Universidad de Alcalá

Escuela Politécnica Superior

MÁSTER EN INGENIERÍA INDUSTRIAL

Trabajo Fin de Máster

Vision-based SLAM for the aerial robot ErleCopter

Autor: Guillermo Patiño González

Tutor: María Elena López Guillén

2018

UNIVERSIDAD DE ALCALÁ
Escuela Politécnica Superior

MÁSTER EN INGENIERÍA INDUSTRIAL

Trabajo Fin de Máster

SLAM based on vision for the aerial robot ErleCopter

Autor: Guillermo Patiño González

Tutor: María Elena López Guillén

Agradecimientos

Me gustaría en primer lugar agradecer a mi familia, especialmente a mis padres y a mi hermana, el apoyo recibido durante todas las etapas de mi vida. El esfuerzo que siempre han realizado por darme lo mejor que podían no tiene palabras.

A mis compañeros y profesores, tanto de la Universidad de Alcalá como de la Mälardalens University, que han hecho de este camino una experiencia que repetiría sin dudar. En especial me gustaría mencionar a mi tutora Elena, por abrirme un camino de la ingeniería que hace que me sienta realizado. Quisiera agradecerle la oportunidad de hacer este proyecto y todas las cosas que he aprendido gracias a ello, así como su entrega como profesora.

Por último a mis amigos más cercanos y a Rebeca, por estar siempre para todo en estos últimos años.

Contents Index

1. INTRODUCTION 21

1.1 UAV's State of the Art 23

1.2 Use of cameras 24

1.3 The raise of the SLAM techniques 25

1.4 Objectives 31

1.5 Work Structure 32

2. TOOLS 35

2.1. Robotic Operating System 37

2.2. Gazebo 40

2.3 ErleCopter 43

2.3.1 Flight modes 44

2.3.2.1 .Hardware 45

2.3.3 Erle-Brain 46

3. MONOCULAR VISUAL SLAM 49

3.1. Introduction 51

3.2. VSLAM algorithms 52

3.2.1 PTAM 53

3.2.1.1 Introduction 53

3.2.1.2 PTAM Algorithm 54

3.2.1.3 Simulation Implementation and launch file 57

3.2.1.4 Real implementation within Erlecopter and launch file 60

3.2.1.5 Results obtained 61

3.2.2 ORB-SLAM 66

3.2.2.1 Introduction 66

3.2.2.2 ORB-SLAM diagram 67

3.2.2.3 Algorithm ORB-SLAM 69

3.2.2.4 Simulation implementation and launch 71

3.2.2.5 Real implementation within Erlecopter and launch file 73

3.2.2.6 Results obtained 75

3.2.3 LSD-SLAM 80

3.2.3.1 Introduction 80

3.2.3.2 Algorithm LSD-SLAM 80

3.2.3.3 Simulation implementation and launch file 82
3.2.3.4 Real implementation within Erlecopter and launch file 83
3.2.3.5 Results obtained 84
3.3. Results 87
3.3.1 Simulation Results 87
3.3.2 Real results 89
4. EXTENDED KALMAN FILTER 91
4.1. Introduction 93
4.2. Extended Kalman Filter algorithm 93
4.2.1 Prediction Model 94
4.2.2 Observation Model 96
4.2.2.1 Scaling Factor Problem 97
4.3. Results 100
5. PID CONTROLLER 107
5.1. Introduction 109
5.2. Implementation and results 109
6. CONCLUSIONS AND FUTURE WORK 117
6.1. Conclusions 119
6.2. Future Work 119
7. USER'S MANUAL 121
7.1. Downloading the necessary tools 123
7.2. Launching the ErleCopter world in Gazebo 123
7.3. Setting the world 124
7.4. Arming the ErleCopter 124
7.5. Publishing a topic and moving the quadcopter 125
7.6. Taking off automatically. 125
7.7. Launching the PID controller 126
7.8. Launching the EKF 126
7.9. Creating a package 126
8. SPECIFICATIONS 129
8.1. Hardware Specifications 131
8.2. Software Specifications 131
9. BUDGET 133
9.1. Equipment cost 135
9.2. Professional fees 135

9.3. Total cost 136

10. BIBLIOGRAPHY 139

Figures Index

Figure 1. 1 Barrucada UAV. 23
Figure 1. 2. Military UAV. Boeing. 23
Figure 1. 3. AR Drone. Four motor model. 24
Figure 1. 4. STABLEY and JUNIOR car 26
Figure 1. 5. Experimental Platform with on board computation 26
Figure 1. 6. Architecture diagram showing the software modules 27
Figure 1. 7. Map generated by flying. 27
Figure 1. 8. Fixed wing experimental platform flying indoors 28
Figure 1. 9. The trajectory flown by the vehicle is shown by the red, green, and blue axes 29
Figure 1. 10. Block diagram illustrating the operational steps of the monocular vision system 29
Figure 1. 11. Experimental results of the proposed ranging and SLAM algorithm. 30
Figure 1. 12. Cartesian (x; y; z) position of the MAV in a hallway 30
Figure 1. 13. Saint Vertigo, the autonomous MAV helicopter 31

Figure 2. 1. ROS Icon 37
Figure 2. 2. ROS file System Level. 38
Figure 2. 3. ROS computation graph Level 39
Figure 2. 4. Gazebo software platform with a Turtlebot. 41
Figure 2. 5. ErleCopter model in Gazebo downloaded in ErleRobotics. 42
Figure 2. 6. Hardware parts of the ErleCopter. 43
Figure 2. 7. Components that model the ErleCopter. 45
Figure 2. 8. Defined components of the ErleCopter. 46
Figure 2. 9. Erlebrain 2 Units. 47
Figure 2. 10. Communication Units of the ErleBrain 2. 48

Figure 3. 1 PTAM Keypoints tracked. Interface. 54
Figure 3. 2 PTAM Block Diagram, ilustrating the monocular worklow steps. 56
Figure 3. 3. Erlecopter monocular camera open in the Ubuntu SO. 58
Figure 3. 4. Gazebo simulator with the Erlecopter model with PTAM running on board. 59
Figure 3. 5. PTAM Map pose estimation view. 59
Figure 3. 6. Erelcopter monocular camera launched in Ubuntu. 61
Figure 3. 7. Path 1 built for the monocular test. 62
Figure 3. 8. Path 2 built for the monocular test. 62
Figure 3. 9. Path 3 built for the monocular test. 63
Figure 3. 10. Results PTAM path 1 X-Y view. 63
Figure 3. 11.Results PTAM path 1 3D view. 64
Figure 3. 12. Results PTAM path 2 X-Y view. 64
Figure 3. 13. Results PTAM path 2 3D view. 65
Figure 3. 14. Results PTAM path 3 X-Y view. 65
Figure 3. 15. Results PTAM path 3 3D view. 66
Figure 3. 16. ORB-SLAM Block diagram illustrating the workflow steps. 67
Figure 3. 17. ORB-SLAM launched in Gazebo with the Erlecopter Model. 73
Figure 3. 18. ORB – SLAM launched on the real robotic platform. 74
Figure 3. 19. ORB-SLAM current frame window, with the detected Keyframes. 75
Figure 3. 20. ORB-SLAM tracking and mapping window. 76

Figure 3. 21. Results ORB-SLAM path 1 3D view. 77
Figure 3. 22. Results ORB-SLAM path 2 X-Y view. 77
Figure 3. 23. Results ORB-SLAM path 2 3D view. 78
Figure 3. 24. Results ORB-SLAM path 2 X-Y view. 78
Figure 3. 25. Results ORB-SLAM path 3 3D view. 79
Figure 3. 26. Results ORB-SLAM path 3 X-Y view. 79
Figure 3. 27. LSD-SLAM Block Diagram illustrating the workflow steps. 80
Figure 3. 28. LSD-SLAM tracking and mapping window. 81
Figure 3. 29. LSD-SLAM launched in Gazebo simulator with the Erlecopter model. 82
Figure 3. 30. LSD-SLAM launched on the real robotic platform. 83
Figure 3. 31. Results LSD-SLAM path 1 3D view. 84
Figure 3. 32. Results LSD-SLAM path 1 X-Y view. 85
Figure 3. 33. Results LSD-SLAM path 2 3D view. 85
Figure 3. 34. Results LSD-SLAM path 2 X-Y view. 86
Figure 3. 35. Results LSD-SLAM path 3 3D view. 86
Figure 3. 36. Results LSD-SLAM Path 3 X-Y view. 87
Figure 3. 37. Real tracking of the 3 vSLAm algorithms on the Erlecopter. X-Y view. 90

Figure 4. 1. Block Diagram of the EKF. Prediction and correction workflow. 94
Figure 4. 2. Yaw, Pitch and Roll angles in the world frame. 95
Figure 4. 3. Relationship between pixel scale and real scale in monocular cameras. 97
Figure 4. 4. First trial . Relationship between the sonar height and the PTAM predicted height. 98
Figure 4. 5. Second trial . Relationship between the sonar height and the PTAM predicted height. 99
Figure 4. 6. Third trial . Relationship between the sonar height and the PTAM predicted height. 99
Figure 4. 7. Results EKF path 1 X-Y view. 100
Figure 4. 8. Results EKF path 1 3D view. 100
Figure 4. 9. Results EKF path 2 X-Y view. 101
Figure 4. 10. Results EKF path 2 3D view. 101
Figure 4. 11. Results EKF path 3 X-Y view. 102
Figure 4. 12. Results EKF path 3 3D view. 102
Figure 4. 13. 3D single-scope view Ground-Truth, EKF and PTAM 103
Figure 4. 14. X-Y single scope view Ground-Truth, EKF and PTAM. 104

Figure 5. 1. Scheme of the local frame of the quadcopter 110
Figure 5. 2. PID controller block diagram. 111
Figure 5. 3. Results PID controller Z position. 113
Figure 5. 4. Results PID controller Y position. 113
Figure 5. 5. Results PID controller X position. 114
Figure 5. 6. Results P controller yaw position. 114
Figure 5. 7. Components of the navigation system. 116

Figure 7. 1. Ardupilot terminal. 124
Figure 7. 2. Erelcopter Console. 124

Tables Index

Table 2. 1. Features of the ErleCopter 44

Table 3. 1. Errors obtained for each vSLAM algorithm. 88

Table 4. 1. Errors obtained for the EKF. 103
Table 4. 2. Average Errors from EKF and PTAM 103
Table 4. 3. Errors obtained from EKF and PTAM in the single scope trial................................ 104

Table 5. 1. Constants of the controllers. 115

Table 9. 1. Equipment cost. 135
Table 9. 2. Professional fees. 136
Table 9. 3. Total cost. 136

Resumen

El objetivo principal de este trabajo, es la implementación de distintos tipos de algoritmos SLAM (mapeado y localización simultáneos) de visión monocular en el robot aéreo ErleCopter, empleando la plataforma software ROS (Robotic Operating System).

Para ello se han escogido un conjunto de tres algoritmos ampliamente utilizados en el campo de la visión artificial: PTAM, ORB-SLAM y LSD-SLAM. Así se llevará a cabo un estudio del funcionamiento de los mismos en el ErleCopter.

Además empleando dichos algoritmos, y fusionando la información extraída por estos con la información de otros sensores presentes en la plataforma robótica, se realizará un EKF (Extended Kalman Filter), de forma que podamos predecir la localización del robot de una manera más exacta en entornos interiores, ante la ausencia de sistemas GPS.

Para comprobar el funcionamiento del sistema se empleará la plataforma de simulación robótica Gazebo.

Por último se realizarán pruebas con el robot real, de forma que podamos observar y extraer conclusiones del funcionamiento de estos algoritmos sobre el propio ErleCopter.

Palabras clave: SLAM, visión, ROS, robótica, EKF, robot aéreo.

Abstract

The main objective of this thesis is the implementation of different SLAM (Simultaneous Localization and Mapping) algorithms within the aerial robot ErleCopter, using the software platform ROS (Robotic Operating System).

To do so, a bunch of three widely known and used algorithms in the field of the artificial vision have been chosen: PTAM, ORB-SLAM y LSD-SALM. So a study of the performance of such algorithms will be carried out in this way.

Besides, working with such algorithms and fusing their information with the one obtained by other sensors existing within the robotic platform, an EKF (Extended Kalman Filter) will be carried out, in order to localize the robot more accurately in indoor environments, given the lack of GPS.

To test the performance of the system, the robotic platform Gazebo will be used in this project.

Finally tests will be made with the real robot, in order to observe and draw conclusions from the performance of these algorithms within the ErleCopter itself.

Key words: SLAM, vision, ROS, robotics, EKF, aerial robot.

CHAPTER 1

INTRODUCTION

1.1 UAV’s State of the Art

In this chapter it will be exposed the amount of different projects based on SLAM techniques, and of course VSALM techniques will be highlighted. It is important to explain how quadcopters and algorithms have been developed in the last years, so the importance of this project can be understood.

Nowadays the main part of the population has the feeling that these vehicles have become part of our daily lives, although the fact is that at the beginning these quadcopters, now considered robots or even toys were made for war purposes, where a particular military area was secured or attack a conflicted area without any human help.

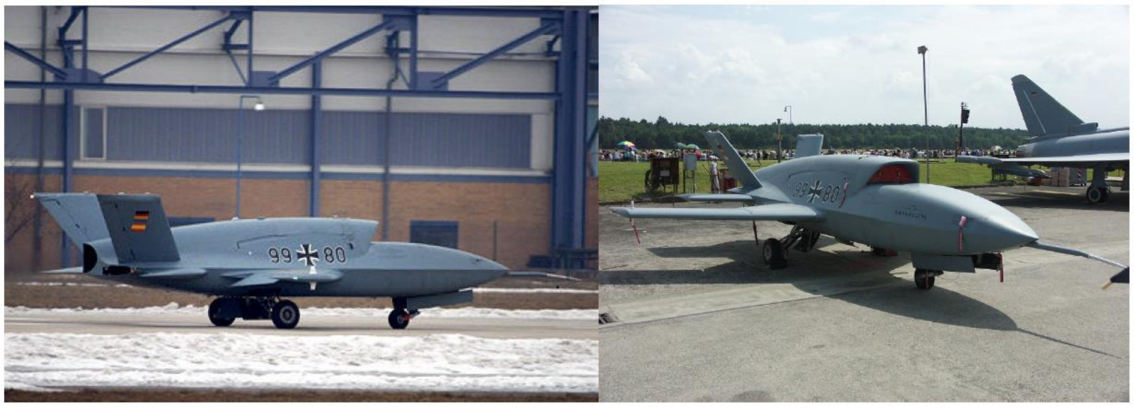


Figure 1. 1 Barrucada UAV.[1]

Basically the meaning of these UAV is Unmanned Aerial Vehicle that can be driven by both an operator or simply autonomously. For instance, a military aeroplane such as the Barrucada or the combat plane Boeing is also considered an UAV.

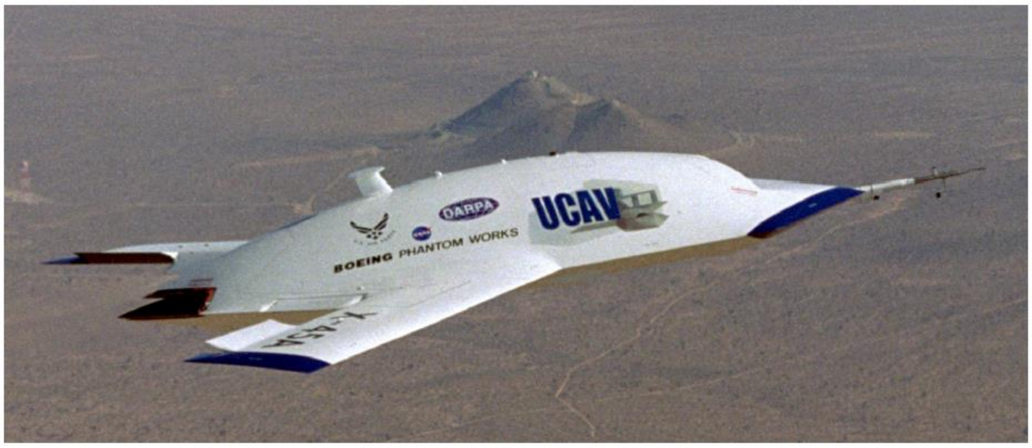


Figure 1. 2. Military UAV. Boeing.[2]

In the last years, given that the technology is moving forward surprisingly fast, there is a particular kind of UAV that's being commercializing, such UAV is a multirotor vehicle in which its rotors are contained in the same plane. Several models can be found in the market such as the AR Drone [3] or the Mikrocopter [4] available in different versions, with four, six and eight coplanar motors.

The four motor models are the most used among researchers all over the world, it can be seen in a lot of publications made both for visual controllers or proving diverse non-linear controllers.

Figure 1. 3. AR Drone. Four motor model.[3]

1.2 Use of cameras

The computer vision field has evolved in last years; nowadays a camera is used not only to take pictures or videos within a robotic platform, but to be used as another sensor.

Within this area several techniques have been found, for instance there have been different research projects based on the use of one camera on board [5], two cameras on board [6] or even the use of cameras out of the platform [7], to localize the robot. The use of a camera as a sensor is mostly employed in indoors projects [1] [8] [9].

Currently, researchers all around the world are focusing their efforts trying to develop autonomous vehicles, so they could fly without any human help, being completely precise and accurate. The first researches were made outdoors [10] given the design and features of a flying vehicle. Thus the localization of the quadcopter could be known at any time.

The research path has changed with the study of the indoors localization. Now, we are heading to a different line of research, the GPS signal cannot be used in these types of projects, so other sensors are needed to localize the robot. Due to the lack of GPS signals it is highly needed a sensor fusion such as scan, camera, odometer, etc.

So we can find lots of research projects trying to control the robot [11], aiming to build a map and localize the robotic platform at a time [12]. Most of these projects use either the camera or the laser or even both of them.

That’s why last trends are based on the idea that using a camera on board, and having other sensors included within the platform the problem of localizing and mapping the environment in which the robot is moving can be solved at a time.

These techniques are called SLAM (simultaneous localization and mapping). They came up to solve the problem that appears if the map in which the drone is going to involve is known. Not always the map can be known beforehand. So using such techniques, a robot can be moving in an unknown environment, but it can get enough information to localize itself using SLAM algorithms. The more information it gets from the sensors the more precise and accurate will be the mapping and the localization.

1.3 The raise of the SLAM techniques

In 1987 at the IEEE International Conference in Robotics and Automation, Randall Smith, Matthew Self and Peter Cheeseman presented what is now considered the first project that describes the representation for spatial information, called the stochastic map [13]. This map contained the estimates of relationships among objects in the map, and their uncertainties, given all the available information. The procedures provided a general solution to the problem of estimating uncertain relative spatial relationships.

Other pioneering work in this field was conducted by the research group of Hugh F. Durrant-Whyte in the early 2000s, which showed that solutions to SLAM exist in the infinite data limit [14]. This finding motivated the search for algorithms which are computationally tractable and approximate the solution.

Figure 1. 4. STABLEY and JUNIOR car [15].

One of the projects that brought SLAM to the worldwide attention was the self-driving STANLEY and JUNIOR car, which won the DARPA challenge in the 2000s [15]. This project was led by Sebastian Thrun at the Stanford University.

After this first trial, a lot of researchers all over the world focused their efforts in the construction and implementation of SLAM techniques within robotics platforms. So in this section, we will talk about some related projects that use these SLAM techniques for indoor navigation using MAV's.

- **Autonomous Multi-Floor Indoor Navigation with a computationally constrained MAV**
  In 2011 at the IEEE International Conference on Robotics and Automation, Shaojie Shen, Nathan Michael and Vijay Kumar, presented this project called "Autonomous Multi-floor Indoor Navigation with computationally constrained MAV" [8] based on a navigation system for indoor environments.

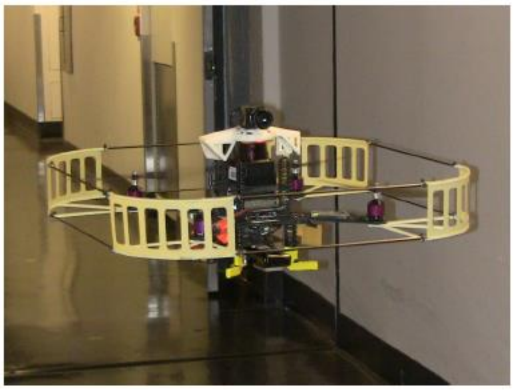


Figure 1. 5. Experimental Platform with on board computation [8].

Its objective was to obtain a system capable of perform an autonomous navigation in indoor environments, especially in buildings with multiple floors. The system consists on a MAV equipped with a scan Hokuyo UTM-30LX, and a camera UI-1220SE and an IMU.

On the other hand, the software consists on a SLAM localization module and a planification module, which is in charge of the navigation of the MAV.

The localization module based on SLAM, employs a 2.5 D environment model, which assumes that the environment is just based on horizontal and vertical planes. It employs an ICP (Iterative Closest Point) algorithm to estimate its position, using the data from the scan, and fusing them whit the data from the IMU.

The navigation module employs a RRT (Rapidly-exploring Random Tree) algorithm to generate trajectories allowing the system to achieve the objectives, avoiding the presented obstacles. Such trajectories are executed using a position control loop, taking the robot as a punctual system that counts with an orientation in the plane.

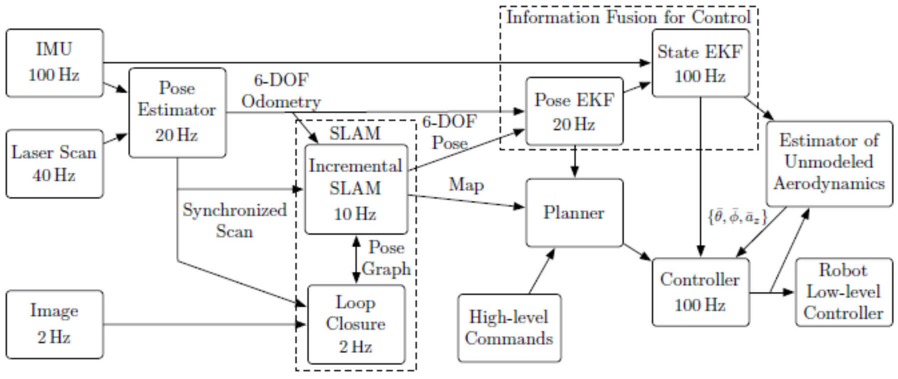


Figure 1. 6. Architecture diagram showing the software modules [8].

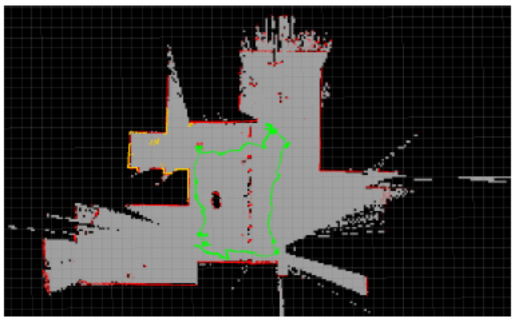


Figure 1. 7. Map generated by flying [8].

- **State Estimation in GPS-Denied Environments Using On board Sensing**
  This project was developed by Adam Bry, Abraham Bachrach and Nicholas Roy at the Massachusetts Institute of Technology.

Figure 1. 8. Fixed wing experimental platform flying indoors [16].

It was developed in 2012 under the name “State Estimation for Aggressive Flight in GPS-Denied Environments Using On board Sensing” [16]. The objective of such project was to develop a state estimation method based on an IMU and a planar laser range finder, suitable for use in a MAV.
The system us capable of accurately estimate the state of a MAV in a 3D unstructured environment without using an external position system.

The localization algorithm is based on an extension of the Gaussian Particle Filter. It also employs an EKF to estimate the state of the MAV. So all in all, this project employs two different filters to get the state of the MAV. First of all, it employs an EKF for the IMU process model, and the GPF for the laser measurement update. Particle filters are efficient enough for effective use in localizing a 2D mobile robot; they require too many particles to be used for the estimation of a 3D MAV. Fortunately, the best aspects of both algorithms can be obtained, and a significant speedup can be realized by employing a hybrid filter that uses an IMU-driven EKF process model with pseudo-measurements computed from Gaussian Particle Filter (GPF) laser measurement updates.

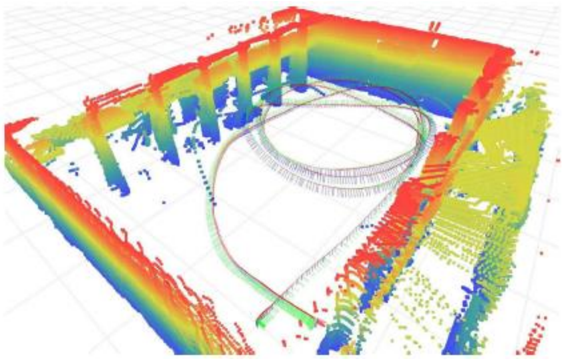
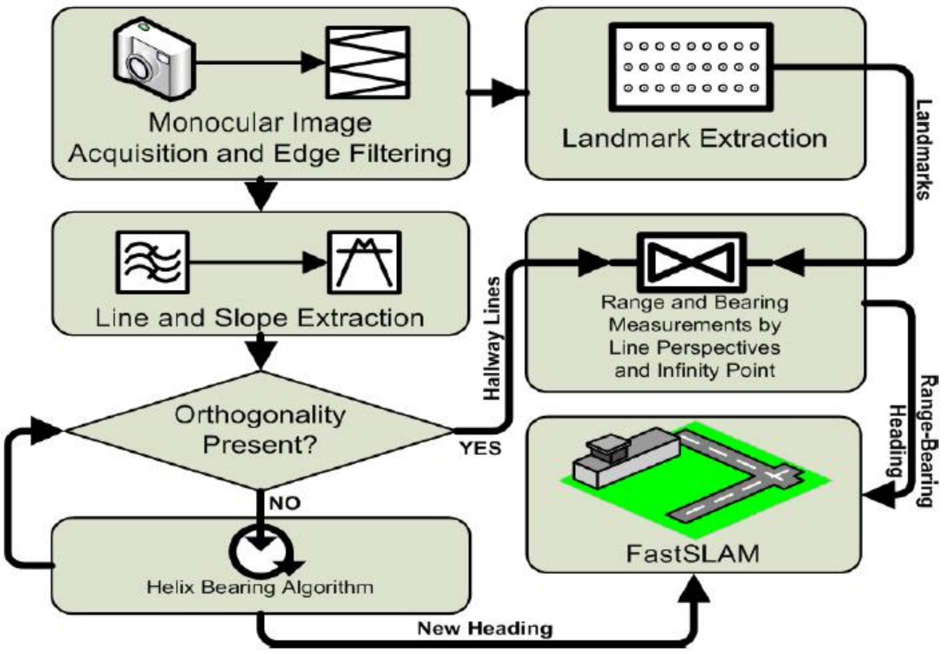


Figure 1. 9. The trajectory flown by the vehicle is shown by the red, green, and blue axes [16].

- **Monocular Vision SLAM for Indoor Aerial Vehicles**
  Developed by Koray Celik, Soon-Jo Chung, Matthew Clausman and Arun K.Somani, this project named "Monocular Vision SLAM for Indoor Aerial Vehicles" [9], presents a novel indoor navigation and ranging strategy by using monocular camera.
  The project addresses to get the localization of a MAV, and the mapping of the environment by using a monocular camera of 1 2 inches in size and less than 2 ounces in mass. The process flow of the proposed method is shown in figure 1.10.



Monocular Image Acquisition and Edge Filtering
Landmark Extraction
Landmarks
Line and Slope Extraction
Hallway Lines
Range and Bearing Measurements by Line Perspectives and Infinity Point
Orthogonality Present?
YES
NO
Range-Bearing Heading
Helix Bearing Algorithm
FastSLAM
New Heading

Figure 1. 10. Block diagram illustrating the operational steps of the monocular vision system [9].

This monocular vision SLAM correctly locates and associates landmarks.

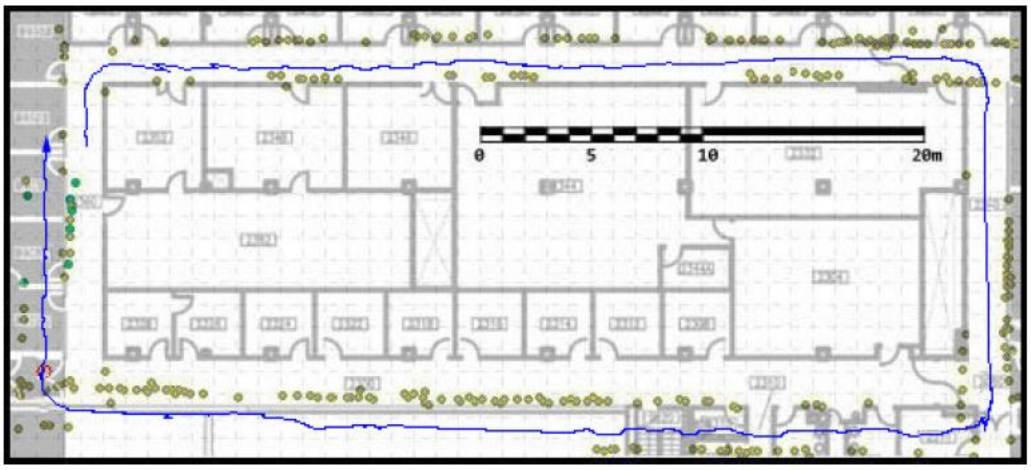


Figure 1. 11. Experimental results of the proposed ranging and SLAM algorithm [9].

A 3D map is also built by the addition of time-varying altitude and wall-positions, as shown in Fig 1.12.

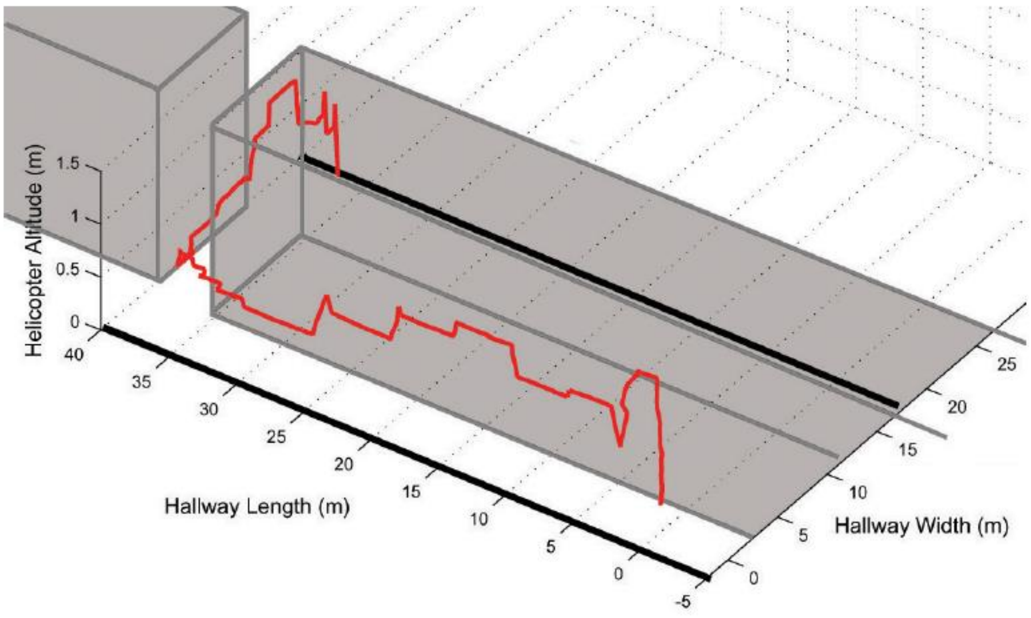


Figure 1. 12. Cartesian (x; y; z) position of the MAV in a hallway [9].

The MAV assumes that it is positioned at (0; 0; 0) Cartesian coordinates at the start of a mission, with the camera pointed at the positive x axis, therefore, the width of the corridor is represented by the y axis.

To get the project done, they used the Saint Vertigo helicopter, one of the smallest and fully self-contained autonomous helicopters in the world capable of both indoor and outdoor operation. This unit performs all image processing and SLAM computations on-board via a 1GHz x86 architecture CPU with SIMD instructions, 1GB DDR2 533MHz RAM, 4GB solid-state mass storage, managed by a performance tuned Linux kernel.

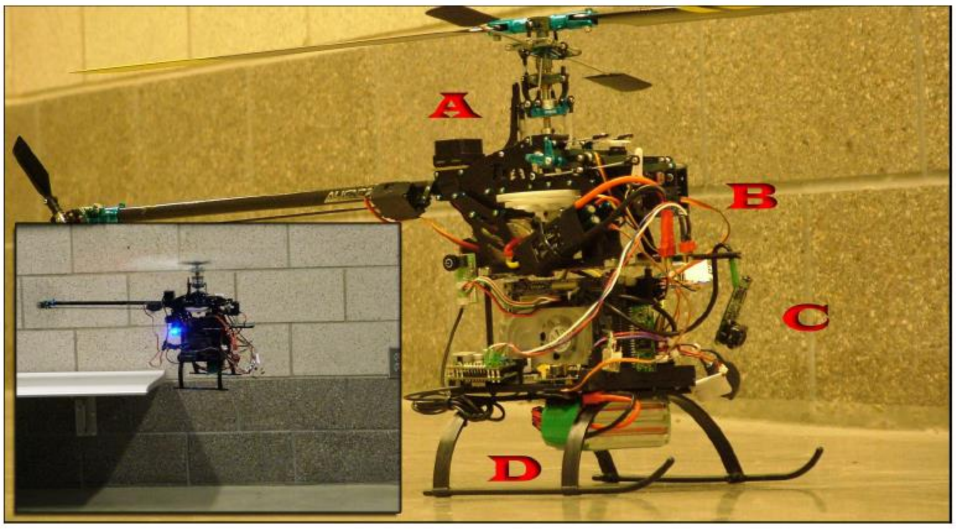


Figure 1. 13. Saint Vertigo, the autonomous MAV helicopter [9].

In essence, the MAV features two independent computers. The flight computer is responsible for flight stabilization, flight automation, and sensory management, including but not limited to tracking the time-varying altitude via an ultrasonic altimeter. The navigation computer is responsible for higher consciousness tasks such as image processing, range measurement, SLAM computations, networking, mass-storage, and possibly, path planning.

1.4 Objectives

Using the Erlecopter drone, the implementation of VSLAM techniques in the brain of this robot will be carried out. Different state-of-art algorithms will be computed; these algorithms are PTAM, LSD-SLAM and ORB-SLAM.

As a software platform, ROS (Robotic Operational System) and GAZEBO will be used, to control and insert all the algorithms that this project will implement. All these platforms and algorithms will be far explained within the next chapters.

Both simulation and the real drone trials will be studied using all the algorithms, locating and mapping the environment at a time. The idea is to use this VSLAM algorithms indoors having the lack of GPS, fusing the information with other sensors like the IMU and the ultrasonic, besides given this information, we will try to make the drone work autonomously. As Sergio García Gonzalo [5] did in his project, a PID and an EKF filter will be implemented. The main difference with this project is the computational system, in [5] the algorithm run in a not embedded CPU or brain, while in this project the algorithm will be running in the drone itself, this means that the ErleCopter has a Raspberry Pi embedded in the robot. So there is no need to have a wireless connection between the robot and the CPU.

So now the project is defined: Implementation of VSLAM algorithms in the own CPU of the ErleCopter, also designing a PID and inserting an EKF, using GAZEBO for the simulation and ROS for both the simulation and the real system.

Summarizing the key objectives of this project:

- Study three different vSLAM techniques: PTAM, ORB-LAM and LSD-SLAM.
- Implement such techniques within the quadcopeter ErleCoper both in the simulation and the real drone.
- Develop an EKF (Extended Kalman Filter) fusing the data from the vSLAM techniques and the rest of the sensors.
- Develop a PID controller for the drone.
- Test the EKF and the vSLAM algorithms for both the simulation and the real quadcopter.

1.5 Work Structure

At this point, it is time to take a look at the structure of this work, so the reader can localize itself, making easier the understanding of current document.

It is based on 9 different chapters, divide as follows:

- **Chapter 1. Introduction:** This is the chapter in which we currently are. The objectives of this project are explained here, and other subjects like the related works, the raise of the slam techniques, or the use of the cameras in robotics systems are also explained.
- **Chapter 2. Tools:** The reader will deal with the tools employed for this project, both hardware and software.
- **Chapter 3. Monocular Visual SLAM:** Explanation of the work flow of each of the three main algorithms, studying their performance and comparing the results.
- **Chapter 4. Extended Kalman Filter:** This is the most technical part of the project. In this chapter the data from the sensors and the algorithms will be fused to develop an EKF to predict the localization of the robot accurately.
- **Chapter 5. PID Controller:** Development of a PID to get a better performance for the robotic system.

- **Chapter 6. Conclusions and future work:** It deals with the extracted conclusions after the ending of the project, the problems that were faced, and the future work given the results obtained.
- **Chapter 7. User's Manual:** Instructions and needed applications to comprehend the project and its work flow.
- **Chapter 8. Specifications**: It contains the specifications of the employed tools.
- **Chapter 9. Budget:** Budget of the project.
- **Chapter 10. Bibliography:** Documentation consulted during the project.

CHAPTER 2

TOOLS

In this chapter the software and hardware tools used to develop the project will be defined and explained.

2.1. Robot Operating System (ROS)

This platform, widely known just as ROS, provides the necessary tools to help us developing and creating any robot application. It provides hardware abstraction, device drivers, libraries, visualizers, message passing, package management and more. ROS is licensed under an open source, BSD license.

Now, it is very important to describe why ROS is so useful for robotic developers. Here we will define a few specific issues in the development of software for robots that ROS can help to resolve:

- Distributed computation. Many modern robot systems rely on software that spans many different processes and runs across several different computers.
- Software reuse. The rapid progress of robotics research has resulted in a growing collection of good algorithms for common tasks such as navigation, motion planning, mapping, and many others. Of course, the existence of these algorithms is only truly useful if there is a way to apply them in new contexts, without the need to reimplement each algorithm for each new system. ROS can help to prevent this kind of pain in different ways.
- Rapid testing. One of the reasons that software development for robots is often more challenging than other kinds of development is that testing can be time consuming and error-prone. Physical robots may not always be available to work with, and when they are, the process is sometimes slow and finicky. Working with ROS provides two effective workarounds to this problem.

All of these issues exist in the project that we are describing. So, all in all, we can now understand its importance.

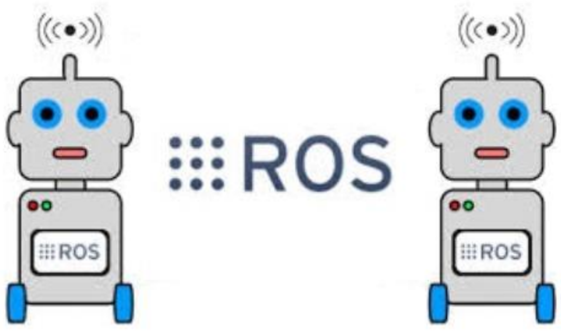


Figure 2. 1. ROS Icon

For a better understanding it is highly recommended the study of the ROS file system level, and the ROS computation graph level:

ROS file System Level

Similar to an Operating System, ROS files are organized in a particular way within the hard disk. Figure 2.2 shows how the ROS files and folder are organized on the disk:

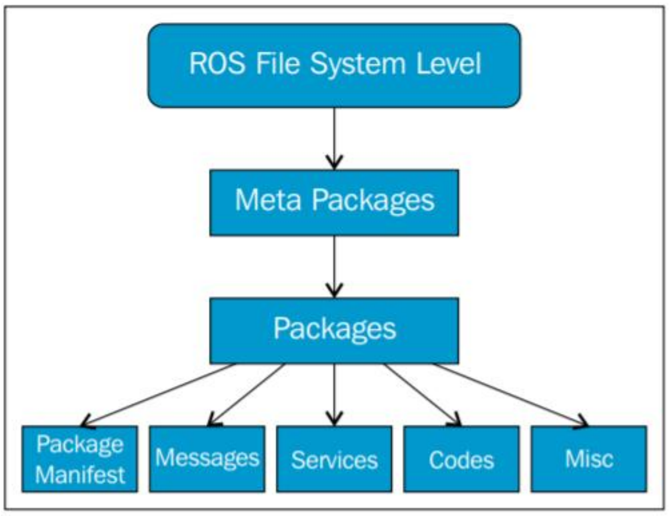


Figure 2. 2. ROS file System Level.

Here is a brief explanation of each component belonging to the file system:

• **Packages:** Are the most basic unit of the ROS software. Packages are the atomic build item and release item in the ROS software.

• **Package manifest:** The package manifest file is inside a package that contains information about the package such as author, license, dependencies, compilation flags, and so on. The package.xml file inside the ROS package is the manifest file of that package.

• **Meta packages:** The term meta package is used for a group of packages for a special purpose

• **Meta packages manifest:** Similar to the package manifest, the main differences are that it might include packages inside it as runtime dependencies and declare an export tag.

• **Messages (.msg):** ROS messages are a type of information that is sent from one ROS process to the other. The extension of the message file is .msg.

• **Services (.srv):** The ROS service is a kind of request/reply interaction between processes. The reply and request data types can be defined inside the srv folder inside the package (my_package/srv/MyServiceType.srv).

• **Repositories:** Most of the ROS packages are maintained using a Version Control System (VCS) such as Git. The collection of packages that share a common VCS can be called repositories.

ROS computation graph level

The computation in ROS is done using a network of processes called ROS nodes. This computation network can be called the computation graph. Its structure is shown in figure 2.3.

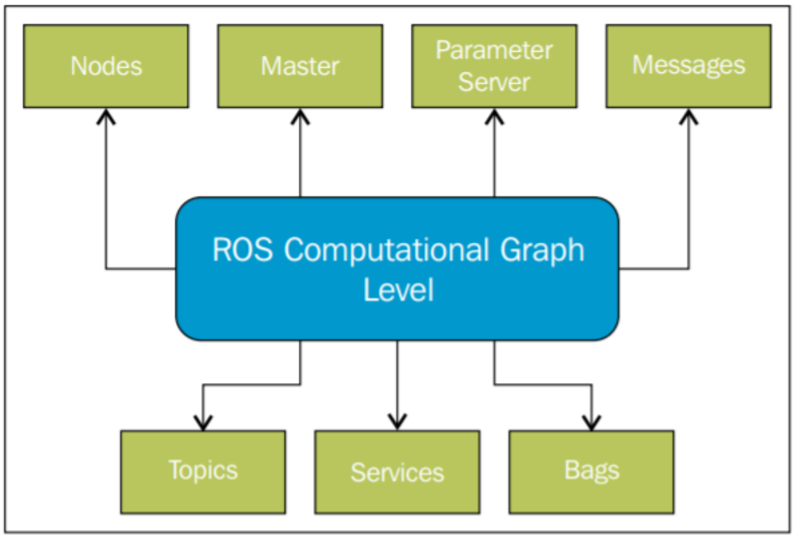


Figure 2. 3. ROS computation graph Level

Let´s briefly define each concept of the graph:

• **Nodes:** Nodes are the processes that perform computation. In a robot, there will be many nodes to perform different kinds of tasks. Using the ROS communication methods, it can communicate with each other and exchange data. One of the aims of ROS nodes is to build simple processes rather than a large process with all functionality.

• **Master:** The ROS Master provides name registration and lookup to the rest of the nodes. Nodes will not be able to find each other, exchange messages, or invoke services without a ROS Master. In a distributed system, we should run the master on one computer, and other remote nodes can find each other by communicating with this master.

• **Parameter Server:** The parameter server allows you to keep the data to be stored in a central location. All nodes can access and modify these values. Parameter server is a part of ROS Master

• **Messages:** Nodes communicate with each other using messages. Messages are simply a data structure containing the typed field, which can hold a set of data and that can be sent to another node.

• **Topics:** Each message in ROS is transported using named buses called topics. When a node sends a message through a topic, then we can say the node is publishing a topic. When a node receives a message through a topic, then we can say that the node is subscribing to a topic. Each topic has a unique name, and any node can access this topic and send data through it as long as they have the right message type.

• **Services:** In some robot applications, a publish/subscribe model will not be enough if it needs a request/response interaction. The publish/subscribe model is a kind of one-way transport system and when we work with a distributed system, we might need a request/response kind of interaction. ROS Services are used in this case. We can define a service definition that contains two parts; one is for requests and the other is for responses. Using ROS Services, we can write a server node and client node. The server node provides the service under a name, and when the client node sends a request message to this server, it will respond and send the result to the client. The client might need to wait until the server responds. The ROS service interaction is like a remote procedure call.

• **Bags:** Bags are a format for saving and playing back ROS message data. Bags are an important mechanism for storing data, such as sensor data, which can be difficult to collect but is necessary for developing and testing robot algorithms. Bags are very useful features when we work with complex robot mechanisms.

Besides, this project will be developed using as a robot the quadcopter ErleCopter, which will be described within this chapter, in the following points. This quadcopeter has a raspberry pi, as a brain, with ROS pre-installed in it, so it is far sensible using ROS to develop a robot application for it.

2.2. Gazebo

Gazebo is a simulator system for 3D environments that makes possible the evaluation of the behaviour of a robot in a virtual world. It allows, among different options, personalize the design of a robot, and create virtual worlds using simply tools like CAD or just importing created models.

Besides, its importance relies on the fact that it’s possible to synchronize this simulator with ROS, so the emulated robots can publish information from its sensors in the nodes, and also send commands and orders to the robot.

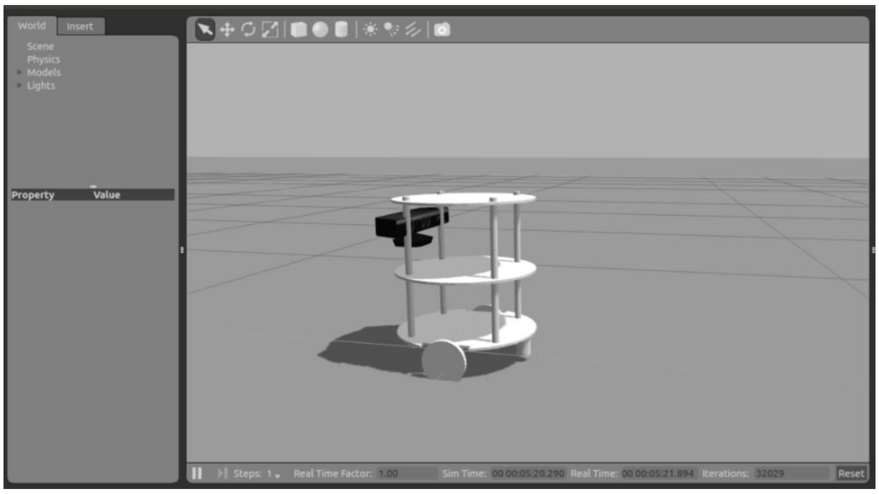


Figure 2. 4. Gazebo software platform with a Turtlebot.

Therefore, a simulation platform is an essential tool in every robotics toolbox. A well-designed simulator makes it possible to rapidly test algorithms, design robots etc. Gazebo offers the ability to accurately and efficiently simulate populations of robots in complex indoor and outdoor environments. It is also a very robust physics engine with high-quality graphics, and convenient programmatic and graphical interfaces.

The ErleCopter is currently implemented in Gazebo, so it will make our project simpler, given the fact that it is very easy to import the model from the ErleCopter webpage. First of all it is needed to configure our environment in our Ubuntu machine, such as installing ROS, APM/Ardupilot, creating a workspace… To do so just follow the steps uploaded at the Erlerobotics official webpage, which you can access here: http://docs.erlerobotics.com/simulation/configuring_your_environment

Anyway, we will explain all the steps needed to install gazebo and how import the model of the Erlecopter to our environment:

Option 1: Install Gazebo using Ubuntu packages

Setup your computer to accept software from packages.osrfoundation.org

```
sudo sh -c 'echo "deb http://packages.osrfoundation.org/gazebo/ubuntu-stable
`lsb_release -cs` main" > /etc/apt/sources.list.d/gazebo-stable.list'
```

Setup keys

```
wget http://packages.osrfoundation.org/gazebo.key -O - | sudo apt-key add -
```

Install gazebo7

```
sudo apt-get update
sudo apt-get remove .*gazebo.* '.*sdformat.*' '.*ignition-math.*' && sudo apt-get update
&& sudo apt-get install gazebo7 libgazebo7-dev drcsim7 -y
```

Option 2: Install Gazebo from source

Compile the workspace

Then compile everything together:

```
cd ~/simulation/ros_catkin_ws
catkin_make --pkg mav_msgs mavros_msgs gazebo_msgs
source devel/setup.bash
catkin_make -j 4
```

Download Gazebo models

```
mkdir -p ~/.gazebo/models
git clone https://github.com/erlerobot/erle_gazebo_models
mv erle_gazebo_models/* ~/.gazebo/models
```

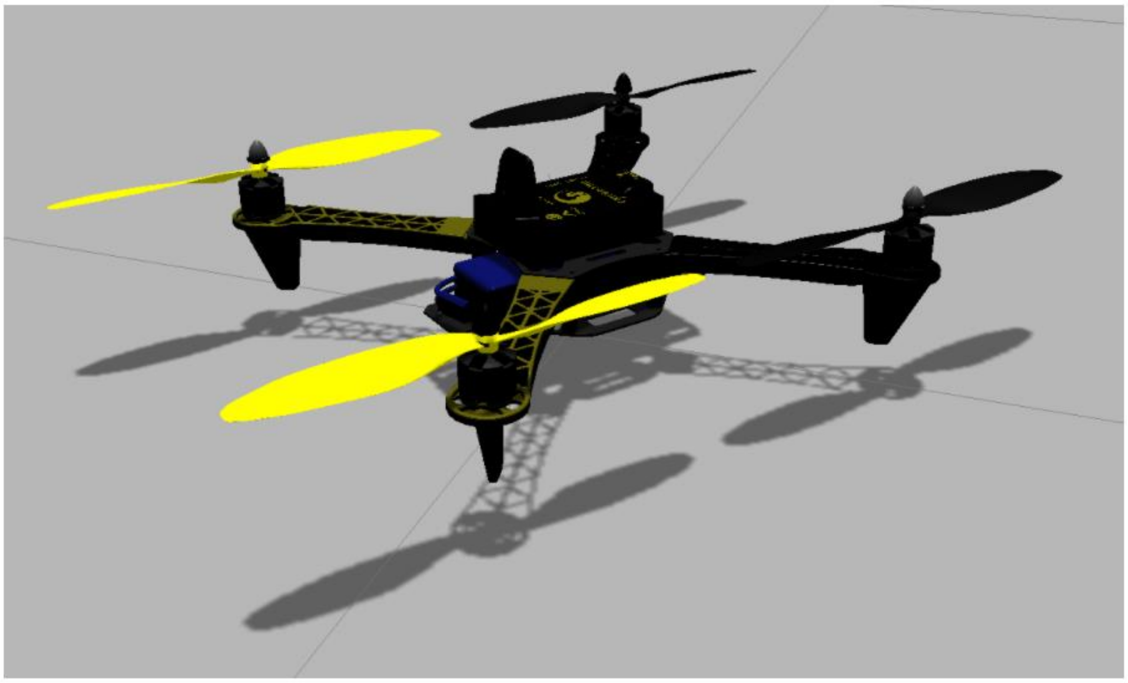


Figure 2. 5. ErleCopter model in Gazebo downloaded in ErleRobotics.

2.3 ErleCopter

TheErleCopter is the first Linux-Based smart drone that uses robotic frameworks such as the described software ROS, and the award winning APM software autopilot to achieve different flightmodes.

Although this quadcopter is ideal for outdoor operations, it is very useful for our research. As we said at the beginning of this project, the aim is to implement a VSLAM system with an EKF to localize the robot and build a map of the environment. At this point, it is important to highlight that this is just half of the work of a bigger project. Nicolás Blanco Fernández [17] is developing a similar project, but using a laser instead of a camera so the combined job between both projects is the fusion of the information obtained from the laser and the camera to minimize the error of the location and the accuracy of the built map. Obviously to carry these sensors, mainly the laser, we need to handle a weight of, approximately, 2 kilograms. Not all the quadcopters that have been already launched at the market fulfil this requirement. That’s why the ErleCopter has been chosen, it was designed with a take-off weight of up to 2 kilograms.

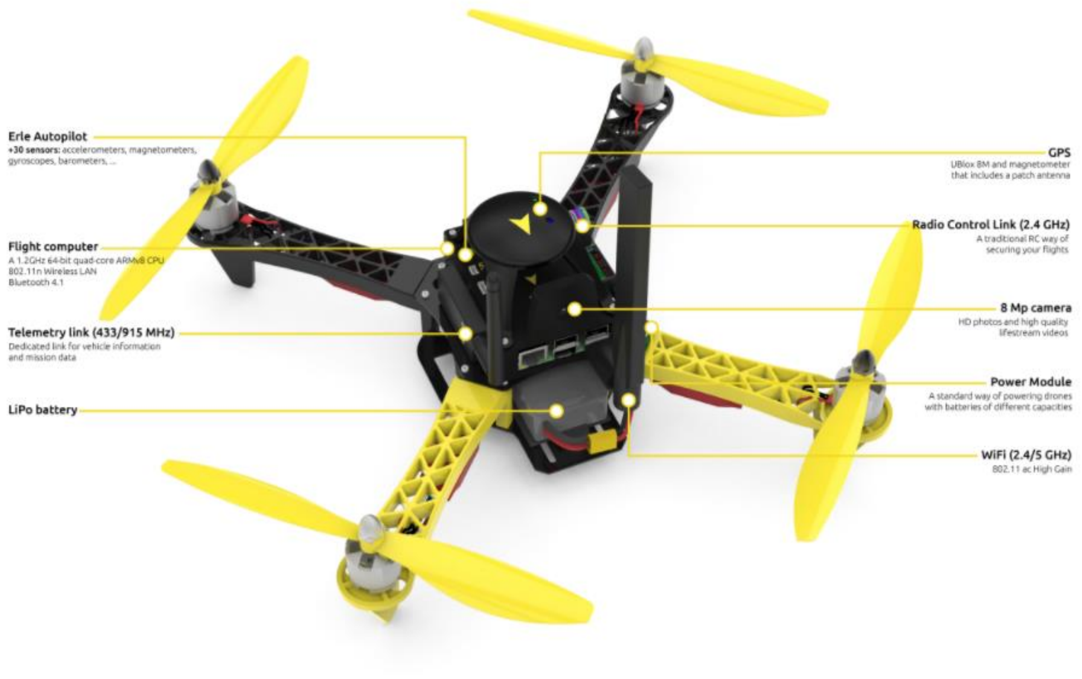
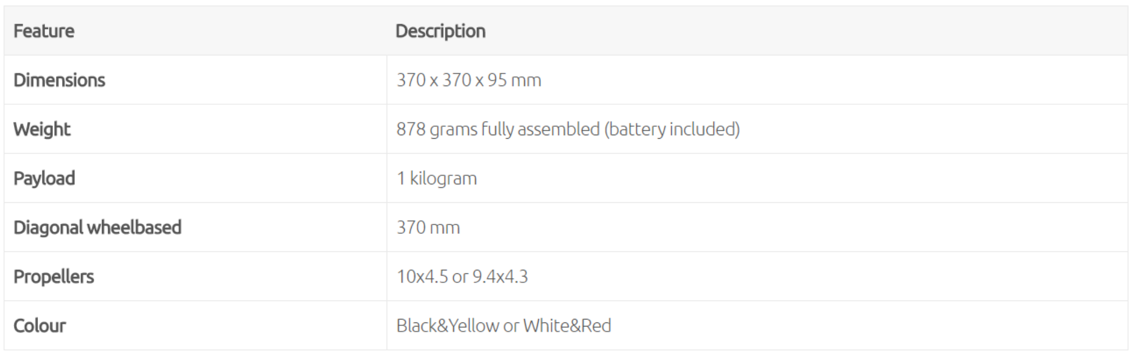

Erle Autopilot
+30 sensors: accelerometers, magnetometers, gyroscopes, barometers, ...
Flight computer
A 1.2GHz 64-bit quad-core ARMv8 CPU
802.11n Wireless LAN
Bluetooth 4.1
Telemetry link (433/915 MHz)
Dedicated link for vehicle information and mission data
LiPo battery
GPS
UBlox 8M and magnetometer that includes a patch antenna
Radio Control Link (2.4 GHz)
A traditional RC way of securing your flights
8 Mp camera
HD photos and high quality lifestream videos
Power Module
A standard way of powering drones with batteries of different capacities
WiFi (2.4/5 GHz)
802.11 ac High Gain

Figure 2. 6. Hardware parts of the ErleCopter.



Feature	Description
Dimensions	370 x 370 x 95 mm
Weight	878 grams fully assembled (battery included)
Payload	1 kilogram
Diagonal wheelbased	370 mm
Propellers	10x4.5 or 9.4x4.3
Colour	Black&Yellow or White&Red

Table 2. 1. Features of the ErleCopter

2.3.1 Flight modes

It is important to know the different flight modes for the ErleCopter. There are several ways to flight.

The ones that do not require GPS lock are:

- **Stabilize:** this mode allows flying the copter manually, but self-levels the roll and pitch axis.
- **Alt Hold:** the Erle-Copter will maintain a consistent altitude, allowing roll, pitch, and yaw to be controlled normally.
- **Acro:** this mode uses the RC sticks to control the angular velocity of the copter. It is useful for aerobatics such as flips or rolls.
- **Land:** Land mode attempts to bring the copter straight down.

The ones that require GPS lock prior to takeoff are:

- **Loiter:** Loiter mode attempts to maintain the current location, heading and altitude.
- **RTL (Return-to-Launch):** the copter navigates from its current position to hover above the home position.
- **Auto:** Erle-Copter will follow a pre-programmed mission script stored in the autopilot which is made up of navigation commands (i.e. waypoints) and "do" commands (i.e. commands that do not affect the location of the copter including triggering a camera shutter).
- **Guided:** allows the Copter to be dynamically guided to a target location wirelessly using a telemetry radio module and ground station application.

- **Drift:** allows the user to fly a multi-copter as if it were a plane with built in automatic coordinated turns.
- **PosHold:** it is similar to Loiter in that the vehicle maintains a constant location, heading, and altitude the difference is that the pilot stick inputs directly control the vehicle's lean angle.
- **Follow Me:** the Copter will follow the pilot while moving, using a telemetry radio and a ground station.
- **Circle:** the vehicle will orbit a point of interest with the nose of the vehicle pointed towards the center.

2.3.1.1 Hardware

This section aims the understanding of how the quadcopter is made; figure 2.7 gathers all the components that model the quadcopter.

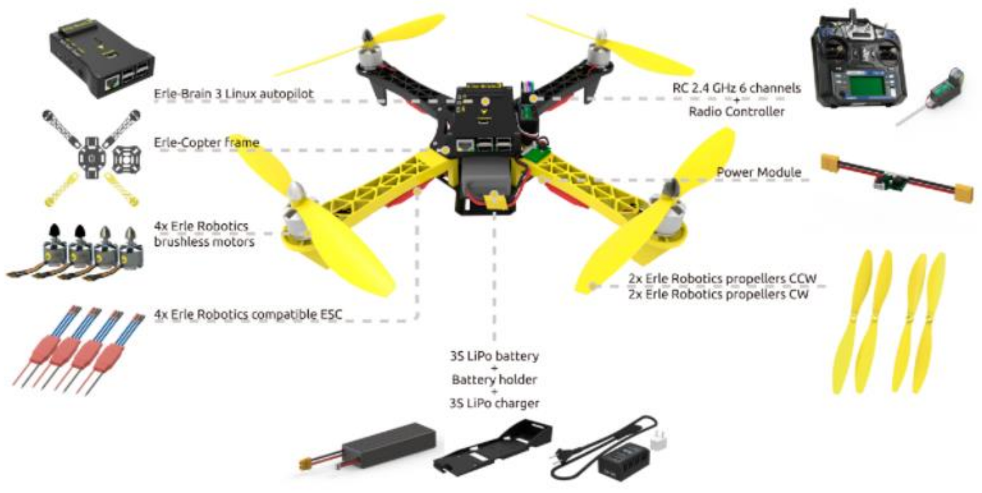
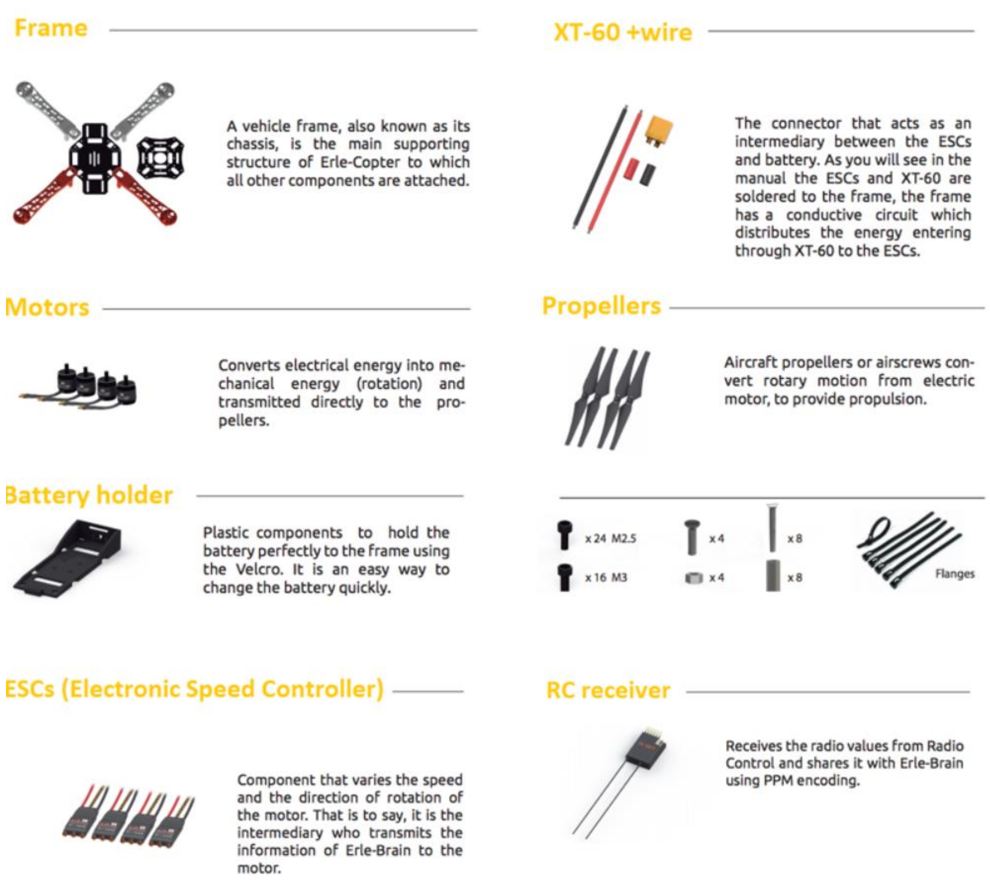

Erle-Brain 3 Linux autopilot
Erle-Copter Frame
4x Erle Robotics brushless motors
4x Erle Robotics compatible ESC
RC 2.4 GHz 6 channels + Radio Controller
Power Module
2x Erle Robotics propellers CCW
2x Erle Robotics propellers CW
3S LiPo battery + Battery holder + 3S LiPo charger

Figure 2. 7. Components that model the ErleCopter.

In figure 2.8, each of these components is briefly defined.


Frame

A vehicle frame, also known as its chassis, is the main supporting structure of Erle-Copter to which all other components are attached.

XT-60 +wire

The connector that acts as an intermediary between the ESCs and battery. As you will see in the manual the ESCs and XT-60 are soldered to the frame, the frame has a conductive circuit which distributes the energy entering through XT-60 to the ESCs.

Motors

Converts electrical energy into mechanical energy (rotation) and transmitted directly to the propellers.

Propellers

Aircraft propellers or airscrews convert rotary motion from electric motor, to provide propulsion.

Battery holder

Plastic components to hold the battery perfectly to the frame using the Velcro. It is an easy way to change the battery quickly.

ESCs (Electronic Speed Controller)

Component that varies the speed and the direction of rotation of the motor. That is to say, it is the intermediary who transmits the information of Erle-Brain to the motor.

RC receiver

Receives the radio values from Radio Control and shares it with Erle-Brain using PPM encoding.

Figure 2. 8. Defined components of the ErleCopter.

2.3.2 Erle-Brain

Drones deployed in real applications have several computational units. Among them the Flight Control Unit (FCU) (a computer that provides basic flight controls, and companion computer) a computational device in charge of higher level behaviors such as image processing or image broadcasting.

Erle-Brain is an all in one Linux brain for drones that provides FCU capabilities and companion computer. Everything in a simple package.

The Erle- Brain is the artificial brain with which the ErleCopter is made. It includes gravity sensors, gyroscopes, and a digital compass.

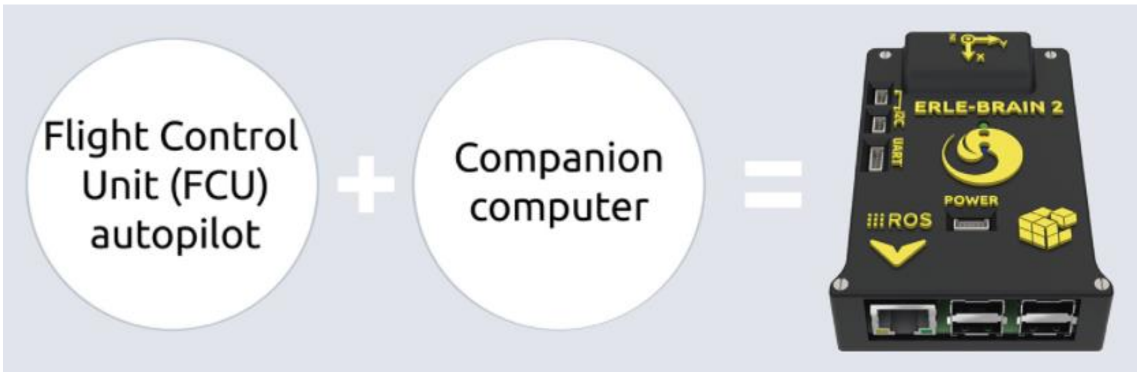


Figure 2. 9. Erlebrain 2 Units.

Obviously, the communications of the brain are a key part within the whole on board unit. These communications are the ones needed to implement all the sensors and the codes to develop the project that is being carried out.

- WiFi: It is highly recommended to use an external Wi-Fi dongle in order to use a 5GHz bandwidth.
- Dongle: Using a dongle Wi-Fi we can create a hotspot using the brain. This way, there is no need to use wires to connect the device. The Wi-Fi can be used for communicate with the ardupilot, transfer files such as logs, plan a mission using a GCS (Ground Control Station), enable video streaming, even control the drone.
- Ethernet: An Ethernet wire can be plugged to the Erle-brain to have internet access, needed to install new software or access the brain from the local network. This allows the communication with other devices.
- USB: Used to attach the dongle, or to include an external storage.
- I2C: The ErleBrain contains two I2C bus connectors, which gives access to the I2Cbus. In this bus a bunch of different sensors and devices can be connected. In this particular an ultrasonic sensor will be plugged to this connector.
- PWM: It has 12 channels of PWM. Each channel has a 25 mA current sink capability a 5V. In these channels, the most typical devices that can be connected are: ESCs, servos, gimbal servos… The PWM can be used for powering the system.
- RC Input: This is the radio controller that must be connected to the channel 14.
- UART: A computer hardware device for asynchronous serial communication in which the data format and transmission speeds are configurable.
- SD Card: The ErleBrain contains one SD Card slot.

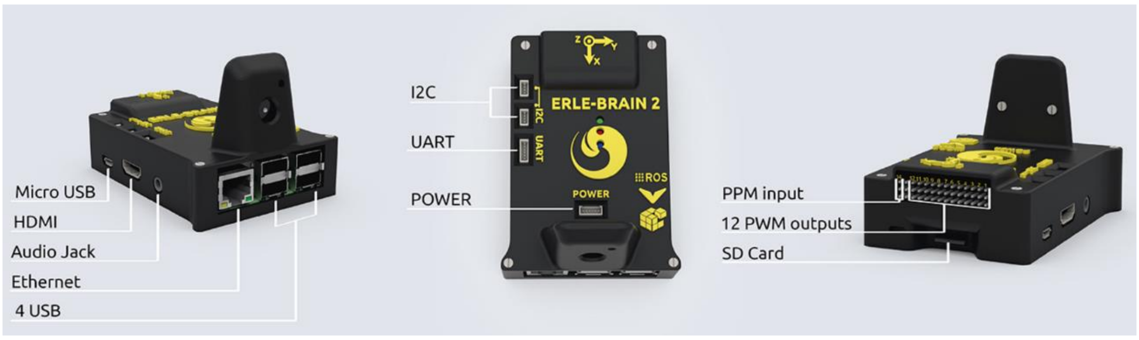


Figure 2. 10. Communication Units of the ErleBrain 2.

The points defined above, talk about the basic system. However, the Erlebrain that will be used for this project also has a Camera integrated in it. It is an 8MP camera with fixed focus lens, 2592 x1944 pixel static images, supports 1080p30, 720p60 and 640x480p60/90 video record. It will be used to work with the VSLAM algorithms that will be described in the following section.

CHAPTER 3

MONOCULAR VISUAL SLAM

3.1. Introduction

Aiming to perform our particular objectives, vSLAM different techniques must be used. The importance of these algorithms stands on the fact that, besides the simple technical features, it is able to run in real-time.

In general, the technical difficulty of vSLAM is higher than that of other sensor-based SLAMs because cameras can acquire less visual input from a limited field of views compared to 360° laser sensing which is typically used in robotics. From such input, camera poses need to be continuously estimated and the 3D structure of an unknown environment is simultaneously reconstructed.

Generally, the framework works mainly with three modules as follows:

1. Initialization
2. Tracking
3. Mapping

To start vSLAM, it is necessary to define a certain coordinate system for camera pose estimation and 3D reconstruction in an unknown environment. Therefore, in the initialization, the global coordinate system should first be defined, and a part of the environment is reconstructed as an initial map in the global coordinate system. After the initialization, tracking and mapping are performed to continuously estimate camera poses. In the tracking, the reconstructed map is tracked in the image to estimate the camera pose of the image with respect to the map. In order to do this, 2D–3D correspondences between the image and the map are first obtained from feature matching or feature tracking in the image. Then, the camera pose is computed from the correspondences by solving the perspective problem. It should be noted that most of vSLAM algorithms assume that intrinsic camera parameters are calibrated beforehand so that they are known. Therefore, a camera pose is normally equivalent to extrinsic camera parameters with translation and rotation of the camera in the global coordinate system. In the mapping, the map is expanded by computing the 3D structure of an environment when the camera observes unknown regions where the mapping is not performed before.

It is also very important to understand that the vSLAM algorithms have two additional modules according to the purposes of applications.

- Relocalization
- Global map optimization

The relocalization is required when the tracking fails due to fast camera motion or some disturbances. In this case, it is necessary to compute the camera pose with respect to the map again. Therefore, this process is called “relocalization.” If the relocalization is not incorporated into vSLAM systems, the systems do not work anymore after the tracking is lost and such systems are not practically useful, that`s why the algorithms used whitin this project will always work with this module.

The other module is global map optimization. The map generally includes accumulative estimation error according to the distance of camera movement. In order to suppress the error, the global map optimization is normally performed. Generally, in this process, the map is refined by considering the consistency of whole map information. When a map is revisited such that a starting region is captured again after some camera movement, reference information that represents the accumulative error from the beginning to the present can be computed. Then, a loop constraint from the reference information is used as a constraint to suppress the error in the global optimization.

There is another technique called loop closing. It is a technique to acquire the reference information. In the loop closing, a closed loop is first searched by matching a current image with previously acquired images. If the loop is detected, it means that the camera captures one of previously observed views. In this case, the accumulative error occurred during camera movement can be estimated.

So, to summarize, the framework of vSLAM algorithms is composed of five modules: initialization, tracking, mapping, relocalization, and global map optimization. Since each vSLAM algorithm employs different methodologies for each module, features of a vSLAM algorithm highly depend on the methodologies employed as it will be seen in the following chapters.

3.2. VSLAM algorithms

In this section we will briefly explain the algorithms that are going to be implemented in our system, and studied as an objective of this project. These bunch of SLAM algorithms use the camera of the system to map and localize the robot. In the following point we will explain three concrete vSLAM algorithms. These particular algorithms will be:

- PTAM
- ORB-SLAM
- LSD-SLAM

So from now on this vSLAM section will be divided in three different subsections, defining its features and its implementation in the ErleCopter using Linux.

3.2.1 PTAM

3.2.1.1 Introduction

One of the various implementations that will be carried out of Monocular SLAM is the PTAM, mainly developed by Georg Kein and David Murray [18].

PTAM stands for Parallel Tracking and Mapping. It is a technology and algorithm that estimates the position of a camera in a three-dimensional environment and to map the position of the points of the visible objects by analyzing and processing information from a video sequence that can be also done at real time.

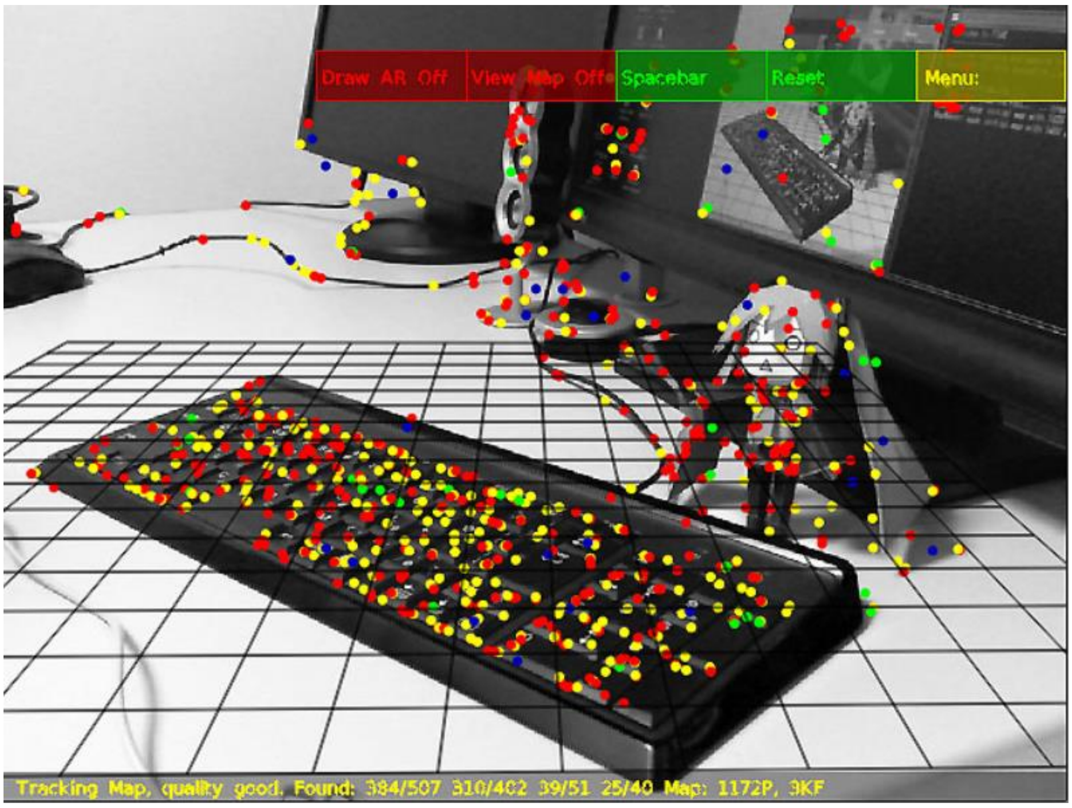
As we have just defined, the process is actually split into two different actions: tracking and mapping.

With the camera moving in the 3D space, it is possible to measure its own position via triangulation and stereo initialization techniques when the same scene is viewed from different points of view. This process is the camera tracking, which aims to calculate as accurately as possible its relative position to the other objects and the movement of the camera in real-time.

The second task is the mapping of the 3D environment in which the camera moves. The simplest way to do so is to measure the position of certain point-features, while other techniques are able to detect straight lines or even extract 3D mesh information from the video stream.

Tracking and mapping are clearly mutually dependent; this means that the camera position is expressed in terms of relative distance from some fixed environment points, while the camera position needs to be known in order to make the mapping of new features possible.

The objective of PTAM is therefore to perform the tracking and mapping tasks in parallel. This method allows a precise and robust real-time tracking, together with an accurate points-based map of the environment.



```
Tracking Map, quality good. Found: 384/507 310/402 39/51 25/40 Map: 1172P, 3KF
```

Figure 3. 1 PTAM Keypoints tracked. Interface.

An important quality of the PTAM method is the fact that mapping is performed only when there are free resources on the background processing thread. This allows the tracking system to follow the camera in real-time regardless of the complexity of the scene, achieving constant frame-rate output particularly useful for Augmented Reality applications. On the other hand, if the camera is stationary in an already-mapped environment, the background thread will allocate resources to analyze again old information in order to improve the quality of the map.

3.2.1.2 PTAM Algorithm

The PTAM algorithm will be divided in three main phases: Initialization, Tracking and Mapping.

To initialize the algorithm PTAM uses a standard five-point stereo algorithm between two keyframes, developed by Subbaarao ,Meer and Genc.

This phase is therefore quick and simple, and it consists on the addition of the very first points to the map so it can be improved adding new features.

It is very important to remark that the distance between the two initialization keyframes will affect the internal scale representation of the system. Although this is not a crucial factor for tracking purposes: the distance can be set to an arbitrary value.

The tracking phase works on each frame using 3D point features. Then the acquired image is processed to generate a pyramid containing multiple levels of the frame at different resolutions. This technique gives the system robustness to scale changes, as each point feature can be matched at multiple distances and resolutions.

Then, after getting the first camera pose estimation, as we already know, based on a small number of points, the pose location is computed. Using the new pose location estimation it is possible to accurately get or predict the position of a larger number of features in the highest resolution level of the pyramid, increasing the accuracy of the camera pose estimation. So now, a new more accurate location is computed and updated.

It is remarkable that when the tracking is lost, the system tries to get a new pose initial estimation as soon as possible.

Once the initialization process has finished, the mapping phase starts. In such phase, the algorithm adds as many point features as possible. The main characteristics of the mapping process are:

- It works on keyframes rather than on every frame sent by the camera, making the calculation very robust, however it won´t be always a real time calculation.
- The keyframes chosen have better average quality than the other frames. This makes it possible to obtain more accurate maps.
- The keyframes can be revisited when there are no new areas to explore, to improve the generated maps.

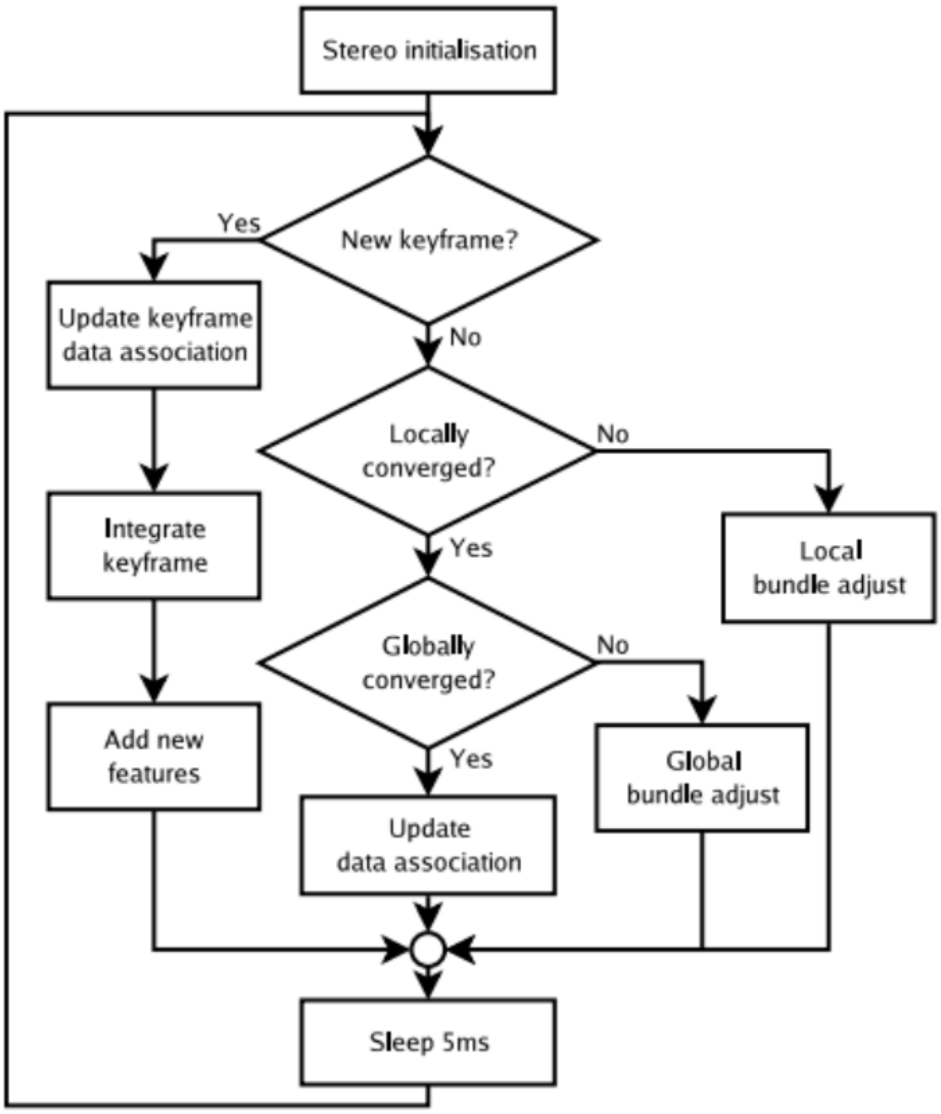


Figure 3. 2 PTAM Block Diagram, ilustrating the monocular worklow steps.

PTAM system is able to map previously unexplored regions automatically. The advantage is that creating this new portions of the map won`t affect the performance of the tracking system as this is done in parallel, by a different thread, in the background.

Keyframes are separated each other by at least 20 frames and a minimum camera distance in order to eliminate stationary camera map corruption. The mapping thread is also able to re-project point features that were not taken into account by the tracking.

If the camera is located in an already explored region in the map, a background threat reanalyzes it, in order to improve the accuracy of the map, adding more feature points. This process allows achieving a good compromise between map expansion speed and accuracy.

3.2.1.3 Simulation Implementation and launch file

Once the Gazebo simulator is installed, with all the necessary features, the PTAM algorithm can be implemented and launched, so any application can be implemented now using the information that comes from such algorithm.

First of all, we need to take a look at the topics provided by the camera using ROS. As we defined some chapters before, the Erlecopter has a camera integrated in the brain. The problem is that normally the raw image from the camera driver is not what it is wanted for visual processing, but rather an undistorted and (if necessary) delayered image. Therefore it is necessary to use the package image_proc. If you are running it on a robot, it’s highly recommended to run there such package. Generally the driver publishes topics */my_camera/image_raw* and */my_camera/image_info* so the command in such general case should be:

```
$ ROS_NAMESPACE =my_camera rosrun image_proc image_proc
```

The topics to which it subscribes in the Erlecopter are *erlecopter/front/image_front_raw* and *erlecopter/front/camera_front_info*, and that’s why it is needed to make a “remap” for such topics as following:

```
<launch>
<remapfrom="/erlecopter/front/image_front_raw"
to="/erlecopter/image_raw"/>
<remap
from="/erlecopter/front/camera_front_info"to="/erlecopter/camera_info"/>
```

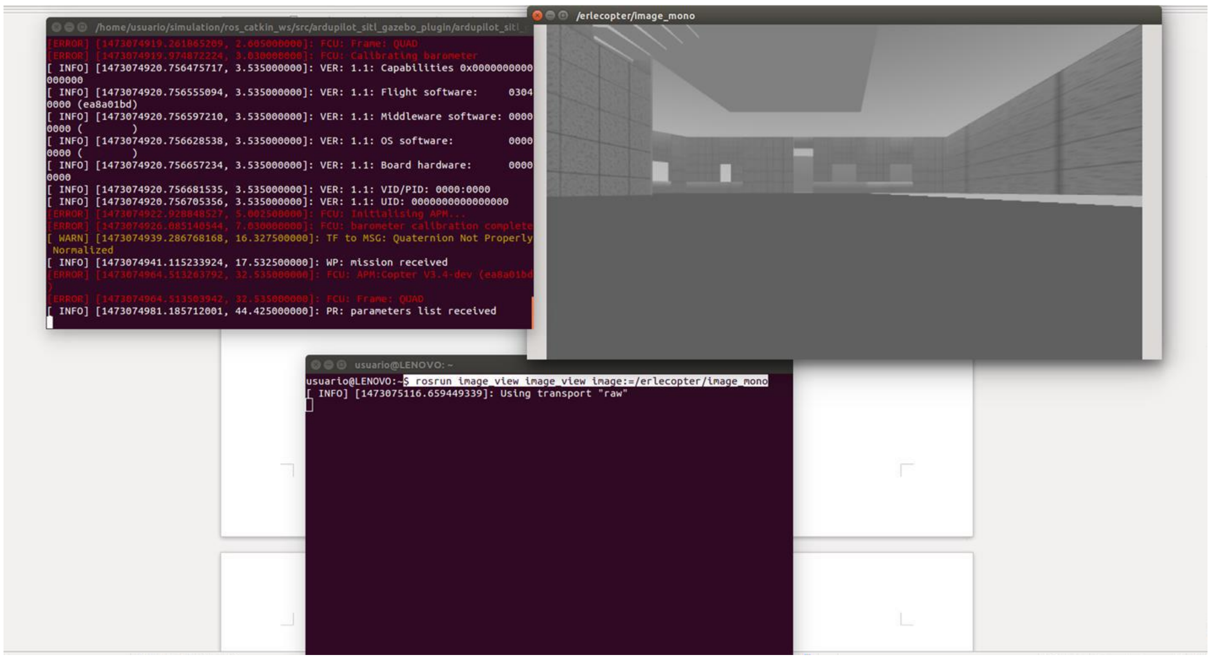
Now, if the simulator is launched and the command rostopic list is executed, it can be seen that now the named topics are *erlecopter/image_raw* and *erlecopter/camera_info*.

So in another terminal we have to execute the following command:

```
$ ROS_NAMESPACE=erlecopter rosrun image_proc image_proc
```

This way will get a black and white image, with its information contained in the topic */erlecopter/image_mono*. To visualize it, we run the following command:

```
$ rosrun image_view image_view image:=/erlecopter/image_mono
```


/home/usuario/simulation/ros_catkin_ws/src/ardupilot_sitl_gazebo_plugin/ardupilot_sitl_
/erlecopter/image_mono
usuario@LENOVO: ~

```
[ERROR] [1473074919.261865209, 2.605000000]: FCU: Frame: QUAD
[ERROR] [1473074919.974872224, 3.030000000]: FCU: Calibrating barometer
[ INFO] [1473074920.756475717, 3.535000000]: VER: 1.1: Capabilities 0x0000000000
000000
[ INFO] [1473074920.756555094, 3.535000000]: VER: 1.1: Flight software:     0304
0000 (ea8a01bd)
[ INFO] [1473074920.756597210, 3.535000000]: VER: 1.1: Middleware software: 0000
0000 (        )
[ INFO] [1473074920.756628538, 3.535000000]: VER: 1.1: OS software:         0000
0000 (        )
[ INFO] [1473074920.756657234, 3.535000000]: VER: 1.1: Board hardware:      0000
0000
[ INFO] [1473074920.756681535, 3.535000000]: VER: 1.1: VID/PID: 0000:0000
[ INFO] [1473074920.756705356, 3.535000000]: VER: 1.1: UID: 0000000000000000
[ERROR] [1473074922.928848527, 5.002500000]: FCU: Initialising APM...
[ERROR] [1473074926.085140544, 7.030000000]: FCU: barometer calibration complete
[ WARN] [1473074939.286768168, 16.327500000]: TF to MSG: Quaternion Not Properly
 Normalized
[ INFO] [1473074941.115233924, 17.532500000]: WP: mission received
[ERROR] [1473074964.513263792, 32.535000000]: FCU: APM:Copter V3.4-dev (ea8a01bd
)
[ERROR] [1473074964.513503942, 32.535000000]: FCU: Frame: QUAD
[ INFO] [1473074981.185712001, 44.425000000]: PR: parameters list received
```

```
usuario@LENOVO:~$ rosrun image_view image_view image:=/erlecopter/image_mono
[ INFO] [1473075116.659449339]: Using transport "raw"
```

Figure 3. 3. Erlecopter monocular camera open in the Ubuntu SO.

Finally, to launch the PTAM algorithm it will be necessary to create a launch file, for example called Erle_Sim.launch, in order to launch the PTAM algorithm:

```
<launch>
<node name="ptam"pkg="ptam" type="ptam" clear_params="true"
output="screen">
<remap from="image" to="$(optenv IMAGE /erlecopter/image_mono)" />
<remap from="pose" to="pose"/>
<rosparam file="$(find ptam)/Cam_PtamFixParams.yaml"/>
</node>
</launch>
```

So now, once the Gazebo simulator is opened, and the image form the camera is converted to black and white, the Erle_Sim.launch file can be run using the following command:

```
$ roslaunch ptam Erle_Sim.launch
```

In order to get the PTAM algorithm working, objects must be added to the Gazebo world, therefore the PTAM algorithm can detect features and carry out the simultaneous mapping and localization.

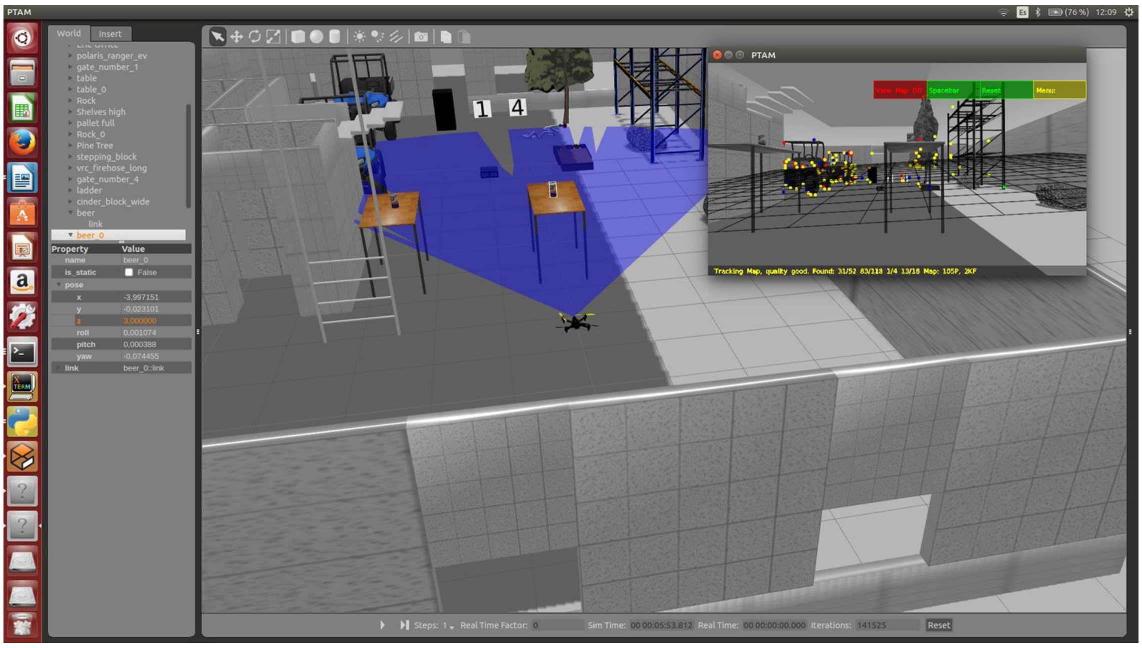


Figure 3. 4. Gazebo simulator with the Erlecopter model with PTAM running on board.

A small window appears, and it can be seen how the PTAM detects the features of the image. Pressing the button *View map off* we can see the window above, were the axis represents the localization of the robot, and the points of the detected features of the image in 3d coordinates.

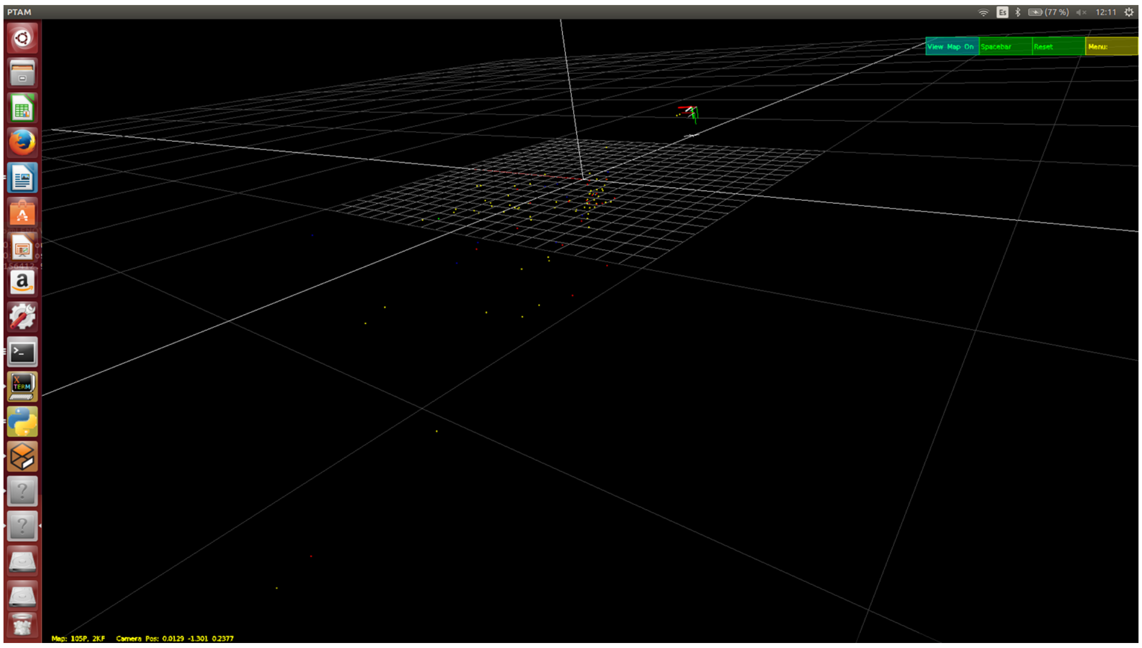


Figure 3. 5. PTAM Map pose estimation view.

3.2.1.4 Real implementation within Erlecopter and launch file

Now, we are going to explain how to use the real camera of the Erlecopter robot. This simply consists on running the algorithm from an external device such a laptop or a simple PC, where the PTAM algorithm must be already installed. The connection between the robot and the device in which the algorithm will run will be a wifi connection.

First of all the camera must be opened, to do so a bunch of steps have to be followed. In a new terminal, the external device is connected to the robot, and then the on-board camera is opened.

```
$ ssh erle@10.0.0.1
$ rosservice call /camera/start_capture
```

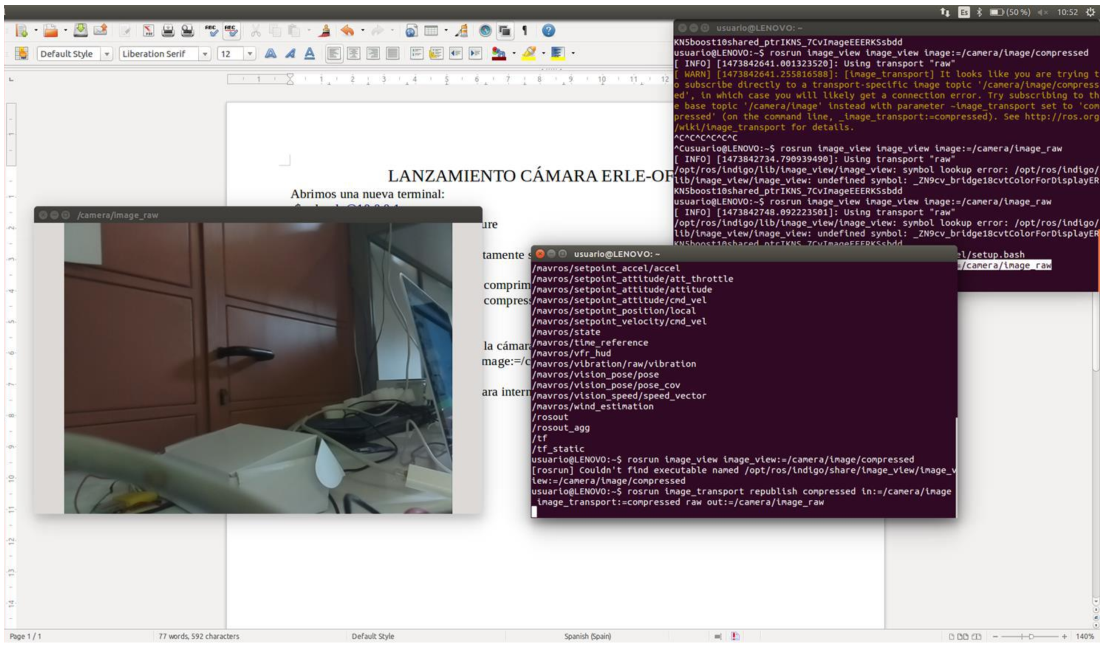
To assure that the camera is opened, we can take a look to camera, where a red light must be on.

After getting this first step done, the image obtained from the camera is in a compressed format, and all the algorithms studied within this thesis work with raw images (particularly PTAM employs raw images in black and white). So the compressed images have to be converted to raw format.

```
$ rosrun image_transport republish compressed in:=/camera/image
_image_transport:=compressed raw out:=/camera/image_raw
```

So now we can visualize the camera in real time.

```
$ source simulation/ros_catkin_ws/devel/setup.bash
$ rosrun image_view image_view image:=/camera/image_raw
```



LANZAMIENTO CÁMARA ERLE-OF

Abrimos una nueva terminal:

```
KN5boost10shared_ptrIKNS_7CvImageEEERKSsbdd
usuario@LENOVO:~$ rosrun image_view image_view image:=/camera/image/compressed
[ INFO] [1473842641.001323520]: Using transport "raw"
[ WARN] [1473842641.255816588]: [image_transport] It looks like you are trying t
o subscribe directly to a transport-specific image topic '/camera/image/compress
ed', in which case you will likely get a connection error. Try subscribing to th
e base topic '/camera/image' instead with parameter ~image_transport set to 'com
pressed' (on the command line, _image_transport:=compressed). See http://ros.org
/wiki/image_transport for details.
^C^C^C^C^C^C^C
^Cusuario@LENOVO:~$ rosrun image_view image_view image:=/camera/image_raw
[ INFO] [1473842734.790939490]: Using transport "raw"
/opt/ros/indigo/lib/image_view/image_view: symbol lookup error: /opt/ros/indigo/
lib/image_view/image_view: undefined symbol: _ZN9cv_bridge18cvtColorForDisplayER
KN5boost10shared_ptrIKNS_7CvImageEEERKSsbdd
usuario@LENOVO:~$ rosrun image_view image_view image:=/camera/image_raw
[ INFO] [1473842748.092223501]: Using transport "raw"
/opt/ros/indigo/lib/image_view/image_view: symbol lookup error: /opt/ros/indigo/
lib/image_view/image_view: undefined symbol: _ZN9cv_bridge18cvtColorForDisplayER
```

```
/mavros/setpoint_accel/accel
/mavros/setpoint_attitude/att_throttle
/mavros/setpoint_attitude/attitude
/mavros/setpoint_attitude/cmd_vel
/mavros/setpoint_position/local
/mavros/setpoint_velocity/cmd_vel
/mavros/state
/mavros/time_reference
/mavros/vfr_hud
/mavros/vibration/raw/vibration
/mavros/vision_pose/pose
/mavros/vision_pose/pose_cov
/mavros/vision_speed/speed_vector
/mavros/wind_estimation
/rosout
/rosout_agg
/tf
/tf_static
usuario@LENOVO:~$ rosrun image_view image_view:=/camera/image/compressed
[rosrun] Couldn't find executable named /opt/ros/indigo/share/image_view/image_v
iew:=/camera/image/compressed
usuario@LENOVO:~$ rosrun image_transport republish compressed in:=/camera/image
 image_transport:=compressed raw out:=/camera/image_raw
```

Figure 3. 6. Erelcopter monocular camera launched in Ubuntu.

3.2.1.5 Results obtained

The interface of the algorithm is composed by just one window that contains different options which can be clicked on. The default window, showed in figure 3.1, shows the keyframes detected by the algorithms. Clicking on the View Map Off , it gets the window of figure 3.5, where the axis plotted on such figure, represent the current position, and the path generated by the algorithm. Besides, the generated Map of points is also represented in this figure.

Now, as it will be done with each of the three algorithms studied in this project, three different paths will be implemented for the quadcopter to fly through. This will allow taking notice of the main advantages and disadvantages of each one, as well as its accuracy and performance in specific conditions. Figures from 3.7 to 3.9 show the implemented paths for this study.

Path 1:

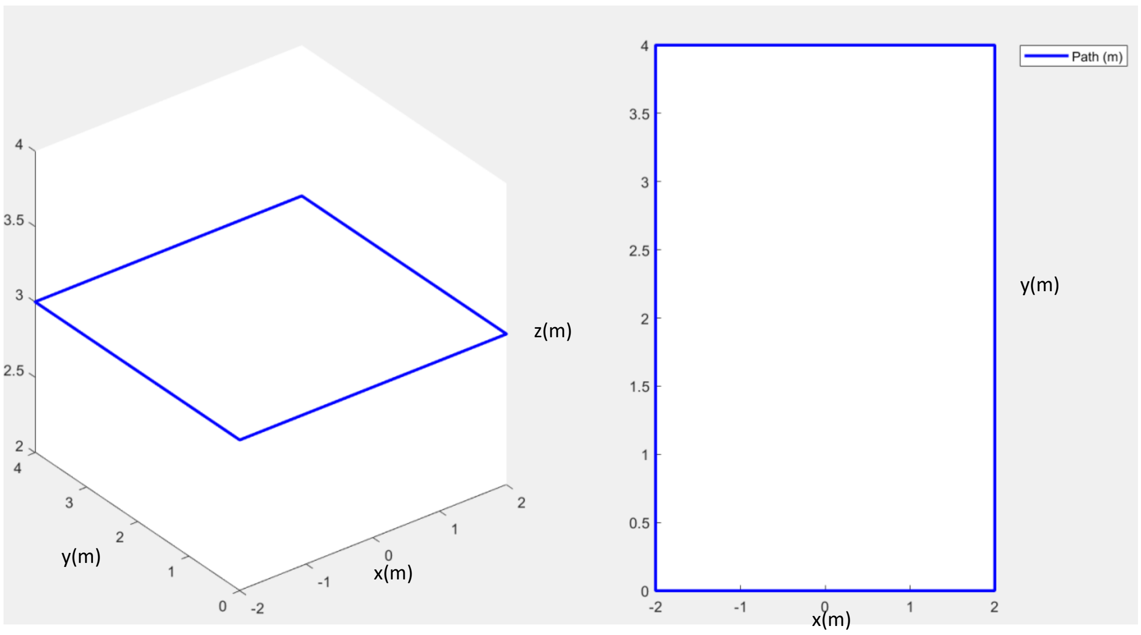


Figure 3. 7. Path 1 built for the monocular test.

Path 2:

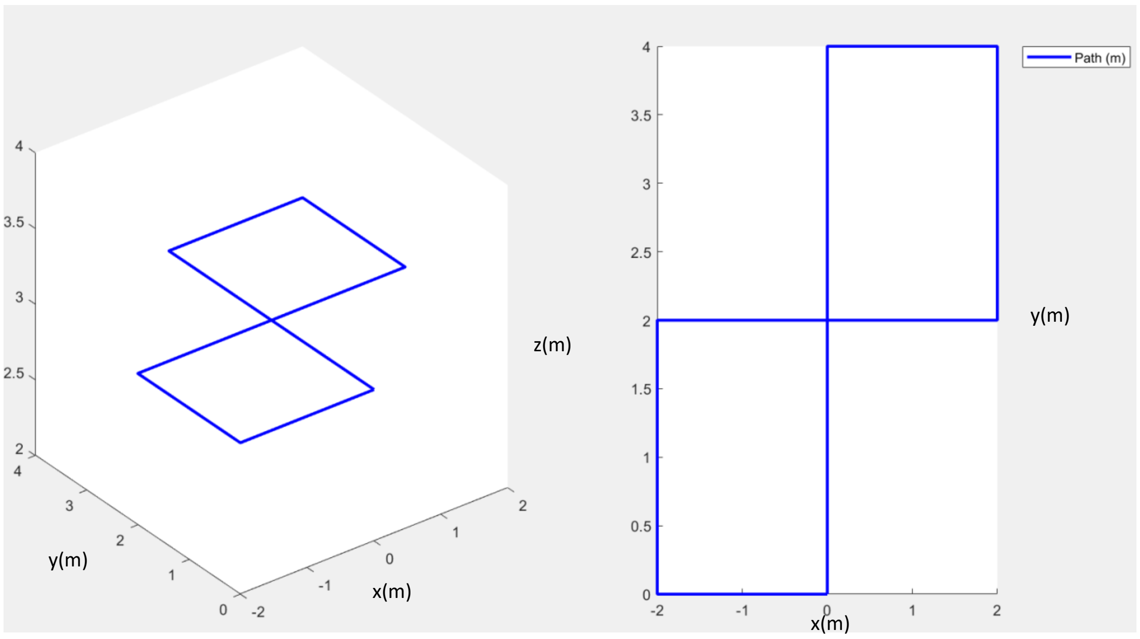


Figure 3. 8. Path 2 built for the monocular test.

Path 3:

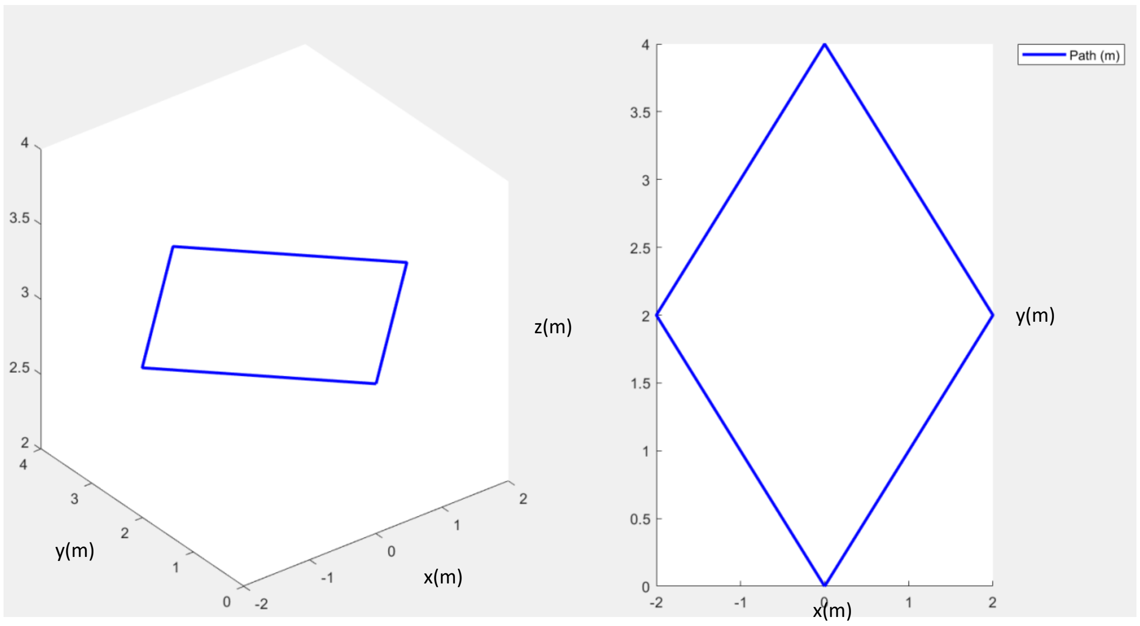


Figure 3. 9. Path 3 built for the monocular test.

The results after carrying out the presented paths and implementing the PTAM within the ErleCopter, once the Gazebo simulator has started are shown below.

Result Path 1:

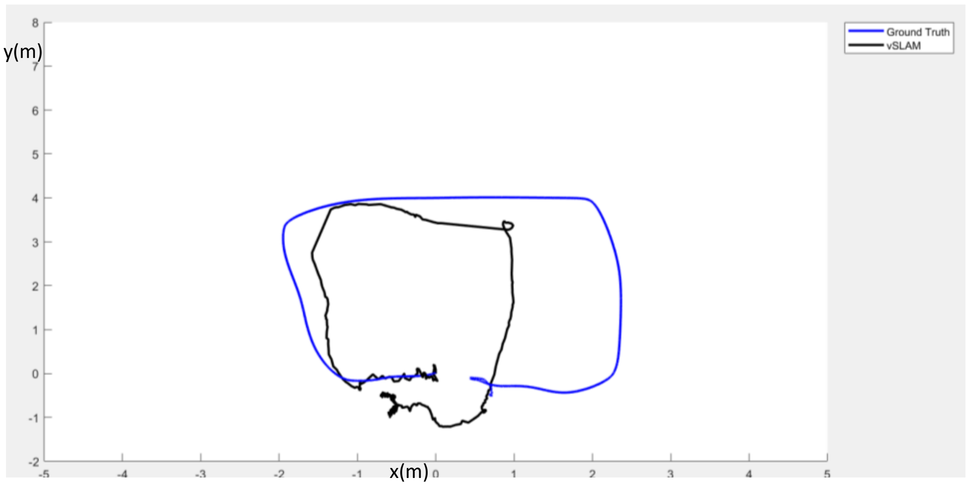


Figure 3. 10. Results PTAM path 1 X-Y view.

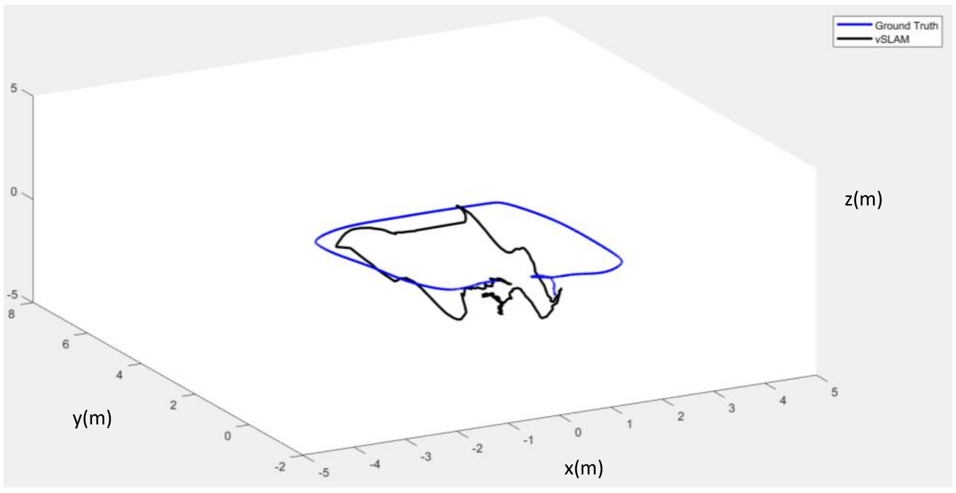


Figure 3. 11.Results PTAM path 1 3D view.

The blue line represents the ground truth of the robot, so to say, the path really followed by the quadcopter, gotten from the gazebo simulator, while the black line represents the prediction of the localization given by the PTAM algorithm. Thus, we can compare the accuracy of the PTAM algorithm, not only for just one case, but for three different paths, so a general overview of its performance can be obtained.

Result Path 2:

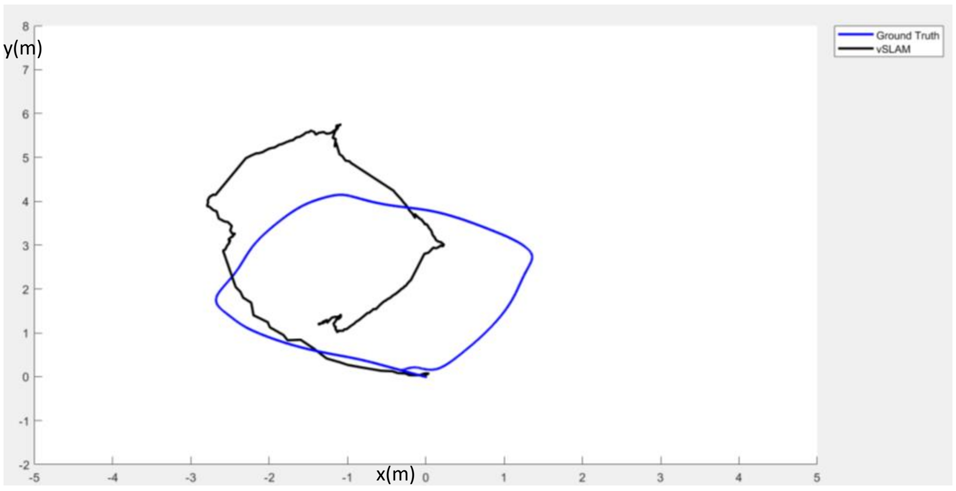


Figure 3. 12. Results PTAM path 2 X-Y view.

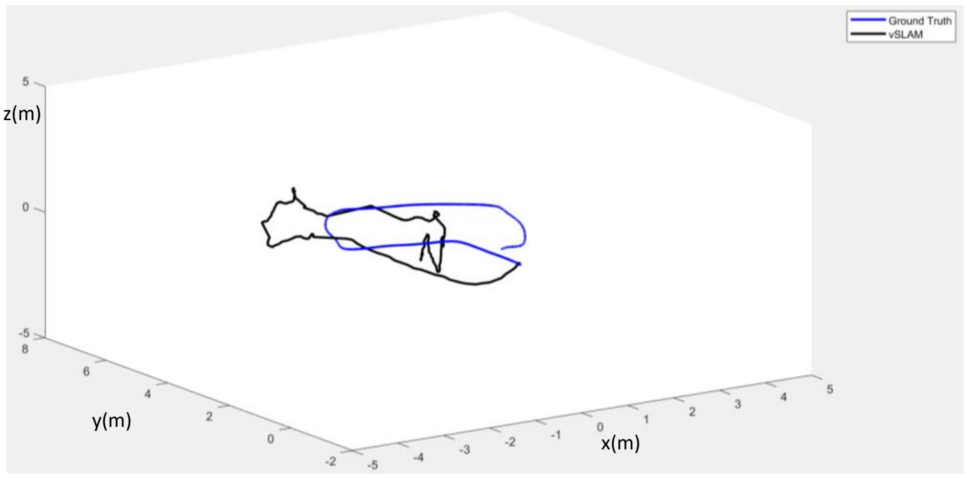


Figure 3. 13. Results PTAM path 2 3D view.

The real path followed by the drone seems to be smoother than the predicted localization given by PTAM. The algorithm predicts a small movement up and down along the path.

Result Path 3:

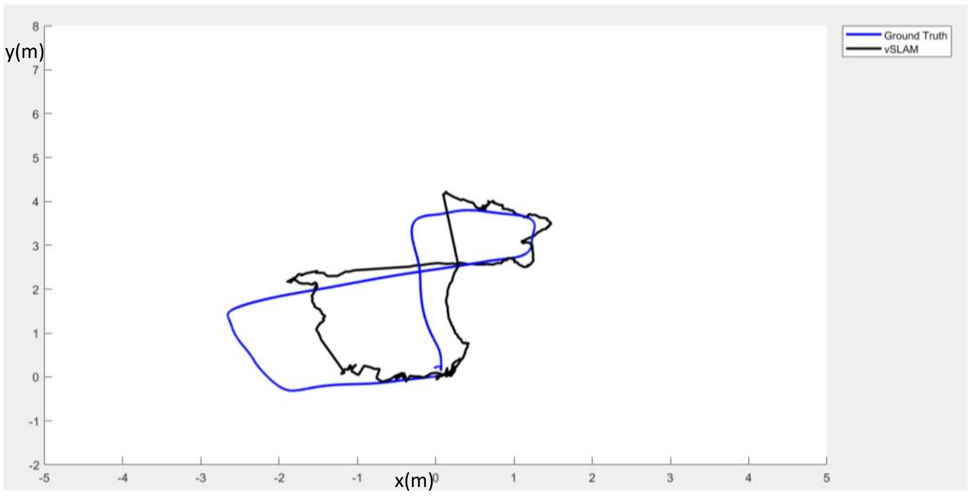


Figure 3. 14. Results PTAM path 3 X-Y view.

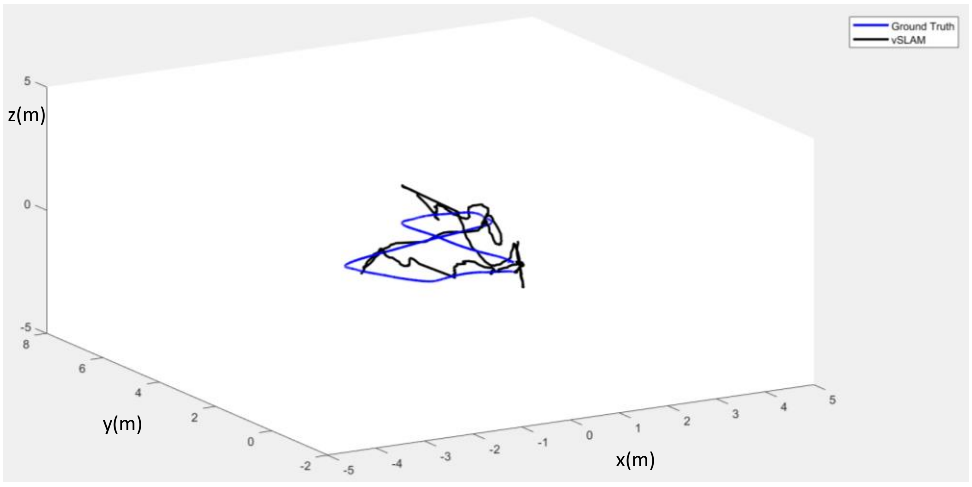


Figure 3. 15. Results PTAM path 3 3D view.

After testing the algorithm using the implemented paths, it can be said that it is computationally lighter than the others. The track gets lost, time to time, even though the movement of the camera is not fast. Besides, it needs a big amount of features to start tracking, and to recover from a loss. Nevertheless, its accuracy will be compared to the ORB-SLAM and LSD-SLAM in the following chapters.

3.2.2 ORB-SLAM

3.2.2.1 Introduction

ORB-SLAM is a widely known method for Monocular SLAM algorithms. This method is based on the recognition of features, besides it is able to operate in real time, and within environments both small and large, outdoors and indoors. Of course, given the purpose of this project it will be used for indoors environments. It uses the same features for all the SLAM tasks: tracking, mapping localization, and close- loop detection. So it detects the keyframes to generate the maps. These keyframes are changing only if the scene content changes.

ORB aims to estimate the trajectory of the camera while it builds the environment in which it “sails”. One of the most important concepts is the bundle adjustment (BA). It is a technique able to precisely estimate position and a geometric reconstruction. Its importance in the ORB is that it is used to optimize the maps and calculated trajectories when a loop closure is detected. Currently, very precise results can be obtained without requiring a high computational cost. To summarize, these are the most highlighted points of this technique:

- It uses the same features for all the SLAM tasks. It makes the system more efficient, simple and reliable.
- It is able to operate in real time.
- Localization in real time for locations previously “browsed”, independently of the existence of illumination or the angle of vision changes.

3.2.2.2 ORB-SLAM diagram

In figure 3.16 it is shown the diagram of this algorithm. Mainly, five modules can be distinguished:

- Tracking
- Local Mapping
- Loop Closing
- Map
- Place Recognition

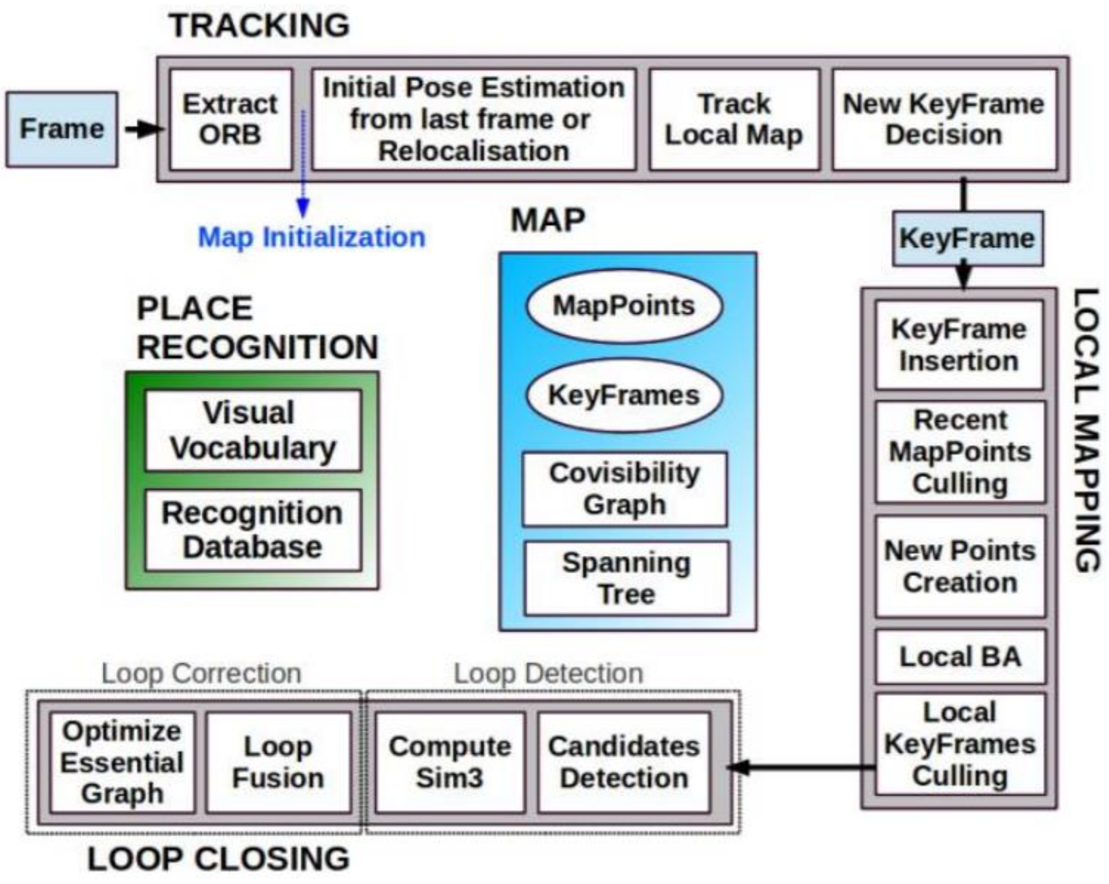


Figure 3. 16. ORB-SLAM Block diagram illustrating the workflow steps.

It uses three threads running in parallel: Tracking, Local Mapping and Loop closing. **The tracking** thread locates the position of the camera in each image, and it “decides” when it’s necessary to include a new keyframe. When the system lost the tracking, the Place recognition module represents a global relocation. Once an initial estimation of the camera is achieved and the features are found, the local map is retrieved using the previously stored keyframes in the database.

The second parallel thread is **the Local Mapping** which processes the new keyframes and represents the local BA (bundle adjustment) to get an optimal reconstruction around the camera position. Then a data filtering on the stored information during the tracking is carried out, aiming to preserve the points with the highest quality. In this module the redundant points are also wiped out.

The third loop is the Loop closing, which looks for the existence of loops each time a new keyframe is gotten. Finally, the closed loop is incorporated to the global map graph.

After this explanation the information contained in each point and keyframe will be briefly remarked below:

Each point in the map contains the following information:

- Its 3D location referred to the global reference system.
- The direction of the view vector.
- The ORB associated descriptor.
- The maximum and minimum distance at which the point can be observed.

Each keyframe contains:

- The transformation of the real world coordinates system referred to the camera reference system.
- The intrinsic parameters of the camera, such as the focal length.
- The ORB features extracted from the image, either if they are o not associated to a map point.

The philosophy of this vSLAM technique is to generously create map points and keyframes, given the fact that after having done such process, a filtering on these keyframes is carried out, wiping the redundant points out. As a result, the obtained map is flexible and can be expanded while the environment is being explored.

3.2.2.3 Algorithm ORB-SLAM

1. Map initialization

The objective of such initialization is to obtain the relative position between two frames to triangulate a bunch of points that are part of the map. The algorithm has to be able to achieve such initialization autonomously and independently of the environment.

To do so, this algorithm stands on the execution in parallel of two geometric models, the first one with the homograph matrix (Hcr) for flat scenes and the other one with the fundamental matrix (Fcr) for not flat scenes. The method to recognize which model is applicable to the image is heuristic and it is the following:

1) The initial correspondences are found. All the ORB features are extracted from the current frame (Fc) and search for matches xc<->xr in the reference frame (Fr). If not enough matches are found, reset the reference frame.
2) Both models are executed at a time. Compute in parallel threads a homography Hcr and a fundamental matrix Fcr:

$$x_c = H_{cr} \cdot x_r \qquad x_c^T \cdot F_{cr} \cdot x_r = 0$$

To make homogeneous the procedure for both models, the number of iterations is prefixed and the same for both models, along with the points to be used at each iteration, 8 for the fundamental matrix, and 4 of them for the homography. At each iteration we compute a score SM for each model M (H for the homography, F for the fundamental matrix):

$$S_M = \sum_i^n (\rho_M (dcr^2(x_c^i, x_r^i, M)) + \rho_M(dcr^2(x_c^i, x_r^i, M))$$

$$\rho_M(d^2) = \begin{cases} \Gamma - d^2 \text{ if } d^2 < T_M \\ 0 \text{ if } d^2 \geq T_M \end{cases}$$

where d_{cr}^2 and d_{cr}^2 are the symmetric transfer errors from one frame to the other. TM is the outlier rejection threshold based on the χ test at 95% (TH = 5.99, TF = 3.84, assuming a standard deviation of 1 pixel in the measurement error). Γ is defined equal to TH so that both models score equally for the same d in their inlier region, again to make the process homogeneous. We keep the homography and fundamental matrix with highest score. If no model could be found (not enough inliers), we restart the process again from step 1.

3) The homography matrix will be chosen if $R_H>0.45$, if not the model of the fundamental matrix will be chosen.

$$R_H=\frac{S_H}{S_H+S_F}$$

4) Once a model is selected we retrieve the motion hypotheses associated. The system will try to triangulate directly the solutions obtained previously, looking for the best one. In case of not finding any accurate solution, it will return to the step number 1.
5) Finally the BA is represented.

2. Tracking

In order to carry out the tracking module, the first thing that should be done is the initial pose estimation. It can start from the previous frame if such frame is satisfactory, if not, the tracking is lost, so a candidate has to be found in the keyframes database.

Once the initial camera pose and a bunch of associated features are estimated, the local map to which such frame corresponds can be projected, and look for more correlations between the points of the map. Once the correlations are found, the map is updated.

Eventually, this module decides if the current frame will become or not a keyframe. To insert a new keyframe the following conditions must be accomplished:

- More than 20 frames must have passed since the last global relocation.

- The local mapping must been deactivated or more than 20 frames must have passed since the last insertion of a new keyframe.

- The current frame must have located at least 50 points.

- The current frame must have at least the 90% of te points contained in the reference frame.

These conditions assure that there were visual changes, a good relocation and a good tracking. Besides, the second condition assures that there won’t be a new keyframe while the local mapping is running.

3. Local Mapping

This module inserts the new keyframes. As it has been observed in previous chapters, one of the most characteristics aspects is the exhaustive point and keyframes filtering that it does. In order to remain part of the map, the points have to pass several conditions during the first three keyframes added after such point became part of the map. The conditions are the following:

- The tracking module must find the treated point in more than the 25% of the frames in which such point should be visible.

- The point must be observed at least in the three following frames, after such point was incorporated to the map.

Once the point is added to the map, it can be wiped out if at any time it is part of less than three keyframes.

Besides, the keyframes must accomplish a bunch of conditions in order to not be eliminated. Those keyframes in which the 90% of their points in the map were observed with the same or a smaller scale in at least another 3 keyframes will be eliminated. This condition assures that the most accurate keyframes will remain.

This module is also in charge of the local BA that optimizes the current keyframe.

4. Loop Closing

The aim of this module is to detect loops between the current keyframe and the last one, processed by the local mapping module.

First, the candidate loops to be closed are detected. Then a transformation between the current keyframe and the loop keyframe is calculated. Finally each point of the map is transformed according to the corrections obtained from any of the keyframes in which this point is observed.

3.2.2.4 Simulation implementation and launch file

Similarly to the PTAM section, the Gazebo simulator with the ErleCopter has to be launched. It is important first to highlight that the ORB algorithm subscribes to the topic */camera/image_raw*, while the published topic of the ErleCopter camera is */erlecopter/front/image_front_raw*. So it is needed a simply remap of this topic.

For example in the provided file erlecopter_spawn.launch, that launches the simulator, we can simply add the following command to remap the topic.

```
<launch>
<remap from="/erlecopter/front/image_front_raw" to="/camera/image_raw"/>
…
```

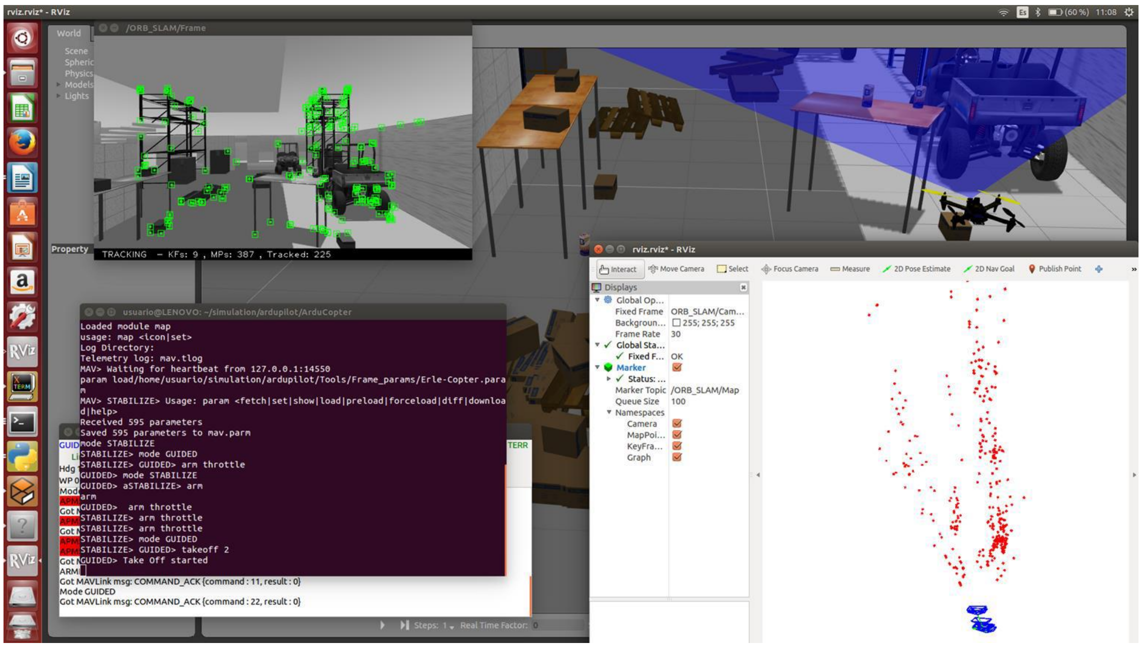
After launching the simulator, the ORB SLAM can be executed as follows.

```
$  roslaunch ORB_SLAM Erle_Simulador.launch
```

Where the file Erle_Simulator.launch doesn’t need any changes, and it is shown below.

```
<launch>
<node pkg="image_view" type="image_view" name="image_view" respawn="false"
output="log">
<remap from="/image" to="/ORB_SLAM/Frame" />
<param name="autosize" value="true"/>
</node>
<node pkg="rviz" type="rviz" name="rviz" args="-d $(find ORB_SLAM)/Data/rviz.rviz"
output="log">
</node>
<node pkg="ORB_SLAM" type="ORB_SLAM" name="ORB_SLAM"
args="Data/ORBvoc.txtData/Settings.yaml" cwd="node" output="screen"/>
</launch>
```

Now, that both the simulated world and the ORB algorithm are working, the SLAM algorithm can be studied and it can be used to develop applications of mapping and localization.



```
TRACKING  - KFs: 9 , MPs: 387 , Tracked: 225
```

```
usuario@LENOVO: ~/simulation/ardupilot/ArduCopter
Loaded module map
usage: map <icon|set>
Log Directory:
Telemetry log: mav.tlog
MAV> Waiting for heartbeat from 127.0.0.1:14550
param load/home/usuario/simulation/ardupilot/Tools/Frame_params/Erle-Copter.para
m
MAV> STABILIZE> Usage: param <fetch|set|show|load|preload|forceload|diff|downloa
d|help>
Received 595 parameters
Saved 595 parameters to mav.parm
mode STABILIZE
STABILIZE> mode GUIDED
STABILIZE> GUIDED> arm throttle
GUIDED> mode STABILIZE
GUIDED> aSTABILIZE> arm
arm
GUIDED>  arm throttle
STABILIZE> arm throttle
STABILIZE> arm throttle
STABILIZE> mode GUIDED
STABILIZE> GUIDED> takeoff 2
GUIDED> Take Off started
```

```
Got MAVLink msg: COMMAND_ACK {command : 11, result : 0}
Mode GUIDED
Got MAVLink msg: COMMAND_ACK {command : 22, result : 0}
```

```
Displays
  Global Op...
    Fixed Frame    ORB_SLAM/Cam...
    Backgroun...   255; 255; 255
    Frame Rate     30
  Global Sta...
    Fixed F...     OK
  Marker
    Status: ...
    Marker Topic   /ORB_SLAM/Map
    Queue Size     100
    Namespaces
      Camera
      MapPoi...
      KeyFra...
      Graph
```

Figure 3. 17. ORB-SLAM launched in Gazebo with the Erlecopter Model.

3.2.2.5 Real implementation within Erlecopter and launch file

Now, as we did in the PTAM section above, the ORB SLAM will be execute from an external device in the real Erlecopter. To do so we need a WiFi connection between the device and the robot.

First of all the camera of the robot must be opened, as it was done in the 3.1.2 section. Once the camera of the drone is opened the second step is to convert the raw image provided to a mono image, given that the algorithm works with mono images.

```
$ ROS_NAMESPACE=camera rosrun image_proc image_proc
```

After executing this command it can be seen that the topic */camera/image_mono* is now being published.

To visualize the image, just execute the following, using the image_view package.

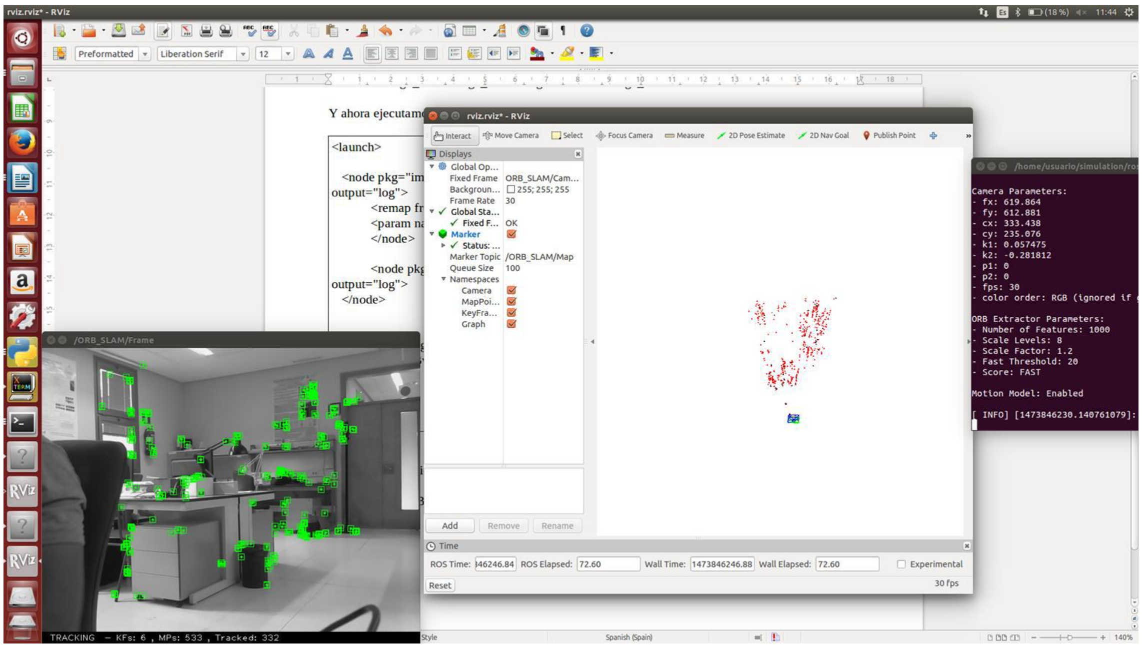
```
$ source simulation/ros_catkin_ws/devel/setup.bash
$ rosrun image_view image_view image:=/camera/image_mono
```

Finally to get the ORB algorithm running the file below has to be launched to visualize the ORB viewer.

```
<launch>
<node pkg="image_view" type="image_view" name="image_view"
respawn="false" output="log">
<remap from="/image" to="/ORB_SLAM/Frame" />
<param name="autosize" value="true"/>
</node>
<node pkg="rviz" type="rviz" name="rviz" args="-d $(find
ORB_SLAM)/Data/rviz.rviz" output="log">
</node>
<node pkg="ORB_SLAM"type="ORB_SLAM" name="ORB_SLAM"
args="Data/ORBvoc.txt Data/Erle.yaml" cwd="node" output="screen"/>
</launch>
```

```
$ source simulation/ros_catkin_ws/devel/setup.bash
$ roslaunch ORB_SLAM Erle_Wifi.launch
```

It can be seen now that the ORB slam algorithm is working in real time, carrying out the mapping and localization modules.



Global Op...	
Fixed Frame	ORB_SLAM/Cam...
Backgroun...	255; 255; 255
Frame Rate	30
Global Sta...	
Fixed F...	OK
Marker	✓
Status: ...	
Marker Topic	/ORB_SLAM/Map
Queue Size	100
Namespaces	
Camera	✓
MapPoi...	✓
KeyFra...	✓
Graph	✓

```
Camera Parameters:
- fx: 619.864
- fy: 612.881
- cx: 333.438
- cy: 235.076
- k1: 0.057475
- k2: -0.281812
- p1: 0
- p2: 0
- fps: 30
- color order: RGB (ignored if

ORB Extractor Parameters:
- Number of Features: 1000
- Scale Levels: 8
- Scale Factor: 1.2
- Fast Threshold: 20
- Score: FAST

Motion Model: Enabled

[ INFO] [1473846230.140761079]:
```

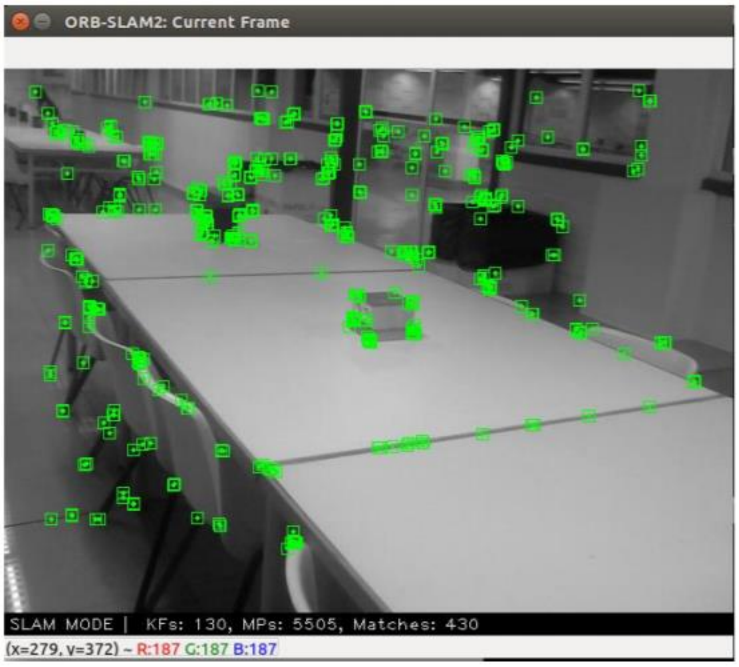
```
TRACKING  - KFs: 6 , MPs: 533 , Tracked: 332
```

Figure 3. 18. ORB – SLAM launched on the real robotic platform.

3.2.2.6 Results obtained

As it can be seen in figure 3.18, the interface of this algorithm is composed by two windows. The first one (composed by little green squares) shows the current frame of the algorithm, from where the following information can be obtained:

- SLAM Mode.
- Number of Keyframes.
- Number of points in the Map.
- Number of found features.
- X and Y position of the cursor on the window.
- RGB information of the point on which the cursor is located.



```
SLAM MODE |  KFs: 130, MPs: 5505, Matches: 430
(x=279, y=372) ~ R:187 G:187 B:187
```

Figure 3. 19. ORB-SLAM current frame window, with the detected Keyframes.

The second window, showed in figure 3.20, contains the generated map of points of the explored environment. In such window, appear several options that allow the user to track the camera.

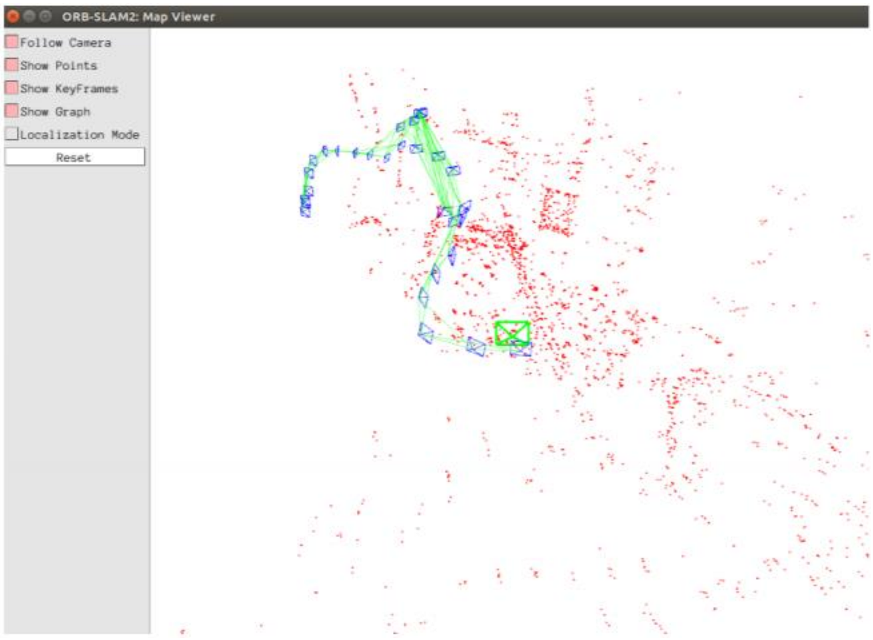


Figure 3. 20. ORB-SLAM tracking and mapping window.

Besides, it is shown the generated map along the movement of the robot. The information given by this second window is:

- Green rectangle: Current position of the camera.
- Blue rectangles: Keyframes.
- Red points: Points of the current local map.
- Black points: Points of the map.

Following the steps from the PTAM chapter, that is to say, taking the same paths developed above, the results after having implemented ORB-SLAM in the simulation are now presented.

Result Path 1:

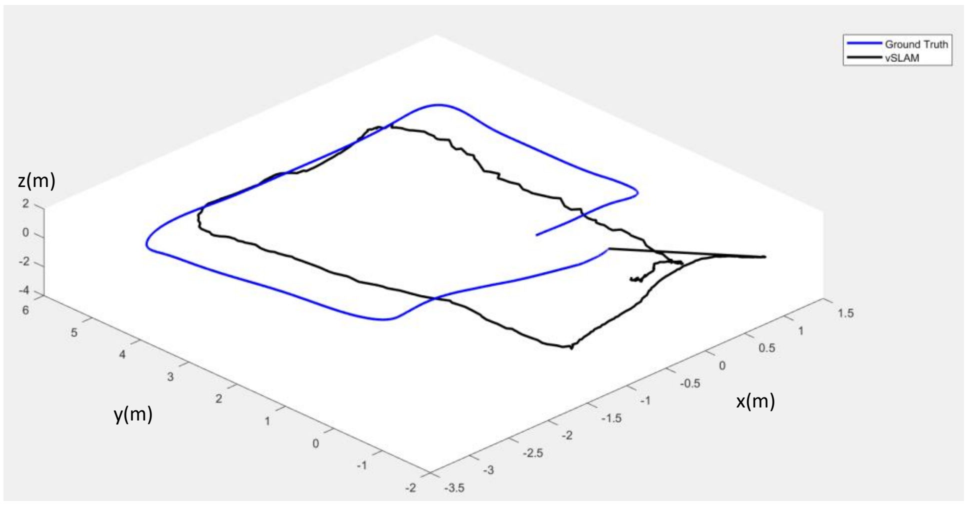


Figure 3. 21. Results ORB-SLAM path 1 3D view.

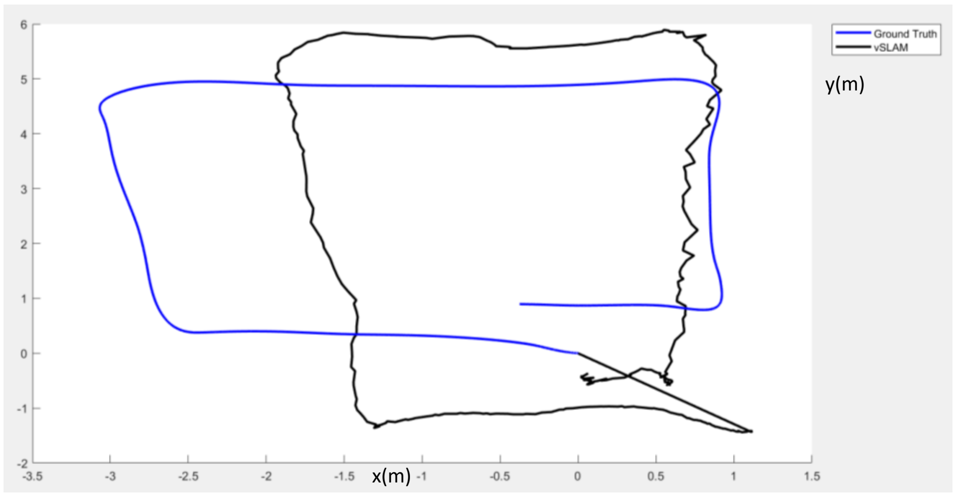


Figure 3. 22. Results ORB-SLAM path 2 X-Y view.

Again, the blue line represents the ground truth of the robotic platform, so to say, the real position of the quadcopter. The black line, in this case, represents the predicted localization given by the ORB-SLAM algorithm along the path.

Result Path 2:

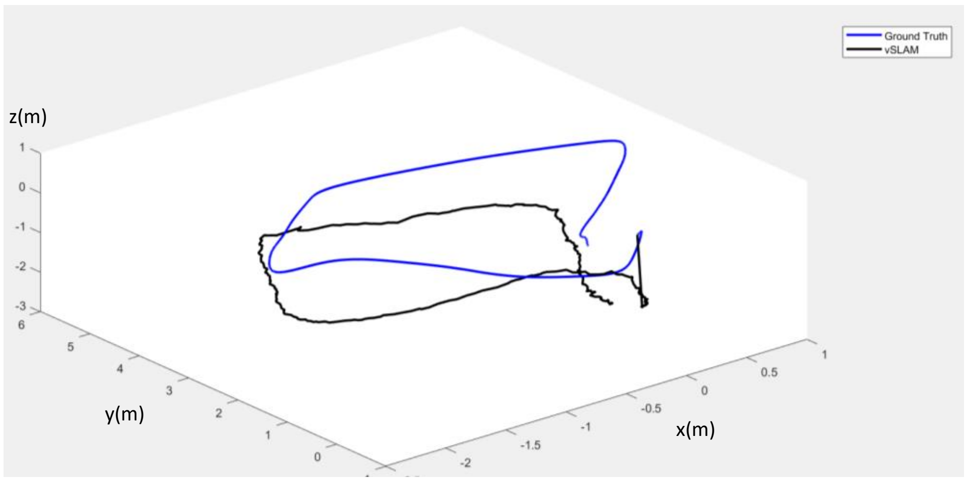


Figure 3. 23. Results ORB-SLAM path 2 3D view.

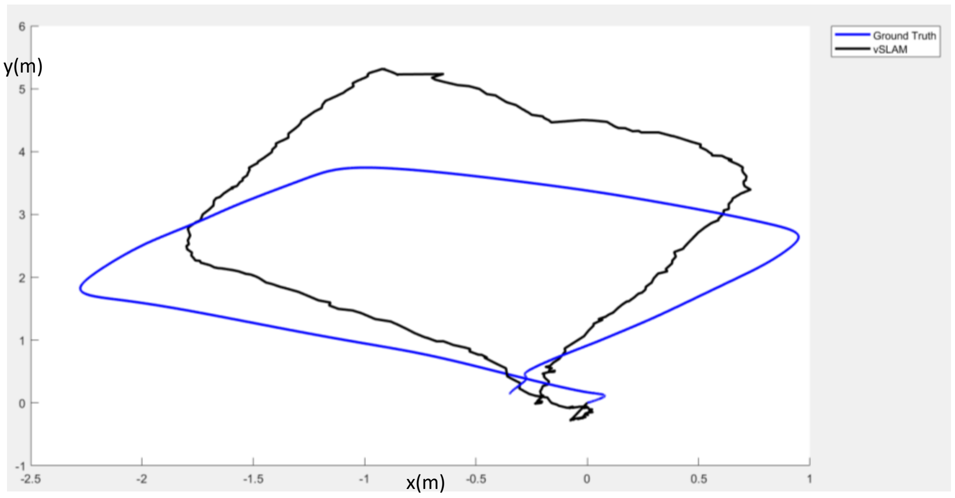


Figure 3. 24. Results ORB-SLAM path 2 X-Y view.

Unlike PTAM, the performance of this algorithm turns to be slightly smoother. Although having again a small deviation along the path, this time sideways, the outcomes of the algorithm, in terms of stability (and not in terms of error) were better.

Result Path 3:

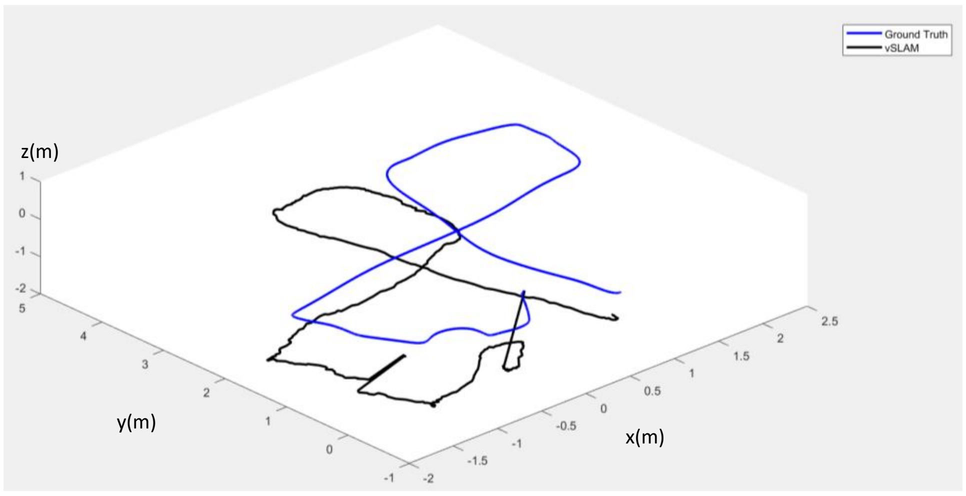


Figure 3. 25. Results ORB-SLAM path 3 3D view.

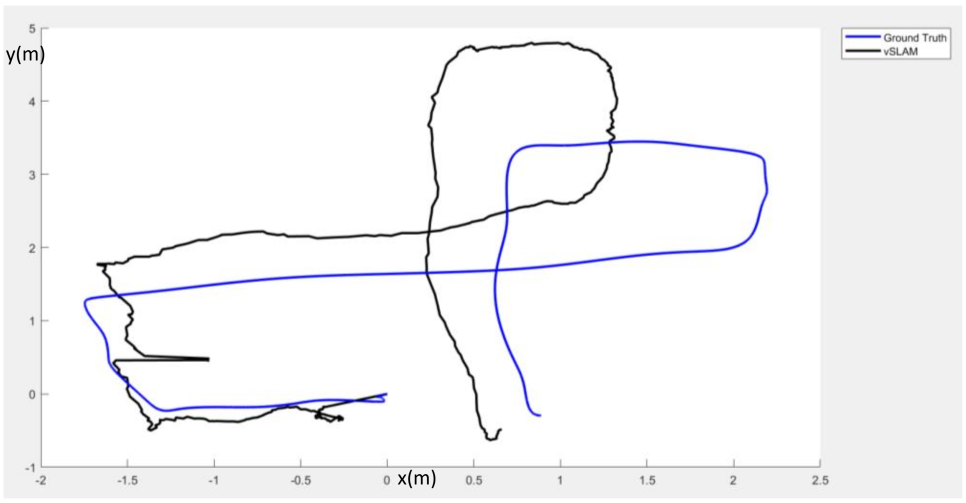


Figure 3. 26. Results ORB-SLAM path 3 X-Y view.

After taking the results obtained from the algorithm, it comes to the conclusion that the ORB algorithm is very sensitive to the strong movements, above all, the rotational movements around the Z axis (yaw angle). However, if the camera is moved smoothly the algorithm works without any lost. It is very computationally heavy, so it needs a high quality processor to be run in.

In the next sections the error will be calculated and compared with the rest of the vSLAM algorithms studied in this project.

3.2.3 LSD-SLAM

3.2.3.1 Introduction

The LSD-SLAM technique presented in this section works completely different than the vSLAM techniques studied before. This method called Large Scale Direct Monocular SLAM builds maps in big scale. Instead of using features, it works on the contrast of the images both for location and mapping. The geometry of the maps is estimated applying filters over the acquired images in gray scale.

Then it takes more information from the geometry and the environment that can be very useful for robots or even for augmented reality purposes.

Like the previous SLAM technique, the world is represented by a number of keyframes connected by position restrictions, which can be optimized using an optimization graph.

3.2.3.2 Algorithm LSD-SLAM

The algorithm can be divided in three main modules: tracking, depth map estimation and map optimization.

The algorithm initialization is carried out through a random intensity map. When the camera is slightly moved, the algorithm blocks the configuration, and the algorithm converges.

The representation of the map is a keyframe graph. Figure 3.27 shows the algorithm scheme.

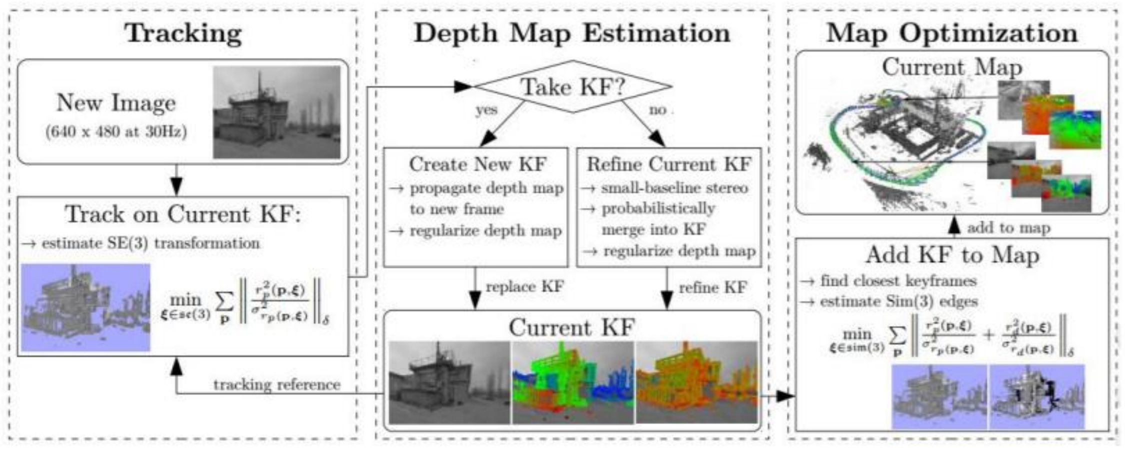

Tracking
New Image
(640 x 480 at 30Hz)
Track on Current KF:
→ estimate SE(3) transformation
$\min_{\xi \in se(3)} \sum_{p} \left\| \frac{r_p^2(p,\xi)}{\sigma^2_{r_p(p,\xi)}} \right\|_\delta$
tracking reference
Depth Map Estimation
Take KF?
yes
no
Create New KF
→ propagate depth map to new frame
→ regularize depth map
Refine Current KF
→ small-baseline stereo
→ probabilistically merge into KF
→ regularize depth map
replace KF
refine KF
Current KF
Map Optimization
Current Map
add to map
Add KF to Map
→ find closest keyframes
→ estimate Sim(3) edges
$\min_{\xi \in sim(3)} \sum_{p} \left\| \frac{r_p^2(p,\xi)}{\sigma^2_{r_p(p,\xi)}} + \frac{r_d^2(p,\xi)}{\sigma^2_{r_d(p,\xi)}} \right\|_\delta$

Figure 3. 27. LSD-SLAM Block Diagram illustrating the workflow steps

1. Tracking

The tracking module tracks the new images obtained by the camera. Such images are estimated as a solid rigid regarding the current keyframe, using the position of the previous keyframe for the initialization.

2. Depth map estimation

This module uses the frames obtained to substitute the current keyframe. The intensity is defined by a pixel by pixel filtering.

If the camera is moving too far from the created map, then it creates a new keyframe of the last frame obtained by the tracking module. Once such frame is taken, its intensity is initialized, projecting its points from the previous keyframe over this one. Finally, the keyframe is substituted, and it is used to track new frames.

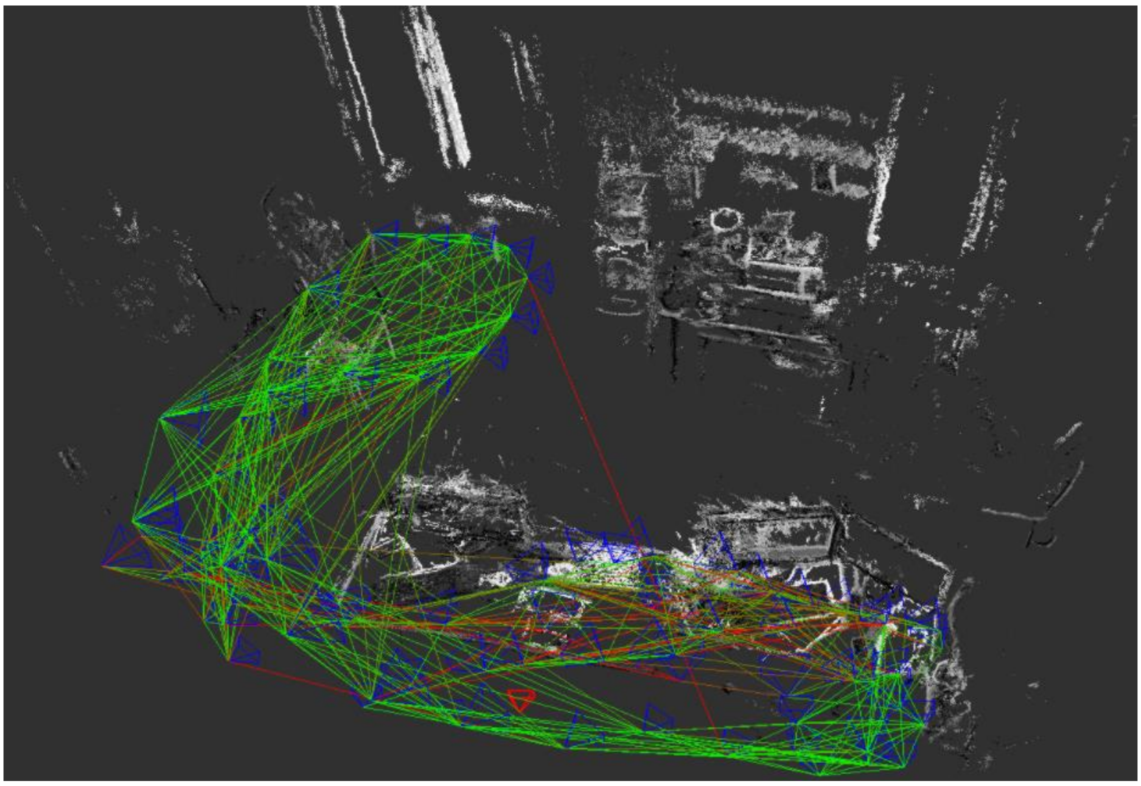


Figure 3. 28. LSD-SLAM tracking and mapping window.

Those frames obtained, although not converted in a keyframe are used to redefine the current keyframe. The result is incorporated to the depth map; which means that new pixels are added to the map.

3. Map optimization

The built map, consists in a bunch of keyframes joined by different restrictions, such map is continuously optimized at the background by the optimization graph.

3.2.3.3 Simulation implementation and launch file

As it was done in the previous chapters, before launching the simulator it is needed to do a remap in some published topics. The LSD SLAM looks for two published topics to start the algorithm. These topics are */image_raw* and */camera_info*, that's why it is necessary to carry out the following remap within the file that launches the robot simulator.

```
<launch>
<remap from="/erlecopter/front/image_front_raw" to="/erlecopter/image_raw"/>
<remap from="/erlecopter/front/camera_front_info"
to="/erlecopter/camera_info"/>
```

Converting the image from the simulated camera to grayscale:

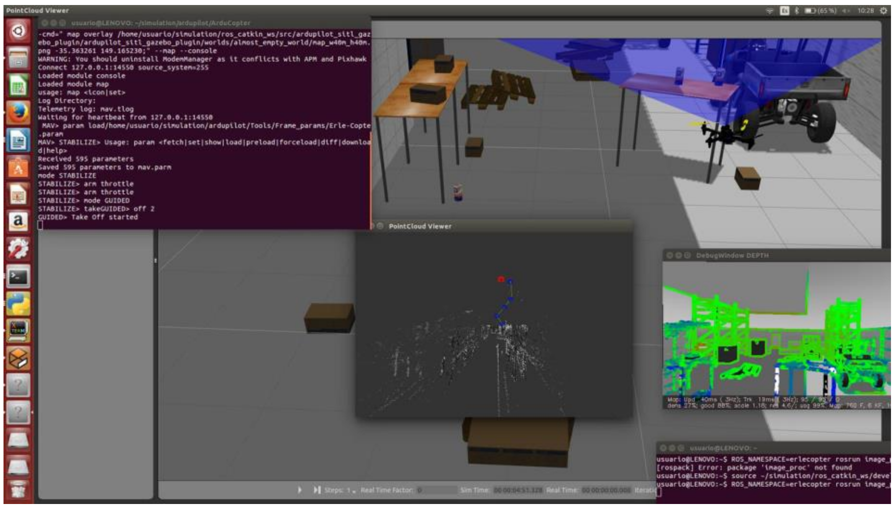
```
$ ROS_NAMESPACE=erlecopter rosrun image_proc image_proc
```

Finally the SLAM LSD can be launched:

```
$rosrun lsd_slam_core live_slam /image:=/erlecopter/image_mono
_calib:=/home/usuario/rosbuild_ws/package_dir/lsd_slam/lsd_slam_core/calib/Erle_S
im_calib.cfg
```

To visualize the LSD viewer the following command must be executed.

```
$ rosrun lsd_slam_viewer viewer
```



```
-cmd=" map overlay /home/usuario/simulation/ros_catkin_ws/src/ardupilot_sitl_gaz
ebo_plugin/ardupilot_sitl_gazebo_plugin/worlds/almost_empty_world/map_w40m_h40m.
png -35.363261 149.165230;" --map --console
WARNING: You should uninstall ModemManager as it conflicts with APM and Pixhawk
Connect 127.0.0.1:14550 source_system=255
Loaded module console
Loaded module map
usage: map <icon|set>
Log Directory:
Telemetry log: mav.tlog
Waiting for heartbeat from 127.0.0.1:14550
 MAV> param load/home/usuario/simulation/ardupilot/Tools/Frame_params/Erle-Copte
.param
MAV> STABILIZE> Usage: param <fetch|set|show|load|preload|forceload|diff|downloa
d|help>
Received 595 parameters
Saved 595 parameters to mav.parm
mode STABILIZE
STABILIZE> arm throttle
STABILIZE> arm throttle
STABILIZE> mode GUIDED
STABILIZE> takeGUIDED> off 2
GUIDED> Take Off started
```

```
usuario@LENOVO:~$ ROS_NAMESPACE=erlecopter rosrun image_
[rospack] Error: package 'image_proc' not found
usuario@LENOVO:~$ source ~/simulation/ros_catkin_ws/deve
usuario@LENOVO:~$ ROS_NAMESPACE=erlecopter rosrun image_
```

Figure 3. 29. LSD-SLAM launched in Gazebo simulator with the Erlecopter model.

3.2.3.4 Real implementation within Erlecopter and launch file

As it was done before the drone will be connected to an external device via WIFi, where the LSD algorithm will be running in real time. Once the camera is opened , the first step is to convert the raw image to mono.

```
$ ROS_NAMESPACE=camera rosrun image_proc image_proc
```

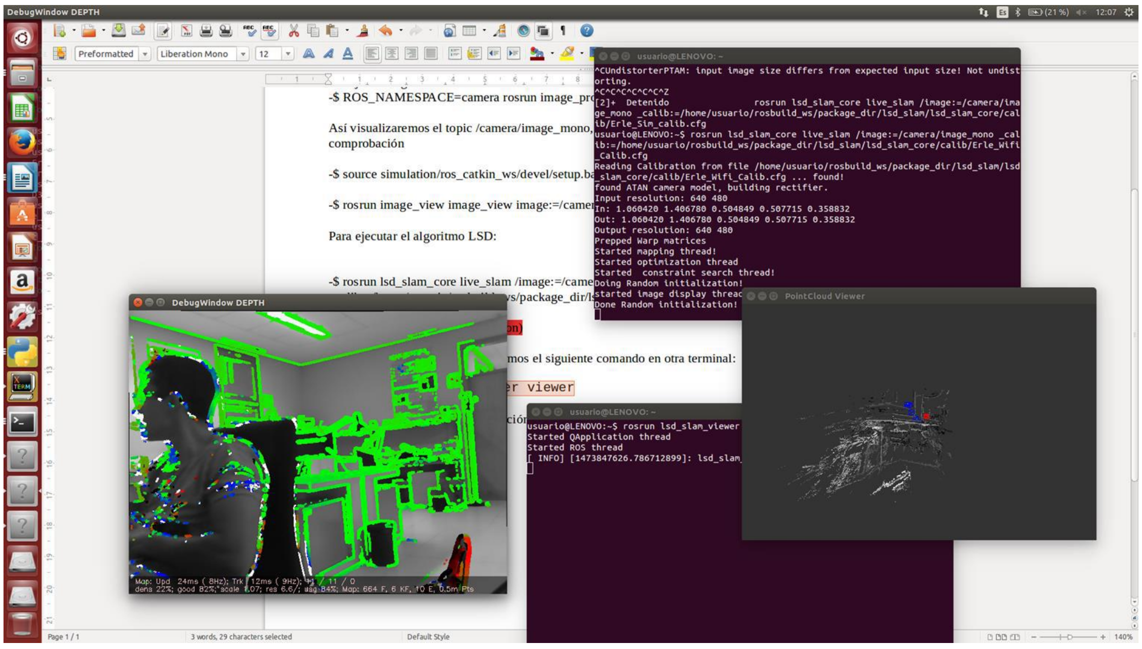
Now it can be visualized that the /camera/image_mono topic is being published in gray scale.

To execute the LSD algorithm:

```
$ rosrun lsd_slam_core live_slam /image:=/camera/image_mono
_calib:=/home/usuario/rosbuild_ws/package_dir/lsd_slam/lsd_slam_core/calib/Erle_
Wifi_Calib.cfg
```

To visualize the tracking and mapping in the LSD viewer:

```
$ rosrun lsd_slam_viewer viewer
```



-$ ROS_NAMESPACE=camera rosrun image_pro

Así visualizaremos el topic /camera/image_mono,
comprobación

-$ source simulation/ros_catkin_ws/devel/setup.ba

-$ rosrun image_view image_view image:=/camer

Para ejecutar el algoritmo LSD:

-$ rosrun lsd_slam_core live_slam /image:=/came

```
^CUndistorterPTAM: input image size differs from expected input size! Not undist
orting.
^C^C^C^C^C^C^Z
[2]+  Detenido                rosrun lsd_slam_core live_slam /image:=/camera/ima
ge_mono _calib:=/home/usuario/rosbuild_ws/package_dir/lsd_slam/lsd_slam_core/cal
ib/Erle_Sim_calib.cfg
usuario@LENOVO:~$ rosrun lsd_slam_core live_slam /image:=/camera/image_mono _cal
ib:=/home/usuario/rosbuild_ws/package_dir/lsd_slam/lsd_slam_core/calib/Erle_Wifi
_Calib.cfg
Reading Calibration from file /home/usuario/rosbuild_ws/package_dir/lsd_slam/lsd
_slam_core/calib/Erle_Wifi_Calib.cfg ... found!
found ATAN camera model, building rectifier.
Input resolution: 640 480
In: 1.060420 1.406780 0.504849 0.507715 0.358832
Out: 1.060420 1.406780 0.504849 0.507715 0.358832
Output resolution: 640 480
Prepped Warp matrices
Started mapping thread!
Started optimization thread
Started  constraint search thread!
Doing Random initialization!
started image display threac
Done Random initialization!
```

```
usuario@LENOVO:~$ rosrun lsd_slam_viewer
Started QApplication thread
Started ROS thread
[ INFO] [1473847626.786712899]: lsd_slam_
```

```
Map: Upd  24ms ( 8Hz); Trk  12ms ( 9Hz); 11 / 11 / 0
dens 22%; good 82%; scale 1.07; res 6.6/; usg 84%; Map: 664 F, 6 KF, 10 E, 0.5m Pts
```

Figure 3. 30. LSD-SLAM launched on the real robotic platform.

3.2.3.5 Results obtained

As the ORB- SLAM algorithm, the interface is composed by two windows. The first one is shown in figure 3.30, in which the intensity map can be seen. In the lowest part of the window, important information can be found, such as the actualization of the map, the tracking, the number of keyframes among other interesting data.

The second window shows the map built from a cloud of points that is being generated along the path followed by the camera. The keyframes selected by the algorithm are represented in blue, while the red color represents the current position of the camera, and finally, the green color represents the path generated.

Again, following the steps from the PTAM and ORB chapter, that is to say, taking the same paths developed above, the results after having implemented LSD-SLAM in the simulation are now presented.

Result Path 1:

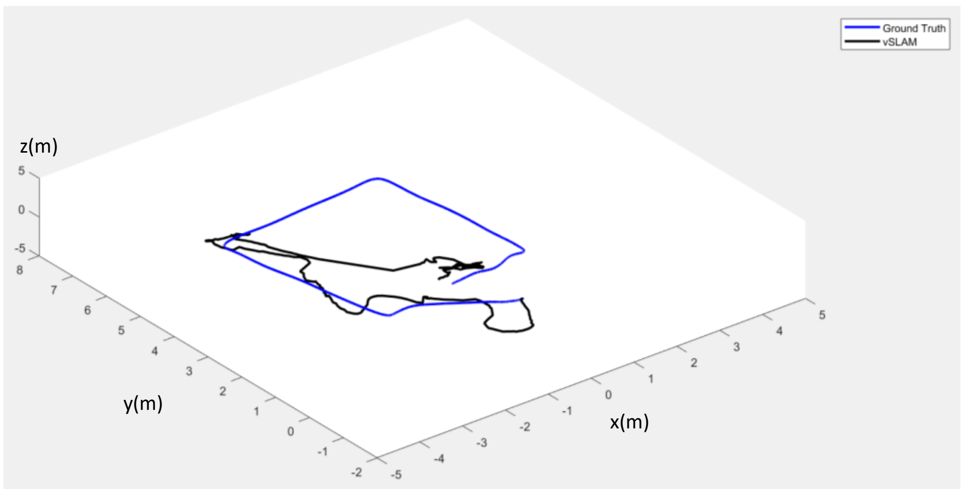


Figure 3. 31. Results LSD-SLAM path 1 3D view.

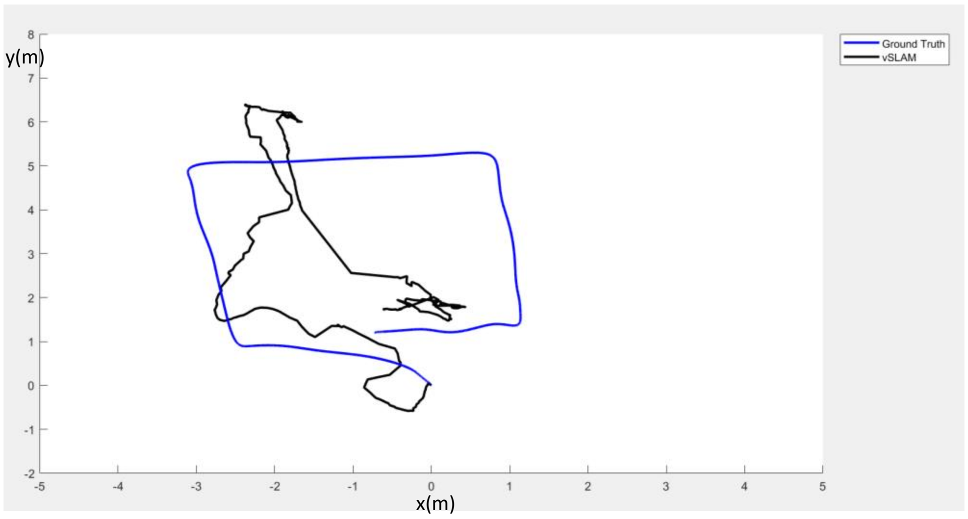


Figure 3. 32. Results LSD-SLAM path 1 X-Y view.

Again, the blue line represents the ground truth of the robot, so to say , the real position of the Erlecopter. The black line represents the localization given by the LSD-SLAM algorithm along the path.

Result Path 2:

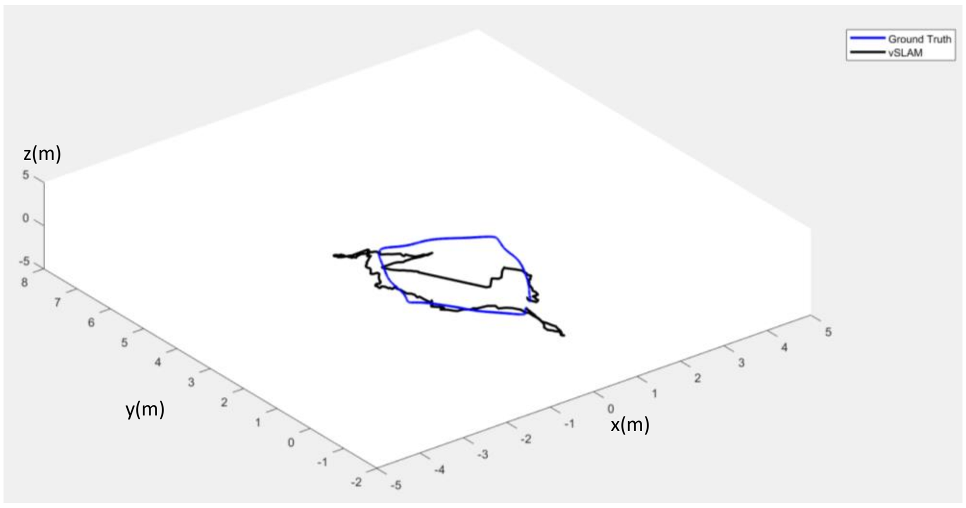


Figure 3. 33. Results LSD-SLAM path 2 3D view.

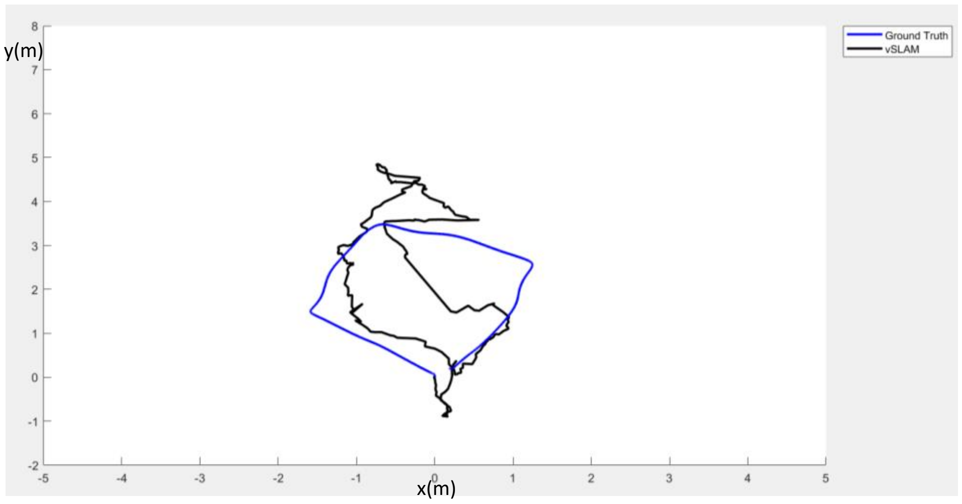


Figure 3. 34. Results LSD-SLAM path 2 X-Y view.

Apparently, the localization given by the LSD-SLAM is far from the real localization in certain sections along the path. This may be due to several losses of tracking during the simulation process.

Result Path 3:

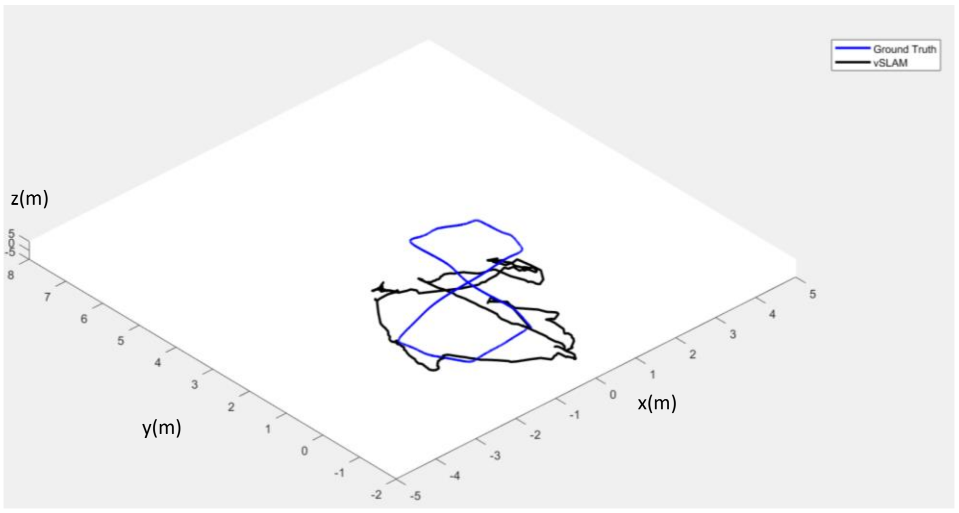


Figure 3. 35. Results LSD-SLAM path 3 3D view.

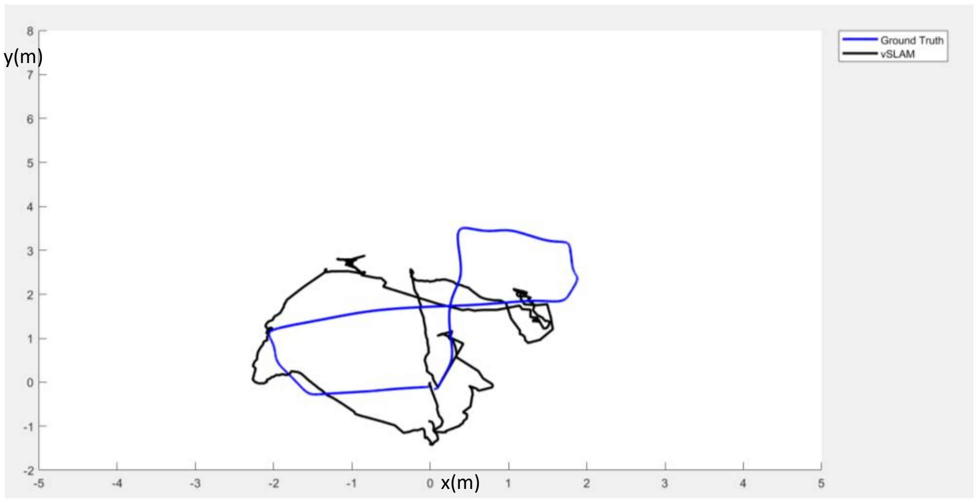


Figure 3. 36. Results LSD-SLAM Path 3 X-Y view.

After testing the algorithm, it was noticed that the execution time is high, given the fact that it is necessary to move the camera extremely slow, or at least it is needed a really powerful computer processor. As the movement of the quadcopter is the same, no matter the employed algorithm, the results, apparently, are worse than the results obtained for the previous algorithms. Plus, the algorithm hardly recovers from a loss of the tracking.

3.3. Results

3.3.1 Simulation Results

In this section, the results obtained from the different tracks developed will be discussed. After having talked about the performance of the three visual algorithms, now a calculation of the MAD (Mean Absolute Deviation) and MSE (Mean Square Error) error will be carried out.

For a better understanding of what these errors means, a briefly explanation of each one is presented:

- MAD: it takes the absolute value of forecast errors and averages them over the entirety of the forecast time periods. Taking an absolute value of a number disregards whether the number is negative or positive and, in this case, avoids the positives and negatives cancelling each other out.
  The next formula represents the calculation of the MAD error:

$$\text{MAD}=\frac{\sum_{i=1}^{n}|Ai-Fi|}{n}$$

Where A represents, in our particular case, the pose X,Y of given by the Ground Truth, so to say, the actual position of the quadcopter, while F represents the estimated position of the quadcopter, and n, the number of poses studied.

- MSE: it measures the average of the squares of the errors or deviations—that is, the difference between the estimator and what is estimated. The MSE is a measure of the quality of an estimator—it is always non-negative, and values closer to zero are better.
  The following formula represents the MSE:

$$\text{MSE}=\frac{\sum_{i=1}^{n}(Ai-Fi)^2}{n}$$

The terms presented in such formula are the same than the ones explained above for the calculation of the MAD error.

Now let's see the errors for the 3 tracks and the three vSLAM algorithms, to get a better understanding of its performance.

VSLAM	TRACK	MAD	MSE
PTAM	Track 1	1.3712	2.7316
	Track 2	1.5177	2.8811
	Track 3	0.6692	0.6650
Average Score		**1.1870**	**2.0922**
ORB-SLAM	Track 1	1.4062	2.3243
	Track 2	0.6791	0.6887
	Track 3	0.7093	0.7313
Average Score		**0.9315**	**1.2481**
LSD-SLAM	Track 1	1.7799	4.5085
	Track 2	0.9317	1.2351
	Track 3	1.0132	1.2481
Average Score		**1.2416**	**2.3266**

Table 3. 1. Errors obtained for each vSLAM algorithm.

As pointed in [1], the best results are obtained for the ORB-SLAM algorithm, while LSD-SLAM and PTAM obtain similar errors.

So before it was shown the qualitative performance of these three algorithms, now a comparison of the errors of such algorithms is shown in the table 3.1. As a conclusion, the ORB gets better performance both for the MSE and the MAD, so if it had to choose one of those to be implemented in an actual quadcopter this would be the most fittable.

The problem face at this point, is that on one hand, the ORB-SLAM gets a better performance, although on the other hand it is computationally heavier than PTAM. LSD-SLAM is also very heavy, by far, heavier than PTAM, and even heavier than ORB-SLAM.

As the Erlecopter is the robotic platform studied within this project, it could be said that both PTAM and ORB-SLAM are suitable, the problem is that the brain of the robotic platform is composed by a simple raspberry pi 2, so ORB-SLAM is too heavy to be launched on board (in real time), also noticing that it may have more sensors on the robot, that could affect the performance of the robotic platform.

3.3.2 Real results

In this project, a real implementation of the vSLAM algorithms was made using the real Erlecopter camera, running off-board in a Lenovo U31, since the on-board CPU (Raspberry-Pi 2) was not able to handle ORB-SLAM or LSD-SLAM due to their CPU consume. It is important to highlight that one of the main targets of this project, was to run the algorithms on board, but the CPU consume, as shown in table X was too high to be supported by the Raspberry Pi 2, so the algorithms had to run off-board, using a Wi-Fi connection between the quadcopter and the Lenevo U-31 laptop. Thus in this subsection, the results of such test will be presented. For that purpose a test was recorded within the Politécnica building in the University of Alcalá.

%CPU Consume	PTAM	ORB-SLAM	LSD-SLAM
	88%	133%	148%

The scale was computed using the downward sonar, using the technique that will be explained in the chapter 4. Thus, figure 3.37 shows the performance of each of these three algorithms.

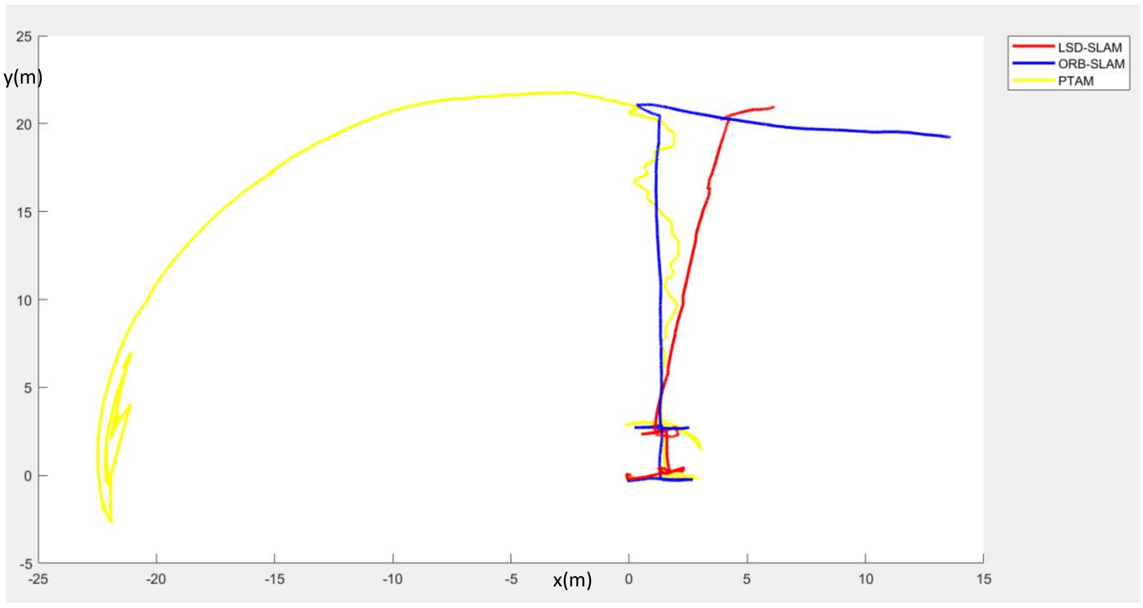


Figure 3. 37. Real tracking of the 3 vSLAm algorithms on the Erlecopter. X-Y view.

As extracted from the simulation test, it can be seen again that ORB-SLAM turns to be the most accurate algorithm. PTAM is able to follow the tracking, although at the end of the corridor a rotational movement around the Z axis, of 45 degrees is developed, and it loses completely the tracking. It can be seen that the developed tracking before losing the path is very unstable compare it with LSD-SLAM and ORB-SLAM, so it can be said that is the most sensitive vSLAm algorithm of all.

Finally LSD-SLAM tends to easily lose the tracking and increase the error. Besides after a rotational movement it cuts the length of the real path.

Since the tests were made in a corridor of a length of 22m, so it can be extracted from figure 3.37 that the estimation of all the algorithms used is slightly shorter than the actual length of the path.

CHAPTER 4

EXTENDED KALMAN FILTER

4.1. Introduction

In 1960 Rudolf E. Kalman developed an algorithm called Kalman Filter (KF), which allows the identification of a hidden state, that can't be measured, within a dynamic linear system, even if the system presents some kind of noise. So the Extended Kalman Filter [16] (EKF) is a nonlinear version of the KF, which linearizes about the estimate of the current mean and covariance, and it is heavily entrenched in nonlinear signal processing applications. This approach is used in robotics systems, given the fact that it presents a solution to the pose estimation problem in SLAM techniques.

4.2. Extended Kalman Filter algorithm

Therefore, the EKF is an algorithm capable of estimating the position of a robotic platform using sensor fusion in nonlinear applications, in this particular case, the mathematical model obtained from the Erlecopter kinematics and dynamics, will be obviously nonlinear.

As showed previously, the EKF operates on the same principle as the regular Kalman Filter, using a linearized model of the system to predict the quadcopter's state though.

The EKF algorithm is represented by the following equations:

$$\widehat{X}_{t-1} = g(u_t,u_{t-1}) \; (1)$$

$$\widehat{P}_t = G_t \cdot P_{t-1} \cdot G^T{}_t + Q_t \, (2)$$

$$K_t = \widehat{P}_t \cdot H^T{}_t \cdot (H_t \cdot \widehat{P}_t \cdot H^T{}_t + R_t)^{-1} \; (3)$$

$$X_{t\,=} \; \widehat{X}_t \; + K_t \cdot (z_t - h(\widehat{X}_t)) \; (4)$$

$$P_{t\,=} \; (I\text{- } K_t \cdot H_t) \cdot \widehat{P}_t \, (5)$$

The above equations represent the EKF algorithm. The equations (1) and (2) are used to predict the state $\widehat{X}_t$ and the covariance matrix $\widehat{P}_t$. This step is called prediction model. On the other hand the equations (3), (4) and (5) represent the observation model, which computes the Kalman gain K_t the current state X_t and the covariance matrix P_t. Besides, the covariance matrix of the model is represented by Q_t , the covariance matrix of the measurements is represented by R_t and G_t and H_t are the Jacobian matrixes of the model and measurement respectively.

The following scheme shows the computational flow of the EKF, how the system works both in the prediction and observation model:

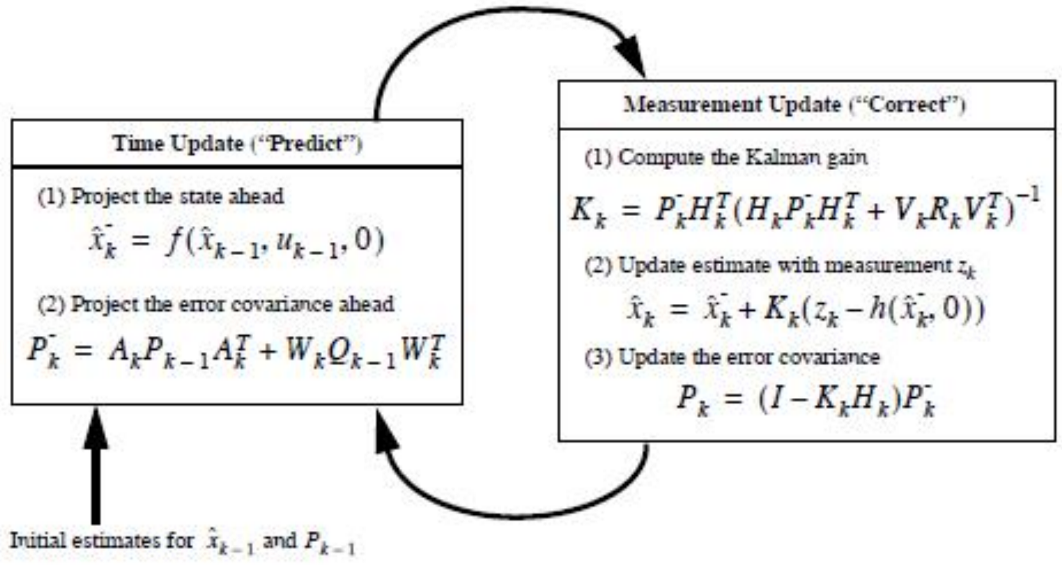

Time Update ("Predict")
(1) Project the state ahead
$\hat{x}_k^- = f(\hat{x}_{k-1}, u_{k-1}, 0)$
(2) Project the error covariance ahead
$P_k^- = A_k P_{k-1} A_k^T + W_k Q_{k-1} W_k^T$
Measurement Update ("Correct")
(1) Compute the Kalman gain
$K_k = P_k^- H_k^T (H_k P_k^- H_k^T + V_k R_k V_k^T)^{-1}$
(2) Update estimate with measurement z_k
$\hat{x}_k = \hat{x}_k^- + K_k(z_k - h(\hat{x}_k^-, 0))$
(3) Update the error covariance
$P_k = (I - K_k H_k) P_k^-$
Initial estimates for $\hat{x}_{k-1}$ and P_{k-1}

Figure 4. 1. Block Diagram of the EKF. Prediction and correction workflow.

4.2.1 Prediction Model

First, from the kinematics and dynamics of the quadcopter, we need to solve the mathematical model of the Erlecopter in order to get the prediction model. Based on the model for the AR Drone, developed by Engel, Sturm and Cremers in [23], we got the mathematical model of the Erlecopter:

State vector: $x(t) = \{x, y, z, Vx, Vy, Vz, \varphi, \sigma, \psi, \dot{\psi}\}$

where x represents the position of the quadcopter in the x-axis, y represents the position of the robot in the y-axis, and finally z represents the position of the robot in the z-axis. In addition, Vx represents the linear velocity of the robot on the x axis, same happens with Vy for the y axis and Vz for the z axis.Besides, φ is the angle of the robot around the y-axis, σ is the angle of the robot around the x-axis and ψ is the angle around the z-axis. Finally $\dot{\psi}$ represents the angular velocity around the z-axis.

Control Inputs: $u\ (t) = \{\hat{V}x, \hat{V}y, \hat{V}z,\ \hat{\dot{\psi}}\}$

where $\hat{V}x$ represents the control of the linear velocity on the x-axis, $\hat{V}y$ represents the control of the linear velocity on the y-axis and $\hat{V}z$ is the control of linear velocity on the z-axis. Finally, $\hat{\dot{\psi}}$ represents the control of the angular velocity around the z-axis.

From now on, the prediction model will be known as a function such as $\dot{x} = h(x,u)$, where φ,σ and ψ, must be Euler angles XYZ.

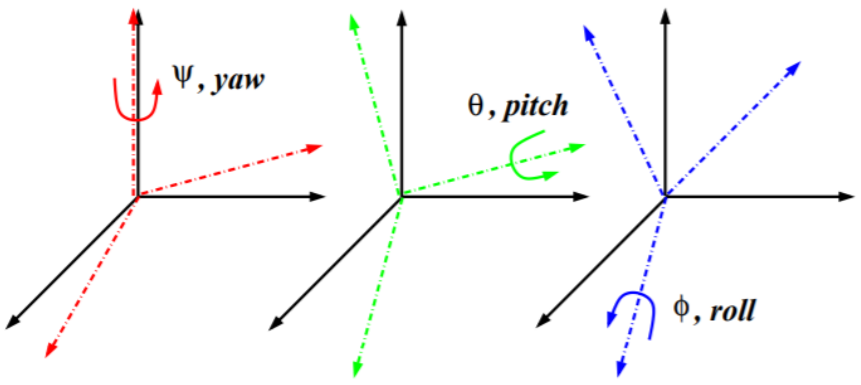


Figure 4. 2. Yaw, Pitch and Roll angles in the world frame.

The horizontal acceleration is proportional to the projection of the Z axis. Therefore R=rot$_z$(ψ)*rot$_y$(σ)*rot$_x$(φ):

$$R = \begin{pmatrix} \cos\sigma\cos\psi & \sin\varphi\sin\sigma\cos\psi - \cos\varphi\sin\psi & \cos\varphi\sin\sigma\cos\psi + \sin\varphi\sin\psi \\ \cos\sigma\sin\psi & \sin\varphi\sin\sigma\sin\psi + \cos\varphi\cos\psi & \cos\varphi\sin\sigma\sin\psi - \sin\varphi\cos\psi \\ -\sin\sigma & \sin\varphi\cos\sigma & \cos\varphi\cos\sigma \end{pmatrix}$$

Taking the third column, it is easy to get the following equations:

$$\ddot{x}= C1 \cdot (C2 \cdot (\sin(\varphi) \cdot\sin(\psi)+\cos(\varphi) \cdot \sin(\sigma)\cdot\cos(\psi)) - Vx)$$

$$\ddot{y}= C1\cdot (C2\cdot (-\sin(\varphi) \cdot\sin(\psi)+\cos(\varphi) \cdot \sin(\sigma)\cdot\cos(\psi)) - Vy)$$

$$\dot{\varphi}=-C3\ (C4\cdot\hat{V}y + \varphi)$$

$$\dot{\sigma}=C3\ (C4\cdot\hat{V}x - \sigma)$$

$$\ddot{\psi}=C5\ (C6\cdot \hat{\dot{\psi}} - \dot{\psi})$$

$$\dot{V}z=C7\ (C8\cdot\hat{V}z - Vz)$$

Therefore the prediction model, using the kinematics equations obtained above is defined as:

$$\begin{pmatrix} x \\ y \\ z \\ Vx \\ Vy \\ Vz \\ \varphi \\ \sigma \\ \psi \\ \dot{\psi} \end{pmatrix}_{t+\Delta t} = \begin{pmatrix} x \\ y \\ z \\ Vx \\ Vy \\ Vz \\ \varphi \\ \sigma \\ \psi \\ \dot{\psi} \end{pmatrix}_{t} + \Delta t \cdot \begin{pmatrix} Vx \\ Vy \\ Vz \\ C1 \cdot (C2 \cdot (\sin(\varphi) \cdot \sin(\psi) + \cos(\varphi) \cdot \sin(\sigma) \cdot \cos(\psi)) - Vx) \\ C1 \cdot (C2 \cdot (-\sin(\varphi) \cdot \sin(\psi) + \cos(\varphi) \cdot \sin(\sigma) \cdot \cos(\psi)) - Vy) \\ C7\,(C8 \cdot \hat{V}z - Vz) \\ -C3\,(C4 \cdot \hat{V}y + \varphi) \\ C3\,(C4 \cdot \hat{V}x - \sigma) \\ \dot{\psi} \\ C5\,(C6 \cdot \hat{\dot{\psi}} - \dot{\psi}) \end{pmatrix}$$

Where, Δt is the change in time between the previous model update to the new model update.

4.2.2 Observation Model

Given the fact that the employed algorithms, either PTAM or ORB-SLAM or LSD-SLAM , can measure directly the 6 DOF of the quadcopter. As shown the obtained model for these measurements is linear:

$$Z_{vslam} = h_{vslam}(x) = \begin{pmatrix} x \\ y \\ z \\ \varphi \\ \sigma \\ \psi \end{pmatrix}$$

Thus, the Jacobian for the measurement system Ht is composed by the camera measurements. The Ht can be presented as:

$$H_t = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}$$

All in all, the EKF has been explained; now, let's solve the problem of the scaling factor in the vSLAM algorithms, due to the lack of knowledge of the real distance of the objects in the pinhole cameras.

4.2.2.1 Scaling Factor Problem

The main problem to deal with, in order to get the data information from the camera algorithms is the ambiguity of the scale issue. Monocular configurations are unable to identify the length of the translational movement only from the features correspondences. As shown, the camera itself cannot know the real depth of the object in the image, so it is impossible for the vSLAM algorithm to compute the tracked movement in a real scale.

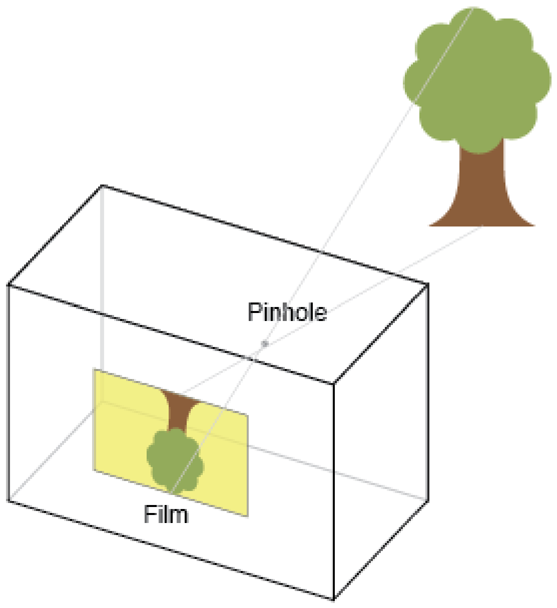


Figure 4. 3. Relationship between pixel scale and real scale in monocular cameras.

To solve this problem, monocular camera systems need some kind of movement from the camera to obtain a couple of frames and compare the features between them in order to extract the key points. As the actual distance of the camera's movement is unknown, the system won't calculate accurately the real scale.

Given the performance of the vSLAM algorithms, in this particular situation, an approach will be done in order to solve the issue. As [5] did in his project, taking the information from the ultrasonic downward sensor, located within the robotic system, the measurement of the altitude in the Z axis direction of the world frame, can be used to solve the problem of the unknown translational movement, so now, the relationship between the frames and the real length is calculated.

Thus the idea is that if the real height is known, and also the one estimated by the vSLAM, then the scale can be calculated as follows:

$$\text{Scale}=\frac{\text{hsonar}}{\text{hvslam}}\ (1)$$

$$Z_{\text{real}}=\text{Scale}\cdot Z_{\text{vslam}}\ (2)$$

$$X_{\text{real}}=\text{Scale}\cdot Z_{\text{vslam}}\ (3)$$

$$Y_{\text{real}}=\text{Scale}\cdot Z_{\text{vslam}}\ (4)$$

After performing several trials, shown in figures from 4.4 to 4.6 , in order to get the real scale of the monocular algorithm, it jumps to the conclusion that the scale is very changing, this means, that each time that the algorithm is launched, the obtained scale is not even close to the one obtained previously.

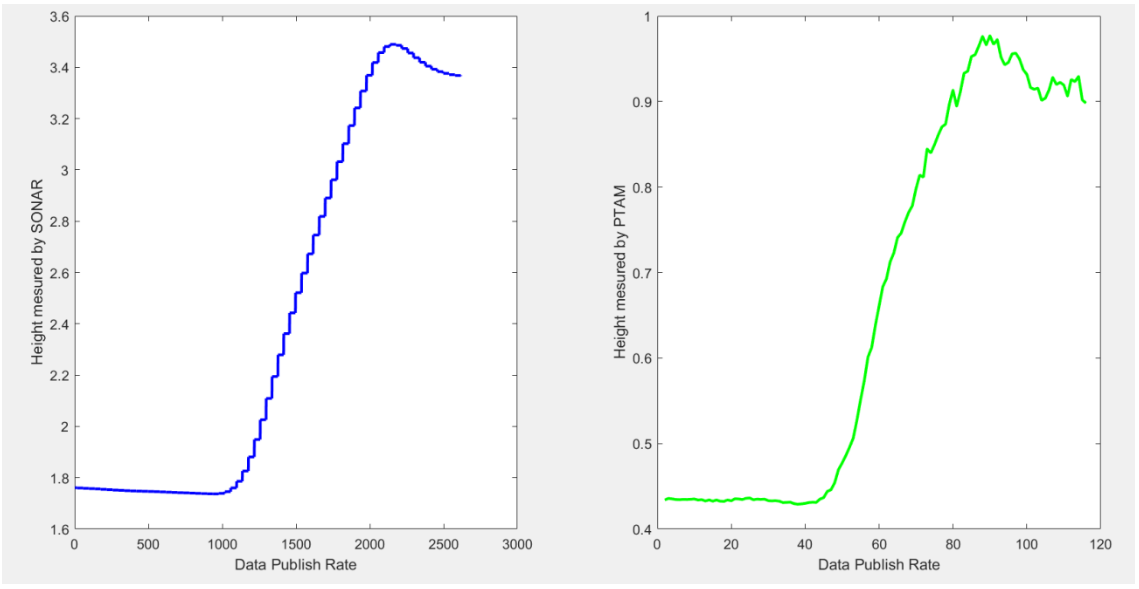


Figure 4. 4. First trial . Relationship between the sonar height and the PTAM predicted height.

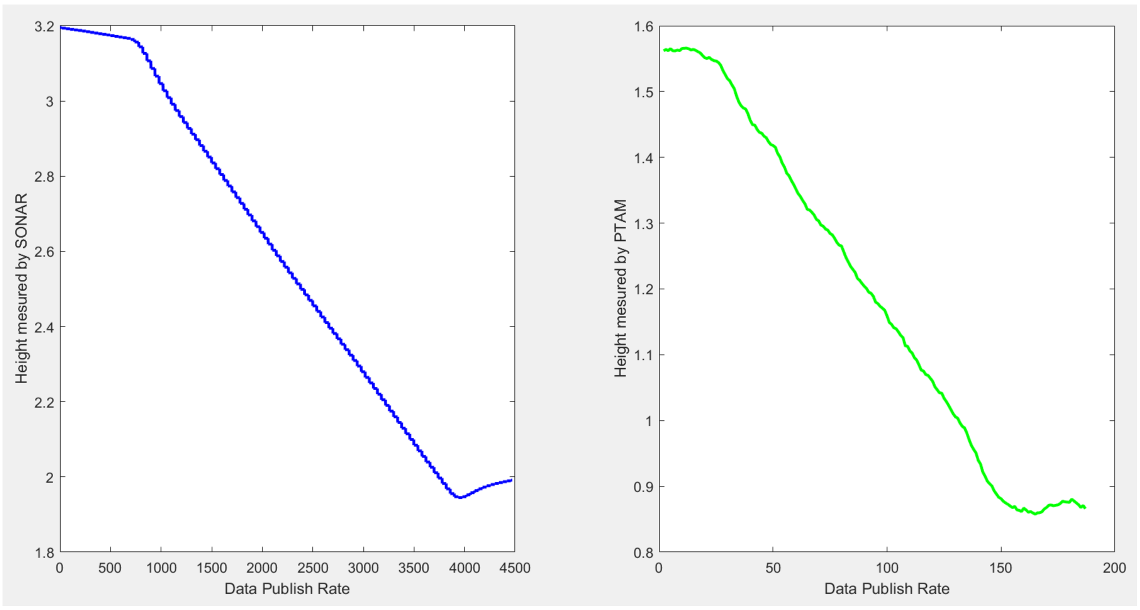


Figure 4. 5. Second trial . Relationship between the sonar height and the PTAM predicted height.

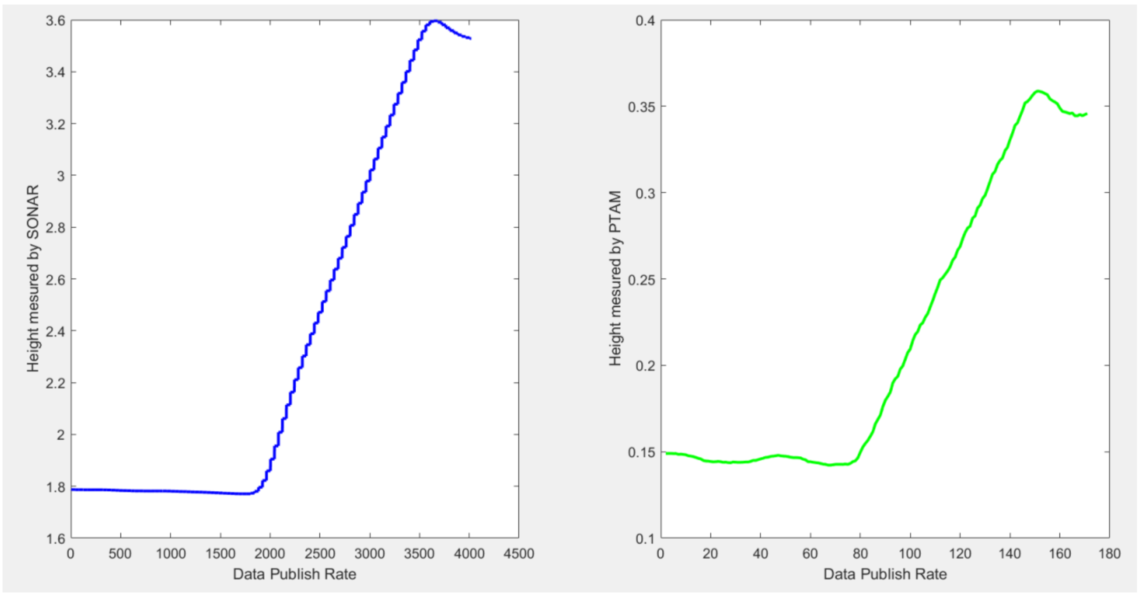


Figure 4. 6. Third trial . Relationship between the sonar height and the PTAM predicted height.

Then, to get a high quality performance of the vSLAM algorithms, each time that the system runs a monocular algorithm, first a pure translational movement in the Z axis direction has to be carried out.

In other projects such as in [5], the scale was calculated with every iteration of the system, which worked at 25HZ. For this project, it will be accurate enough to calculate the real scale at the beginning of the actuation of the developed system.

4.3. Results

At this point the results obtained from the development of the EKF explained above and its performance will be presented, and it will take conclusions from the results.

Not only the performance will be studied, but the MSE and MAD errors, as it was done in the previous chapters for the vSLAM algorithms, therefore it will be possible to notice the improvement of the pose estimation of the system.

- **Results Path 1:**

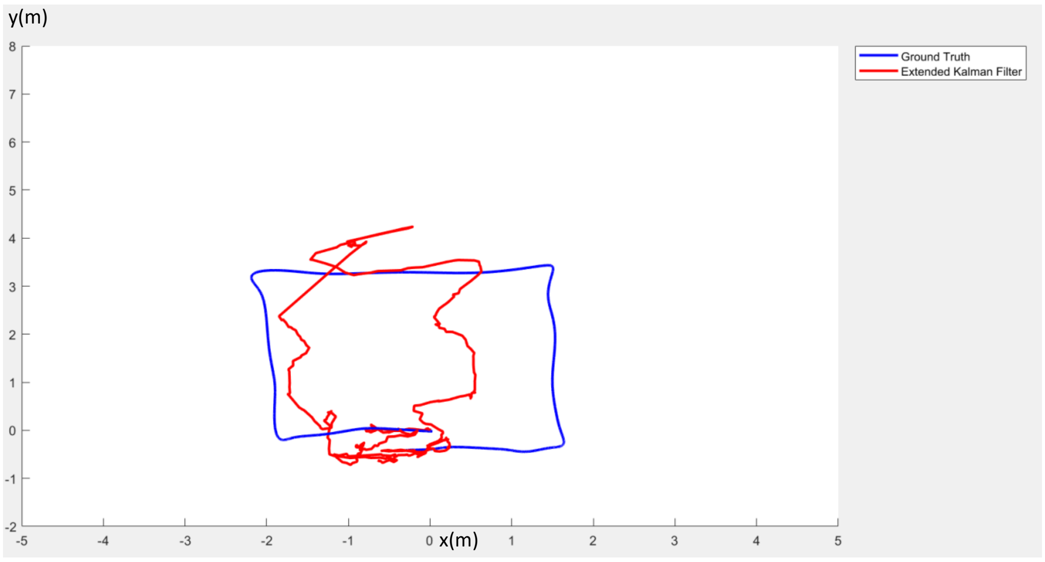


Figure 4. 7. Results EKF path 1 X-Y view.

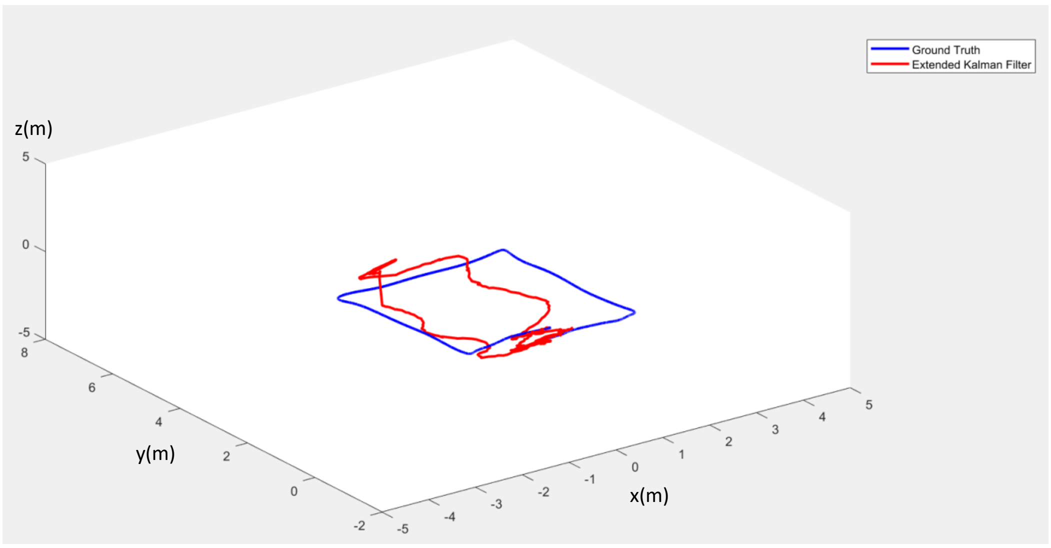


Figure 4. 8. Results EKF path 1 3D view.

- **Results Path 2**:

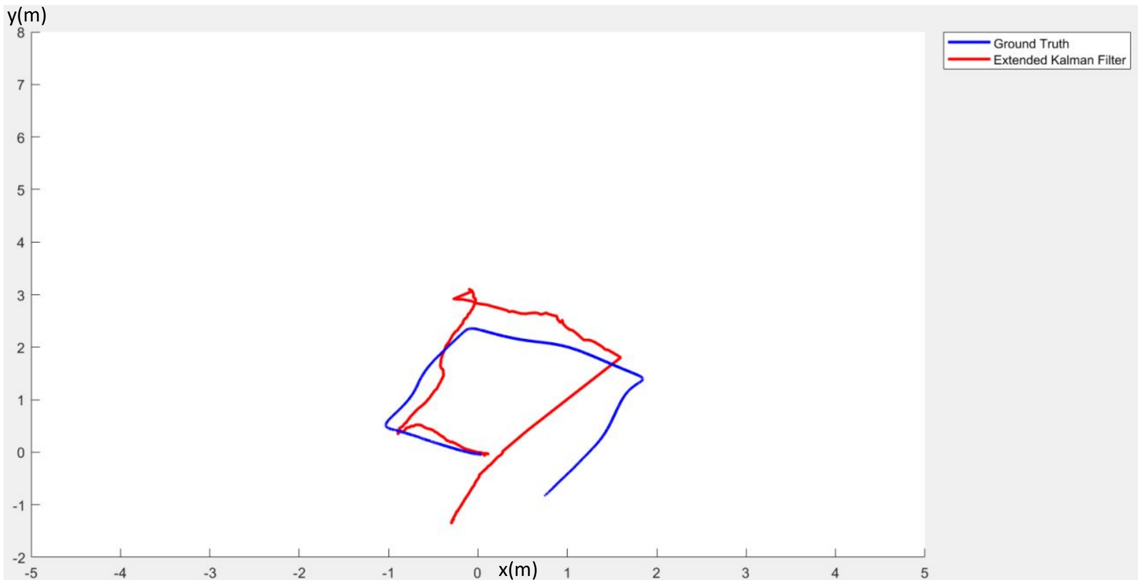


Figure 4. 9. Results EKF path 2 X-Y view.

It is important to remark that this EKF is using PTAM as the observation model; as consequence it must be compared with the results presented in the chapter number 3 for such monocular algorithm.

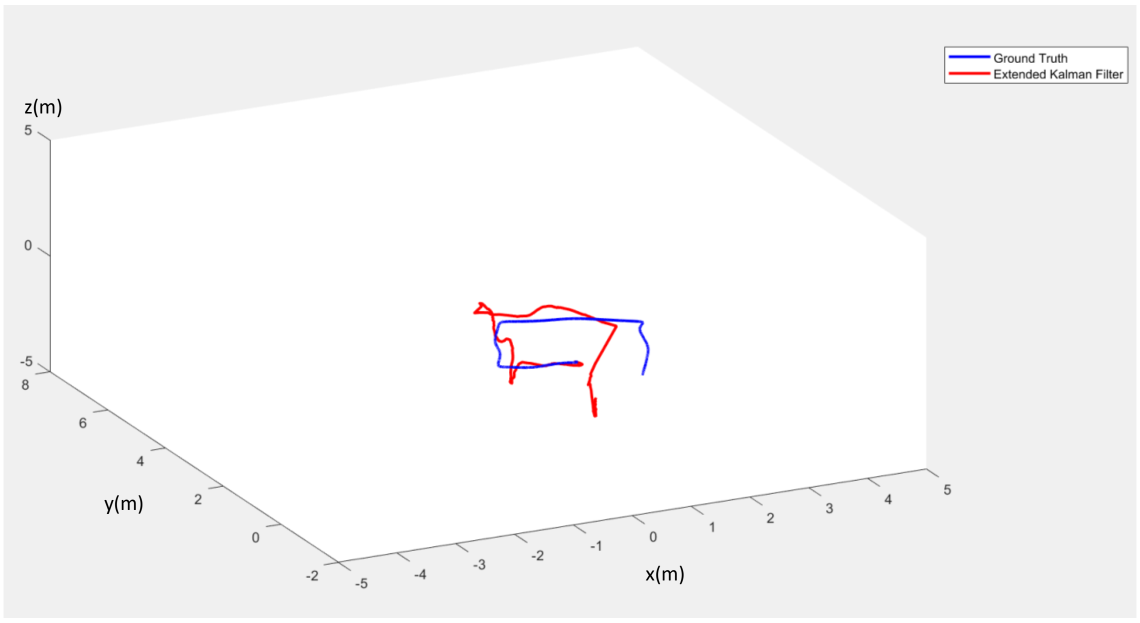


Figure 4. 10. Results EKF path 2 3D view.

- **Results Path 3:**

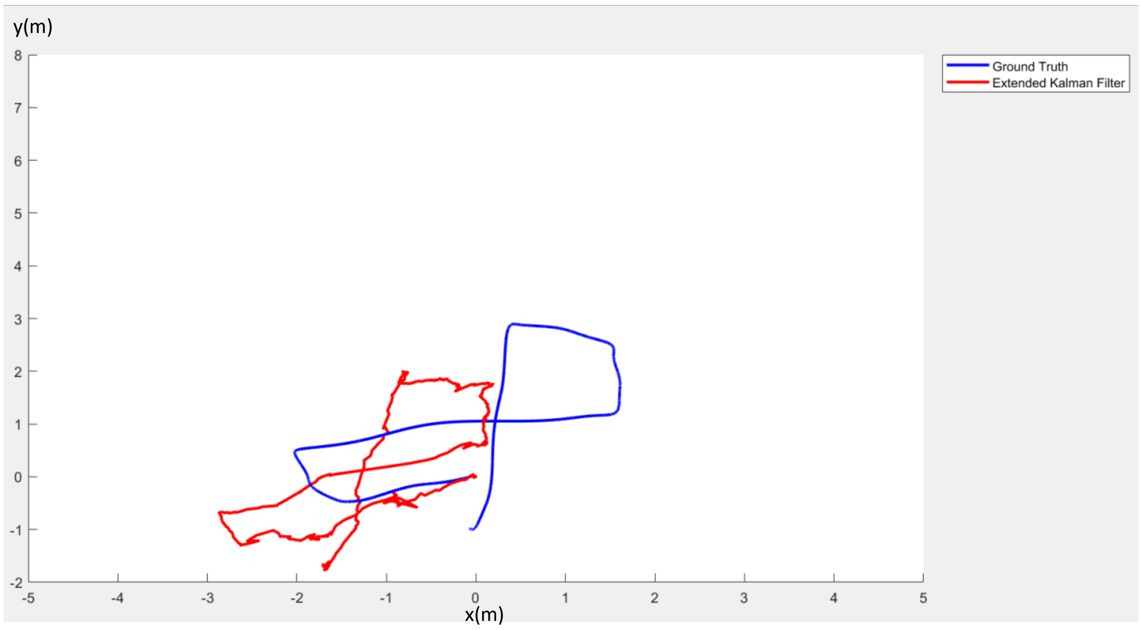


Figure 4. 11. Results EKF path 3 X-Y view.

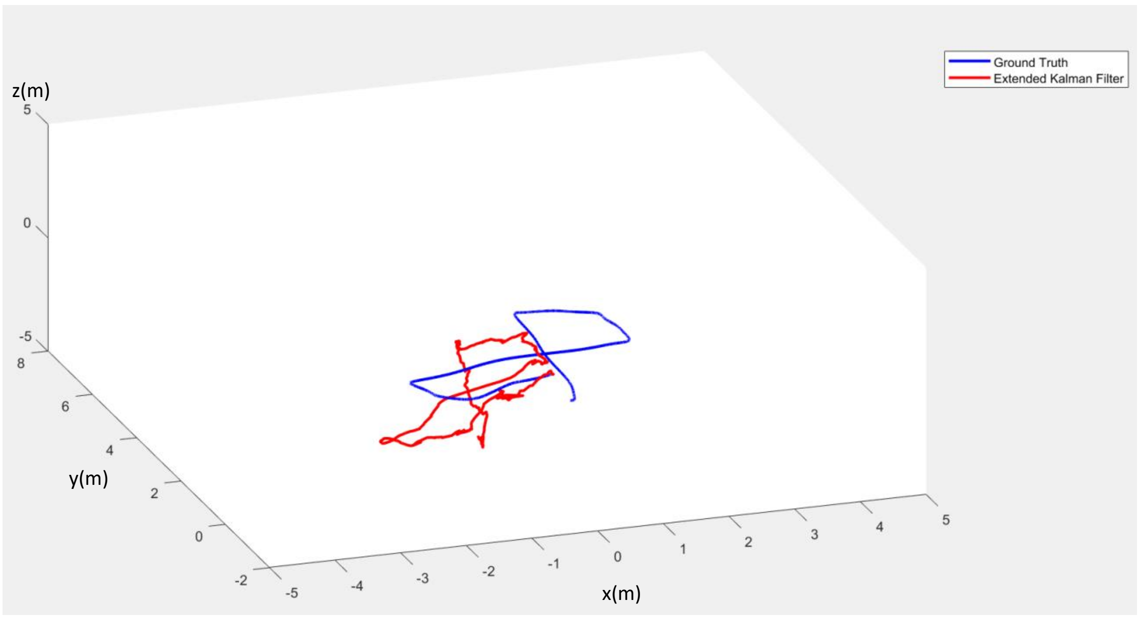


Figure 4. 12. Results EKF path 3 3D view.

These three paths show the performance developed by the implemented EKF algorithm. It can be seen that EKF estimates a position close to the real one, obtained from the ground truth system.

For this purpose, the table 4.1 stores the MAD and MSE errors calculated for the Extended Kalman Filter.

	TRACK	MAD	MSE
EKF	Track 1	0.8605	0.9194
	Track 2	0.6297	0.7185
	Track 3	0.8480	1.0358
Average Score		**0.7794**	**0.8912**

Table 4. 1. Errors obtained for the EKF.

Comparing this errors with the table 3.1, showed in previous chapters, it can be seen that the improvement of the results is highly remarkable. The table 4.2 compares the obtained errors after having run the 3 tracks:

Algorithm	MAD	MSE
EKF	**0.7794**	**0.8912**
PTAM	**1.1870**	**1.2922**

Table 4. 2. Average Errors from EKF and PTAM

Therefore, it can be said that the EKF presented works properly, reducing the error of the monocular algorithms, and presenting a more accurate pose estimation for the quadcopter, as also shown in figures 4.2 and 4.3.

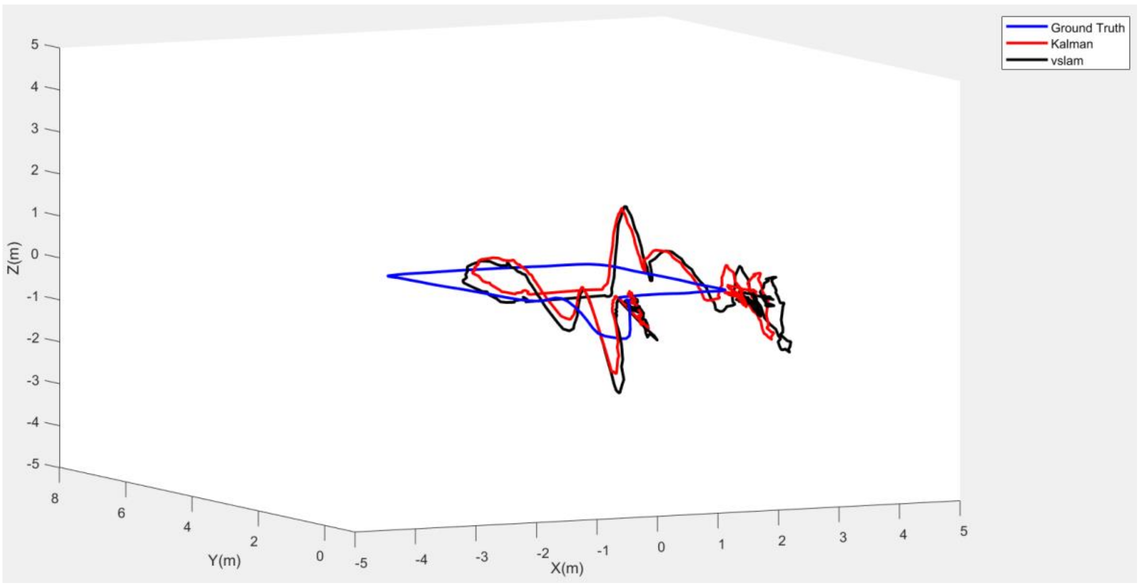


Figure 4. 13. 3D single-scope view Ground-Truth, EKF and PTAM

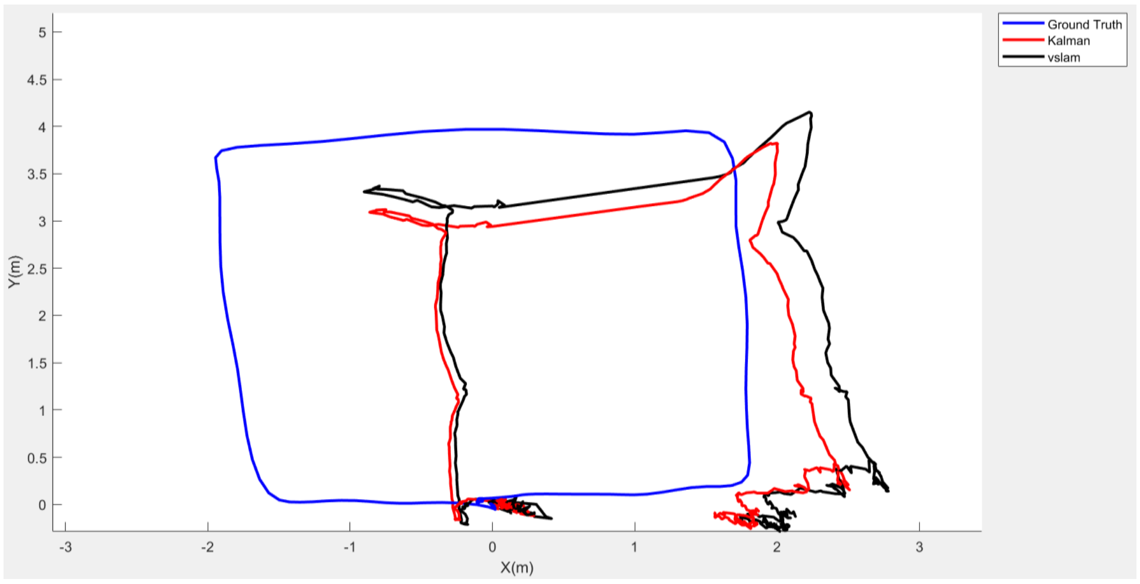


Figure 4. 14. X-Y single scope view Ground-Truth, EKF and PTAM.

Algorithm	MAD	MSE
EKF	1.1251	1.4412
PTAM	1.2357	1.2357

Table 4. 3. Errors obtained from EKF and PTAM in the single scope trial.

CHAPTER 5

PID CONTROLLER

In this section a PID controller for the robot will be designed in order to move and control the robot, making possible to develop a trajectory for SLAM algorithm purposes, to localize and map the robot and the environment.

5.1. Introduction

The idea of any control system is to calculate a discrepancy (error) between the objective (the desired position introduced by the user) and the reality (the real position of the robot). Such error will be used as a feedback to modify the control variable (radio control velocity).

A PID controller is a very well-known controller, used because its simplicity, it doesn´t require characterization of the system. It is based in 3 different terms (proportional, integral and differential) relating the error with some constants that can be determined through a trial and error process.

5.2. Implementation and results

In this particular case, *P_{ref}* will be the desired position, and *pos* will be the real position of the drone, or at least the estimated position. The error will be defined as:

$$e(t) = (P_{ref} - pos)$$

Using this error, the actuation variables will be modified based on the formula shown below:

$$pid = Kp \cdot e(t) + Ki \int e(t)dt + Kd\frac{de(t)}{dt}$$

For the moment, the PID can be considered as a simple value, that in case of being positive indicates that the robot must move forward and in case of being negative backwards. As the radio control variable is the one that has to be controlled, first we are going to explain how it works.

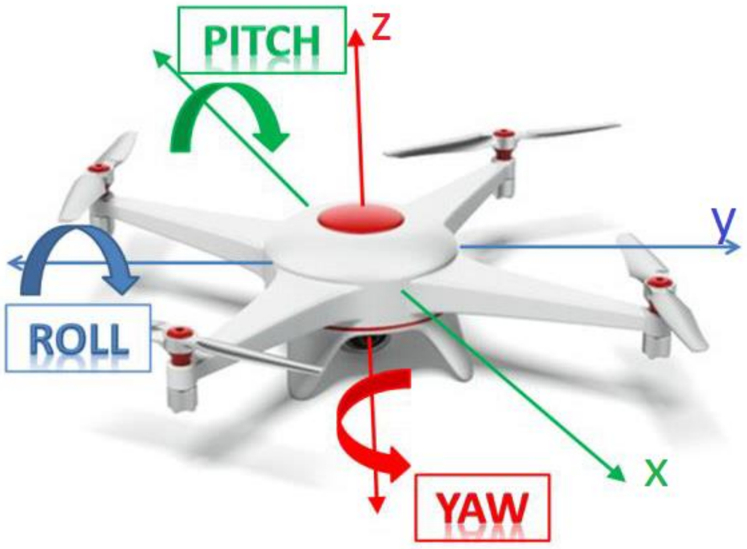


Figure 5. 1. Scheme of the local frame of the quadcopter

Figure 5.1 shows the different axis of the robot system, x, y, z, roll, pitch, yaw. Now to understand how the radio control works to move the drone we are going to present a set of examples. The radio control stands on an array of integer numbers, in the range from 1100 to 1900, where 1100 means "move the drone as quick as possible to the negative position from the local drone frame reference" and 1900 means "move the drone as quick as possible to the positive position from the local drone axis" and 1500 will mean "don't move, keep your position" (notice that the drone will be working in ALT HOLD mode, in each of the different modes the integer used will have a different performance). We are going to work in this definition deeper, in order to clarify the performance of the robot. So it is time to present the radio control array:

$$[V_x, V_y, V_z, V_{yaw}, 1500, 1500, 1500, 1500, 1500]$$

where the last four spots of the array of control are useless and don't have any relevant meaning.

So let's move the drone by writing and publishing a topic:

- Take-off slowly:
  $ rostopic pub -1 /mavros/rc/override mavros_msgs/OverrideRCIn '[1500, 1500, ***1600***, 1500, 1500,1500,1500,1500]'
- Move forward slowly:
  $ rostopic pub -1 /mavros/rc/override mavros_msgs/OverrideRCIn '[1500, ***1400***, 1500, 1500, 1500,1500,1500,1500]'
- Move to the right slowly:
  $ rostopic pub -1 /mavros/rc/override mavros_msgs/OverrideRCIn '[***1600***, 1500, 1500, 1500, 1500,1500,1500,1500]'

The values within the array are the PWM values of the radio controller. So after seeing how this works we can clearly see then, that the PID value from the formula will move between [1100, 1900], controlling the position of the drone trough the PWM actuators.

Now it is important to understand and explain each of the terms of this PID expression. The first term is proportional to the error, the second one to the integral and the last one goes with the derivative term.

The idea is that the first term aims to reduce the current error. For example for the Z axis, if the error is positive (the drone hasn't arrived to the reference position) pid will be positive, therefore the PWM value will increase to get to the reference position. On the other hand if the error is negative (the drone has gone further to the reference position), then PID will be negative.

The second term is based on the influence of the "past" of the error: the performance won't be the same if we get a punctual error or if the error has been gotten for a while. In the second case the response to this error will be more energetic.

Finally the third term reflects the "future" of the error, that is to say the prediction of this error. Imagine we are 5 units away from our desire position, that is to say, error=5. It is not the same if we are getting closer to this desire position or if we are getting away from such objective. In the first case probably it won't be necessary any actuation, but in the second one it is needed a correction.

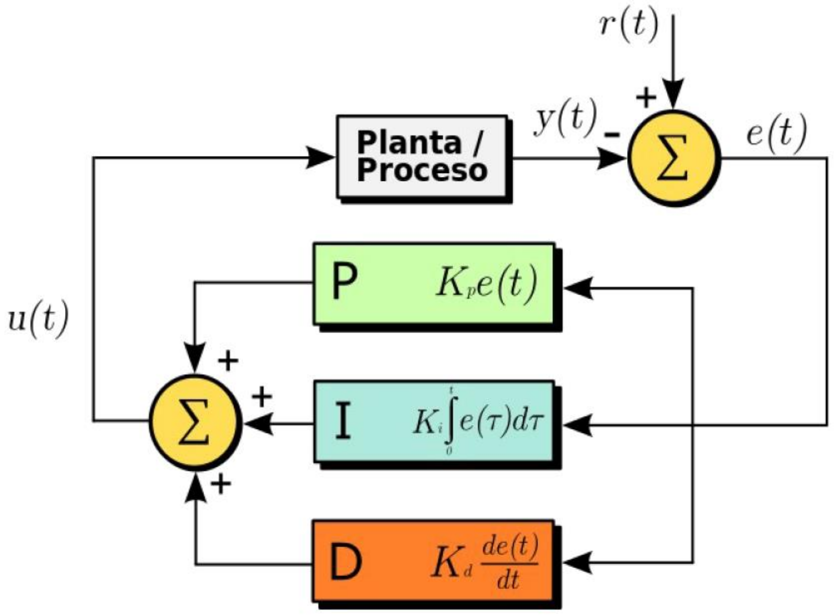


Figure 5. 2. PID controller block diagram.

In this particular case, it will be implemented a discrete controller, so the formula shown above can be approximated to the following, in which each time the position is measured and the error e_n is calculated:

$$e_n = (P_{ref} - pos_n)$$

To get the control term, the instant error is traduced naturally in the error measured at the time n, but for the integral part of the formula and the derivative part will need some numeric approximations:

- The integral is substitute for a summation of values previous to the error. Besides it is limited in the time, adding N errors, instead of the complete amount of errors. The h term (interval of time between samples) but such term can be absorbed by the integral constant K_i.
- The numeric estimation of the derivative part, the quickest is to substitute for the difference between consecutives errors, though accurately estimations could be found. Again, the factor *h* that appears can be absorbed by the K_d constant, so it won't be necessary introduce it.

$$\frac{de(t)}{dt} \approx \frac{e_n - e_{n-1}}{h}$$

The discrete variant of the control term will be therefore:

$$pid_n = K_p \cdot e_n + \sum_{k=1}^{N} e_{n-k} + K_d(e_n - e_{n-1})$$

Not all the terms explained will be implemented. It will depend on the application, that's why we can simply design a proportional controller, wiping the derivative and integral parts out, or a PD controller wiping just the integral part out.

After explaining how the PID is going to be implemented, it is important to remark that in this particular case what will be implemented is a controller for the X, Y and Z axis, and also for the yaw positions. So, all in all four controllers will be designed for this purpose, one for each position.

Using a trial an error method, the gaining constants K_p, K_i and K_d have been adjusted for each controller. The results obtained are shown below:

- **PID for Z position**

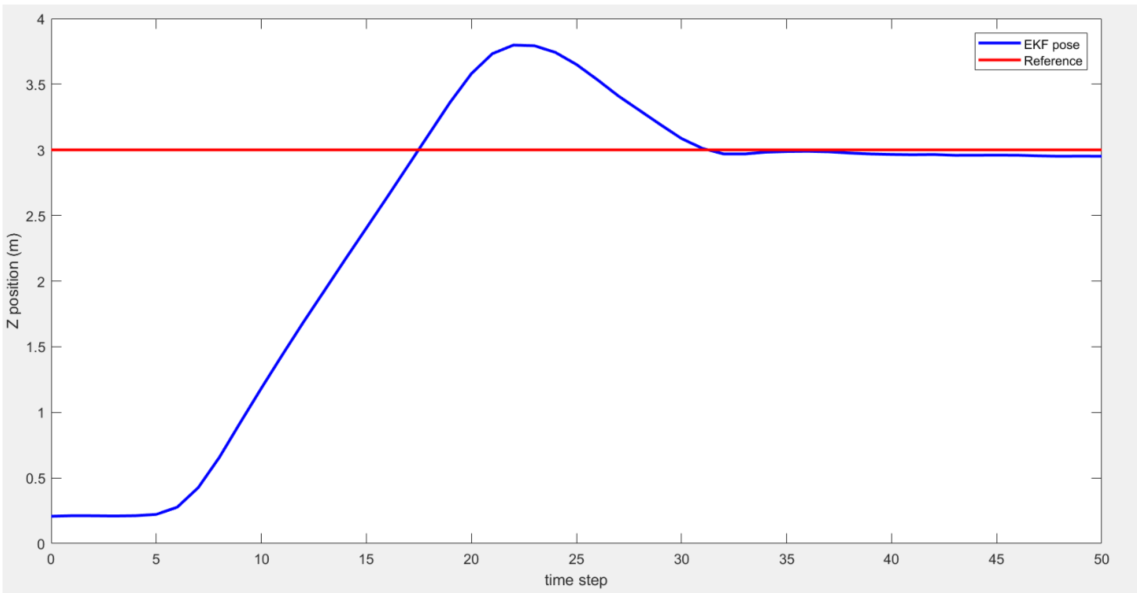


Figure 5. 3. Results PID controller Z position.

- **PID for Y position**

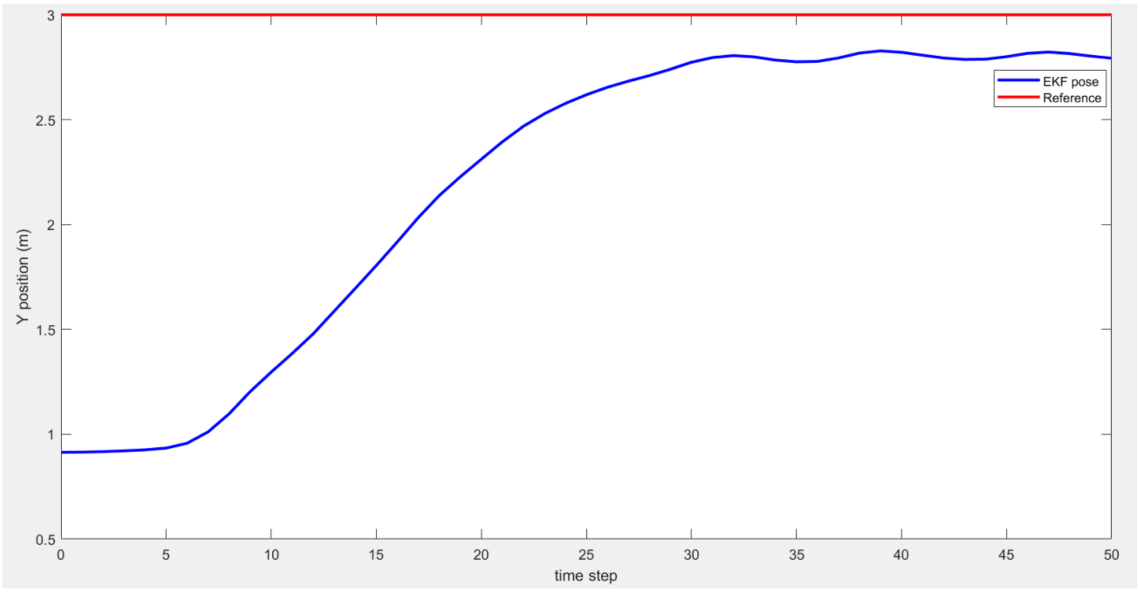


Figure 5. 4. Results PID controller Y position.

- **PID for X position**

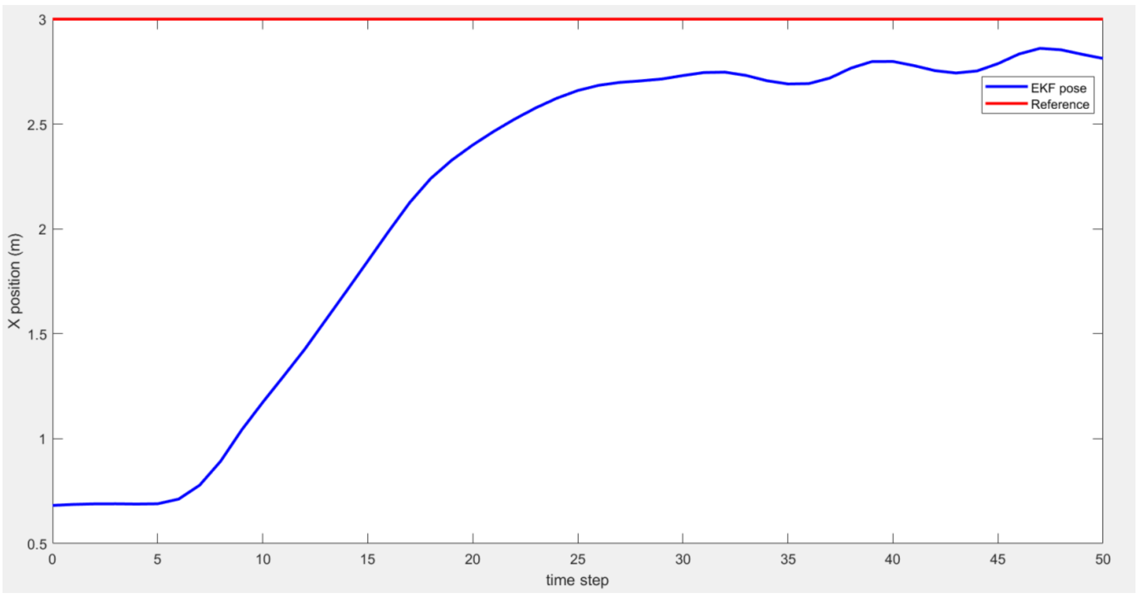


Figure 5. 5. Results PID controller X position.

- **P controller for yaw angle**

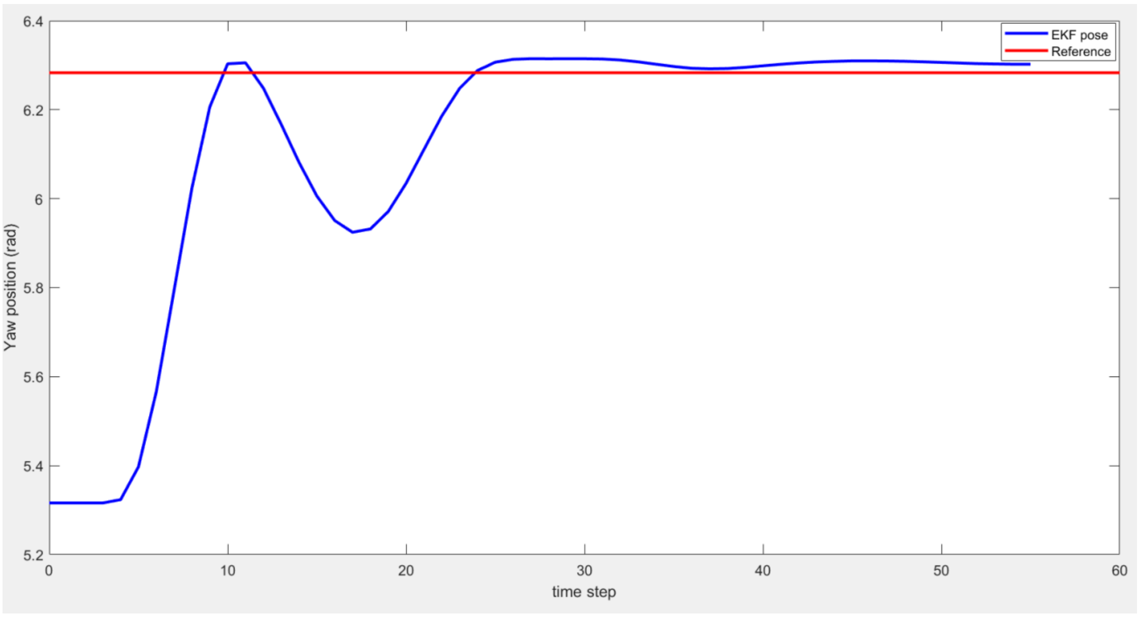


Figure 5. 6. Results P controller yaw position.

The table 5.1 represents the numeric results obtained for each controller.

Position	Type	Kp	Ki	Kd
Z	PID	0.4	1.0	0.7
Y	PID	0.1	0.15	4.0
X	PID	0.1	0.4	4.0
Yaw	Proportional	2.5	-	-

Table 5. 1. Constants of the controllers.

Now it will be explained how the PID controller for the Z position was implemented, this will be enough to understand the behaviour of a PID controller and how the others were also implemented.

First it will be necessary to set the reference and get the real position of the drone. Now we can calculate the proportional error:

$$E_Z = R_Z - P_Z$$

where R_z is the reference and P_z is the real and current position.

The second step is to get the integral error, to do so, we need to add the last 20 samples of the current error, and get the average:

$$E_{avg} = \sum_{k=1}^{n=20} E_{z_{n-k}}$$

And then the integral error is calculated as:

$$E_{iz} = \frac{E_{avg}}{20}$$

The last step is to get the derivative error, which can be obtained as the subtraction of the current error minus the previous one:

$$E_{dz} = E_{z_n} - E_{z_{n-1}}$$

Now the constants K_p, K_i and K_d are set to a random value, and using the trial and error method while taking a look at the behaviour of the system the values of the constants are updated until a good solution is obtained (K_p =0.4, K_i =1, K_d =0.7).

Then calculate *PID*$_z$ from the next formula:

$$PID_z = K_{pz} \cdot E_z + K_{iz} \cdot E_{iz} + K_{dz} \cdot E_{dz}$$

As it was explained before, in this chapter, the objective is to set a *PWM* value to control the position of the drone, as it was explained above, this *PWM* are set in a range between 1100 and 1900, where 1500 means “stay in your current position”. So this *PID*$_z$ will return a number between [-1,1] so if the result, for example is 50, automatically the outcome of PID_z will be 1 and if the result is -200 the outcomes will be automatically -1. Then to set the required *PWM*:

$$\mathrm{PWM_z} = 1500 + 400 \cdot \mathrm{PID_z}$$

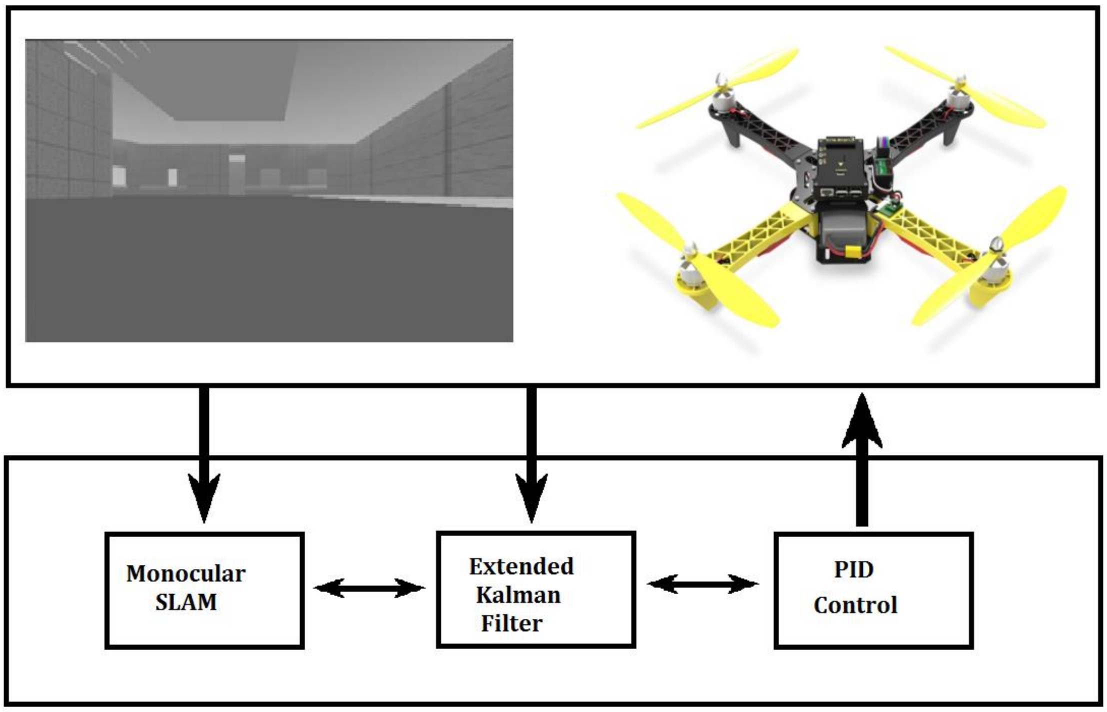


Figure 5. 7. Components of the navigation system.

Now that the PID has been developed, the navigation system will consist of three major components: a monocular SLAM implementation for visual tracking, an EKF for data fusion and prediction and a PID control for pose stabilization, all of it implemented in the gazebo simulator as figure 5.7 shows.

CHAPTER 6

CONCLUSIONS AND FUTURE WORK

6.1. Conclusions

In this project it has been employed a low-cost aerial vehicle, the ErleCopter, with ROS as a software platform to work with it. Gazebo has been the simulator used to test all the algorithms and control the robotic platform.

As it can be seen it is possible to use commercial low cost quadcopters to perform an estimation of the current localization and mapping, of course using the vSLAM techniques presented within this project.

The main limit faced in this project was the lack of resources of the robotic platforms, this means, that given the processor (raspberry pi 2) and the vSLAM algorithms studied before, it wasn't able to perform an on board tracking of the robot, mainly, as it was said in previous chapters, ORB-SLAM and LSD-SLAM need a powered processor to work properly, otherwise the algorithm gets frozen and stuck, losing continuously the tracking and mapping of the platform.

On the other hand, the implemented EKF, as it was shown, works more accurate than just using a simple vSLAM algorithm, although, the idea was to fuse such algorithms with the data from a laser, that was part of another project. The timing wasn't correct, so it had to be done just using the information obtained from the camera.

It is important to remark the importance of the data fusion in order to achieve a proper performance of the current system. Adding a sensor such as monocular camera brought a feedback of the position helping the system to estimate the pose of the robotic platform, and correcting the prediction model took from the ErleCopter kinematics and dynamics.

The real ErleCopter, after several trials turned out to be very unstable, talking in controllability terms, besides it has a lot of different modes which change its behaviour during the flight. For the simulator a PID was implemented, taking the control of the robotic platform using the data obtained from the EKF.

6.2. Future Work

Thus, it proposes different future works to keep investigating along this path.

The first proposal consists on the improvement of the controllability of the real platform, making possible a real flight of the ErleCopter indoors.

Another option would be the research of vSLAM algorithms, finding a compromise between the weight of the algorithm and its accuracy. In this project PTAM was found the best option for this purpose, however it might be more algorithms that might improve this aspect: lower the computability weight, improving the accuracy of the SLAM technique.

Finding another aerial robotic platform, with a better processor, and implementing on it a heavy vSLAM technique could solve the problem of the in real time flight, that couldn't be done in this project.

Other problem that could be faced following the ErleCopter trend, besides from the SLAM, is making such robotic platform autonomous. For this project, several paths were built and followed by the quadcopter using the implemented PID, such path was treated as a reference of the robotic platform. Therefore an explorer autonomous Erelcopter quadcopter as a future application, meshing information from a bunch of different sensors cold turn useful.

All in all it can be said that the objectives for this project were completed: a study of the three main vSLAM algorithms, an implementation of an EKF for the robotic platform, and an implementation of a PID to improve the controllability of such platform.

CHAPTER 7

USER’S MANUAL

In this chapter, the instructions to use the several packages found in this project will be explained. The launch of the vSLAM algorithms both in the real platform and in the simulator are explained in chapter 3, so no reference of them will be shown here.

7.1. Downloading the necessary tools

- Install ROS, Gazebo and the ErleCopter Gazebo models, following the steps: http://docs.erlerobotics.com/simulation/configuring_your_environment
- Install LSD-SLAM following the steps given in the GitHub website: https://github.com/tum-vision/lsd_slam
- Install ORB-SLAM following the steps given in the GitHub website: https://github.com/raulmur/ORB_SLAM
- Install PTAM following the steps given in the GitHub website: http://wiki.ros.org/ethzasl_ptam

7.2. Launching the ErleCopter world in Gazebo

Open a terminal and set the following commands:

```
source ~/simulation/ros_catkin_ws/devel/setup.bash
cd simulation/ardupilot/ArduCopter
../Tools/autotest/sim_vehicle.sh -j 4 -f Gazebo --map –console
param load/home/usuario/simulation/ardupilot/Tools/Frame_params/Erle-
Copter.param
```

In another terminal:

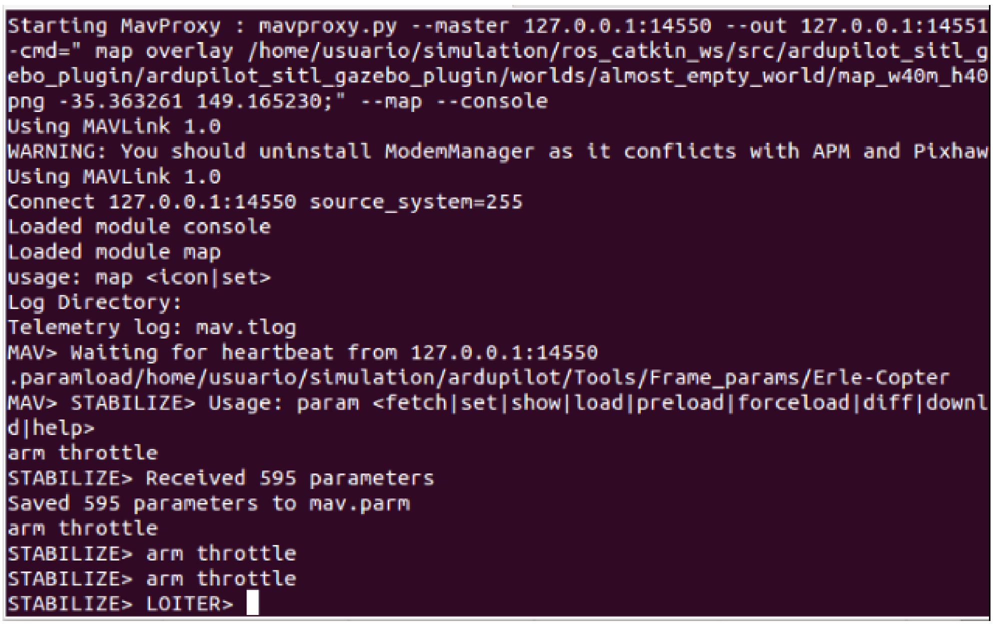
```
source ~/simulation/ros_catkin_ws/devel/setup.bash
roslaunch ardupilot_sitl_gazebo_plugin erlecopter_spawn.launch
```

7.3. Setting the world

To change the appearance of the world, you can just add objects in it by clicking on the left upper corner of the Gazebo simulator, and saving the file as it appears ros_catkin_ws/src/ardupilot_stil_gazebo_plugin/worlds/empty.world

7.4. Arming the ErleCopter

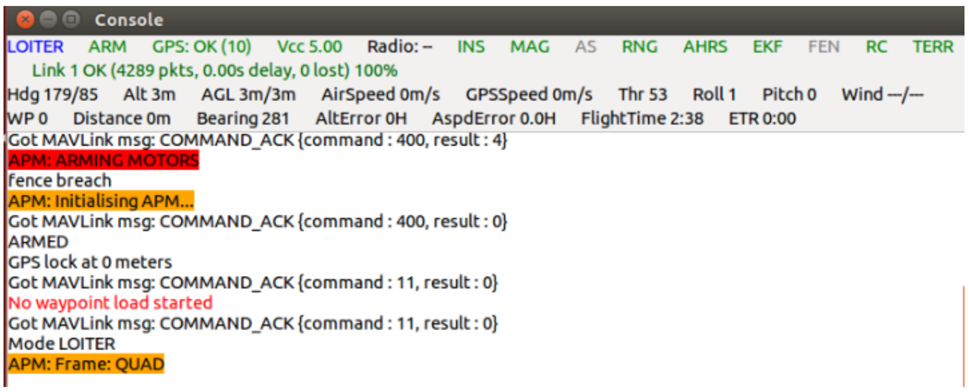
In the ardupilot terminal



```
Starting MavProxy : mavproxy.py --master 127.0.0.1:14550 --out 127.0.0.1:14551
-cmd=" map overlay /home/usuario/simulation/ros_catkin_ws/src/ardupilot_sitl_g
ebo_plugin/ardupilot_sitl_gazebo_plugin/worlds/almost_empty_world/map_w40m_h40
png -35.363261 149.165230;" --map --console
Using MAVLink 1.0
WARNING: You should uninstall ModemManager as it conflicts with APM and Pixhaw
Using MAVLink 1.0
Connect 127.0.0.1:14550 source_system=255
Loaded module console
Loaded module map
usage: map <icon|set>
Log Directory:
Telemetry log: mav.tlog
MAV> Waiting for heartbeat from 127.0.0.1:14550
.paramload/home/usuario/simulation/ardupilot/Tools/Frame_params/Erle-Copter
MAV> STABILIZE> Usage: param <fetch|set|show|load|preload|forceload|diff|downl
d|help>
arm throttle
STABILIZE> Received 595 parameters
Saved 595 parameters to mav.parm
arm throttle
STABILIZE> arm throttle
STABILIZE> arm throttle
STABILIZE> LOITER>
```

Figure 7. 1. Ardupilot terminal.

- Now type arm throttle, and press intro
- To change the mode of the Erlecopter you can type in the same terminal:
- Mode name_of_the_mode, for instance, mode LOITER.
- All the changes will appear in the Console window, shown in figure 7.2.



```
Console
LOITER  ARM  GPS: OK (10)  Vcc 5.00  Radio: --  INS  MAG  AS  RNG  AHRS  EKF  FEN  RC  TERR
   Link 1 OK (4289 pkts, 0.00s delay, 0 lost) 100%
Hdg 179/85  Alt 3m  AGL 3m/3m  AirSpeed 0m/s  GPSSpeed 0m/s  Thr 53  Roll 1  Pitch 0  Wind --/--
WP 0  Distance 0m  Bearing 281  AltError 0H  AspdError 0.0H  FlightTime 2:38  ETR 0:00
Got MAVLink msg: COMMAND_ACK {command : 400, result : 4}
APM: ARMING MOTORS
fence breach
APM: Initialising APM...
Got MAVLink msg: COMMAND_ACK {command : 400, result : 0}
ARMED
GPS lock at 0 meters
Got MAVLink msg: COMMAND_ACK {command : 11, result : 0}
No waypoint load started
Got MAVLink msg: COMMAND_ACK {command : 11, result : 0}
Mode LOITER
APM: Frame: QUAD
```

Figure 7. 2. Erelcopter Console

7.5. Publishing a topic and moving the quadcopter

- Once the simulator is running, (only for the first time) type in the ardupilot terminal $ rosrun mavros mavparam set SYSIS_MYGCS 1, you should get 1 as an answer.
- Again in the ardupilot terminal (also only for the first time) type param set ARMING CHECK 0 and param set SYSID_MYGCS 1.
- Then arm the quadcopter.
- To move the drone upwards: rostopic pub -1 /mavros/rc/override mavros_msgs/OverrideRCIn '[1500, 1500, 1700, 1500, 1500, 1500, 1500, 1500]'
- To move the drone to the right: rostopic pub -1 /mavros/rc/override mavros_msgs/OverrideRCIn '[1600, 1500, 1500, 1500, 1500, 1500, 1500, 1500]'
- To move the drone forward: rostopic pub -1 /mavros/rc/override mavros_msgs/OverrideRCIn '[1500, 1400, 1500, 1500, 1500, 1500, 1500, 1500]'
- To move it around the yaw angle: rostopic pub -1 /mavros/rc/override mavros_msgs/OverrideRCIn '[1500, 1500, 1500, 1600, 1500, 1500, 1500, 1500]'
- To maintain the quadcopter at the same position: :rostopic pub -1 /mavros/rc/override mavros_msgs/OverrideRCIn '[1500, 1500, 1500, 1500, 1500, 1500, 1500, 1500]'

7.6. Taking off automatically

- Launch the Gazebo simulator as the point 7.2
- Arm the quadcopter and set the STABILIZE mode
- In another terminal run the created package:

```
source ~/simulation/ros_catkin_ws/devel/setup.bash
rosrun ros_despegue ros_despegue
```

The copter will take off until it reaches 2 meters of height, where it will stop and stay stable.

7.7. Launching the PID controller

- Follow the steps of the subchapter 7.2
- Set the path as a reference, in another terminal:

```
source ~/simulation/ros_catkin_ws/devel/setup.bash
rosrun ros_local_reference ros_local_reference
```

- Run the PID package

```
source ~/simulation/ros_catkin_ws/devel/setup.bash
rosrun ros_pid_erle ros_pid_erle
```

7.8. Launching the EKF

- Follow the steps of the subchapter 7.2
- Open PTAM as in chapter 3. Once PTAM is running get the scale factor:

```
source ~/simulation/ros_catkin_ws/devel/setup.bash
rosrun escala_vslam escala_vslam
```

- Move the drone with a pure translational movement of the z axis, upwards or downwards to get the scale using the ultrasonic sensor.
- Set the path as a reference, in another terminal:

```
source ~/simulation/ros_catkin_ws/devel/setup.bash
rosrun ros_local_reference ros_local_reference
```

- Run the EKF package to estimate the ErleCopter position and introduce the calculated scale.

```
source ~/simulation/ros_catkin_ws/devel/setup.bash
rosrun ekf_erle ekf_erle
```

7.9. Creating a package

- Create a package folder in the workspace (example package ros_practica_1)

```
cd ~simulation/ros_catkin_ws/src
catkin_create_pkg ros_practica_1 std_msgs rospy roscpp tf
```

- Once the package is created, you should modify the Cmakelists.txt as follows:

```
cmake_minimum_required(VERSION 2.8.3)
project(ros_practica_1)

## Find catkin macros and any catkin packages
find_package(catkin REQUIRED COMPONENTS roscpp rospy std_msgs
genmsg)

# Generate added messages and services with any dependencies listed here
 generate_messages(DEPENDENCIES std_msgs)

# Declare a catkin package
catkin_package()

include_directories(include ${catkin_INCLUDE_DIRS})

add_executable(ros_practica_1 src/main.cpp)
target_link_libraries(ros_practica_1 ${catkin_LIBRARIES})
add_dependencies(ros_practica_1 ros_practica_1_generate_messages_cpp
})

## Mark executables and/or libraries for installation
install(TARGETS ros_practica_1
   ARCHIVE DESTINATION ${CATKIN_PACKAGE_LIB_DESTINATION}
   LIBRARY DESTINATION ${CATKIN_PACKAGE_LIB_DESTINATION}
   RUNTIME DESTINATION ${CATKIN_PACKAGE_BIN_DESTINATION})
```

- Now inside the src folder of the created package folder create your main.cpp program.
- Compile the package:

```
source ~/simulation/ros_catkin_ws/devel/setup.bash
catkin_make --pkg ros_practica_1
```

- Execute the package:

```
source ~/simulation/ros_catkin_ws/devel/setup.bash
rosrun ros_practica_1 ros_practica_1
```

CHAPTER 8

SPECIFICATIONS

Now, a list containing the main software and hardware tools employed within this project is shown:

8.1. Hardware Specifications

- PC Intel i5-6500 of 64 bits and 3GHz with 8 GB of RAM
- Erlecopter quadcopter with a monocular camera on board.

8.2. Software Specifications

- Operating System Ubuntu 14.04 LTS 64 bits
- Framework ROS Indigo
- Matlab R2017a

CHAPTER 9

BUDGET

This chapter will describe the theoretical cost of the whole project. It will include the equipment cost and the professional fees. Finally, the taxes will be added for getting the total cost of the project.

9.1. Equipment cost

In this section, the cost of the different materials (hardware and software) is detailed and the VAT (21%) is included.

Item		Unit price (euro)	Unit	Total cost
Hardware	ErleCopter Drone	1159	1	1159
	Lenovo U31 Laptop	749	1	749
Hardware total cost				**1908**
Software	Ubuntu v14.04	0	1	0
	Robot Operating System	0	1	0
	ROS packages	0	1	0
	Matlab (Student edition)	69	1	69
	Microsoft Office 2010	74.99		74.99
Software total cost				**143.99**
Equipment total cost				**2051.99**

Table 9. 1. Equipment cost.

9.2. Professional fees

In this section the different professional fees are calculated. These fees are calculated as gross incomes. The following table includes all the professional activities related with the project.

Activity	Price (euro/hour)	Time (hours)	Total cost (euro)
Engineering	3.15	400	1260
Writing up	3.15	65	205
Fees total cost			**1465**

Table 9. 2. Professional fees

9.3. Total cost

The theoretical total cost of the whole project is itemized in this section and presented in below:

Equipment cost	2051.99
Professional fees	1465
Printing	60
Total	**3576.99**

Table 9. 3. Total cost.

10. BIBLIOGRAPHY

[1] Barrucada Documentation [Online]. Available: http://www.army-technology.com

[2] Boeing Documentation [Online]. Available: http://www.boeing.com

[3] Parrot Documentation [Online]. Available: http://www.parrot.com

[4] Mikrocopter Documentation [Online]. Available: http://www.mikrokopter.de

[5] Sergio García Gonzalo,"Visual SLAM Algorithms for Aerial Robots",University of Alcalá,2016.

[6] Seung-Hun Kim, Changwoo Park ," Localization of Robot with Ceiling-View Cameras in Indoor Environment", ICMAR ,2012.

[7] Jae-Hong Shima and Young-Im Chob," A Mobile Robot Localization using External Surveillance Cameras at Indoor", HARMS, 2015.

[8] Shaojie Shen, Nathan Michael and Vijay Kumar," Autonomous Multi-Floor Indoor Navigation with a computationally constrained MAV", IEEE International Conference on Robotics and Automation, 2011.

[9] Koray Celik, Soon-Jo Chung, Matthew Clausman and Arun K.Somani, "Monocular Vision SLAM for Indoor Aerial Vehicles", IEEE, 2013.

[10] Cameron Roberts,"GPS Guided Autonomous Drone", University of Evansville,2016.

[11] Sunhong Park, Shuji Hashimoto," Indoor localization for autonomous mobile robot based on passive RFID", IEEE, 2008.

[12] Rodrigo Munguía, Sarquis Urzua," Vision-Based SLAM System for Unmanned Aerial Vehicles", MDPI, 2015.

[13] Peter Cheeseman, Randall Smith," Estimating Uncertain Spatial Relationships in Robotics", SRI International, 2003.

[14] Gamini Dissanayake, Paul Newman," A solution to the simultaneous localization and map building (SLAM) problem", IEEE , 2001.

[15] Sebastian Thrun, Mike Montemerlo," Stanley: The Robot that Won the DARPA Grand Challenge", Stanford University , 2006.

[16] Adam Bry, Abraham Bachrach and Nicholas Roy," State Estimation in GPS-Denied Environments Using On board Sensing", MIT,2012.

[17] Nicolás Blanco Fernández,”SLAM basado en láser para el robot aéreo ErleCopter”, University of Alcalá.

[18] Klein, Georg, and David Murray. "Parallel tracking and mapping for small AR workspaces.”, IEEE, 2007.

[19] Lentin Joseph,”Mastering ROS for robotics programming”, Packt Publishing, 2016.

[20] ROS Wiki [Online]. Available: http://wiki.ros.org/es

[21] ErleRobotics Documentation [Online]. Available: http://erlerobotics.com

[22] López, E., Barea, R., Gómez, A., Saltos, A., Bergasa, L., Molinos, E., Nemra, A.,”Indoor SLAM for micro aerial vehicles using visual and laser sensor fusion”,Second Iberian Robotics Conference: Advances in Robotics, Volumen 1 ,2015.

[23] Engel, J., Schöps, T., Cremers, D.,”LSD-SLAM: Large-scale direct monocular SLAM” In Computer Vision (ECCV), 2014.

[24]] Engel, J., Sturm, J., Cremers,D.,”Accurate Figure Flying with a Quadrocopter Using Onboard Visual and Inertial Sensing” , TUM,2012.

[25] López Torres,P. “Análisis de algoritmos para localización y mapeado simultáneo de objetos”, Universidad de Sevilla, 2016.

[26] “Gazebo Website”[Online]Available: http://gazebosim.org/

[27] Brito Domingues,J. “Quadrotor Prototype”,Universidade Técnica de Lisboa,2009.

[28] G.A Einicke and L.B. White “Robust extended kalman filtering”, IEEE Transactions on Signal Processing,vol. 47,1990.

[29] Saltos Vásquez, Álvaro Andrés. “Desarrollo de un Sistema de SLAM para el robot AR.Drone”, University of Alcalá, 2015.

[30] Gómez Rubio,Alejandro. “Control de robots aéreos en entornos interiores con ROS”, University of Alcalá, 2017.