Universidad de Alcalá
Escuela Politécnica Superior

MÁSTER EN INGENIERÍA DE TELECOMUNICACIÓN

Trabajo Fin de Máster

Visual SLAM Algorithms for Aerial Robots

Autor: Sergio García Gonzalo

Tutor/es: María Elena López Guillén

2016

Universidad de Alcalá

ESCUELA POLITÉCNICA SUPERIOR

MASTER UNIVERSITARIO EN
INGENIERÍA DE TELECOMUNICACIÓN

TRABAJO DE FIN DE MÁSTER

VISUAL SLAM ALGORITHMS FOR AERIAL ROBOTS

Sergio García Gonzalo

2016

UNIVERSIDAD DE ALCALÁ
Escuela Politécnica Superior

MÁSTER EN INGENIERÍA DE TELECOMUNICACIÓN

Trabajo Fin de Máster

VISUAL SLAM ALGORITHMS FOR AERIAL ROBOTS

Autor: Sergio García Gonzalo
Tutora: María Elena López Guillén

TRIBUNAL:

Presidente: Manuel Rosa Zurera

Vocal 1º: Miguel Ángel García Garrido

Vocal 2º: María Elena López Guillén

FECHA: Junio 2016

Agradecimientos

"*All we have to decide is what to do with the time that is given us*."
— J.R.R. Tolkien.

Ante todo quiero agradecer a mi familia su incondicional apoyo y paciencia, las cuales me han permitido llegar a ser lo que soy y poder estar escribiendo estas líneas. Gracias a mi padre, a mi madre y a mi hermano por todo.

Quisiera expresar mi agradecimiento a todo aquel que me haya ayudado a llegar hasta aquí. Considero que sin la guía, experiencia y afán de mi tutora, la Dra. Elena López Guillén, no hubiese descubierto el afán por la investigación ni el campo con el que tanto he disfrutado llevando a cabo este trabajo. Quiero agradecer también a todo aquel profesor que haya buscado transmitirme sus conocimientos, ya que muchos de ellos me han ayudado no solo formándome, sino también ofreciendo su consejo.

Aprovecho para saludar a toda la gente que he conocido a lo largo de mi estancia en la universidad, tanto en España como durante mi estancia Erasmus en Suecia. Agradezco todos los buenos momentos, pero sobre todo quiero dar las gracias a aquellos que después de años siguen prestándome su ayuda. También quiero saludar y agradecer a todos mis amigos los momentos los cuales me han permitido librarme del stress.

Por último, pero no por ello menos importante, voy a agradecer todo el esfuerzo realizado por Arantxa. Todo lo que pueda decir sobre su entrega y paciencia es poco, ayudándome entre otras cosas a redactar textos que tratan temas de los cuales conoce poco o nada y aguantándome en mis días menos amables.

RESUMEN

En este Trabajo de Fin de Máster se estudian técnicas de Monocular Visual SLAM (VSLAM a partir de ahora) implementadas sobre robots aéreos. Estas técnicas se caracterizan por el uso de una sola cámara para estimar la posición y la profundidad para así poder crear un mapa del entorno del robot. Tras un estudio del estado del arte de algoritmos de monocular VSLAM se ha decidido implementar las técnicas LSD-SLAM (Large-Scale Direct Monocular SLAM), y ORB-SLAM (Oriented FAST and Rotated BRIEF SLAM). También se realiza un estudio de PTAM, una técnica desarrollada previamente a las anteriormente mentadas pero que sirve para entenderlas mejor de forma que se pueda establecer una comparativa.

Los algoritmos mencionados en el anterior párrafo se implementan sobre el contexto de rescate y/o navegación de reconocimiento con micro vehículos aéreos (*Micro Aerial Vehicles* - MAV). En este tipo de aplicaciones, el MAV debe utilizar sus propios sensores incorporados para navegar de forma autónoma en entornos interiores desconocidos, hostiles y sin cobertura de GPS –como ruinas o edificios semiderruidos–.

Para su aplicación en la estimación de la posición de un robot aéreo, la información obtenida mediante VSLAM se fusiona con la obtenida de la Unidad de Medición Inercial (*Inertial Measurement Unit* - IMU) –presente en todos los vehículos aéreos–y otros sensores abordo, utilizando un Filtro de Kalman Extendido (*Extended Kalman Filter* - EKF). Además, se utiliza la información de los sensores a bordo del robot para resolver el problema de la ambigüedad de escala propia de los algoritmos de VSLAM monocular.

Por último, y utilizando la estimación de posición obtenida anteriormente, se desarrolla la capacidad de controlar el robot aéreo en tres dimensiones mediante el uso de la cámara frontal y la IMU, actuando sobre los motores del robot en función de órdenes enviadas en tiempo real o programadas previamente.

La implementación se ha realizado sobre un robot aéreo comercial de bajo coste, el cual no es posible programar de forma sencilla. Por esta razón el control se realiza desde un *Ground System* siendo éste un PC remoto. Este PC tendrá instalado ROS (*Robot Operating System*) como entorno de desarrollo.

***Palabras Clave**: Micro vehículos aéreos; Monocular VSLAM; navegación en interiores; fusión sensorial; mapeado y localización simultáneas; Robot Operating System.*

ABSTRACT

In this thesis Monocular Visual SLAM (VSLAM in the following) techniques implemented on Micro Aerial Vehicles (MAV in the following) are studied. These techniques use only one camera to estimate the position and depth in order to create a map of robot's environment. After a study of the state-of-art monocular VSLAM algorithms, we decided to implement two of these algorithms in our system: LSD-SLAM (Large-Scale Direct Monocular SLAM) and ORB-SLAM (Oriented FAST and Rotated BRIEF SLAM), although there will be a study of PTAM too. PTAM is a VSLAM technique developed years before ORB and LSD but helps to understand both so we can establish a comparative.

These algorithms are implemented in the context of rescue and/or recognition navigation tasks in indoor environments. In this kind of applications, the MAV must rely on its own onboard sensors to autonomously navigate in unknown, hostile and GPS denied environments –such as ruined or semi-demolished buildings–.

For the estimation of MAV's position, the obtained information from VSLAM is fused with the one obtained from the Inertial Measurement Unit (IMU in the following) –present in all MAVs– and other onboard sensors, using an Extended Kalman Filter (EKF in the following). Furthermore, the information from the onboard sensors is used to solve the problem of scale ambiguity common in most of monocular VSLAM algorithms.

Finally, and from the previous position estimation, the frontal camera and the IMU are used to develop the ability of control the MAV in 3D. This control works in MAV's thrusters depending on the real-time or previously programmed sent commands.

The system has been implemented over a commercial low-cost aerial robot. This robot is not easily programmed, so the control has been managed from a *Ground System*. This system is a remote PC with ROS (*Robot Operating System*) installed as an Integrated Development Environment.

Keywords: *Micro aerial vehicles; Monocular VSLAM; indoor navigation; sensor fusion; simultaneous localization and mapping; Robot Operating System*

CONTENTS INDEX

FIGURES INDEX

TABLES INDEX

CHAPTER 1: INTRODUCTION

1.1. Rise of MAVs

Drones are fast, agile and versatile robots that can be implemented in a wide spectrum of projects. Due to it and the tendency of the technology to the miniaturization, these robots are living a golden age of development. It is possible to find drones from the ones that can be held in one hand to others that can carry a person as payload. Specifically, MAVs have become an important tool not only in the military domain, but also in civilian environments. Particularly quadcopters are becoming more popular, especially for observational and exploration purposes in indoor and outdoor environments, but also for data collection, object manipulation or simply as high-tech toys.

Fig. 1. Different sizes of drones.

There are numerous examples where MAVs are successfully used in practice, for example for exploratory tasks such as inspecting the damaged nuclear reactors in Fukushima in March 2011, and for aerial based observation and monitoring of potentially dangerous situations, such as protests or large scale sport events.

There are however many more potential applications: a swarm of small, light and cheap quadcopters could be deployed to find survivors in collapsed buildings without risking human lives. Equipped with high-resolution cameras, MAVs could also be used as flying photographers, providing aerial based videos of sport events or simply taking holiday photos from a whole new perspective.

The main advantage of these robots is that they are unmanned, so they perform missions that are too “dull, dirty or dangerous”. Furthermore, having a flying behaviour similar to a traditional helicopter, a quadrocopter is able to land and start vertically, stay perfectly still in the air and move in any given direction at any time without having to turn first. This enables quadrocopters –contrary to traditional airplanes– to manoeuvre in extremely constrained indoor spaces such as corridors or offices, and makes them ideally suited for stationary observation or exploration in obstacle-dense or indoor environments.

The growing research on MAVs and the consequent improvement of technologies like microcomputers and onboard sensor devices has increased the performance requirements of such kind of systems. Enabled by GPS and MEMS inertial sensors, MAVs that can fly in outdoor environments without human intervention have been developed. Unfortunately, most indoor environments remain without access to external positioning systems, and autonomous MAVs are very limited in their ability to operate in these areas.

1.2. Key Challenges

In the ground robotics domain, combining wheel odometry with sensors such as laser range-finders, sonars, or cameras in a probabilistic SLAM (Simultaneous Localization and Mapping) framework has proven very successful. Many algorithms exist that accurately localize ground robots in large-scale environments; however, experiments with these algorithms are usually performed with stable, slow moving robots, which cannot handle even moderately rough terrain.

Unfortunately, mounting equivalent sensors onto a MAV and using an existing SLAM algorithm does not result in the same success. MAVs face a number of unique challenges that make developing algorithms for them far more difficult than their indoor ground robot counterparts. The requirements and assumptions that can be made with flying robots are sufficiently different that they must be explicitly reasoned about and managed differently.

These are the main key challenges when developing autonomous navigation systems for MAVs:

- **Limited Sensing Payload.** MAVs have a maximum amount of vertical thrust that they can generate to remain airborne, which severely limits the amount of payload available for sensing and computation compared to similar sized ground vehicles. This weight limitation eliminates popular sensors such as SICK laser scanners, large-aperture cameras, high-fidelity IMUs, RGB-D cameras or even the management of a stereo system. Instead, indoor air robots must rely on lightweight Hokuyo laser scanners, micro cameras and lower-quality MEMS-based IMUs, which generally have limited ranges, fields-of-view and are noisier compared to their ground equivalents.

- **Limited Onboard Computation.** Despite the advances within the community, SLAM algorithms continue to be computationally demanding even for powerful desktop computers and are therefore not usable on today's small embedded computer systems that might be mounted onboard MAVs. The computation can be offloaded to a powerful ground-station by transmitting the sensor data wirelessly; however, communication bandwidth then becomes a bottleneck that constrains sensor options. For example, camera data must be compressed with lossy algorithms before it can be transmitted over wireless links, which adds noise and delay to the measurements. The delay is in addition to the time taken to transmit the data over the wireless link. The noise from the lossy compression artefacts can be particularly damaging for feature detectors that look for high frequency information such as corners in an image. Additionally, while the delay can often be ignored for slow moving, passively stable ground robots, MAVs have fast and unstable dynamics, making control under large sensor delay conditions impossible.

- **Indirect Relative Position Estimates.** Air vehicles do not maintain physical contact with their surroundings and are therefore unable to measure odometry directly, which most SLAM algorithms require to initialize the estimates of the vehicle's motion between time steps. Although one can compute the relative motion by double-integrating accelerations, lightweight MEMs IMUs are often subject to unsteady biases that result in large drift rates. We must then recover the vehicle's relative motion indirectly using exteroceptive sensors –sensors that determine the measurements of objects relative to the robot's frame of reference–, and computing the vehicle's motion relative to reference points in the environment.

- **Fast Dynamics.** MAVs have fast dynamics, which results in a host of sensing, estimation, control and planning implications for the vehicle. When confronted with noisy sensor measurements, filtering techniques such as Kalman Filters are often used to obtain better estimates of the true vehicle state. However, the averaging process implicit in these filters mean that multiple measurements must be observed before the estimate of the underlying state will change. Smoothing the data generates a cleaner signal, but adds delay to the state estimates. While delays may have insignificant effects on vehicles with slow dynamics, the effects are amplified by the MAV's fast dynamics. Additionally, the named "*Ground effect*" may occur when flying close to the ground, ceiling or walls.

- **Need to Estimate Velocity.** The underdamped nature of the dynamics model implies that simple proportional control techniques are insufficient to stabilize the vehicle, since any delay in the system will result in unstable oscillations. For this reason, we must add damping to the system through the feedback controller, which emphasizes the importance of obtaining accurate and timely state estimates for both position and velocity. Traditionally, most SLAM algorithms for ground robots completely ignore the velocity states. MAVs do not incorporate sensors that can measure the current speed, so it has to be estimated by other means. For instance, the Bebop Drone (used in this work) puts to use a vertical camera placed in its bottom to estimate the horizontal velocity. In order to enable the drone to keep its position in spite of wind, an optical-flow based motion estimation algorithm utilizing the full 60 fps from the floor camera is performed onboard, estimating the drone's horizontal speed. The exact way these values are determined however is not documented.

- **Constant Motion.** Unlike ground vehicles, a MAV cannot simply stop and perform more sensing when its state estimates have large uncertainties. Instead, the vehicle is likely to be unable to estimate its velocity accurately, and as a result, it may pick up speed or oscillate, degrading the sensor measurements further. Thus, planning algorithms for air vehicles must not only be biased towards paths with smooth motions, but must also explicitly reason about uncertainty in path planning.

- **3D Motion.** Finally, MAVs operate in a truly 3D environment since they can hover at different heights. While it is reasonable for a ground robot to focus on estimating a 2D map of the environment, for air vehicles, the 2D cross section of a 3D environment can change drastically with height and attitude, as obstacles suddenly appear or disappear.

1.3. The ISLAMAV Project

This work is part of the ISLAMAV Project –developed by the RobeSafe Group of the Electronics Department of the University of Alcalá– whose final goal is the development of a MAV-based inspection system that will recognize indoor ruined or semi-ruined buildings in the context of rescue missions. This kind of environments will be unknown and GPS-denied, so the MAV will have to trust in its own onboard sensors. In order to achieve this goal several measurements from different sensors are fused to improve the pose estimation for MAVs in indoor environments. As a strategy of the fusion algorithm, each of the sensors must be able to provide its own pose estimation to endow the system with some redundancy that allows it to work in different environmental conditions.

Fig. 2. MAV put to use in a mission inside a ruined building.

The software architecture of the whole navigation system proposed in the ISLAMAV Project is shown in *Fig. 3*. As it can be seen, the SLAM system fuses the information of three sensorial systems: a scan-matcher module based on laser measurements, a VSLAM system based on a monocular camera, and the rest of onboard sensors (IMU, ultrasounds, etc.). The usage of monocular VSLAM is justified because due to their low weight and cost, monocular cameras are included in most of the commercial MAVs. However, its usage is constrained to environments with specific features and lighting conditions, and so a laser sensor will improve the performance of the SLAM system in indoor environments due to its high working rate and its direct and accurate range detection.

One of the requirements of the ISLAMAV Project is that the sensorial system has to be modular and configurable. So, this thesis focuses on the development of the monocular VLSAM module and the fusion with the onboard sensor measurements using an Extended Kalman Filter.

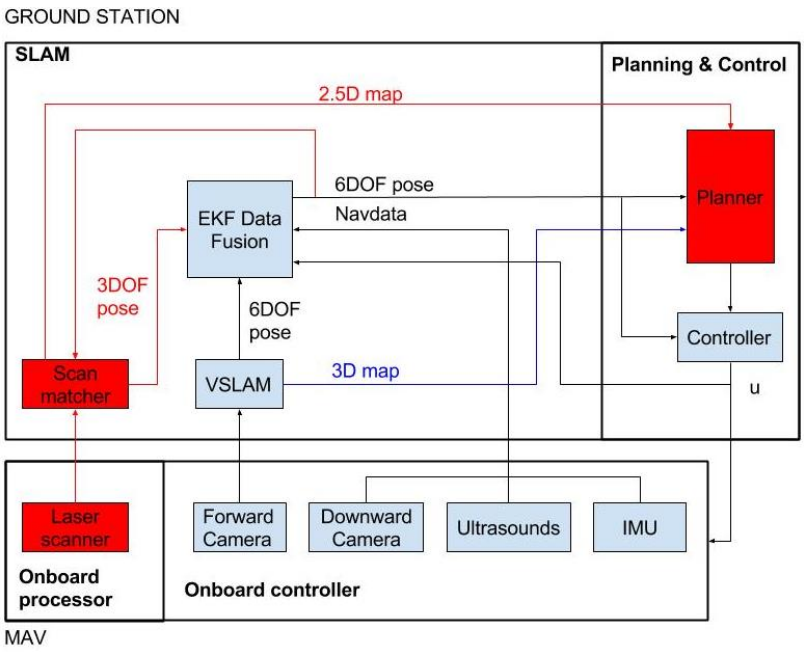


Fig. 3. Software architecture of the ISLAMAV Project: red modules correspond with out of the scope work; the blue modules are the ones implemented in this thesis.

To face the computational requirements, the system is composed of a flight and a ground unit, so that code can be distributed in different nodes using ROS (Robot Operating System). The ground unit will be implemented as a laptop with ROS installed on it. We had to divide the system in these two parts due to some problems related with the use of MAVs explained before: limited onboard computation and limited sensing payload.

The VSLAM algorithms that were chosen in order to calculate the pose estimation (along with the measurements from the other onboard sensors) and the map of the environment are: LSD-SLAM and ORB-SLAM. Both of them are put to use so a comparison between the two methods can be performed. The differences between them, as well as the strengths and weaknesses will be explained in Chapter 5.

One of the main problems of monocular camera VSLAM algorithms is the fact that it cannot calculate the scale of the data of tracking and mapping. It leads to a system that is not working with real-scale data, what could affect the integrity of an aerial robot. To solve this problem, our system uses the data from other onboard sensors to calculate the dynamic scale of the SLAM to return the real-time pose of the MAV without scale ambiguity.

In this work, up-to-date VLSAM algorithms are fused with measurements from other onboard sensors (IMU, sonar, vertical camera, etc.) to solve the SLAM problem in complex indoor environments and robustly estimate the 6DOF (six-degrees-of freedom) pose of the MAV, using a distributed system with a flight unit and a ground station.

In order to fuse measurements from the VSLAM algorithms and other onboard sensors, an EKF is implemented. Moreover, the system is able to calculate the dynamic scale of the measurements, what makes it a scale-aware system. Due to it, the EKF and the control stage work with real scaled data, in contrast to other monocular VSLAM systems.

The problem of autonomous indoor MAV localization was addressed as a software challenge, focusing on high-level algorithms integration rather than specific hardware. For this reason, we use a low-cost commercial platform with minor modifications and an open-source development platform (ROS), so drivers of sensors and some algorithms can be used without development.

1.4. Problem Statement and Initial Objective

The initial objective of this work is to study different monocular VSLAM techniques and its ability to be implemented in aerial robots in order to estimate their 6DOF position, taking into account the special constraints of this kind of platforms. The obtained pose estimation will be scaled using other onboard measurements, and finally will be fused with these measurements to improve the estimation. Besides, a position controller will be designed in order to guide the MAV to commanded target positions. Finally, another important objective is to implement the system in a real robotic platform to obtain experimental results that can be used to validate the study.

1.5. Outline

The remaining sections of this document are organized as follows:

In Chapter 2, a study of the state of the art is performed in order to explain some related work and to place this work in the field of study.

Chapter 3 presents the formulated hypothesis for this work. The necessary steps followed in order to validate this hypothesis are mentioned.

In Chapter 4, an overview of the system and the two sides of the architecture – both hardware and software– are explained.

Chapter 5 talks about the VSLAM algorithms and specifies which ones will be used for this work and why. A comparison performed by means of a benchmark is presented.

In Chapter 6, the data fusion and how it is achieved in this work is explained. All the models implemented in our EKF are described in detail.

Chapter 7 describes the PID controller developed in this thesis.

Chapter 8 explains the results obtained in real experiments, comparing estimation and tracking from a ground truth. How this ground truth system was elaborated is explained too.

In Chapter 9, the main conclusions and future work lines are summarized.

The last sections correspond to the diagrams, specifications, budget, user guide and bibliography of this work. The thesis also includes an appendix where the achieved additional activities are listed and explained.

CHAPTER 2: STATE OF THE ART

In this chapter a brief review of the main techniques used to develop autonomous navigation systems for aerial robots is presented. This study justifies the use of monocular cameras as the main sensor for navigation over other proposals for low-cost MAVs. It also includes a description of the closest related projects in order to contextualize the developed work.

2.1. Autonomous navigation of MAVs

Since the rise of use and research in the field of MAVs, there have been numerous efforts to fly quadrocopters autonomously. Most of these efforts have been made for outdoor situations. In this kind of environments the obstacles density is lower and the GPS signal will be enabled in almost all situations. For instance, in (Mellinger et al., 2011) an algorithm for addressing the controller design and the trajectory generation for a quadrotor manoeuvring in three dimensions is explained. Other authors use GPS-based methods in order to localize the drone in outdoor environments (Vago et al., 2015). Other applications are developed thinking in animal-based algorithms as in (Senanayake et al., 2016) (Lindsey et al., 2011) (Kushleyev et al., 2012) that performed a system based on a collaborative swarm of aerial robots.

However, in this work the system will have to accomplish a simultaneous localization and mapping of indoor, unknown, and GPS-denied environments. The autonomous navigation of MAVs in this kind of environments is even today an open area of research. It is not possible to use the classic odometry systems –based on encoders– of ground robots for MAVs. Due to it, this kind of systems must be replaced by inertial systems so new sensor-based strategies for the localization have been developed. Besides, the absence of a previous map of the environment makes mandatory the implementation of simultaneous localization and mapping (SLAM) solutions, as well as robust state estimation and control methods. For these solutions, the use of different sensors has been proposed in the literature as: range laser scanners (Grzonka et al., 2009); monocular cameras (Achtelik et al., 2012); stereo cameras (Fraundorfer et al., 2012) or RGB-D sensors (Huang et al., 2011) (Bylow et al., 2013). As it has been explained before, one of the main problems when using this kind of sensors in MAVs is the limited sensing payload. The drone put to use must be able to fly steady carrying the chosen sensor. This is not possible to most of the commercial low-cost drones and a specific and more expensive MAV may be used for this situation. In addition, such drone able to carry a heavy payload is usually too big to be managed in indoor environments. The drone's size may be a problem due to several reasons: it could be dangerous if flying near humans, and the strength of the thrusters could be enough to not allow the MAV to fly steady near walls or the floor due to the "ground effect".

So, the need to use small MAVs in indoor environments requires selecting the most appropriate sensors. Monocular cameras are light, small and cheap and indeed they are usually included in most low-cost commercial drones. Their inherent scale

ambiguity problem can be solved by taking advantage of other typical onboard sensors such as IMU or ultrasound sensors. That is the reason why this proposal will be explored in this thesis in order to solve the SLAM problem in MAVs.

2.2. Related Projects

In this section, some works whose framework is close to the one treated on this thesis will be explained. They correspond to the main successful indoor navigation systems for MAVs developed in the last years. Some of them use heavy sensors that cannot be used in small low-cost MAVs, but propose software architectures that have inspired this work. Others propose monocular camera-based navigation systems and will be contextualized with respect to this thesis.

One of the main works that served as a reference for the ISLAMAV project approach –within this thesis fits– is the one developed by Galton (Galton et al., 2009) in the Massachusetts Institute of Technology. This work presents a solution for enabling a quadrotor helicopter, equipped with a laser rangefinder sensor, to autonomously explore and map unstructured and unknown indoor environments. An overview of their solution to the key problems, including a multilevel sensing and control hierarchy, a high-speed laser scan-matching algorithm, an EKF for data fusion, a high-level SLAM implementation, and an exploration planner are provided. Finally, they show experimental results demonstrating the helicopter's ability to navigate accurately and autonomously in unknown environments.

In this work, the authors fuse measurements from a *Hokuyo* range laser scanner with the ones from the IMU. Thanks to it, they achieve a robust SLAM system that can face adverse situations as bad illuminated environments. For this implementation, they have to use a drone that is able to carry a heavy payload –*Hummingbird* from Ascending Technologies, as seen in *Fig. 4*–, a much more expensive hardware architecture than the proposed for this thesis.

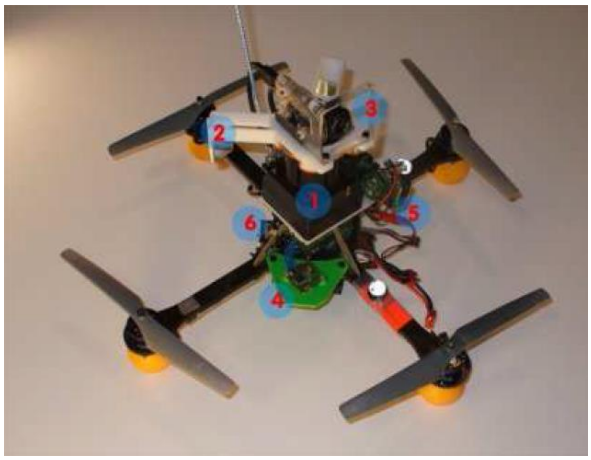


Fig. 4. Hummningbird drone carrying a Hokuyo laser sensor (Galton et al., 2009).

On the other hand, although their hardware platform target is not the same as ours, they have developed an autonomous system able to fly and avoid obstacles thanks to an estimation of the drone's relative position within its environment and a controller. The software architecture of the navigation system is shown in *Fig. 5*, and it has served as reference for the autonomous navigation side of this thesis.

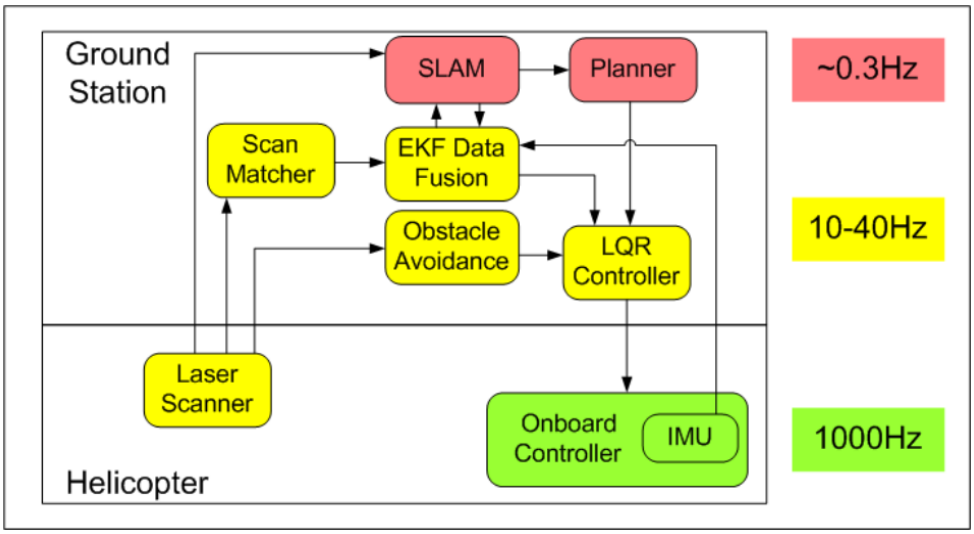


Fig. 5. Schematic of the sensing, control and planning architecture (Galton et al., 2009).

Few years later, another work is proposed in (Bachrach et al., 2012), a new system for visual odometry and mapping using an RGB-D camera and its application to autonomous flight. By leveraging results from recent state-of-the-art algorithms and hardware, their system enables 3D flight in cluttered environments using only onboard sensor data. All computation and sensing required for local position control are performed onboard the vehicle, reducing the dependence on unreliable wireless links. However, even with accurate 3D sensing and position estimation, some parts of the environment have more perceptual structure than others, leading to state estimates that vary in accuracy across the environment. If the vehicle plans a path without regard to how well it can localize itself along that path, it runs the risk of becoming lost or worse. The authors show how the Belief Roadmap (BRM) algorithm (Prentice et al., 2008) –a belief space extension of the Probabilistic Roadmap algorithm– can be used to plan vehicle trajectories that incorporate the sensing model of the RGB-D camera. They evaluate the effectiveness of their system for controlling a quadrotor, demonstrate its use for constructing detailed 3D maps of an indoor environment and discuss its limitations.

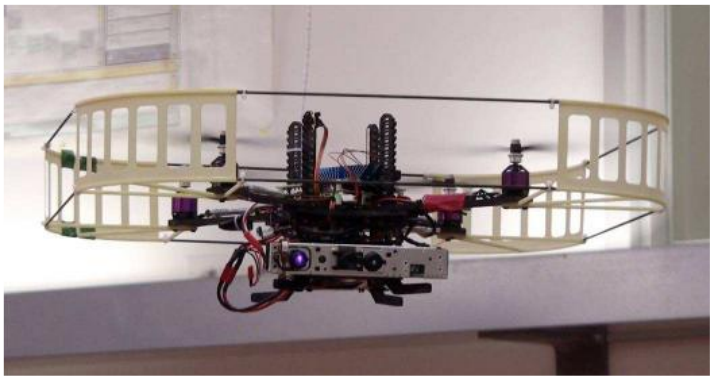


Fig. 6. High-cost MAV with a RGB-D camera mounted on its base (Bachrach et al., 2012).

Although the system is a SLAM solution for indoor environments for MAVs, there is a huge difference between this work and the one proposed in this thesis: the hardware architecture. The drone put to use can lift a RGB-D camera, which weighs typically more than a kilogram. Furthermore, although the position control and the

relative localization estimation is made offboard –that means, performed by a ground station– the SLAM is computed inside the drone, that is, performed by the onboard processor. Both features make the MAV a high-cost platform, which is the opposite of the idea suggested in this thesis. On the other hand, the fusion of a visual sensor with measurements from the IMU makes this work a close relative and a source of ideas for the system developed in this thesis.

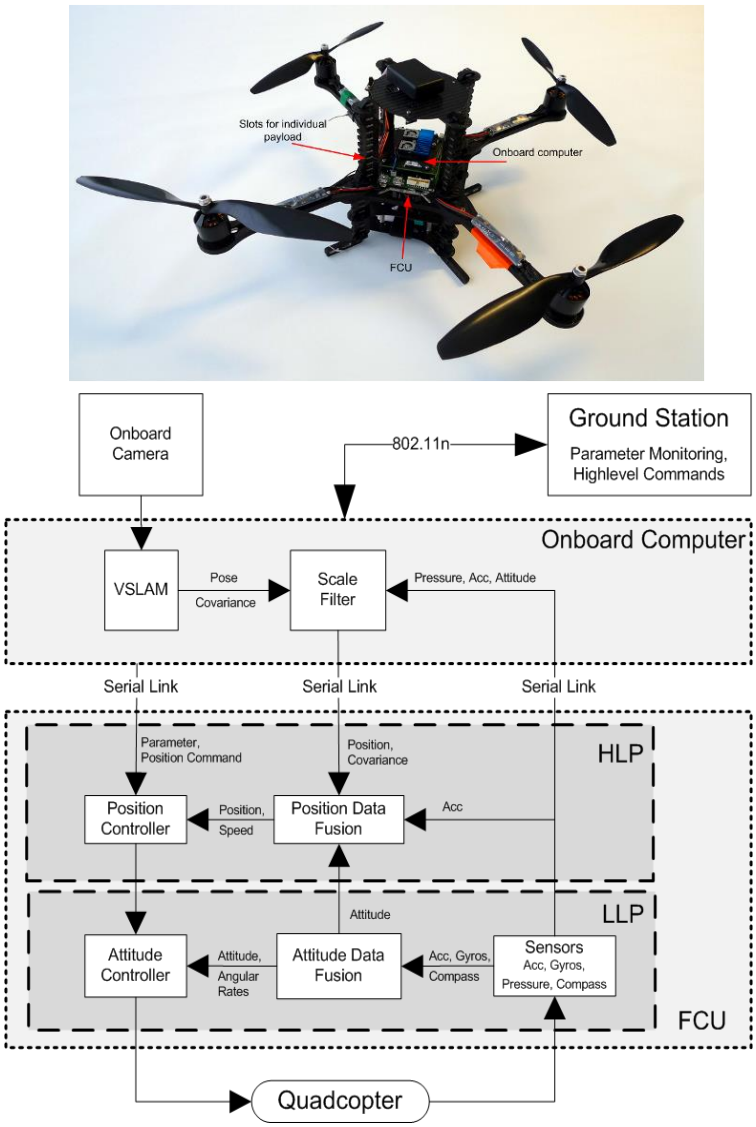


Fig. 7. System overview of the work in. (Achteleik et al., 2011).

The research presented in (Achteleik et al., 2011) is closer to the research made for this thesis than the previous ones. The SLAM system fuses information from the monocular camera with measurements from the IMU. The authors give a solution to overcome the issue of having a low frequency onboard visual pose update versus the high agility of an MAV. This is solved by filtering visual information with inputs from inertial sensors, as can be seen in *Fig. 7*. Then, as their system is based on monocular vision, they present a solution to estimate the metric visual scale aid of an air pressure sensor. All computation is running onboard and is tightly integrated on the MAV to avoid jitter and latencies. This framework enables stable flights indoors and outdoors even under windy conditions.

There are two main differences between both works that must be highlighted. As seen in the two previous reviewed works, the authors put to use expensive MAVs in order to reach their goals. In (Galton et al., 2009) and (Bachrach et al., 2012), they were used to lift a heavy load, while for the work described in this section this kind of drone is implemented due to its onboard computer. In (Achteleik et al., 2011) the authors use a Pelican quadrocopter from Ascending Technologies. The other remarkable difference is –as just said– that they perform all the system onboard, while in our case all the computing has to be done by an external ground system due to the limitations of the platform. Therefore, they need a powerful computer on the drone.

Another difference is the visual SLAM technique put to use for this system. The authors chose PTAM (Klein et al., 2007) as the algorithm used for the visual localization and mapping, which is an old and not very reliable technique. In this thesis newer, more efficient and more robust algorithms have been chosen. Furthermore, the method applied in this work in order to calculate the absolute scale for the monocular VSLAM estimations employs an onboard pressure sensor. The measurements from this sensor are very noisy and prone to drift due to changing weather conditions –the usage of this sensor was considerate for our work but was declined after several tests due to these problems–. Therefore, the authors had to design an EKF in order to fuse all data from different sensors and incorporate the scale and pressure sensor drift in the states. Thanks to it, they could achieve good results despite the noisy sensors.

Finally, in (Engel, 2011) the authors developed a system that enables a quadrocopter to localize and navigate autonomously in previously unknown and GPS-denied environments. This approach uses a monocular camera onboard the quadrocopter and does not require artificial markers or external sensors.

Their approach consists of three main components, as it can be seen in *Fig. 8*. Firstly, the authors used a monocular, keyframe-based simultaneous localization and mapping (SLAM) system for pose estimation. Secondly, they implemented an extended Kalman filter, which includes a full model of the drone's flight and control dynamics to fuse and synchronize all available data and to compensate for delays arising from the communication process and the computations required. Finally, they used a PID controller to control the position and orientation of the drone.

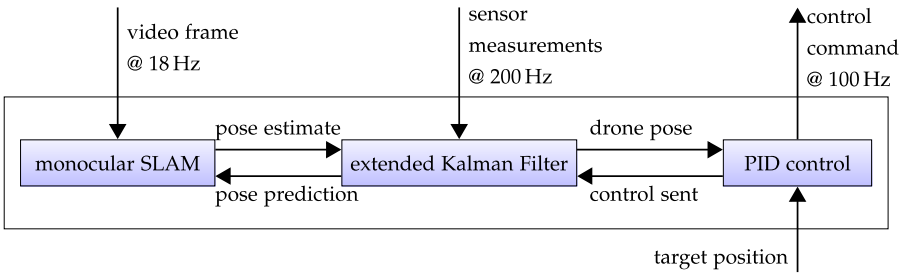


Fig. 8. Architecture of the system proposed in (Engel, 2011).

Furthermore, the authors proposed a method to estimate the absolute scale of the generated visual map from inertial and altitude measurements, which is based on a statistical formulation of the problem. Following a maximum likelihood (ML) approach, they derive a closed-form solution for the ML estimator of the scale.

The authors implemented their approach on a real robot and extensively tested and evaluated it in different real-world environments. As hardware platform they used the Parrot AR.Drone; demonstrating what can be achieved with modern, low-cost and commercially available hardware platforms as tool for robotics research. In their approach, all computations are performed on a ground station, which is connected to the drone via wireless LAN.

This work is the main reference for the work developed in this thesis. It makes this research the closest one in the investigation concerning. Due to its implementation in low-cost MAVs, they were not able to carry heavy sensors as RGB-D cameras and/or range laser scanners. It forced them to use the available sensors in most of commercial drones –monocular cameras and the IMU–. Although there are several similarities between this work and this thesis, there are some differences too. The most important contributions of this thesis are:

- Platform: While the platform used in (Engel, 2011) is the AR.Drone[1] the one chosen for this thesis is the Bebop drone[2]. The main improvements of the Bebop are its higher stability and lower dimensions. The flight of the Bebop drone is steadier than the one of the AR.Drone, something crucial when working with visual SLAM. Furthermore, a stable flight can avoid crashes when working with these aerial robots that have such fast dynamics. Other important improvement is the performance of the camera. Not only the field of view (FOV) has been increased but the resolution of the video. It allows the VSLAM algorithm to estimate the drone's position in a better way without pre-processing the image –with the delay that it implies–.

a) b)

Fig. 9. Bebop drone (a) vs. AR.Drone 2.0 (b) of Parrot.

- Visual SLAM technique: The VSLAM algorithm chosen in (Engel, 2011) is PTAM (Klein et al., 2007). This is an out-dated algorithm that causes drifts and errors when applied to fast platforms such as aerial robots. One of the initial objectives of this thesis was to study the ability of different up-to-date visual SLAM methods to be applied in aerial robots, and so this is one of the items in which an effort has been done.

[1] http://www.parrot.com/es/productos/ardrone-2/
[2] http://www.parrot.com/products/bebop-drone/

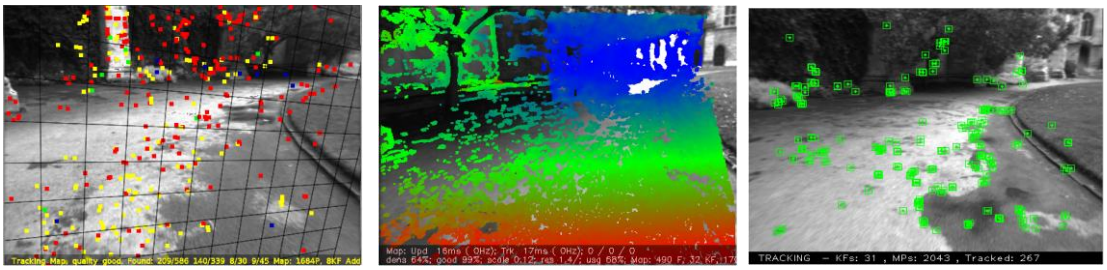
The next section briefly reviews recent visual SLAM techniques in order to choose the ones to be studied and applied in the SLAM system proposed in this thesis.

2.3. Visual SLAM techniques

There are different sensors that can be implemented in a Visual SLAM system. While some of them are typically implemented in ground robots due to its weight, there are others that can be used in MAVs. As seen before, there are some research or professional-oriented MAVs that can also carry a heavy load and therefore the kind of sensors usually included in ground-robot systems, but this is not the scope of this project. These drones are able to lift such sensors as RGB-D or stereo cameras systems that are able to return not only a video stream but also a depth map of the environment. It simplifies the implementation of a VSLAM system and making it autonomous. However, in light and low-cost MAVs the best solution is to use the included monocular camera.

In order to perform the simultaneous localization and mapping of the environment by means of visual information, the process of visual odometry (VO) must be accomplished. Visual odometry is the process of determining the position and orientation of a robot by analysing the associated camera images, thus, to estimate the 6DOF position of the MAV. The VO approaches can be classified into two main categories based on the number of cameras adopted: monocular and stereo VO methods. A stereo pair is applied as minimum number configuration of cameras for solving scale ambiguity problem –as will be explained in Chapter 5- in order to carry on the stereo visual odometry (Brand et al., 2014). However, stereo camera systems are not the focus of this work but the monocular ones.

In the literature, (Klein et al., 2007) has proposed the most representative monocular keyframe-based tracking and mapping system, PTAM (Parallel Tracking And Mapping), for real time pose estimation applications (*Fig. 10.a*). In (Forster et al., 2014) a semi-direct monocular visual odometry algorithm is also presented, i.e. SVO (Semi-direct Visual Odometry). This algorithm can be implemented on an onboard embedded computer –in the case of the paper, in an Odroid U2– which runs at 55 FPS and outputs a sparse 3D reconstructed environment model. In (Newcombe et al., 2011), the work DTAM (Dense Tracking And Mapping), a real-time probabilistic monocular pose estimation method for 3D dense environment reconstruction is proposed. In (Engel et al., 2014 a) the authors describe a direct monocular simultaneous localization and mapping algorithm for building consistent, semi-dense reconstructions of the environments, the LSD-SLAM method (*Fig. 10.b*). Finally, in (Mur-Artal et al., 2015) a keyframe-based monocular SLAM system with ORB features that can estimate the 6DOF pose and reconstruct a sparse environment model is presented (ORB-SLAM – *Fig. 10.c*).



(a) PTAM (b) LSD-SLAM (c) ORB-SLAM

Fig. 10. VSLAM algorithms put to use for this work. Images from (Mur-Artal and Tardós et al., 2015).

The two last algorithms –LSD-SLAM and ORB-SLAM– are two of the best VSLAM methods due to their robustness and performance. However, these recent methods have not been applied to aerial robots yet. A detailed description and comparison of the three methods shown in *Fig. 10* –PTAM, LSD-SLAM and ORB-SLAM– is shown in Chapter 5. Also, its application to aerial navigation is explored in this thesis.

CHAPTER 3: HYPOTHESIS AND METHODOLOGY

After showing the problem statement of this work and performing a background research about related projects, we are in the position to formulate the hypothesis that will be developed in this thesis, and to set a series of specific objectives from the initial statement and the methodology to reach them.

3.1. Hypothesis formulation

After the study of the state of the art in MAVs autonomous navigation, it has been found that one of the main problems when developing reliable SLAM systems is the payload limitation, which restricts the kind and number of sensors to be used. For indoor applications, where the size of the drone has to be small and GPS signal is not available, this problem is particularly hard. In these situations it is required to use only light onboard sensors such as monocular cameras or inertial measurement units, but developing robust SLAM systems with this constraints is still a research challenge.

The hypothesis of this work is that the application of recent monocular VSLAM techniques to aerial robots is possible by fusing the results with other onboard sensors in order to solve the scale ambiguity problem and to improve the results of the position and map estimation. It is intended to demonstrate this hypothesis on commercial low-cost drones, whose computational onbard power is very limited. For this reason the SLAM system will be executed in a ground control unit, taking into account the delays of the wireless link in the control loop.

3.2. Method for testing the hypothesis: specific goals

In order to achieve a conclusion and to validate the hypothesis some objectives are needed to overcome. These objectives and the methodology to achieve them are the following:

- Choose a robot development environment that facilitates the integration of the necessary codes. Robot Operating System (ROS)[3] is a very popular platform today, and one of the most widespread in the research field, why it has been chosen for this work. It allows to create distributed network systems and provides the services expected from an operating system, including hardware abstraction, low-level device control, implementation of commonly-used functionality, message-passing between processes and package management.

[3] http://www.ros.org/

- Perform a research in the area of hardware platforms for drones and its available drivers for ROS. Due to their small size, light weight, low cost and to the possibility of controlling them from a remote station using ROS drivers, two platforms of Parrot will be studied: the AR.Drone and the Bebop drone.

- Perform a research of the available monocular VSLAM algorithms for ROS and select the best ones. As it has been seen in the state of the art, PTAM is a classical monocular VSLAM method that has already been applied to drones. It is available as a ROS package. The two recent methods LSD-SLAM and ORB-SLAM are both available as ROS packages too, but they have not been applied neither compared in aerial robots. The three methods will be tested and adjusted using the cameras of the AR.Drone and Bebop drones.

- Study the scale ambiguity problem of the last algorithms and provide a solution that, using other onboard sensors (such as IMU or sonar), will get the real scale of the obtained map.

- Develop an Extended Kalman Filter (EKF) to fuse the VSLAM and onboard sensors measurements in order to improve the estimation of the 6DOF pose of the drone and the local map. To do this, movement and observation models will have to be studied for the drone and its sensors.

- Develop a PID controller that, using the estimated pose of the global SLAM system (output of the EKF), allows the drone to reach position goals.

- Develop a ground-truth system that allows us to validate the estimated pose. A typical ground-truth system for aerial robots, due to their fast dynamics, is a motion capture system. However, as this is not available, a simplification will be designed based on a camera on the ceiling, which will permit to measure some of the variables of the system using an external reference.

- Perform experiments in order to collect enough data to analyse and validate our proposal, calculating the errors and adjusting variables –as the coefficients of the PID controller, the working period of the system, the added Gaussian noise, etc.– to obtain an optimal response of the SLAM system

- Present reviewed results and a conclusion about the hypothesis.

CHAPTER 4: SYSTEM OVERVIEW

In this thesis, the problem of autonomous indoor MAV localization is addressed as a software challenge, focusing on high-level algorithms integration rather than specific hardware. For this reason, a low-cost commercial platform with minor modifications and an open-source development platform (ROS) are used, so that drivers of sensors and some algorithms can be used without development.

Through this chapter an overview of the whole system is presented, starting with the hardware architecture and the reasons of why the platform put to use was chosen. Some hardware specifications are shown too. Next to it, the software architecture is explained. As said before, the objective is to develop a software system that could perform a SLAM addressing the MAV as a black box. Thus, in this project the most important side of the architecture is the software.

4.1. Hardware Architecture

The quadrotor MAV used for this work –shown in *Fig. 11*– is the Bebop from Parrot, a lighter (400 gr) and smaller (33x38x3.6cm) drone than the earlier AR.Drone 2.0. The last was also put to use for the performance, but due to its lower flight stability its usage was declined. Bebop MAV can carry up to 200g of payload for about 5 minutes and it is equipped with a frontal "Fisheye" camera. It has another vertical camera, which is used for stabilization and horizontal velocity estimation. Besides, it has an ultrasonic altimeter, a 3-axis accelerometer, 2 gyroscopes and a barometer. It incorporates an onboard controller 8 times more powerful than the one from the AR.Drone 2.0 (dual-core processor Parrot P7), a quad-core graphic processor, flash memory of 8Gb and a Linux distribution. It is controlled via Wi-Fi –it provides its own network– and a SDK is available for application development.

Fig. 11. Bebop Drone from Parrot

This model of drone was chosen between all the low-cost commercial models of MAVs because of its steadiness, something crucial when flying these robots in indoor environments with a big amount of obstacles. Furthermore, a driver for ROS was already developed, as it will be explained in the next section.

Although the Bebop comes with some software for basic functionality, it is neither open-source nor easy to modify, and so it is treated as a black box, using only the available W-LAN communication channels to access and control it from a remote station, in this case a laptop, as it is shown in *Fig. 12*.

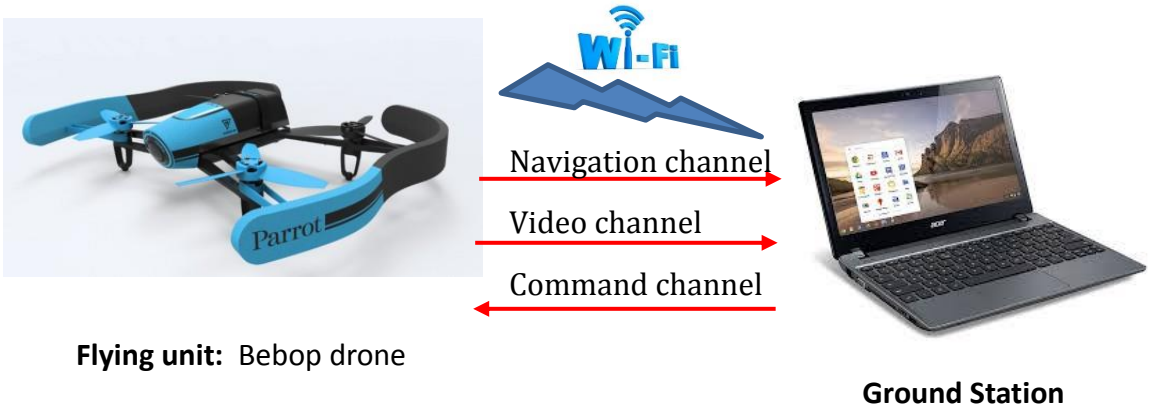


Fig. 12. Hardware and communications architecture

Specifically, these are the inputs/outputs used by the SLAM system that will be executed in the ground station:

- Video channel, to receive the video stream of the forwards facing camera, with maximal supported resolution of 640x368 and frame rate of 30fps.

- Navigation channel, to read onboard sensor measurements every 5ms. The data used by the system are:

  1. Drone orientation as roll, pitch and yaw angles ($\bar{\Phi}, \bar{\Theta}, \bar{\Psi}$).
  2. Horizontal velocity in drone's coordinate system ($\overline{vdx}, \overline{vdy}$), calculated onboard by an optical-flow based motion estimation algorithm.
  3. Drone height $\bar{h}$, obtained from the ultrasound altimeter measurements.

- Command channel, to send the drone control packages, with the desired velocities of x and y axis (in world coordinates); vertical speed and yaw rotational velocity:

$$u = (\widehat{vx}, \widehat{vy}, \widehat{vz}, \hat{\Psi}) \quad (1)$$

4.2. Software Architecture

For the development of the software architecture, ROS meta-operating system was put to use –the whole project has been developed for ROS Indigo on Ubuntu 14.04–. ROS implements *packages* in order to perform different applications for robotics. These *packages* contain *nodes*, which could be programmed in C++ or Python. The *nodes* achieve specific tasks for the whole *package*. The *nodes* are communicated by means of *topics* and *messages*. In this work *topics* are mostly used, and represent a channel of information where different nodes could read and/or write.

The SLAM system explained in this work consists of three major components: (a) a monocular VSLAM system that obtains a 6DOF pose estimation (and a 3D map of the environment); (b) an Extended Kalman Filter that fuses the last estimation with the navigation data provided by the onboard sensors of the MAV to obtain a robust 6DOF estimation of the position of the robot in the generated map; and (c) a PID controller that allows the MAV to reach goal poses using the estimated position. All of these components will be deeply explained in their corresponding chapters. In the following the implementation in ROS is explained.

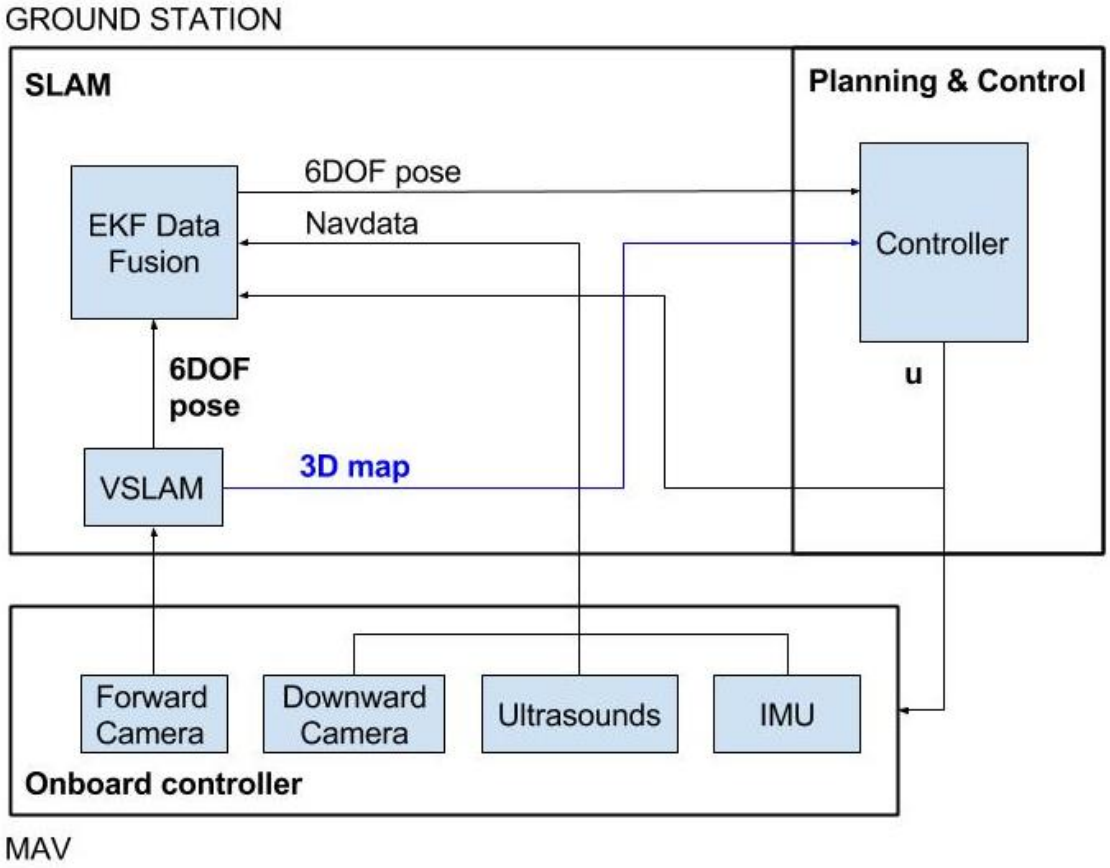


Fig. 13. Software architecture

As seen in the previous figure, all the computing is performed in the ground station. In *Fig. 14* we show the ROS-based implementation of the system. The drone's ROS driver (*bebop_autonomy)* reads the information obtained by the onboard sensors in order to compute the estimation and motion control. The forward camera brings the video stream needed for executing the VSLAM. The downward facing camera allows us to read the horizontal velocities –using an onboard implemented algorithm (Bristeau et

al., 2011) –; the ultrasound sensors inform about the distance between the floor and the drone and the IMU brings us direct measurements from gyroscopes and accelerometers. These three last sensors –grouped in the channel $h_{NAVDATA}$– allow the system to perform the data fusion by means of the EKF. Then, knowing the current estimation of the position and a goal, the PID controller calculates the command **u** and sends it to the drone through the drone's driver.

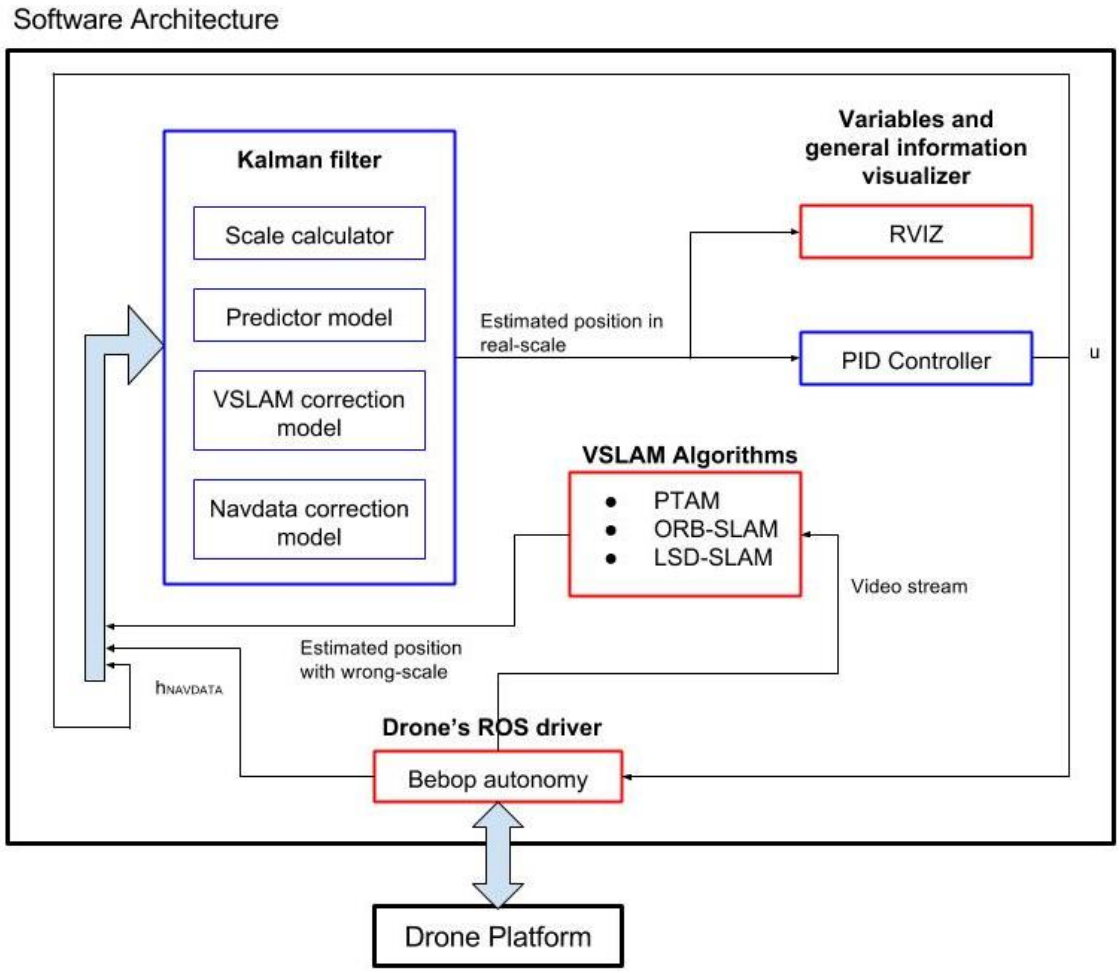


Fig. 14. Software architecture overview.

In *Fig. 14*, blue blocks represent the packages developed for this thesis, while the red ones are previously programmed packages available for ROS. Thus, the drivers for both of the drones put to use were previously developed –something kept in mind when the drone's models were chosen– so were the compared monocular VSLAM algorithms. Regarding to this, the work accomplished for this thesis was to learn how to use these packages and to implement them in our system. The tool Rviz –a 3D visualizer developed for ROS– is also implemented in ROS and is used to visualize results and debugging the code. *Fig. 15* shows a screenshot of Rviz during an execution of the SLAM system, where the trajectory followed by the drone and the obtained map are shown in red and black respectively. It is possible to display also a video stream, the position estimated by a laser or a topic of type *Odometry*. This last kind of marker is employed to display the estimation of the system by means of the node *odom_publisher* –also included in the package of the EKF–. Furthermore, the node *robot_tf_publisher* replaces the estimated position of the drone from the location of the forwards facing camera to the drone's centre –which is around 105mm behind in the X axis–. It allows

the system to estimate the drone's position from the correct frame and helps when comparing the recorded results with the ground truth.

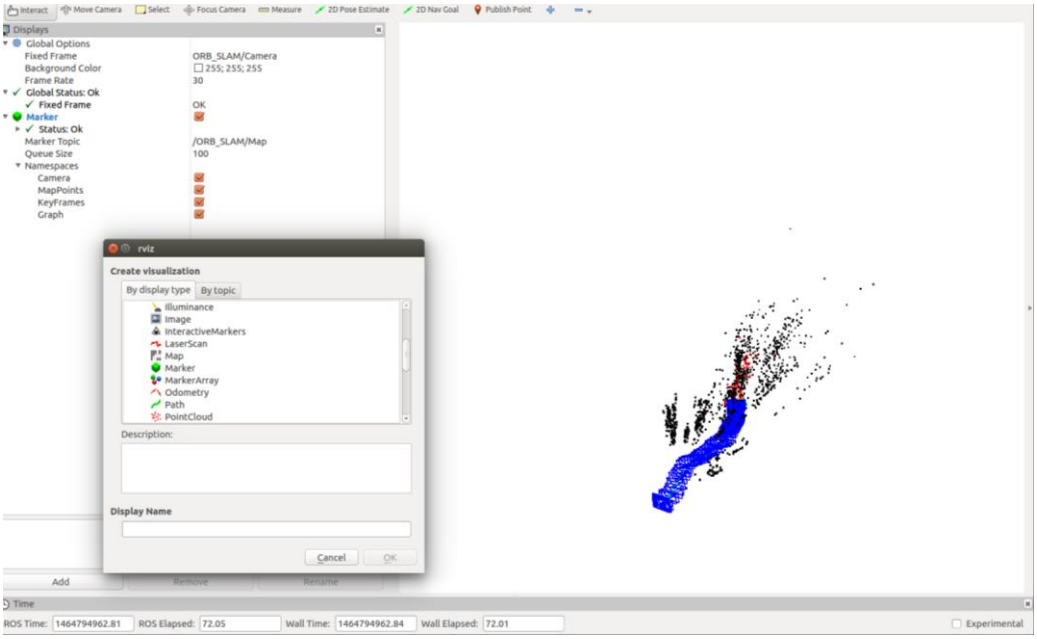


Fig. 15. Rviz performance.

On the other hand, two packages were specifically developed for this thesis: the EKF and the PID Controller. Both of them will be deeply explained in their respective chapters.

Another node was developed in order to read the estimated pose by ORB-SLAM so it can be recorded. It helped us to obtain the required information for the comparison performed between both VSLAM methods by means of a benchmark –explained in section 5.3.4–. This node is in charge of reading the pose estimated by ORB-SLAM between the data published by the array in /tf –it is possible by detecting the data whose parent and child frames are ORB_SLAM/World and ORB_SLAM_Camera respectively–. Then, the node assigns a timestamp value (obtained from the /camera/rgb/camera_info topic given in a bagfile provided by the benchmark) to each of the estimated poses. Finally, it publishes the estimation with an assigned timestamp.

The whole software was developed so it could be launched using both drones –AR.Drone and Bebop from Parrot, shown in *Fig. 9*–. It detects which drone is being used so it adapts the performance to it.

CHAPTER 5: MONOCULAR VISUAL SLAM

5.1. Introduction

SLAM is defined as "the computational problem of constructing or updating a map of an unknown environment while simultaneously keeping track of an agent's location within it". Monocular VSLAM techniques are a kind of SLAM that uses a monocular camera in order to construct that map while tracking the agent's location.

This method was chosen because monocular cameras are included in most commercial drones, so it could be launched in most of them. Another reason for choosing this method is the power consumption and the weight of other sensors that consume and weight much more than an embedded monocular camera. However, the utilization of VSLAM techniques comes with problems that do not appear when using other visual sensors –such as RGB-D cameras or stereo camera systems– or even laser:

- Need of movement in order to build the map and track the agent's position.
- Scale ambiguity.
- Weakness against pure rotational and/or fast movements.

Other camera sensors measure the depth of an image in different ways. For example, a ToF (Time of Flight) camera –as the RGB-D model– calculates the distance between the camera and the object using the speed of light. A RGB-D camera as the one in *Fig. 16* implements a RGB colour camera, an IR (infrared) emitter and a receptor. While the RGB-D camera captures each frame, the IR emitter sends a pattern of light that bounds on the object and is received by the IR receptor. Using the measured passed time between the emission and reception of the IR the depth of the image is reconstructed. Also the deformation of the received pattern is computed, in order to reconstruct the relief of the image.

Fig. 16. Kinect camera by Microsoft, a RGB-D camera model

On the contrary, a monocular camera system needs the camera to move in order to perform the SLAM. Monocular systems cannot compare one frame with another without moving the camera –while stereo systems can do just because two or more cameras form it–. This comparison of frames is mandatory in order to perform the SLAM, as the stereo pair is needed to reconstruct the 3D map with 2D images by means of a disparity map.

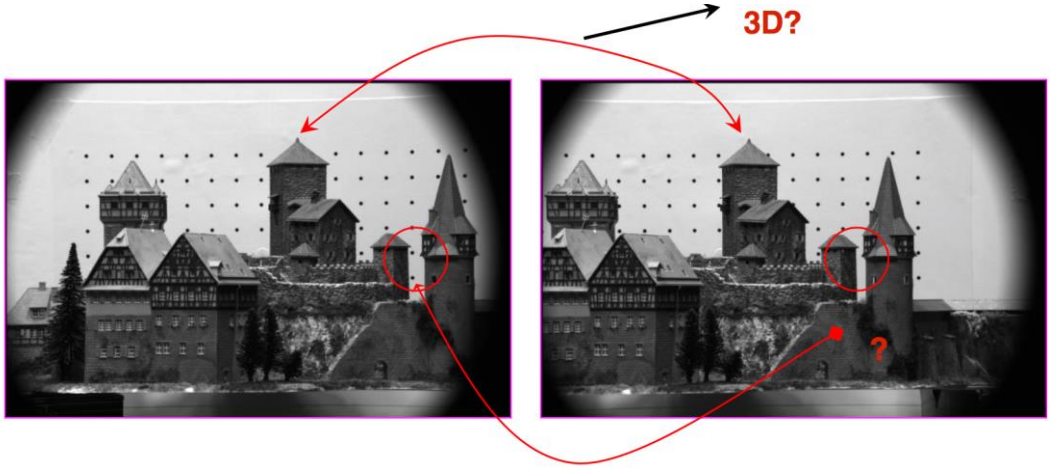


Fig. 17. Stereo pair

In *Fig. 17* an example of a stereo pair can be observed. In both images the same castle is represented, but each of them from a different point of view. The dot pattern at the back helps the system to compare the relative position of each building of the castle. Matching the relative position the keypoints can be extracted and the position of each of them. Furthermore, the depth of the image can be processed, as well as the 3D reconstruction can be performed. This is how the disparity map between two frames is computed. So, in order to reconstruct the 3D map of the camera's environment a couple of frames are needed at least. In this work, as the hardware architecture has only one camera, the MAV should fly around in order to reconstruct its environment. Some systems can calculate the disparity map using a pair of cameras, and they are called "stereo systems". On one hand, these stereo systems can calculate the disparity map by their own. On the other hand, they have an important disadvantage: their FOV –Field Of View– is limited to the field covered by both cameras. It makes stereo systems an inadequate chose if the robot has to cover large environments.

5.2. Scale ambiguity of monocular systems

The main problem that must be faced when working with monocular VSLAM is the ambiguity of the scale issue. The monocular configuration cannot identify the length of translational movement –also known as scale factor– only from feature correspondences. *Fig. 18* represents this issue in a graphical way. The camera does not know the real depth of the object in the image, so it cannot compute its real scale. As seen before, RGB-D cameras calculate the real depth of each frame, so they can compute the real scale thanks to it. Other systems, as the ones formed by binocular cameras –explained before– are capable of this measurement.

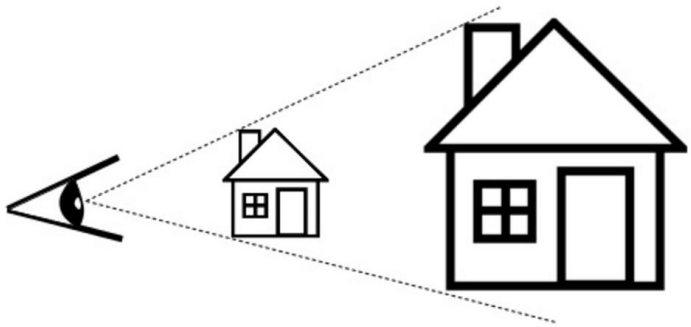


Fig. 18. Scale ambiguity problem

Stereo camera systems can calculate the depth using triangulation of an image. Due to it, they can build a disparity map –as seen before– comparing the video feed from each of the cameras. Once the disparity map has been built, the scale can be directly calculated. As the real distance between both cameras is known, the mathematical process can be easily performed. Firstly, the real depth must be obtained. As seen in *Fig. 19*, the variables needed for the calculation are the focal length, the distance between both cameras and the disparity –the difference between the points of projection in the two cameras–, expressed in pixels.

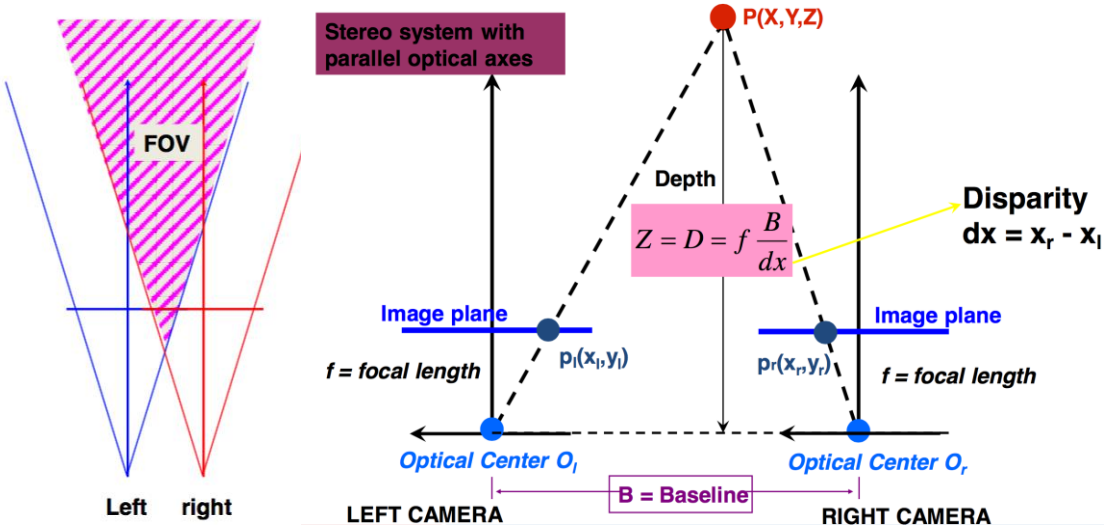


Fig. 19. Principles of stereo vision

As explained before, 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 keypoints. Since the real distance of this camera's movement is not known, the system is not able to calculate the depth. Thus, it is not able to calculate the real scale.

Some approaches have been developed in order to calculate this unknown factor. A few of them measure an object which size is previously known so the 3D scale could be estimated (Tournier et al., 2006). Others employ measurements from the IMU –3D acceleration from the accelerometers, altitude from altimeter and attitude from

gyroscopes– (Engel et al., 2014 b) (Johnson et al., 2008). For this work the altitude is measured with an ultrasonic downward sensor, so the scale can be computed. The idea is that if the real height is known (h_{SONAR}) as well as the estimated by the VSLAM, the scale could be directly calculated as follows:

$$scale = \frac{h_{SONAR}}{h_{VSLAM}} \tag{2}$$

$$x_{REAL-SCALE} = x_{VSLAM} \cdot scale \tag{3}$$

$$y_{REAL-SCALE} = y_{VSLAM} \cdot scale \tag{4}$$

$$z_{REAL-SCALE} = z_{VSLAM} \cdot scale \tag{5}$$

This idea was firstly developed in (Nützi et al., 2011). Our work employs this method to achieve a scale-aware system. The scale is calculated with every iteration of the system, which works at 25Hz.

On the other hand, the problem of VSLAM algorithm's weakness facing pure rotational movements can also be solved by fusing the visual information with the measurements from the IMU. The information obtained from the gyroscopes are robust readings of the drone's orientation. It allows the system to compare the information between both sources –always considering the measurements from the NAVDATA channel more reliable in this matter– and calculate the real orientation. Even if the VSLAM algorithm loses the track, the system will keep estimating the drone's position using the models of prediction and NAVDATA correction until the visual algorithm reengage that track.

5.3. Monocular VSLAM Methods

The following lines describe an overview of the monocular VSLAM algorithms used in this thesis. As an introduction to visual SLAM techniques, two big groups are defined and studied in this work: feature-based methods and direct methods.

- *Feature-Based Methods*. The fundamental idea behind feature-based approaches –both filtering-based and keyframe-based– is to split the overall problem –estimating geometric information from images– into two sequential steps: first, a set of feature observations is extracted from the image. Second, the camera position and scene geometry are computed as a function of these feature observations only.

  While this decoupling simplifies the overall problem, it comes with an important limitation: only information that conforms to the feature type can be used. In particular, when using keypoints, information contained in straight or curved edges –which especially in man-made environments make up a large part of the image– is discarded. Several approaches have been made in the past to remedy this by including edge-based or even region-based features. Yet, since the estimation of the high-dimensional feature space is tedious, they are rarely used in practice. To obtain dense reconstructions, the estimated camera poses can be used to subsequently

reconstruct dense maps, using multiview stereo.

- *Direct Methods.* Direct visual odometry (VO) methods circumvent this limitation by optimizing the geometry directly on the image intensities, which enables the use of all information in the image. In addition to higher accuracy and robustness –in particular in environments with little keypoints– it provides substantially more information about the geometry of the environment, which can be very valuable for robotics or augmented reality applications.

  Direct methods are able to perform dense or semi dense reconstructions of the environment, while the camera is localized so that it directly optimizes over image pixel intensities. These direct approaches do not need to extract features and can avoid the corresponding artefacts, being clearly more robust to blur. In addition their denser reconstructions compared to the sparse point map of Feature-Based methods are more useful for other tasks than just camera localization.

  However, apart from these benefits, direct methods have their own limitations. Firstly, these methods assume a surface reflectance model that in real scenes produces its own artefacts. These methods typically match pixels from a narrow baseline as the reflectance model is violated from wide baseline and many erroneous correspondences would appear. This has a great impact in reconstruction accuracy, which requires wide baseline observations to reduce depth uncertainty. Finally, because direct methods are in general very computationally demanding, the map is just incrementally expanded as in DTAM. Otherwise, map optimization is reduced to a pose graph optimization, discarding all sensor measurements as in LSD-SLAM. In contrast, feature-based methods are able to match features from wide baselines –thanks to their viewpoint invariance– and perform bundle adjustment that jointly optimizes camera poses and points over sensor measurements.

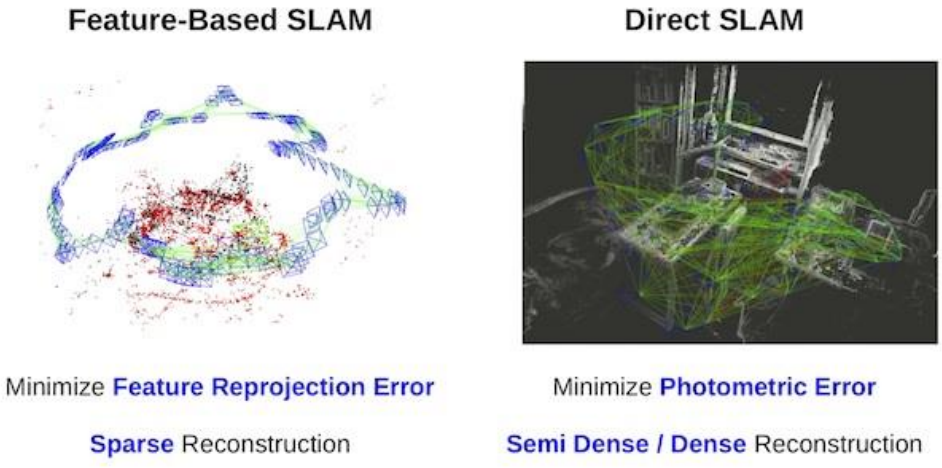


Fig. 20. Types of monocular VSLAM

There are other types of VSLAM algorithms, but the study of them was declined after consideration. Some of them are out-of-date, or simply the performance of Feature-Based and Direct SLAM is better. Others, as for example dense visual SLAM were declined because of other factors. For this method, the performance of the localization and specifically the reconstruction of the environment's map are better, but the computational requirements are too high. High efficiency GPU (Graphics Processor Unit) implementation is mandatory in order to implement Dense SLAM (Newcombe et al., 2011).

One of the biggest problems when working with SLAM techniques is a chicken or the egg causality dilemma. Both Feature-Based and Direct methods build a map using information extracted from the environment, but they also need to estimate the position of the camera related to this map. Algorithms usually divide both processes in two parallel threads so they are executed concurrently. Other methods first build an environment map –as for example, ORB-SLAM and PTAM at the initialization– with enough keypoints so the localization can be performed. Others –as LSD-SLAM– estimate a random depth value at the beginning and it is not recalculated until some keyframes are captured. It leads to initialization errors sometimes. Then, after the initialization both localization and mapping are processed at the same time in a parallel way.

In order to develop the system, a study of the state of the art related with monocular VSLAM algorithms was performed –see Chapter 2–. After that study, LSD-SLAM (Large-Scale Direct Monocular SLAM) and ORB-SLAM (Oriented FAST and Rotated BRIEF SLAM) –both available as ROS packages– were chosen. Both algorithms are up-to-date and are among the highest performance in monocular VSLAM techniques. Each algorithm belongs to one of the two big groups defined before, so it allows this work to make a comparative between the two methods with up-to-date algorithms that represent them.

The study of PTAM technique helped us to understand the basis of VSLAM. This algorithm is quite out-of-date, but as said before it is useful in order to comprehend feature-based monocular SLAM algorithms. Furthermore, an important reference for this work is *tum_ardrone* (Engel., 2011), a ROS package which uses PTAM as the VSLAM technique fused with other measurements from a MAV.

Due to the age and lack of performance of this technique, LSD-SLAM and ORB-SLAM algorithms replace PTAM in this work. This allows the system to be more accurate and reliable than *tum_ardrone* in this respect.

5.3.1. PTAM

PTAM is a reference between Feature-Based methods. The algorithm, developed by Klein and Murray (Klein et al., 2007), was the first work to introduce the idea of splitting camera tracking and mapping in parallel threads. It also demonstrated to be successful for real-time augmented reality applications in small environments.

As other algorithms, it divides tracking and mapping in two separated tasks: one thread deals with the task of robustly tracking erratic hand-held motion, while the other produces a 3D map of point features from previously observed video frames. This

allows the use of computationally expensive batch optimisation techniques as Bundle Adjustment. The result is a system that produces detailed maps with thousands of landmarks which can be tracked at frame-rate.

The first thing that must be completed in order to use the algorithm is the initialization. In contrast to ORB-SLAM, PTAM needs human intervention. When the initialization begins –pushing the spacebar–, the camera should be moved smoothly sideways –avoiding rotational movements– so the system can recognize enough feature points in order to build a map. *Fig. 21* represents a frame captured by the camera while it extracts feature points during initialization. Once there are enough captured points –it must be decided by the user– the spacebar should be pressed again. Then, PTAM starts the simultaneous localization and mapping.

Fig. 21. Initialization of PTAM

Fig 22. and *Fig. 23* display how the PTAM visual interface looks like when it is initialized. In *Fig. 22* the video streaming mode is selected. Feature points are printed on it. The colour of each of this points symbolise the "edge level" of the feature. The sharpest of the features are the red ones, while the blue ones have smoother edges. The interface displays other information, as the number of keypoints, keyframes, etc. A grid is overlaped with the video stream and represents the "initial plane" of the map. It is used as a reference between both interfaces of the system.

Fig. 22. PTAM performance

Fig 23 displays the map of the environment features. With the plain grid used for the *world frame* as a reference the feature points map is built around it. This interface displays the position of the camera based on the environment's map, representing the translation (X, Y, Z) and the rotation (R, P, Y). It displays also this pose translation with scale at the bottom.

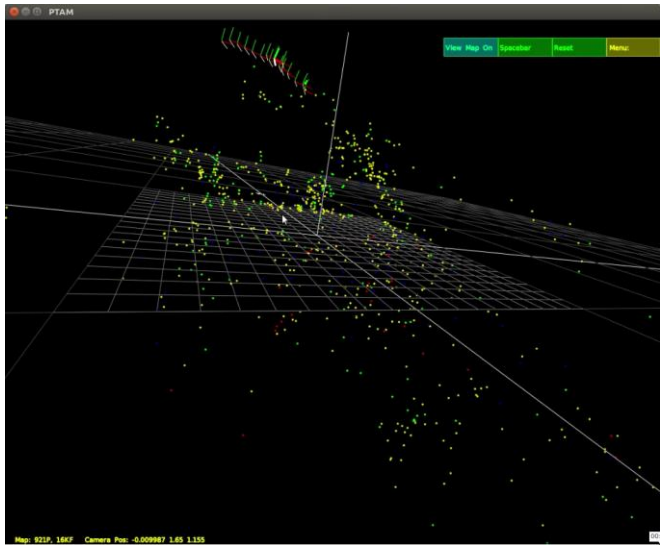


Fig. 23. Map built by PTAM

The map created by this algorithm consists in a big number of points –features located in a world coordinate frame– where each of them represents a locally planar textured patch in the world.

An overview of the technique is displayed in the next block diagram:

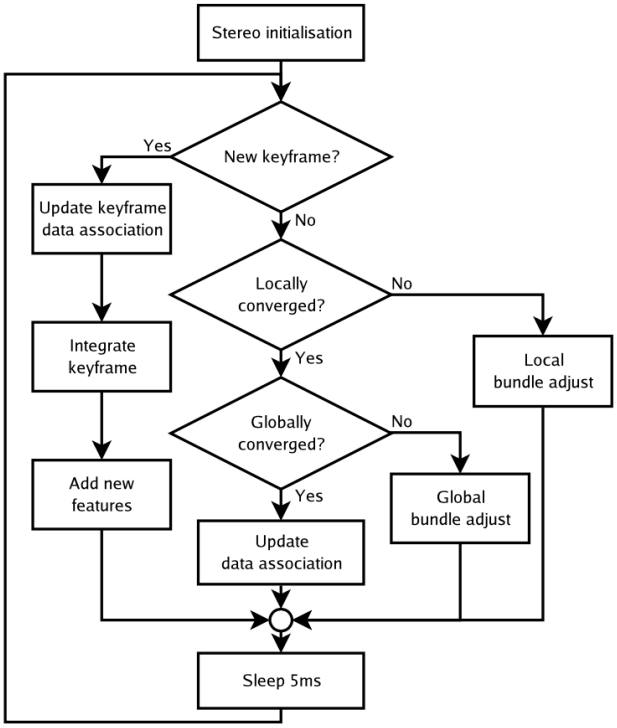


Fig. 24. Overview of the PTAM algorithm

Some other tips can be obtained from (Klein et al., 2007):

- Tracking and Mapping are separated, and run in two parallel threads.
- Mapping is based on keyframes, which are processed using batch techniques (Bundle Adjustment).
- The map is densely intialised from a stereo pair (5-Point Algorithm)
- New points are initialised with an epipolar search.
- Large numbers (thousands) of points are mapped.

In this algorithm the chicken or the egg causality dilemma mentioned before is solved creating a map of the environment firstly –as explained in the initialization– and then starting to estimate the real-time camera's pose relative to this map. As seen in *Fig. 24* the tracking stage could be summarized with the following steps:

1. A new frame is acquired from the camera, and a prior pose estimate is generated from a motion model.
2. Map points are projected into the image according to the frame's prior pose estimate.
3. A small number (50) of the coarsest-scale features are searched for in the image.
4. The camera pose is updated from these coarse matches.
5. A larger number (1000) of points is re-projected and searched for in the image.
6. A final pose estimate for the frame is computed from all the matches found.

PTAM has some important limitations, as the lack of a loop closing mechanism, the low invariance to viewpoint of its relocalization method, the need of human intervention for map initialization, and its restriction to small scenes. Because of it the Feature-Based method used in this work is ORB-SLAM, a method that avoids all of these limitations.

5.3.2. ORB-SLAM

ORB-SLAM is also a feature-based monocular SLAM method. This technique estimates the camera's position in an extremely accurate way. It makes it perfect for be implemented over a system based on a MAV due to its fast and unstable dynamics. Furthermore, ORB-SLAM does not commit into the failures of PTAM explained in the previous section. On the other hand, due to ORB-SLAM is a Featured-based method this algorithm will need a big amount of features in the environment in order to perform the SLAM in a proper way –while LSD-SLAM would need less–. As LSD-SLAM, this algorithm will need information from the environment what will not be available in dark zones.

The performance of the algorithm is explained in *Fig. 25* and *Fig. 26*. The first one represents the map built by the system with a video stream recording the same path followed in *Fig. 30 (below)*. Blue "pyramids" that appear on the image represent the position of the camera when a keyframe is captured and the red one is the current position of the camera in real time. The green line is the path followed by the camera.

The coloured dots are the point features extracted by ORB-SLAM. If they are coloured in black, it means that they are not currently in use, but they are stored in the system. If they are not coloured in black, they have appeared in the latest seconds of the video stream.

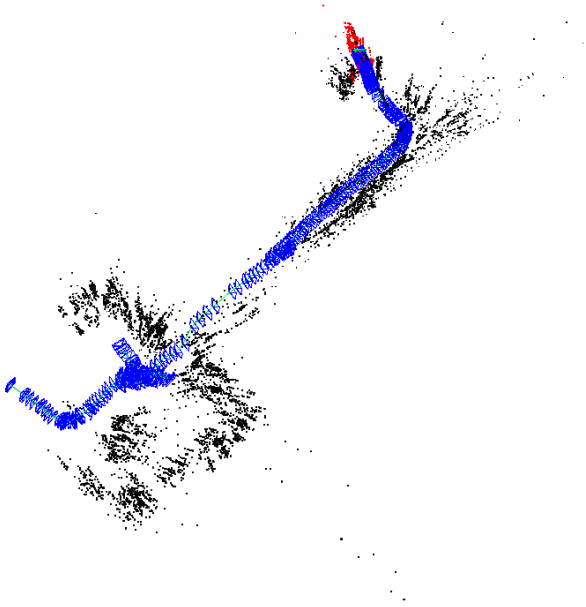


Fig. 25. ORB-SLAM map built indoors

ORB-SLAM is a reliable and robust algorithm but looking at *Fig. 25* it appears to be obvious why the fusion with other sensors is needed. While the tracking goes well at the beginning, when the camera turns left at the corner the algorithm fails –the turn degree was around 90º–. It also incurs in a mistake measuring the length of the corridor after it (it is shortened).

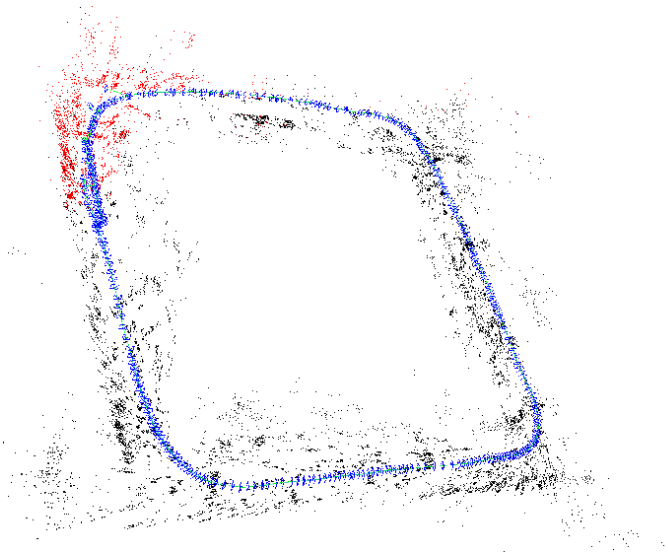


Fig. 26. ORB-SLAM map built outdoors

On the other hand, *Fig. 26* shows the operation of the system outdoors. Although the focus of the work is not the tracking and mapping in this situation, it is important to see the performance when building a big map with thousands of keypoints and lots of

keyframes. The figure represents the track of the camera moving around a square of around $35m^2$ with a loop closing at the end. Thanks to a smart culling of keyframes the system can build maps of a big size without consuming many computational resources. Another remarkable feature is the loop-closing accuracy of the method. The loop closing thread compares on each iteration the last keyframe processed by the local mapping with all of the neighbours and tries to detect and close loops. If the algorithm detects a loop –the keypoints of the last processed keyframe matches with the keypoints of another keyframe stored– it fuses duplicated map points. It also modifies the map and the current position making translation and rotation transforms to achieve the actual position of the camera.

As seen in *Fig. 27*, this system incorporates three threads that run in parallel: the tracking, the local mapping and the loop closing. The tracking thread is always trying to localise the camera in the environment with every frame. ORB-SLAM introduces some features that improve the performance of the algorithm: a constant velocity motion model that roughly predict the new camera pose and then perform an initial matching with the previous frame. If the tracking is lost –e.g. due to occlusions, abrupt movement–, the place recognition module is used to perform a global relocalization. But what makes fast and robust ORB-SLAM is that this relocalization is fully embedded in the tracking procedure, and that the keyframe insertion policy is generous. Being embedded in the tracking thread makes the relocalization faster than if it were a separated thread. Furthermore, because of the generous policy of keyframes insertion the tracking of the camera's pose is really reliable without being worried about the amount of data being stored as the keyframe culling procedure in the local mapping thread will later discard redundant keyframes.

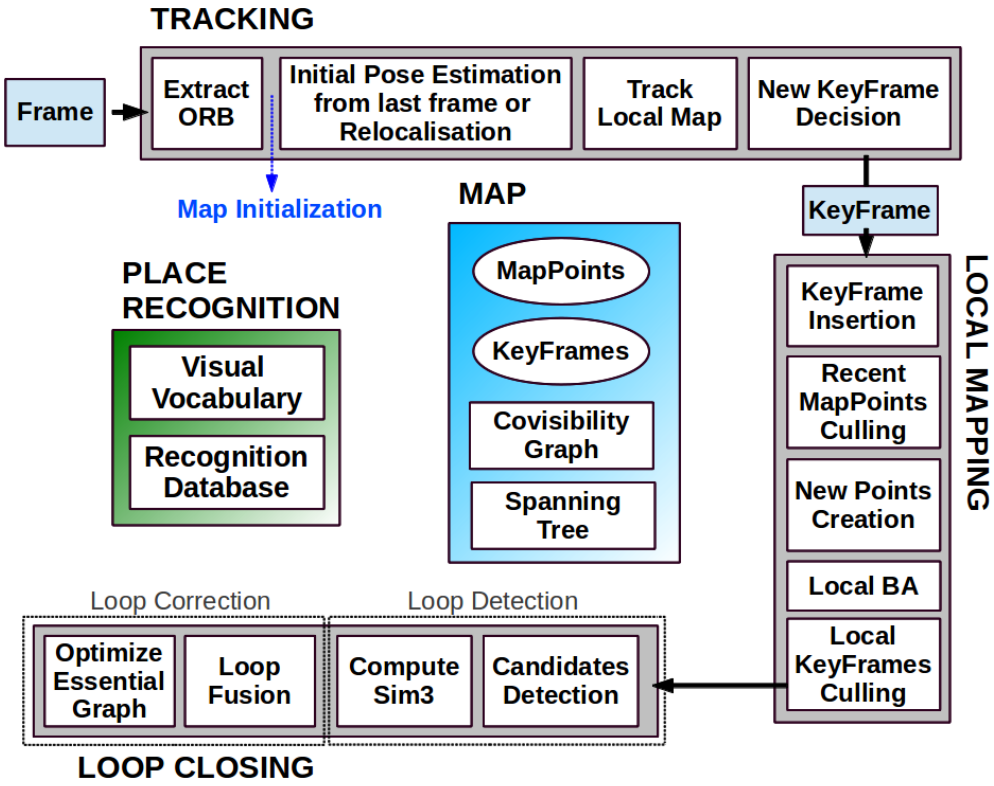


Fig. 27. Overview of the ORB-SLAM algorithm

All data is published and displayed using Rviz. There, the point map can be visualized along with the current position of the camera. The parent and child frames should be selected in order to achieve a correct visualization. The algorithm itself performs translation and rotation transformations between both frames –world and camera frames– which correspond with parent and child frames respectively.

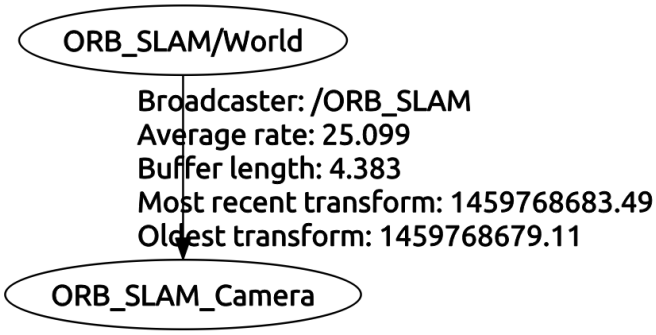


Fig. 28. World-camera frame transformation

5.3.3. LSD-SLAM

LSD-SLAM is a direct (feature-less) monocular SLAM algorithm which, along with highly accurate pose estimation based on direct image alignment, reconstructs the 3D environment in real-time as pose-graph of keyframes with associated semi-dense depth maps.

LSD-SLAM is able to estimate the camera's position and build large-scale semi-dense maps of its environment in real time. In contrast with dense visual SLAM –which could perform a better performance building the map of the environment, but offline due to the computing requirements– this technique allows to observe the environment of the MAV during the flight. However, as said before, the final system –which includes this project– will include a laser SLAM block. This block will bring the system a 2.5D map, so the 3D map built by LSD-SLAM is deprecated. Due to the later implementation of the laser SLAM node and its 2,5D map, only the 6DOF pose estimation of this algorithm is taken as an input to the data fusion filter. Laser's map was chosen instead the one created by LSD-SLAM because of the better accuracy of the first one and due to the computational requirements needed by the last one.

Fig. 29. Video stream and inverse depth map of LSD-SLAM.

Fig. 29 represents the video streaming from the camera and the inverse depth map that LSD-SLAM uses to create the map of the environment. The closest points of the image are represented with green and the farthest in red and black.

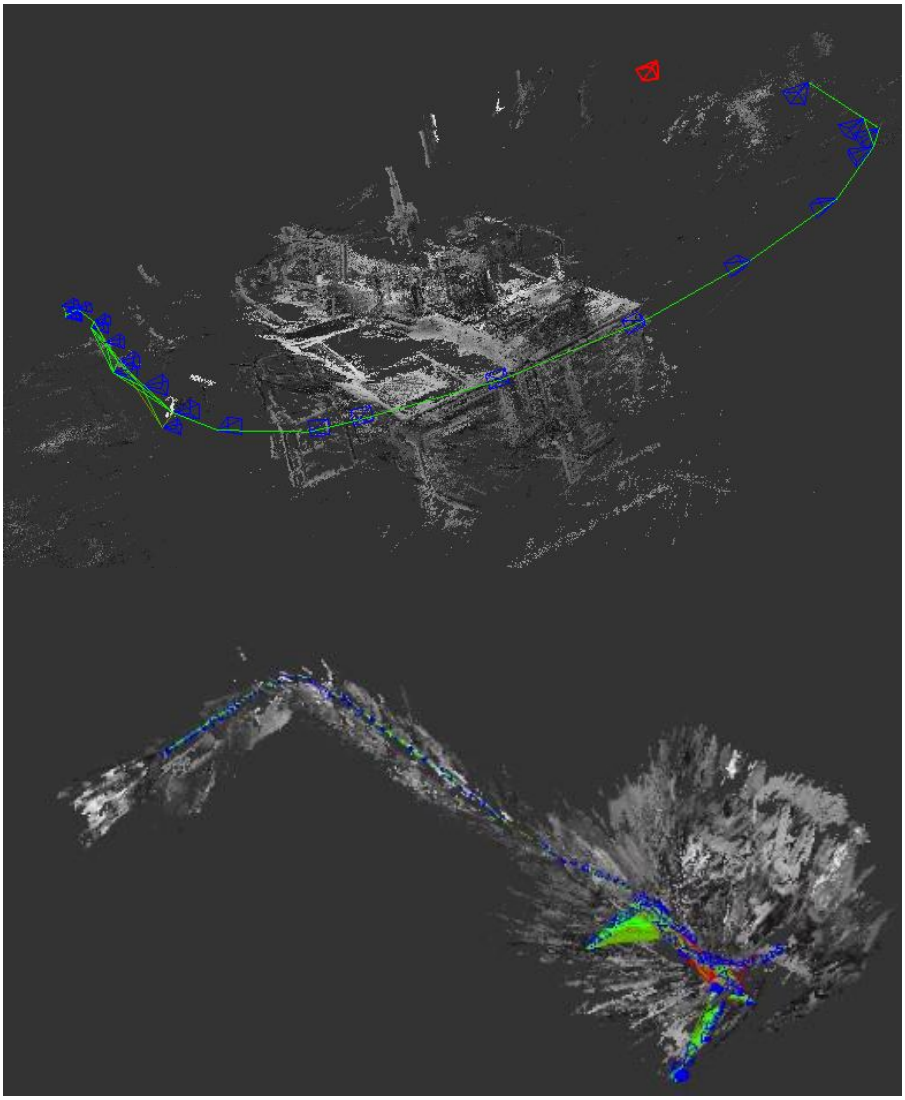


Fig. 30. Results of LSD-SLAM. The first picture represents the translation of MAV's camera around a room. The second one represents the results of the translation around the same room and along two corridors.

In *Fig. 30* the 3D semi-dense map built by the algorithm is shown. The first map represents a desk –a small part of the environment–. The second one displays the path followed by the camera when it was moving around a room and going across a couple of corridors. It proves the capability of the system of mapping big-scaled environments. The size of the environment to be mapped could be even bigger. While results are good in this case, the system needs a high amount of visual characteristics that are not available in dark zones, where it needs to be fused with other sensors. Furthermore, it is very sensitive to pure rotational movement.

In both pictures the green line indicates the track where the camera went over. This track is defined by the 6DOF pose estimated by the algorithm. The blue marks are the camera's poses where the VSLAM algorithm captured a keyframe. As more

keyframes obtaining ratio is defined, more accurate will be the map and the estimate pose, but more computational requirements will be needed. The red marks correspond with the actual pose of the camera. This position is given in real-time. The grey-scale shapes are the 3D objects of the environment mapped by LSD-SLAM. As more keyframes are correctly captured (without drift or depth mistakes), the map of the environment will be better defined. Due to it, the pose estimation will be more robust and reliable.

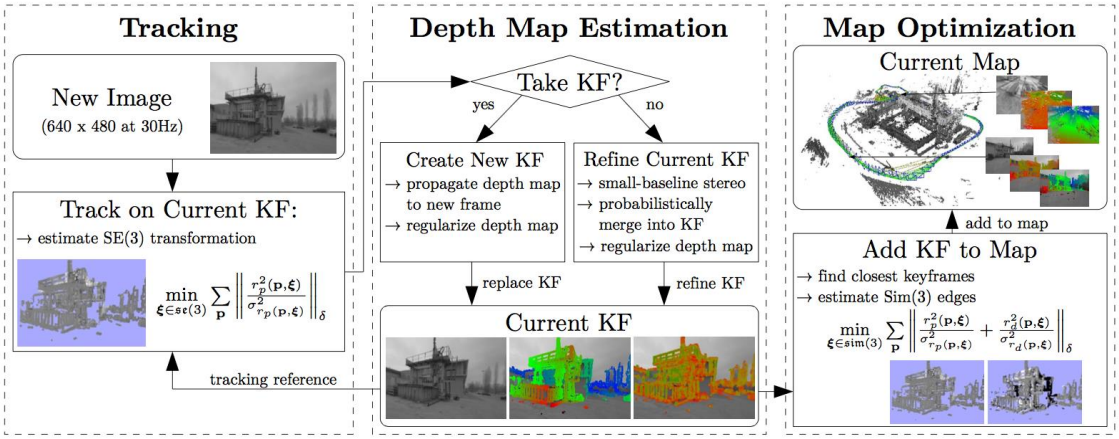


Fig. 31. Overview of the LSD-SLAM algorithm.

An overview of the complete LSD-SLAM algorithm is displayed in *Fig. 31*. The first stage, *Tracking*, involves two steps: capture a new image and track on the current keyframe (KF). When the camera captures a new image the system estimates its current position with respect to the current keyframe –using the pose of the previous frame as initialization–.

In the second stage, *Depth Map Estimation*, the system decides if the new image captured by the camera should become a new KF. Tracked frames that do not become a keyframe are used to refine the current keyframe. The result is incorporated into the existing depth map, thereby refining it and potentially adding new pixels. Once a new frame is chosen to become a keyframe, its depth map is initialized by projecting points from the previous keyframe into it. Finally, it replaces the previous keyframe and is used for tracking subsequent new frames.

During the last stage, *Map Optimization*, the algorithm adds the new KF –the current keyframe that could have been replaced or refined– to the map and tries to optimize it. This stage is responsible of the map's building and adding new details to it.

Even when LSD-SLAM is a Direct method of VSLAM, it does extract and use a small number of keypoints of each keyframe in order to optimize the loop-closuring. This method allows LSD-SLAM not only to use keyframes but also keypoints when trying to look for loop-closures, what makes this algorithm more robust.

5.3.4. Comparison

In order to compare the two VSLAM algorithms put to use for this thesis a benchmark was applied. The chosen benchmark was *RGB-D SLAM Dataset and Benchmark* [4] of the Computer Vision Group from TUM (Technische Universität München). The authors of this benchmark provide some datasets with measurements from sensors –they use a RGB-D camera for the SLAM, but it can be used in the same way for monocular VSLAM methods–. They also give the ground-truth pose of the camera along with the video feed from it. For this experiment, the dataset *rgbd_dataset_freiburg1_xyz* was put to use. This dataset contains a video recorded from a camera that describes smooth and rotation free movements that are perfect for the comparison.

We made the two VSLAM algorithms –LSD-SLAM and ORB-SLAM– to process the recorded video before mentioned five times each. The estimated pose of each of these processes is compared with the ground-truth by means of the online tool provided by the authors of the benchmark[5] and its results recorded. Then, a median of the five recorded values of each field given back by the online tool is performed. The results are presented in *Table 1*. As said before, the monocular VSLAM algorithms cannot calculate the real scale of its estimations. Due to it, the estimations extracted from the dataset of each algorithm were pre-processed. Thanks to it the real-scale was calculated with a Matlab script and added as an argument in the online tool.

	LSD-SLAM	ORB-SLAM
Compared pose pairs	782	283
Absolute translational error (RMSE)	0.0609	0.049
Absolute translational error (mean)	0.0474	0.0310
Absolute translational error (median)	0.0357	0.0185
Absolute translational error (std)	0.0382	0.0361
Absolute translational error (min)	0.033	0.0023
Absolute translational error (max)	0.2792	0.2608

Table 1. Comparison between both monocular VSLAM algorithms. The results are given in meters.

The calculations given back by the online tool are:

- The compared pose pairs. Each of the methods has a different number of compared pairs due to not all of the estimated poses are compared. It happened because of the assigned timestamp to each of the measurements. The time of each estimation must match with the timestamp of the given by the ground-truth. Otherwise, the comparison and therefore the error calculation cannot be performed. Furthermore, each of the methods has its own period time. For instance, the results from LSD-SLAM were just recorded from the topic

[4] http://vision.in.tum.de/data/datasets/rgbd-dataset
[5] http://vision.in.tum.de/data/datasets/rgbd-dataset/online_evaluation

/lsd_slam/pose –the topic provided by the method–. Due to it, new poses are recorded each time the algorithm detects a new frame –even if it is not a keyframe–. On the other hand, as the pose estimation by ORB-SLAM is published in the topic /tf as an array a reader and publisher node had to be developed.

- The following fields correspond with different ways of expressing the absolute translational error between pose pairs. The first one is the Root-mean-square deviation (RMSE). This method is the most used when trying to express the error between an estimator and real values.
- The third field corresponds with the mean of the vector of errors between pairs computed by the benchmark.
- The next field is the same as the previous but using a median instead of the mean.
- This field displays the standard error or deviation (std) of the recorded value of errors.
- The two last fields express the minimum and maximum error calculated by the benchmark respectively.

According to these results, the performance of ORB-SLAM is slightly better than the one of LSD-SLAM. However, the video from the used dataset has many features to extract, which benefits a Featured-Based method as ORB-SLAM. On the other hand, the initialization from LSD-SLAM could be better –if it does not initialize in a bad way due to its random values given to the estimated depth at the beginning– because ORB-SLAM needs to build a point's map of the environment before starting the tracking. It leads to the loss of tracking of the first camera's movements. Furthermore, ORB-SLAM needs an amount of features in the environment in order to build a map and performance the SLAM. If the environment does not offer characteristics as corners or sharp edges and consist of soft edges or round-shapes ORB-SLAM will not achieve good results or even could not initialize. On the contrary, LSD-SLAM could face these kinds of environments and perform the SLAM –as explained at the beginning of the Section 5.3–. However, ORB-SLAM is more robust facing pure rotational movements. This conclusion was reached by means of the trial-and-error approach –LSD-SLAM loss the track way more times than ORB-SLAM if the camera suffered pure rotational movement–.

Another parameter to be discussed is the execution time. It is similar for both methods –the median in a certain number of flights is 38.2ms for LSD-SLAM and 35.2ms for ORB-SLAM–. Nevertheless, ORB-SLAM needs a script that could distinguish between the whole array of values that is /tf which are the data corresponding with ORB-SLAM. It means that the sampling period of the ORB-SLAM estimation could be chosen, but not the inner execution time of the algorithm.

Finally, the computer requirements for both algorithms should be compared. Given that both of them could be run without its visualization tool –*lsd_slam_viewer* for LSD-SLAM and *rviz* for ORB-SLAM–, which is the most computational consuming part of the algorithm, the requirements are not perceptible for a CPU. This work was developed to be operated from a ground station, so the computational requirements are not highly important.

CHAPTER 6: DATA FUSION WITH EKF

The developed system of this thesis consists in a quadrocopter, which will send measurements from some sensors to a ground control unit in charge of computing an estimation of the drone position. This estimation of its position, orientation and velocity is called its state. The state of a MAV is dynamic, so it changes with time. Thus, the changing state of our robot must be estimated periodically and this is why the sensors are put to use. The problem of real-world sensors is that they are subject to measurement errors –called noise–. Due to it, state's estimation from a unique sensor will lead to unstable and poor results. However, if these sensors acquire information about the same state they could be fused. Fusing data allows the system to improve its accuracy reducing the effects of noise. Furthermore, it improves the system performance by adding redundancy to it, what leads to a better estimation of the state and more robustness facing challenging situations –that could deny the measurements from one or more sensors, for example–. For this fusion the Kalman Filters (KF) are used.

The Kalman filter is a well-known method to filter and fuse noisy measurements of a dynamic system to get a good estimate of the current state. It assumes that all observed and latent variables have a (multivariate) Gaussian distribution, the measurements are subject to independent, Gaussian noise and the system is linear. Due to the system developed for this thesis is not linear, this kind of filter could not be implemented. On the other hand, the Extended Kalman Filter (EKF) drops the assumption of a non-linear system, making it applicable to a much wider range of real-world problems.

In this thesis, we use an EKF to estimate the state of the drone, fusing visual pose estimates provided by VSLAM algorithms (and corrected with real-scale) with sensor measurements provided by the other onboard sensors. In the following sections, we describe the Kalman filter used. In particular we define the state space as well as the state transition model and the observation models. We also describe how the model parameters are determined.

6.1. The State Space

The state vector of the EKF is defined to be:

$$\boldsymbol{\chi}_t \coloneqq \left(x_t, y_t, z_t, vx_t, vy_t, vz_t, \Phi_t, \Theta_t, \Psi_t, \dot{\Psi}_t\right)^T \in \Re^{10} \tag{6}$$

where (x_t, y_t, z_t) is the position of the MAV in meters (m); (vx_t, vy_t, vz_t) the velocity in meters/second (m/s); $(\Phi_t, \Theta_t, \Psi_t)$ the roll, pitch and yaw angles in radians (rad); and $\left(\dot{\Psi}_t\right)$ the yaw-rotational speed in radians/second (rad/s). All of them are evaluated in world coordinates.

In the following, any variable with a line over it means that this is a measured variable. That is, $\overline{\Psi}$ is the measured yaw angle. Moreover, if a variable has an angle over it will be a variable that belongs to the command vector. So, $\widehat{\dot{\Psi}}$ is the yaw rotational speed that is sent to the drone. If the variable appears without any of the

previous symbols, it represents the estimated value of that variable. On the other hand, for better readability, the time argument is omitted when clear from context.

6.2. The Prediction Model

The prediction model is based on the full motion model of the quadcopter's flight dynamics and reaction to control commands derived in (Engel et al., 2014). A new calibration of the model parameters has been done for the Bebop Drone and for the AR.drone 2.0.

The model establishes that the horizontal acceleration of the MAV is proportional to the horizontal force acting upon the quadcopter, that is, the accelerating force minus the drag force. The drag is proportional to the horizontal velocity of the quadcopter, while the accelerating force is proportional to a projection of the z-axis of the drone onto the horizontal plane, which leads to:

$$\dot{vx} = K_1(K_2(cos\Phi\ sin\Theta\cos\Psi + \sin\Phi\sin\Psi) - vx) \tag{7}$$

$$\dot{vy} = K_1(K_2(cos\Phi\ sin\Theta\sin\Psi - \sin\Phi\cos\Psi) - vy) \tag{8}$$

where K_1 and K_2 are model constants: K_2 defines the maximal speed attained with respect to a given attitude, while K_1 defines how fast the speed adjusts to a changed attitude. The drone is assumed to behave the same in x and y direction.

Besides, the influence of the sent control command $\mathbf{u} = \left(\widehat{vx}, \widehat{vy}, \widehat{vz}, \widehat{\dot{\Psi}}\right)$ is described by the following linear model:

$$\dot{\Phi} = -K_3(K_4\widehat{vy} + \Phi) \tag{9}$$

$$\dot{\Theta} = K_3(K_4\widehat{vx} - \Theta) \tag{10}$$

$$\dot{vz} = K_7(K_8\widehat{vz} - vz) \tag{11}$$

$$\ddot{\Psi} = K_5\left(K_6\widehat{\dot{\Psi}} - \dot{\Psi}\right) \tag{12}$$

where K_3 to K_8 are model constants which are determined experimentally in next subsection. Again, the behaviour of the drone is assumed to be the same with respect to roll and pitch angles.

From equations (7) to (12) the overall state transition function is obtained:

$$\begin{pmatrix} x \\ y \\ z \\ vx \\ vy \\ vz \\ \Phi \\ \Theta \\ \Psi \\ \dot{\Psi} \end{pmatrix}_{t+1} \leftarrow \begin{pmatrix} x \\ y \\ z \\ vx \\ vy \\ vz \\ \Phi \\ \Theta \\ \Psi \\ \dot{\Psi} \end{pmatrix}_{t} + \Delta_t \begin{pmatrix} vx \\ vy \\ vz \\ K_1(K_2(cos\Phi\ sin\Theta\cos\Psi + \sin\Phi\sin\Psi) - vx) \\ K_1(K_2(cos\Phi\ sin\Theta\sin\Psi - \sin\Phi\cos\Psi) - vy) \\ K_7(K_8\widehat{vz} - vz) \\ -K_3(K_4\widehat{vy} + \Phi) \\ K_3(K_4\widehat{vx} - \Theta) \\ \dot{\Psi} \\ K_5\left(K_6\widehat{\dot{\Psi}} - \dot{\Psi}\right) \end{pmatrix} \tag{13}$$

6.2.1. Calibration of model parameters

The proportional coefficients K_1 to K_8 were estimated from data collected in a series of test flights. The coefficients are calculated by pairs, which are related in the state equations –as seen in (13)–. For instance, K_7 and K_8 appear in the state equation (11):

$$\dot{vz} = K_7(K_8\widehat{vz} - vz)$$

This equation characterizes the evolution of the drone's vertical velocity when a command of speed in the z-axis is applied to it. If the previous equation is transformed to the Laplace domain we get the following:

$$\dot{vz} = K_7(K_8\widehat{vz} - vz) \rightarrow s \cdot \mathrm{Vz(s)} = K_7\big(K_8 Vz(s) - Vz(s)\big) \tag{14}$$

$$\frac{Vz(s)}{\widehat{V}z(s)} = \frac{K_8}{\left(\frac{1}{K_7}\right)s+1} \tag{15}$$

If the transfer function of a first order system is known:

$$F(s) = \frac{k}{\tau s+1} \tag{16}$$

The parameters are obtained just by matching:

$$\tau = \frac{1}{K_7} \tag{17}$$

$$k = K_8 \tag{18}$$

Thus, K_8 represents the static gain between the vertical velocity sent to the drone –which was normalised between ± 1– and the actual reached vertical velocity. K_7 is the inverse of the time constant of this evolution.

Following the previous example, how was the data collected in order to obtain the coefficients K_7 and K_8 is now explained. We sent a unit step as a command in vz to the drone, which correspond to the maximum vertical speed between the normalized values. While the drone was flying up –the vertical speed was increased–, reached a goal height and landed –the vertical speed was decreased–, the z-axis speed was recorded. The collected data is displayed in *Fig. 32*.

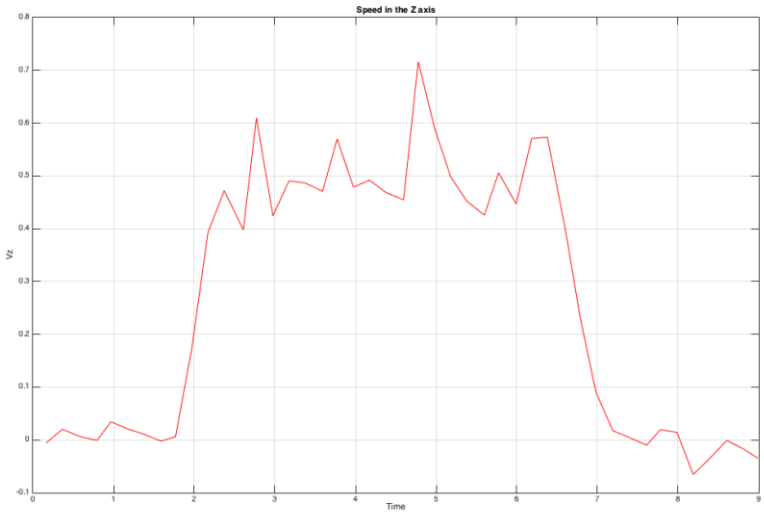


Fig. 32. Displayed results of a test flight.

A closer look is displayed in the next figure in order to make easier the analysis, where the important numerical values are marked:

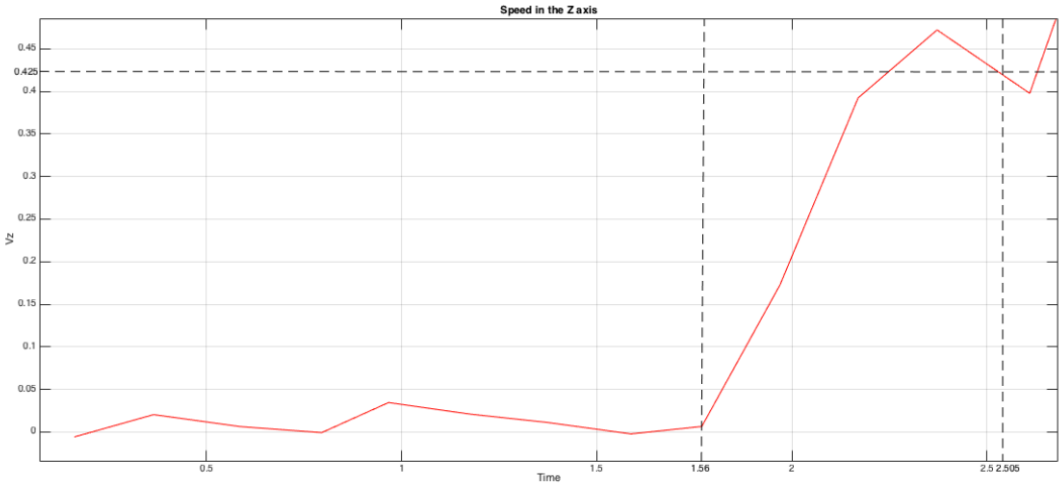


Fig. 33. Closer view of the previous figure. Some lines were added to the picture to mark the values of the speed and time.

A roughly constant value is reached at 0.425 m/s. As seen in the transfer function of (16), this value corresponds to the constant *k*. And, according to (18), it matches with the value of the parameter K_8. Now, the time constant τ must be obtained. The time passed between the beginning of the speed's increment in the z-axis and the moment the constant value is reached is measured. This time t is 2.349s. An approximation of this passed time says that it corresponds with five times the value of the time constant. It leads to a value of $\tau = 0.4698$. According to (17), the value of K_7 is the inverse of τ, so:

$$K_7 = 2.1286$$

$$K_8 = 0.425$$

And the identification of the rest of K_i parameters could be performed in the same way. Each of the variable's pairs corresponds to different velocity functions. For example, K_5 and K_6 are related with $\dot{\Psi}$ –these relationships can be seen in (13) –. So, the way to calculate the values of the rest of the constant pairs is repeating the same experiment previously explained using the corresponding velocity –vx, vy or $\dot{\Psi}$–.

6.3. The Observation Model

The observation model calculates the expected measurements based on the current state of the drone. As two distinct observation sources are used, two separate observations models are required.

6.3.1. NAVDATA Observation Model

This model relates the onboard measurements obtained through the navigation channel of the quadcopter –that we called "NAVDATA" in *Fig.3, 12 and 13*– described in section *4.1* and the state vector. The quadcopter measures its horizontal speed $(\overline{vdx}, \overline{vdy})$ in its local coordinate system, which is transformed into the world frame (vx, vy). The roll and pitch angles measured by the gyroscope are direct observations of the corresponding state variables. On the other hand, the height and yaw measurements are differentiated as observations of the respective velocities. The resulting measurement vector $z_{NAVDATA}$ and observation function $h_{NAVDATA}(\boldsymbol{\chi}_t)$ are:

$$z_{NAVDATA} := \left(\overline{vdx}, \overline{vdy}, z_{t-1} + \bar{h}_t - \bar{h}_{t-1}, \bar{\Phi}, \bar{\Theta}, \Psi_{t-1} + \bar{\Psi}_t - \bar{\Psi}_{t-1}\right) \tag{19}$$

$$h_{NAVDATA}(\boldsymbol{\chi}) := \begin{pmatrix} vx_t cos\Psi + vy \cdot sin\Psi \\ -vx \cdot sin\Psi + vy \cdot cos\Psi \\ z \\ \Phi \\ \Theta \\ \Psi \end{pmatrix} \tag{20}$$

The height and yaw angle variables are not direct measurements in order to avoid mistakes. Firstly, the height is calculated as an increment between the current measurement and the previous one. This increment is added to the preceding altitude estimation. Thanks to it, the system will not fall in measurement errors if the drone flies over an object. If this step would not be taken, the system will consider the next height measure the difference between the ultrasound altimeter and the top of the object instead of the difference with the floor. In the same way, this improvement avoids measurement mistakes if the MAV flies over a pit.

The yaw angle relative estimation is made in order to avoid measurement mistakes during the initialization. Due to the fact that the driver of the used drone does not incorporate a working flat trim option –at least for ROS, it does exist in the application for Android– the system is not able to set the measurements from the IMU to zero at desire. So, it was programmed that the first estimation of RPY was zero in the

initialization of the EKF node and the following estimations would be the increment of those measurements.

6.3.2. VSLAM Observation Model

When the VSLAM algorithm successfully tracks a video frame, its 6DOF pose estimation is transformed from the coordinate system of the front camera to the coordinate system of the quadcopter, leading to a direct observation of the quadcopter's pose given by:

$$z_{VSLAM,t} := f\left(E_{DC}E_{C,t}\right) \quad \in \Re^6 \tag{21}$$

$$h_{VSLAM}(\chi) := (x, y, z, \Phi, \Theta, \Psi)^T \quad \in \Re^6 \tag{22}$$

where $E_{C,t} \in SE(3)$ is the estimated scale-aware camera pose, $E_{DC} \in SE(3)$ the constant transformation from the camera to the quadcopter coordinate system. $f : SE(3) \rightarrow \Re^6$ is the transformation from an element of SE(3) to the roll-pitch-yaw representation $(x, y, z, \Phi, \Theta, \Psi)$.

6.4. Delay compensation

One of the main problems of using a low-cost drone is the delay caused by the Wi-Fi communication and the computational times. Other MAVs which implement onboard processing avoids this issue, but ours cannot carry an external processor neither processes it by itself. The delays in the estimation lead to a poor control even if the estimation is correct. Some tests were made before implementing the delay correction to the EKF and the problematic was evident. The estimation of the MAV's position was behind the current position of the drone in the terms of time, which sometimes leaded to a wrong command calculation. Furthermore, due to the time that takes to the command to reach the drone and take effect, it was almost always too late for the drone to react –the command was calculated for a drone's position that is not the current one–. Thus, and thanks to one of the main profits of using EKF is that the delay can be corrected with its usage an algorithm was developed.

The time needed on each iteration for the whole system is explained below. In first place, a frame is captured by the drone's front camera. This frame is sent via Wi-Fi to the ground station. The time it takes varies depending on the Wi-Fi nets in the surroundings. We have assumed a delay of approximately 120ms for it. Then, the VSLAM algorithm processes the frame and performs an estimation of the position. This step takes a median of around 30ms –depending on the VSLAM algorithm, as seen in 5.3.4–. With this estimation and the measurements of the NAVDATA channel –which are considered to be almost immediate due to they are received each 5ms, a much smaller time than the period– the data fusion is performed by means of the EKF. It is performed each 40ms. This period was chosen in order to assure that one new frame would be processed on each iteration of the system. With the estimation from the EKF, the immediately PID controller calculates and sends the new command u. As it has to be sent via Wi-Fi, it takes around 80ms to reach the MAV and take effect.

The frame that is processed by the ground station on every iteration corresponds to a previous instant and the command that controls the drone's motion is calculated in an instant previous to the moment it takes effect. We have estimated these delays as multiples of T –the system period, 40ms–. N constant correspond with the delay of the

frame transmission and M with the one of the command. For the next example, N will be 3 and M 2. If we consider that the current iteration is the third (3T), the frame that is currently being processed by the system –as said before, we are not considering the delay from the NAVDATA channel– was captured 3 system iterations before. Furthermore, the u command calculated in this iteration will not take effect until the fifth iteration. For this reason, a system that put to use the predictor to avoid these delays was developed. *Fig. 34* is presented for a better comprehension of the algorithm:

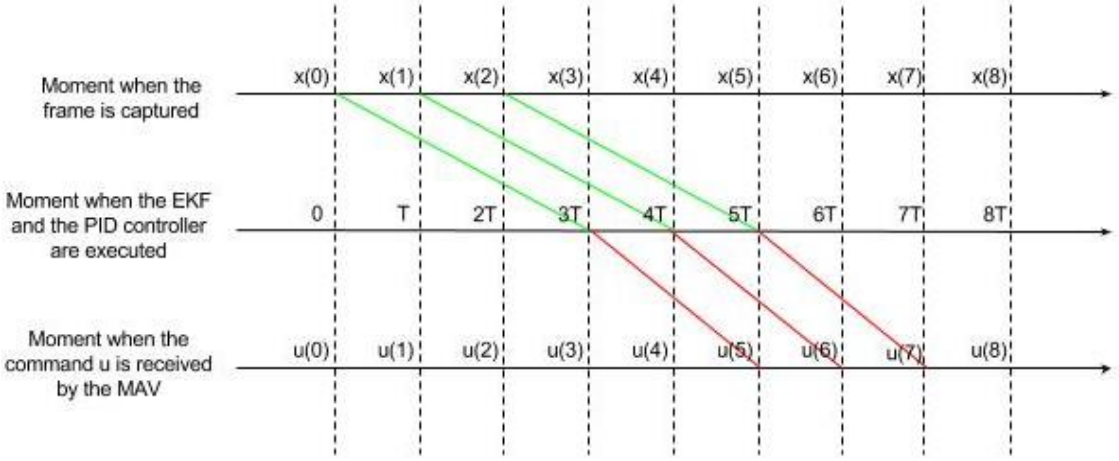


Fig. 34. Delay correction

On each iteration –let's keep on assuming we are on the third one– the pose's estimation made by the EKF is corrected with the IMU and VSLAM correction models. The frame processed by the VSLAM algorithm in order to correct the estimation was taken in the first iteration –x(0)–. With the new corrected estimation the system predicts the next N+M drone's positions based on this corrected estimation and the recorded previously sent control commands. With this prediction the system is able to calculate the control command which will correspond with the future position of the drone –assuming the delays caused by the Wi-Fi communications in both directions–.The algorithm performs the following calculations:

First, the correction is performed:

$$Correction\ by\ VSLAM\ model\ \rightarrow x(0) = f(VSLAM)$$

$$Correction\ by\ Navdata\ \ model\ \rightarrow x(0) = f(NAVDATA)$$

Then, the next N+M drone's position is predicted:

$$x(1) = f(x(0), u(0))$$
$$x(2) = f(x(1), u(1))$$
$$\dots$$
$$x(1+N+M) = f(x(1+N+M), u(1+N+M))$$

And finally, the PID controller calculates the control command based on the last predicted estimation:

$$u(1+N+M) = f\big(x(1+N+M)\big)$$

This control command is then recorded in a buffer in order to predict the next N+M drone's positions in the following system iterations.

6.5. Implementation

The EKF has been implemented as a ROS package. This package has a node with four include files that depend on it. Each of these includes contains a C++ object with a working function on it. One of those files is the scale calculator. It uses measurements from the onboard sensors and data received from the VSLAM package –LSD or ORB, depending on which one is being used at the moment– to perform the calculation of the absolute scale of the readings from the VSLAM system. The other three files correspond to the three models implemented in the EKF –prediction, VSLAM correction and NAVDATA correction–. They fuse measurements from the NAVDATA with the information from the visual method to achieve a robust, scale-aware estimation of the MAV's location. All of these include files contain a function utilized for perform its task. The EKF node is also adaptive. For instance, if the video stream from the MAV is "frozen" –something that happens in most of the flights due to the inner software of the drone and that depends on the number and strength of the surrounding Wi-Fi networks– the node can detect it. Then, if this "freezing" is successfully detected the filter starts to estimate the MAV's position using only the prediction and IMU correction models. Due to it, the estimation keeps on even in situations where the vision is disabled or the visual tracking is lost –which could lead to wrong estimations–. The implementation of the EKF along with the adaptive ability are represented in *Fig. 35*.

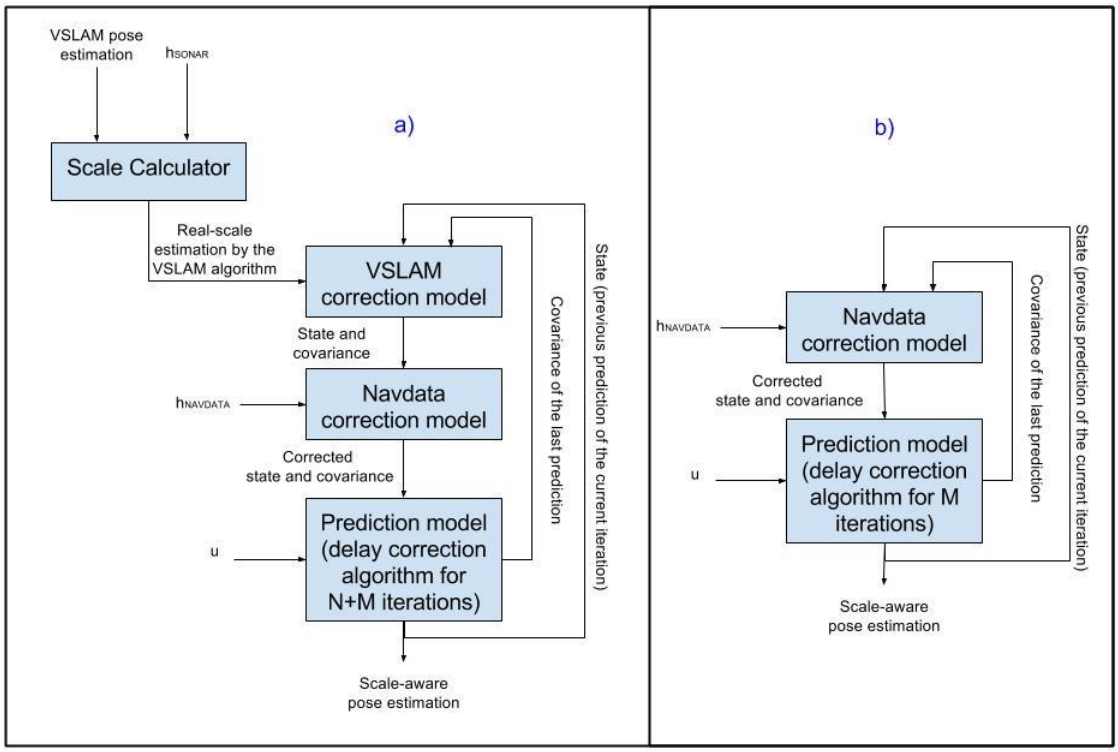


Fig. 35. Implementation of the EKF. The a) diagram represents the system when the video stream is being received, so all models are being used along with the scale calculator. If the system detects that the video stream is frozen, the EKF will only implement the prediction and NAVDATA correction model, as in b).

CHAPTER 7: PID CONTROLLER

Control theory deals with the problem of controlling the behaviour of a dynamic system, i.e. a (physical) system that changes its state over time and which can be controlled by one or more system input values. The general goal is to calculate system input values u(t), such that the system reaches and holds a desired state. In other words, the measured error e(t) between a given setpoint r(t) and the measured output of the system y(t) is to be minimized over time. In particular, the goal is to quickly reach the desired setpoint and hold it without oscillating around it, counteracting any random disturbances introduced into the system by the environment. This process is schematically represented in *Fig. 36*.

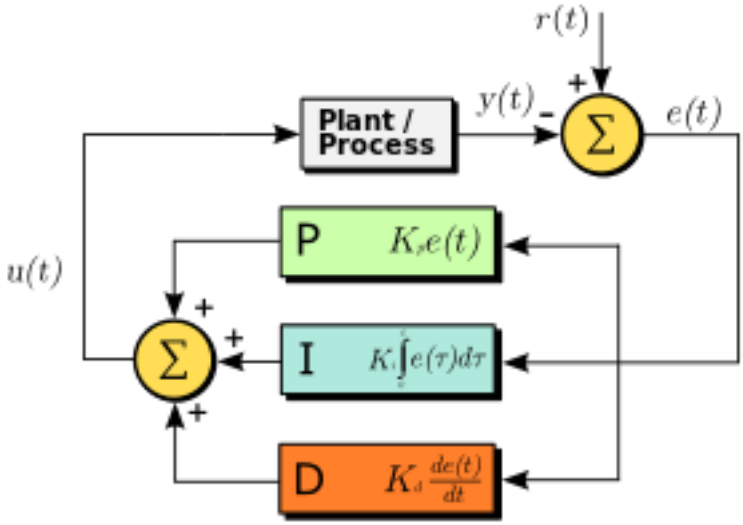


Fig. 36. Schematics of a PID control.

In this chapter, a proportional-integral-derivative controller (PID controller) –a generic control loop feedback mechanism widely used in industrial control systems– is presented. This mechanism is used in our approach to directly control the quadrocopter, in order to reach a desired setpoint. It is based on three separate control threads, the control signal being a weighted sum of all three terms.

7.1. Proportional term

The proportional term produces an output value that is proportional to the current error value. The proportional response can be adjusted by multiplying the error by a constant Kp, called the proportional gain constant.

The proportional term is given by:

$$P_{OUT} = K_p \cdot e(t)$$

A high proportional gain results in a large change in the output for a given change in the error. If the proportional gain is too high, the system can become unstable. In contrast, a small gain results in a small output response to a large input error, and a less

responsive or less sensitive controller. If the proportional gain is too low, the control action may be too small when responding to system disturbances.

7.2. Integral term

The contribution from the integral term is proportional to both the magnitude of the error and the duration of the error. The integral term in a PID controller is the sum of the instantaneous error over time and gives the accumulated offset that should have been corrected previously. The accumulated error is then multiplied by the integral gain (Ki) and added to the controller output.

The integral term is given by:

$$I_{OUT} = K_i \int_o^t e(\tau) d\tau$$

The integral term accelerates the movement of the process towards the setpoint and eliminates the residual steady-state error that occurs with a pure proportional controller. However, since the integral term responds to accumulated errors from the past, it can cause the present value to overshoot the setpoint value.

7.3. Derivative term

The derivative of the process error is calculated by determining the slope of the error over time and multiplying this rate of change by the derivative gain Kd. The magnitude of the contribution of the derivative term to the overall control action is termed the derivative gain, Kd.

The derivative term is given by:

$$D_{OUT} = K_d \frac{de(t)}{dt}$$

Derivative action predicts system behaviour and thus improves settling time and stability of the system. An ideal derivative is not causal, so that implementations of PID controllers include an additional low pass filtering for the derivative term to limit the high frequency gain and noise.

In *Fig. 37* the response of the system when using just the proportional term of the controller is displayed at the left. The output reaches its goal –the reference–, but after a while and with a lot of oscillation. In real-life systems purely proportional controllers cause severe overshoot, leading to strong oscillations. In the same figure, at its right, the same system with its reference is represented, this time using a PD controller. In this case, the derivative term dampen occurring oscillations: the higher the rate of change of the error, the more this term contributes towards slowing down this rate of change, reducing overshoot and oscillations. This is an ideal system, as there is no bias in it that could cause a steady state error.

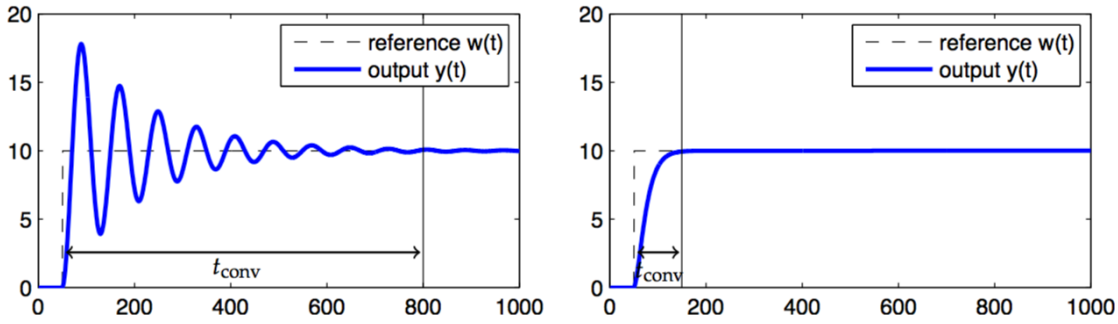


Fig. 37. At the left, a P control performance. At the right, the same but adding a derivative term.

In *Fig. 38*, another system is reviewed. While the reference is the same, this system has some bias. Due to it, a PD controller reaches a steady state, but it has an offset. In order to compensate this offset the integral term is included. However, this term needs to be treated with caution as it may increase convergence time and cause strong oscillations –as seen in the figure, an overshooting appears when the integral term is added to the control–.

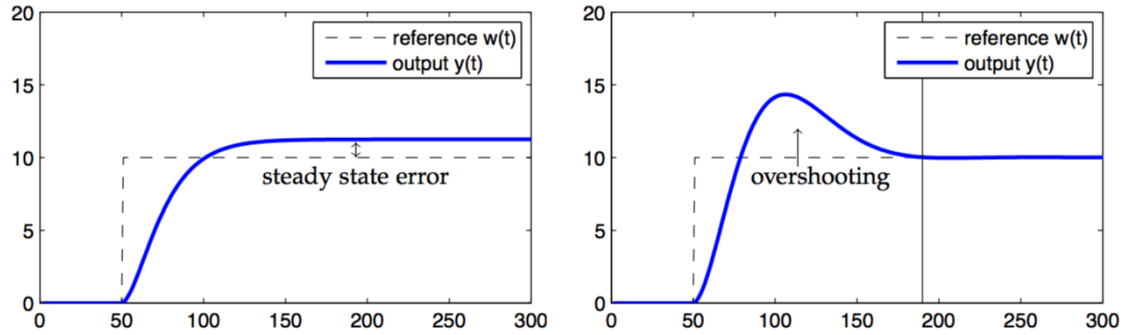


Fig. 38. At the left, a PD controller performance. At the right, the same but adding an integral term.

7.4. Implemented Controller

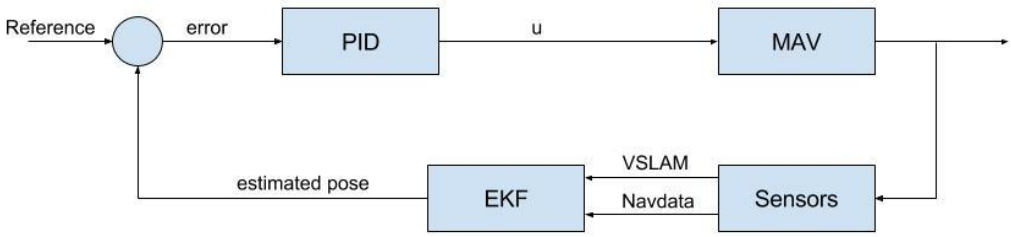


Fig. 39. PID controller blocks diagram.

A PID controller was included in this work in order to control the motion of the MAV based on its estimated position. The controller requires a reference $\boldsymbol{r} = (\hat{x}, \hat{y}, \hat{z}, \hat{\Psi})$ as the desired position of the drone in relation with the surroundings. The EKF will bring the estimation of the pose, as shown in *Fig. 39*. The difference between the reference and the estimated pose is the error that will be minimized by the PID controller by sending to the MAV an appropriate control command $\boldsymbol{u}=(\widehat{vx}, \widehat{vy}, \widehat{vz}, \hat{\Psi})$. Then, the MAV receives this command and with the reaction of its thrusters changes its position. Then, the sensors measure the new readings from the MAV –data from the

onboard sensors and new keyframes from the vision algorithm–. The block PID is where the error signal is computed in order to calculate the command signal by means of the following equations:

$$\widehat{vx} = cos\Psi[Kp(\hat{x} - x) + Kd \cdot \dot{x}] + sin\Psi[Kp(\hat{y} - y) + Kd \cdot \dot{y}] \quad (23)$$

$$\widehat{vy} = -sin\Psi[Kp(\hat{x} - x) + Kd \cdot \dot{x}] + cos\Psi[Kp(\hat{y} - y) + Kd \cdot \dot{y}] \quad (24)$$

$$\widehat{vz} = Kp \cdot (\hat{z} - z) + Kd \cdot \dot{z} + Ki \cdot \int(\hat{z} - z) \quad (25)$$

$$\hat{\dot{\Psi}} = Kp(\hat{\Psi} - \Psi) \quad (26)$$

The only velocity command that must be controlled by an integral term is the altitude, as seen after several experiments. Also, the yaw position is well adjusted using just the proportional part of the controller. On the other hand, the velocities of X and Y axis need the derivative term in order to dampen oscillations in the estimation of the pose –otherwise, it could not reach a stable estimation–. Due to the steady state error obtained because of the PD controller is almost negligible and the integral term caused dangerous overshooting this last term was declined for keeping the MAV safe.

The PID controller allows the algorithm to drive the MAV along a series of points in the map so it can follow a specific track, as will be shown in the experimental results of Chapter 8. The controller has been implemented as a package in ROS. This package contains a node with an include file related to it. The include file was written with a function in it in charge of perform a simple PID control. It receives a goal position, the current estimated position and calculates the errors in x, y, z and Ψ between them. These errors allow obtaining the control signals that will be sent to the drone in order to reach the desired position. It is executed on each iteration –each 40ms–.

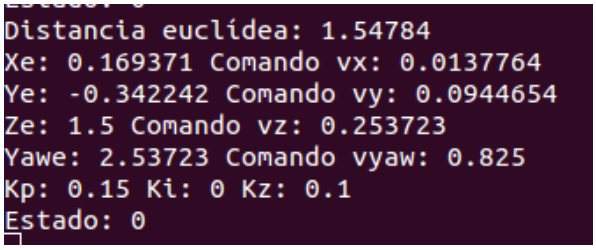


Fig. 40. Information given by the PID controller main script.

The main script node is in charge of coordinating the signals received and sent from the package and to guide the PID controller along all of the stages of the previously configured path. This script returns by the prompt information about the current estimation and the commands sent to the MAV. Firstly, the node calculates the errors –the distances between the estimated position in this iteration and the goal–. Then, the commands that will be sent are computed and the Euclidean distance is estimated. The Euclidean distance is used as a reference that allows the system to know when the MAV has reached the goal position. Furthermore, the PID coefficients –Kp, Ki, Kd– were configured as variables so they change depending on this distance (if the drone is so far from the next goal position, the coefficients are increased in order to make it move faster and if it is close are decreased in order to make the movements softer).

CHAPTER 8: RESULTS

In this chapter, we design and perform several experiments to test and validate the proposed SLAM system. All these experiments have been done using the Bebop of Parrot as flying unit, and a laptop as ground station with the SLAM system running on it. In order to validate the estimated pose and characterize the different errors, it has been necessary to design a ground truth system based on an external sensor. The first section of this chapter describes the ground truth system, and later we show the different experiments and analyse the obtained results.

8.1. Ground Truth System

Since a motion capture system to obtain a reliable measure of the actual pose of the MAV was not available, we have designed a simplified system that allows us to approximately estimate some coordinates of the MAV's pose under certain assumptions. The value of this ground system is that, although some approximations will be done, it is an external system that has not cumulative error.

The ground truth system is based on a monocular camera that was positioned on the ceiling of the laboratory, as it is shown in *Fig. 41*. Adding a pair of distinguishable artificial markers to the MAV (two coloured circles) –as it is shown in *Fig. 42*– it is possible to estimate the x, y, z and Ψ coordinates of the MAV under some assumptions.

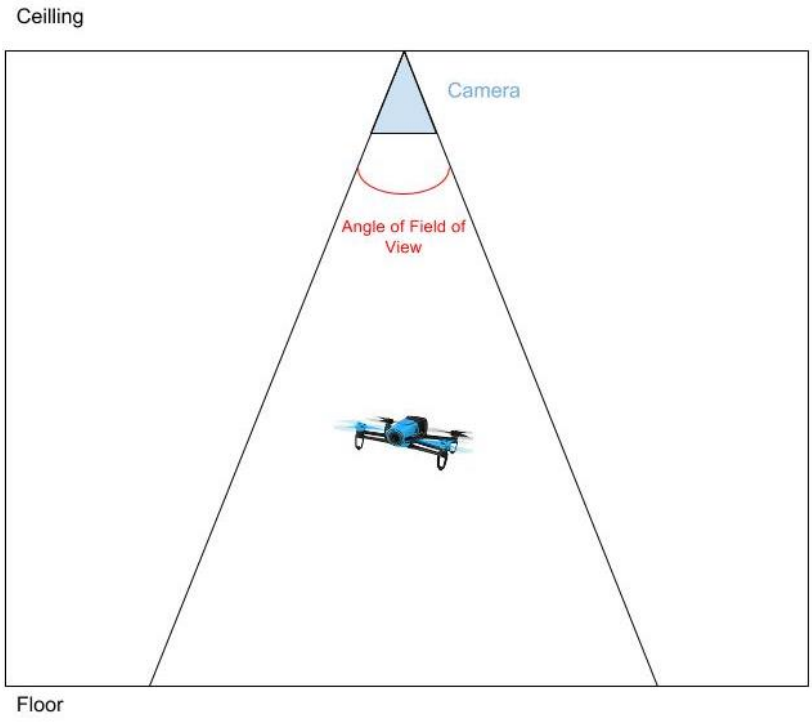


Fig. 41. Ground truth system

Fig. 42. Bebop drone with both coloured circles incorporated as markers for the ground truth system.

8.1.1. Tracking Algorithm

In order to track the real position of the drone, we have coded a script in Matlab that processes the recorded video stream of the camera. This algorithm calculates MAV's real position thanks to the coloured circles incorporated at both sides, knowing that the centre of the drone is in the middle of both circles and the orientation can be obtained if it is known that the green circle is at the right. This script looks for two circles –the colour of both is known– and extrapolates the 6DOF pose of the drone thanks to it, using some geometry and approximations. The main assumption is that the drone is always in a horizontal plane, which is quite realistic because horizontal velocities of the MAV are very small. Under this assumption, the distance between markers inform us about the height of the MAV, and the x, y and Ψ coordinates can be extracted.

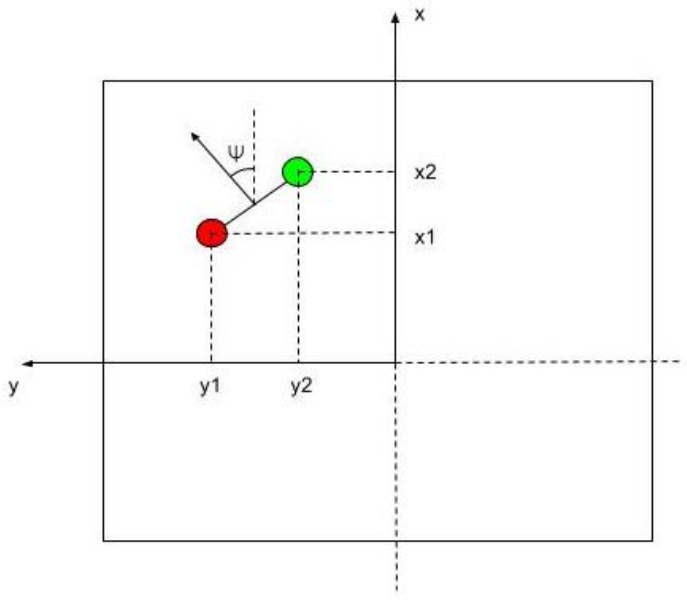


Fig. 43. Measurements from the ceilling camera

Firstly, a calibration is made in order to calculate a constant (related with the instrinsics of the camera) that will help the system to calculate the position of the MAV in real scale. This constant is called *k* and needs an initial measure where the drone is placed on the floor.

$$k = \frac{l \cdot D}{L} \tag{27}$$

where l is the euclidean distance between the centroids of both circles in pixels; L is the euclidean distance between the centroids of both circles in cm and D is the distance between the camera and the floor. Once the constant is obtained the same equation allows the system to recalculate the distance D (distance from the camera to the MAV) on each photogram of the video depending on the value of the variable l. This is the way the algorithm calculates the height of the MAV with just one camera: using the distance between both circles in pixels as it will be bigger when the drone is closer to the camera and smaller when it is further. So, each value of D is calculated as follows:

$$D = \frac{L \cdot k}{l} \tag{28}$$

Knowing the changing value of the variables on each iteration, the position in real scale of the drone can now be obtained:

$$X = \frac{x_1 + x_2}{2} \cdot \frac{D}{k} \tag{29}$$

$$Y = \frac{y_1 + y_2}{2} \cdot \frac{D}{k} \tag{30}$$

The ground-truth position of the drone P(X,Y) is calculated knowing the position of each circle p1(x_1,y_1), p2(x_2,y_2). Furthermore, due to the total height from the camera to the floor is known –h_T=4.35 meters–, the altitude of the drone can be calculated with the following equation:

$$Z = h_T - D \tag{31}$$

And with trigonometric, the orientation can be obtained too:

$$\Psi = atan\left(\frac{x_2 - x_1}{y_2 - y_1}\right) \tag{32}$$

The representation of the tracking of a MAV's flight is represented in *Fig 44*. Firstly, the position of both circles is obtained and then the centre of the drone is calculated.

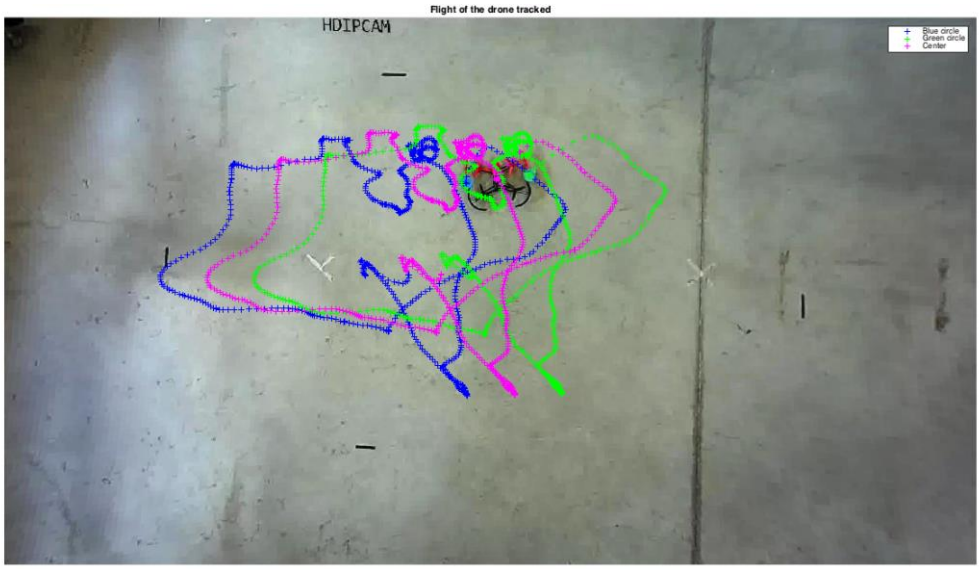


Fig. 44. Tracking of the flight of the drone. Blue crosses represent the locations of the blue circle and the green crosses the locations of the green one. The purple crosses represent the calculated center of the MAV.

In order to achieve a better representation, the toolbox *Robotics Toolbox* for Matlab was put to use. It was useful for the calculating of the homogenous transform of the localizations of the MAV and its plots. *Fig. 45* represents a flight of the drone tracked but with some of the homogenous transforms printed over it. In this way it is easier to represent not only the tracking of the path followed by the drone, but the orientation that it had by the moment.

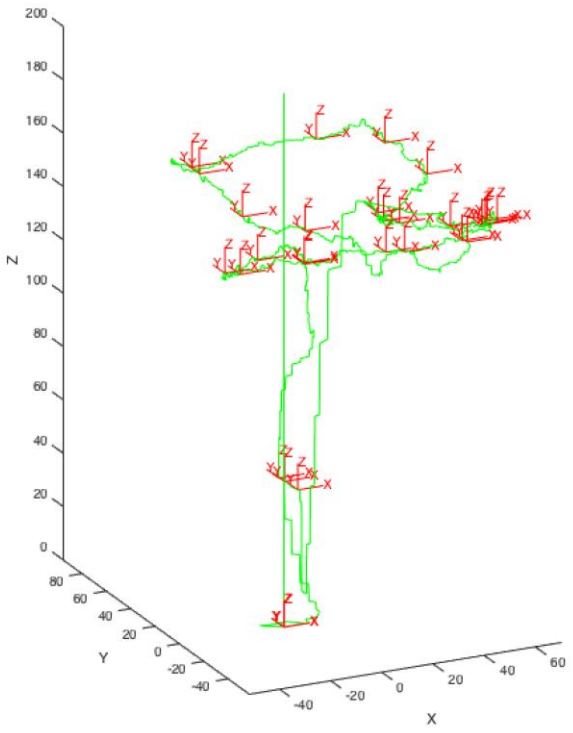


Fig. 45. Flight of the drone tracked with homogenous transforms printed.

Thus, the way followed to test the system is to record the performance of the system and make a processing of the images obtained. Firstly, each frame from the video is extracted and then undistorted with Matlab. Finally, the whole set of undistorted frames is processed in order to track the real and undistorted position of the drone. Then, the Matlab script accomplish the comparison and the error extraction.

8.1.2. Camera Calibration

A fundamental part for setting up the whole ground truth system was the camera calibration. It had to be done before for all the cameras that we used (the ones from both drones, from the laptop…), but the process of calibration that is here explained is the one for the camera in the laboratory's ceiling.

The method used to do so was the *Camera Calibration Toolbox for Matlab*[6]. As its name indicates, it is a toolbox developed for Matlab. This toolbox was used for this work instead any other because of its effectiveness and accuracy. The negative side of the method is that it is not automatic and needs some human interaction.

The first thing that should be done in order to calibrate the camera is to acquire a set of pictures taken with the target camera where a pattern –typically a chessboard– appears in different positions and orientations. The pattern should be captured in different positions, altitudes and inclinations in order to measure the intrinsics in different heights and also the skew.

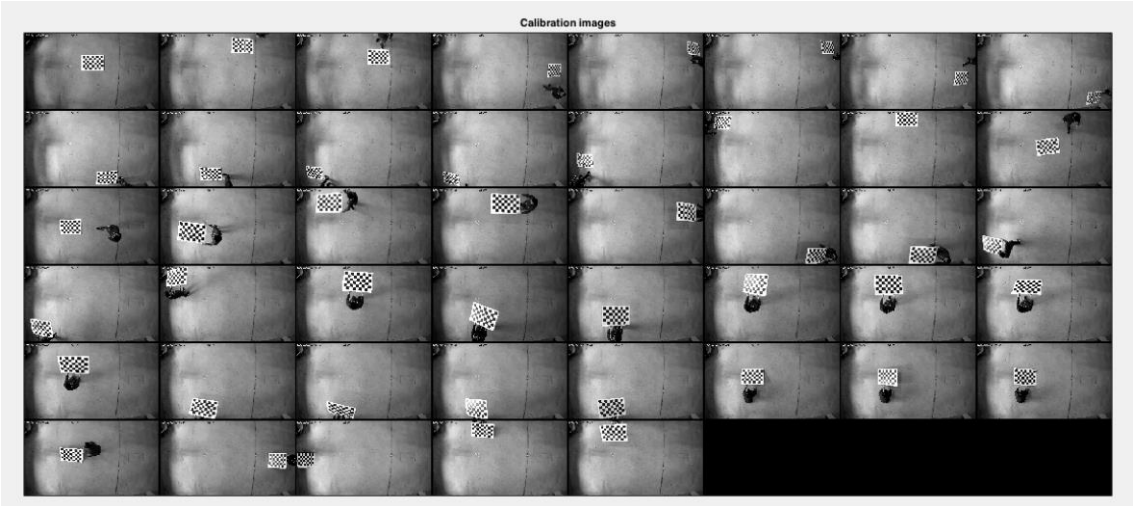


Fig. 46. Set of pictures taken to calibrate the camera.

Once the toolbox has read the set of pictures, it is necessary to mark the four corners of the chessboard on each of the pictures. As said before, most of calibration methods perform this step automatically, but not this one.

[6] http://www.vision.caltech.edu/bouguetj/calib_doc/index.html

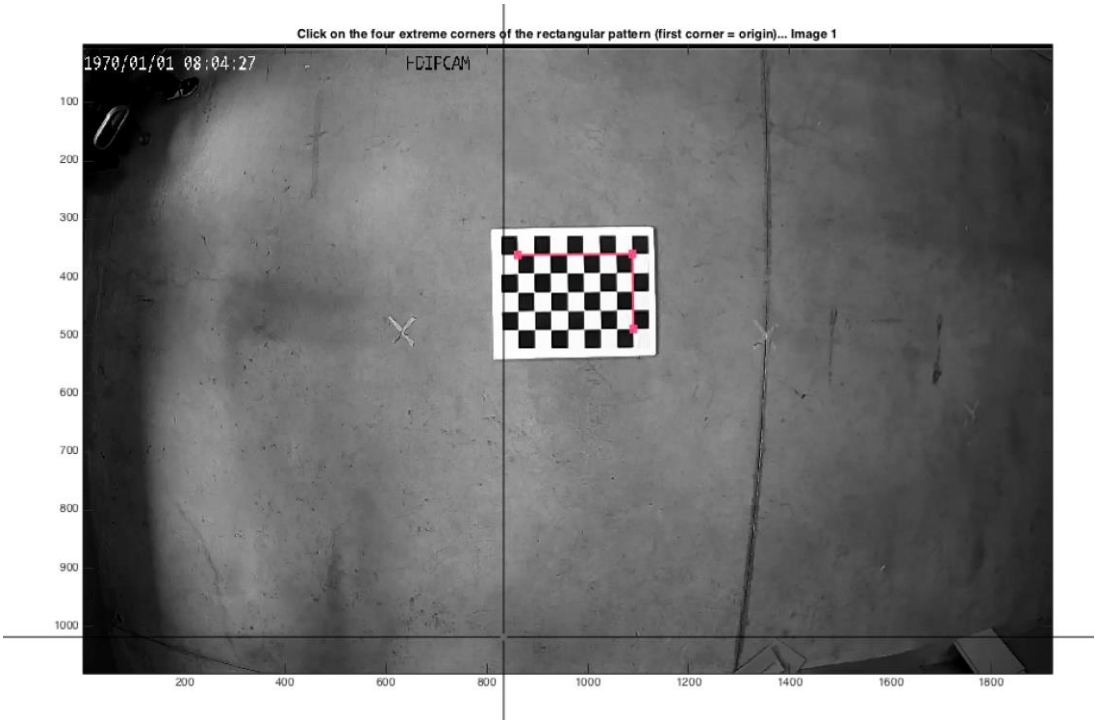


Fig. 47. Selecting the four corners of the chessboard.

This step will take a while, but due to the results this time is worth spent. If the system recognise all the squares between the four marked corners, an image similar to *Fig. 48* should appear:

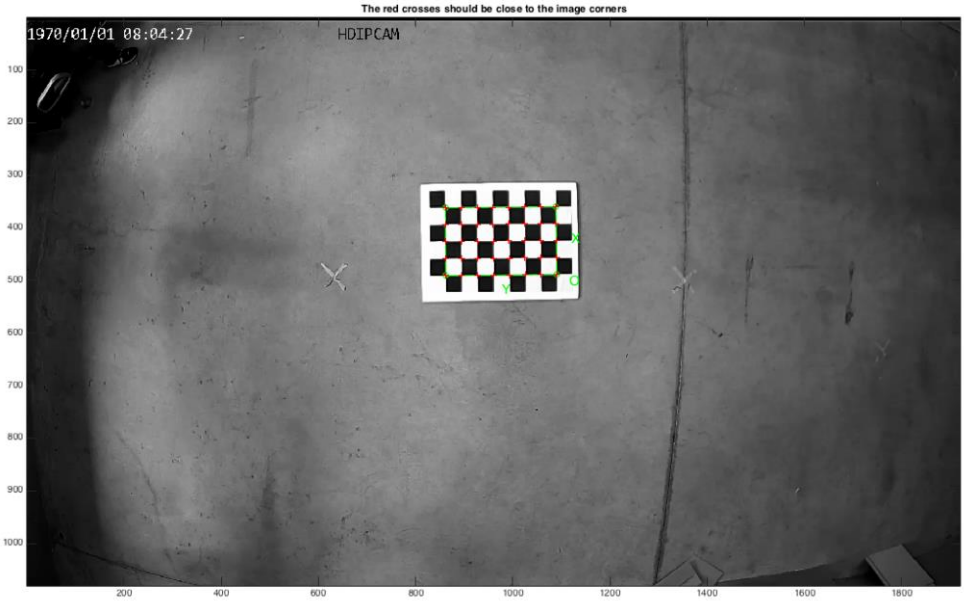


Fig. 48. All squares recognized.

Once the corner extraction is performed, the calibration main step may be accomplished. The system will return both extrinsic and intrinsic parameters, but without some modifications the error will be large. *Fig. 49* displays the error in pixels. Each cross represents the error depending on the picture. The script allows clicking on each of the crosses and it returns the error numerically and from which picture it comes from.

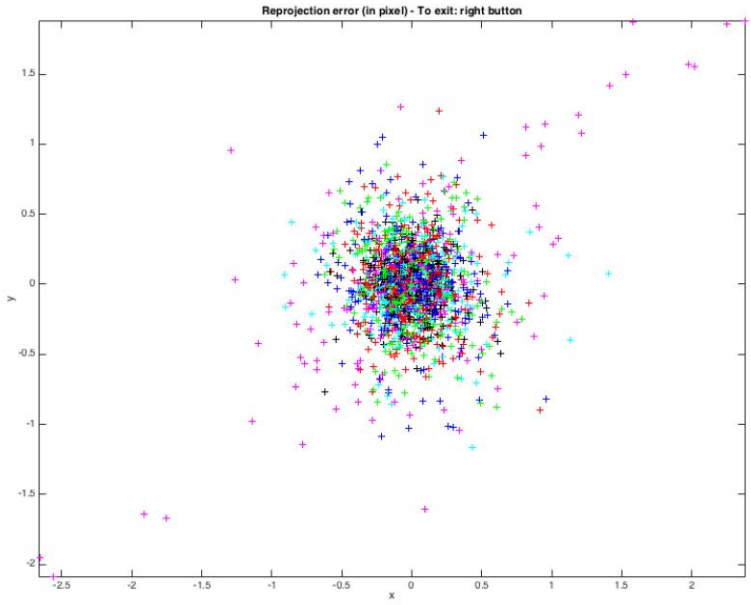


Fig. 49. Reprojection error without corner recalculation

If the pictures that are giving some problems are known –in this case by clicking on each of the target crosses–, the algorithm allows recalculating the corners depending on a given window. These windows represent an area –which the algorithm asks for its size– on the image where the algorithm can look for the position of the corner and recompose it. After this step, the calibration step should be done once more. Now the results should be better than before. These steps can be repeated as many times needed in order to achieve satisfying results.

Fig. 50 represents the reprojection error where the corner recalculation have been done a few times. The error in pixels is significantly reduced. A good calibration will allow a reprojection error of less than one pixel.

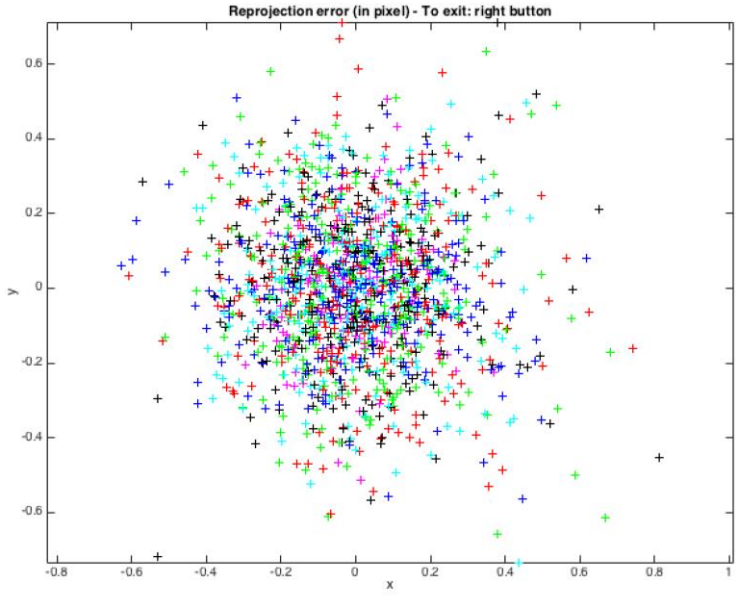


Fig. 50. Reprojection error with corner recalculation.

After all this steps the results can be evaluated. Matlab will return them in the following format, where intrinsic parameters are presented:

```
%-- Focal length:
fc = [ 1381.464674497992746 ; 1369.588046959953999 ];

%-- Principal point:
cc = [ 908.141070410591510 ; 411.958838419264737 ];

%-- Skew coefficient:
alpha_c = 0.000000000000000;

%-- Distortion coefficients:
kc = [ -0.367231995407932 ; 0.097184689401318 ; 0.001232102399891 ; -
0.002651682895832 ; 0.000000000000000 ];

%-- Focal length uncertainty:
fc_error = [ 17.506239189864644 ; 16.988183955035929 ];

%-- Principal point uncertainty:
cc_error = [ 10.953708474264381 ; 13.016513363654543 ];

%-- Skew coefficient uncertainty:
alpha_c_error = 0.000000000000000;

%-- Distortion coefficients uncertainty:
kc_error = [ 0.008490237042604 ; 0.005221438985826 ; 0.001529055899329
; 0.001302242978082 ; 0.000000000000000 ];
```

The program will also return some other parameters. The extrinsics of each picture are some of those parameters:

```
%-- Image #1:
omc_1 = [ 3.070369e+00 ; -1.192283e-02 ; 2.160154e-02 ];
Tc_1  = [ -1.424322e+02 ; 2.392306e+02 ; 4.190836e+03 ];
omc_error_1 = [ 2.365183e-02 ; 1.849515e-03 ; 2.900762e-02 ];
Tc_error_1  = [ 3.327460e+01 ; 3.985263e+01 ; 5.380459e+01 ];
```

Where *omc_i* and *Tc_i* are rotation and translation vectors respectively.

Another interesting function of this toolbox is that it displays extrinsic parameters of the camera in the world or camera coordinates. It allows comparing the results and the pictures used for the calibrations with the data obtained by the toolbox. Plotted in *Fig. 51* the extrinsic parameters –translation and rotation vectors– appear world and camera centred. It is measured in pixels.

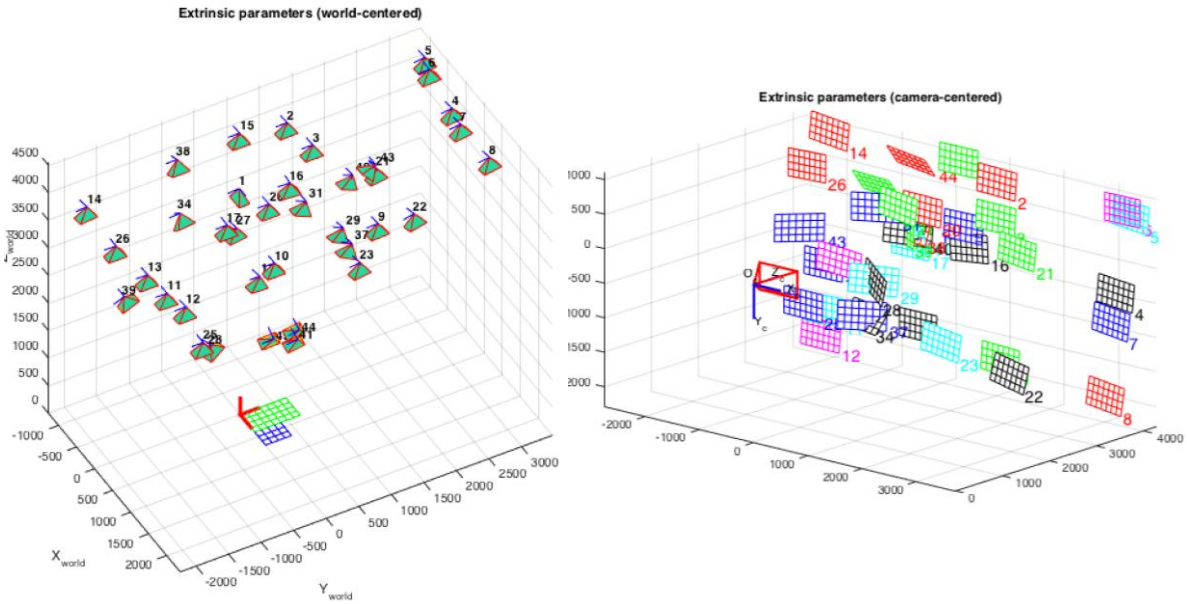


Fig. 51. Display of intrinsics and extrinsics.

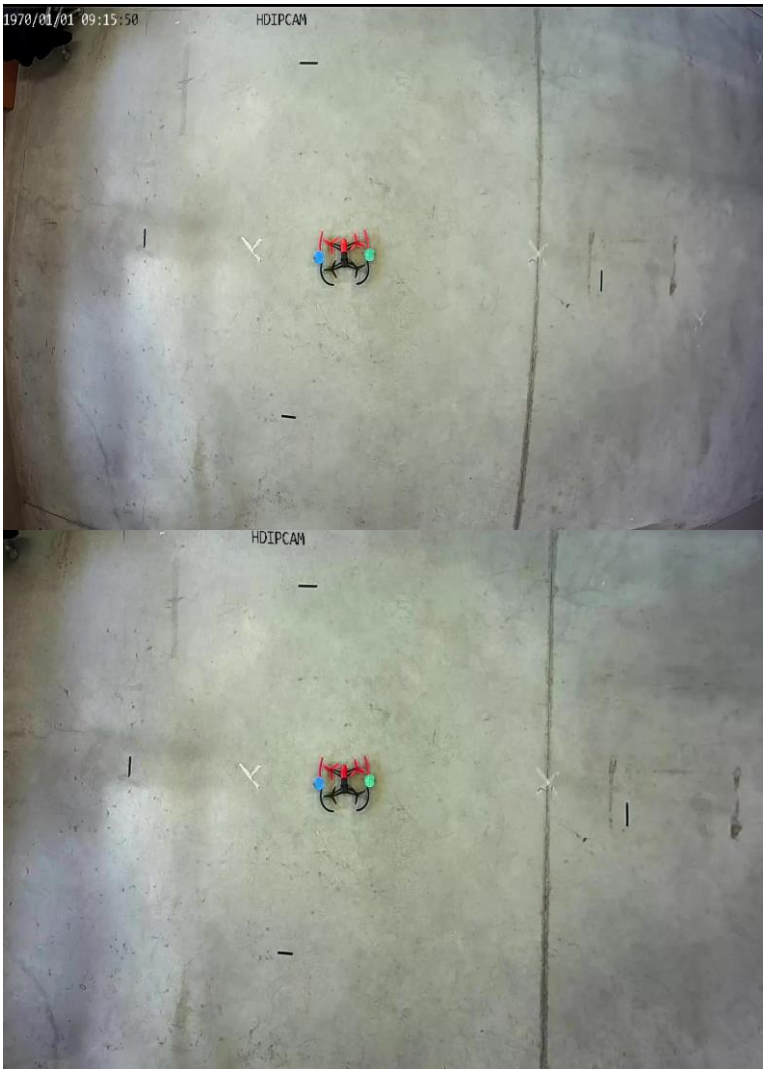


Fig. 52. Difference between original image and its undistorted version.

Once the calibration is successfully finished an undistortion of a set of images can be made. Thus, each frame of a video can be processed looking for the real position of the drone. *Fig. 52* represents the difference between the original image and its undistorted version.

8.2. Experimental results

This section lists and describes the experimental results and under which conditions were. The results will be explained using different models and will be compared between each other.

8.2.1. Test Conditions and Benchmark

For a better interpretation of the results achieved in this work, a benchmark was developed for the Matlab software. The data from a certain flight was recorded –both the information from the ground-truth and the pose estimation of the system– so it could be compared afterwards. Furthermore, the needed data for the performance of the EKF was also recorded, so this package could be launched at will. Thus, we run the algorithm in different situations, always comparing the results using both VSLAM methods used for this work –LSD-SLAM and ORB-SLAM–. During the recorded flight we made the drone to move along a series of points performing a rectangle of 120cmx60cm. The size and shape of the performed figure by the drone were chosen due to some limitations imposed by our ground truth. Firstly, it only can track in a reliable way the X and Y axis so any planar geometric figure could have been elected. Furthermore, the camera has a limited field of view, which is larger in the width axis than in the height one. These reasons made us to choose the shape and the length of the track.

8.2.2. Performance of the whole system

Firstly, the performance of the whole system will be displayed and compared between VSLAM algorithms. This experiment is made with and without initialization stage for both methods. Afterwards, the improvements introduced by the sensor fusion in the EKF are validated.

a) Performance without initialization stage for the VSLAM algorithms

In the first experiment, the system had not an initialization stage for its VSLAM algorithms. As explained in the Chapter 5, some monocular VSLAM techniques need an initialization stage in which the algorithm extracts enough features from the environment so it could build a point's map. In this case, ORB-SLAM require of it, as it will be seen in the following. On the other hand, it is not required for LSD-SLAM. Nonetheless, one of the flight's data set recorded when using LSD-SLAM was discarded because it had a wrong initialization –due to the random depth values that the algorithm sets at the beginning–. The tracking of the system using each of the methods is shown in *Fig. 53*. This figure displays the performance of the system using all the models, comparing the results when using each of the VSLAM algorithms. Five tests were made using each algorithm putting to use the same benchmark –all of them represented by a coloured line, as expressed in the legend–. The track calculated by the ground truth system is represented too as a path of red crosses. Finally, the desired track is represented as a magenta thick line. We presented the results obtained when using

LSD-SLAM above and the ones obtained when using ORB-SLAM below. It is important to remark that this work is focused in obtain a scale-aware, accurate estimation system but some upgrades where made to it in order to improve the performance of the PID controller. Despite of it, due to some reasons that will be explained in the next chapter the drone is not able to follow exactly the marked path.

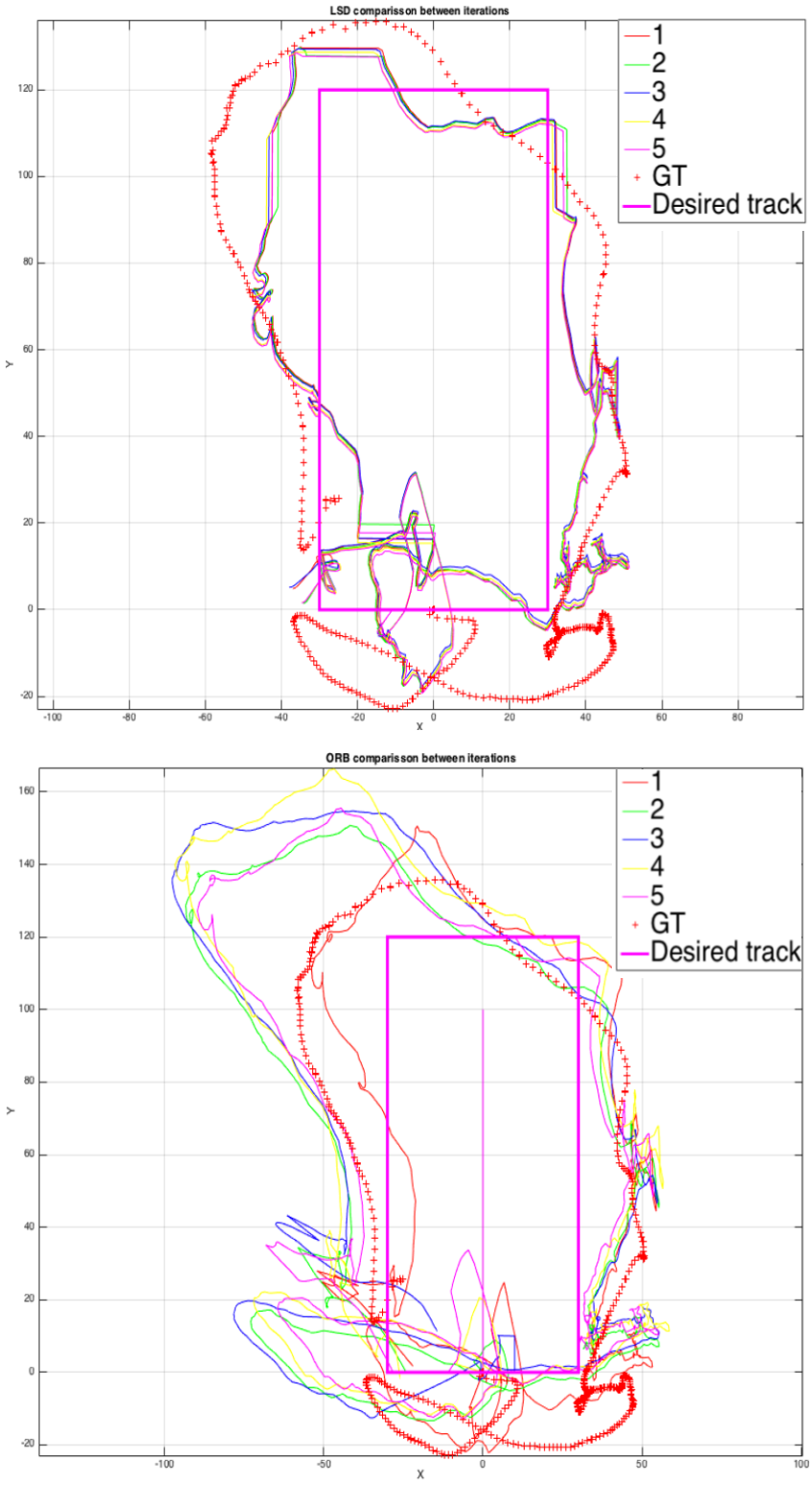


Fig. 53. Performance of the system using each of the VSLAM methods. Several tests were accomplished, each of them represented as a coloured line. The red dots shape represents the ground-truth tracking.

One of the prerequisites of the system developed in this work becomes clear looking at *Fig. 53*. In order to calculate the scale, the system needs to start the VSLAM algorithm's measurements from the floor along with the node EKF node. If not, the scale is not properly calculated –due to it have to be a differential calculation–. As seen in (2-5) equations, the scale is calculated dividing the height estimated by the VSLAM algorithm with the real height measured by the vertical ultrasounds sensor. These equations take for granted that the first measurement from both sources is taken from the floor. If not, the equations should have looked like this:

$$scale = \frac{h_{SONAR_{Current}} - h_{SONAR_{First-frame}}}{h_{VSLAM_{Current}} - h_{VSLAM_{First-frame}}}$$

$$x_{REAL-SCALE} = x_{VSLAM} \cdot scale$$

$$y_{REAL-SCALE} = y_{VSLAM} \cdot scale$$

$$z_{REAL-SCALE} = z_{VSLAM} \cdot scale$$

This method was tested, but due to the dynamic changes in the estimations' scale of the VSLAM algorithms it leads to drift and scale calculation errors. By the way, as said before, if ORB-SLAM does not have an initialization stage before the take-off, the estimations will not be as accurate as they could be using the current scale calculation method. Furthermore, the algorithm will lose some tracking frames because it would need the first frames of the video stream for the features extraction and the initial points map building, so not all the camera's movements will be tracked. It increases the estimation error. The results are presented in the following table. The errors are presented in the same way than in 5.3.4, but this time in centimetres. In the following, the results will be expressed in this way.

	LSD-SLAM	ORB-SLAM
Compared pose pairs	207	288
Absolute translational error (RMSE)	6.9894	8.0597
Absolute translational error (mean)	5.0335	5.9480
Absolute translational error (median)	3.4622	4.4534
Absolute translational error (std)	4.8494	5.4387
Absolute translational error (min)	0.0123	0.0194
Absolute translational error (max)	18.7031	26.3586

Table 2. Results without initialization stage. Error results are given in centimeters.

Even in this adverse situation the highest RMSE error is around 8cm for the pose's estimation of the drone, making this system a robust, scale-aware and accurate method.

b) Performance with initialization stage for the VSLAM algorithms

In the following experiment, the flight of the drone will be preceded by an

initialization stage –the drone is moved smoothly on the floor in all directions without taking-off–. Thanks to it, the ORB-SLAM algorithm could build the points map and therefore start the tracking before the take-off.

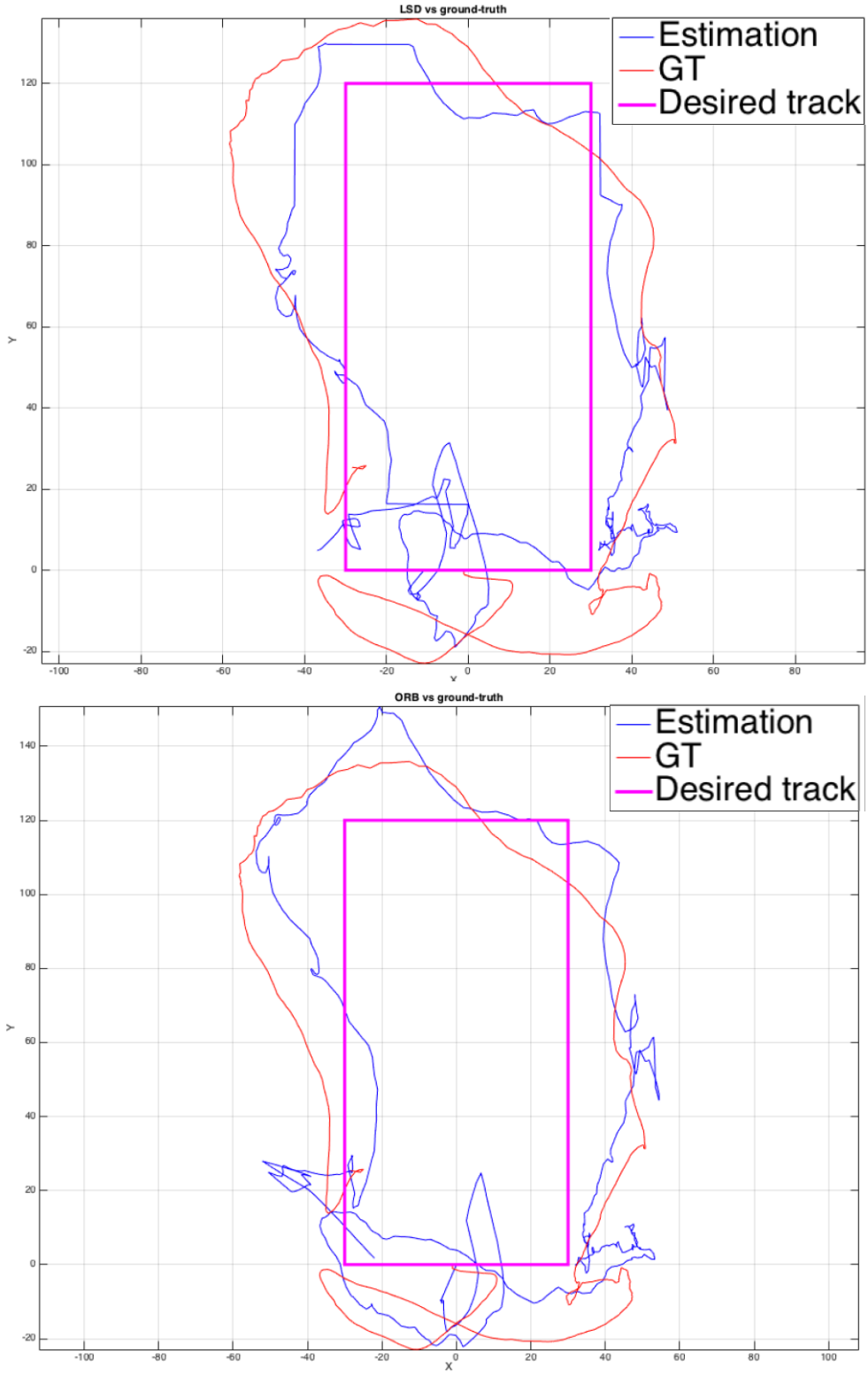


Fig. 54. Results obtained with an initialization stage. The blue line represents the estimated tracking of the drone, while the red dots shape represents the ground truth tracking.

The results now achieved are shown in the following chart:

	LSD-SLAM	ORB-SLAM
Compared pose pairs	210	348
Absolute translational error (RMSE)	6.7410	5.3203
Absolute translational error (mean)	4.8385	3.9001
Absolute translational error (median)	3.2138	2.7995
Absolute translational error (std)	4.6936	3.6186
Absolute translational error (min)	0.0022	0.0061
Absolute translational error (max)	18.2302	14.5284

Table 3. Results with an initialization stage. Error results are given in centimeters.

Now, the error is drastically reduced for ORB-SLAM, while LSD-SLAM achieves similar results to the obtained without the initialization stage. The system achieves better results putting to use ORB-SLAM with an initialization stage because of two reasons: the drone's pose started to being calculated from the floor, so the scale was properly calculated; and because the estimation started before the camera began to move, so all the frames were tracked.

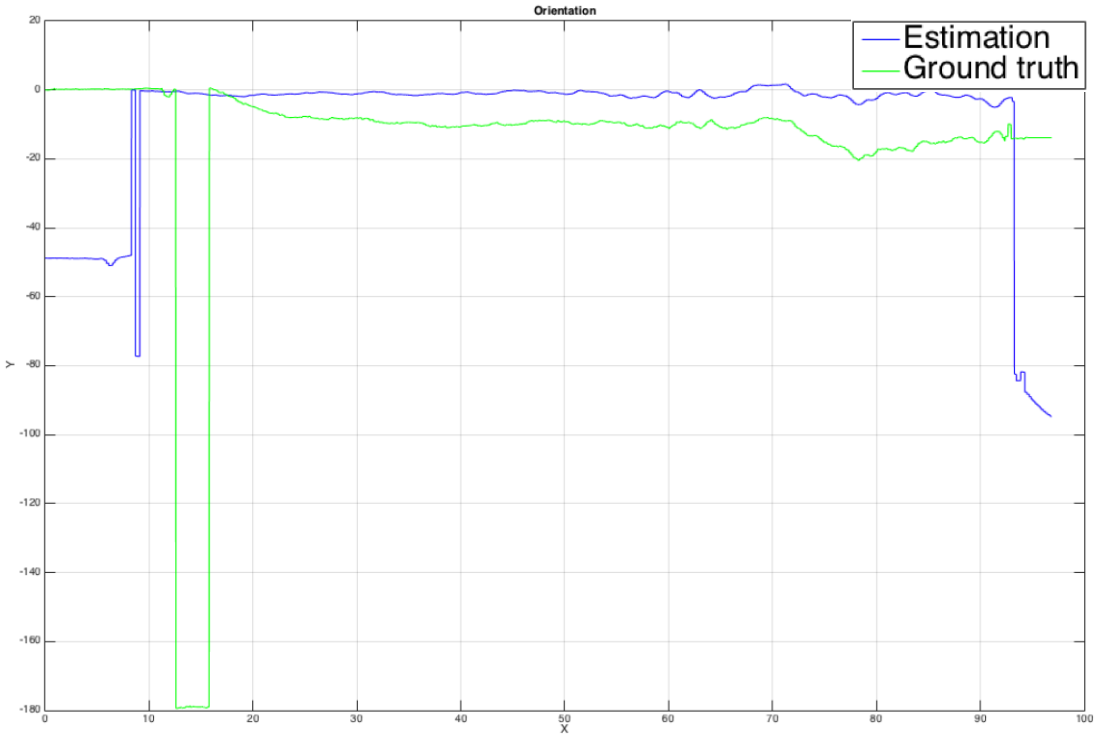


Fig. 55. Comparison between the estimated yaw orientation and the tracked by the ground truth.

Not only the absolute translational error should be measured, the error in the estimation of the orientation is presented too. In the following figure the yaw estimated by the system and the tracked by the ground truth are compared, both expressed in degrees. As said previously, the estimation of the yaw is not absolute but differential. It means that the yaw is estimated in relation with the initial values so when the EKF node is launched the first yaw value that the filter estimates –under some conditions– would be the initial point. These differential measurements are performed only if the drone is flying at one meter or higher. This decision was taken due to the readings from the gyroscope "jumps" when the drone changes rapidly its altitude. The tracked yaw also fails in its measurements due to the abrupt take-off of the drone that makes the drone to strongly incline, incapacitating the yaw calculation by the ground system.

A closer look of the previous pictures makes easier to understand the measurements. While the estimations are always close to 0 –the maximum estimated difference is 5 degrees–, the ground truth says that there is an error in that estimation caused by the drift.

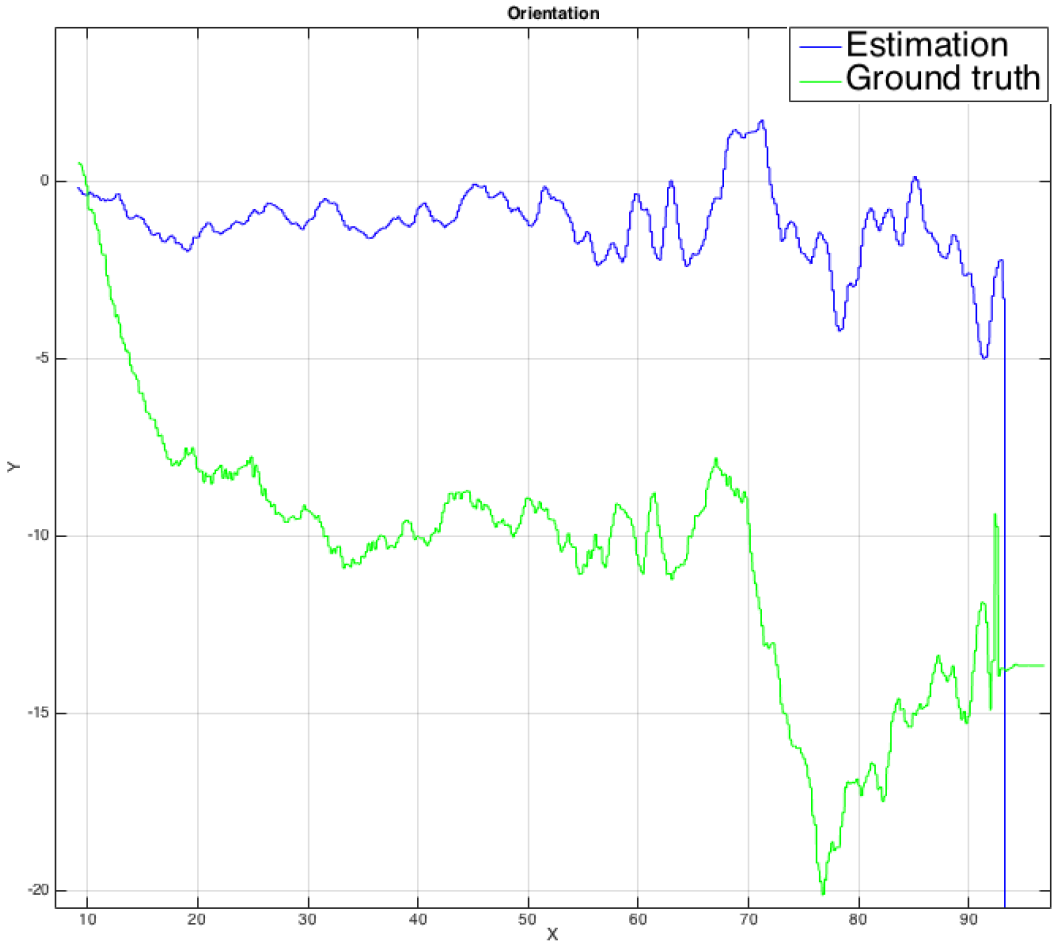


Fig. 56. Closer look to the results presented in the previous figure.

Avoiding the sections of the recorded data where the EKF was not estimating the orientation, the results of the error's calculations indicate that the RMSE error is around 6.5 degrees. Although there is a visible error caused to the drift in the yaw estimation, it is insignificant and it does not affect the performance of the system.

	All models with ORB-SLAM
Compared pose pairs	3411
Orientation error (RMSE)	6.5365
Orientation error (mean)	5.3486
Orientation error (median)	4.8821
Orientation error (std)	0.6029
Orientation error (min)	$2.5317 \cdot 10^{-5}$
Orientation error (max)	15.3799

Table 4. Errors in the yaw measurements. The results are expressed in degrees.

Finally, we compared the track followed by the drone and the desired one and analyse the error. Although the main goal of this work is to develop a robust, scale-aware estimation system, the performance of the included PID controller will be analysed too. The next figure displays the performance of the system using all models versus the desired track for this experiment.

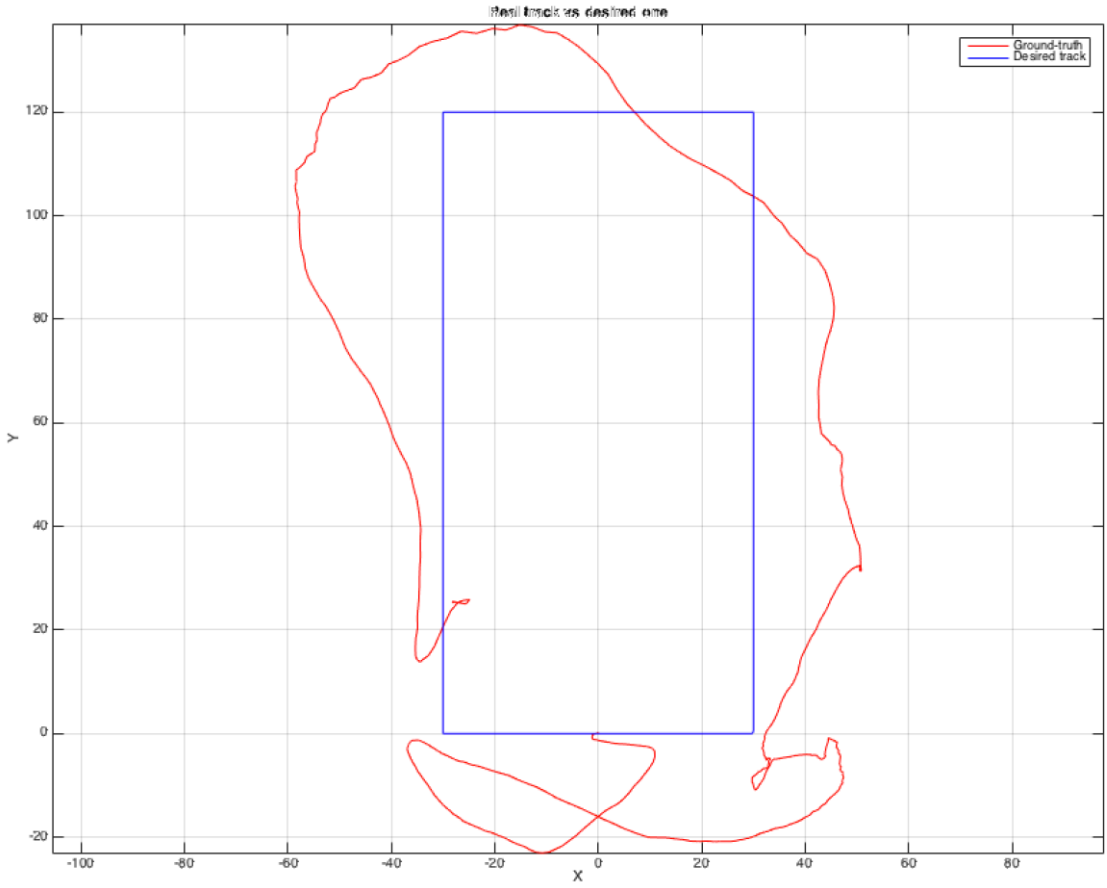


Fig. 57. Followed vs. desired tracks in red and blue respectively.

Even when the error induced by the delays was drastically reduced –as explained before– it cannot be completely eliminated. Due to it and because of its fast dynamics the drone cannot follow exactly the desired track, but it can reach a series of goal points. The next chart explains the errors committed between the followed track and the one that the MAV should have followed.

	All models with ORB-SLAM
Compared pose pairs	3411
Orientation error (RMSE)	12.7408
Orientation error (mean)	10.0462
Orientation error (median)	8.5733
Orientation error (std)	2.7175
Orientation error (min)	$7.8382 \cdot 10^{-14}$
Orientation error (max)	28.5918

Table 5. Error between the desired track and the followed by the drone, given in centimeters.

The RMSE error is less than 13cm, so the drone can follow appropriately the marked path even with the disadvantages associated to the using of low-cost MAVs. It could be achieved due to a fast and accurate estimation system working with a delay correction algorithm.

8.3. Sensor fusion improvements

In this section the results achieved by the different models are presented. The results, displayed in the figures, make clear the need of data fusion. There is a lack of precision in the pose's estimation when the system is not using all the models of the EKF or even it may reach messy results –as in the case of using only the NAVDATA correction model–. All the figures are presented with the track of the ground truth and the desired path –the red crosses track and the magenta line, respectively– for a better comprehension of the results. . It is important to remark that the exposed results –not only for this case, but the previous ones too– where only appears one line for each experiment does not means that only one flight was needed in order to record the data. The plotted data is the median of 5 flights in all the cases in order to compare in a better way the results and to know which algorithm causes more problems –as bad initializations–. It is displayed in this way to make every figure self-explanatory and clean.

In the first figure the performance of the system using only the predictor model is compared with the results achieved by the data fusion between the predictor model and the NAVDATA correction model –the green and blue lines, respectively–. The pose's estimation could be compared with the real position of the drone. Even though the estimation is not that bad –keeping in mind that in this case the system does not have a sensor reading that could bring a feedback– the error committed between the estimation and the real track is not admissible. If the prediction model is fused with the data from the NAVDATA correction model the performance is visible improved. The system lacks of a feedback in this case too, but the correction made by the NAVDATA model improves the results of the estimation. The results obtained in the five flights were similar between them for both experiments and all of the flights were successful.

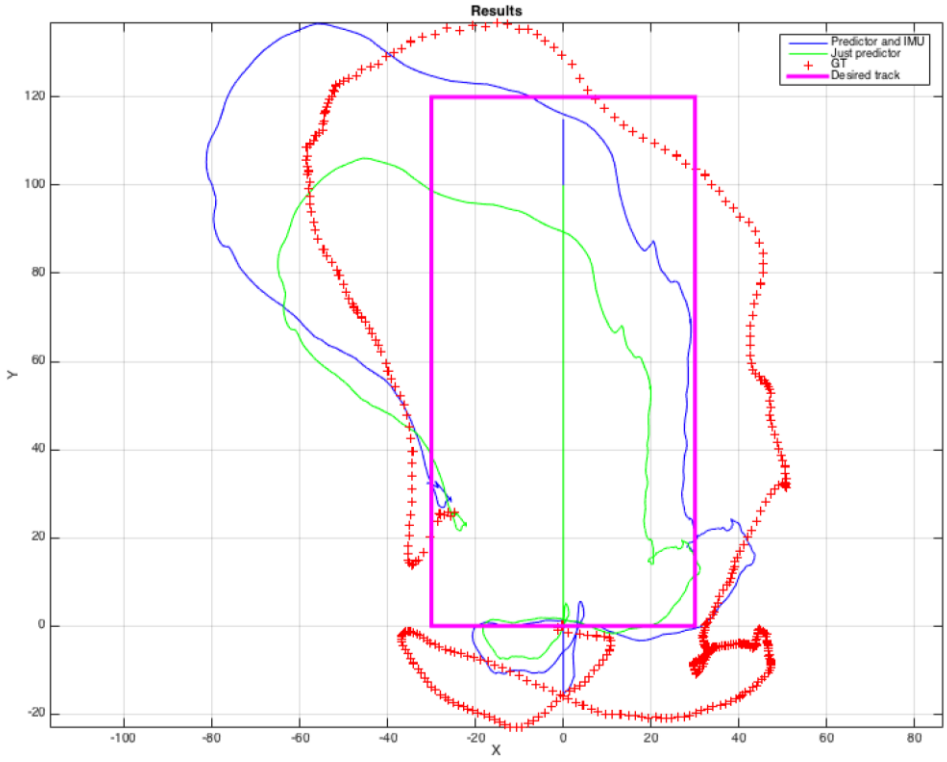


Fig. 58. Comparative of the results achieved when using just the predictor model with the results achieved using the predictor and NAVDATA correction models –blue and green line respectively–.

The results can be understood in a better way reading the errors presented in the next table. The performance improvement is clear just by looking at the figure and helps to understand the results presented in the table. The error between the estimation and the real track followed by the drone is clearly reduced –around 5cm– when fusing data.

	Only predictor model	Predictor and NAVDATA correction model
Compared pose pairs	228	332
Absolute translational error (RMSE)	16.9341	11.5263
Absolute translational error (mean)	14.2136	9.4269
Absolute translational error (median)	13.0264	8.7872
Absolute translational error (std)	9.2054	6.6324
Absolute translational error (min)	0.0158	0.0870
Absolute translational error (max)	39.5941	23.4986

Table 6. Comparison between the performance of the system using only the predictor model and the fusion of the predictor and the NAVDATA correction model.

In the following figure the performance of the system using only the VSLAM correction model is presented. This figure and the table helps to compare the results obtained using each of the VSLAM algorithms. The data sets collected from the five flights were similar for each experiment and both of them had one bad initialization, so the data from that flight was removed.

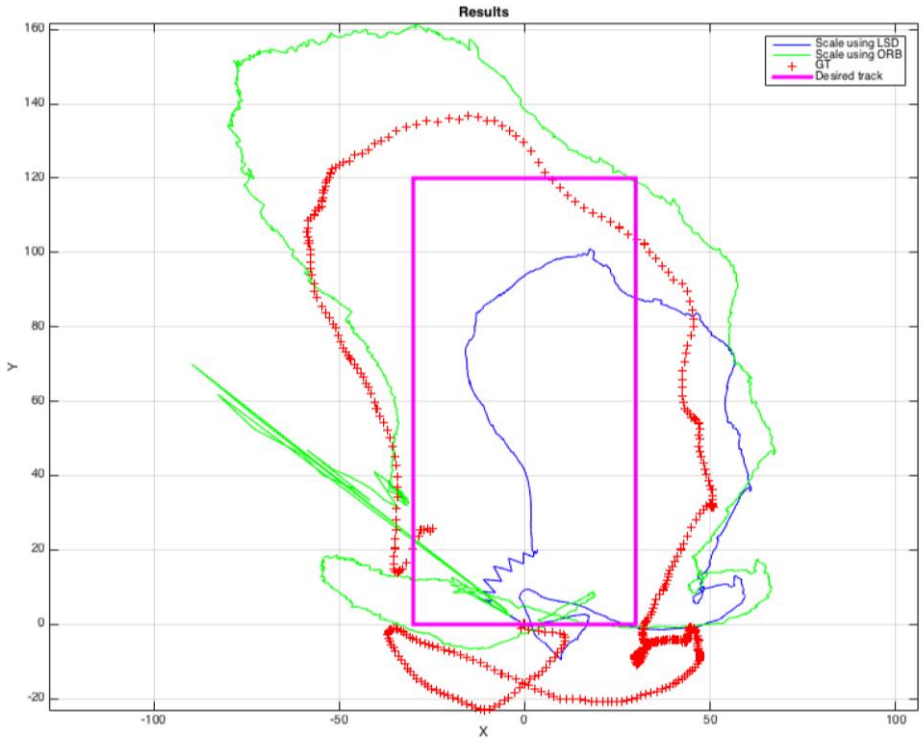


Fig. 59. Comparative of the results achieved when using just the VSLAM correction model with LSD-SLAM and ORB-SLAM algorithms –blue and green line respectively–.

There is a big difference between the scale estimated by each of the algorithms. It leads to a big difference in the depth estimation by the VSLAM algorithm and therefore in the errors committed by the system. The performance using ORB-SLAM is way better so was in the experiment made in the section 5.3.4. For this reason, the algorithm that will be used in the future for the experiments will be ORB-SLAM.

	Only VLSAM correction model using LSD-SLAM	Only VLSAM correction model using ORB-SLAM
Compared pose pairs	168	158
Absolute translational error (RMSE)	16.8754	10.7131
Absolute translational error (mean)	11.9105	8.4936
Absolute translational error (median)	8.4262	6.8763
Absolute translational error (std)	11.9548	6.5291
Absolute translational error (min)	0.0028	0.1222
Absolute translational error (max)	49.7062	29.8959

Table 7. Comparison between the performances of the system using only the VSLAM correction model putting to use each of the VSLAM algorithms studied in this work.

For the last experiment, all the previous recorded data sets are plotted together. Is easier to understand the improvement of the performance just by looking at the next figure.

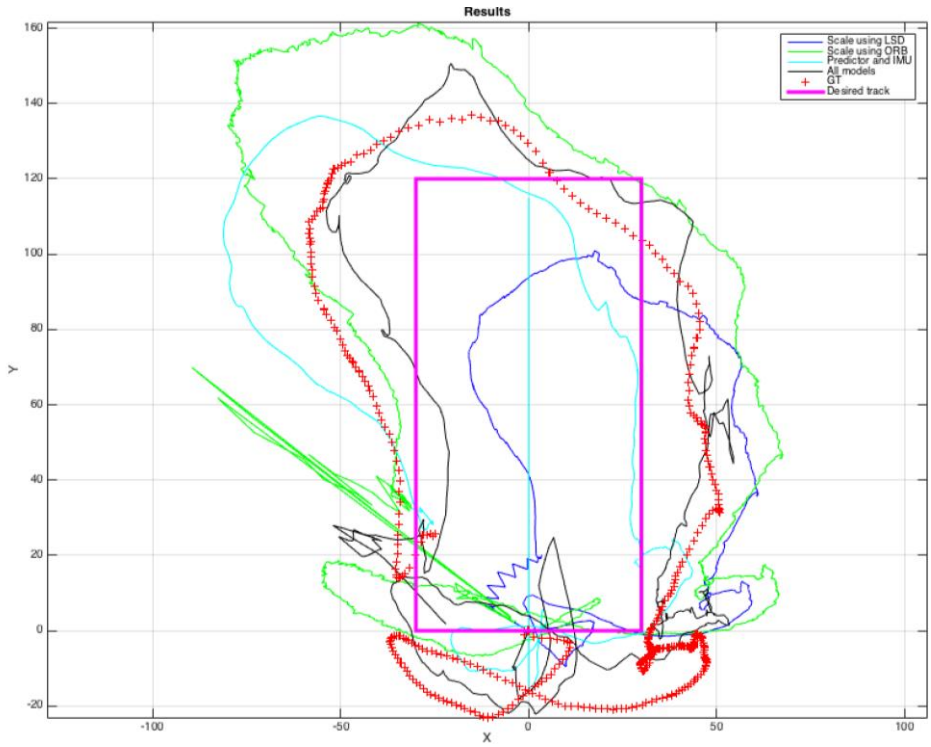


Fig. 60. Comparative of the results achieved when using just the VSLAM correction model with LSD-SLAM and ORB-SLAM algorithms –blue and green line respectively–; the predictor and NAVDATA correction models –in cyan– and all the models together –in black–.

The results presented in the next table demonstrate the improvement of the performance by data fusion. The estimation gets more accurate when fusing the data from the models previously explained. The results obtained in the previous experiment with the predictor and NAVDATA correction models together are improved by adding a correction model from a sensor that bring measurements that can be read as feedback. The errors are compared with the best results obtained in the previous experiments. The error is drastically reduced to the half, getting an RMSE error of around 5cm. It makes this system an accurate and robust estimation algorithm.

	All models	Only VLSAM correction model using ORB-SLAM	Predictor and NAVDATA correction model
Compared pose pairs	346	158	332
Absolute translational error (RMSE)	5.3248	10.7131	11.5263
Absolute translational error (mean)	3.9765	8.4936	9.4269
Absolute translational error (median)	3.0542	6.8763	8.7872
Absolute translational error (std)	3.5413	6.5291	6.6324
Absolute translational error (min)	0.0105	0.1222	0.0870
Absolute translational error (max)	14.8238	29.8959	23.4986

Table 8. Validation of data fusion. Errors given in centimeters.

CHAPTER 9: CONCLUSIONS AND FUTURE WORK

After all the processes presented in the previous chapters we accomplish the conclusions that will be explained below. After it, the main future work lines that could be achieved are discussed.

9.1. Conclusions

This work demonstrates that it is possible to use a commercial and low-cost drone as the hardware platform of a pose estimation and motion control system. The Bebop drone is a new, stable and reliable MAV recently released. Its firmware is also periodically reviewed and updated by their creators, so it is in a continuous improvement. It also implements a ROS driver that allows the MAV to communicate with a computer via Wi-Fi. On the other hand, the VSLAM algorithms that have been chosen to study in this thesis were also recently developed so their performance is the best between each monocular VSLAM method. Therefore, both the hardware architecture and the VSLAM techniques put to use are in the state-of-the-art.

It is important to remark the importance of the data fusion in order to achieve the performance of the current system. It not only improves the accuracy of the pose estimation in our case, but also prevents from errors caused by drifting and scale dynamic calculation. The fact of adding a sensor such as a monocular camera that brings a feedback of the position helps to the system in the calculation of the estimation, correcting it when models as the predictor or the NAVDATA correction –both sensitive to drift errors– are failing in that estimation.

The implementation of an EKF not only helps with the previously commented facts but also reducing the errors caused by the delays. In our case, the main source of these problems is the communication channel. The communication channel between the ground station and the chosen model of drone is a Wi-Fi network with delays that lay between 60 and 200ms, depending on the bandwidth used by nearby wireless LAN networks. Thanks to the EKF, the pose estimation and motion control errors caused by the delay was drastically reduced.

Despite the previously commented improvements implemented on the system, there is a main weakness on this work that is the performance of the motion controller. It is caused because the error produced by the delay is not completely removed, in addition with the fast dynamics of a MAV. It leads to a system that could achieve results as presented in Chapter 8 related with the motion control. Another limitation of the system developed in this work is the real scale calculation for the pose and environment map. As explained in the previous chapter, it needs to be launched from the floor, and it also needs the VSLAM algorithm to be ready and estimating the drone's pose.

9.2. Future work

In the future, the software architecture will be implemented over another hardware platform. The chosen MAV –the Erle-Copter[7]– has a developed driver for ROS and is able to carry up to 1kg. It allows the drone to carry an onboard computer and other sensors such as a scanner laser rangefinder. The computer permits the drone to perform the whole system onboard without having to communicate it to a ground-station. It avoids the delays caused by the wireless network, solving the main problem of the current system. The only delays that the system will face would be the ones caused by the computational times, which are insignificant for it –even if they produce any error it could be reduced by the developed algorithm included in the EKF node–.

Another advantage related to the change of hardware platform will be the capability of the drone to carry heavy sensors such as Hokuyo's. The main goal of the ISLAMAV project is to develop a MAV-based inspection system able to recognize indoor ruined or semi-ruined buildings in the context of rescue mission. This system will fuse the data from the models put to use in this work with a laser correction model. The measurements from a laser are faster and more robust facing fast and/or pure rotational movements than the computed by a monocular VSLAM model. It will also provide a 2D map of the environment required for the future development of the MAV's autonomy. Furthermore, the data fusion will improve the pose estimation of the system –as explained before– and add some recursion to it.

The fact of having two sources that could bring a feedback to the system will allow us to develop an intelligent algorithm that could not only add both correction models to the estimation but also to choose between both of them depending on the situation. For example, if the MAV is flying in an environment with poor illumination or directly in the darkness it could detect it and temporally disable the VSLAM correction model. Or if the drone notices that the detectable objects of the environment are too far for the reliable performance of the laser the algorithm could temporally disable the laser correction model.

Finally, those improvements will allow us to test the system in adverse situations that could not be considered before.

[7] http://erlerobotics.com/blog/erle-copter-es/

CHAPTER 10: DIAGRAMS

Different diagrams that are important for the comprehension of the system are displayed in this chapter. The first two figures –*Fig. 61* and *Fig. 62*– are flowcharts of the developed nodes of the EKF and the PID controller, respectively. The black circle represents the start of the algorithm and the same circle surrounded by another circle the ending. The PID controller node has a certain start and end –when the drone has reached all the goal points– but this is not the case of the EKF node. The control loop of this node is infinite and it will not end until the user kills the process –pressing the buttons CONTROL+C in the terminal where the process is running–.

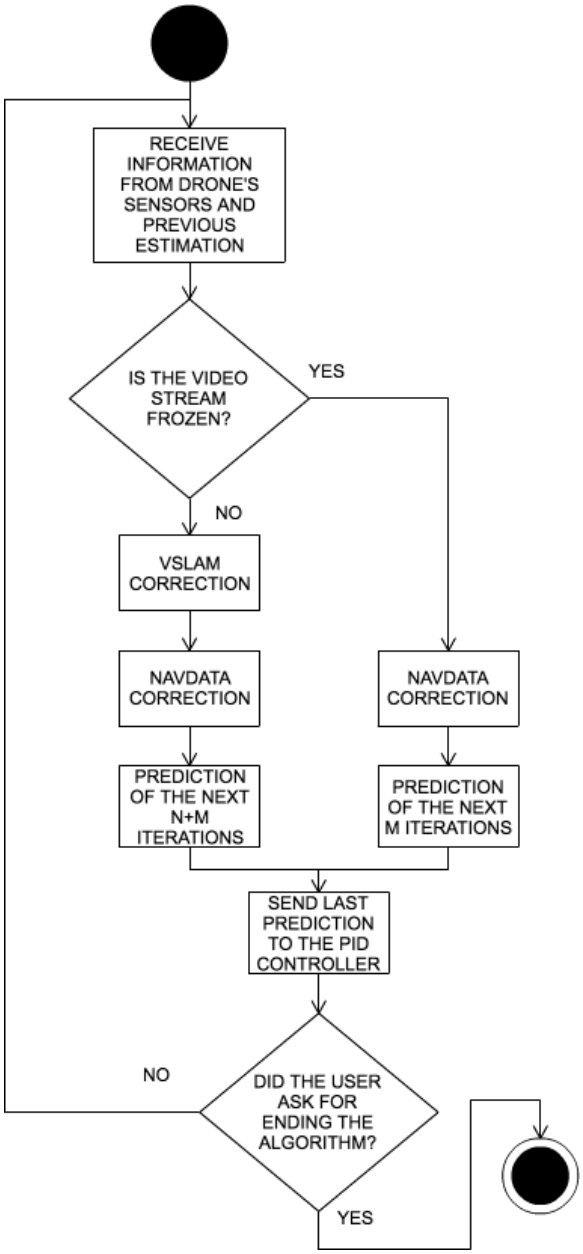


Fig. 61. Flowchart of the EKF node.

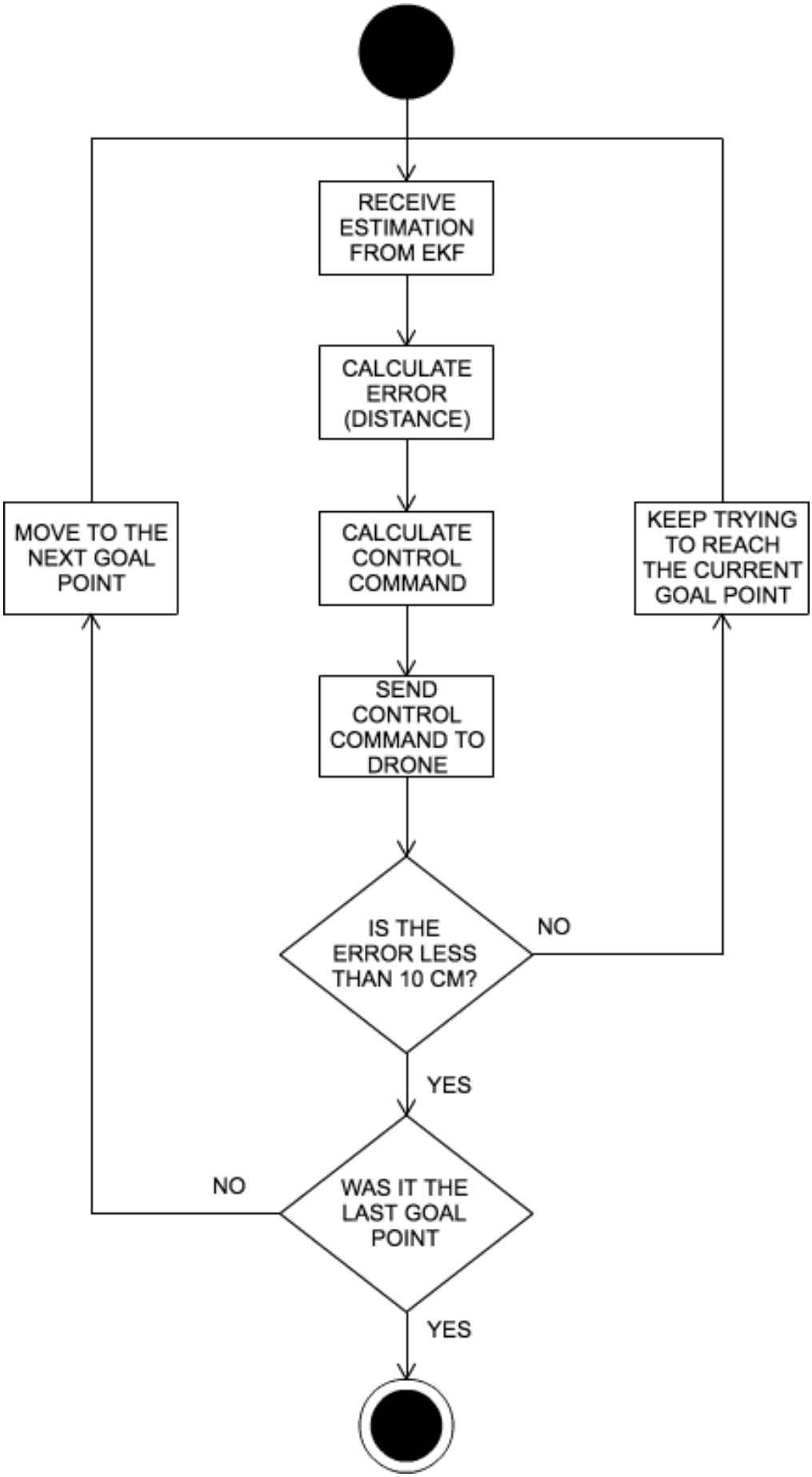


Fig. 62. Flowchart of the PID controller node.

Fig. 63 and *Fig. 64* are diagrams of the whole software architecture depending on the method used –the first one using LSD-SLAM and the second one ORB-SLAM–. These diagrams can be built by ROS with the command `rosrun rqt_graph rqt_graph`. The container rectangles symbolize packages and the others are topics that communicate two nodes. The round shapes are the nodes. The nodes are communicated between them by topics, whose connections are represented by arrows.

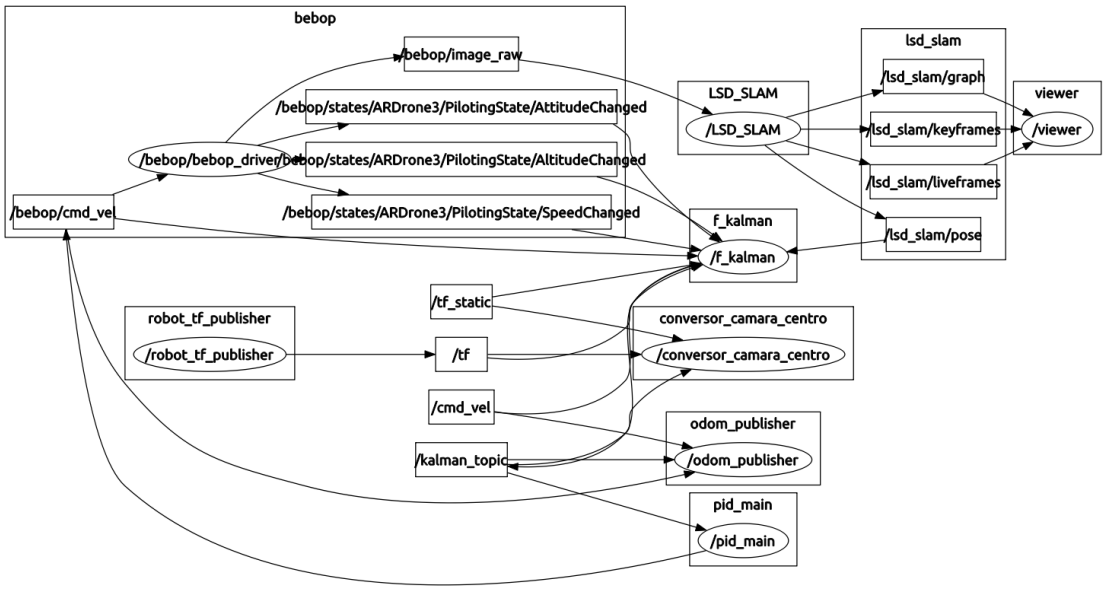


Fig. 63. System using LSD-SLAM nodes tree

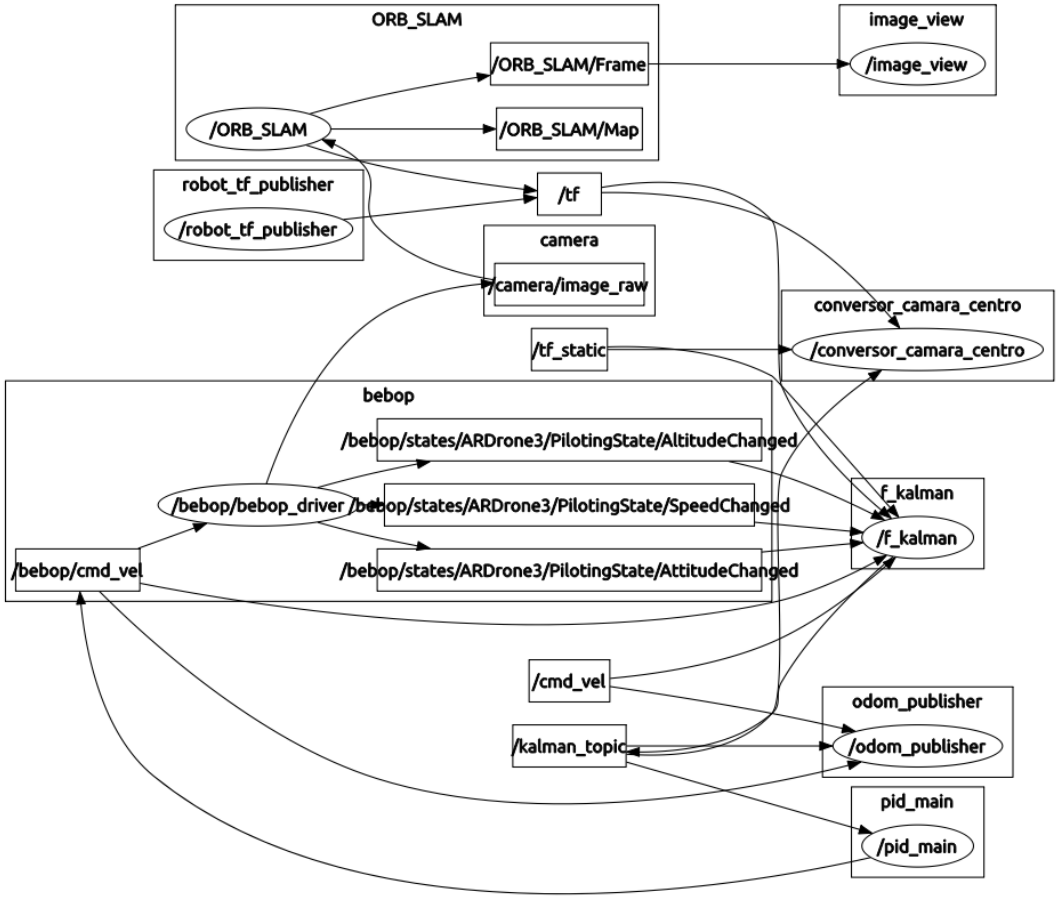


Fig. 64. System using ORB-SLAM nodes tree

CHAPTER 11: SPECIFICATIONS

For this project, the following conditions need to be satisfied:

- Computer using GNU/Linux 14.04 with approximately 4 Gb of free space and a Wi-Fi network card. The map visualization is not necessary for the performance of the developed system, but it needs a powerful computer –which minimum specifications are not specified– in order to display it. The following software tools are required:

  – ROS - full version.

    The next three items are ROS packages.

  – Ardrone_autonomy/bebop_autonomy.
  – LSD_SLAM and ORB_SLAM.
  – Rviz.

- Matlab software (optional) for displaying trajectories and measurements. *Robotic* toolbox will be compulsory. It would be necessary for the camera calibration of an optional ground-truth system. In this case the *Camera Calibration Toolbox* is needed.

- Bebop drone / AR.drone 2.0.

CHAPTER 12: 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.

12.1. 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 (euro)
Hardware	Bebop Drone	399	1	399
	AR.Drone 2.0	249	1	249
	ASUS Laptop	599	1	599
	IP Camera	79,88	1	79,88
	Switch D-Link	20	1	20
	POE injector	30,42	1	30,42
Hardware total cost				1377,3
Software	Ubuntu v14.04	0	1	0
	Robot Operating System	0	1	0
	ROS packages	0	1	0
	Matlab (Student edition)	69	1	69
	VLC	0	1	0
Software total cost				69
Equipment total cost				1446,3

Table 9. Equipment budget (VAT included), presented in euros.

12.2. 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	20	600	12000
Writing up	15	150	2250
Fees total cost			14250

Table 10. Professional fees (gross salary), expressed in euros.

12.3. Total cost

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

Equipment cost	1446,3
Professional fees	14250
Printing	90
Transport	500
Total	**16286,3**

Table 11. Total cost, expressed in euros.

CHAPTER 13: USER GUIDE

In order to use the system developed in this work it is mandatory to follow some steps that will set up the computer. Firstly, which tools have to be downloaded and installed are listed. It is detailed also how to install them and the commands needed to launch the system later. Then we present a brief explanation of how to install the nodes developed for this. The chapter ends with a guide of how to launch all the nodes in order to run the whole system.

13.1. Download the necessary tools

The next steps describe how to download some tools needed for the performance of the system:

1. Install ROS following the steps given in the official ROS website (http://wiki.ros.org/ROS/Installation). Create a workspace as explained in: http://wiki.ros.org/ROS/Tutorials/InstallingandConfiguringROSEnvironment.
2. Install bebop_autonomy or ardrone_autonomy following the steps given in the next websites: (http://bebop-autonomy.readthedocs.io/en/latest/installation.html / http://ardrone-autonomy.readthedocs.io/en/latest/installation.html).
3. Install LSD-SLAM following the steps given in the GitHub website (https://github.com/tum-vision/lsd_slam)
4. Install ORB-SLAM following the steps given in the GitHub website (https://github.com/raulmur/ORB_SLAM)

13.2. Install the nodes

Install the nodes developed for this work. Create a package (or two, if you want to separate both of them) as explained in: http://wiki.ros.org/ROS/Tutorials/CreatingPackage. Place the folders of each node (EKF and PID) into the src folder of the package. Move to the root folder and execute the order `catkin_make`.

13.3. Launching the nodes

This section guides the user along the commands that have to be launched in order to use the whole system:

13.3.1. Establishing the communication

First, it is necessary to establish the communication between the drone and the ground station. Make sure the MAV is on and connect the computer to its Wi-Fi network. Once it is connected, open a new terminal and source the setup file of the drone's driver (this step is only needed when using the Bebop drone due to its ROS driver can only be installed as a new workspace):

```
source ~/bebop_autonomy/devel/setup.bash
```

If you installed the ardrone_autonomy in the same workspace where you have all your packages and the source is always established for this workspace the previous step can be avoided. If it is not, write down the following order in a terminal:

```
source ~/your_workspace/devel/setup.bash
```

Now launch the ROS driver for the drone. If you are using the Bebop drone:

```
roslaunch bebop_driver bebop_node.launch
```

If you are using AR.Drone 2.0 (there are also some example launch files in the launch directory if you want to configure parameters):

```
rosrun ardrone_autonomy ardrone_driver
```

13.3.2. Launching the VSLAM algorithm

The following commands explain how to launch each VSLAM algorithm.

a) LSD-SLAM

We will start with LSD-SLAM. If you want to run the visualizer included in this package, write the following order:

```
rosrun lsd_slam_viewer viewer
```

The previous step can be avoided because it only displays the built map of the environment and the track of the camera but is not needed for the package's performance. Furthermore, it is the node that more computational requirements has, so if you are running the system in a computer with low performance do not use it. The next order launchs the LSD-SLAM estimation and tracking node:

```
rosrun lsd_slam_core live_slam /image:=<yourstreamtopic> /c
amera_info:=<yourcamera_infotopic>
```

Both drones put to use for this work provide a camera_info topic. When using this topic only the image dimensions and the K matrix from the camera info messages

will be used, hence the video has to be rectified. Alternatively, you can specify a calibration file using:

```
rosrun lsd_slam_core live_slam /image:=<yourstreamtopic> _c
alib:=<calibration_file>
```

In this case, the camera_info topic is ignored, and images may also be radially distorted. This work provides one calibration file for each drone, but if you want to calibrate your drone's camera on your own, follow the steps given in the next website: http://wiki.ros.org/camera_calibration.

For an improvement of the performance of this algorithm the amount of keyframes and the weight of each of them should be increased using the dynamic configuration:

```
rosrun rqt_reconfigure rqt_reconfigure
```

The parameters that should be changed are KFUsageWeight and KFDistWeight. A value of 6 for each of them may be enough. Now, the algorithm will take much more keyframes, what means a larger map that leads to more loop closures. Therefore, the algorithm performance will be slower. If your computer can handle it, the change makes the package to generally give much better and more robust results.

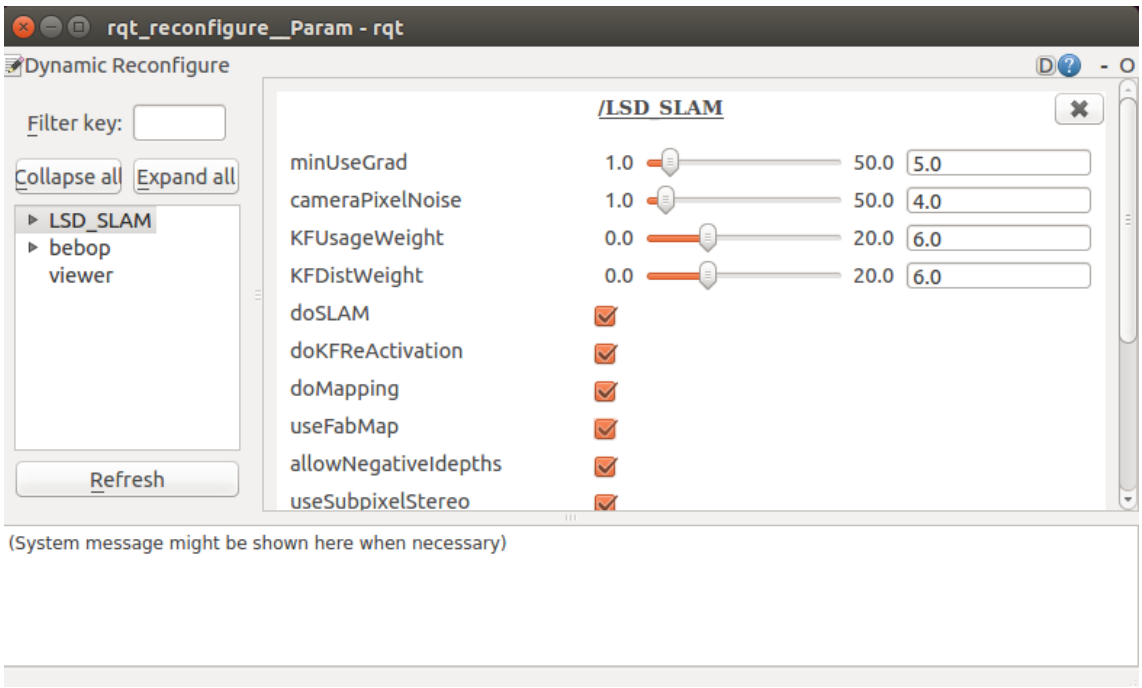


Fig. 65. Dynamic configuration.

b) ORB-SLAM

Here it is explained how to launch ORB-SLAM. There are two ways to launch this package. As before, it could be launched using a visualizer that is not mandatory for the performance of the package or it could be launched in a low-consuming way. The visualizer that this package uses is Rviz, a powerful software that provides some

interesting tools. ORB-SLAM could be launched customizing one launch file and adding it to the root folder (there are attached some of them in the package).

```
roslaunch ORB_SLAM your_launch_file.launch
```

The other way avoids the visualizer, economizing the computational consumption. Copy the next orders in two new terminals in order to launch it:

```
rosrun ORB_SLAM ORB_SLAM Data/ORBvoc.txt Data/ calibration_file.yaml

rosrun image_view image_view image:=/ORB_SLAM/Frame _autosize:=true
```

The file ORBvoc.txt is provided by the ORB_SLAM package (it is highly recommended to decompress the .tar.gz file in which it is contained for speeding up the performance of the algorithm). The calibration_file.yaml is provided by our system for each of the drone's cameras.

ORB-SLAM needs to extract an amount of features from the environment in order to create an initial point's map. Until this map is not built the package cannot start the tracking algorithm. Our system needs to start reading the pose estimations of the VSLAM from the floor in order to calculate correctly the scale (as explained in previous chapters), so ORB-SLAM may need an initialization stage. Move smoothly your drone backwards and forwards and also from left to right and vice versa without separating it from the floor. Do it until the package says that the initial map has been correctly built. The next figure makes a comparison of how looks the system initiated and not initiated:

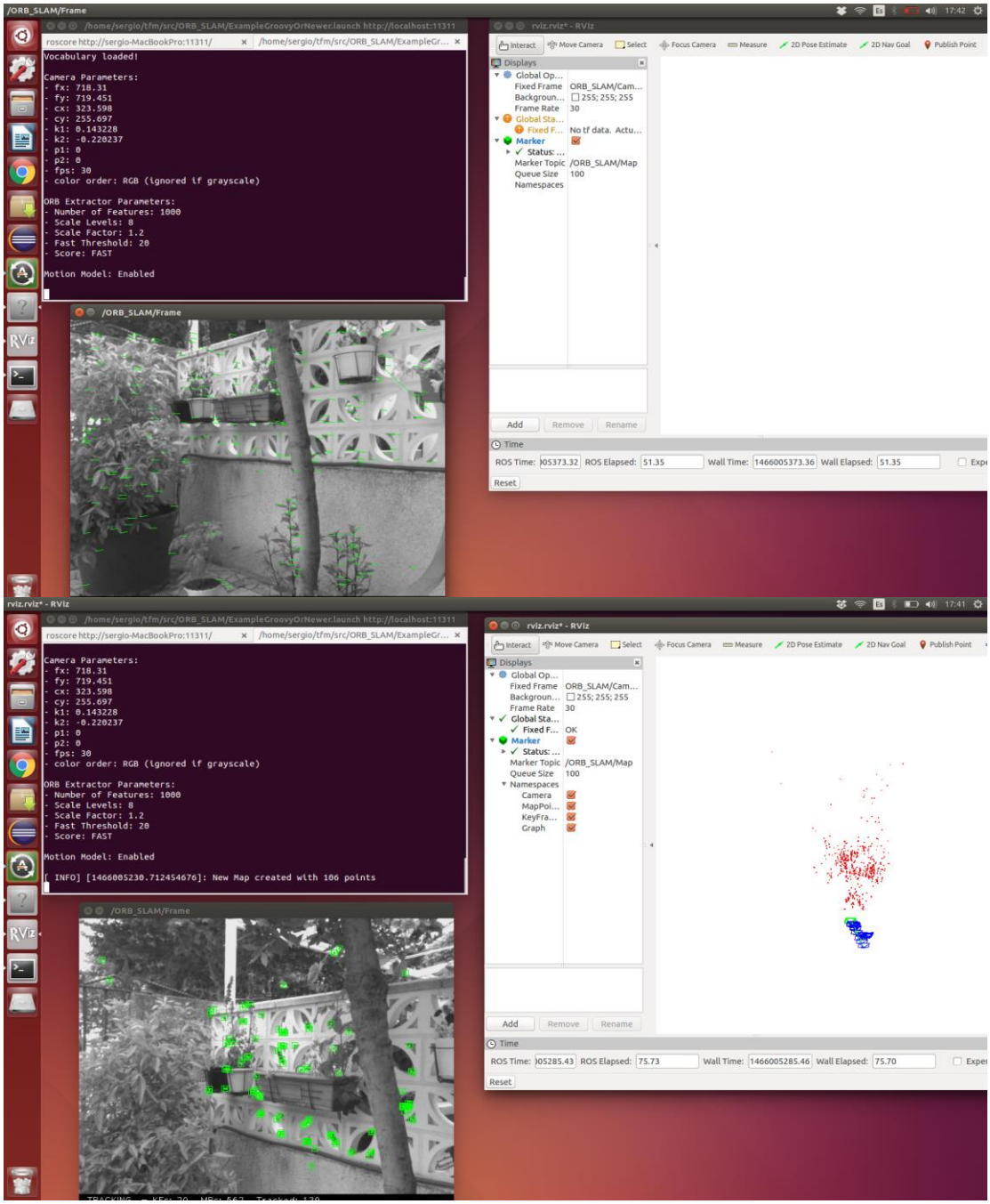


Fig. 66. Comparison of both stages of ORB-SLAM.

13.3.3. Launching the EKF node

Once there is a communication between the drone and the ground-station and the VSLAM algorithm is performing its estimation the EKF node can be launched. Write the following in a new terminal:

```
rosrun EKF EKF
```

The system will start to estimate the real position of the camera. The estimation is published in the topic /kalman_topic and can be displayed with the following

command:

```
rostopic echo /kalman_topic
```

An optional node can be launched too. It transforms the position of the frame from the camera to the centre of the drone. To launch it, use the following order:

```
rosrun EKF camera_frame_converter
```

The new estimation is published in /kalman_topic_rect, and can be displayed with the following command:

```
rostopic echo /kalman_topic_rect
```

13.3.4. Taking-off the drone

It is important to make a flat trim before the take-off in order to avoid problems. The flat trim process will send a request to the drone to re-calibrate its rotation estimates assuming that it is on a flat surface. Do not call this service while the drone is flying or while the drone is not actually on a flat surface. When using the AR.Drone 2.0, the driver ardrone_autonomy provides a service that we can call:

```
rosservice call ardrone/flattrim
```

But the service is not yet developed for the Bebop drone, so the flat trim must be performed by means of the free android application provided by bebop (*FreeFlight 3*). It could be performed just by clicking in the flat trim button indicated in the next figure:

Fig. 67. Free Flight 3 android application interface.

Once the flat trim is made make your drone to take-off. If you are using the Bebop drone copy the next command into a new terminal:

```
rostopic pub bebop/takeoff std_msgs/Empty
```

If you want to land it use:

```
rostopic pub bebop/land std_msgs/Empty
```

But if you are using an AR.Drone 2.0 use the following orders for the take-off and the landing:

```
rostopic pub /ardrone/takeoff std_msgs/Empty
rostopic pub /ardrone/land std_msgs/Empty
```

13.3.5. Launching the PID controller

Once the drone is flying and its current real-scale position is being estimated the PID controller could be launched. Type the following command in a new terminal:

```
rosrun PID PID_main
```

The drone will now fly along the pre-configured in the PID controller script series of points. When it reaches the last point it will fly down to an altitude of one meter and finally land.

BIBLIOGRAPHY

(Achtelik et al., 2011) Achtelik, Markus, Achtelik, Michael, Weiss, S., Siegwart, R.: Onboard IMU and Monocular Vision Based Control for MAVs in Unknown In- and Outdoor Environments. In: *Robotics and automation (ICRA)*. (2011).

(Achtelik et al., 2012) Achtelik, M., Lynen, S., Weiss, S., Kneip, L., Chli, M., Siegwart, R.: Visual-inertial SLAM for a small helicopter in large outdoor environments. In: *Proceedings IEEE Conference on Intelligent Robots and Systems (IROS)*, pp. 2651-2652. (2012).

(Ahrens et al., 2009) Ahrens, S., Levine, D., Andrews, G., How, J.P.: Vision-based guidance and control of a hovering vehicle in unknown, gps-denied environments. In: IEEE International Conference on Robotics and Automation (ICRA), pp. 2643-2648. (2009).

(Bachrach et al., 2012) Bachrach, A., Prentice, S., Ruijie, P., Huang, A., Krainin, M., Maturana, D., Fox, D. and RoyEstimation, N.: Planning and Mapping for Autonomous Flight Using an RGB-D Camera in GPS-denied Environments. In: *The International Journal of Robotics Research September 1*, 2012 31:1320-1343. (2012).

(Brand et al., 2014) Brand, C., Schuster, M., Hirschmuller, H., and Suppa, M.: Stereo- vision based obstacle mapping for indoor/outdoor SLAM. In: *Intelligent Robots and Systems (IROS 2014), 2014 IEEE/RSJ International Conference*, pages 1846–1853. (2014).

(Bristeau et al., 2011) Bristeau, P-J., Callou, F., Vissiere, D., Peit, N.: The navigation and control technology inside the AR.Drone micro UAV. In: *18th IFAC World Congress*, pp. 1477-1484. (2011).

(Brockers et al., 2014) Brockers, R., Humenberger, M., Weiss, S., Matthies, L.: Towards autonomous navigation of miniature UAV. In: *2014 IEEE Conference on Computer Vision and Pattern Recognition Workshops*. (2014).

(Bylow et al., 2013) Bylow, E., Sturm, J., Kerl, C., Kahl, F., Cremers, D.: Real-time camera tracking and 3D reconstruction using signed distance functions. In: *Proceedings of Robotics: Science and Systems (RSS)*. (2013).

(Engel et al., 2014 a) Engel, J., Schöps, T., Cremers, D.: LSD-SLAM: Large-scale direct monocular SLAM. In: *Computer Vision (ECCV 2014)*, pp.834-849. (2014).

(Engel et al., 2014 b) Engel, J., Sturm, J., Cremers, D.: Scale-Aware Navigation of a Low- Cost Quadrocopter with a Monocular Camera. In: *Robotics and Autonomous Systems (RAS)*, volume 62, 2014. (2014).

(Engel, 2011) Engel, J: Autonomous Camera-Based Navigation of a Quadrocopter. Master's thesis, TUM. (2011).

(Forster et al., 2014) Forster, C., Pizzoli, M., Scaramuzza, D.: SVO: Fast Semi-Direct Monocular Visual Odometry. In: *IEEE International Conference on Robotics and Automation (ICRA)*. (2014).

(Fraundorfer et al., 2012) Fraundorfer, F., Heng, L., Honegger, D., Lee, L., Meier, L., Tanskanen, P.,Pollefeys, M.: Vision-based autonomous mapping and exploration using a quadrotor MAV. In: *Proceedings IEEE Inter. Conference on Intelligent Robots and Systems (IROS)*. (2012).

(Galton, 2009) Galton, A.: Autonomous Flight in Unstructured and Unknown Indoor Environments. Master's thesis, MIT. (2009).

(Grzonka et al., 2009) Grzonka, S., Grisetti, G., Burgard, W.: Towards a navigation system for autonomous indoor flying. In: *Proceedings IEEE Intelligent Conference on Robotics and Automation (ICRA)*. (2009).

(Huang et al., 2011) A.S. Huang, A. Bachrach, P. Henry, M. Krainin, D. Maturana, D. Fox and N. Roy: Visual odometry and mapping for autonomous flight using and RGB-D camera. In: *Proceedings IEEE Intelligent Symposium of Robotics Research (ISRR)*. (2011).

(Johnson et al., 2008) Johnson, N.G.: Vision-assisted control of a hovering air vehicle in an indoor setting. Master's thesis, BYU. (2008).

(Kemp, 2006) Kemp, C.: Visual Control of a Miniature Quad-Rotor Helicopter. PhD Thesis, Churchill College, University of Cambridge. (2006).

(Klein et al., 2007) Klein, G., Murray D.: Parallel Tracking and Mapping for Small AR Workspaces. In: *Proc. International Symposium on Mixed and Augmented Reality (ISMAR'07, Nara)*. (2007).

(Kohlbrecher et al., 2011) Kohlbrecher, S., Stryk, O., Meyer, J., Klingauf, U.: A Flexible and Scalable SLAM System with full 3D Motion Estimation. In: *Int. Conference Safety Security and Rescue Robotics (SSRR)*, pp. 155-160. (2011)

(Kushleyev et al., 2012) Kushleyev, A., Mellinger, D., Kumar, V: Towards a swarm of agile micro quadrotors. In: *Proceedings of Robotics: Science and Systems (RRS)*. (2012).

(Lindsey et al, 2011) Lindsey, Q., Mellinger, D. Kumar, V.: Construction of cubic structures with quadrotor teams. In: *Proceedings on Robotics: Science and Systems (RSS)*. (2011).

(López et al., 2015) 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. In: *Robot 2015: Second Iberian Robotics Conference: Advances in Robotics, Volumen 1*. (2015).

(Lourakis et al., 2013) Lourakis, M., Zabulis, X.: Accurate Scale Factor Estimation in 3D Reconstruction. In: *Volume 8047 of the series Lecture Notes in Computer Science*, pp 498-506. (2013).

(Mellinger et al., 2011) Mellinger, D. Kumar, V.: Minimum snap trajectory generation and control for quadrotors. In: *Proc. IEEE Intelligent Conference on Robotics and Automation (ICRA)*. pp. 2520-2525. (2011).

(Mur-Artal et al., 2015) Mur-Artal, R., Montiel, J., Tardos, J.: ORB-SLAM: A Versatile and Accurate Monocular SLAM System. In: *IEEE Transactions on Robotics*, vol. 31, no. 5, pp. 1147-1163. (2015).

(Newcombe et al., 2011) Newcombe, R., Lovegrove, S. and Davison, A.: DTAM: Dense Tracking and Mapping in Real-Time. In: *ICCV*. (2011).

(Nützi et al., 2011) Nützi, G., Weiss, S., Scaramuzza, D., Siegwart, R.: Fusion of IMU and Vision for Absolute Scale Estimation in Monocular SLAM. In: *Journal of Intelligent & Robotic Systems*. January 2011, Volume 61, Issue 1, pp 287-299. (2011).

(Prentice, 2008) Prentice, S., Roy, N.: The Belief Roadmap: Efficient Planning in Linear POMDPs by Factoring the Covariance. In: *Proceedings of the 12th International Symposium of Robotics Research (ISRR)*, Y. Nakamura (ed.), Hiroshima, Japan. (2008).

(Senanayake, 2016) Senanayake, M., Senthooran, I., Barca, J C., Chung, H., Kamruzzaman, J. and Murshed, M.: Search and Tracking Algorithms for Swarms of Robots, a Survey. In: *Journal of Robotics and Autonomous Systems*, pp. 422-434. (2016).

(Teulière et al., 2015) Teulière, C., Marchand, E.,Eck, L.: 3-D Model-Based tracking for UAV Indoor Localization. In: *IEEE Transactions on cybernetics*, vol. 45, no5. (2015).

(Tournier et al. 2006) Tournier, G.P., Valenti, M., How, J.P., Feron, E.: Estimation and control of a quadrotor vehicle using monocular vision and moiré patterns. In: *Proceedings of AIAA GNC*, Keystone, Colorado. (2006).

(Vago et al., 2015) Vago, L., Santos, A. and Sarcinelli-Filho, M.: Outdoor Waypoint Navigation with the AR.Drone Quadrotor. In: *International Conference on Unmanned Aircraft Systems (ICUAS)*. (2015).

Ascending Technologies GmbH, website:
http://www.asctec.de.

Camera Calibration Toolbox for Matlab:
http://www.vision.caltech.edu/bouguetj/calib_doc/index.html

Erle robotics website:
http://erlerobotics.com/blog/erle-copter-es/

Parrot website:
http://www.parrot.com/products/bebop-drone/

RGB-D SLAM Dataset and Benchmark, website:
http://vision.in.tum.de/data/datasets/rgbd-dataset

Robot Operating System, website:
http://www.ros.org/

Wikipedia, website:
https://en.wikipedia.org

APPENDIX: ADDITIONAL ACTIVITIES

In order to achieve the required knowledge for the subject "Introducción al TFM" some additional activities were performed. In the draft, we proposed some of them as:

1. Approach of the thesis using the scientific method. Documentation about the scientific methodology was studied in order to suggest a hypothesis. The experiments and tests performed for the validation of this hypothesis are presented in this work.
2. Development of a complete state of the art related with the technologies and techniques included in the thesis. The state of the art is included in the thesis. Furthermore, documentation about the use of bibliographic sources in technique investigation was studied.
3. Writing of a scientific paper, which will be sent to international conference in the field of robotics. The accepted paper is attached in its corresponding section along with the written proof of conference attendance.
4. Implementation of all the developed algorithms over a real platform that allows to validate the obtained results, using for it an experimental methodology. The methodology is explained along the work and the results exposed represent real information obtained from the MAV's fly instead from a simulation.
5. Attendance to conferences and lectures that are interesting for the development of the work.

1. Approach of the thesis using the scientific method

The information regarding the use of the scientific method studied for this thesis was obtained from the following links:

a) http://www.aulafacil.com/cursos/t639/ciencia/investigacion/ciencia-ymetodo-cientifico

This link contains information about a course named "Science and Scientific Method". It gives some definition of basic terms and rules that define them. It starts with the definition of what is a method and what is the science. Then, the information from both definitions and rules is fused and gives as a result an explanation about the scientific method. The process of it is presented as a series of stages that should be reached, according to Bunge, M. (1989), "*La investigación científica"*:

I. Descubrimiento de una deficiencia en las teorías anteriores o de un problema.
II. Planteamiento preciso de la cuestión.
III. Búsqueda de conocimientos relevantes al problema, tales como datos empíricos y de técnicas de medición.
IV. Tentativa de solucionar el problema con ayuda de los medios identificados.
V. Invención de nuevas ideas o producción de nuevos datos empíricos que puedan ayudar a resolver el problema.

VI. Utilización del instrumento conceptual o empírico disponible para la obtención de una solución del problema.
VII. Investigación de las consecuencias de la solución obtenida.
VIII. Confrontar la solución con la totalidad de las teorías y con la información empírica pertinente.
IX. Análisis de la corrección de todo el proceso seguido.

A part of the previous proposed steps is the formulation of the hypothesis. Thus, the definition of it is presented. They are definded as follows: "Las hipótesis, por tanto, son tentativas de explicación de los hechos y fenómenos a estudiar, que se formulan al comienzo de una investigación mediante una suposición o conjetura verosímil destinada a ser probada por la comprobación de los hechos". Some aspects that should be kept in mind when formulating the hypothesis are listed. The author of the course then explains that the person that follows the scientific method cannot just formulate and validate a series of unrelated hypothesis but that person must construct one or more theories that guide the research and fuse the information from the validated hypothesis. Then, the basic objectives of these theories, that help to differentiate between the scientific and the non-scientific are listed, according to Bunge.

Finally, the author explain in detail different models of the scientific method followed by the point of view of different authors (Popper, Kuhn, Lakatos, etc.).

b) http://www.aulafacil.com/cursos/t679/ciencia/investigacion/investigacion

This link also contains an online course. It is called "Investigation". It also starts with some definitions of different terms as science, epistemology, the scientific method and the scientific perspective. Some features related with this definitions are listed, as in the case of the science. Once the term of science is defined, it makes a differentiation between the factual science (the one that study objective facts that occur in the nature) and the formal science (that are concerned with characterizing abstract structures described by sign systems). The scientific method is explained as in the previous course and it conducts to the definition of the scientific perspective. The author defines the scientific investigator as follows: "El investigador científico es un individuo que aplica procedimientos formales, sistemáticos, para obtener información acerca de algún aspecto que le interesa de la realidad". There is a differentiation between the methods of obtaining knowledge making a reference to the previous course, which explains the difference between the regular knowledge and the scientific knowledge. The first one is imprecise, subjective and lacks of a method while the second one has been obtained by means of a scientific method and could be tested again in order to improve it.

The next points of the course talk about the design of an investigation project. It explains the main objectives that compound this kind of projects. Next to it, how a research should be developed and the basics of the writing of an investigation work are explained. Finally, an example of a report is presented.

2. Development of a complete state of the art

Some documentation about the use of bibliographic sources was studied. The following links contain information related to it:

a) http://mtu-pnp.blogspot.com.es/2013/07/la-investigacionbibliografica.html

The link of this section has information about the bibliographic investigation. It starts defining what is an investigation problem, which is the reason that motive the investigation itself. The authors explain that is the first step in the sequence: problem → investigation → solution. Then they continue talking about the features of the bibliographic investigation. The first of this features is the definition, and they present some examples of this feature. I would like to highlight this one: "el proceso de búsqueda de información en documentos para determinar cuál es el conocimiento existente en un área particular". The authors of the website then remark the importance of the field that they are talking about. Next to it, a list of different kind of publications appears in this order: paper, treatise, monograph, journal and annual. They talk then about the methodology of making an enquiry (when the investigator is starting to learn about some field) and after it about the methodology of the bibliographic investigation. It explains how to collect, select, incorporate and organize the references.

b) http://tecnicasdeinvestigacion.blogspot.com.es/2010/05/fuentesbibliograficas.html

The web related with the link of this section talks about the bibliographic sources. The author of the web differentiate between three types of sources: primary, secondary and tertiary. The first ones are the kind of sources that contain new information. Some examples of them are: books, publications, thesis, etc. The secondary sources organize the information about the primary sources in the form of summaries or index. It allows the user to obtain and use the information contained in the primary sources. The secondary sources are called too "reference manuals". Examples of secondary sources are: bibliographies, dictionaries, etc. The last kind of sources are the tertiary ones. They collect secondary documents in order to guide the user to secondary and primary sources making easier the location of the information. Directories, catalogues and internet itself are examples of this kind of sources.

3. Writing of a scientific paper

In this section a text which talks about what is a scientific article and how to write one is explained. Then, the paper exposed for the ICARSC'2016 conference is detailed.

a) http://www.ugr.es/~filosofia/recursos/innovacion/convo-2005/trabajo-escrito/como-elaborar-un-articulo-cientifico.htm

The text of the link starts defining a scientific article as a written and published report that describes original results of an investigation. It is important to remark that this kind of articles are not just summaries which the author should keep but a report that is clear enough so third persons could understand its message.

The author of the text continues listing some features that a scientific article should reach. Different schemes that could be followed when writing the article are explained (in this work we have followed the second scheme, including the conclusions into the discussion section). The rules that must be kept in mind for the development of a scientific article are listed, which were considered when writing our article –excepting the way of mark the bibliographic references, as we used the IEEE style–. Finally, the

principal sections of a proper article are listed and described in detail.

b) "Indoor SLAM for Micro Aerial Vehicles Control using Monocular Camera and Sensor Fusion"

The results obtained in this thesis were sent as a paper to two conferences: the IEEE International Conference on Autonomous Robot Systems and Competitions ICARSC 2016[8], in Bragança (Portugal) and the Workshop on Autonomous Vehicles in Off-Road Scenarios within the 2016 IEEE Intelligent Vehicles Symposium IV'16[9], Gothenburg (Sweden). It was accepted for both of them, but we chose to present it in the ICARSC'2016. We declined to present it in the IV because the paper was accepted for a workshop (Workshop on Autonomous Vehicles in Off-Road Scenarios) and we could present the work in a conference itself at the ICARSC. Both of the symposiums were part of the IEEE program.

The data of the publication are:

Authors: Sergio García, M. Elena López, Rafael Barea, L. Miguel Bergasa, Alejandro Gómez and Eduardo J. Molinos
Title: Indoor SLAM for Micro Aerial Vehicles Control using Monocular Camera and Sensor Fusion
Publication: Proceedings of the IEEE International Conference on Autonomous Robot Systems and Competitions ICARSC 2016 (ISBN: 978-1-5090-2255-7)
Date: May 2016

Firstly, the camera ready version of the work is presented. After it, the written proof of assistance is attached. It is important to remark that I have attended to most of the lectures given in the conference (since the Wednesday 4th of May to the Friday 6th). It allowed me to learn about robotics from other points of view so new ideas could be added to my work (new hardware platforms, recently-developed VSLAM algorithms, other methods for scale calculation, etc.).

[8] http://icarsc2016.ipb.pt/
[9] http://iv2016.org/

Indoor SLAM for Micro Aerial Vehicles Control using Monocular Camera and Sensor Fusion

Sergio García, M. Elena López, Rafael Barea, Luis M. Bergasa, Alejandro Gómez, Eduardo J. Molinos

Electronics Department
University of Alcalá
Alcalá de Henares (Madrid), Spain
sergio.garciagonzalo@edu.uah.es, elena@depeca.uah.es

***Abstract*— This paper represents research in progress in Simultaneous Localization and Mapping (SLAM) for Micro Aerial Vehicles (MAVs) in the context of rescue and/or recognition navigation tasks in indoor environments. In this kind of applications, the MAV must rely on its own onboard sensors to autonomously navigate in unknown, hostile and GPS denied environments, such as ruined or semi-demolished buildings. This article aims to investigate a SLAM technique that fuses visual information and measurements from the inertial measurement unit (IMU), to robustly obtain the 6DOF pose estimation of a MAV within a local map of the environment. The monocular visual SLAM algorithm along with the IMU calculate the pose estimation through an Extended Kalman Filter (EKF). The system consists of a low-cost commercial drone and a remote control unit to computationally afford the SLAM algorithms using a distributed node system based on ROS (Robot Operating System). Some experimental results show how sensor fusion improves the position estimation and the obtained map under different test conditions.**

Keywords—micro aerial vehicles; indoor navigation; sensor fusion; simultaneous localization and mapping; robot operating system

I. INTRODUCTION

Micro Aerial Vehicles (MAVs) have been widely used in various areas ranging from military to civilian domains, including surveillance operations, weather observation, disaster relief and civil engineering inspections. Enabled by GPS and MEMS inertial sensors, MAVs that display an impressive array of capabilities in outdoor environments have been developed [1,2,3]. Unfortunately, most indoor environments remain without access to external positioning systems, and autonomous MAVs are very limited in their ability to operate in these areas.

The two main challenges of indoor MAV navigation are the denied reception of GPS signal and the constraints of the indoor aerial platforms. Unlike the conventional GPS/INS based navigation in which the MAV global position and velocity are directly obtained, indoor navigation needs to get these information by sophisticated algorithms based on relative sensing. Besides, indoor MAVs are usually designed to be small and having very limited payload, and this results in limited onboard computational power which makes the algorithms even harder to be implemented.

Especially, pose estimation is essential for many navigation tasks, including localization, mapping and control. The technique used depends mainly on the available on board sensors, which in aerial navigation must be carefully chosen due to payload limitations. Through their low weight and consumption, most commercial MAVs incorporate at least one monocular camera, so VSLAM (Visual SLAM) techniques have been widely used [4, 5]. However, most of these works have been limited to small workspaces that have definite image features and sufficient sunlight. Furthermore, computational time is too high for the fast dynamics of aerial vehicles, making difficult to control them. On the other hand, despite their greater weight and consumption, range sensors such as RGB-D cameras or laser range sensors have also been used on MAVs due to their fast distance detection.

The work presented in this paper is part of the ISLAMAV project –develop by the *RobeSafe* Group of the Electronics Department of the University of Alcalá– whose final objective is to fuse several sensors to improve the pose estimation for MAVs in indoor environments. As a strategy of the fusion algorithm, each of the sensors must be able to provide its own pose estimation to endow the system with some redundancy that allows it to work in different environmental conditions. In [6] we presented the whole architecture –which includes laser, vision and inertial sensing-, while in this paper we focus only on monocular camera and IMU fusion.

To face the computational requirements, the system is composed of a flight and a ground unit, so that code can be distributed in different nodes using ROS (Robot Operating System).

The study explained in this paper uses two monocular VSLAM algorithms to calculate the pose estimation (along with the measurements from the IMU) and the map of the environment: LSD-SLAM [7] and ORB-SLAM [8].

One of the main problems of monocular camera VSLAM algorithms is the fact that it cannot calculate the scale of the data of tracking and mapping. It leads to a system that is not working with real-scale data, what could affect the integrity of an aerial robot. To solve this problem, our system uses the data from the IMU to calculate the dynamic scale of the SLAM and return the real-time pose of the MAV without scale ambiguity.

The remaining part of this paper is organized as follows. Section 2 discusses related work. Section 3 describes the overall system. The SLAM approach is explained in section 4. The experimental results are presented in Section 5. Fi-

nally, it is followed by the conclusion and future work in Section 6.

II. Related Work

The most challenging part of SLAM for MAVs is to obtain the 6DOF pose of the vehicle without odometry information. To do this, different sensor sources have been suggested, such as laser range sensors [9], monocular cameras [4], stereo cameras [5] or RGB-D sensors [10].

Due to weight limitations (in addition to power consumption), most of the works only use the onboard camera and IMU to apply VSLAM (Visual SLAM) techniques [11,12,13,14,15,16,17]. These systems demonstrate autonomous flight in limited indoor environments using VSLAM techniques that are out-dated, what results in inaccurate estimations and poor control results. The work developed in [17] has been the main reference for our research. But VSLAM algorithm, hardware architecture and some other improvements –as the scale calculation method, or the ability to include another SLAM stage based on laser- have been implemented.

In this work, up-to-date VLSAM algorithms are fused with measurements from the IMU to solve the SLAM problem in complex indoor environments and robustly estimate the 6DOF pose of the MAV, using a distributed system with a flight unit and a ground station. Furthermore, the system is able to calculate the dynamic scale of the measurements, what makes it a scale-aware system. Due to it, the EKF and the control stage work with real scaled data, in contrast to other monocular VSLAM systems.

III. System Overview

We address the problem of autonomous indoor MAV localization as a software challenge, focusing on high-level algorithms integration rather than specific hardware. For this reason, we use a low-cost commercial platform with minor modifications and an open-source development platform (ROS), so that drivers of sensors and some algorithms can be used without development.

A. Hardware Architecture

Our quadrotor MAV, shown in *Fig. 1*, is the Bebop from Parrot [18], a lighter (400 gr) and smaller (33x38x3.6cm) drone than the earlier AR.Drone 2.0. This MAV can carry up to 200g of payload for about 5min and is equipped with a frontal "Fisheye" camera. It counts with another vertical camera, which is used for stabilization and horizontal velocity estimation. Besides, it has an ultrasonic altimeter, a 3-axis accelerometer, 2 gyroscopes and a barometer. It incorporates an onboard controller 8 times more powerful than the one from the AR.Drone 2.0 (dual-core processor Parrot P7), a quad-core graphic processor, flash memory of 8Gb and a Linux distribution. It is controlled via Wi-Fi (it provides its own net) and a SDK is available for application development.

Although the Bebop comes with some software for basic functionality, it's neither open-source nor easy to modify, and so we treat the drone as a black box, using only the available W-LAN communication channels to access and control it.

Fig. 1. Bebop Drone from Parrot, the commercial drone used as flying unit in our experiments.

Specifically, these are the inputs/outputs we use in our SLAM system:

- Video channel, to receive the video stream of the forwards facing camera, with maximal supported resolution of 640x368 and frame rate of 30fps.
- Navigation channel, to read onboard sensor measurements every 5ms. The data used by our system are:
  1. Drone orientation as roll, pitch and yaw angles $(\overline{\Phi}, \overline{\Theta}, \overline{\Psi})$.
  2. Horizontal velocity in drone's coordinate system $(\overline{vdx}, \overline{vdy})$, calculated onboard by an optical-flow based motion estimation algorithm [19].
  3. Drone height $\overline{h}$, obtained from the ultrasound altimeter measurements.
- Command channel, to send the drone control packages, with the desired velocities of x and y axis; vertical speed and yaw rotational velocity:

$$\boldsymbol{u} = \left(\widehat{vx}, \widehat{vy}, \widehat{vz}, \widehat{\Psi}\right) \quad (1)$$

B. Software Architecture

As it's shown in *Fig. 2*, the onboard controller and processor perform sensor readings and basic control of the MAV. The ground station executes our SLAM system and also the control and planning strategies, the last ones being out of the scope of this paper.

The SLAM system explained in this paper consist of two major components: (a) a monocular VSLAM system that obtains a 6DOF pose estimation (and a 3D map of the environment); (b) an Extended Kalman Filter that fuses the last estimation with the navigation data provided by the onboard sensors of the MAV to obtain a robust 6DOF estimation of the position of the robot in the generated map. Besides, we have implemented a PID controller that allows the MAV to reach goal poses using the estimated position.

IV. SLAM Approach

In the following subsections, we describe the modules of the SLAM system.

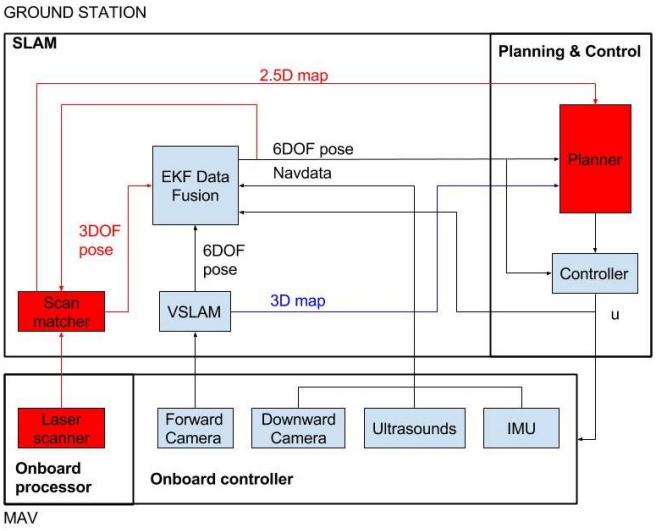


Fig. 2. Software architecture of the ISLAMAV project (red modules are out of the scope of this paper).

A. *Monocular VSLAM*

After a study of the state-of-art monocular VSLAM algorithms, we decided to implement two of these algorithms in our system: LSD-SLAM (Large-Scale Direct Monocular SLAM) and ORB-SLAM (Oriented FAST and Rotated BRIEF SLAM), both available as ROS packages.

LSD-SLAM is a direct (feature-less) monocular SLAM algorithm which, along with highly accurate pose estimation based on direct image alignment, reconstructs the 3D environment in real-time as pose-graph of keyframes with associated semi-dense depth maps. Due to the later implementation of the laser SLAM node and its 2,5D map, we are only using the 6DOF pose estimation of this algorithm as an input to the data fusion filter. We chose to use the laser's map instead the one created by LSD-SLAM because of the better accuracy of the first one and due to the computational requirements needed by the last one.

Fig. 3 shows the 3D map and pose estimation obtained by the LSD-SLAM technique in a room (up); and the 3D map and pose estimation obtained in the same room and across two corridors (down). Although results are good in this case, the system needs a high amount of visual characteristics that are not available in dark zones, where it needs to be fused with other sensors. Furthermore, it is very sensitive to pure rotational movement.

On the other hand, ORB-SLAM is a feature-based monocular SLAM. ORB-SLAM estimates the drone's position in an extremely accurate way. It makes it perfect for be implemented over a system based on a MAV due to its fast and unstable dynamics. Furthermore, thanks to a smart development of the algorithm it is able to do a reliable loop closing.

Fig. 4 shows the pose estimation obtained with ORB-SLAM in the same environment of *Fig. 3*. It can be deduced that data from other sensors is needed to correctly estimate the position of the MAV. Although the tracking is correct in the room and along the corridor, it fails calculating the rotation angle after turning the corner. Furthermore, the changing scale makes to get a wrong estimation of distances (the length of the corridor after the corner is shortened).

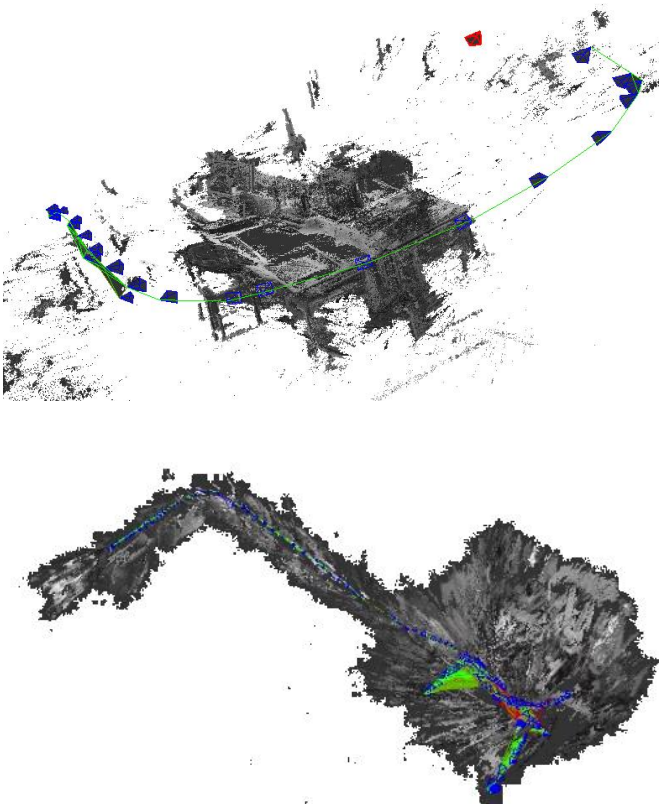


Fig. 3. Results of LSD-SLAM. The first picture represents the translation of MAV's camera around a room. The second one represents the results of the translation around the same room and along two corridors. The green line indicates the track where the camera went over. The blue marks are the camera's poses where the VSLAM algorithm captured a keyframe. The red marks correspond with the actual pose of the camera. The grey-scale shapes are the 3D map of the environment made by LSD-SLAM.

Fig. 5 shows the results obtained when the algorithm estimates de position of the camera around a square of 35m^2 approximately. The loop closure algorithm allows the VSLAM technique to accurately track the real time pose of the camera.

As said before, one of the main problems when working with monocular VSLAM is scale ambiguity. As we need to work with a scale-aware system, we developed a method to calculate the scale based on onboard sensing. Due to it, our system works with real-scale magnitudes. To solve this problem, the system uses the altitude measurements from the altimeter and VSLAM for calculating the scale as follows:

$$scale = \frac{h_{IMU}}{h_{VSLAM}} \quad (2)$$

$$x_{REAL-SCALE} = x_{VSLAM} \cdot scale \quad (3)$$

$$y_{REAL-SCALE} = y_{VSLAM} \cdot scale \quad (4)$$

$$z_{REAL-SCALE} = z_{VSLAM} \cdot scale \quad (5)$$

The scale is calculated at each iteration of the node before the data fusion to avoid problems due to dynamic changes.

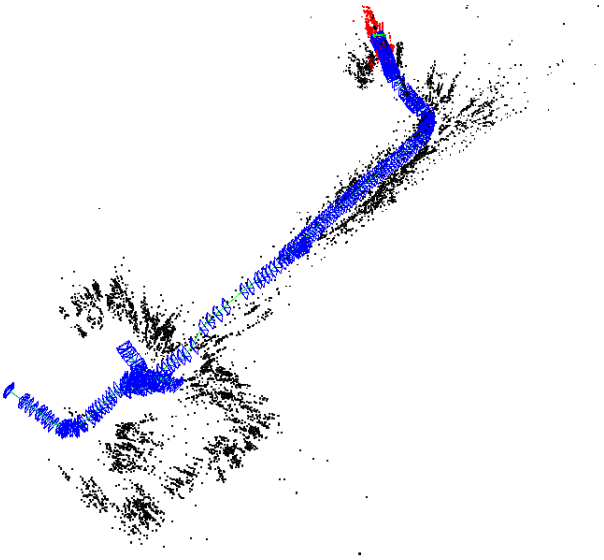


Fig. 4. Results of ORB-SLAM estimating the pose of the camera in the same environment of *Fig. 3.*

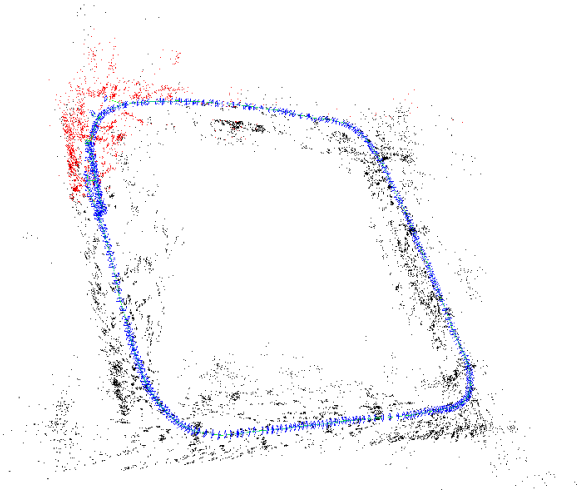


Fig. 5. Results of ORB-SLAM estimating the pose of the camera around a square.

B. *Data Fusion with EKF*

In order to fuse all available data, we employ an Extended Kalman Filter (EKF). This EKF is also used to compensate for the different time delays in the system, as detailed described in [17], arising from wireless LAN communication and computationally complex visual tracking.

The EKF uses the following state vector:

$$\boldsymbol{\chi}_t \coloneqq \left(x_t, y_t, z_t, vx_t, vy_t, vz_t, \Phi_t, \Theta_t, \Psi_t, \dot{\Psi}_t\right)^T \in \Re^{10} \quad (6)$$

where $(x_t, y_t. z_t)$ is the position of the MAV in meters (m); (vx_t, vy_t, vz_t) the velocity in meters/second (m/s); Φ_t, Θ_t, Ψ_t the roll, pitch and yaw angles in radians (rad); and $\left(\dot{\Psi}_t\right)$ the yaw-rotational speed in radians/second (rad/s). All of them are evaluated in world coordinates. In the following, we define the prediction and observation models.

1) *Prediction Model*

The prediction model is based on the full motion model of the quadcopter's flight dynamics and reaction to control commands derived in [17]. A new calibration of the model parameters has been done for the Bebop Drone.

The model establishes that the horizontal acceleration of the MAV is proportional to the horizontal force acting upon the quadcopter, that is, the accelerating force minus the drag force. The drag is proportional to the horizontal velocity of the quadcopter, while the accelerating force is proportional to a projection of the z-axis of the drone onto the horizontal plane, which leads to:

$$vx_t = K_1\left(K_2(cos\Phi\ sin\Theta\ \cos\Psi + \sin\Phi\ \sin\Psi)\right) \quad (7)$$

$$vy_t = K_1\left(K_2(cos\Phi\ sin\Theta\ \sin\Psi - \sin\Phi\ \cos\Psi)\right) \quad (8)$$

Furthermore, the influence of the sent control command $\mathbf{u} = \left(\widehat{v\dot{x}}, \widehat{v\dot{y}}, \widehat{v\dot{z}}, \widehat{\dot{\Psi}}\right)$ is described by the following linear model:

$$\dot{\Phi}_t = -K_3(K_4\widehat{v\dot{y}}_t + \Phi_t) \quad (9)$$

$$\dot{\Theta}_t = K_3(K_4\widehat{v\dot{x}}_t - \Theta_t) \quad (10)$$

$$\dot{vz}_t = K_7(K_8\widehat{v\dot{z}}_t - \mathrm{vz}_t) \quad (11)$$

$$\ddot{\Psi}_t = K_5\left(K_6\widehat{\dot{\Psi}}_t - \dot{\Psi}_t\right) \quad (12)$$

We estimated the proportional coefficients K_1 to K_8 from data collected in a series of test flights. From equations (7) to (12) we obtain the overall state transition function:

$$\begin{pmatrix} x \\ y \\ z \\ vx \\ vy \\ vz \\ \Phi \\ \Theta \\ \Psi \\ \dot{\Psi} \end{pmatrix}_{t+1} \leftarrow \begin{pmatrix} x \\ y \\ z \\ vx \\ vy \\ vz \\ \Phi \\ \Theta \\ \Psi \\ \dot{\Psi} \end{pmatrix}_{t} + \Delta_t \begin{pmatrix} vx_t \\ vy_t \\ vz_t \\ K_1(K_2(cos\Phi\ sin\Theta\ \cos\Psi + \sin\Phi\ \sin\Psi) - vx_t) \\ K_1(K_2(cos\Phi\ sin\Theta\ \sin\Psi - \sin\Phi\ \cos\Psi) - vy_t) \\ K_7(K_8\widehat{v\dot{z}}_t - vz_t) \\ -K_3(K_4\widehat{v\dot{y}}_t + \Phi_t) \\ K_3(K_4\widehat{v\dot{x}}_t - \Theta_t) \\ \dot{\Psi}_t \\ K_5\left(K_6\widehat{\dot{\Psi}}_t - \dot{\Psi}_t\right) \end{pmatrix} \quad (13)$$

2) *Inertial Navigation Observation Model*

This model relates the onboard measurements obtained through the navigation channel of the quadcopter described in section *III.A* –that we called "navdata" in *Fig. 2*– and the state vector. The quadcopter measures its horizontal speed $\left(\overline{vdx}, \overline{vdy}\right)$ in its local coordinate system, which we transform into the world frame (vx, vy). The roll and pitch angles measured by the accelerometer are direct observations of the corresponding state variables. On the other hand, we differentiate the height measurement and the yaw measurement as observations of the respective velocities. The resulting measurement vector z_{NAVDATA} and observation function $h_{NAVDATA}(\boldsymbol{\chi}_t)$ are:

$$z_{NAVDATA} := \left(\overline{vdx}, \overline{vdy}, \bar{h}_t - \bar{h}_{t-1}, \bar{\Phi}, \bar{\Theta}, \bar{\Psi}_t - \bar{\Psi}_{t-1}\right) \quad (14)$$

$$h_{NAVDATA}(\boldsymbol{\chi}_t) := \begin{pmatrix} vx_t cos\Psi_t + vy_t sin\Psi_t \\ -vx_t sin\Psi_t + vy_t cos\Psi_t \\ z_t \\ \Phi_t \\ \Theta_t \\ \Psi_t \end{pmatrix} \quad (15)$$

3) VSLAM Obvservation Model

When the VSLAM algorithm successfully tracks a video frame, its 6DOF pose estimation is transformed from the coordinate system of the front camera to the coordinate system of the quadcopter, leading to a direct observation of the quadcopter's pose given by:

$$z_{VSLAM,t} := f\left(E_{DC} E_{C,t}\right) \in \Re^6 \quad (16)$$

$$h_{VSLAM}(\boldsymbol{\chi}_t) := (x_t, y_t, z_t, \Phi_t, \Theta_t, \Psi_t)^T \in \Re^6 \quad (17)$$

where $E_{C,t} \in SE(3)$ is the estimated scale-aware camera pose, $E_{DC} \in SE(3)$ the constant transformation from the camera to the quadcopter coordinate system and $f: SE(3) \rightarrow \Re^6$ the transformation from an element of SE(3) to the roll-pitch-yaw representation $(x, y, z, \Phi, \Theta, \Psi)$.

C. PID Controller

A PID controller was developed in order to control the movements of the MAV based on the estimated position. A reference $(\hat{x}, \hat{y}, \hat{z}, \hat{\Psi})$ is needed as the desired position of the drone in relation with the surroundings. The EKF will bring the estimation of the pose, as shown in Fig. 6. The difference between the reference and the estimated pose is the error that will be minimized by the PID controller, by sending to the MAV an appropriate control command $\mathbf{u}=\left(\widehat{vx}, \widehat{vy}, \widehat{vz}, \widehat{\Psi}\right)$, that is calculated in the following way:

$$\widehat{vx} = cos\Psi[Kp(\hat{x} - x) + Kd \cdot \dot{x}] + sin\Psi[Kp(\hat{y} - y) + Kd \cdot \dot{y}] \quad (18)$$

$$\widehat{vy} = -sin\Psi[Kp(\hat{x} - x) + Kd \cdot \dot{x}] + cos\Psi[Kp(\hat{y} - y) + Kd \cdot \dot{y}] \quad (19)$$

$$\widehat{vz} = Kp \cdot (\hat{z} - z) + Kd \cdot \dot{z} + Ki \cdot \int(\hat{z} - z) \quad (20)$$

$$\widehat{\Psi} = Kp(\hat{\Psi} - \Psi) \quad (21)$$

It allows the algorithm to drive the MAV along a series of points in the map so it can follow a specific track.

V. Results

For the purpose of testing our system with a reliable ground truth, we used a horizontal motion detector camera, which was installed in the ceiling of the test environment. It allows us to measure the XY movements of the drone using an external sensor. It is not possible to sense the altitude with this method, so we trust in the altimeter integrated in the MAV as the ground truth. This procedure allows us to contrast the position estimated by our algorithm with the true position detected by the external camera.

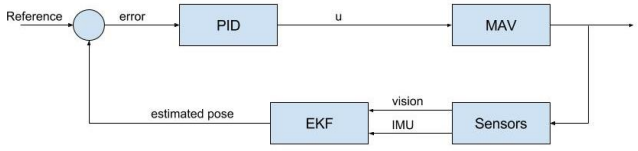


Fig. 6. PID Controller Blocks Diagram

ORB-SLAM was used during the tests which results are represented in *Fig. 7*, *Fig. 8 and Fig. 9*. We used this VSLAM algorithm instead of LSD-SLAM because we didn't need the 3D map that LSD-SLAM could bring us –so the computational requirements were avoided–. For this work the light conditions were optimal and we provided enough features to be extracted by the algorithm to ensure its performance. In other conditions –for example lack of features, where LSD-SLAM execution stands out–we could have chosen other algorithm. Furthermore, we realized that ORB-SLAM represents a more robust VSLAM technique facing pure rotational movement and fast translations.

As said before, the PID controller allows the MAV to execute a path through a series of points. As a test, we made the drone to fly trying to recreate a square of 1mx1m –which is plotted as a green square in *Fig 7*, *Fig. 8* and *Fig. 9*.

As a first test, we run the algorithm with each of the stages of the EKF separately –this is, only with prediction stage, only with IMU correction stage and only with VSLAM correction stage–, shown in *Fig. 7*. As we are not able to test the vertical precision of the algorithm –where the IMU performance stands out– the better tracking of prediction and VLSAM correction stages are obvious. To represent the bad performance of the algorithm when it's using only the IMU measurements, *Fig. 7* plots the results of the method with all its drift. Below, *Fig. 8* includes the same graph but zoomed in order to make easy to see the differences between implementations.

The performance's improvement of the system with the addition of the stages summarized on *IV-B* is evaluated on the *Fig 9*. As shown, the system is most accurate with prediction and both IMU and VSLAM correction stages. That precision is the cause of this project and why we are making the fusion of VSLAM and IMU measurements –and as explained in *VI*, laser as a future new stage–.

VI. Conclusions and Future Work

This paper shows work in progress and initial results of an indoor SLAM system for MAVs that fuses measurements from a monocular camera and onboard sensors to obtain a better estimation of the 6DOF pose of the MAV and a map (3D if LSD-SLAM is being used) of the local environment.

This work provides a scale aware tracking and mapping system, which will be incorporated to the whole architecture of the ISLAMAV project [6]. This will conclude in a system that could calculate in real time the position of the drone without drift and a 2.5D template or map of the environment. This will be extremely useful to estimate the real position of the MAV. Furthermore, this system will be more robust facing problems as lighting changes.

In future work comparisons between the performances of the system using each algorithm should be made and displayed.

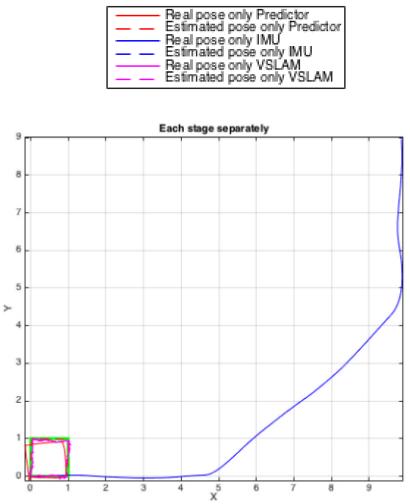


Fig. 7. Zoom out of different stages implemented separately.

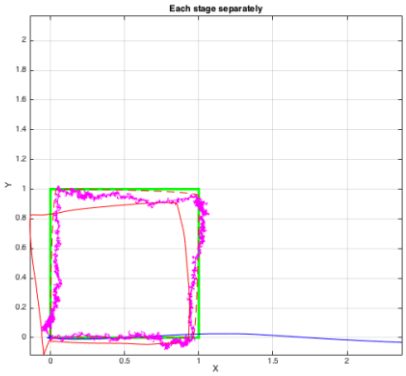


Fig. 8. Zoom in of different stages implemented separately.

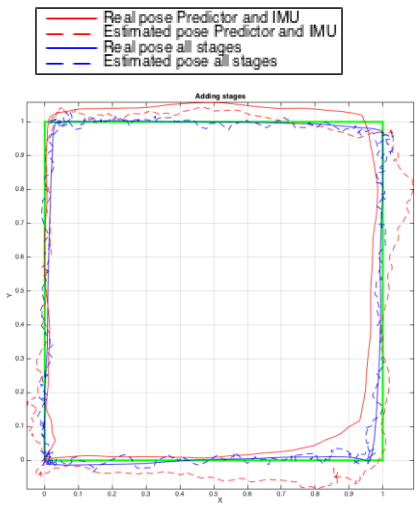


Fig. 9. Performance of added stages.

VII. Acknowledgment

This work is supported in part by the ISLAMAV Project (CCG2015/EXP-044), which is funded by the University of Alcalá and in part by the RoboCity2030 III-CM project (S2013/MIT-2748) funded by I+D Programs in Comunidad de Madrid and cofounded by Structured Funds of the UE.

References

[1] Mellinger, D. Kumar, V.: Minimum snap trajectory generation and control for quadrotors. In: Proc.IEEE Intelligent Conference on Robotics and Automation (ICRA). pp. 2520-2525. (2011).

[2] Lindsey, Q., Mellinger, D. Kumar, V.: Construction of cubic structures with quadrotor teams. In: Proceedings on Robotics: Science and Systems (RSS). (2011).

[3] Kushleyev, A., Mellinger, D., Kumar, V: Towards a swarm of agile micro quadrotors. In: Proceedings of Robotics: Science and Systems (RRS). (2012).

[4] Achtelik, M., Lynen, S., Weiss, S., Kneip, L.,: Visual-inertial SLAM for a small helicopter in large outdoor environments. In: Proceedings IEEE Conference on Intelligent Robots and Systems (IROS).pp. 2651-2652 (2012).

[5] Fraundorfer, F., Heng, L., Honegger, D., Tanskanen, P.,.: Vision-based autonomous mapping and exploration using a quadrotor MAV. In: Proceedings IEEE Inter. Conference on Intelligent Robots and Systems (IROS). (2012).

[6] 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. In Robot 2015: Second Iberian Robotics Conference: Advances in Robotics, Volumen 1 (2015).

[7] Engel, J., Schöps, T., Cremers, D.: LSD-SLAM: Large-scale direct monocular SLAM. In Computer Vision (ECCV 2014), pp.834-849. (2014).

[8] Mur-Artal, R., Montiel, J., Tardos, J.: ORB-SLAM: A Versatile and Accurate Monocular SLAM System. In IEEE Transactions on Robotics, vol. 31, no. 5, pp. 1147-1163 (2015).

[9] Grzonka, S., Grisetti, G., Burgard, W.: Towards a navigation system for autonomous indoor flying. In: Proceedings IEEE Intelligent Conference on Robotics and Automation (ICRA). (2009).

[10] Bylow, E., Sturm, J., Kerl, C., Kahl, F., Cremers, D.: Real-time camera tracking and 3D reconstruction using signed distance functions. In: Proceedings of Robotics: Science and Systems (RSS). (2013).

[11] Tournier, G.P., Valenti, M., How, J.P., Feron, E.: Estimation and control of a quadrotor vehicle using monocular vision and moiré patterns. In: Proceedings of AIAA GNC, Keystone, Colorado. (2006).

[12] Johnson, N.G.: Vision-assisted control of a hovering air vehicle in an indoor setting. Master's thesis, BYU. (2008).

[13] Kemp, C.: Visual Control of a Miniature Quad-Rotor Helicopter. PhD Thesis, Churchill College, University of Cambridge. (2006).

[14] Ahrens, S., Levine, D., Andrews, G., How, J.P.: Vision-based guidance and control of a hovering vehicle in unknown, gps-denied environments. In: IEEE International Conference on Robotics and Automation (ICRA), pp. 2643-2648. (2009)I.S. Jacobs and C.P. Bean, "Fine particles, thin films and exchange anisotropy," in Magnetism, vol. III, G.T. Rado and H. Suhl, Eds. New York: Academic, 1963, pp. 271-350.

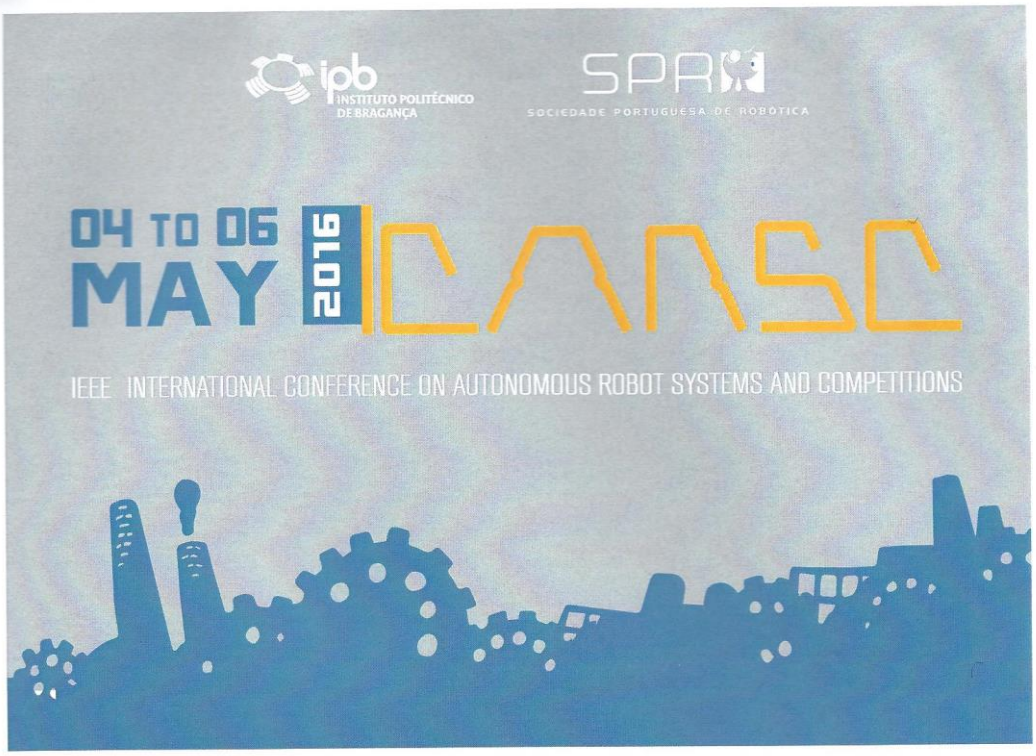
[15] Teulière, C., Marchand, E., Eck, L.: 3-D Model-Based tracking for UAV Indoor Localization. In: IEEE Transactions on cybernetics, vol. 45, nº5. (2015).

[16] Brockers, R., Humenberger, M., Weiss, S., Matthies, L.: Towards autonomous navigation of miniature UAV. In: 2014 IEEE Conference on Computer Vision and Pattern Recognition Workshops. (2014).

[17] Engel, J., Sturm, J., Cremers, D.: Scale-Aware Navigation of a Low-Cost Quadrocopter with a Monocular Camera. In Robotics and Autonomous Systems (RAS), volume 62, 2014 (2014).

[18] http://www.parrot.com/products/bebop-drone/

[19] Bristeau, P-J., Callou, F., Vissiere, D., Peit, N.: The navigation and control technology inside the AR.Drone micro UAV. 18th IFAC World Congress, pp. 1477-1484. (2011).



CERTIFICATE

This is to certify that

Sergio García Gonzalo

has presented a poster communication entitled **"Indoor SLAM for Micro Aerial Vehicles Control using Monocular Camera and Sensor Fusion"** co-authored by **Sergio García, M. Elena López, Rafael Barea, Luis M. Bergasa, Alejandro Gómez, Eduardo J. Molinos**, at the International Conference on Autonomous Robot Systems and Competitions held from May 4 to 6, 2016, in Bragança, Portugal.

José Lima
(Chair of the Organizing Committee)

4. Implementation of all the developed algorithms over a real platform.

The implementation over a real platform and the reached results are deeply explained in the book of the thesis.

5. Attendance to conferences and lectures.

Some conferences and lectures that dealt with related fields of this work were attended. As they exposed innovative ideas in the state of the art it was interesting to listen and discuss about different concepts in order to develop the research in the field of this work.

The writing proof of attendance to the conference of ICARSC is attached in the following. Some lectures from the conference were related to MAVS, but most of them put to use a laser as the main sensor for data fusion. Others have developed a system based on swarm strategies and the rest did not have results or their results were simulated. After it, the writing proof of attendance to the lecture given by the Dra. Marta Salas García (from Zaragoza's University) about "Layout Aware Visual Tracking and Mapping" is attached too. The author of the work talked about the monocular camera-based system that she developed. It is a monocular VSLAM featured-based algorithm intended of work indoors that is able to recognize different rooms and know when is inside each of them. It reduces the computer requirements (the features that are not in use because they were extracted in a room where the camera is not at the moment are not processed). It also recognize windows and doors so it is able to avoid typical problems related with monocular VSLAM algorithms.

I have also attended to the classes of the subject "Sistemas de Percepción" of the "Master Universitario en Industriales" in Alcalá University by Dr. Luis Miguel Bergasa Pascual; where the camera calibration procedures were explained.

It is also expected the attendance to the Workshop "Robotics for Inspection and Maintenance (ROBIM)" that it is going to be held in Madrid the 7th of July. It is organized by the Robotics and Mechatronics Spanish Robotics Network (REDROM) and the Spanish Society for Research and Development on Robotics (SEIDROB). This Workshop includes several sessions about aerial robotics and computer vision for drones.

CERTIFICATE OF ATTENDANCE

This is to certify that

Sergio García Gonzalo

has attended the International Conference on Autonomous Robot Systems and Competitions held from May 4 to 6, 2016, in Bragança, Portugal.

José Lima
(Chair of the Organizing Committee)

Universidad de Alcalá

Roberto Javier López Sastre
DPTO DE TEORIA DE LA SEÑAL Y COMUNICACIONES
Despacho S342 , Escuela Politécnica Superior
Universidad de Alcalá
Campus Universitario. - Crta. A2, Km. 33,600
28805 Alcalá de Henares
Teléfonos: 91 885 67 20
Fax: 91 885 66 99
Email : robertoj.lopez@uah.es

Dr. Roberto Javier López Sastre, con DNI 08993627Y, en calidad de organizador de la conferencia impartida por la Dr. Marta Salas García, de la Universidad de Zaragoza, con título, "Layout Aware Visual Tracking and Mapping", celebrada el día 20 de enero de 2016 en la Sala de Juntas S304 del Departamento de Teoría de la Señal y Comunicaciones, de la Escuela Politécnica Superior de la UAH, y de duración 1 hora,

CERTIFICA que:

Don/Doña Sergio García Gonzalo , con DNI **03143202E,** asistió a la misma.

Para que conste, firmo el presente certificado de asistencia, en Alcalá de Henares, a 21 de enero de 2016.

Fdo.: D. Roberto Javier López Sastre

Universidad de Alcalá
Escuela Politécnica Superior

ESCUELA POLITECNICA
SUPERIOR

Universidad
de Alcalá