous interface software to unify access to sensors and actuators regardless of manufacturer or model of the device. This allows developed algorithms for a given robot, can be exported to a different one to perform a task, enabling code reuse.

To help the developer responsible for making applications for robots, there are several software platforms (open or closed source code). These platforms typically are installed on an operating system, the platform is the bridge between the robot and the developer, and takes advantage of the characteristics of modern operating systems such as hardware abstraction, the multitasking capabilities and ease to create distributed applications with external libraries. Usually, if the robot model is widespread, the developer would also include simulators where code before 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 a human operator, uses aerodynamic forces to provide vehicle 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 autonomously, 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, even supervision of a person.

UAVs typically fall into one of six functional categories, however, there exist 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: delivering cargo.
- 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, sensors, etc.. The cheapening of UAVs technology has produce and 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 UAVs, it have been used only in military environments could be used for forest services, agriculture, environment, hydrology, cartography, natural disasters or geology.

1.2.1 Applications

UAV's have been traditionally use 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) [3]. For years INTA have been working in a research program to develop the necessary technologies to design and build a range of unmanned aircraft. The result of these activities, the Institute has developed the following products:

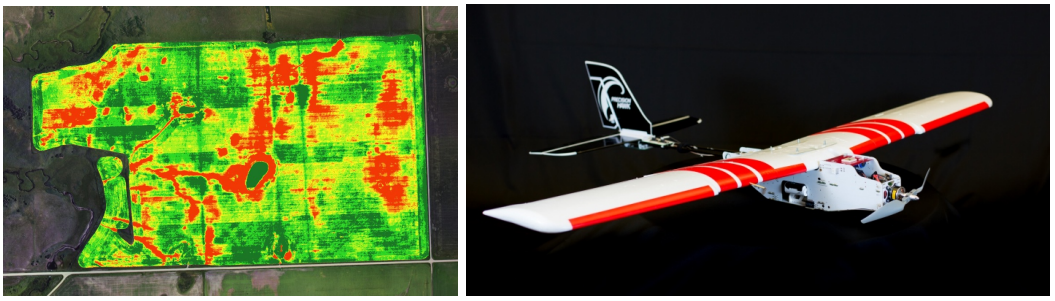
- SIVA: A sophisticated unmanned aerial surveillance multiple applications in the civil and military field, which can be used as a vehicle for real-time observation.
- ALO: A low cost observation system and high reliability suitable for the acquisition of aerial images in civilian and military missions short range.
- ALBA: A complete system of aerial guided target suitable for improving the operation of anti-aircraft artillery units through their training under real fire.
- MILANO [4]: An strategic system of monitoring and observing composed by UAV's linked via satellite with a ground control station. From the station is planned, monitors and controls both the aircraft and the payloads shipped. The aircraft has a range of more than 20 hours and can operate at altitudes up to 26,000 feet.
- DIANA [5]: 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 commertial and popular application is Google's Project Wing, the code name for the company's drone delivery service it hopes to launch in 2017. The company first publicly 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 could deliver 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 consist on the adquisition of information of the terrain for the optiimizing the resources expent on the labor (figure 1.7 a). There exist a wide variety of information useful for this sector, as water resources, vegetation analysis, 3D mapping or desease detection. This advance on agriculture would save a great deal of resources and would helps also to the sustainability of the agriculture; optimizing the use of pesticides and water. There exist a wide variety of enterprises and projects dedicate to precision agriculure, one great examen 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 nw UAV testing and development are the competition for UAV's, that have been growing worldwide.

In these competitions, where teams have to compete in creating an application to overcome a particular challenge it is to encourage research into these devices. They are normally focused on college students although some competitions for high school students.

The UAV Outback Challenge [6] 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 challenges in the world and one of the highest stakes UAV challenges, 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 tendency 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) [7] is an aerial robotics competition, founded in 1991 at the Georgia Institute of Technology, in competing universities around the world. The main purpose of the competition is to advance research behaviors aerial robotics. From 1991 through 2013 they have proposed a total of 7 missions. Each is to develop a fully autonomous behaviours that is able to pass the test. New missions are created every time the above 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 centres in quadcopters vehicles. Its facilities include the Flying Machine Arena (FMA) [8] 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 allow 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 to move 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 must be executed in the vehicle for a certain movement, and then the wireless network this command is sent to the vehicle performing the action.

FMA group carry on several research projects, in which it could be find interesting projects as:

- "Quadcopter Pole Acrobatics" [9] system that allows quadcopters to balance an inverted pendulum, throw it in to the air, and catch and balance it again on second vehicle. Based on first principles 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 derived and the corresponding position is predicted by simulating the current position and velocity estimates forward in time. Algorithm is introduced that 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.

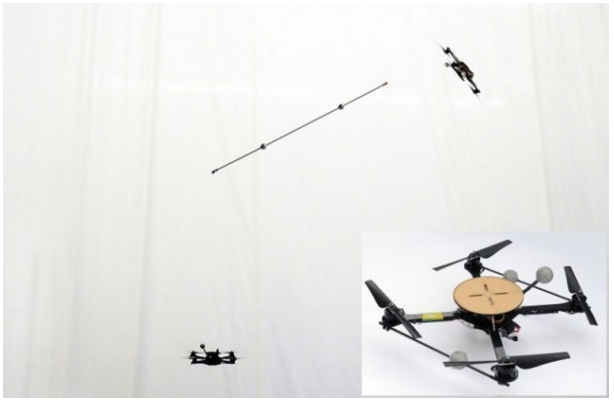


FIGURE 1.8: Poleacro quadcopter

- "Omni-Directional Aerial Vehicle" [10] design and control of a novel six degrees-of-freedom aerial vehicle. Based on a static force and torque analysis for generic actuator configurations, we derive 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 and capable of flying at any orientation (figure 1.9).

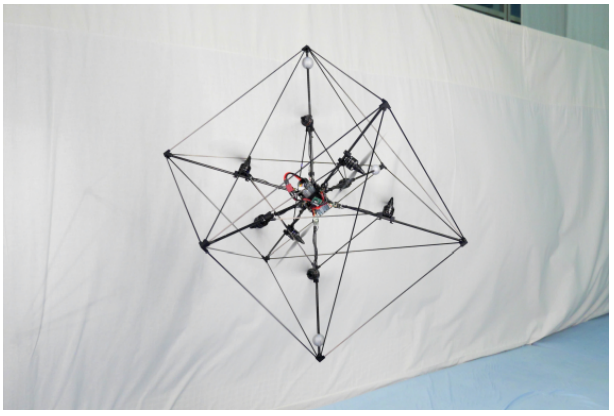


FIGURE 1.9: Omni-Directional Aerial Vehicle

- "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. 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 fashion. 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 [11], 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.

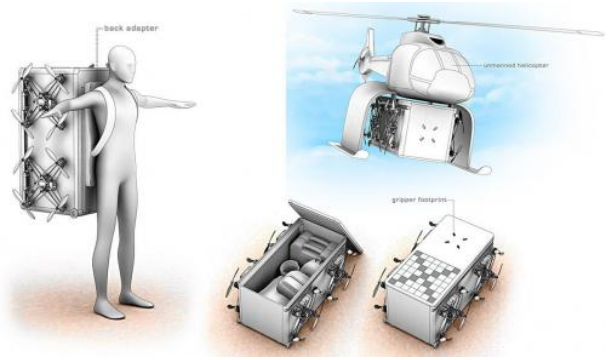


FIGURE 1.10: Sherpa Box

The technological platform and the alpine rescuing scenario are the occasion 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), [12] 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 involving two Spanish universities, "Universidad de Sevilla" and the "Universidad Politécnica de Cataluña".

1.3 Multi-Rotor Aircraft

UAVs introduced in the previous chapters have CTOL (Conventional Take-Off and Landing) capabilities, vehicles that need a runway for take-off and landing. The UAVs type of VTOL (Vertical Take-Off and Landing) are vehicles with the ability to hover and 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 VTOL aircraft capabilities as the aircraft 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- and double-rotor helicopters which use complex variable pitch rotors whose pitch varies 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 aircraft are frequently used in radio control aircraft and UAV projects in which the names tricopter, quadcopter,

hexacopter and octocopter are frequently used to refer to 3, 4, 6 and 8 rotor helicopters, respectively.

In order to allow more power and stability at 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).

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 propeller rotor axis. For maintaining an 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, compensates that momentum. The figure 1.12 shows a the forces and momentums involved.

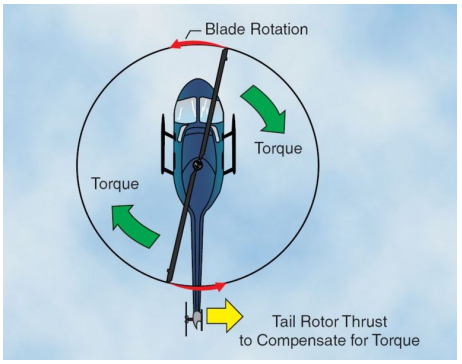
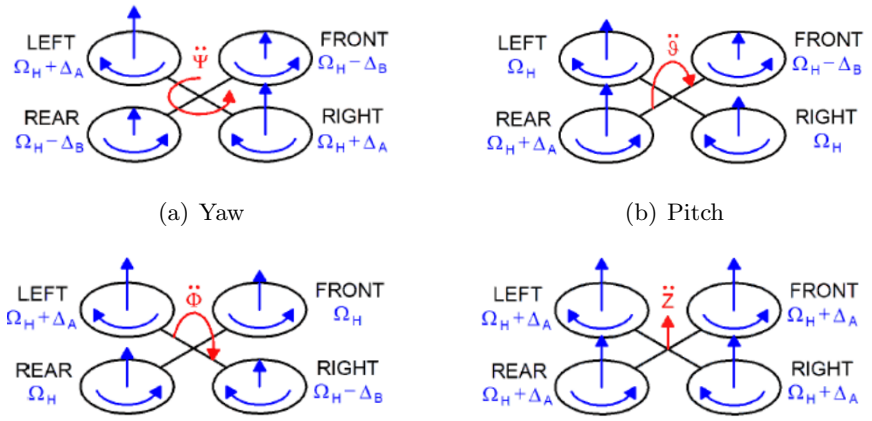


FIGURE 1.12: Torque compensation

Multirotors, take advantage of these momentum generated by its rotor to stabilise itself and manoeuvre. This vehicles typically uses pairs of rotor propellers to compensate each other momentum and varies their spinning velocity to achieve and control the momentum desired. The simplest way to explain this quality 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 one end, for example the rear, the front, the quadcopter would shift causing the vehicle turn head towards the floor because the raise of lift vector in that pair of rotor; 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 compensate. This movement is known as pitch, and produce the horizontal linear movement of the aircraft due to the inclination of the lift vector. The same effect is applied in different pair 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 these setup, we reach the 6 DOF [13](figure 1.14) desired for a complete control of the flight of a multirotor: linear velocities (x,y,z) and angular velocities (pitch, roll and

yaw).

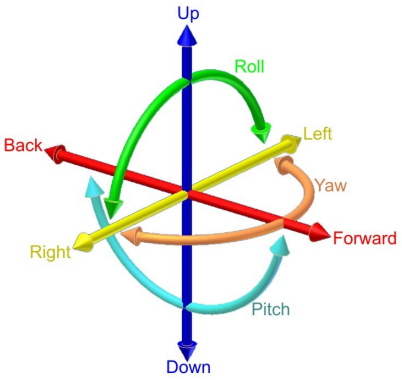


FIGURE 1.14: Roll, Pitch and Yaw

1.4 UAV Background at the URJC Robotis Laboratory

In 2013, within the group of robotics at the Rey Juan Carlos University, it have been launched a new line of research on UAV‘s, 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 antecedent of this project. It have been designed as drivers numerous robotic components, algorithms and support behavioral simulations, as robots working with NAO, ArDrone, FX-61 Phantom, Pioneer and Kobuki, among others. Among the projects carried out, 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 to JdeRobot applications. Within this work also includes a tool for remotely controlling the vehicle and read information from its sensors through a graphic interface, the “uav_viewer” component. In addition, an algorithm for vision and autonomous navigation have been developed within the component, this algorithm

allows the quadcopter to track objects in three dimensions using as source of information the front and ventral cameras.

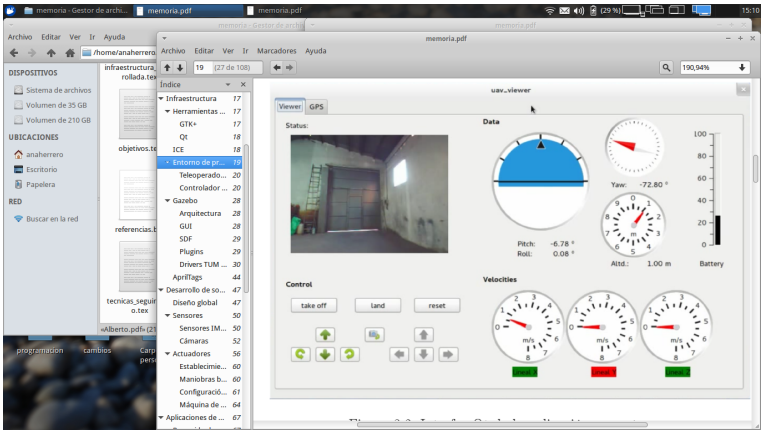


FIGURE 1.15: uav_viewer component intercace

Daniel Yagüe's project aim is to provide a support for aerial robots in the JdeRobot environment within the Gazebo simulator, so that there exist 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 follow through the ventral camera, locating objects in a determined area and location and tracking of other aerial vehicle.

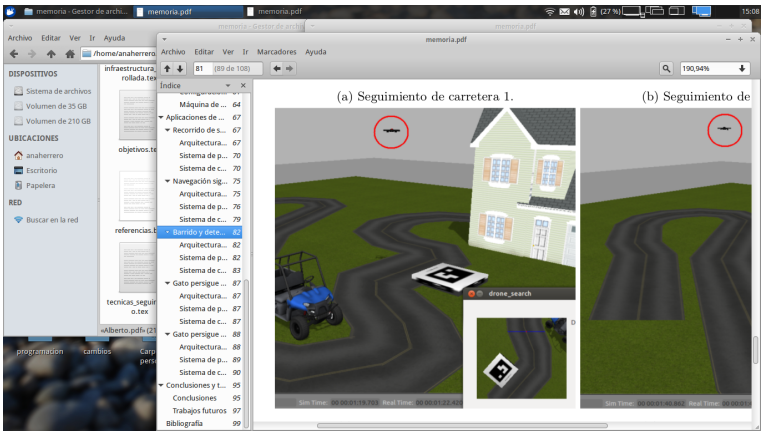


FIGURE 1.16: Road tracking Daniel Yagüe´s project

Chapter 2

Objectives

In this chapter the purpose and goals and requirements of the End-of-degree project are going to be describe, as well as the working methodology and work plan to achieve it successfully.

2.1 Project Objective

The general achievement of the project is to design and build an UAV vehicle, provide it with step-up autonomous intelligence and make it capable to fulfil an outdoor society useful mission, in the most robust and efficient way possible. For meeting the objective, the problem have been divided into three simpler goals:

- Design and construction of an open-source aerial vehicle able to offer to this project and future ones s reliable and effective platform for a long research.
- Program the desire software to provide the vehicle connection with JdeRobot [14] infrastructure and supply all the communication channels and information offerd by the vehicle sensors to design an autonomous task.
- Plan and execute an useful autonomous mission, exportable to real society problem solutions.

2.2 Requirements

The requirements of the project have to be define in order to ensure its consistency, its correct development and to do not lose focus in any step of the project.

- The developed infrastructure integrated in JdeRobot platform.
- The infrastructure would perform its mission having concern of its level of abstraction, disturbing as less as possible higher or lower levels obs abstraction, as the vehicle stabilisation process for example.
- The interfaces between components would make use, as much as possible, 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 this one.
- The communication channels should stream 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 defined by Barry Boehm in 1986, is based on a spiral in which each iteration represents a set of activities. These activities do not have priority, they will be chosen at the stage of risk analysis. Thus, each iteration is divided into the following activities:

- Determine targets: this activity in order to define the current iteration. Following this model the final goal of the project is divided into sub-goals.
- Risk Analysis: This activity is carried out several studies in order to meet potential threats or unwanted events that may occur in the current target.

- Develop and test: at this point verify the correct operation performed in iteration to correct the errors and that they do not continue in the following iterations will be necessary.

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

FIGURE 2.1: Spiral development model

To realise 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 were planning the next. If, however, had not been completed, make the current target deeper to correct errors or re-plan it. During the course of the project a web logbook [15] was agreed 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 [16]) accessible from the Internet. Thus the tutor and any other interested person has access at any time to code.

2.4 Work Planning

Once project have been consider validated, the execution phase of the project has follow the next procedure:

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 available (fix wing, helicopter, multirotor, etc...).

b) Design: market study, preliminary design and final components choice regarding requirements established, 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

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

b) Develop the driver software: server that supply to the infrastructure all the communication channels necessary and provide information captured by sensors.

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 in which the project use as starting point. The code developed in this project trust on the performance of the software following described.

3.1 Ardupilot

ArduPilot [17] (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 automatically 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 plug a great variety of sensors for configure the vehicle for the purpose desired.

Today, the ArduPilot project has evolved to a range of hardware and software products, including the APM and Pixhawk/PX4 [18] 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 [19], where low cost and availability enables its hobbyist use in small remotely piloted aircraft, 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, stabilish the vehicle and 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 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 [20] is a full-featured ground station application for the ArduPilot open source autopilot projectdeveloped by Michael Oborne and is part of 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 un the figure 3.1.

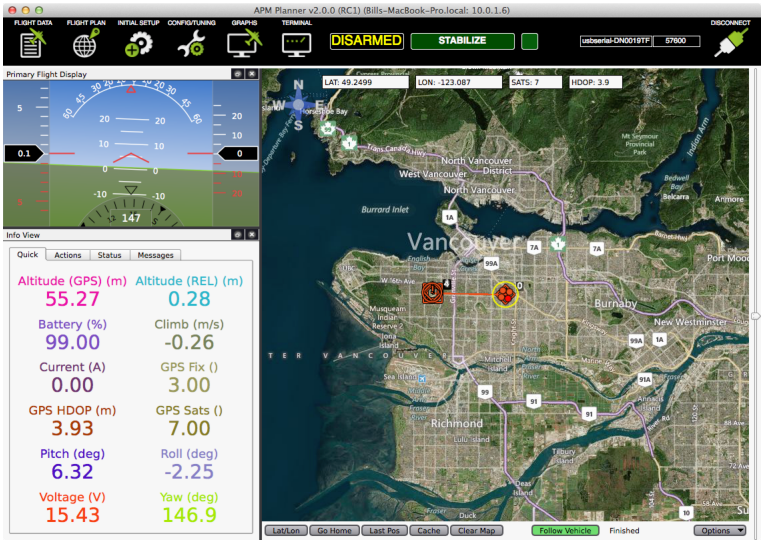


FIGURE 3.1: Mission Planner

- 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 or other maps.
- 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 is upload to the PixHawk and initial configuration of the vehicle.

3.3 MAVLink Protocol

MAVLink (Micro Air Vehicle Communication Protocol) version 1.0 [21] 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 is 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 [22].

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 his send sequence. Allows to detect packet loss.
- System ID: Identification of the SENDING system. Allows to differentiate different systems on the same network.
- Component ID: Identification of the SENDING component. Allows to differentiate different components of the same system
- Message ID: Identification of the message - the id 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)

Message detailed fields could be found here [23].

However, this protocol do not limits to a message marshalling library, MAVLink ecosystem encompass 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, a great deal of projects are based on this protocol, such as ArduPilot, ETH Flying Machine Arena or Sky-Drones [24] among others.

3.4 MAVProxy

MAVProxy is a fully-functioning GCS (group communication system) for UAV's, version 1.4.38 is the realese used in this project. The intent is for 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 100
- It is open source.
- It's 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 [14] 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 the 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) [25], is an object-oriented middleware that provides remote procedure calls, grid computing and client / server functionality developed by ZeroC 10 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 can develop a driver in C ++ for a particular robot that is embedded in the robot, on the other hand itself, 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 greater complexity without additional effort.

3.7 OpenCV

OpenCV (Open Source Computer Vision Library) [26] is a library of artificial vision developed by Intel, and now maintained by Willow Garage [27]. It is under a BSD license, which allows free use for commercial purposes. This has meant 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. In 2010 an interface for GPUs based on CUDA and OpenCL in 2012 for developed. Having been developed by Intel it provides system integrated performance primitives Intel, 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. Secondly, 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 achieve as much as possible the following features:

- Medium size, between 2-5kg weight and less than 1.5m size
- Open source software
- Reliable
- Several communication channels
- Outdoors usable

- 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 is directly reflected on manoeuvrability and endurance of the vehicle.

Many facts take part in the vehicle design and there is not a unique way to achieve similar 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 its 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: Effect of the configuration variables to be considered

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

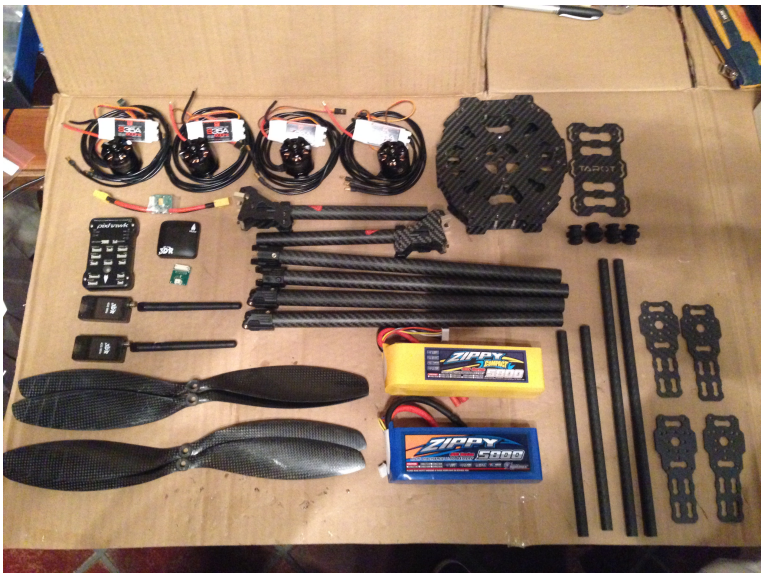


FIGURE 4.1: Pieces and componets of the vehicle

Integration process is task of placing, connecting and securing the different components, making an 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 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 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
- Lipo Battery supported: 3S - 4S

The vehicle could be divided into different subsystems connected together (Figure 4.3), each one design and built for the fullfilment of the tasks required:

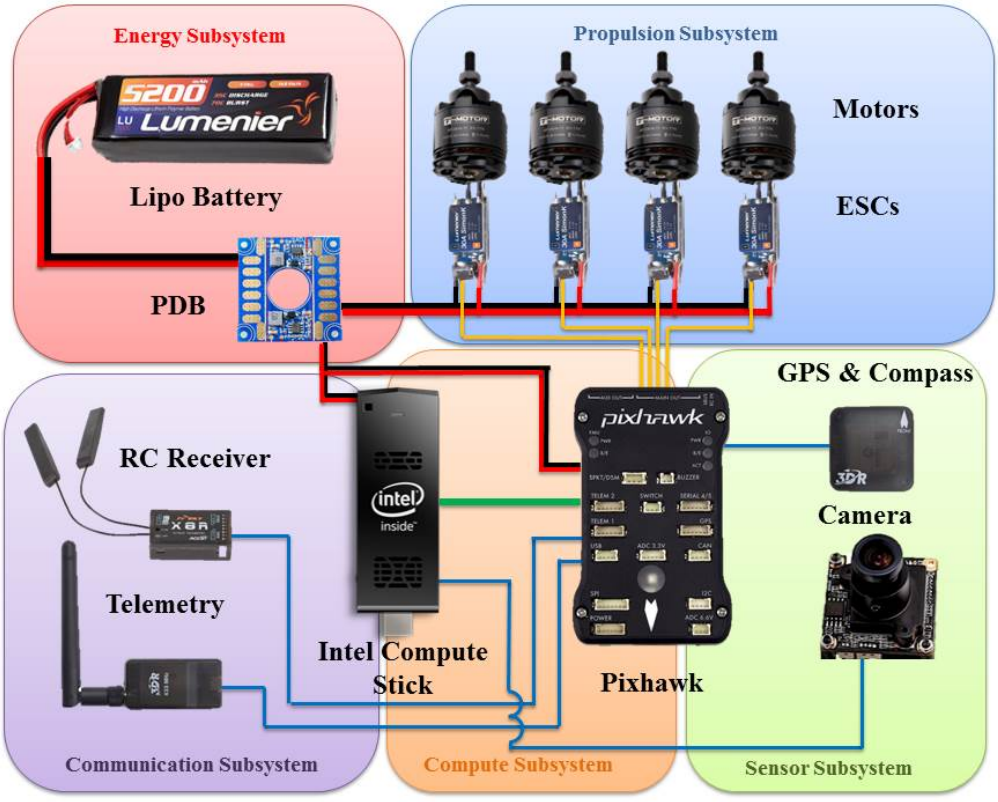


FIGURE 4.3: Hardware design

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 cloth board, with 3K hollow twill pure carbon fibre tube, a full CNC machining design with higher standards. All the carbon and CNC of the full frame weighs only 476 grams. The Full folding design makes for a high portable design 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
- 60mm rail mount
- Size motor to motor: 650mm
- Weight: 476g
- Height from ground to lower rods: 180mm
- Height from ground to top: 220mm

4.3 Energy subsystem

The energy subsystem is responsable for providing the different componets sufficient power for their correct performance. The power source of the vehicle is an unique DC source, it is the 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 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 supplies different voltages to feed different components.

4.4 Propulsion subsystem

Composed by four electric motors, propellers and ESC's (Electric Speed Controller), it is responsable for the generation of the forces and momentums for making the vehicle fly.

The ESC are conected to the PDB for energy feeding and to the Pixhawk board to signal input, and are in charged 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 propeller transmit the electric force into movement and aerodynamic forces and momentums.

T-motor MT2814 710KV 440W antigravity series

T-motor is a worldwide leading supplier of components and systems for the Aerial photograph, some Industry and commercial application in the precision drive technology sector.

Specifications:

- 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
- Idle 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

Responsable of obtaining the enviroment data, this subsystem 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 actittude and images of the surrounding and gives the information to the Pixhawk board and the ICS.

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 mini PC, it provides communication channel, via ssh synchronization, for running software components inside the on-board mini PC.
- Bluethooth: installed also inside ICS, 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 of the lower level of control, it acts as intermediary between the higher level of control, and the sensors and actuators. It stabilises the vehicle with the attitude information and compute point-to-point navigation and includes an autopilot.
- Intel Compute Stick (ICS): in charged of high level control, ICS processes the images, obtains information given by Pixhawk and makes decision which are delivered to the lower 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 exciting world of autonomous 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 delivers 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, collaboration, or cloud applications.

ICS comuter make use of Linux Ubuntu 14.04 as operating system and the high level software components used in the project is intalled in it, as JdeRobot. This computer performs the computational efford 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 configuration was performed: update firmware, calibrate some sensors (compass, gyros and accelerometers), calibrate the transmitter and establish 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 work of the motors and the radiolink for safety reasons. Then, the flight test were performed:

- Fully manual: the main objective was to verify all the components are connected and working correctly, almost without software intervention.
- Autonomous stabilisation and landing: in this case, we verified the correct working and 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 correct GPS and IMU data fusion inside the Pixhawk.

FIGURE 4.5: HoverWasp flight test

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

- 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 metheorologycal conditions.
- Chande the propeller from 14 inches to 12 inches of size, to maintaing general amount of thrust with the power source change, reducing also the motors amperage drainage.
- Elimination of analogic video communication channel.
- Change of the RC channel to improve behaviour agains electromagnetic noise.

Once the flight tests of the vehicle platform were performed, the quadcopter is ready to start the development, the next objective for the system was to be connected with JdeRobot.

Chapter 5

Software Driver

This chapter describes the software component developed for the project, it is responsible for accessing the sensors and actuators of the vehicle based on MAVLink protocol communication messages, and translate it into JdeRobot interfaces.

5.1 Design

MAVLinkServer is a driver based on MAVProxy and developed to act as a translator middleware, it is design as a JdeRobot driver in Python language. This commponent is also responssible for maintaining communication channels open and actualized, both upstream and downstream.

MAVLinkServer relies in the MAVProxy parser, the part of the program that is in charged of the management of the MAVLink messages. It establishes connection with Pixhawk autopilot, maintains communication channel operative, acquires and interprets messages and creates and sends new ones with the information proportioned by the application.

The developed code is mainly in charged 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.

These two sides of the component provide a reliable and muli-compatible driver thanks to the ICE characteristics; MAVLinkServer would connect with application written in other different languages through JdeRobot interfaces. The figure 5.1 represents an scheme of the performance of MAVLinkServer and its connection to other components.

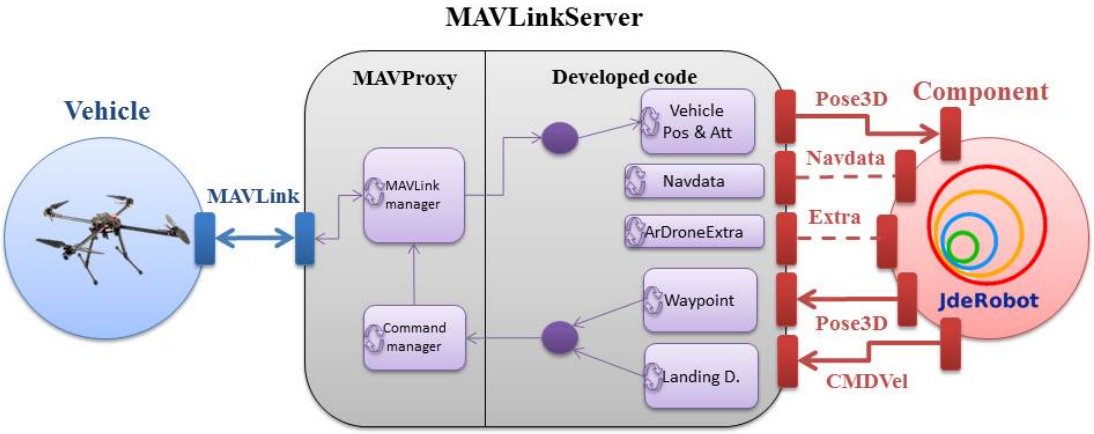


FIGURE 5.1: server Scheme

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 do not implement them. Facing this problem, MAVLinkServer was redesign to make use of a higher level of control: waypoints using Pose3D interface.

The used JdeRobot interfaces in this project are Pose3D (one for the vehicle position and attitude and other for waypoint position) and CMDVel. Note that Pose3D make use of quaternions instead of Euler angles, this is done to avoid singularity and computationally they are less intense compared to other attitude parameters 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 previously). MAVLinkServer supplies all the JdeRobot interfaces for the use of any external component. Here it is an example of Pose3D interface.

```
__author__ = 'AeroCano'

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
    self.z = _z
    self.h = _h
    self.q0 = _q0
    self.q1 = _q1
    self.q2 = _q2
    self.q3 = _q3

    print "Pose3D start"

def setPose3DData(self, data, current=None):

    lock.acquire()

    self.x = data.x
    self.y = data.y
    self.z = data.z
    self.h = data.h
    self.q0 = data.q0
    self.q1 = data.q1
    self.q2 = data.q2
    self.q3 = data.q3

    lock.release()

    return 0

def getPose3DData(self, current=None):

    time.sleep(0.05) # 20Hz (50ms) rate to tx Pose3D
```

```
        lock.acquire()

        data = jderobot.Pose3DData()
        data.x = self.x
        data.y = self.y
        data.z = self.z
        data.h = self.h
        data.q0 = self.q0
        data.q1 = self.q1
        data.q2 = self.q2
        data.q3 = self.q3

        lock.release()

        return data
```

This class inherits "jderobot.Pose3D" and the corresponding functions are defined. It can be notice 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 writing or reading errors and interfere in the component performance. With the use of the locks, 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.

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

```
__author__ = 'AeroCano'

import jderobot, time, threading

lock = threading.Lock()

class CMDVelI(jderobot.CMDVel):
```

```
def __init__(self,lx,ly,lz,ax,ay,az):

    self.linearX = lx
    self.linearY = ly
    self.linearZ = lz
    self.angularX = ax
    self.angularY = ay
    self.angularZ = az

    print "cmdvel start"

def __del__(self):

    print "cmdvel end"

def setCMDVelData(self, data, current=None):

    lock.acquire()

    self.linearX = data.linearX
    self.linearY = data.linearY
    self.linearZ = data.linearZ
    self.angularX = data.angularX
    self.angularY = data.angularY
    self.angularZ = data.angularZ

    lock.release()

    return 0

def getCMDVelData(self, current=None):

    time.sleep(0.05)  # 20Hz (50ms) rate to rx CMDVel
```

```
        lock.acquire()

        data = jderobot.CMDVelData()
        data.linearX = self.linearX
        data.linearY = self.linearY
        data.linearZ = self.linearZ
        data.angularX = self.angularX
        data.angularY = self.angularY
        data.angularZ = self.angularZ

        lock.release()

        return data
```

MAVLinkServer launches several threats to communication channel, one for each. Even Pose3D and CMDVel are the ones to be use, the driver offer the remaining interfaces (NavData and ArDroneExtra) for compatibility and future uses.

```
PH_Pose3D = Pose3DI(0,0,0,0,0,0,0,0,0) #1 to avoid indeterminations
PH_CMDVel = CMDVelI(0,0,0,0,0,0) #1 to avoid indeterminations
PH_Extra = ExtraI()
WP_Pose3D = Pose3DI(0,0,0,0,0,0,0,0,0)

##########################

#Open an ICE TX communication and leave it open in a parallel threat

PoseTheading = threading.Thread(target=openPose3DChannel, args=(PH_Pose3D,), name='Po
PoseTheading.daemon = True
PoseTheading.start()

# Open an ICE RX communication and leave it open in a parallel threat
```

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

# Open an ICE TX communication and leave it open in a parallel threat

CMDVelTheading = threading.Thread(target=openExtraChannel, args=(PH_Extra,), name='Ex
CMDVelTheading.daemon = True
CMDVelTheading.start()

# Open an ICE channel empty

CMDVelTheading = threading.Thread(target=openNavdataChannel, args=(), name='Navdata_T
CMDVelTheading.daemon = True
CMDVelTheading.start()



# Open an ICE TX communication and leave it open in a parallel threat

PoseTheading = threading.Thread(target=openPose3DChannelWP, args=(WP_Pose3D,), name='
PoseTheading.daemon = True
PoseTheading.start()

# Open an MAVLink TX communication and leave it open in a parallel threat

PoseTheading = threading.Thread(target=sendWayPoint2Vehicle, args=(WP_Pose3D,), name=
PoseTheading.start()

# Open an MAVLink TX communication and leave it open in a parallel threat

PoseTheading = threading.Thread(target=landDecision, args=(PH_CMDVel,), name='LandDec
PoseTheading.daemon = True
PoseTheading.start()
```

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

```
def openPose3DChannelWP(Pose3D):

    status = 0
    ic = None
    Pose2Rx = Pose3D #Pose3D.getPose3DData()

    try:
        ic = Ice.initialize(sys.argv)
        adapter = ic.createObjectAdapterWithEndpoints("Pose3DAdapter", "default -p 99
        object = Pose2Rx
        #print object.getPose3DData()
        adapter.add(object, ic.stringToIdentity("Pose3D"))
        adapter.activate()
        ic.waitForShutdown()

    except:
        traceback.print_exc()
     status = 1

    if ic:
     # Clean up
     try:
     ic.destroy()
     except:
         traceback.print_exc()
    status = 1

sys.exit(status)
```

%

MAVproxy is constantly handling MAVLink messages in a “while true” loop, this component takes advantage of it and complements its task with the refreshing of the sensor information required. As a high level driver, this program makes use of treated information of the vehicle, does not interfere the data fusion performed by Pixhawk and it trusts on its, more than tested, reliability. The following code is programmed after the MavProxy tasks inside the mentioned loop:

```
Rollvalue = mpstate.status.msgs['ATTITUDE'].roll    #rad
Pitchvalue = mpstate.status.msgs['ATTITUDE'].pitch  #rad
Yawvalue = mpstate.status.msgs['ATTITUDE'].yaw      #rad

# ESTIMATED: fused GPS and accelerometers
PoseLatLonHei = {}
PoseLatLonHei['lat'] = math.radians((mpstate.status.msgs['GLOBAL_POSITION_INT'].lat)/
PoseLatLonHei['lon'] = math.radians((mpstate.status.msgs['GLOBAL_POSITION_INT'].lon)/
PoseLatLonHei['hei'] = (mpstate.status.msgs['GLOBAL_POSITION_INT'].relative_alt)/1000

PH_quat = quaternion.Quaternion([Rollvalue, Pitchvalue, Yawvalue])
PH_xyz = global2cartesian(PoseLatLonHei)

#print PH_quat
#print PH_xyz

data = jderobot.Pose3DData()
data.x = PH_xyz['x']
data.y = PH_xyz['y']
data.z = PH_xyz['z']
data.h = 1
data.q0 = PH_quat.__getitem__(0)
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 threat is launched, as the ICE channel ones, which its corresponding associated functions. It makes use of the MAVProxy order management, in this case, with the command "guided lat lon hei".

```
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 component has an special function dedicated to this task and the managment of the possible situations:

```
def landDecision(CMDVel):

    while True:
        time.sleep(1)
```

```
        command = CMDVel.getCMDVelData()

    if (command.linearZ == -1):
        print'Lading decision: True'
        process_stdin('mode land')

        while (command.linearZ == -1):
            time.sleep(1)
            command = CMDVel.getCMDVelData()

        print'Target Lost, recovering trajectory'
        process_stdin('mode guided')
%
```

Finally, Pixhawk make use of different functions in order to complete the functionality of the driver. Among other, a function to change from GPS coordinates (lat, long, alt) to global Cartesian coordinates (x,y,z) or some simple functions has been defined for the correct usage of quaternions [29].

```
def global2cartesian(poseLatLonHei):

    wgs84_radius = 6378137 #meters
    wgs84_flattening = 1 - 1 / 298.257223563
    eartPerim = wgs84_radius * 2 * math.pi

    earthRadiusLon = wgs84_radius * math.cos(poseLatLonHei['lat'])/wgs84_flattening
    eartPerimLon = earthRadiusLon * 2 * math.pi

    poseXYZ = {}
    poseXYZ['x'] = poseLatLonHei['lon'] * eartPerimLon / (2*math.pi)
    poseXYZ['y'] = poseLatLonHei['lat'] * eartPerim / (2*math.pi)
    poseXYZ['z'] = poseLatLonHei['hei']
```

```
    return poseXYZ

def cartesian2global(poseXYZ):

    wgs84_radius = 6378137  # meters
    wgs84_flattening = 1 - 1 / 298.257223563
    eartPerim = wgs84_radius * 2 * math.pi
    referenceLat = 40.1912 ## Suposed to be Vehicle lattitude

    radLat = math.radians(referenceLat)
    earthRadiusLon = wgs84_radius * math.cos(radLat)/wgs84_flattening
    eartPerimLon = earthRadiusLon * 2 * math.pi

    poseLatLonHei = {}
    poseLatLonHei['lat'] = poseXYZ['y'] * 360 / eartPerim
    poseLatLonHei['lon'] = poseXYZ['x'] * 360 / eartPerimLon
    poseLatLonHei['hei'] = poseXYZ['z']

    return poseLatLonHei

def body2NED(CMDVel, Pose3D):


    q1 = [0, CMDVel.linearX, CMDVel.linearY, CMDVel.linearZ]
    q2 = [Pose3D.q0, Pose3D.q1, Pose3D.q2, Pose3D.q3]

    q1 = qNormal(q1)
    q2 = qNormal(q2)

    #rotation = q2*q1*q2'

    q2inverse = qInverse(q2)
    qtempotal = qMultiply(q1,q2inverse)
    q = qMultiply(q2,qtempotal)
```

```
    rotatedVector = q[1:len(q)] #obtain [q1,q2,q3]

    return rotatedVector

def qMultiply (q1,q2):

    q1 = qNormal(q1)
    q2 = qNormal(q2)

    # quaternion1
    w1 = q1[0]
    x1 = q1[1]
    y1 = q1[2]
    z1 = q1[3]

    #quaternion2
    w2 = q2[0]
    x2 = q2[1]
    y2 = q2[2]
    z2 = q2[3]

    w = w1*w2 - x1*x2 - y1*y2 - z1*z2
    x = w1*x2 + x1*w2 + y1*z2 - z1*y2
    y = w1*y2 + y1*w2 + z1*x2 - x1*z2
    z = w1*z2 + z1*w2 + x1*y2 - y1*x2

    q = [w,x,y,z]

    q = qNormal(q)
    return q

def qNormal(q1):
```

```
    qmodule = math.sqrt(q1[0]*q1[0] + q1[1]*q1[1] + q1[2]*q1[2] + q1[3]*q1[3])
    q = [0,0,0,0]

    if (qmodule == 0):
        qmodule = 0.000000000001

    q[0] = q1[0] / qmodule
    q[1] = q1[1] / qmodule
    q[2] = q1[2] / qmodule
    q[3] = q1[3] / qmodule

    return q

def qConjugate(q1):

    q1 = qNormal(q1)
    q = [0,0,0,0]
    q[0] = q1[0]
    q[1] = -q1[1]
    q[2] = -q1[2]
    q[3] = -q1[3]

    q = qNormal(q)
    return q

def qInverse(q1):

    q1 = qNormal(q1)
    qconjugate = qConjugate(q1)
    qmodule = math.sqrt(q1[0] * q1[0] + q1[1] * q1[1] + q1[2] * q1[2] + q1[3] * q1[3]

    if (qmodule == 0):
        qmodule = 0.000000000001
```

```
q = [0,0,0,0]
q[0] = qconjugate[0] / qmodule
q[1] = qconjugate[1] / qmodule
q[2] = qconjugate[2] / qmodule
q[3] = qconjugate[3] / qmodule

q = qNormal(q)
return q
```

5.3 Information pipeline

MAVLinkServer is a multy-threat component, the flow of the information and the operation that the driver pursues is the following tasks path:

- The driver starts running all the different threads, opening its communication channel and defining the information that would be in each one. This project make use of two communication channels based on Pose3D, one for vehicle position and attitude and other for waypoints definition; and a CMDVel channel for landing decision.
- MAVLink Messages from Pixhawk came into the driver, and they are been interpreted in order to acquire the position and attitude information.
- Acquired information is treated to transform it into JdeRobot standards (Pose3D). Latitude and longitude are transformed into global xy coordinates, using WGS84 as earth model, and Euler angles attitude is transformed into quaternions system.
- Pose3D is written in the corresponding local classes in a controlled way, making use of the lock system.
- Classes are published in the corresponding ICE channels as the threats are running, to other components to access to it.
- Commands are received through CMDVel and waypoints Pose3D channels, ones external decision has been published into the ICE channel.

- Information is extracted from the corresponding classes with the mentioned lock system.
- Information is treated and a reference frame change is performed in order to be synchronised with Pixhawk. Body referenced waypoints are transformed into GPS coordinates system making use of the quaternion functions designed.
- Waypoints and landing decision are delivered to MAVProxy command manager.
- Commands and translated into a MAVLink message and sent to the vehicle.

It can be seen that each thread has a task and each one does not have the same workload, for this reason the times of the control loops of each wire are different. However the threats are working and the driver and even they 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 the component, it is necessary to carry on some experimental test.

First verifying the correct adquisition and trasformation of the sensors data, the correct performance of the different threads launched and publication of the data throught ICE communication channels. To do so, The project make use of a well-tested tool that belogs to JdeRobot, Uav-viewer developed by Martin Florido.

The second experiment was to verify the correct adquisition of the commands provided by other component and the right delivery of them to the vehicle via MAVProxy and MAVLink.

During these experiment some bugs were detected, belonging to the implementation of the velocity commands on MAVProxy. As they were completely out of the scope of this project, MAVLinkServer was redesign to accomplish waypoint orders, as mentioned before.

Chapter 6

Application component

Once presented in the previous chapter the software infrastructure to access the sensors and actuators of the UAV, in this chapter the developed application that implements autonomous predefined navigation and visual navigation techniques will be described. The visual control is a control system in which the feedback obtained from a video camera is used as source of information.

6.1 Behaviour design

This application aims to perform a target search mission around a designated localisation; ones target has been spotted, loiter over its position and land in the area. This mission use the information supplied by the vehicle 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-position-known" probabilistic method which holds the concept that a target is more probable to be found around the last position known, if an established trajectory or movement is not recognise. The scan method chosen in this project is the spiral, in which the vehicle describes a squared spiral trajectory starting in the middle and covering further distance each revolution.

MAVLinkClient has four different functions: First one is to design and manage the mission and establish trajectory, the second one is to calculate the next waypoints and

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

MAVLinkClient is a JdeRobot component completely developed by 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, produce its own decisions and perform and autonomous task. The figure 6.1 represents an scheme of the performance of the component.

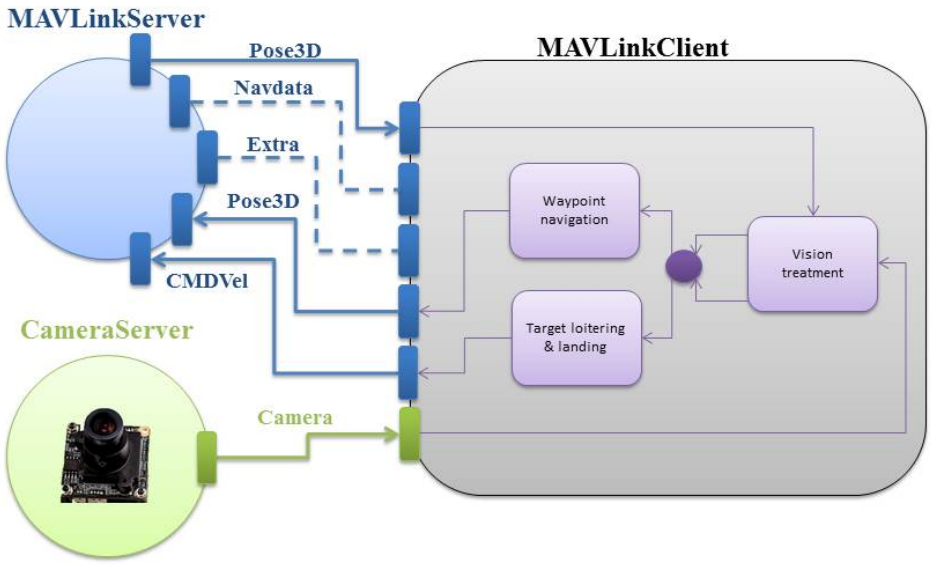


FIGURE 6.1: Client Scheme

6.2 Implementation

MAVLinkClient is a multy-thread component that make use of the available Ice channels and send diferrent command depending on the information provided by the driver and the camera. This component also needs an additional channel that sould be launch appart from the ones supplied by the driver: cameraserver, a jderobot tool. 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='ClientPHPos
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='Cl
PoseTheading.daemon = True
PoseTheading.start()



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

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

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

As in the driver component, each threat make use of its own function were the ICE configuration is performed and the handle of the information 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
```

```
        if not datos:
            raise RuntimeError("Invalid proxy")



        while True:
            time.sleep(0.02)
            data = datos.getPose3DData()
            Pose3D.setPose3DData(data)
            #print Pose3D

    except:
        traceback.print_exc()
        status = 1

    if ic:
        # Clean up
        try:
            ic.destroy()
        except:
            traceback.print_exc()
            status = 1

def txPose3DWP2Server(Pose3D):

    status = 0
    ic = None
    try:
        ic = Ice.initialize(sys.argv)
        base = ic.stringToProxy("Pose3D:default -p 9994")
        datos = jderobot.Pose3DPrx.checkedCast(base)
        #print datos
        if not datos:
            raise RuntimeError("Invalid proxy")
```

```
        while True:
            time.sleep(0.02)

            Pose3D2send = Pose3D.getPose3DData()
            datos.setPose3DData(Pose3D2send)

            # print Pose3D

    except:
        traceback.print_exc()
        status = 1

    if ic:
        # Clean up
        try:
            ic.destroy()
        except:
            traceback.print_exc()
            status = 1

def txCMDVel2Server(CMDVel):

    status = 0
    ic = None

    try:
        ic = Ice.initialize(sys.argv)
        base = ic.stringToProxy("CMDVel:default -p 9997")
        datos = jderobot.CMDVelPrx.checkedCast(base)
        #print datos

        if not datos:
            raise RuntimeError("Invalid proxy")
```

```
        while True:
            time.sleep(0.05)
            CMDVel2send = CMDVel.getCMDVelData()
            datos.setCMDVelData(CMDVel2send)
            #print CMDVel2send


    except:
        traceback.print_exc()
        status = 1

    if ic:
        # Clean up
        try:
            ic.destroy()
        except:
            traceback.print_exc()
            status = 1


def rxCamera():

    status = 0
    ic = None

    try:
        ic = Ice.initialize(sys.argv)
        base = ic.stringToProxy("Camera:default -p 9999")
        datos = jderobot.CameraPrx.checkedCast(base)
        # print datos
        if not datos:
            raise RuntimeError("Invalid proxy")
```

```
        while True:
            time.sleep(0.5)
            global PH_Camera
            global CameraHeight
            global CameraWidth
            cameraImage = datos.getImageData("RGB8")
            CameraHeight = cameraImage.description.height
            CameraWidth = cameraImage.description.width

            lock.acquire()
            PH_Camera = np.frombuffer(cameraImage.pixelData, dtype=np.uint8)
            PH_Camera.shape = CameraHeight, CameraWidth, 3
            lock.release()
            # cv2.imshow('PH_Camera', PH_Camera)

    except:
            traceback.print_exc()
            status = 1

    if ic:
        # Clean up
        try:
            ic.destroy()
        except:
            traceback.print_exc()
            status = 1
```

6.3 Scan navigation

The vehicle should scan the selected area, so the first step is define the trajectory. The spiral desired is defined by four values:

- Centre point: (x,y) in global coordinates in metres

- Mission height in metres.
- Scan distance: distance between parallel sides of the spiral in metres.
- Number of spins: number of revolution of the spiral before restarting trajectory.

The next function design the trajectory based on the information given and returns a list of waypoints of the trajectory, in order to the remaining application to use and modify the trajectory:

```
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
```

This function is in charged 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 and the vehicle. A waypoint is consider 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

    if (dist <= ReachedDist):
        trajectory = trajectory[1:len(trajectory)] #pop
        print "Waypoint reached"

    return trajectory
```

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

```
    print 'Searching for target'

    # command, updatedTrajectory = nextWaypointCMDVel(xyz, trajectory)
    updatedTrajectory = nextWaypointPose3D(vehicleXYZ, trajectory)
    trajectory = updatedTrajectory

    #restart trajectory when finish
    if (len(trajectory) == 0):
        trajectory = DefTrajectory
        print "Trajectory restarted:"
```

```
        print trajectory

    #send waypoint to server
    wayPoint = jderobot.Pose3DData()
    wayPoint.x = trajectory[0][0]
    wayPoint.y = trajectory[0][1]
    wayPoint.z = trajectory[0][2]
    WP_Pose3D.setPose3DData(wayPoint)
    # print wayPoint
```

If during the trajectory follow mode, the target is found, the vehicle change its mission and tries to loiter over it; using the camera and vision treatment to identify the relative position between the vehicle and the target.

6.4 Perception

The detection of an object and the decision of found it is the core funtion of these component, and its decision decides the mode sent to the driver. This code is inserted inside a the loop of ejecution of the component.

```
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, sm
OrangeCenter, OrangeArea, OrangeFound = detection(Image2Analyze, hminO, hmaxO, sminO,
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]
    targetBuffer.append(True)
else:
    targetBuffer.append(False)

targetBufferData = targetBuffer.get()

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

Image detection can be done following multiple ways. This project has chosen to identify objects from its colour, his attribute detection has some disadvantages compared to other detection methods. The biggest drawback is the light of the environment where the robot is and the shadows that can project other objects to 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 it was decided to change the RGB colour space to HSV, which although does not end solve the problem of light, is robust enough for detecting an object in an environment of changing light, as long as the changes light are not very sharp.

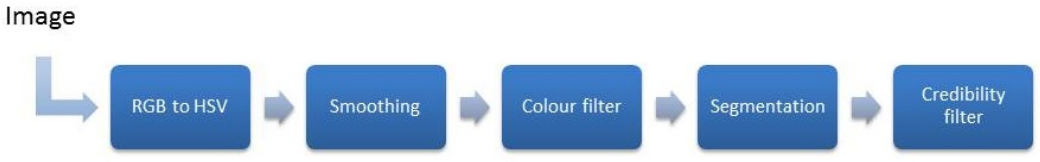


FIGURE 6.2: *Pipeline* of the perception system of `object_tracking`

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

```
def detection(image, hmin, hmax, smin, smax, vmin, vmax):
    # ------- Thresold Image -------#

    rawImage = image

    ksize = (5, 5)
    sigma = 9
    gaussImage = cv2.GaussianBlur(rawImage, ksize, sigma)

    hsvImage = cv2.cvtColor(gaussImage, cv2.COLOR_RGB2HSV)

    matrixMin = np.array([hmin, smin, vmin])
    matrixMax = np.array([hmax, smax, vmax])

    filteredImage = cv2.inRange(hsvImage, matrixMin, matrixMax)

    # ------- Detect Object -------#

    center = []
    area = 0
    center = (0,0)
    colorFound = False


    detImage = rawImage

    contour, hierarchy = cv2.findContours(filteredImage, cv2.RETR_EXTERNAL, cv2.CHAIN

    if len(contour) > 0:
        contourdx = -1
        cv2.drawContours(detImage, contour, contourdx, (255, 255, 0))

    for i in range(len(contour)):
        contarray = contour[i]
```

```
        epsilon = 5
        closed = True
        approxCurve = cv2.approxPolyDP(contarray, epsilon, closed)

        rectangle = cv2.boundingRect(approxCurve)
        rectX, rectY, rectW, rectH = rectangle

        if rectW < maxTargetLado and rectW > minTargetLado and rectH < maxTargetLado a
            myRectangle = rectangle
        else:
            myRectangle = 0, 0, 0, 0

        myRectX, myRectY, myRectW, myRectH = myRectangle
        myPoint1 = myRectX, myRectY
        myPoint2 = myRectX + myRectW, myRectY + myRectH
        center = myRectX + (myRectW / 2), myRectY + (myRectH / 2)
        area = myRectW * myRectH

        # cv2.rectangle(detImage,myPoint1,myPoint2,(255,0,0),2)
        cv2.circle(detImage, center, 1, (255, 0, 0), 2)

        minTargetArea = minTargetLado * minTargetLado

        if (area > minTargetArea)and(center != (0,0)):
            colorFound = True
        else:
            colorFound = False

    return center, area, colorFound
```

As a result, the component obtain a positive or negative identification of the object, and in the case, the horizontal relative position between the vehicle and the target. Each

step of the function have a lot of computational efford, the detection is the heaviest part of the component. the detection is perform twice, one for each colour og 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 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 very changing environments, it is convenient to use another colour space more robust to changes in light, for example the HSV space, represented in the figure 6.3.

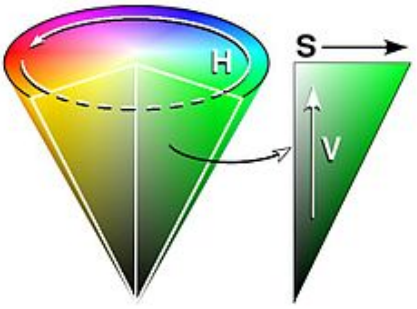


FIGURE 6.3: Cone colors space HSV

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 has values ranging from 0 to 100.

To changes in brightness changes 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 dye and a saturation, 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 processing stage image, there are some techniques that improve some aspect of the image, as smoothing technique. Noise, signal degradation, errors in the recruitment process or image transmission or lack of uniform illumination are image characteristics that can interfere the object analysis. The perception system component implements smoothing technique Gaussian Blur, which is the result of convoluting an image with the Gaussian function (figure 6.4):

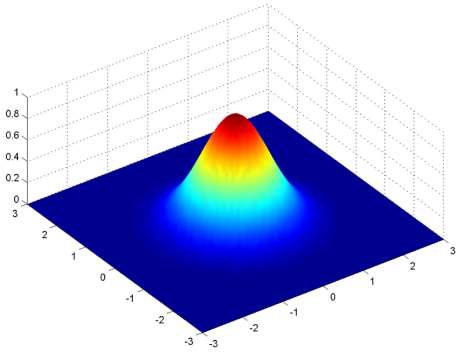


FIGURE 6.4: Gauss function in 2D

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 Gauss) and adjacent pixels receive a smaller weight depending on the distance from the original pixel increase. 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 6.5.

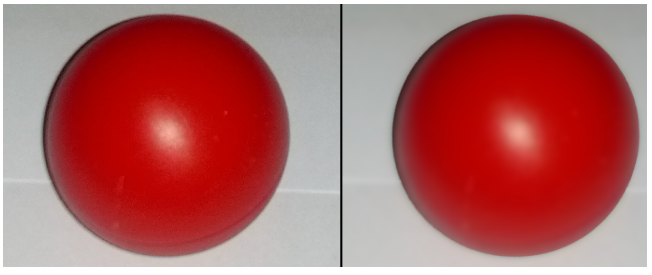


FIGURE 6.5: *Gassuian Blur*

6.4.3 Colour Filtering

The choice of colour is an important aspect object detection, a vivid colour will be easily detectable, colour with more muted tones will be more difficult to detect. The object tracking application allows us to select the colours that we want to identify the object, defining their HSV values. By applying thresholds to a colour image 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 6.6. The result is the binary image. inRange() function, belonging to OpenCV, apply thresholds to the image.

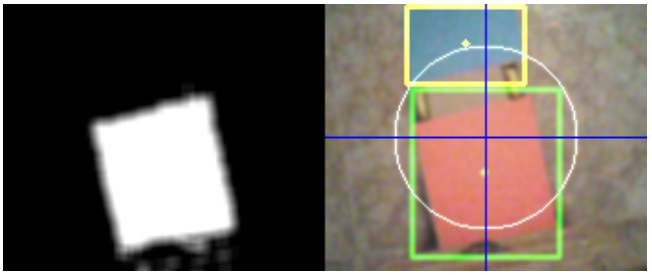


FIGURE 6.6: Image thresholding

6.4.4 Segmentation

The purpose of object detection is to obtain the centre coordinates 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 system of perception component performs a series of steps:

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

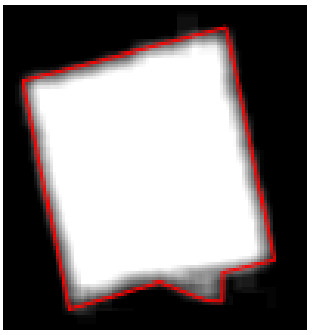


FIGURE 6.7: Contours detected with the function *findContours()*

- 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 within it all the points indicated as a parameter to the function.
- After calculating the coordinates of the centre of the rectangle, they will be in the reference system of the image itself, so you have to make a change to the reference system used in object tracking, centred image.

6.4.5 Filter

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 these changes do not mean that the object is actually moved. For example, a change in ambient light cause the sensing system detects an object different size different from the size of the same object previously detected. To combat against this behaviour and the application has implemented a "Buffer" for making a "credibility filter". This filter takes into account the previously captured images (a determined number samples) and store a positive or negative variable in order to determine if the target have been identify or not.

When the number of positive identification is higher than an established threshold, a confirmation is generated and the code change it task from following trajectory to loitering control.

The number of samples and the threshold gives the filter its behaviour and change the response and the consistency of the whole program.

6.5 Loitering Control and Landing

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

Loitering control will take as reference the image the central point and the difference of it with the point estimated by the perceptual system 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 error using a functions which depends on the height pf the vehicle using some functions.

```
def global2cartesian(poseLatLonHei):

    wgs84_radius = 6378137 #meters
    wgs84_flattening = 1 - 1 / 298.257223563
    eartPerim = wgs84_radius * 2 * math.pi

    earthRadiusLon = wgs84_radius * math.cos(poseLatLonHei['lat'])/wgs84_flattening
    eartPerimLon = earthRadiusLon * 2 * math.pi

    poseXYZ = {}
    poseXYZ['x'] = poseLatLonHei['lon'] * eartPerimLon / (2*math.pi)
    poseXYZ['y'] = poseLatLonHei['lat'] * eartPerim / (2*math.pi)
    poseXYZ['z'] = poseLatLonHei['hei']
    return poseXYZ

def pixel2metres(pixelXY, pixelNum, height, fov):

    height = MissionHeight #only for testing
```

```
    vehicleAngle = fov/2
    groundAngle = (math.pi/2) - vehicleAngle
    hipoten = height / math.sin(groundAngle)
    groundSide = math.sqrt((hipoten*hipoten) - (height*height))

    metersXpixel = 2*groundSide / pixelNum

    metresXY = [metersXpixel*pixelXY[0] , metersXpixel*pixelXY[1]]

    return metresXY

def frameChange2D(vector, angle):

    rotatedX = vector[0]*math.cos(angle) + vector[1]*math.sin(angle)
    rotatedY = vector[1]*math.cos(angle) - vector[0]*math.sin(angle)

    rotatedVector = (rotatedX,rotatedY)

    return rotatedVector
```

With this information, the component estimates the position of the target and generates and send to the server 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 precision is necessary to be defined. the vehicle 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 precision, the component send a landing order to the server via CMDVel Ice channel.

The landing decision is irrevocable from the program point of view, however, the component still processing images and making decision for recover mission in case of manually landing decision suspension.

6.6 Information pipeline

MAVLinkClient is a multy-threat component at the communication level, nevertheless, the information and main task if a lineal decision making line following this path:

- MAVLinkClient starts running all the different threads, receiving information of the communication channel and defining the information that would besent to the driver.
- The component define the trajectory and return a list of waypoints.
- The images are updated and analized to detect the target
- Buffer store the positive or negative detections and elavorates 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 if the target is calculated and try to cought it.
- When the target position is reached, landing order 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 define the success of the project and the validation of it a a whole autonomous system.

The first test is perform to validate the detection algorithm and filter cariables set-up. The figure 6.8 shows an example of the test carried out, where it can be notice that the function detect perfectly the colours of the target and calculate its central point.

The risk of testing the component directly on the vehicle is high, thats why the project make use of grazebo simulator for testing the code. In the figure 6.8 the simulated vehicle is receiving the information provided by the component with the artificial vision algorithm being executing at the same time.

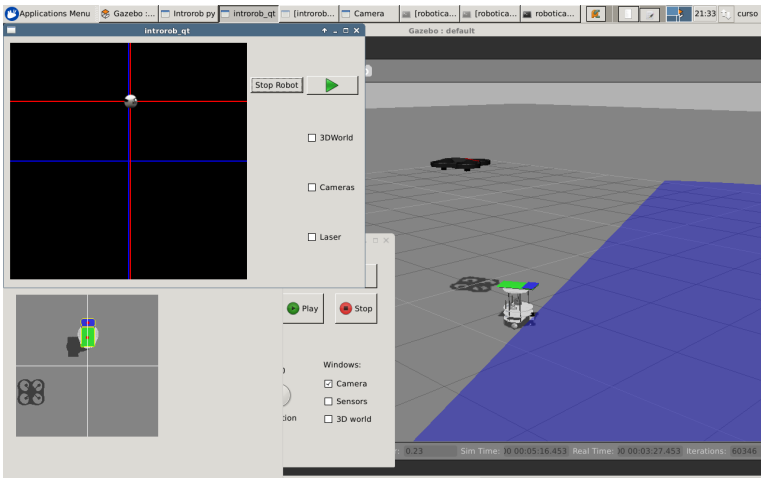


FIGURE 6.8: MAVLinkClient simulation test

With the succes of the previous tests, it was time to perform the final experiment. MAVLinkServer, cameraserver and MAVLinkClient were launched in the ICS via ssh with and external computer to execute the mission. The vehicle scans the area following the desired trajectory, idenifying the target and making the decision of landing in the surroundings; Completing the mission succesfully.

This is the final step of this project.

Chapter 7

Conclusions and Next Steps

In this chapter, a critical analysis of the outcomes development in the previous chapters is done to obtain conclusions about the vehicle design and construction and the development of the driver and the application. Finally, a rating of the future works or next steps that can take this Project as a benchmark will be done.

7.1 Conclusions

The primary objective of this proyect is to design, buld 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 have been divide into three main tasks: Design and construction of an aerial vehicle, develop the JdeRobot driver for the used platform and program a component for the management of the mission; each of one described in their corresponding chapter.

- The HoverWasp vehicle desgined and built from scratch have reached the requirement stablished. It is a reliable and useful platform in which new development on the UAV sector could be carried on. It is a custom vehicle thought to be similar than a huge variety of UAV's in order to other project to take advantege on the work of these project. Multirotor dynamics and electronics are the main

knowledges adquired in this part, as well as the engineering efford done for its design.

- MAVLinkServer component is a sofware driver that could be use to connect any MAVLink based vehicle to the JdeRobot software. The communication channels and interfaced have been specially used following an intuitive criteria for its future use. Programming capabilities, the undertanding and use of quaternions and cartography have been the learnt topics in this section.

- MAVLinkClient is the component spetially thought for this project. It make use of high level point-to-point control of the UAV to complete an autonomous mission. Computer vision and its theory has been the determining fact in thr creation of this component

Given the result of the previous chapters, it can be said that the objective have been achieve successfully; all the phases of the project have been covered , from requirements analysis to final validation. In addition, the progess have been live documented in the project webpage [15] and code updated to the repositry [16]. The software developed in this project have been updated to the JdeRobot, being proud contributor of the development of the URJC robotics group.

Along the development of the proyect, there have been a huge amount of knowledge that it have been necessary to be adquired. The field of UAV's gathers information that comes from different areas; programming, telecominications, 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 enginerring learnt theory. It is a crucial step for a prepared ingeneer have the oportunity to experiment and face real world problems, that at the end, is the assingment for which we have been trained.

7.2 Next Steps

The value of this project is not only the goals achieved or the efford made for meet them; there exist a greater value on the amount of development that can be carried on using

this project as a benchmark.

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

Also, the vehicle takking off is too agresive for the safe use of it. An smoother control would benefit the platform and would almost cancel the human intervention in the missions carried on.

Other improvement would be design a graphical interface for MAVLinkClient, refining it use and making it more confortable and visual for wider public.

Moreover, the change of the camera, from visible spectrum to heat detection cam, would give a concrete search and rescue application of the software developed on this project and giving the society an effective solution for a real problem.

Bibliography

[1] Stanford Racing Team. Stanley team website. http://cs.stanford.edu/group/roadrunner/. [Online].

[2] US Airforce Research Laboratory. Wright-patterson airforce base. http://www.wpafb.af.mil/AFRL/. [Online].

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

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

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

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

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

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

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

[10] 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).

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

[27] Willow Garage. Hardware and software for personal robotic applications. https://www.willowgarage.com/. [Online].

[28] OpenSSH. Premier conectivity tool for remote loging website. http://www.openssh.com/. [Online].

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