UCL
Université catholique de Louvain

epl
ÉCOLE POLYTECHNIQUE DE LOUVAIN

Towards 3D Visual SLAM for an Autonomous Quadcopter Running on ROS

Dissertation presented by

Arnaud JACQUES and **Alexandre LECLÈRE**

for obtaining the Master's degree in

Engineering in Applied Mathematics

and

Electro-mechanical Engineering (*Option: Mechatronics*)

respectively

Supervisor **Pr. Julien HENDRICKX**

Advisor **Eng. François WIELANT**

Readers **Dr. François BAUDART**

Pr. Jean-Charles DELVENNE

Academic year 2015-2016

Abstract

In a context where multi-copters gain in attractiveness each day to solve novel challenges or replace older technologies, the need for autonomous behaviour is being pointed out as a key feature for unlocking lots of applications. This master thesis is conducted at UCL for trying to understand the current challenges to unlock multi-copter autonomy. We point out that the lack of robustness of all current state-of-the-art implementations which are based only upon on-board sensors, leads, as a common flaw, to a lack of accuracy of the pose estimation. Indeed, for GPS-denied environments such as indoors, there exist poor absolute references to build an accurate and robust belief of the position and orientation of the drone. Unfortunately, indoor situations need precisely the best pose estimation since they are confined places not allowing too large errors while executing movement. We propose an implementation of the direct keyframe-based visual SLAM approach based on features detected in the images of the video-stream of an embedded camera. We show that the results obtained on a real low cost quadcopter, the Parrot AR.Drone, improve substantially the pose estimation, notably by canceling the drift on sensor readings. However, this technique presents strong limitations when confronted to untextured environments, due to the lack of reference keypoints detected in the images. To palliate this, we explore the state of the literature to propose some promising advances to replace or enhance this approach, such as replacing conventional cameras by RGB-D (Red Green Blue - Depth) cameras or DVS (Dynamic Vision Sensors), or replacing computationally costly software parts by a hardware implementation, or better fusing the available sensors informations.

Contents

List of Abbreviations

AHRS	Attitude and Heading Reference System
AI	Artificial Intelligence
AR/VR	Augmented Reality/ Virtual Reality
BRIEF	Binary Robust Independent Elementary Features
BRISK	Binary Robust Invariant Scalable Keypoints
BW	Black and White
CAN	Controller Area Network (protocol)
CoSLAM	Collaborative SLAM
CV	Computer Vision
DOF	Degrees Of Freedom
DoG	Difference of Gaussians
DVS	Dynamic Vision Sensor
ETHZ	Swiss Federal Institute of Technology in Zurich
FPGA	Field Programmable Gate Array
FREAK	Fast Retina Keypoint
GPS	Global Positioning System
I2C	Inter-Integrated Circuit (protocol)
IMU	Inertial Measurement Unit
INS	Inertial Navigation Systems
LINS	Laser-aided INS
MAVLINK	Micro Air Vehicle Link (protocol)
MCPTAM	Multi-Camera PTAM
ORB	Oriented FAST and Rotated BRIEF (descriptor)
OS	Operating System
PCL	Point Cloud Library
PTAM	Parallel Tracking and Mapping
PTAMM	Parallel Tracking and Multiple Mapping
PWM	Pulse Width Modulation
RTK	Real Time Kinematic
ROS	Robot Operating System
RTOS	Real-Time OS
RGB	Red Green Blue
RGBD	RGB-Depth (sensor)
SDK	Software Development Kit
SIFT	Scale-Invariant Feature Transform
SLAM	Simultaneous Localisation an Mapping
SPI	Serial Peripheral Interface (protocol)

SURF	Speeded Up Robust Features
SVO	Semi-direct monocular Visual Odometry
TUM	Technische Universität München
UART	Universal Asynchronous Receiver Transmitter (protocol)
VINS	Vision-aided INS
Wi-Fi	Wireless Fidelity (protocol)

List of Figures

List of Tables

Chapter 1

Introduction

"Perception is key to intelligent behaviour. While the field of Artificial Intelligence has made impressive strides in replicating some aspects of cognition, such as planning and plan execution, machine perception remains fragile and task-specific." [1]

1.1 On the drones in general

The word drone can be understood in many different ways. Therefore, we first of all must make clear what we will refer to when using this term. In this work, we will refer to UAV's (Unmanned Aerial Vehicles) of the VTOL class (vertical take-off and landing), and more specifically, to the multi-rotor type (rotorcrafts with more than two rotors). These can include tricopters, quadcopters, hexacopters, octocopters, and so on. In the implementation part of this text however, only quadrotors will be used to test some of the below techniques.

Recently, this class of flying robots has gained a great interest due to its compactness and low cost. For these reasons, it has been considered for more and more applications, to replace older technologies, or to solve newer challenges. Some examples include:

- Rescue operations [2] [3] [4] [5]
- Construction [6] [7]
- Industrial supervision [8] [9]
- Package delivery [10]
- Replacing fireworks [11]
- Aerial photography [12] [13]
- Accessing dangerous places [14]
- Ambulance drones [15]

All these applications could gain in efficiency and robustness if drones were to be more autonomous. For example, we can imagine a professional photographer wanting to take pictures of a large crowd. If he loses control over the drone due to some malfunction of its remote control, how will the drone react? In the ideal case, the drone should land autonomously, avoiding environing infrastructures, and safely avoiding the crowd.

As another example, accessing dangerous places like nuclear disaster zones. In 2014 for example, Japan used drones for radiation measurements at Fukushima [14]. This could prevent

a human supervisor to be exposed to radiations during inspection tasks. Such applications imply NLOS (non-line-of-sight) situations. In this case, no operator would be able to remote control the drone since the drone is not at sight. Therefore, a drone capable of exploring an environment would be of great use. This can imply the ability to map an a priori unknown environment, while simultaneously locating in it (see Subsection 2.2.3) and using the generated map for path planning purposes (see Section 2.4).

These examples emphasize the need for technologies enabling autonomous flight of multi-rotors. Chapter 2 gives an overview of the kind of technologies that currently are the focus of attention.

1.2 Context of this work

In this context of a world swarming with drones, UCL (Université catholique de Louvain) was recently fortified with multiple research teams working with them. This particular work takes place in the INMA departement (Mathematical Engineering). Our lab is composed of two groups of 2 students in the last year of engineering studies, one lab engineer, and one professor (supervisor). Each group is composed of one student in mechatronics and one student in mathematical engineering. In this lab, work is currently done to get drones to fly in an autonomous way. The long term objectives are to:

1. Enable multiple agent systems flight in order to deploy collaborative algorithms, since this is an application of our promoter's field of research.

2. Gain expertise about drones for UCL (this project could be used in the context of the *construction with drones* project [7], also led at UCL). Collaboration is actually being put in place, notably through the creation of a *DroneZone*, a flying arena where multiple research teams meet to test and exchange about their advances.

1.3 Problem statement

Towards autonomy The INMA department has acquired six drones Parrot AR.Drone, which are low-cost, commercially available quadrotors. It was asked for the two groups of two students to collaborate on the implementation of code to make them fly autonomously. Autonomy means that the quadrotors have to be able to take their decisions independently in a-priori unknown environments. So, to observe and to interact with its environment, the quadcopter should require only onboard sensors, no external devices of any kind, neither passive nor active. To have a truly autonomous robot, we also want to give minimal amount of information about the environment within which the drone has to fly. Indeed, the system must have no prior knowledge on what is present around the drone. Therefore, we will attempt to choose algorithms with minimal requirements on the environment.

ROS One main difference in the problem statement of the previous master thesis conducted on this topic (see Section 1.6), is that the code must run on the ROS platform (Robot Operating System).

Collaborative work In addition to this, the presence of a second group in the lab implied some collaborative work. It was decided to work together on the code architecture, and then to split in an efficient way the code modules among our groups (see Section 4.7 for the repartition of code modules). So, for example, our team has been attributed the implementation of all the nodes which implied computer vision problems. This was done to avoid the need for the four of us to learn quickly about this vast domain. It was thus also required in the problem statement that the code of our two groups had to run all together by the end of the year.

State estimation From this distribution of tasks also resulted our specific role of providing the state estimation of the drone. Indeed, before executing complex tasks and impressive operations, one primary concern is to give the drone the knowledge of its state and the state of the environment. By state of the drone, it is meant the state of its internals, but also the state of the quadcopter in its environment. The estimation of the position and the orientation of the robot relative to its environment and the description of the environment are key enabling features for any sophisticated task. The concatenation of its position and orientation is commonly referred as the *pose* in the field of robotics. These two concerns will be the main topics of this master thesis. These features are all the more challenging as a flying vehicle operates in a full three-dimensional environment, and the movement of a quadcopter is continuously subject to disturbances. SLAM (Simultaneous Localization And Mapping) is defined as the combined (and recursive) problem of state estimation with respect to a map of the environment that is built incrementally in real-time while the environment is being explored.

Imposed sensors and low cost hardware As we want the drone to accomplish tasks autonomously, no global positioning system (GPS), nor external transmitter or external cameras will be used to localize the drone. Cutting-edge quadcopters embed high-accuracy sensors. In our implementation, a low cost quadcopter is used. This means sensors are more prone to noise that has to be treated.

Towards swarms of quadcopters Our design choices must also be guided by the long term objective of multi-agent systems for quadcopters.

1.4 Approach

Our approach can be summarized by the following steps:

1. Thoroughly inspect the current state of the art in the field of research on autonomous quadcopters.
2. Test some state-of-the-art techniques available as open-source software.
3. Design a software architecture to tackle the autonomy problem (in collaboration with the second group).
4. Use real quadrotors in an indoor environment to develop an original approach (no assumption will be made that prohibits outdoor generalization).
5. Report the results.
6. Pass down our work to the next year's students by providing good documentation, code readability and modularity.

1.5 Resources and constraints

Ressources

- **Drones:** 6 Parrot AR.Drone, with embedded proprietary software and closed-source hardware (further technical specifications are detailed in Subsection 3.1.1).
- **Computation capacity:** 1 computer running Ubuntu 14.04, with an Intel® Core™i5 CPU 650 @ 3.2 Ghz x 4, and an 8 GB RAM. This computer is used for development purposes only, and the assumption is made that the code running on it in real-time will be able to run on an embedded computer in the future.
- **Connection from the PC to the drone:** Wi-Fi is used, through a TPlink® TL-WR841N router, which enables multi-drone IP addressing.
- **Human ressources:**
  - Eng. F. Wielant: the advisor.
  - Pr. J. Hendrickx: the supervisor.
  - A second team of two master-thesis students which we collaborate with: J. Gérardy and F. Schiltz, working on the topic "Tracking of a mobile target by a group of drone" ("Suivi d'une cible mobile par des drones autonomes").
- **Work space:** One laboratory dedicated to our two teams at the INMA Departement, and the *DroneZone* of UCL.
- **Time:** The equivalent of 5 months of work per student.

Constraints

- Only low cost drones are available for development, with closed-source hardware and software.
- No GPS is available for navigation.
- Our lab at the INMA department is not very large. This implies limitations due to wall-interference effects. Second, the current lightning infrastructure is composed of neon lights, which degrades vision-based perception.
- No external measurement system is available (for example a motion capture system to provide ground-truth measurements (see Subsection 2.2.5).
- The collaboration with another team implies increased organizational complexity.

1.6 Previous years

Our master thesis forms a part of a continuing work realized by other students, namely

- Nicolas BEGHIN Thibault MARTIN, *Localisation et Cartographie Simultanées (SLAM) par un Quadricoptère AR.Drone Autonome*, 2013.

  > One of the objectives of this first master thesis was to familiarize with the Parrot AR.Drone 2.0 and the software development kit (SDK) proposed by Parrot. A 2D visual SLAM algorithm has also been implemented, based on the detection of predetermined objects of interest disposed in the environment. These objects were colored circular pucks disposed on the ground. By 2D-SLAM, it was meant that only the position in the x-y plane of the ground was taken into account, and that the map was flat (a 2D representation of the world).

- Jean HERMAN, *Analyse d'image pour la localisation autonome et la cartographie par un drone quadricoptère*, 2014.

  > The need for predetermined features was a limitation of the implementation of the previous year. More advanced computer vision capacities were envisaged to remedy this problem. Several feature detection and description methods were tested, mainly in the MATLAB® environment.

- Brieuc DE RADIGUES et Florent VAN HIJFTE, *Simultaneous localisation and mapping (SLAM) and investigation of multi-drone capacities*, 2015.

  > With the expertise acquired the two previous years, the visual SLAM algorithm was adapted. Some performance issues were also enhanced, like timing constraints. They also explored the multi-drone capacities with the Parrot AR.Drone, using facilities implemented by Nicolas Rowier, then computer specialist for the INMA Departement (see [16]).

These previous master theses set the scene for our current thesis. However, the basis code that had been transmitted until now was fit for the Parrot SDK. Unfortunately, this initially chosen software framework did not ease the development and the extension of existing code basis. This framework was also very specific to the Parrot hardware.

For this reason, from this year on, another software environment will be used: ROS. This collaborative robotic middleware has gained a lot of popularity since 2010 (the same year as the Parrot SDK creation). This means that all the functionalities of the previous years thesis must be adapted or even re-implemented. This will be further discussed in Chapter 4.

Once this first objective is achieved with the help of the second group, a series of other functionalities will be implemented, with as main preoccupation to enhance the pose estimation, by notably improving the 2D-SLAM approach and to partially extend it to 3D-SLAM (i.e. a 3D pose estimation and a 3D map). Indeed, as already stated, the key feature for unlocking the autonomy for a robot is the estimation of its pose in its environment. Without this knowledge, applications for quadcopters are limited. The simple idea of a quadcopter going back to home after a journey is not feasible without position and orientation estimation.

The precise and concrete objectives to be achieved by the end of the year are fixed in the next section.

1.7 Objectives

This year we are going to fulfill the following achievements.

Working in tight collaboration with the other group:

- Pass to the ROS environment.
- Propose an appropriate code architecture.
- Re-implement the past years benchmark mission with our own techniques, trying to re-use a maximum of ROS-maintained modules of code.

Working independently:

- Test the ROS modules for LSD SLAM and PTAM, two state-of-the-art implementations of SLAM.
- Implement our own keyframe-based SLAM algorithm to palliate limitations of current implementations and to gain more insight into the problem.
- Provide an upgrade of the current sensor-based pose estimation by a visual-corrected pose (implement a first fusion approach). The fused pose must be usable by the controller node implemented by the other group).
- Implement a visual feature-based target recognition, and provide a target pose estimation (visual tracking of a pre-defined target) to the strategy node (where it must be usable by the second group for higher level purposes).
- Provide a 3D framework for 3D-SLAM (no 2D hypothesis will be taken in the whole code architecture and functionalities, apart from the projection from scene to camera, which momentarily replaces triangulation between the previously cited keyframes, to recover depth information of the mapped features).

1.8 Outline

Throughout this report, we will summarize the work conducted by our team of two during the current academic year. The different parts include:

- The results of our research about the current state of the art (Chapter 2).
- Precisions about the software and hardware used to implement and develop our original solution (Chapter 3).
- The design choices and technical details about our implementation (Chapter 4).
- Results and performances achieved with our system, notably the accuracy of the pose estimation and code execution timings (Chapter 5 and Appendices A, B and C for complementary measurements).
- Conclusions about the work achieved and further perspectives (Chapter 6).
- Our contribution to the ergonomics of the development environment (Appendix D).

Chapter 2

Related work

The examples of applications of multi-rotors given in Section 1.1 emphasized the need for technologies enabling autonomous flight of multi-rotors. In this state of the art section, we are going to review some actual challenges in this field of research (see Figure 2.1).

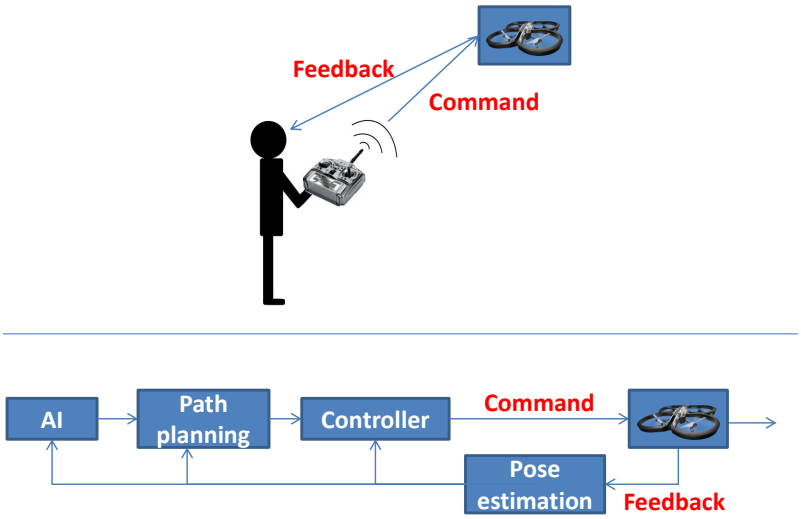


Figure 2.1: Replacing a human operator with on-board intelligence

AI (Artificial intelligence) To be able to operate in an autonomous fashion, a drone must be able to locally make decisions based on sensor informations. In this text, AI refer to the highest level of implemented algorithms, also referred to as strategy. These algorithms perform discrete choices amongst a set of possible moves. Very complex behaviours can be implemented to mimic, for example, human intelligence. A technique to implement such behaviours is for example the "behaviour-based Robotics" paradigm [17]. All along this report, the ability to fly in an autonomous swarm will be emphasized, since it is a natural extension of autonomy. This includes collaborating with other robots of the swarm or on the ground, exchanging sensor information to improve world perception, the ability to perform formation flights, and many others. In this context, an application of the behaviour-based Robotics paradigm has already been developed [18]. Other techniques exist, some of them voluntarily does not try to mimic human intelligence [19]. Artificial intelligence will not be further investigated in this text.

Path planning Path planning is the ability for an autonomous robot to generate a trajectory to navigate in its environment. This includes avoiding obstacles and to generate a list of way-points breaking down the movement. This list of way-points is then directly fed to the controller, where motor commands are generated. Some examples of actual challenges include the computation of optimal trajectories, safe and robust avoidance of obstacles, and so on. Since the second team of our lab, the *team tracking*, has worked on this particular part of the final solution, path planning will not be presented in more details in this report.

Control At an even lower level than AI and path planning, a position controller is needed for enabling the drone to autopilot itself from one point to another in world coordinates. The state of the art for general performance of position control will be addressed in Section 2.1.

Pose estimation In robotics and in computer vision, the pose is defined as the combination of position and orientation of an object relative to a reference coordinate system [20]. To be able to behave autonomously, a drone must know where it is. Thus it must know its pose relative to a coordinate system attached to the world. Since no sensor is able to exactly inform on the 6 DOF (degrees of freedom) 3D pose of the drone (x, y, z, roll, pitch, yaw), an estimation has to be built upon interpretation of the embedded sensor readings. Estimation of its own pose relative to the world is also known as egomotion, and the accuracy of the estimate is critical for controlled flight. This will be the main topic addressed in this work. The state of the art for egomotion of drones will be presented in Section 2.2.

Hardware aspects Power autonomy is a good example of current challenge for drones. UK firm *Intelligent Energy*[1] has developed a hydrogen fuel cell-powered range extender that can keep drones aloft for up to two hours at a time [21]. Battery autonomy will not be further developed in this text. Although, some other hardware aspects will be addressed in Section 2.5.

[1] `http://www.intelligent-energy.com/`

2.1 Agility of micro quadrotors

Lastly, very impressive videos have swarmed on the Internet showing how agilely drones could fly. Examples are:

- Multiple drones collaborating to play music [22]
- A Swarm of drones performing formation flight, reproducing complex 3D figures very accurately [23], and the associated paper [24]
- A TED talk showing the state of the art in fully exploiting the quadrotors dynamics [25]
- Construction with Quadrotor Teams [26] and the associated paper [27], as well as [28].

All of those performances were possible because of a good pose estimation of each drone. In the context of these works, this was achieved by using a ground based positioning system like a motion capture system.
In such a system, multiple ground-based cameras are used to track specific tags glued on each drone. Then, computer vision algorithms are used to estimate the pose of each drone in world coordinates. Finally, the pose computed is sent back to the drone controllers at high rate, allowing very accurate pose stabilization and motion control.

Unfortunately, such systems are not suited for truly autonomous flight in undetermined environment since a beforehand preparation of the environment is needed. However, the use of offboard sensors allows research to focus on control issues without dealing with the challenges of onboard perception [29].

In a recent video [30] however, R. d'Andrea's team at ETH Zurich could perform a flight of a swarm composed of 33 drones. R. d'Andrea claims that "there are no external cameras, each flying machine uses on-board sensors to determine its location in space". However, the exact technology used is not revealed, only the fact that it is developed at *Verity Studios*[2], a spin-off company of ETH Zurich's Institute for Dynamic Systems and Control[3].

This section highlighted the need for good pose estimation without external devices or any beforehand environment preparation (i.e. all embedded on the drone). We are now going to review different techniques and recent breakthrough to be able to achieve this.

[2] http://veritystudios.com/
[3] http://flyingmachinearena.org/spin-offs/

2.2 Pose estimation

How could a drone fly autonomously if it does not even know where it is?

2.2.1 Wireless positioning systems

In outdoor environment, a very common solution is to use the information from the global positioning system (GPS). Unfortunately, this technology does not scale too well to indoor environments, where satellite coverage is not always available.

Besides the GPS, the following survey [31] provides an overview of the existing wireless indoor positioning solutions, attempts to classify them, and finally gives performance comparisons. This article suggests that the only viable wireless technologies for indoor pose estimation for automation and control purposes are ultrasound and UWB (Ultra-Wideband) radio frequency systems. The application of UWB on quadcopters has been studied in [32]. For this publication, they used a Decawave DWM1000 UWB modules, able to locate indoor tagged objects to a precision of 10 cm at a distance of up to 300 m in Line-of-Sight mode and 40 m in Non-Line-of-Sight mode [33].

However, in this publication, a ground based system is still needed to triangulate the UWB signals received from fixed-position emitting beacons. This goes against our initial requirements. In addition to this, despite the outstanding features of UWB for positioning, such as good penetrability through objects and high accuracy, the signal remains affected by multi-path problems requiring complex range estimation algorithms to maintain good performances [34].

We could also imagine a system where only the distances between the drones in a swarm are used, by measuring the time of arrival (TOA) of the UWB signals emitted and received by and from each drone. This way, there is no more need for a ground station, and a collaboration strategy can be used to compensate for deficiencies in the data owned by a singular drone [35]. Indeed, if we know the position from one drone to another drone with very good accuracy, this relative position information can be used to allow the drones to share and fuse their individual pose estimation based on their respective on-board sensors. Following this principle, the larger the swarm is, the more sensor readings are available for fusion, and thus the more accurate the pose estimation of all the drones in the swarm will be. In [36], this approach has been simulated on a swarm of wheeled mobile robots to emphasize the gain of accuracy on pose estimation in non-line-of-sight (NLOS) positioning scenarios. However, they still considered the case of an UWB ground positioning system.

2.2.2 Visual odometry

With visual odometry, the pose is estimated from an interpretation of embedded camera images. Two distinct approaches exist for obtaining measurement data for motion estimation [37]:

- Optical flow methods represent motion in the image plane as sampled, continuous velocity fields
- Feature based methods rely on the recognition of image points of interest (keypoints) between images (see Section 2.3)

However, visual odometry is not sufficient for safe and robust autonomous flight. Indeed, if somehow the drone loses track of where it is (if the camera is briefly obstructed for example), it will not be able to recover from the error on the pose estimation. This is called "dead reckoning": the current pose is computed based on a previously determined pose. So, all errors accumulate over time. This is why visual odometry is rarely used alone, and is rather used in the context of:

- SLAM (see Subsection 2.2.3)
- Visual-IMU fusion (see Subsection 2.2.4)

Other drawbacks of visual odometry are its lack of robustness to illumination conditions, scene texture (which determines the hardness to detect robust keypoints) and a high computational cost, introducing prohibitive delays for control applications.

2.2.3 SLAM

SLAM stands for Simultaneous Localization And Mapping. This problem is a well-known problem in robotics and is an active topic of research. It occurs when no map of the environment is known a priori and the vehicle has an uncertain motion.

It can be described as follows: a robot equipped with various sensors starts at known coordinates in an unknown environment. The motion of the robot is uncertain. The goal is to provide a continuous estimation of the position and orientation of the robot. This estimation is based on a map which is incrementally built by the robot from scratch [38].

Different kinds of sensors can be used to perform SLAM, like range sensors (light-based, ultrasound-based) or cameras. Each sensor can provide a given amount of information and even if the mathematical formulation holds, the implementation of a SLAM algorithm has to be adapted to each kind of sensor.

Several kinds of maps can be considered. For simplicity, we will consider the most popular approach: the map is an Euclidean space containing a collection of landmarks that can be easily described with the embedded sensors.

SLAM is often described as a probabilistic Markov chain [38][39]. Let us briefly present this description.

- The robot's location at time t is noted x_t.
- The path is the sequence of robot's poses: $X_T = \{x_0, x_1 \dots x_T\}$.
- The control measurements are noted $U_T = \{u_0, u_1 \dots u_T\}$.
- The i-th landmark position is noted θ^i and m is the model of the world containing all θ^i.
- For each θ^i we dispose over several measurements made at some time t: $Z_T^i = \{z_0^i, z_1^i \dots z_T^i\}$.

All these can be represented as the following graphical model in Figure 2.2. We can formally define a motion model $p(x_t|x_{t-1}, u_t)$ and an observation model $p(z_t^i|\theta^i, x_t)$. We have to distinguish the *full SLAM* problem, for which the aim is to estimate the posterior of the full path $p(X_T, m|Z_T, U_T)$, from the *online SLAM* problem, which estimates $p(x_t, m|Z_T, U_T)$. The *full SLAM* problem is generally not solved in real-time but is however useful to refine the map built online.

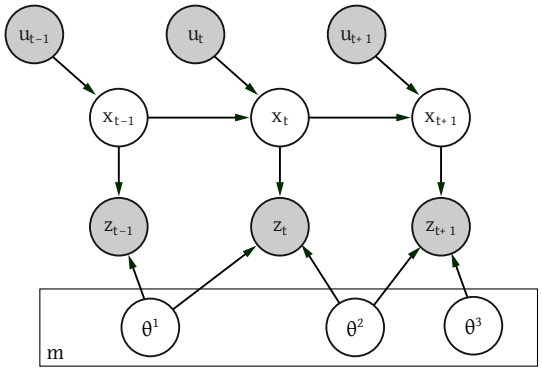


Figure 2.2: SLAM graphical model.
x_t are the successive robot's location,
u_t the successive commands,
θ^i the landmark's location,
z_t the observations of landmarks from location x_t,
each arc represent a constraint.

inspired by [38][39]

Several variants of the SLAM problem are reported in the literature [38] and criteria are proposed in [40] to compare different SLAM algorithms. We do not pretend here to compare all algorithms using all these criteria, but this list presents a good summary of challenges with real time SLAM algorithms.

- **Single or Multi-hypothesis trackers**: one or several vehicles and world states are considered,
- **Localization error**: measure of the accuracy,
- **Computation time**: time to achieve one step,
- **Computational cost** and **Scalability**: complexity often depends on the size of the map (the number of landmarks) and the number of landmarks observed at each step,
- **Distance of loop-closing**: maximum length of the path traveled before a correct match with a previously visited area,
- **Robustness against noise and uncertainties**
- **Non-linearity and non-Gaussian noise**
- **Passive or Active**: the algorithm is only observing or participate to the environment exploration,
- **Robustness to dynamic environment**: big challenge when other agents are in the environment
- **Single or Multi-robot**: the movement of the other agents is known and they have to collaborate to build and use a unique map.

In [38], the authors propose to classify SLAM algorithms in three main paradigms, EKF-SLAM, Graph-based optimization, and particle filter based.

EKF-SLAM is the first formulation to tackle the SLAM. EKF stands for Extended Kalman Filter, which is an adaptation of the Kalman filter for nonlinear systems. EKF uses a motion model as well as an observation model of the sensors. The state of the system incorporates both the state of the robot and the positioning of the landmarks. In general, this algorithm suffers from scaling limitations in large environments.

Graph based SLAM relies on the building of a graphical model (see Figure 2.2), where nodes correspond to locations and landmarks. Successive visited locations are linked to each other and a landmark is linked to locations from which it was observed. The problem is solved using

nonlinear sparse optimization. This formulation cannot be used online because of its complexity. But some adaptations can be made, such as removing old locations.

The last paradigm is the particle filter based SLAM. Each particle can be considered as a guess of the robot state, according to probabilities derived from the motion model and the information from the sensors. The drawback is the number of particles which grows considerably with the state vector.

The probabilistic approach of the 2D SLAM problem has proven to be successful for several robot configurations.

D. Ribas et al. used an EKF-SLAM method with sonar imaging to detect planar structures and to localize an autonomous underwater vehicle in a marina environment (see Figure 2.3).

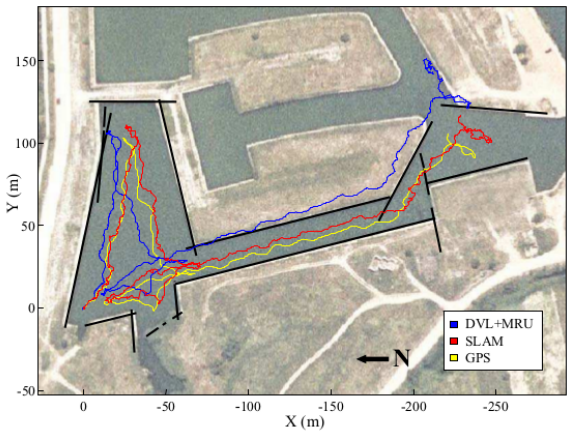


Figure 2.3: Ictineu underwater vehicle and marina located in the Costa Brava (Spain) with the resulting path
D. Ribas et al.[41][42]

J. Levinson et al. used the GraphSLAM algorithm to build a map and a particle filter to localize a moving car in an urban environment using GPS, IMU, wheel odometry and LIDAR equipment. Their approach described in [43] leads to real-time localization with 10cm accuracy which cannot be achieved with only a GPS (see Figure 2.4).

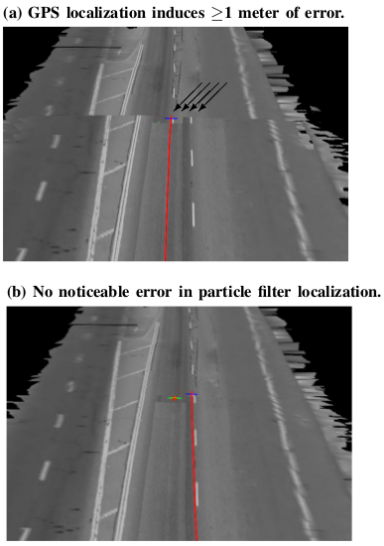


Figure 2.4: Acquisition vehicle used by J. Levinson et al. [43]
and performance illustration

T. N. Yap et al. [44] used a particle filter to localize a robot equipped with low-cost noisy sonars in large indoors environments. They managed to map a 350m^2 environment within 30 minutes, traveling 300m with a wheeled robot (see Figure 2.5).

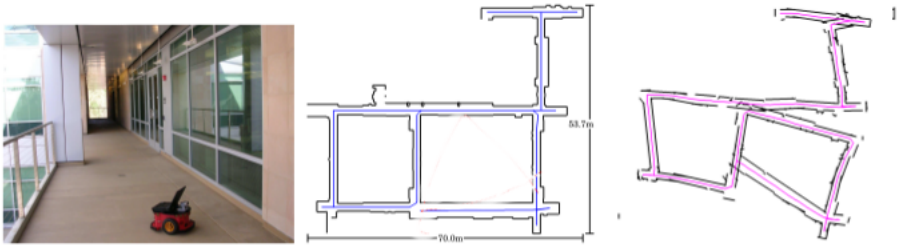


Figure 2.5: Work of T. N. Yap et al. [44]: ground true map (center), map build with SLAM (right)

Monocular Visual SLAM

Visual SLAM are SLAM algorithms which use a camera as sensor. The camera is embedded on the vehicle and captures observations of the environment. A lot of solutions to the SLAM problem use large and heavy sensors with high range and accuracy. The advantages of cameras are their light weight and their low power consumption, which makes them attractive for flying vehicles.

For concision, only monocular rgb cameras will be discussed in this section. Other methods for stereo-cameras, rgb-depth cameras, clusters of cameras, etc. exist as well, but will not be addressed in more details here. We selected a few representative examples. Visual SLAM methods use a lot of computer vision theory for both image processing and multi-view reconstruction. We try here to present algorithms for an uninformed reader. One important remark is that with one monocular camera only, the map built by the SLAM algorithm is said to be consistent up to a scaling factor. This means that without any other sensor, the map suffers of one ambiguity: the distance unit is unknown. This is due to the fact that monocular camera do not provide any depth measure.

As for visual odometry, two approaches exist:

- **Keypoint-based methods :** Algorithms which need to extract feature observations from the image. Features are discrete informations contained in pictures, like object corners, and are easily recognizable and memorable from a computer vision point of view. These keypoints are both used as landmark stored into the map and as features to track the camera motion.
- **Dense mapping and tracking :** Instead of working with extracted features, they directly work on the image: tracking is done using whole-image alignment and the world is made of dense surfaces.

Let us first consider remarkable keypoint-based implementations.

MonoSLAM presented by A. Davison et al in [45] is the first real time successful SLAM for an uncontrolled camera. They used a 3D probabilistic feature-based map updated by an EKF. Their approach is based on a specific motion model for smooth camera movement. This model is used in the EKF prediction step. They describe this model as a "constant velocity, constant angular velocity model".

In [46], G. Klein and D. Murray presented their work on PTAM (Parallel Tracking and Mapping). For PTAM, the problem is split in two subtasks, namely camera motion tracking and landmarks mapping. This approach is different from filtering methods, which use one step to update the state vector. The map is said to be keyframe-based. Each keyframe is like a snapshot containing features observed from one location. Compared to filtering approaches which store information in probability distributions, keyframe-based methods keep back a subset of past observations. The tracking task is divided as follows:

1. A first position estimate is based on a camera motion model and is used to select a small number of keypoints contained in the map.

2. The keypoints of the first step are used to provide a second estimate and a larger number is finally used to give accurate result.

PTAM uses bundle adjustment for building the map and refining the location of keyframes according to observations. According to [47], **bundle adjustment** is an optimization problem that minimizes the re-projection error given visual observations and a camera viewing model. The bundle adjustment refines the reconstruction of both the successive camera and keyframes locations. This process is done in a separate thread because this process cannot be done online. The map structure using keyframes allows to perform bundle adjustment on a subset of the map.

Several other algorithms are derived from PTAM, which is a successful implementation of a realtime visual SLAM. For example, MCPTAM [48] (Multi-Camera PTAM) is an adaptation for a cluster of cameras and PTAMM [49] (Parallel Tracking and Multiple Mapping) provides a search across multiple independent submaps.

Only keypoint-based methods were considered so far. These features are discrete information and their detection has the disadvantage to be environment dependent. Other SLAM algorithms are based on dense mapping and tracking.

DTAM by R. Newcombe et al. is the first dense SLAM algorithm. They demonstrated in [50] that they can achieve better performance (using GPU[4] hardware) than feature-based methods when dealing with rapid motion.

For better performances, J. Engel et al. developed a semi-dense method in [51]. Their software called Large-Scale Direct monocular SLAM (LSD-SLAM [52]) uses a pose graph with keyframes to build consistent a large-scale map of the environment. The results they achieved are impressive (see video at [53]).

C. Forster et al. published the same year another implementation of semi-direct monocular SLAM called SVO [54]. It is also keyframe-based. One difference with LSD-SLAM resides in the feature extraction and image alignment algorithms.

Dense and semi-dense maps are richer representation and could be useful to perform obstacle avoidance because surfaces are better described (see Section 2.4).

[4]GPU (Graphics Processing Unit): electronic hardware to speed up heavy computation on images, often compared to general-purpose CPU (Central Processing Unit)

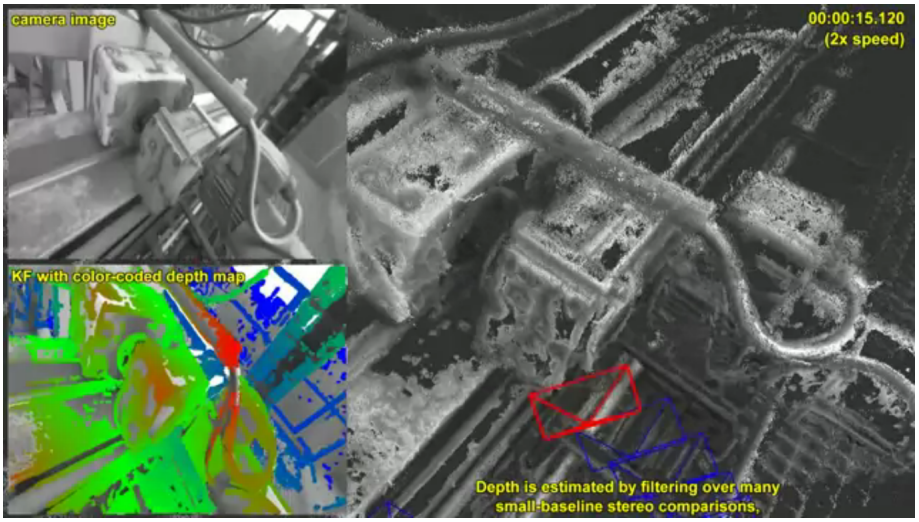


Figure 2.6: LSD-SLAM of J. Engel et al. in [51] [53]

2.2.4 IMU based navigation

Pure IMU navigation

One of the most intuitive ways to estimate the pose of an UAV with embedded sensors in GPS denied environment is to use an IMU (Inertial Measurement Unit). An IMU has generally 6 DOF (degrees of freedom) and is composed of one accelerometer which provides 3D linear accelerations, and one gyroscope which provides 3D angular velocity measurements. An IMU can also be said to have 9 DOF, in which case it also includes a magnetometer, which provides a magnetic orientation of the chip with respect to the earth's magnetic field (thus an absolute reference).

More on IMU navigation can be found in [55]. The basic principle to obtain the pose from these measurements is to integrate the rotational velocity and linear acceleration signals. However, due to integration of sensor noise and bias, pose estimates based on IMU data alone will quickly accumulate errors. To reduce the impact of these errors, so-called aided Inertial Navigation Systems (INS) have been proposed [56]:

- **Laser-aided INS (LINS)** These methods typically rely on the existence of structural planes and are not easily generalizable to cluttered environments.
- **Vision-aided INS (VINS)** Fuse data from a camera and an IMU and can operate in both structured and unstructured areas. VINS methods have the additional benefit that both inertial and visual sensors are lightweight, inexpensive and passive, hence requiring a smaller power budget compared to LINS.

In the context of this work, since drones are to be able to navigate in a priori unknown - thus possibly cluttered - environments, we are particularly interested in a VINS solution.

VINS (Vision-aided Inertial Navigation Systems)

Important literature coverage exists about this topic. Interesting papers about Vision aided IMU navigation (including state space model development for Kalman filtering) are, for example: [57] [58] [59] [60] [61] [62] [63] [64] [65], and references therein.

Mourikis et al. [66] applied IMU and camera fusion for a concrete problem of planetary landing. It explains pretty well the problems encountered in GPS-denied environments.

However, these papers do not address the problem of estimator inconsistency in VINS [5]. Indeed, in [67] it is shown that linearized estimation approaches, such as the Extended Kalman Filter (EKF), can fundamentally alter the system observability properties. They thus often suffer from inconsistency when applied to VINS. While in most of the recent research, emphasis has been put on 2D, very little is known about the inconsistency of 3D localization. This is primarily due to the complexity of the motion and measurement models involved in estimating a 15 (instead of 3) dimensional state. In this paper, Observability-Constrained VINS (OC-VINS) methodology is proposed to address the inconsistency issue to track the 6 D.O.F. pose of a camera.

All previous references to VINS methods supposed a feature-based visual odometry. However, it is also possible to fuse IMU information to optical flow measurements. A solution has been proposed in [69], where the fusion model has been extensively tested on a real quadrotor.

A few other VINS techniques have been tested at the *M.A.R.S. Lab* (Multiple Autonomous Robotic Systems)[6]. Comments and comparisons are provided online [70].

Hardware

Low cost IMU's performances are in constant progress while their cost keep plummeting down. But pose estimation based upon on-chip IMU's is still suffering of angular velocity and linear acceleration biases. To avoid integration errors, a new type of device has been imagined, the ARHS (Attitude and Heading Reference System). It provides 3D orientation by integrating gyroscopes and fusing this data with accelerometer data and magnetometer data. With sensor fusion, drift from the gyroscopes integration is compensated by reference vectors, namely gravity and the earth magnetic field. This results in a drift-free orientation. The key difference with an IMU is thus the addition of an on-board processing system to provide 3D orientation [71].

Besides the hardware component, a software implementation of such filters is also possible, namely the Mahony [72] and Madgewick [73] filters are among the best known. Opensource code for a Madgewick filter in multiple languages are available at [74]. In [75] Mahony and Madgwick filters are compared to Kalman and to complementary filters. A tutorial to implement it on a Navio2 board (see Section 2.5) is available at [76].

[5] As defined in [68], a state estimator is consistent if the estimation errors are zero-mean. At the opposite, an inconsistent estimators errors grow over time, possibly causing divergence.

[6] `http://mars.cs.umn.edu/`

2.2.5 Motion capture systems

This technique has already been discussed, and the fact that it needs a preparation of the environment prohibits its use for truly autonomous flight applications. Other drawbacks are its very high cost, and the fact that it does not scale too well for large swarms of drones, since drones can temporarily hide from the cameras line of sights by flying behind each other.

However, for development purposes, it presents some interesting benefits. For example, having a good pose estimation is very useful in order to be able to concentrate on the controller, the dynamics of flight, and the ability to quickly execute maneuvering (agility) [30].

Second, it is very difficult to develop all-embedded pose estimation techniques without disposing over a ground-truth measurement. Following this idea, [77] proposed a benchmarking tool for MAV visual pose estimation using a motion capture system.

For these reasons, equipping the lab with such a system could be of great benefit for further development of control and pose estimation techniques in a totally independent and decoupled fashion.

A commercial example of such a system is provided by the firm *VICON motion systems Ltd.* [78], which offers millimeter resolution with camera frame rates of up until 420 fps. This system fits peculiarly to the problem addressed in this work, since a ROS driver exists for this product [7] (see Subsection 3.2.2 for more about ROS). A previous example of application in the field of research on multicopters is related to the construction of the "Flying Machine Arena", at ETH Zurich, which provides ground truth for pose measurements with sub-centimeter accuracy at a frequency of 200 Hz [79].

2.3 Computer vision

As already explained, a way to estimate the pose in an absolute fashion is to use visual SLAM. This technique relies on extracting some points of interest detected in an image. A point of interest is useful if it can be detected in several images and corresponds to a same point in the real world observed by the camera. By comparing and tracking the detected keypoints between consecutive images of a video stream, we can estimate the motion of the camera in the scene. Thus, we can recover the motion of the vehicle. In this section, methods will be presented to detect, describe, match and track these points of interest called features or keypoints.

Feature detection and description

A State of the art in detecting keypoints in an image for solving visual SLAM problem is given in [80][81].

The aim of feature detection is to decide whether any pixel of an image is a point of interest or not. The features detected will be used for solving the correspondence problem, i.e. the data association between two different images. To be efficient, the detection must be invariant to several disturbances induced by the measurement sensor, the camera. These disturbances are notably the orientation, the perspective, the scale, the illumination and partial occlusion.

[7] `http://wiki.ros.org/vicon_bridge`

There are mainly two families of detectors: corner detectors and blob detectors. The first type detects keypoints at the intersection of lines. On the other side, blob detectors use a function to describe the keypoints location in an image. Either the derivatives of the function w.r.t. the position or the local extrema of the function are used.

Another requirement to solve the correspondence problem is that keypoints must be described by recognizable features to be able to compare and associate similar pairs. Therefore, a keypoint descriptor must be highly distinctive and robust to the same disturbances previously cited for detectors.

Let us cite the most common techniques for keypoint detection.

Harris and Stephens proposed in 1988 [82] the so called Harris corner detector, which is an improvement of the Moravec's corner detector. Harris corner detector uses the local autocorrelation of the image. But it is not invariant to scale.

Scale-Invariant Feature Transform is introduced by Lowe in 1999 [83]. SIFT keypoints are the maxima or minima of a Difference-of-Gaussan (DoG) function of the image rescaled at different levels. This detector became a reference for its robustness.

In 2002, Mikolajczyk and Schmid used the Harris detector and the Laplace function to select the scale of the keypoints detected [84]. Harris detector has good stability: high repeatability of keypoints detection allows robust matches.

Features From Accelerated Segment Test (FAST) is presented by Rosten and Drummond (2005) [85]. The test uses a 16 pixels circle to determine if a point is a corner. A non-maximum suppression algorithm is used to select less keypoints. This detector is faster than other detectors and select a large number of keypoints.

Bay et al. (2008) has been inspired by SIFT to develop Speeded up Robust Features (SURF) [86]. They used an approximation of the Gaussian smoothing and the determinant of the Hessian to select the location and the scale of features. SURF is considered to be robust to illumination changes, other properties are comparable to SIFT. The time performances are better than SIFT but SURF is slower than Harris and FAST.

In 2012, the KAZE (*wind*, in Japanese) detector is proposed by P. F. Alcantarilla et al. [87], the keypoints are detected using nonlinear scale spaces by means of nonlinear diffusion filtering. The comparison with Gaussian blur is illustrated in Figure 2.7.

Figure 2.7: Nonlinear diffusion filter used by KAZE [87]

On the first row the image is blurred using the Gaussian filter (linear diffusion), on the second row the nonlinear diffusion filter is used. The object boundaries are better retained with the nonlinear filter, and noise reduction is effective.

For what concerns the description, a first naive idea of descriptor could be to use local derivatives around the feature. Several papers were published to detect scale and orientation

based on derivatives and grayvalue. For example, the SIFT descriptor [88] is an histogram for gradient magnitudes and orientation. Its dimension is 128 as explained on Figure 2.8. Several improvements to SIFT descriptors were proposed like SURF, PCA-SIFT, etc. [89].

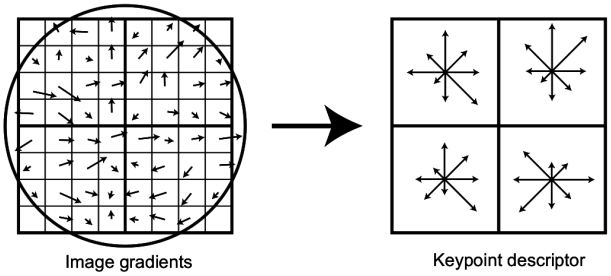


Figure 2.8: SIFT descriptor [88]

Gradients magnitude and orientation are computed around the keypoint (on the left).
8 bins-histograms summarize 4x4 subregions (on the right).
The figure illustrates 2x2 descriptor array, SIFT uses 4x4 array, SIFT dimension is 4x4x8=128.

But other types of descriptors are described in the literature. For instance, binary descriptors are composed of three parts: a sampling pattern to determine points around the keypoints that will be used, a measure of the orientation and a sampling pair that are compared to build the descriptor. For example, BRIEF [90] is the first binary string descriptor, while ORB [91] is an improvement of BRIEF. Binary descriptors computation and comparison are faster than SIFT-like descriptors but are less robust.

To compare descriptors, the euclidean distance is generally used for descriptors like SIFT and SURF, while the hamming distance is better suited for binary descriptors.

Hardware

Some hardware aspects can also be addressed, like the impact of using a fish-eye camera on the effectiveness of visual SLAM algorithms [92]. Indeed, it might be desirable to have a larger field of view to take a larger portion of the environment into account. However, the field of view of a camera is a sensitive parameter of the camera model, from which can greatly depend the performance of pose estimation, for example.

Another hardware aspect is the number of fps (frames per second) treated by the computer vision algorithms. This number is obviously upper-bounded by the camera frame rate, but is often less, since detection of keypoints is very computation costly. To address both problems, DVS (Dynamic Vision Sensor) are a new type of cameras that capitalizes its resources on capturing only pixel changes, thus constantly adapting the image emission rate. These types of sensors are able to reach much larger fps, thus avoiding blur during highly dynamic movements. Second, since only pixel changes are transmitted, it is possible to greatly simplify the computer vision algorithms so that they perform quicker (on an event-based fashion). In [93], an example is given for a DVS camera used for egomotion during High-Speed maneuvering. A very impressive video is also available at [94].

Yet another state-of-the-art sensor to replace common cameras are RGB-D sensors (Red, Green, Blue - Depth). These devices instantaneously acquire both color images and depth of the surrounding environment. Indeed, a depth information is coded at each pixel of the image, providing a "depth image". As of 2012, most popular sensors were the Kinect sensor and SwissRanger SR-3000 [95]. But other commercial and more and more lightweight solutions exist and the evolution of the cost, which remains the main limitation, is closely watched by the robotics

community [96]. A lot of work has already been done on (semi-)dense scene reconstruction, odometry, RGBD-SLAM, etc (see [95] [97] [98] [99] among many others).

Finally, because the computational costs of computer-vision algorithms are very restrictive for real-time applications, and because a large market for VR/AR (Virtual Reality/Augmented Reality) is emerging worldwide, there are currently projects of hardware optimisation for computer vision tasks by designing new types of dedicated processing units [100]. Meanwhile, some researchers have implemented hardware versions of descriptors an FPGA (Field Programmable Gate Arrays), such as SIFT and SURF [101] [102], and some SLAM and feature detection algorithms have also been adapted to be run on GPU for substantial timing improvements [50].

2.4 Mapping

We have seen in the SLAM section that a drone can create a map of an unknown environment at the same time as it explores and locates itself in it. Once the map is created, it can be used for other purposes such as localisation. Indeed, whether or not it has been used for location, other applications could use a map, in real-time (in flight) or offline (after flight). Some examples include:

Offline scene reconstruction

Pix4Dmapper[8] and *Dronemapper*[9] are examples of commercial services that provide software that automatically converts images into georeferenced 2D maps and 3D models like NDVI (normalized difference vegetation index), orthomosaics, textured 3D models, and so on. Application fields include

- archeology
- agriculture
- environmental monitoring
- mining
- etc.

Online scene reconstruction

In robotics, a dense reconstruction is desirable to interact with the environment, as in obstacle avoidance, path planning, and manipulation tasks [103]. This is why dense reconstruction should be possible on-board.

One original application that builds upon recent developments in live dense reconstruction is [104], which uses the dense map to address the problem of airground localization (see Figure 2.9).

Another example lies in active mapping. In [105], a method was proposed to choose motion trajectories that "minimize perceptual ambiguities inferred by the texture in the scene". In other words, the map is used to help the drone moving in a way that helps the map to keep consistent with the world. This is why it is called active mapping. Another work on active mapping [106]

[8] https://pix4d.com/

[9] https://dronemapper.com/

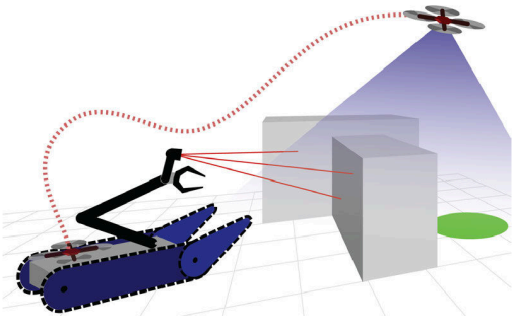


Figure 2.9: A live dense reconstruction map helps air and ground robots to communicate about their environment [104]

proposes other methods to evaluate the quality of camera motions with respect to the generation of new useful map points.

Higher level interpretations of the map are also possible, such as in [2], where river and road recognition are used for rescue operations, to optimally search an outdoor area to find a victim.

Use of a 3D point cloud for navigation tasks (Path planning)

Apart from dense reconstruction, voxels (contraction of volumetric pixels) can also be used. In this approach, the world is represented as a 3D grid. Each element of the grid (voxel) can be safe for navigation or not, depending on whether some object has been detected inside the elementary volume or not. A simplistic way to build and update the voxel grid using a point-cloud would be to compute the number of keypoints inside each voxel. If the voxel contains more than a certain number of points, it is set to non navigable. However, such a map can quickly grow too big for online applications, due to memory limitations and computational cost. To reduce this problem, octrees can be used. This technique permits to refine the voxel size only where it is necessary (see Figure 2.10[10]).

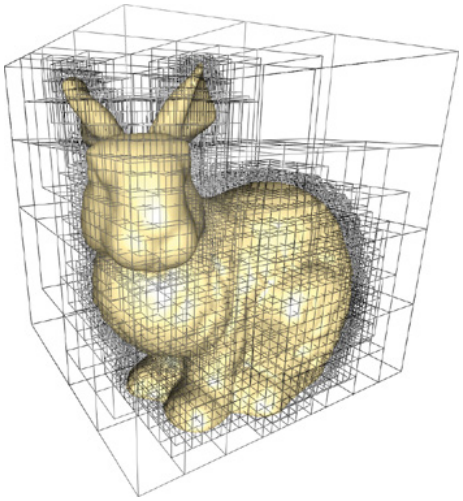


Figure 2.10: Illustration of octrees principle

Voxel grids as well as octree techniques are both available in PCL (Point Cloud Library[11]). Illustration of successful implementations of both techniques for range sensors-equipped drones are available at [107] [108].

[10] image taken from `http://http.developer.nvidia.com/GPUGems2/gpugems2_chapter37.html`

[11] `http://pointclouds.org/`

While in the above work, voxels are placed all around the drone to enable path planning, in [109], they build a fixed size map covering the area immediately underneath the robot. This results in an elevation map that can be used for safe landing of the drone (see [110] for a convincing video of an actual on-drone implementation).

However, navigation using an on-board generated map is not always robust, in the sense that a misrepresentation of a scene can lead to spurious displacements. In [111], the authors argue that motion planning for vision-controlled robots should be perception aware. This means that the robot must favour richly textured mapped areas to minimize the localization uncertainty during a goalreaching task. They also describe how to incorporate photometric information of the scene, in addition to the geometric information of displacement, to compute the uncertainty of vision-based localization.

Finally, let us point out that some point-cloud interpretation features are implemented in PCL, which we will use in the context of this work, like plane recognition [112].

Collaborative Mapping

With collaborative mapping, multiple robots can work together to map larger areas. Present challenges are mainly to communicate efficiently the map fragments, fuse them correctly, and to perform this computation in a way that does not interfere with the localization and control loops. In [113], an 'Internet of drones' is used to perform map communication, storage, and computation (see[12] for an impressive video). A commercial example of a cloud for drones is UAVIA[13], proposed by a french company. Cloud based technology also enables long distance control. On their website, we can read that the firm was able to fly a drone in San Francisco from its base in Paris, enabling aerial inspections at distance.

But yet another challenge is the decentralised computation on the map in an autonomous swarm of robots. This point is addressed in [114] [115] and references therein.

In the case of collaborative SLAM, the robots share the map in which they localise in an absolute manner. This enables collaboration on higher level tasks than just exploring and mapping. Interesting papers addressing this topic are [116] and [117].

2.5 Hardware

In this section, we will briefly review different flying platforms, drone controller boards, and embedded computers that are interesting for the type of applications we consider, or simply for development purposes.

[12] https://www.youtube.com/watch?v=sZBSQrks5Hw
[13] http://uavia.eu/

Drones

Flyability SA, an EPFL spin-off, has recently finished to develop the *Gimball* [118], a drone tolerant to collisions. Its configuration makes it particularly interesting for applications such as confined environments flight (indoor, caves, complex industrial infrastructures, etc.) and swarm flight. The *fleye* [119] is another example of collision tolerant drone using only one rotor. Unfortunately, these examples have limited development purposes, since they are not open-source.

The *Econocopter* [120], and mostly the *AeroQuad* [121] are examples of low cost totally open-source (for hardware as well as for software) drones. The second one is maintained by a large community. Its hardware parts are sold as spare parts on the website, including large choices of configurations and parts variations.

Some very well packaged commercial platforms are promising strong research and educational capabilities, such as the DJI *Matrice 100* "fully customizable and programmable flight platform" [122]. But their cost seem prohibitive for drones that are to endorse harsh manipulation during development. A second drawback (as it is the case with the hardware currently used at our lab [123]) is that we never know beforehand the amount of un-penetrable, un-documented firmware that is typical of non open-source platforms.

But from all developer-oriented flying platforms, the *Crazyflie 2.0* [124] is a ros compatible open-source nano-copter that has made its proof, notably for aggressive controlled manoeuvres and in swarm flight. It is used, amongst others, at ETH Zurich, University of Illinois, Technische Universität München, NASA, Stanford University, and so on. However, even if its very low weight makes it well suited for aggressive maneuvering, it is less suited for sensor integration, with a maximum payload of 15 g.

Autopilots

As it has already been mentioned, our current drone (AR.Drone 2.0) is a limited development platform. Indeed, since it is closed source, we do not know exactly what controller is on board, nor which sensor fusion is performed. These variables can interfere with our algorithms, notably our position controller and state estimation algorithms. To counter this, the electronics inside could be replaced by a combination of a custom modular controller board (often referred to as an autopilot) and an on-board computer.

Autopilots are small, robot oriented, lightweight compaction of a micro-controller (to run the stabilization controller and or multiple controller modes), various communication ports (UART, I2C, SPI, CAN, Wi-Fi, ...), sensors (essentially IMU's and barometers), and PWM (Pulse Width Modulation) signals to the motors.
The word flight controller may also be encountered. The difference between autopilots and flight controllers is not strictly fixed, but the term flight controller is generally used for lower-level boards, including less sensors, and with only a velocity control rather than a position control. For this reason, these chips are less suited for autonomous flight than for remote controlled flight.
The software part of the controller is often referred to as "flight stack". As of today, the two main open-source flight stacks are the APM [125] and PX4 [126], maintained by the Dronecode

association (Linux foundation). But others exist, like *Autoquad* [127] (open-source software, closed source hardware), or *Paparazzi* [128]. Examples of boards are:

- The *Pixhawk* [129]. It is developed by an open-hardware project aiming at providing high-end autopilot hardware to the academic, hobby and industrial communities. Recently, a *Pixracer* autopilot has also become available on their website. The advantage of this board is its interfaceability with ROS, through the MAVLINK (Micro Air Vehicle Link) protocol, for which a ROS driver is available: MAVROS.
- The *APM2* (Also Ardupilot Mega) [130], based on the Arduino Mega platform, also runs the APM flight stack.
- The above mentioned AeroQuad uses the STM32-based AeroQuad32 flight controller board [131], or an Arduino microcontroller (Mega 2560 or Uno). A separate sensor board is available (accelerometer, gyroscope, magnetometer, barometer, ultrasonic sensors and GPS).

Embedded computers

Autopilots are not to be confused with on-board computers, which offer extended computation capacities for running custom high-level software. Embedded computers (single-board computers or computers-on-module (COMs) or companion computers) have to be envisaged for long term development. Meanwhile, we are using a fixed PC for development purposes. Indeed, the assumption is made that today's developments that need high computing resources, will soon be able to run on smaller and cheaper devices. Examples of currently available open-source single-board computers are: Raspberry Pi, Odroid, Beagleboard, OLinuXino, Gumstix, 4Gmetry, and many others. Multiple declinations of these boards have been proposed to favour interconnection with autopilots. Some of them come with a pre installation of ROS, MAVLINK, OpenCV and other interesting open-source features.

New generation autopilots

The Pixhawk autopilot and the associated software has proven its viability through many research projects. However it has limited processing power. So, to perform more advanced tasks, a companion PC board was used, leaving the Pixhawk to just perform flight control. This past year, two open-source projects have proposed to run the well proven PX4/APM flight stacks directly on a real-time Linux:

- The *Navio2* [132]: it embeds multiple sensors, and the APM flight stack is executed directly on Raspberry Pi with real-time Linux kernel. Developer applications can be run alongside. It can communicate via MAVLINK.
- The *Erle-brain 2* [133] is even more recent. It runs Debian Linux on a BeagleBone Black single board computer, and also runs the ROS Hydro Medusa and even provides support for ROS 2.0. Very similar to the *Navio 2*, a thorough comparison is provided at [134]. Erle Robotics also propose an "Erle-copter", a resulting Linux-quadcopter.

Total hardware architecture

In [103] one can find an example of successful integration of an Odroid card and a Pixhawk on a Parrot Ar.Drone to perform autonomous vision-based flight and live dense 3D mapping (see Figure 2.11).

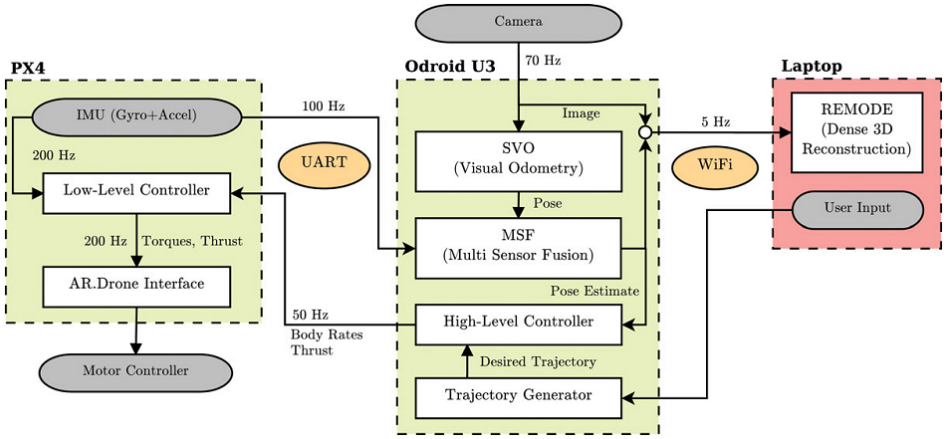


Figure 2.11: Hardware configuration used in [103]

2.6 Autonomous flight with quadrotors

The previous sections of our state of the art all addressed one technical aspect at a time. This section provides some examples of successful works on real drones.

C. Bills et al. [135] proposed a method based on a monocular camera to determine the direction of flight for a quadcopter in indoor environment. Their solution do not use any mapping. A perspective cues detection coupled with an appearance based classifier allow to identify the kind of indoor environment (see Figure 2.12). In corridors, parallel lines converges to a vanishing point which is used to locate the end of the corridor. In staircases, horizontal lines indicates the direction of the upward direction and the center of the stairs.

SLAM framework with ground robots is a successful research area in robotics. Using these with aerial vehicles is more complicated than mounting the same sensors and running the same algorithm. The specific behaviour of flying robots has to be taken into account. These challenges summarized in [136] are the following:

- **Limited payload**: sensors are limited,
- **Indirect odometry**: there is no wheel encoder to measure traveling distances,
- **Fast dynamics and constant motion**: delays are more difficult to handle.

C. Troiani et al. [137] tested state-of-the-art VINS on a real quadcopter, focusing on low computational-complexity algorithms. J. W. Langelaan [60] also provides results for pose estimation using a VINS.

D. Scaramuzza et al. [29] give a very insightful report describing the results and the technical aspects faced during the SFLY project: "The Swarm of Micro Flying Robots (SFLY) project was a

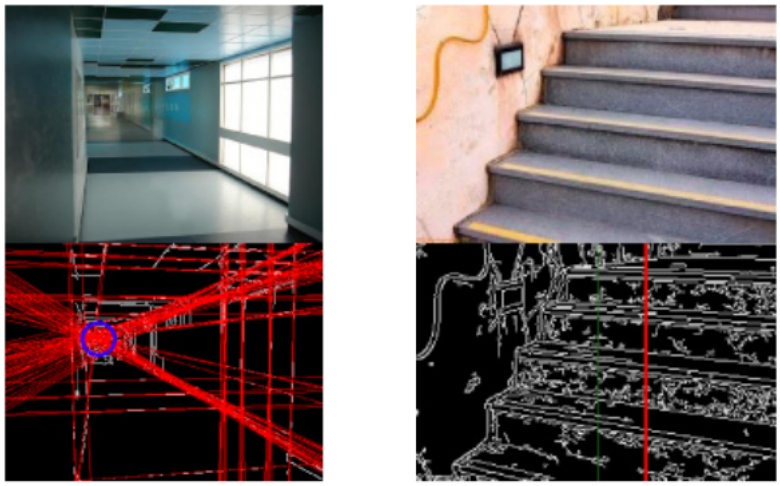


Figure 2.12: A perspective cues detection in corridors and staircases by C. Bills et al. [135]

European Union funded project with the goal of creating a swarm of vision-controlled microaerial vehicles (MAVs) capable of autonomous navigation, three-dimensional (3-D) mapping, and optimal surveillance coverage in GPS-denied environments". However, in SFLY, dense 3D maps are computed offline and are available only several minutes after landing.

A. Bachrach et al. presented in [138] a 2D SLAM solution for a quadcopter equipped with a laser rangefinder. The resulting map is composed of a grid of cells. This map is used by a high level planner to perform an exploration strategy.

V. Kumar et al. also used a quadcopter to perform autonomous flight in a multi-floor indoor environment [139][140]. The environment is assumed to be composed of vertical walls and horizontal ground planes, all piecewise constant. The vehicle is equipped with a laser rangefinder. They chose to develop an occupancy grid-based incremental SLAM. The map stores the obstacles position. The pose estimation is done with the iterative closest point (ICP) algorithm: ICP minimizes the distance between two point clouds, the one stored in the map and the last observations. They proposed a speed-up based on a grid based search. A camera is also embedded on the quadcopter: vision-based technique allows to detect loop-closure (see Figure 2.13). Finally, sensor fusion is performed with an EKF.

At the Technische Universität München (TUM), J. Engel et al. implemented a navigation system for the Parrot AR.Drone in small environment. Their approach described in [141] [142] is composed of three parts: a visual SLAM, an EKF is used to fuse visual pose estimation with other sensors and finally they use a PID as position controller. The visual SLAM is performed using PTAM on the frontal camera. Their work is publicly available under the name `tum_ardrone`.

Y. Fan and M. Aramrattana used `tum-ardrone` and developed a pillar detection and mapping system described in [143].

Nick Dijkshoorn [144] and Jens Nyman [145] wrote two impressive master theses. The first one proposes a SLAM solution, and the other proposes a wall detection algorithm based on PTAM. For the second one, a repository of the code is available[14].

[14] `https://github.com/nymanjens/ardrone-exploration`

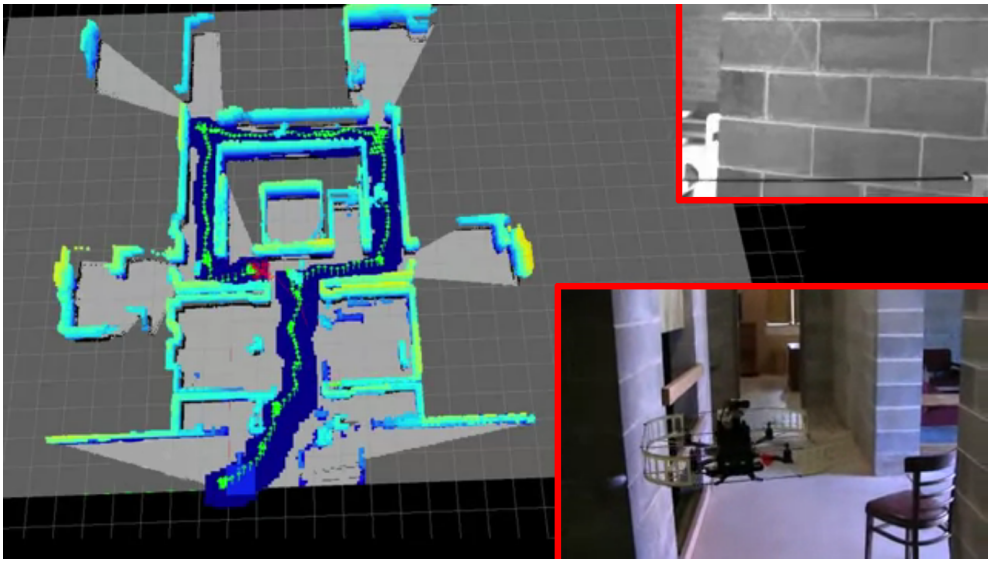


Figure 2.13: Map build and Vehicle used by V. Kumar et al.
(vehicle sold by Ascending Technologies, GmbH) [139][140]

B. Steder et al. [146] developed a SLAM method for flying vehicles. The aerial vehicle carries a down-looking stereo camera as well as an attitude sensor. It uses the graph-based formulation to build a visual map. To achieve online performance only a part of the graph is optimized when new constraints are added. They successfully mapped a 190m trajectory with a camera mounted on a rod as seen on Figure 2.14.

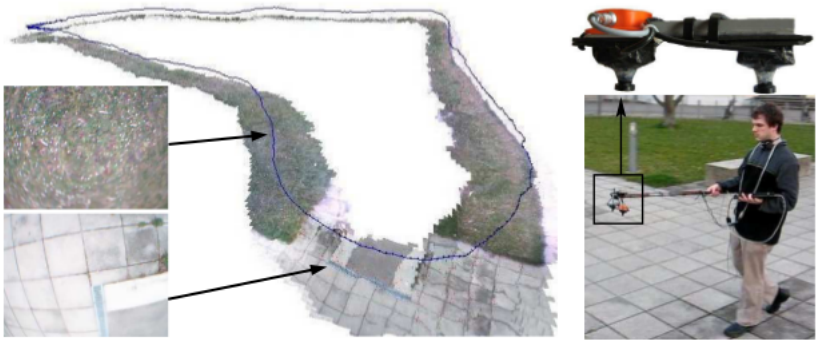


Figure 2.14: Device used by B. Steder et al. [146] and resulting visual map

As a conclusion, a lot of projects exist related to the topic of this thesis, as well as many practical implementations, and it would be impossible to provide an exhaustive list. But the implementations cited above all present limited performance and robustness. It was noted that a common implicit challenge was at the origin of the multiple problems: the pose estimation. Despite each providing an insightful state of the art, very little emphasis was put on this particular challenge. However, as we have seen in Section 2.1, once the pose is correctly estimated, very accurate and dynamic control can be achieved with current state-of-the-art drone controllers. This shows that the gap between achieving very complex tasks in prepared environments and achieving the same tasks in unknown environments is a good pose estimation. This final goal must be kept in mind as being the main challenge.

Chapter 3

Platform Architecture

In this chapter, two major topics will be addressed: the hardware and the software architecture of our development platform. The hardware part will focus on the quadcopter hardware as well as the other components of the platform which allow to work with several drones. For the software part, the new framework of this year - ROS - will be shortly explained. Then, the main open-source libraries will be introduced. Finally, the software architecture on which both groups of this year agreed will be presented.

3.1 Hardware Architecture

3.1.1 Hardware used

As stated before, a low-cost, commercially available quadcopter was used: the Parrot AR.Drone 2.0 illustrated in Figure 3.1 and Figure 3.2. It costs about 250 €, and weights 380 g with indoor shell and 420 g with outdoor shell. It can hardly be extended with more electronics, since each gram added strongly modifies its flight behaviour. Its maximum payload was estimated to 45 grams before really altering its basic flight functions. In spite of this, a small hardware modification was done to equip the drone with a camera of better quality, as explained in Subsection 3.1.2.

Figure 3.1: Parrot AR.Drone 2.0 with indoor shell (left) and with outdoor shell (right) [147]

In Figure 3.2, the original boards and sensors can be seen, these include:

Sensors

- 3 axis accelerometer $\pm$ 50 mg precision
- 3 axis gyroscope 2000°/second precision
- 3 axis magnetometer 6° precision
- Pressure sensor $\pm$ 10 Pa precision
- Ultrasound sensors for ground altitude measurement
- 60 FPS vertical QVGA camera for ground speed measurement
- HD Camera, 720p 30FPS
  - Wide angle lens : 92° diagonal
  - H264 encoding base profile
  - Fotos in JPEG-Format

Figure 3.2: Exploded view and inside view of the Parrot AR.Drone 2.0 [147]

Onboard processing capabilities

- 1GHz 32 bit ARM Cortex A8 processor with 800MHz video DSP TMS320DMC64x
- Linux 2.6.32
- 1GB DDR2 RAM at 200MHz
- Wi-Fi b g n
- 1 8 MIPS AVR CPU per motor controller

Onboard controller and sensor fusion The Parrot AR.Drone is known for its state-of-the-art indoor navigation systems combining low-cost inertial sensors, computer vision techniques, sonar, and accounting for aerodynamics models [148]. The sensor fusion that is performed on-board accounts for the low performance of each of its low-cost sensors, using for example inter-calibration. This implies a good flight quality for a very low cost. Unfortunately, this on-board navigation system is closed-source, thus ideal for an end-user, but irksome for a developer. For this reason, depending on the direction taken by the research in the INMA department, a change of hardware could be envisaged. Thus, all that was implemented this year was intended to be the less hardware-dependent possible.

A paper [148] is available for the first version of the Parrot AR.Drone, describing all the control and pose-estimation scheme of the on-board software, as well as the manual factory calibration principles that are applied on each drone produced. It can be consulted to give an idea of what is implemented in the Parrot AR.Drone 2.0, but with no guarantee that the newer version includes the exact same technology.

External processing unit As stated before, the software running on board is closed-source and the procedure to run self-made pieces of code directly on the drone is more a matter of hacking and is not well described. Some projects overwrite this software like Paparazzi UAV [128].

The approach chosen in this work is to use a traditional computer to perform all the processes we implemented. This means that all computations are centralized on one motionless unit. This is not perfectly attuned with the quadcopters autonomy constraint. However, our code architecture is designed as if it was embedded on the drone. The computer is used for development purpose only.

The computer unit has the following specifications:

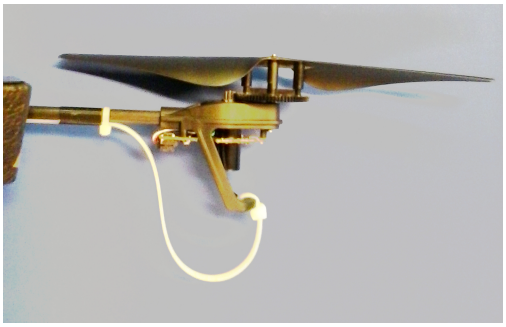
Intel 650 @ 3.2 Ghz x 4, and an 8 GB RAM running Ubuntu 14.04

3.1.2 Hardware modification

Since the bottom camera presents a significantly lower quality than the front camera, it was first envisaged to use the front camera for our developments into SLAM techniques. This approach also seems more logical for indoor navigation. However, being able to use the bottom camera revealed to be also interesting for multiple reasons, like the ability to track a moving target by flying over it. Moreover, for future developments, it is interesting to be able to use both cameras at the same time, which is not possible with the current Parrot hardware configuration. In addition to this, the poor quality of the bottom camera has been pointed out to be a limiting factor for some computer vision algorithms in previous master theses, and the replacement by a better quality camera has been proposed as a possible improvement. For these reasons, it was decided to upgrade the bottom camera.

After investigating several possibilities, an important number of candidate cameras have been found. With the help of the second team, a synthetic table containing the most relevant choices was established (see Appendix E). The main criteria were the cost, the weight, the quality and the ease of integration.

(a)



(b)

(c)

Figure 3.3: Parrot AR.Drone 2.0 modification. (a) The new bottom facing camera. (b) Raised legs system. (c) The "T" for doubling the power supply to the motherboards.

Finally, it was chosen to opt for a second Parrot front camera, mounted on a second motherboard ship. This modification was the simplest one, considering both the weight, the price and quality, and that all the ROS interfacing had already been done for the Parrot hardware. At the software level, we imagined a grouping of nodes from a ROS launch file that now permits to use both cameras indistinctly. Finally, once mounted with the help of Eng. F. Wielant, a lightweight system was put in place to raise up the drone legs in order to avoid collisions between the camera and the ground. The final result is shown in Figure 3.3, for a total added weight of about 45 g, which corresponds to our estimated maximum payload.

It is worthwhile noting that the SLAM code that was developed mainly with the new bottom camera, is also able to run on the basis bottom camera with practically no code manipulation and no performance alteration. This can be explained by the nature of the visual-feature descriptors we use (see Subsection 4.2.2).

3.1.3 Multi-drone

The multi-drone capacities with the Parrot AR.Drone were explored during the last academic year with the help of Nicolas Rowier, then computer specialist for the INMA Departement. The initial drone network configuration behave as a Wi-Fi access point. This network topology does not allow the use of multiple drones simultaneously. A bash script has been implemented to reconfigure drones in order to communicate with them trough one single router (simple illustration in Figure 3.4, see [16] for details). This script does not reconfigure the drone software permanently and the script has to be executed at each drone reboot (when the batteries are replaced). This script has been improved by our care as mentioned in Appendix D.

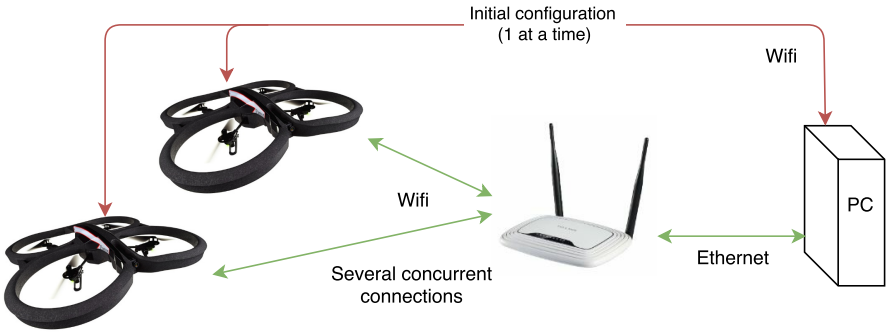


Figure 3.4: Multi-drone development platform with Parrot AR.Drone 2.0

3.2 Software Architecture

3.2.1 Parrot Software Development Kit (SDK)

Previous master thesis at INMA used the Parrot Software Development Kit (SDK) for the Parrot AR.Drone. This C library is an attempt of Parrot to attract developers to enlarge the panel of augmented reality games using their flying platform. It allows interacting with the Parrot AR.Drone through Wi-Fi, receive the navigation informations, send control commands and acquire the live video stream. However, several features to develop a modular code base are not provided with this kit (in contrast to ROS, described below) such as communication facilities between processes, a runtime parameter management system, a hardware abstraction layer, etc. The resulting code became difficult to maintain, and this was one of the main reasons for which it was decided to change the framework.

The Robotic Operating System (ROS) is a set of open source libraries and tools designed for the development of robot applications. It has been decided to use the ROS framework in this master thesis. The reasons will become clear in the next subsection.

3.2.2 Principles of the Robot Operating System (ROS)

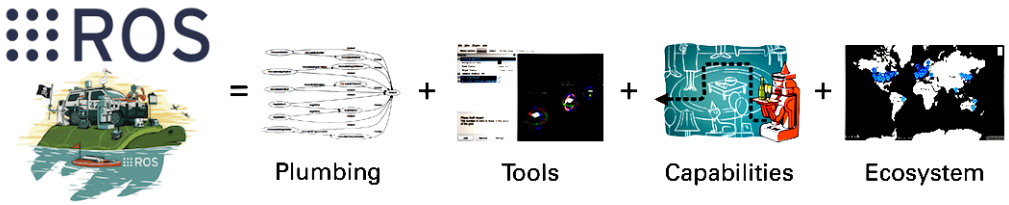


Figure 3.5: The ROS equation [149]

ROS is strictly speaking not an operating system. It is rather a framework or a toolbox that is actually an over-layer for a standard Linux OS. ROS provides functionalities from a robotic platform operating system that are desirable for a developer, like hardware abstraction through drivers, message-passing facilities between processes, a package management system, etc. The development environment is enhanced with building tools and inspection tools.

The package management system is the key feature to develop re-usable software. A large library of packages is available on the official website to encourage collaborative projects.

The processes executed at runtime are called nodes. The node is the basic abstraction which renders ROS so "modular at a fine-grained scale". Their dependences form a graph. These are "individually designed and loosely coupled at runtime". The Master is like the maestro who coordinates nodes. A node can contain any module of code, like a controller, a path planning algorithm, etc. which inputs and outputs are clearly defined, permitting a "black-box" approach.

Nodes have two communication channels. The first way to exchange messages is through topics. These are one-way broadcast communications. A node can publish a message on a topic and this message is received by nodes subscribed to this topic. Messages are data structures. Topics are addressed like in a file system with paths forming a succession of group names ended by the topic name. The second communication paradigm is based on the request-response principle. A first node can provide a service. Another node can send a request. The first one processes the demand, provides an answer and closes the communication.

ROS also manages a parameters server, which is useful to pass parameters to nodes at runtime. These can be modified during the execution and influence the node execution if the node implementation supports it (this can be helpful for future developments).

A last feature is the use of launch files. These files (formatted in `xml`) are an easy way to configure all the parameters needed to run a node and to launch several nodes in one call in the terminal. In our setting, it also provides a nice way to launch all the software for concurrent drones.

All and all, ROS provides a better development environment than the SDK, due to its large-scale collaborative aspects, its modular approach that encourages granularity of the code, the fact that it is open-sourced and well documented, well maintained because of its growing popularity in the robotics community, and provides lots of libraries and packages useful for us (like `ardrone_autonomy`, see below). Even external popular open-source libraries now provide ROS compatibility and support, like the OpenCV and PCL libraries (see below).

Unfortunately, the current implementation of the ROS master does respect the real-time paradigm. Indeed, since it runs on the top of a non-real time linux kernel, the execution timings of each node are non deterministic. However, propositions have been made to enforce real-time compatibility [150] of an upcoming ROS 2.0 [151], or permit to combine real-time components and standard ROS utilisation [152].

3.2.3 Libraries

The aim of using existing open-source libraries is to avoid re-coding what is already functional and available, and thus avoid reinventing the wheel.

OpenCV

OpenCV is a computer vision library which is distributed with an open source license. It provides a clean API to perform various tasks from picture analysis to structure from motion reconstruction. Some of the available functionalities need efficient machine learning algorithms which are implemented within the library. This library is easy to interface with the ROS video messages type. The use of different algorithms of this library is described in Chapter 4.

PCL

The Point Cloud Library (PCL) is an open source library for 3D point cloud processing. It provides various tools for perception in robotics like noisy point cloud filtering and scene segmentation. Another feature of interest for multi-robots mapping is "point cloud registration": this is the procedure to find the transformation to align several point clouds as illustrated on Figure 3.6.

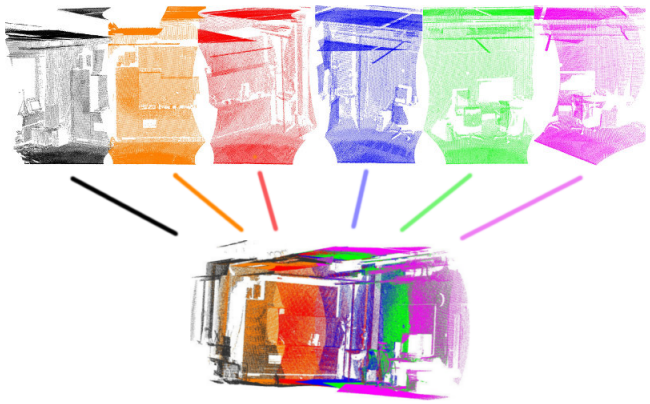


Figure 3.6: 3D point cloud registration

The use of this library permits to save time on the implementation and to rely on well-maintained algorithms. Not only does it save time when it comes to implement a map structure by our own, but also will it permit more easily the use of advanced techniques in future developments. More on this will be found in Subsection 4.5.1.

`ardrone_autonomy`

This piece of software is a ROS package initially developed at the Brown University and is currently maintained in the Autonomy Lab of Simon Fraser University[1]. This driver internally uses a modified version of the Parrot SDK to expose the Parrot AR.Drone to ROS facilities. It provides topics to read sensors, video streams and to send control commands.

[1] `http://autonomylab.org/ardrone_autonomy/`

3.2.4 Multi-drone

As mentioned above, Nicolas Rowier provided a nice way to reconfigure drones and to communicate with them trough one router. Each drone has its own IP address. Here, the focus will be put on how we managed to let several drones to be addressed inside the ROS software.

A multi-drone configuration can be handled easily thanks to the ROS framework. Indeed, launch files allow to configure several groups to wrap a collection of related nodes. In this way, each collection of nodes corresponds to one quadcopter. The execution of the software is the same as if one quadcopter was running at the time, with the exception that all topics and services are prefixed by the group name. This was implemented so that all topics are used 'internally'. Indeed, the topics names are resolved inside the name space of the group with their relative path. Topics devoted to inter-drones communications simply need to be prefixed by the group name of the corresponding drone. We solved this by the implementation of a node called `swarm_initialisation`. Each drone sends a message to attest it is available and the `swarm_initialisation` node broadcasts messages to notify the group name of each quadcopter. This approach was chosen because it does not limit the possibilities for any choice of topological multi-agent network for inter-drone communication.

For future developments, when the developed software will be running directly on the drone hardware, ROS is designed to run across multiple machines and only one machine needs to run the master ROS core. This matches perfectly with the multi-robot needs.

3.2.5 Code architecture

The major concern in the code architecture is a clean structure and modularity. This must be achieved to allow working on smaller pieces without understanding the whole detail of each components. Figure 3.7 shows the code architecture imagined by our two teams of students, revised on many occasions until April. It should be compared to the present implementation described in Section 4.7, where a more detailed overview is given.

Some clarifications are needed regarding what is explained in the report of the other group of students [153]. On the diagram in Figure 3.7, there is no direct communication between the Mapping, the Strategy and the Path planning nodes. Indeed, these two last nodes implemented by the other group use an internal structure to represent the explored and unexplored areas. This representation is also referred to as "map" while it is not linked to the 3D map involved in this text. Their implementation uses a 2D grid of cells with three states (wall, free and unexplored, and explored). It is used by their exploration algorithm to perform a target searching in a squared room with no obstacles. Neither an obstacle nor a wall detection is used and the room dimensions must be known in advance. This exploration structure is at the moment not based on any observation and must be differentiated with the map produced by the Mapping node which stores visual keypoints detected with computer vision algorithms. The details are explained in the next chapter.

In future developments, the 2D grid contained in the path planning process should be replaced by a direct interpretation of our 3D map. Our implementation choices were oriented towards this end (see Subsection 4.5.1).

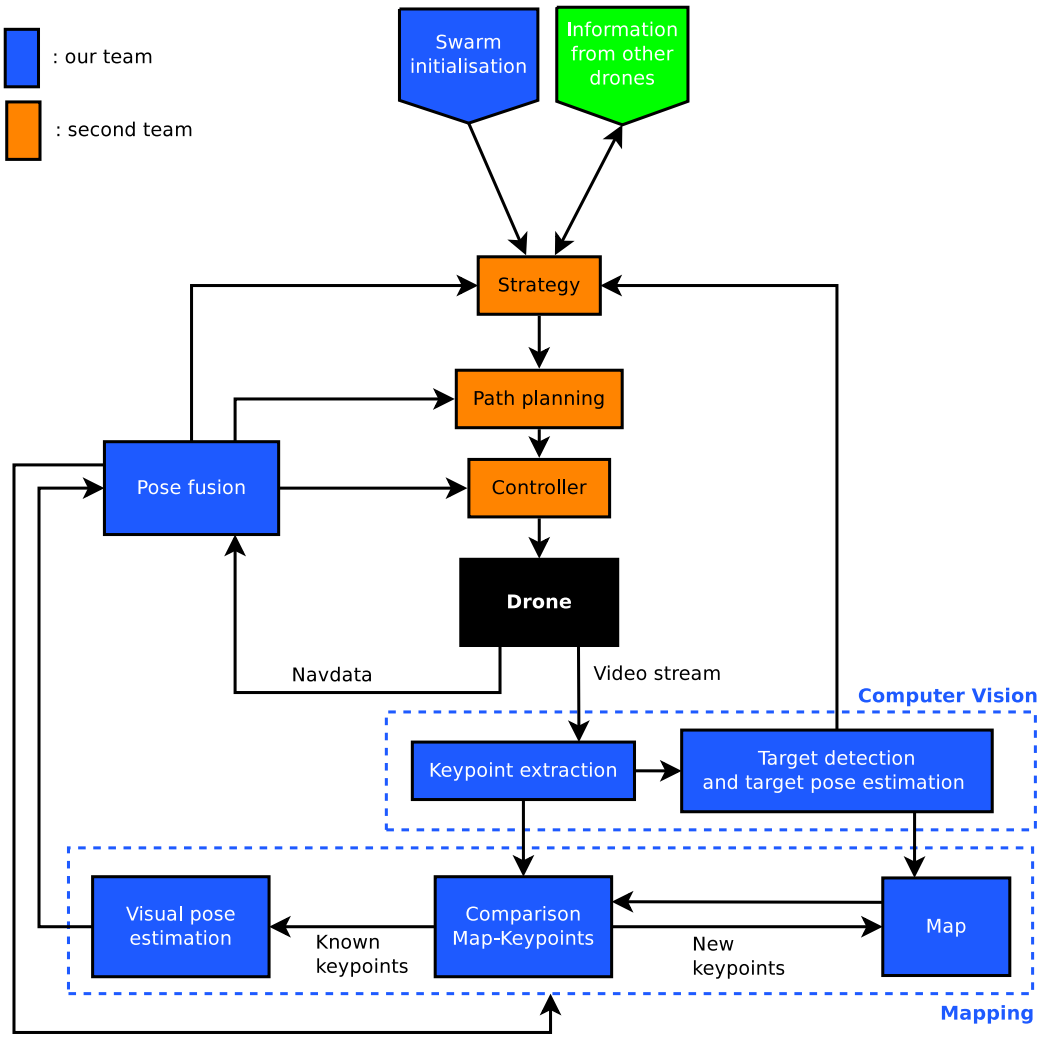


Figure 3.7: The code architecture imagined by our two teams of students

Chapter 4

Design and Implementation

This chapter is devoted to the explanation of our design choices and to the description of implementation details.

A lot of topics related to our implementation are introduced in this chapter. The text is separated into several sections:

- SLAM: several solutions are envisaged to conclude we need a new flexible implementation;
- Computer Vision: as it was chosen to use an embedded camera, an adapted observation model and information extraction are discussed;
- Target recognition and position estimation: a simple approach is presented to detect a 2D moving target;
- Visual odometry: in this section we present the method used to compute the pose estimation based on visual informations;
- Mapping: the strategy to build the map is discussed;
- Pose estimation: our original approach to put in place a simple fusion between direct sensor readings and our visual odometry is explained here;
- Code architecture: the previous sections were devoted to the explanation of several choices of algorithms. Here we give the summary of all interactions between the different parts of our implementation and the code design choices.

4.1 SLAM solutions tested

As discussed earlier, our main concern is to provide a sufficiently accurate estimation of the position and orientation of the drone to achieve various tasks. SLAM (Simultaneous Localization And Mapping) is the centerpiece of our developments towards this end.

As explained in Subsection 2.2.3, the SLAM problem for a quadcopter can be described as follows. The quadcopter starts at known coordinates in an unknown environment. The goal is to provide a continuous estimation of the position and orientation (pose) of the robot. This pose estimation is based on a map that is built in real-time. This is in fact a recursive problem. Indeed, to build the map, one needs an information about the scene in which the drone operates, but also the drone position when it *sees* the scene. Thus the pose is needed for the construction of the map.

The motion of a quadcopter is more prone to uncertainty than, for instance, the motion of a wheeled robot. Indeed, the absence of contact with the ground surface prevents to estimate the position by measuring a wheel revolution, which is the principle of classical odometers. Besides, the use of only proprioceptive sensors like Inertial Navigation System (INS) leads to errors on the drone state which accumulates with the elapsing time (see Appendix B). To build a map, sensors able to observe the state of the environment are needed. The only ones available on the Parrot AR.Drone are the cameras[1]. In this case, the term *visual SLAM* is used.

The visual SLAM technique relies on extracting some points of interest detected in an image. A point of interest is useful if it can be detected in several images and corresponds to a same point in the real world observed by the camera. These are the points that are incrementally added to the map. By comparing and tracking the detected points between the current image of the video stream and points already mapped, we can estimate the motion of the camera in the scene. Thus, we can recover the motion of the drone.

After exploring the state of the art (see Chapter 2), it appears that mainly three monocular visual SLAM algorithms have proven to function on low cost quadcopters: LSD-SLAM, PTAM, and SVO. These three successful implementations are available on ROS. We tested the first two in order to evaluate the eventual re-usability of some pieces of software that are maintained by a large community of researchers, hobbyists, and roboticists. However, we had to reject the idea of purely reusing these packages into our architecture for the following reasons.

PTAM in `tum_ardrone` PTAM presents the benefit that it has a fast implementation due to its architecture which decouples the tracking of the robot motion and the mapping of the environment. The `tum_ardrone` modification of PTAM is very interesting because a sensor fusion is directly coupled with the map building process. In this way, the estimated pose of the drone uses both the visual information and readings from proprioceptive sensors. The main drawback is that only a small area can be mapped. Indeed, at the initialisation of the algorithm, a few views are used to build the map. This process is not repeated. This implies that no more keypoints are added if a new area with unknown keypoints is discovered, preventing us from using this implementation for exploration. Indeed, this setting was chosen by the author in order to perform one precise

[1] there is also the ultrasonic sensor but it is pointing downwards and is thus not well suited to describe the environment

behaviour: the drone has to stabilize around a given position. It can slightly move, but with the restriction to stay close to the mapped area, and it is able to return to its position after some disturbances (for example, if someone pushes it or if the drone stops seeing the mapped area for a brief instant). `tum_ardrone` is the first SLAM solution we tested on ROS. Considerable time was spent to understand that the implementation was limited by the behaviour described above. The modification of the software seemed to be no good alternative, since a large community on the internet had already faced this problem, with no known solution to this day.

PTAM (adapted by ETHZASL) Another PTAM implementation for ROS is provided by the ETHZASL. It is accompanied with a sensor fusion software. The particularity is that the map built by PTAM does not use the fused information and seems difficult to be adapted. This map corresponds therefore not necessarily to the estimated pose used to control the quadcopter. According to us, this mismatch leads to the impossibility to use the map to perform, for example, path planning with obstacle avoidance. Even though these features were not planned for implementation this year, this seemed to be a limiting factor for future developments.

LSD-SLAM As a difference with PTAM, LSD-SLAM does not need to extract and to match features (see Figure 4.1). For this reason, its promises are [154]:

- much denser 3D reconstruction
- higher accuracy and more robustness in sparsely textured environments (e.g. indoors)
- much less outliers

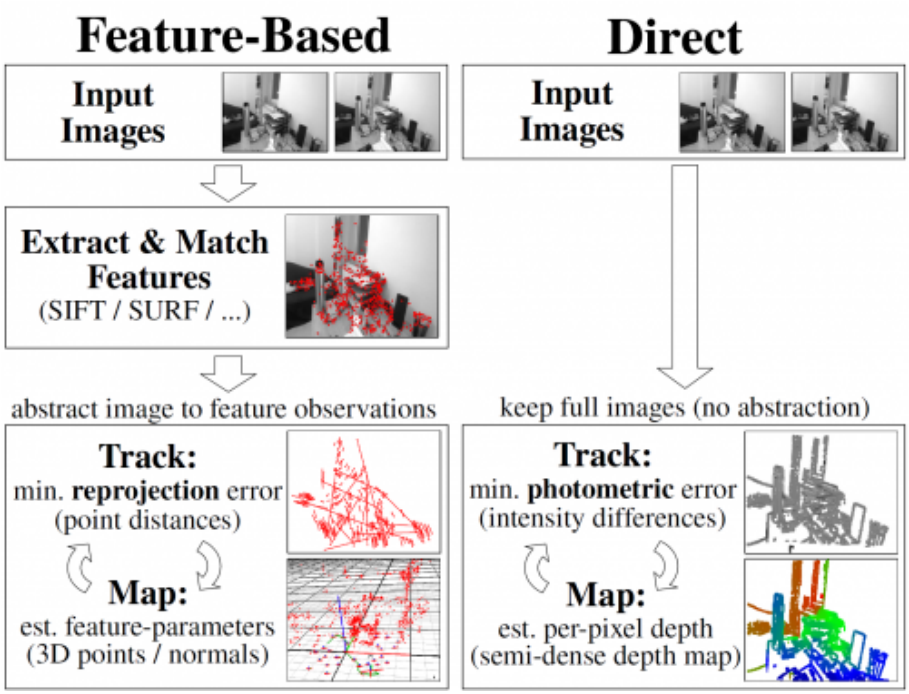


Figure 4.1: LSD-SLAM: Difference to keypoint-based methods [154]

In spite of this, the results obtained in our lab were deceiving since simple displacements already induced a lot of artifacts in the map. However, an interesting way to solve the large *loop*

closure issue can be used along with this implementation. For instance, if a drone makes a large loop trajectory, all the errors accumulated during the mapping will prevent the drone from knowing it is back at a previously known position, since both its map's scale and pose estimation will have drifted w.r.t. the real world. To avoid this, a complementary tool must be installed: *OpenFABMAP* [155], which ensures to recognize older scenes and permits to correct all the previously mapped points to add coherence to the map. Unfortunately, this requires training data (e.g. a collection of images of previously seen area is stored which is memory-hungry). But *OpenFABMAP* presents the advantage of being an independent tool that can therefore still be used in a *homemade* implementation of SLAM.

More generally During our tests of the PTAM and LSD-SLAM implementations on our drones in our lab, we noted

- Each of these solutions is provided as one large ROS node, which does not respect the ROS philosophy of modularity. Indeed, they are provided as turnkey solutions, thus not suited for development purposes, and limited documentation on the software implementation is provided;
- The structure of the map built is hardly accessible for other nodes performing other tasks;
- The computer vision part is not implemented in a separated node which avoid us, for example, to suppress points of interest on a known moving target (see Section 4.3).
- Both PTAM and LSD-SLAM, as well as SVO, are known to still present limited performance and robustness at the present time. Even if they could be used like turnkey solutions, doing so would deprive us of the opportunity to develop our knowledge in the area of visual localization for quadcopters.

As a conclusion, instead of putting together some black boxes, it was chosen to create our own implementation architecture and to implement a self made solution. Even though the result may not reach the same performances as state-of-the-art solutions, we would not be limited by design choices we have no influence upon. This leads us to better understand the internals of visual SLAM. The next sections of this chapter are intended to explain some of our accumulated knowledge on the subject and the choices which led us to our final implementation.

4.2 Computer vision

The first step towards visual SLAM consists of being able to manipulate images from a video stream. This includes intricate format conversions, the establishing of a projection to the world model, camera calibration, the ability to detect, extract, and match visual features, egomotion, and object recognition.

All these topics are part of the computer vision field. These are all detailed here, except for the two last of them, which have been described in the dedicated the specific Sections 4.4 "Visual Odometry" and 4.3 "Target recognition and position estimation".

Before entering the core of the matter, the terminology of this text must be clarified. The word *camera* will be used to represent monocular digital video camera. A video camera provides a video stream, a sequence of color images. Once the images have gone through the ROS driver for Parrot AR.Drone i.e. `ardrone_autonomy`, they are available in ROS format. At that moment, each image is represented as a full matrix. Each matrix element is called a pixel and carries the color information, i.e. an integer triplet (RGB) coded between 0 and 255.

As previously mentioned, the quadcopter Parrot AR.Drone embeds two cameras, one pointing forwards and the other downwards. Only one camera can be set to stream at a time. To process the video stream, the *OpenCV* library, which is easy to use with ROS (tools to perform format conversion are provided) has been used to manipulate the images and to perform some advanced computer vision algorithms.

4.2.1 A camera model and perspective geometry

In this section, an observation model is presented, which leads to the definition of some characteristics of a camera. This model will be useful in order to

1. Calibrate the camera to undistort the incoming images.
2. Establish a world-to-camera model (for egomotion).
3. The reciprocal model will be used to be able to construct the map.

Two different models are presented in this section:

- The pinhole camera model
- The lens camera model

The first model is the simplest and the most easy one to use. The second one is more adapted to real cameras.

Let us first define a 3D coordinates system attached to the camera (on the optics), with the z-axis pointing towards the scene (see Figure 4.2). This is called the camera coordinates system.

To describe the position of an element in an image acquired by the camera (on the image plane), the conventional image coordinate system described on Figure 4.3 will also be used[2].

[2] this does not respect the conventional matrix indexes

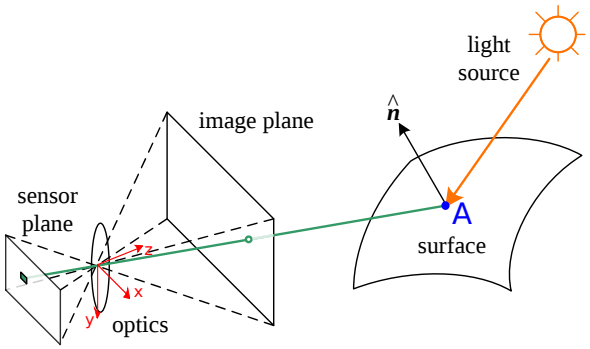


Figure 4.2: Camera projection model and camera coordinate system (in red) (adapted from [156])

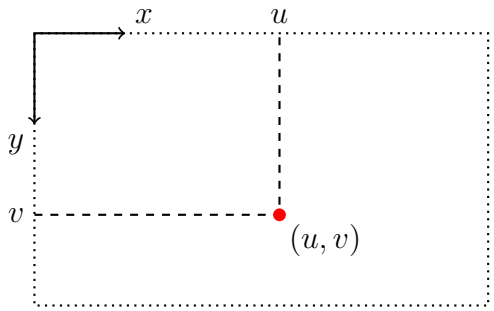


Figure 4.3: Image coordinate system: the origin is at the upper left corner, x-axis is pointing rightwards, y-axis is pointing downwards.

Pinhole Camera Model

We consider a center of projection C (the origin of the coordinate system) and the image plane $\pi \equiv z = f$ with f the focal length. The projection of a point $A = [X, Y, Z]^T$ is the intersection of the image plane and the line joining A and C, as illustrated on Figure 4.2.

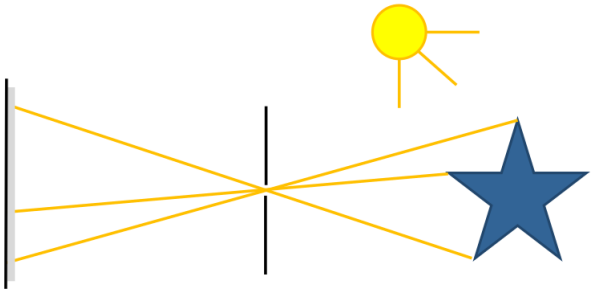


Figure 4.4: Simplified 2D pinhole camera projection [157]

This mapping corresponds to

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} \mapsto \begin{bmatrix} fX/Z \\ fY/Z \end{bmatrix}$$

Unfortunately, this model is too simple to describe real cameras. The following parameters are needed to take into account the camera projection. Non squared pixels are modeled by two distinct focal lengths for each axis f_x, f_y. The skew factor s models oblique pixels. The principal point offset coordinates (c_x, c_y) model the fact that the center of projection is not at the center of

the image. With the use of homogeneous coordinates, it can be expressed as:

$$\begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \mapsto \begin{bmatrix} f_x X + sY + c_x Z \\ f_y Y + c_y Z \\ Z \end{bmatrix} = \underbrace{\begin{bmatrix} f_x & s & c_x \\ & f_y & c_y \\ & & 1 \end{bmatrix}}_{K} \underbrace{\begin{bmatrix} 1 & & & 0 \\ & 1 & & 0 \\ & & 1 & 0 \end{bmatrix}}_{P} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \tag{4.1}$$

The matrix K is called the camera calibration matrix and P the camera projection matrix. K includes all the intrinsic parameters of the camera while P expresses the change from the world coordinate system of A to the coordinate system attached to the camera. P is the concatenation of a rotation matrix R and the translation vector t.

$$P = [R|t]$$

Lens Camera Model

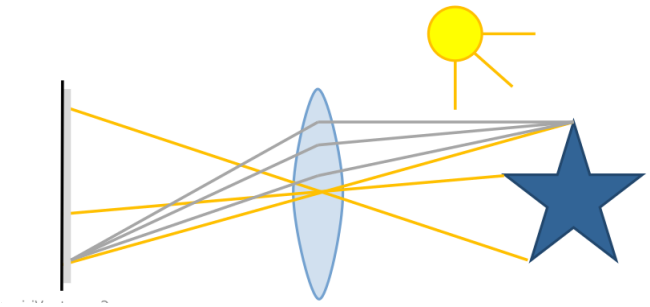


Figure 4.5: Simplified 2D lens camera projection [157]

With a perfect lens, the pinhole camera model is sufficient. But real lenses are not perfect and camera images are subject to radial and tangential distortion.

Due to radial distortion, straight lines will appear curved, producing what is called the "fish-eye" effect. Similarly, tangential distortion occurs because the lens is not aligned perfectly parallel to the imaging plane [158].

On way to describe those radial and tangential distortions is to use the Brown–Conrady distortion model:

$$\begin{aligned} r^2 &= (x_u - x_c)^2 + (y_u - y_c)^2 \\ x_d &= x_u(1 + K_1 r^2 + K_2 r^4 + K_3 r^6) + (2P_1 x_u y_u + P_2(r^2 + 2x_u)) \\ y_d &= y_u(1 + K_1 r^2 + K_2 r^4 + K_3 r^6) + (2P_2 x_u y_u + P_1(r^2 + 2y_u)) \end{aligned}$$

where (x_d, y_d) is the distorted image point and (x_u, y_u) the undistorted one, r is the radial distance, (x_c, y_c) is the center of distortion, K_n and P_n are respectively the radial and tangential distortion coefficients.

Calibration

Some of the algorithms used in the next sections consider a linear camera model as described above. Thus, they require undistorted images to function properly. Indeed, as the distortion depends on the relative position and orientation between an object and the camera, this can lead to some difficulties to recognize previously seen areas for instance. So it was chosen that the camera video stream had to be preprocessed to remove distortions.

Luckily, the parameters presented above are constants for a given camera. Thus, if they are known, a remapping of the pixels can be done. A camera is said to be calibrated if the camera matrix K and the distortion coefficients are known.

To identify these coefficients, the *OpenCV* function `calibrateCamera`[3] was used through the convenient ROS wrapping function `camera_calibration`[4]. Then, ROS allows to transmit the camera parameters along with the image stream messages, according to a standard camera driver scheme. To effectively correct the images, the ROS node `image_proc`[5] has been used.

To synthesize, Figure 4.6 shows the main idea of what was effectively implemented. The actions of the `image_proc` node are highlighted in orange. It is important to note that when the distortion is corrected, some empty pixels are generated on the borders of the image since it is not perfectly rectangular anymore. Therefore, `image_proc` crops these borders in order to keep a rectangular image. Then, the pixel width on height ratio is changed in order to have the same image dimension for the output image as for the input image. Unfortunately, this affects the following nodes in an odd fashion. Therefore, the green part of the scheme was added.

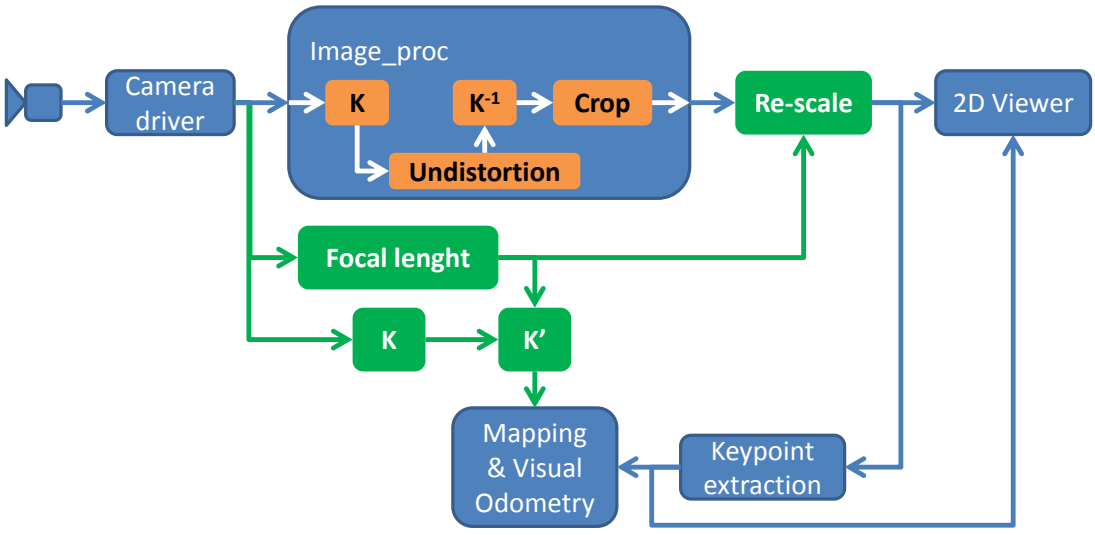


Figure 4.6: Insertion of the undistortion stage in the general image pipeline scheme

[3] `http://docs.opencv.org/2.4/modules/calib3d/doc/camera_calibration_and_3d_reconstruction.html`

[4] `http://wiki.ros.org/camera_calibration?distro=indigo`

[5] `http://wiki.ros.org/image_proc?distro=indigo`

The simplest strategy that has been imagined to counter this problem was to rescale the image using the focal length ratio. Then, a new intrinsic matrix is computed from the old matrix K and this applied ratio. This new matrix K' will be the one applied in the projection model between the world and the drone (the pose estimation computed with the *Solve_PnP* algorithm, see below), then between the drone and the map (to add new keypoints to the pointcloud).

Finally, the result of the corrected image can be observed from the *2D Viewer*, as suggested by Figure 4.6. In Figure 4.7, a comparison is shown between an image coming directly from the camera and an image displayed in the *2D Viewer*.

Figure 4.7: Original picture (left) and undistorted picture (right)

4.2.2 Feature based computer vision

Our objective is to track self motion of the drone based on the camera observation of the environment, which is supposed to be static. To do so, we need to be able to compare successive pictures captured by the camera to match the common parts. It is also needed to match the current scene with previously visited areas. This is known as the *Correspondence problem*. To illustrate this problem, another application is the fusion of two images, which is the first step to build a panorama (see Figure 4.8).

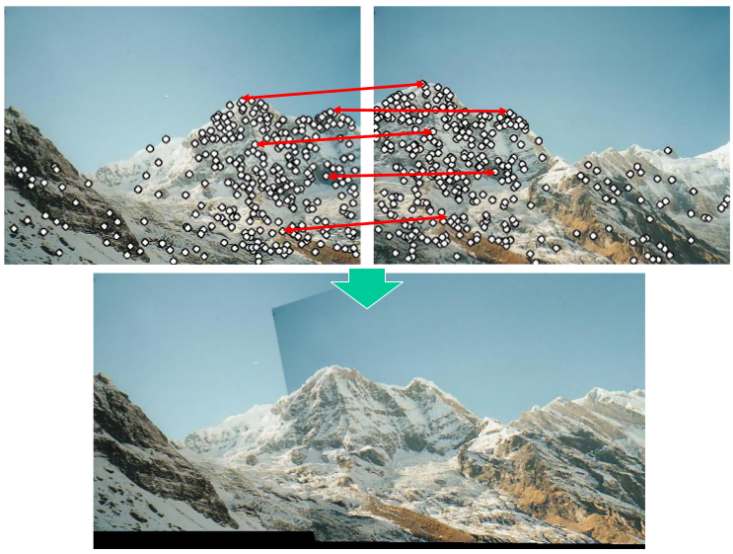


Figure 4.8: Correspondence problem: match two images to create a panorama [159]

In computer vision, there are several approaches to process the information contained in pictures. One approach is to lower the dimensionality of the data by extracting small patches that can be easily described and compared, these are called features. As will be seen in the next section, features are also well suited to be stored in a map.

Keypoints detection and features description

Once a new image is received, the first operation is the detection of keypoints. The next step is to describe all these keypoints with features to allow comparisons. In the following text, *keypoint* and *feature* will be used to name an observed point that can be described and easily recognizable (see Figure 4.9). The locations of keypoints are given according to the coordinate system of Figure 4.3. Let us enumerate what we expect from such features. A reliable detector and descriptor must be insensitive to several factors: viewpoint (rotation, scaling, shearing, perspective), brightness, blurring, and partial occlusion.

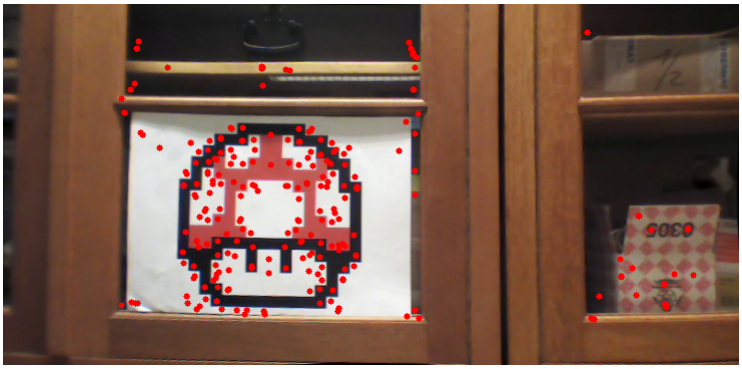


Figure 4.9: Keypoints detection (with SIFT). The keypoints (in red) are superposed on the image in the `2D Viewer` node.

As mentioned in the Section 2.3, the literature proposes several keypoints detectors and descriptors. A detailed comparison of performances of each is out of the scope of this section. Nevertheless, our choice is motivated by some technical criteria. First of all, since the *OpenCV* library was chosen, it is convenient to use detectors and descriptors for which an implementation is available within this library. Fortunately this library is popular and several keypoints detectors are available. Secondly, we need a sufficiently fast algorithm for real-time performances. Finally, the robustness of the detected keypoints is necessary to ensure successful matching as discussed in the next subsection.

The master thesis of the previous years pointed out several state of the art algorithms for keypoint detection and description. Two detectors were finally used last year:

- **Harris corners**: these were chosen to build a map (see Section 4.5). These present the advantage of highlighting natural corners and edges as features. This led to a map which was easier to read for a human eye.
- **SIFT**: these were used to estimate the 2D displacement between successive images and to detect a target (see Section 4.3).

The advantages of the Harris detector are that it is fast and the location of keypoints can be anticipated for a human so that results are easy to interpret. SIFT detector and descriptor is slower. On the other hand it is more robust.

This year, our goal was to use one type of feature to perform both the map building, the pose estimation and the target recognition. This permits to avoid several computations. Moreover, it was decided to keep a code architecture that easily permits to switch from one type of detector and descriptor to another. In this way, several combination of detectors/descriptors could be tested and the performances have been compared (see Table 4.1).

	Robustness	Computation time	Remarks
SIFT/SIFT	+++	- - -	Very robust but slow detection.
SURF/SIFT	++	- -	Still robust enough, and a bit faster than SIFT/SIFT.
FAST/SIFT	- - -	-	FAST is aptly named, but does not provide orientation for the SIFT descriptor computation.
SURF/SURF	-	+	Detection nearly as robust as SIFT, but poor description and less robust to the neon flickering effect. Little computational cost.
ORB/SIFT	- -	++	ORB is a FAST descriptor with orientation but is less robust and the detected keypoints lead to poor descriptions with SIFT.
ORB/ORB	- - -	+++	The Binary descriptor ORB is computationally efficient and is lighter (for memory) than SIFT descriptor but ORB is not sufficiently robust to rotations for our application.

Table 4.1: Comparison of different Detectors/Descriptors

This table clearly shows the compromise that has to be made between robustness and computation cost, and, to a lesser extent, memory consumption.

Finally a combination of SURF detector and SIFT descriptor was chosen, since this combination provided the best compromise between robustness and computational cost. It is the feature detection that is the most resource-intensive. However, it is clear that this choice can be reverted any time if further improvements are to be made.

Some parameters can be chosen in order to adjust the detection algorithms. In particular, for the SURF detector [160], a threshold for Hessian keypoint detector, the number of pyramid octaves and the number of octave layers within each octave were selected manually in order to enhance results in our indoor environment. All these parameters were also made easily accessible for future users to be adapted by grouping them in the main header of our code.

Feature matching

Feature matching uses the descriptions of the features to determine which pairs of keypoints correspond in two images. The authors of the initial SIFT paper [88] recommend to use the Euclidean distance to compare two SIFT descriptors.

To find all these pairs, a significant amount of nearest neighbor matching queries are performed. The use of an exhaustive search (brute-force) leads to a complexity of $\mathcal{O}(n^2)$ where n is the number of keypoints in one picture. This is not affordable for a real time application. One solution is to use an approximate nearest neighbor search. This class of algorithms reduce the complexity of matching. The trade-off is that there is no guarantee that the solution is the exact one and the results are not the same for two identical queries. We use the *FLANN*[6] (Fast Library for Approximate Nearest Neighbors) through OpenCV to perform the matching operation. This library performs fast approximate nearest neighbor searches. It is designed for high dimensional space and selects the best algorithm depending on the dataset.

Figure 4.10 shows an example of matching between two different images of the same scene. It also illustrates how SIFT features are invariant to rotations and thus permits to match two images that were flipped w.r.t. each other.

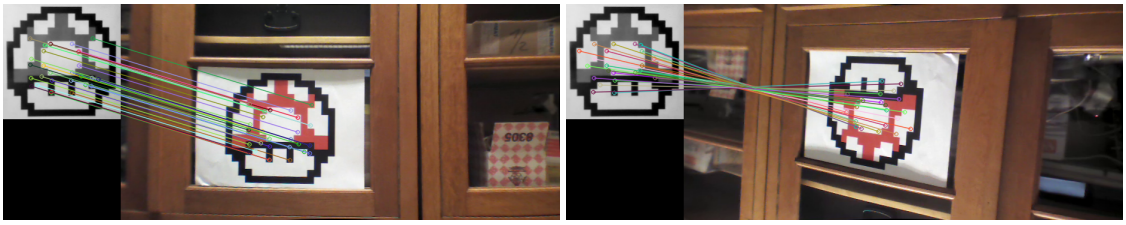


Figure 4.10: Matching example using SIFT descriptors

The use of the matching operation is described in the Section 4.3 for target detection purposes and Subsection 4.4.2 for pose estimation.

Optical flow for keypoints tracking

Optical flow techniques are dense methods (which use all pixels) to estimate the 2D motion of keypoints between two images. This 2D motion is caused by the difference of 3D motion between the camera and the scene. These methods are well suited in specific cases:

- displacements have to be small and can be approximated by a 2D displacements of keypoints in images.
- the tracking has to be performed between successive image with a minimum change in luminosity.
- the occlusion of keypoint (out of range or hidden) leads directly to the lost of tracking.

The principal advantage over a computation of keypoints detection and matching is its time performances.

In our case, the OpenCV implementation of the Lucas–Kanade method [161] was used to compute optical flow and estimate keypoints position when the displacement is sufficiently small. Indeed optical flow methods need the assumption of a constant flow in local neighborhood of each keypoint. The optical flow equations are solved in their neighbourhood for each pixel using the weighted least square solution. These equations need the assumption that the brightness of keypoints remains similar between consecutive images in the video stream, it is called the *brightness constancy constraint*. An important remark is that the flow cannot be estimated in a purely uniform area because the method only uses a local neighborhood.

[6] `http://www.cs.ubc.ca/research/flann/`

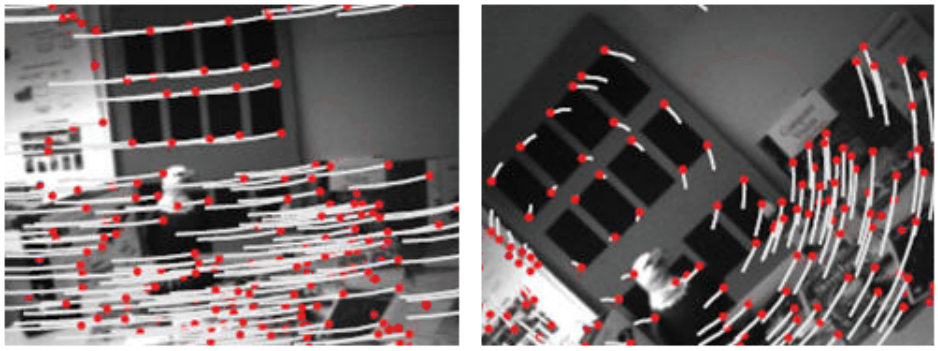


Figure 4.11: Tracking example [162]

Let us formulate the problem to solve. $I(x, y)$ and $J(x, y)$ denote the grayscale value respectively in the first and in the second image. We consider a keypoint $u = (u_x, v_x)$ in image I and we want to recover its coordinates $v = (v_x, v_y)$ in the image J. As u and v are location of a same keypoint we want that $I(u)$ and $J(v)$ are similar. We note $d = (d_x, d_y)$ the displacement such that $v = (u_x + d_x, u_y + d_y)$.

The straightforward least square problem for the pixel u is the following one:

$$\min_d (I(u) - J(u + d))^2$$

To take into account the neighborhood $\mathcal{N}$ of the keypoint, we need to make an assumption on its deformation between two consecutive pictures. As the displacement is considered to be small and if the neighborhood is sufficiently small, it is assumed the neighborhood is subject to an affine transformation. This transformation is generally parametrized as

$$A = \begin{bmatrix} 1 + d_{xx} & d_{xy} \\ d_{yx} & 1 + d_{yy} \end{bmatrix}$$

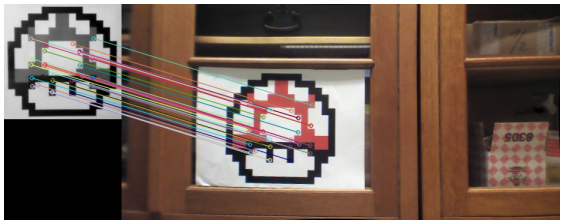
Furthermore, a weighted window W can be used to give more weight to pixels in the neighborhood that are closer to the keypoint coordinate u.

The weighted least square problem to solve is finally:

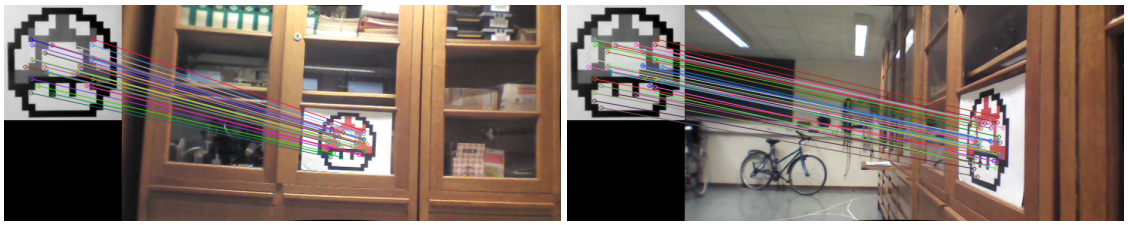
$$\min_{A,d} \sum_{n \in \mathcal{N}} W_n (I(u + n) - J(u + d + An))^2$$

The size of the neighborhood has to be predetermined. The OpenCV implementation uses a pyramidal implementation described in [163]. In short, a pyramidal algorithm means that the process of estimating 2D motion is done on the images downscaled at several resolutions. This ensures the algorithm to recover large and small motions. This is also called a coarse-to-fine method.

Coupled with the matching algorithms described earlier, when the perspective distortions are more important, the tracking of an object across several views of a video stream can be considered using keypoints tracking as illustrated in Figure 4.12. This cannot be achieved with only keypoints detection and matching features description, in a picture by picture way, since the description of feature is not sufficiently robust under such important distortions.



(a) initial view



(b) (c)

Figure 4.12: Keypoints tracking with important perspective distortions: the initial view (a) is processed with keypoints detection, keypoints tracking is used across several pictures until views (b) or (c)

4.2.3 Summary

This section sums up what is needed to process a new image of the video stream.

The first step is the image correction according to the calibration of the camera to avoid non-squared pixels and image distortion due to camera lens.

The second step is the computation of keypoints locations. The first method to be used is the tracking of keypoints detected in the previous image of the video stream. This tracking is performed with Lucas-Kanade optical flow method. If not enough keypoints are recovered (either the tracking fails or they are not in the vision field anymore), a keypoint detector is used. The SURF detector was used for this purpose.

Finally, the SIFT descriptor was used to describe new keypoints. Further steps do not use the pixels of an image anymore but only the descriptors computed at this step.

Figure 4.13 illustrates this process. The details of the Target Detection operation is described in the next section.

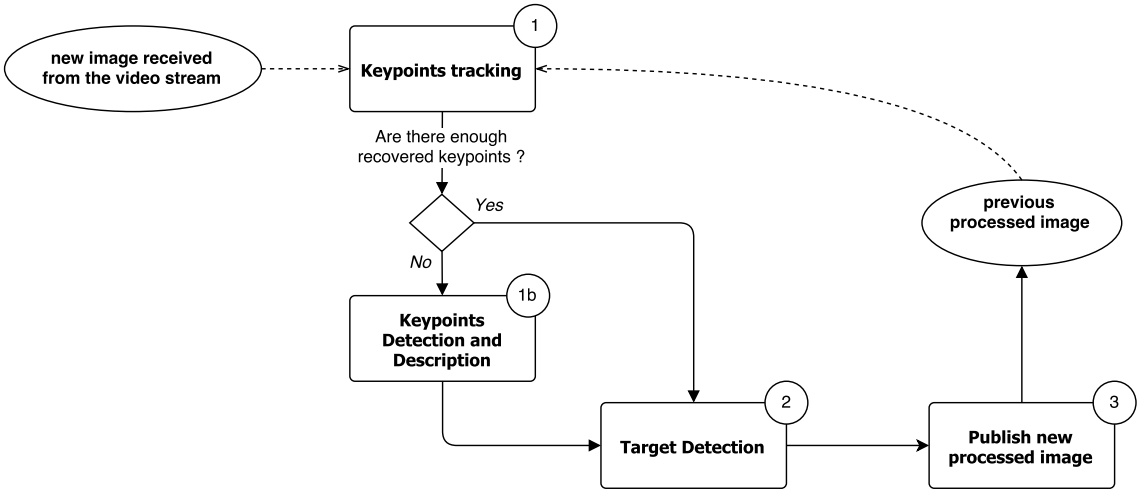


Figure 4.13: Keypoints extraction procedure

4.3 Target recognition and position estimation

As a reminder, one of the first objectives of this year, before entering the core of the matter, was to re-implement the past years benchmark mission with our own techniques, in the new framework (ROS). This was done collaboratively with the other group of master students.

This mission consisted in letting a drone explore a given area, searching for a particular object (here, an image on the ground). Then, it had to land beside this image and call a second drone, by communicating its current position. The second drone would then take off and join directly the first drone in a straight line, before landing beside it.

This mission thus required the ability to recognize a pre-defined pattern, or *target*, in flight. For the time being, the target is supposed to be flat (this is satisfied if the height of flight is sufficiently larger than the height of the target vehicle) and to belong to the ground plane. However, two improvements were done according to what was possible last year:

1. The target does not need to be static anymore. Indeed, the drone can follow it.

2. The drone does not simply land beside the target, but actively estimates its pose.

Unfortunately, a moving target does not respect the assumption of a static environment anymore. This has implications on both the mapping process and the related egomotion of the quadcopter. Indeed, the drone may not be considered as a part of the environment, and it must absolutely be avoided to map the moving target. Because, if the moving target is mapped along with the environment, the map will contain a lot of repetitions of the target at places where it is not anymore, leading to a deterioration of the map.

This confirms that the appearance of the moving target should be known beforehand. It allows to avoid adding some features which belong to the tracked target. Otherwise, there would be no guarantee that the quadcopter pose estimation will not be relative to the moving vehicle rather than relative to the static environment.

4.3.1 Removing the target from the mapping and egomotion processes

To estimate the presence of the target in an image, descriptors matching is used (see Subsection 4.2.2). The features detection and description of the target image (the model) is performed before the processing of any camera image. This set of features is compared with each set of camera image features using the FLANN library. Over a certain percentage of correspondences, the target is declared to be "detected".

As the matching is not perfect and the detection can find some features that are absent in the model, it is necessary to remove some extra keypoints to ensure no moving features will be added to the map. This is performed using a rigid perspective transformation [164] to estimate the piece of image within which any keypoints must be discarded (see Figure 4.14).

This rigid perspective transformation is parametrized as $H \in \mathbb{R}^3$. Let us note (x_i, y_i) the location of the i-th keypoints in the camera image and (x'_i, y'_i) the corresponding location on the

Figure 4.14: Target detection and removing the target keypoints within the green box from the image pipeline

model image. Using homogeneous coordinates, the perfect transformation respects:

$$s_i' \begin{bmatrix} x_i' \\ y_i' \\ 1 \end{bmatrix} = H \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix}$$

To find H, the back-projection error is minimized using the Levenberg-Marquardt method.

$$\min_H \sum_i \left(x_i' - \frac{h_{11}x_i + h_{12}y_i + h_{13}}{h_{31}x_i + h_{32}y_i + h_{33}} \right)^2 + \left(y_i' - \frac{h_{21}x_i + h_{22}y_i + h_{23}}{h_{31}x_i + h_{32}y_i + h_{33}} \right)^2$$

The Levenberg-Marquardt minimization method is also known as the damped least square [165]. A damping factor is re-evaluated at each iteration using some heuristic, so that the method is closer to the gradient descent or to the Gauss-Newton method. The damping factor is a kind of regularization factor. This is useful for ill-conditioned problems, problems for which a small error in the input data implies larger error in the output.

4.3.2 Target pose estimation

The pose of the target is estimated using the projection described in Subsection 4.5.3 on the target homography (the green box of Figure 4.14, which lies effectively in the physical plane of the target picture.

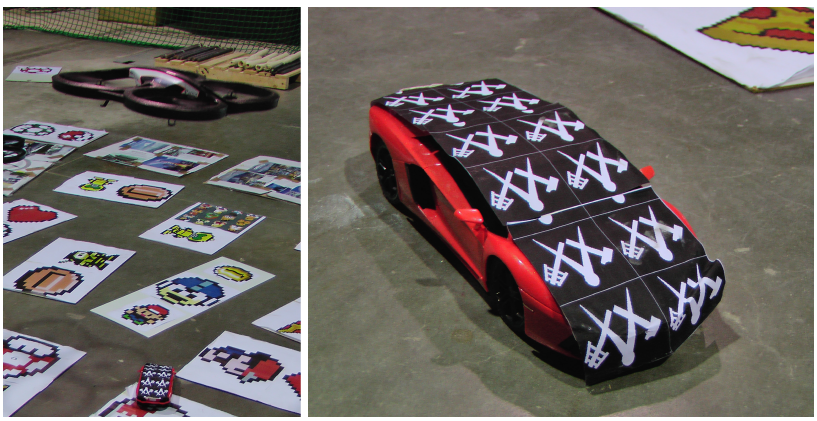


Figure 4.15: Configuration for a wheeled vehicle tracking which has to be known in advance and distinguishable from the static environment [153]

Now that all the ingredients are assembled, it is possible to imagine a benchmark mission to test the effectiveness of our moving target pose estimation. This was one of the achievements of the second group, and the video is available on the internet[7]. In this video, a small wheeled vehicle was used as moving target (see Figure 4.15). It is sufficiently flat to respect our prior assumptions, but it presented poor visual texture. So, some black and white pictures have been pasted on it. Then, our common version of last years benchmark mission has been slightly modified to let the first drone fly along the target while it is moving, and call the second drone only when its battery is lacking energy. Figure 4.16 shows a zoom on the parts of the code on which the two teams of this year collaborated to achieve the improved version of last years mission.

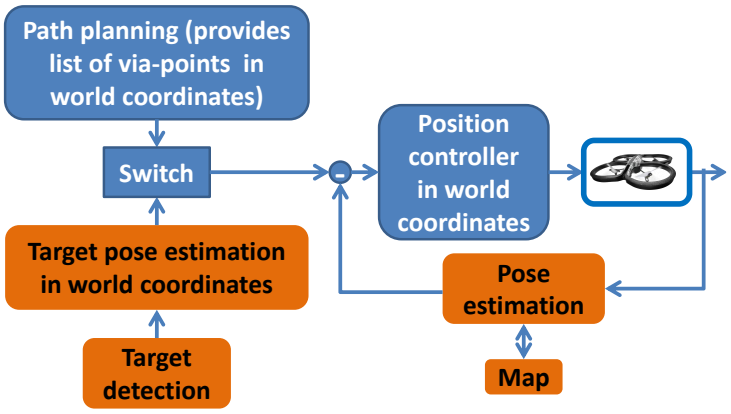


Figure 4.16: Orange: parts achieved by our team; blue: parts achieved by the second team

It is worthwhile noting that the implementation of target following that has been put in place here does not permit the target to move too fast. Indeed, more advanced solutions have to be envisaged to obtain a robust tracking when the target is moving fast. The other group chose to use our target detection as is. To enhance the target tracking, algorithms like PnP (explained in Subsection 4.4.2) coupled with a Kalman filter taking into account a relative motion model can be envisaged. Lucas-Kanade tracking can also be improved with the use of the IMU information [162]. These techniques were not part of the objectives of our group.

[7] see `https://www.youtube.com/watch?v=IaHg9J8_VjA`

4.4 Visual odometry

This sections covers the concepts of *pose* and *visual odometry*. Therefore, for a good understanding, the reader is expected to master the concepts exposed in the introduction of Chapter 2 and Section 2.2.

As the quadcopter has to evolve in a 3D environment (by the nature of its flight), the full 6 DOF Pose has to be estimated. Two methods were envisaged:

1. Evaluate the motion between two consecutive pictures.

2. Take into account one picture and a 3D model of the scene (e.g. a map, see Section 4.5).

4.4.1 Method 1: Epipolar geometry and the Essential matrix

Before entering into the details of the implementation, epipolar geometry and constraints inherent to the camera observation model will be presented. This theory is common with the one necessary to understand 3D reconstruction with a monocular camera or a stereo camera. The main difference between a monocular camera compared to a stereo camera is that the displacement between two views has also to be estimated. This adds more uncertainty (which can be attenuated with additional information from other sensors).

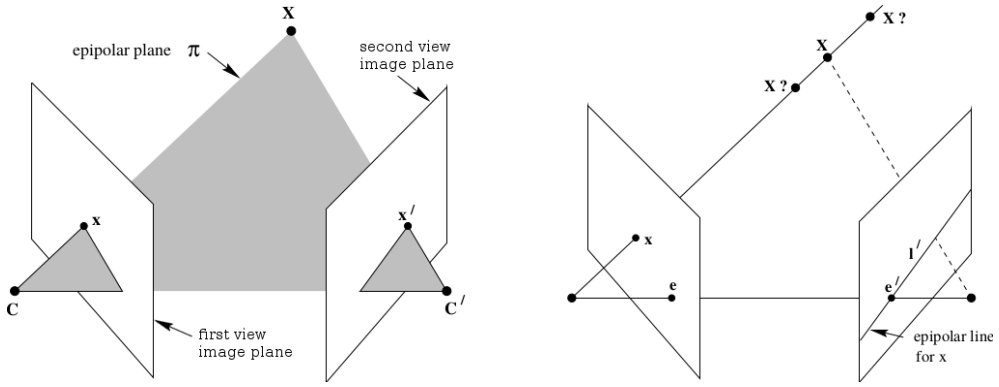


Figure 4.17: Epipolar constraint (from [166] on p.240)

Suppose we dispose over two camera views, the first one at center C and the second at center C'. This situation is illustrated at the Figure 4.17. The two image planes can be parallel but C and C' must not coincide. A 3D point noted X is observed, x is its projection on the first view, x' on the second. The line linking C and C' is called the baseline and the plane π containing the baseline and the point X is called an epipolar plane. The epipolar geometry describes how the projection x on the first view constraints the projection on the second view. x' lies on l', the projection of the line joining x and X called the epipolar line corresponding to x. There is a mapping (in the mathematical sense) from $x \mapsto l'$. This relation is described by the fundamental matrix[8] F which satisfies the correspondence condition $x'^T F x = 0$. When one calibrated camera is used with calibration matrix K and normalized coordinates, the matrix which satisfies the correspondence condition is called the Essential matrix $E = K^T F K$.

[8]The complete geometric and algebraic derivations of the fundamental matrix are described in [166] on p.241

This matrix E also satisfies the relation $E = t \times R$ where t is the translation vector and R the rotation matrix between the two camera views. The projection matrix $P = [R|t]$ can be retrieved using a singular values decomposition (SVD) of E. However, there remains a scale ambiguity. Indeed, E has only 5 degrees of freedom while R and t have both 3 degrees of freedom. Consequently, all methods using only a monocular camera are always up to one scale factor. As will be seen later in the Section 4.5, to ensure to have a reference distance measurement to palliate to that scale factor, another sensor must be used to provide an absolute measure of distance.

The essential matrix can be computed using for example the *8-Point algorithm*. This algorithm needs at least eight points correspondence between two views, but several alternatives exists. This topic is not detailed in this text, since the second method has finally been selected to counter the scale ambiguity.

4.4.2 Method 2: Perspective-n-Point

Perspective-n-Point (PnP) is a class of algorithms which can solve the problem of estimating the 6 DOF pose of a calibrated camera. Given n matching points correspondences between 3D points and their 2D projections, these algorithms estimate the position and the orientation of the camera. In other words, the objective is to recover the matrix $P = [R|t]$.

Suppose now that a 3D model of the environment of the drone is known, in other words a map containing 3D points that can be matched with the 2D points in an image. Using this procedure, there is no more issue of having an up to a scale factor translation estimation when the essential matrix estimation is considered between two successive pictures. The ambiguity is solved because the scale is contained within the 3D model. Nevertheless, this scaling factor has to be determined at least once. In our case, it was decided to use the ultrasonic sensor during the take-off operation, when the mapping is initialised, to inform about the distance from the camera to the ground, and thus provide the scale once and for all. One could also imagine to re-calibrate during the flight This would imply to rescale all or part off the built map using for example Bundle Adjustment (see Subsection 4.5.4).

One particular case is when $n = 3$, which is then called the P3P problem. This can be put into equations using the cosine law (generalized Pythagoras law) for the three triangles illustrated on Figure 4.18.

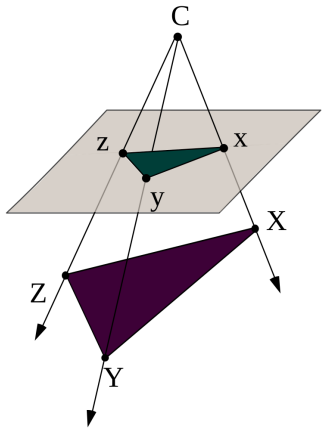


Figure 4.18: P3P Problem:
Let C be the center of projection of the calibrated camera and 4 points correspondences such that W, X, Y, Z are the 3D points and w, x, y, z are their respective projections on the camera image plane.
Three points (X, Y, Z) are used to solve the P3P system of equations to find four sets of distances $\{||CX||, ||CY||, ||CZ||\}$.
The fourth point W is used to select the right solution amongst the four sets.
Finally, the set of distances is converted according to the pose convention (position and orientation in the 3D points frame).

The full derivation, though, will not be presented in this text. The resulting equations lead to a set of two quadratic equations which can admit up to four solutions which have physical meaning (positive distances, etc.). A fourth point is used to select the correct solution. The detailed explanation is available in [167].

4.4.3 Enhancing PnP with RANSAC

If more than four points correspondences are available but not all correspondences are reliable, it can be useful to reject unreliable pairs. The Figure 4.19 illustrates the effect of an unreliable point in the estimation of a 2D line.

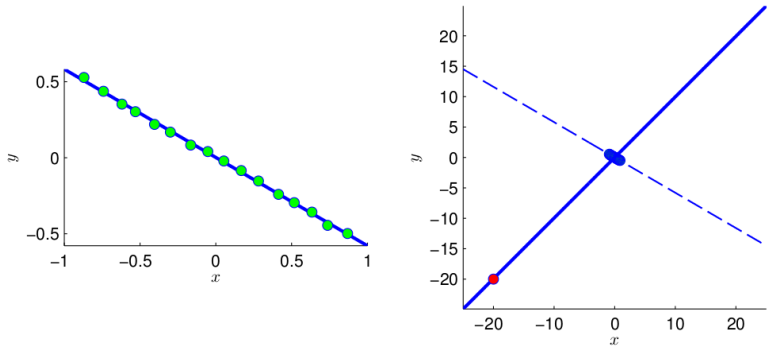


Figure 4.19: Simple example of the effect of an outlier (in red) on a line interpolation with the least square method and without RANSAC (from [168])

The RANSAC (Random Sample And Consensus) algorithm can be used to select reliable data. The rejected data are said to be outliers. Suppose there exists a model and a set of parameters which can generate the observations in absence of noise, and that the maximum amplitude of the noise is known. An outlier is an observation for which the error with the model is greater than the tolerated perturbation. The breakdown point is the percentage of outliers in the contaminated data set.

RANSAC is an iterative algorithm based on the hypothesize-and-test framework:

- **hypothesize**: A random subset of correspondences is used to estimate a transformation.
- **test**: All points are transformed and the reprojection errors are measured. Correspondences with a reprojection error below some predefined threshold are considered inliers to the candidate solution.

This procedure is repeated a large number of times and the solution having the largest number of inliers is chosen. Using RANSAC does not guarantee an optimal choice of correspondences [169].

The *standard method* to solve geometric vision problems is to use the solution obtained by RANSAC as a starting guess for a local optimization minimizing the norm of the reprojection errors [169]. This is has been done with *EPnP*.

4.4.4 EPnP: an alternative to P3P

Once the set of inliers is determined using RANSAC on P3P, another Perspective-n-Point algorithm can be used on this set, to provide a better pose estimation. For example, the 'EPnP' algorithm, which computes the camera pose with more than four points. Indeed, since it uses more points, it is generally more accurate than P3P. This method gives a non-linear solution with a linear running time [170]. Finally, even if EPnP can also be used along with RANSAC, it has been chosen not to do so, since it has been shown [171] that this combination is less stable than with P3P.

4.4.5 Practical implementation of the PnP - RANSAC - EPnP chain

Practically, the *OpenCV* implementation '`solvePnPRansac`' has been used for the P3P and RANSAC combination [172], while for EPnP, the '`solvePnP`' has been used with a parameter specifying the desired type of PnP algorithm. The solution obtained by P3P RANSAC is used as a starting guess and only inliers are given to EPnP.

The whole process of pose estimation using a 3D model of the environment is implemented into the mapping node and is referred to as "PnP RANSAC" (see next section). Its result will be called "visual pose estimation" from now on.

4.5 Mapping

As explained above, the whole mapping process has the main objective to help improve the pose estimation of the drone, through the SLAM (Simultaneous Localization And Mapping) technique. However, once it is created, it could of course be used for other purposes, such as path-planning, dense reconstruction, simply 3D (collaborative) mapping, etc. (see Section 2.4 for more). These features require a higher-level interpretation of the created map, and are not part of our short-term objectives. Nevertheless, the whole mapping node will be subject to design choices that must take these future potential developments into account.

4.5.1 Design choices

The map in itself is a collection of keypoints located in the reference world frame. The elements of this data structure are objects containing a triplet of coordinates and a description vector of the feature. Other properties can be added like the number of views within which the keypoint was detected. The structure of one of our 3D points is shown in Figure 4.20. The fact of mapping both the keypoint (XYZ) and its descriptor is referred to as feature-based mapping in literature. Alternatives include for example edge-based or voxel-based[9] mapping, or even dense mapping, like in LSD-SLAM. The two first mapping methods were rejected, because the resulting maps do not permit to localize as accurately as with a feature-based map. They correspond to a higher-level interpretation of the latter. Thus, such maps can still be obtained any time from our map.

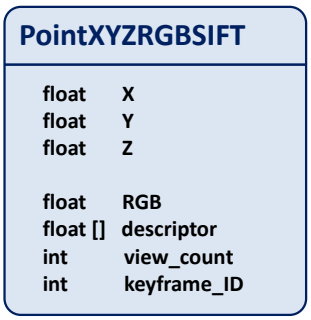


Figure 4.20: Structure of the objects that are placed in the map: PointXYZRGBSIFT

It was chosen to use the `Point Cloud Library` (PCL), which provides data structures for 3D point clouds. As explained in Subsection 3.2.3, this open-source library is well maintained, adapted for ROS, and provides a lot of high-level functions to manage and interpret the map[10] :

- Filters,
- Voxel maps, octrees and kdtrees (see Section 2.4),
- Dense reconstruction,
- Surface identification,
- Map fusion (useful for collaborative mapping with drones in future developments),
- Visualizer (used for our `3D Viewer` node),

[9] As explained in Section 2.4, with a voxel approach, the world is represented as a 3D grid. Each element of the grid is called a voxel.

[10] `http://www.pointclouds.org/documentation/`

These features can be very useful for future developments.

In addition to the choice of 3D points structure and map management library, it was chosen to base our implementation of the map as an organization structured into keyframes. A keyframe is a data structure containing several keypoints observed from the same view (extracted from the same image of the video stream). Keyframes can partially overlap (some keypoints are contained in several keyframes). This structure presents several advantages.

Firstly, it allows to build the map incrementally with a minimum number of incoherences. Indeed, each time keypoints are added into the map, they are added with some re-projection error (see Subsection 4.5.3 and Appendix C). Thus, errors can accumulate over time, creating some drift of the map representation of the world w.r.t. the real world. This has no influence on the absoluteness of visual odometry, which remains absolute w.r.t. the map (which is, in fact, required by definition of SLAM). But at a lower level, this can render the matching between current view and map less robust, leading to more current "*tracking lost*" situations[11] and a less precise visual odometry. Thus, adding blocks of coherent keypoints rather than adding keypoint per keypoint ensures that the map is constituted of piecewise coherent ensembles of keypoints. For visual odometry, it appears much more robust to refer to one reference ensemble of coherent keypoints (i.e. a keyframe) at a time, than referring to a big cloud of points containing large relative errors. This finally leads to the technique known as "Keyframe-Based SLAM" in literature.

Secondly, this is well suited for the kind of queries we have to do on the map. Indeed the queries are of the type "what is visible from the current pose of the quadcopter?", the answer is likely to be either nothing known, or several keypoints seen together previously. The search amongst previously added keyframes is also more efficient, since distant keyframes can easily be rejected.

Thirdly, it is easier to keep trace of the used keypoints at the previous iteration of the pose estimation.

Fourth, the keypoints are not added continuously in the map, but only if necessary. This reduces the overall overhead of this portion of code, and also limits the growth of the map. The latter should not be neglected, since it must be kept in mind that the map does not stop growing until the end of the flight, possibly leading to memory issues or prohibitive access times to the map.

Finally, it will be useful for future improvements like *Bundle Adjustment* (see Subsection 4.5.4).

The next section will cover two tasks related to SLAM: the building of a map based on visual informations collected by the computer vision node, and the pose estimation of the quadcopter based on the last visual informations received (see Section 4.4). These two tasks are performed into one single ROS node: the `Mapping` node. Another choice would be to isolate those two jobs into two separate nodes. But this would seriously affect the efficiency, since these tasks need to be able to perform the same type of queries on the map.

[11] see Subsection 4.5.2

4.5.2 The mapping strategy

Let us now describe one iteration of our mapping algorithm. The Figure 4.21 sums up all the operations. The mapping node subscribes to the channel fed by the computer vision node. Each message contains all detected keypoints in one image taken by the camera. The iteration of the mapping begins with the reception of one of these messages.

The first operation is to match the last detected keypoints with those stored in the map. To avoid to search in the whole map at each iteration, and thus avoid to query a big data set (the size of the map, i.e. the number of keypoints in the list, grows rapidly), the assumption of small displacement is first considered. If it is not verified, a search is performed on a bigger subset of the map.

Practically, the current detected keypoints are compared with those contained in the keyframe which matched at the previous iteration (called reference keyframe). At this stage, the search does not include other keyframes. It results that either the keyframe contains enough correspondence, or the pose of the drone in the map is lost. This situation corresponds to the **tracking lost** case. In this case, other sensors (on-board IMU, gyroscopes and optical flow) are used, because all visual references needed to compute the visual odometry have been lost. In case of success, PnP RANSAC is performed. If enough keypoints among the correspondences are inliers, PnP returns a sufficiently reliable pose estimation. The visual pose is published. Otherwise the pose is again estimated only with the other sensors.

When the set of received keypoints has a too small intersection with the reference keyframe, this means either that it is necessary to switch to another keyframe previously stored, or a new keyframe has to be built. This is determined by performing a search amongst keyframes visible from the current estimated pose of the quadcopter. To determine what the drone should see from its position, the equations of the four planes delimiting its field of view must be found. Then these planes must be transformed from the coordinate system attached to the drone to the coordinate system attached to the world. Finally, a simple test is performed to determine if each of the stored keyframes is at the visible side of the plane or not. The mathematics are quite irksome and thus will not be exposed in this text.

Let us note that this searching operation is also performed when the pose was estimated with other sensors. If a new keyframe is built, it becomes the reference keyframe for the next iteration.

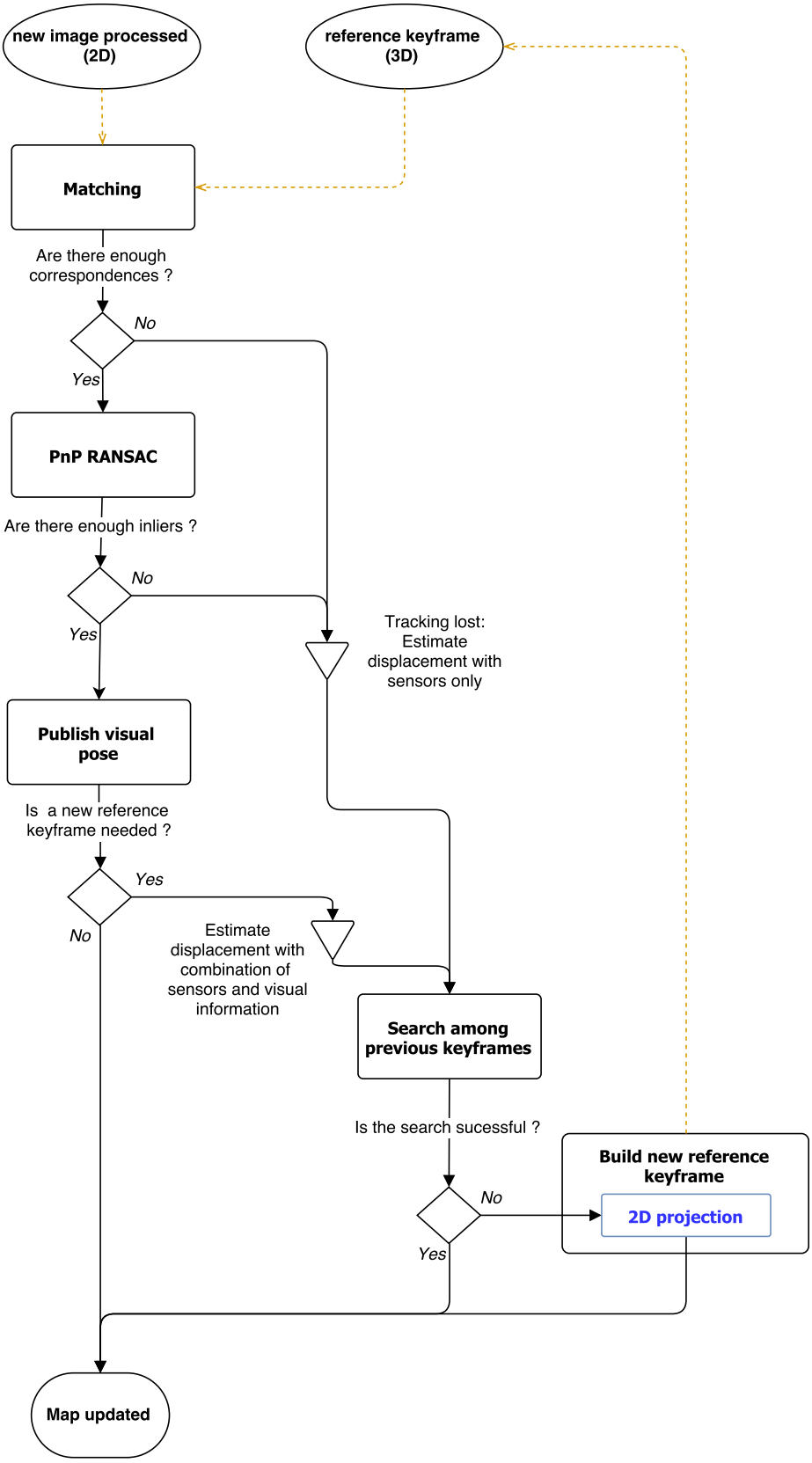


Figure 4.21: Visual pose estimation and mapping decision tree (in blue: the provisional perspective projection (2D simplification), see Figure 4.25.)

4.5.3 Workspace transformation

Before achieving a complete 3D mapping, our development schedule included an intermediary objective which was to provide an implementation ready for 3D but with temporarily simplifying assumptions already used previous years.

These assumptions are described as following:

1. The camera is pointing downwards.
2. The ground is flat.
3. All detected keypoints belong to the ground plane.

Indeed, many modules of code are necessary to implement 3D mapping. Thus, a good way to test all modules individually is to make these assumptions first, to ensure the algorithm is able to correctly map in 2D. The module that will be tested here corresponds to the "workspace transformation". Its function is to perform the "2D projection" from the image plane to the flat ground plane. This module should thus disappear and be replaced by a triangulation module, without any other modification to the other modules. The workspace transformation involves less steps than triangulation and is thus easier to debug. The generalization to triangulation was not yet fully implemented for reasons explained in Subsection 4.5.4.

To derive the projection of the keypoints detected on the image plane to the ground plane, several coordinate systems will be used: the normalized image plane coordinate system (defined on Figure 4.3 of the image that is calibrated and undistorted), the camera coordinate system, the quadcopter coordinates system and finally the world coordinate system. These reference frames are represented on Figure 4.22.

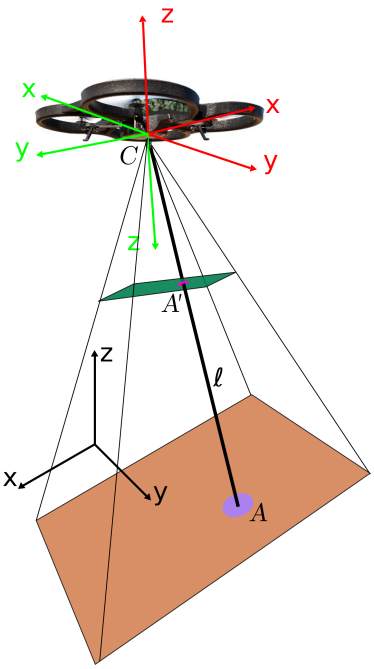


Figure 4.22: Summary of all coordinate systems:

- in black: world frame [m]
  (inertial reference frame)
- in red: drone frame [m]
  origin: center of mass of the drone
  x forward, y leftward, z upward
- in green: camera frame [m]
  origin: center of mass of the camera
  z pointing to the scene

see Figure 4.3 for the undistorted image frame

planes:
dark green: normalized ($z = 1$ in camera coordinates)
image plane (calibrated and undistorted)
light salmon: ground plane ($z = 0$ in world coordinates)

Outline of the method

To determine the location in world coordinates of a detected keypoint A with coordinates (u, v) in the image, an intersection of loci is used. First, the equation of the ground plane is tranformed to the camera coordinate system. Then, the intersection between this plane and the ray joining the center of the camera C and the keypoint on the image plane is computed. Finally, the intersection point is expressed in the world coordinate system.

Mathematical formulation and exact solution to the geometric problem

In the camera coordinates, each ray converges to the camera center $C = (0, 0, 0)$. At a particular point A observed at (u, v) on the undistorted image, we have from equation (4.1)[12]:

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} \sim \begin{bmatrix} f_x X + c_x Z \\ f_y Y + c_y Z \\ Z \end{bmatrix} = \underbrace{\begin{bmatrix} f_x & 0 & c_x \\ & f_y & c_y \\ & & 1 \end{bmatrix}}_{K} \underbrace{\begin{bmatrix} 1 & & & 0 \\ & 1 & & 0 \\ & & 1 & 0 \end{bmatrix}}_{P} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

$Z = 1$ is arbitrarily chosen, the distance of the normalized image plane to C, so the point A' in the camera coordinates is located at

$$A'_c = \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} (u - c_x)/f_x \\ (v - c_y)/f_y \\ 1 \end{bmatrix}$$

The direction vector ℓ of the ray joining C and A in the camera coordinates is

$$\ell_c = \overrightarrow{CA'} = \begin{bmatrix} (u - c_x)/f_x \\ (v - c_y)/f_y \\ 1 \end{bmatrix}$$

A plane is fully described by its normal and its distance to the origin. For the ground plane γ, we have:

$$\begin{aligned} n_w &= \begin{bmatrix} 0, 0, 1 \end{bmatrix}^T \\ d_w &= 0 \\ \gamma &\equiv n_w \cdot \begin{bmatrix} X, Y, Z \end{bmatrix}^T + d_w = 0 \end{aligned}$$

expressed in the world coordinates. To express it in the drone coordinates, we need the transformation matrix between the two frames.

The quadcopter location in world coordinates is supposed to be known and is described as a 6DOF pose. This pose will be used to compute the transformation matrix (rotation and translation) between the world frame and the drone frame $[R_{w\to d}, t_{w\to d}]$.

The pose of the quadcopter is composed of a position vector $t_{w\to d} = \begin{bmatrix} x, y, h \end{bmatrix}^T$ (in world coordinates) and an orientation vector $\begin{bmatrix} \theta, \varphi, \psi \end{bmatrix}^T$, called roll, pitch and yaw angles. The corresponding rotation matrix is the following one:

[12] We suppose here that the skew factor $s = 0$, which is the case for the cameras we use

$$\begin{bmatrix}\theta\\ \varphi\\ \psi\end{bmatrix} \mapsto R_{w\to d} = \begin{bmatrix}1 & 0 & 0\\ 0 & \cos(\theta) & -\sin(\theta)\\ 0 & \sin(\theta) & \cos(\theta)\end{bmatrix} \begin{bmatrix}\cos(\varphi) & 0 & \sin(\varphi)\\ 0 & 1 & 0\\ -\sin(\varphi) & 0 & \cos(\varphi)\end{bmatrix} \begin{bmatrix}\cos(\psi) & -\sin(\psi) & 0\\ \sin(\psi) & \cos(\psi) & 0\\ 0 & 0 & 1\end{bmatrix}$$

The transformation matrix $[R_{d\to c}, t_{d\to c}]$ between the drone and the camera does not vary over time. Finally we know the ground to camera transformation:

$$[R_{w\to c}, t_{w\to c}] = [R_{d\to c}R_{w\to d}, t_{w\to d} + R_{w\to d}t_{d\to c}]$$

for the following, we assume $t_{d\to c} = 0$, so the distance from camera to ground is equal to h, the altitude of the quadcopter in the world frame.

The normal of the ground plane and the distance to the origin in the camera coordinates is

$$\begin{aligned} n_c &= R_{w\to c}n_w \\ d_c &= h \end{aligned}$$

The intersection of the ground plane and the ray is finally computed as

$$\begin{aligned} t &= -h/(n_c \cdot \ell_c) \\ A_c &= t \times \ell_c \\ A_w &= R_{c\to w}A_c + t_{c\to w} \end{aligned}$$

where A_c are the projection coordinates in the camera frame coordinates and A_w its coordinates in the world frame.

Validation

Figure 4.23 gives an insightful illustration of the job performed by the workspace transformation. The flat images on the ground, seen from a certain perspective, are mapped with a correct aspect ratio onto the "ground" of the map. This is visible since it is looked at from above in the `3D map Viewer` (see the orientation of the axis in the figure, the axis origin corresponds to the position of the drone in world coordinates).

An experimental validation of this perspective projection is presented in Appendix C.

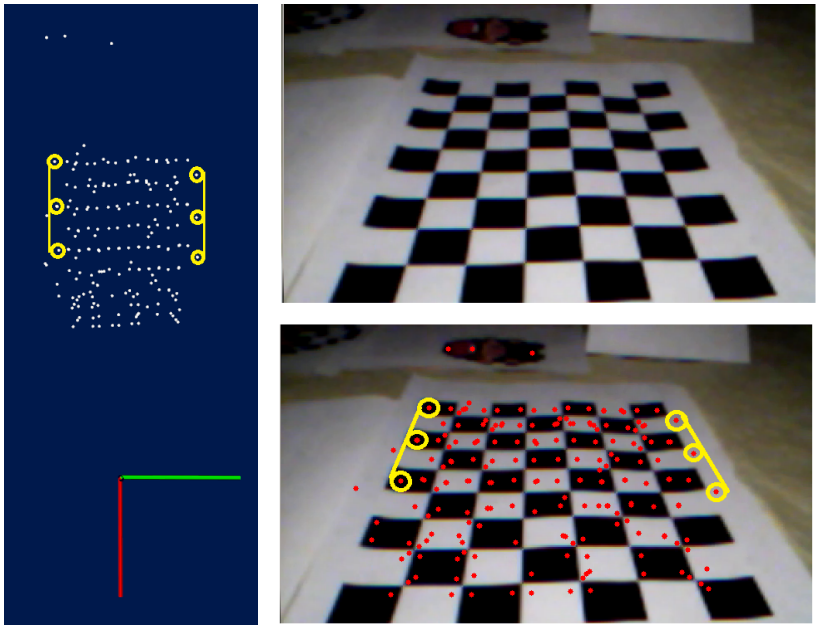


Figure 4.23: Illustration of the perspective transformation for extreme angles: the rectangular pattern (a checkerboard), which appears like a trapeze on the image, is printed correctly in the map (the red dots are the detected keypoints, the yellow lines were added to highlight the (non-)parallelism of the edges of the pattern).

4.5.4 Towards 3D Reconstruction

Let us take a look at the 3D reconstruction problem using a monocular camera.

Given the correspondence points in two images and the camera projection matrices (i.e. poses of the calibrated camera) for both views, the problem of finding the 3D points world coordinates is called **triangulation**. This can appear as a simple problem because computing the intersection of two lines is an easy task (see Figure 4.24). But it becomes less trivial if the points coordinates in the image plane or the pose estimation are not error free, because then, the rays do not necessarily cross anymore. Several methods exist to handle this, like the *midpoint method*, which selects the midpoint of the perpendicular that intersects both rays. However, this method is known in the literature for giving bad results. Other methods, called *optimal correction*, use constrained minimization. The re-projection error is minimized so that the projection satisfies the correspondence condition (with the Essential matrix described in Section 4.4).

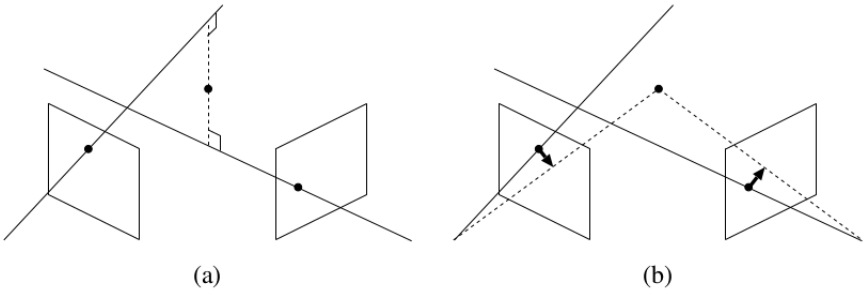


Figure 4.24: Triangulation: (a) midpoint method, (b) optimal correction [173]

If the camera projection matrices are unknown (but the camera is still calibrated), then the problem suffers from the same scale ambiguity as the computation of the Essential matrix from correspondence points. It means that with only two camera views, a metric reconstruction is not possible and the reconstruction is up to an unknown scaling factor (see Section 4.4).

This ambiguity can be cleared up if 3D coordinates of at least five observed points, called **ground control points**, are known to compute an homography and estimate the scale.

With our code settings, there are two situations where the pose cannot be estimated from the camera observations: at initialisation, because the map is still empty, and when the tracking is lost because no mapped points are matched (or visible).

For the initialisation, one technical solution is to put a known object at known coordinates visible at the start of the algorithm. The object is described by its keypoints and their metric 3D coordinates. This contradicts the assumption of a completely unknown environment but it is affordable in many situations. For example, the quadcopter can take off from a known base platform with a distinguishable symbol.

However, this solution does not tackle the *tracking lost* problem (see Subsection 4.5.2). Since the quadcopter embeds other sensors, the pose can be estimated using only proprioceptive sensors. A design choice must also be made: is the mapping stopped until some known mapped points are recovered? Or should the (non visual) pose estimation be used to perform triangulation instead? These choices deserve further reflexion and tests.

As illustrated on Figure 4.25, the main missing piece of code is the triangulation module. The green parts of this scheme should be compared to the blue part of Figure 4.21. This module implies to store several `processed_image` messages from the computer vision node in order to match and triangulate the new points to add to the map. The current conditions to add a new keyframe certainly need some adaptations as well, as the selection of the retained `processed_image` should be used. However, data structures will need no major changes and the whole visual pose estimation process is already 3D, as well as the map structure and its 3D viewer.

Another reason why 3D SLAM has not been fully activated yet is because we suspect the need for *Bundle Adjustment* (see [47]) to achieve good results (i.e. acceptable robustness to depth-estimation errors). As a reminder, Bundle Adjustment is the minimization problem of the re-projection error given visual observations and a camera viewing model. It refines both 3D points coordinates and poses of the camera. As mentioned before, observations are prone to errors and Bundle Adjustment is a way to take more than two views and take into account the uncertainty on poses. This refinement process needs to be performed in parallel with the mapping process.

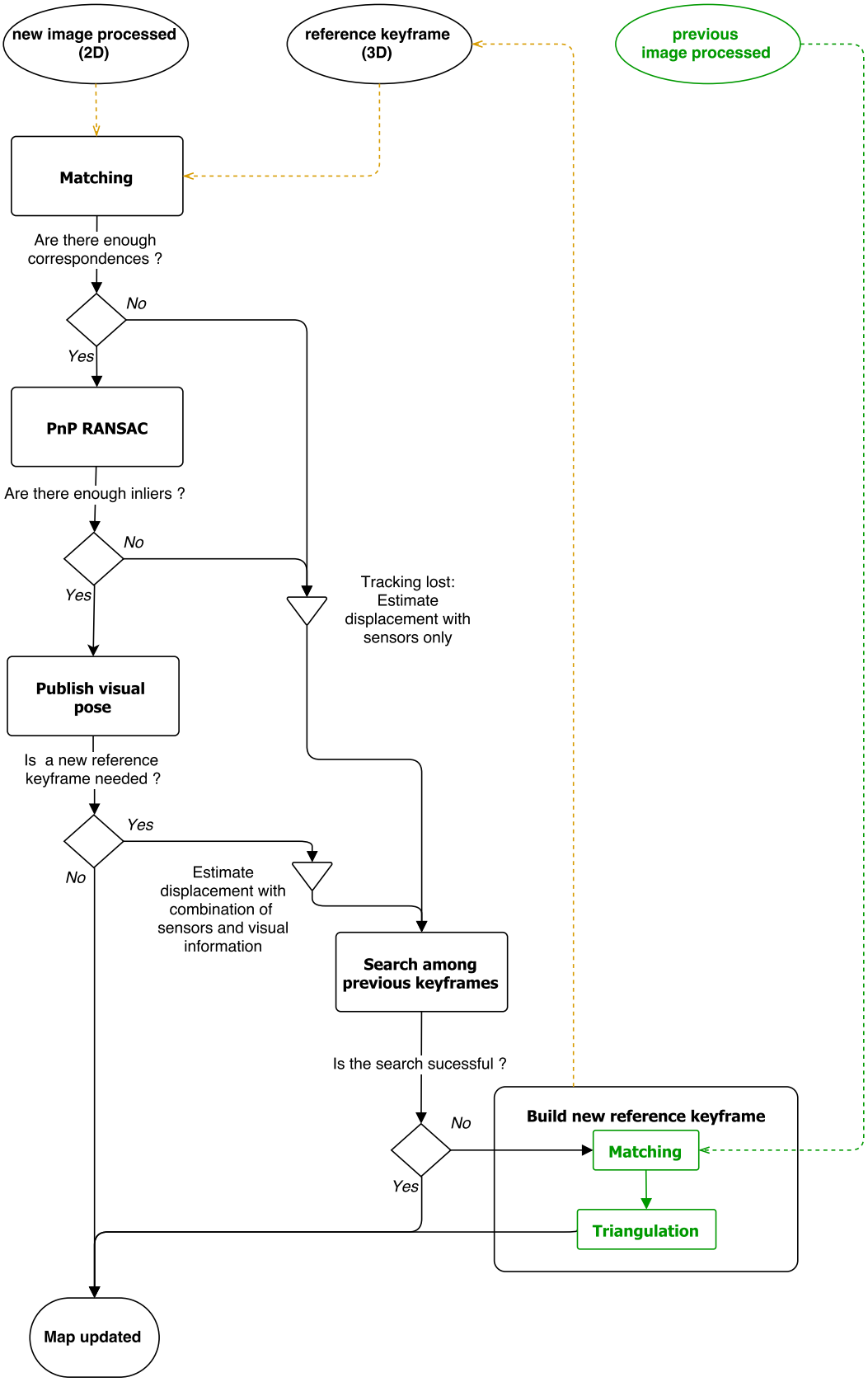


Figure 4.25: Visual pose estimation and mapping decision tree with missing blocks for 3D mapping (in green)

4.6 Pose estimation

In this section, a simple approach will be presented to take into account the different pose estimations into one final estimator. The two sources that must be taken into account are the "`navdata`" quantities and our visual odometry. As a remainder, the `navdata` is the ensemble of data directly produced by the Parrot embedded software. It includes namely fused sensor readings of the gyroscopes (for three axes angular positions), optical flow sensor readings (x and y velocities) and ultrasonic sensor readings (altitude).

Unfortunately, these measures suffer from severe and non deterministic drift (see Appendix B). For this reason, the `navdata` alone can not be used for autonomous tasks. Indeed, the drone needs a reliable representation of its pose in its environment.

On the other hand, the SLAM algorithm that we implemented suffers some bad properties:

- The visual estimator is available at much lower frequency (see Appendix A) due to high computational cost.
- The visual estimator is available at variable frequency. Which is undesirable for the controller, for instance.
- Some intervals between two receptions are too large due to an extended "tracking loss" situation, and depend on the environment (its visual texture).
- Using only the visual pose estimation while building the map leads to a mismatch between position estimation in the real world and in the map. This is problematic in a visual tracking loss situation, because there would be no way to recover any viable position estimation anymore, and the map would stop being incremented.
- The visual estimator is subject to noise that can hardly be filtered without decreasing the information delay due to its slow update rate.

A way of combining the strengths and weaknesses of these two sources of information is to imagine a strategy that helps the drone to decide of a fused quantity it should rely on. This can be done by advanced sensor fusion techniques, like Kalman Filters (KF) or Extended Kalman Filters (EKF) (see Section 2.2). But simpler techniques can also be used, like simple laws on when to rely exclusively on one information source. The node `pose_estimation` was created for the purpose of containing all sensor fusion strategies, outlier filtering, unit conversions, etc.

At the present time, a simple but effective approach has been implemented. It can be summarized as follows:

1. The on-board sensors (navdata) that are subject to drift are corrected by the visual-pose estimation whenever it is available.
2. The on-board sensors that are sufficiently good are used directly.

A measurement is considered good enough either because an efficient on-board fusion technique is used or because the measurement device provides an absolute measurement in

itself. Indeed, in mobile robotics, it is important to distinguish between proprioceptive sensors and exteroceptive sensors. The first category measures a local observation from which it deduces something about the displacement (like incremental odometers), while the second category directly bases its observation on its environment, which provides an absolute reference. This second category is referred to as absolute sensors in this text. These include for example the visual (feature based) information, and the ultrasonic sensor, which provides an absolute distance measurement.

Altitude, roll and pitch

Thus, the altitude measured with the ultrasonic sensor will be used as is. The pitch and roll information, however, are deduced from an angular velocity (provided by gyroscopes). Thus, an integration must be done to recover the angular position, implying an incremental estimation. The incrementation is subject to error accumulation and is obviously not an absolute way to measure the angular position. Nevertheless, a closed-source fusion is done on-board, leading to very accurate (seemingly absolute) angular positions while the drone is in flight. This can be explained by the use of several other quantities available on-board:

- The IMU, providing as absolute reference the downwards gravitational acceleration.
- The magnetometer, providing as absolute reference the earth's magnetic field.
- The motor commands, informing on how probably the drone is actually tilted or flat.

Yaw, x and y

In the x and y directions, an optical flow sensor is used (or rather, an optical flow algorithm executed on-board using the bottom camera video stream). It provides linear velocities that must also be integrated to recover the position. This integration also induces drift in the estimator, but luckily, this integration is done at our side of the code, giving us more control on what is actually done. This will be used to implement the fusion with the visual information. Finally, the yaw information keeps subject to important drifts, even in flight. It will thus also be corrected in our fusion technique.

Appendix B gives a more detailed analysis of the drift problems related to the `navdata` sensor readings, while Figure 4.26 summarizes the pose estimation process. The green hexagons are representing the fusion that is described below.

Fusion mechanisms for x, y, and yaw

The mechanism is simple. Between each visual pose estimation received from the SLAM technique, only the `navdata` pose is taken into account. Then, when the visual estimator arrives, the `navdata` estimators for x, y and yaw are re-calibrated. By re-calibration, it should be understood "replacement". Unfortunately, since there exists no way of re-calibrating these quantities by software directly into the Parrot system, a buffer variable is needed, where each variation of `navdata` measurement must be incremented progressively. It is this buffer that must be flushed to be replaces by a visual pose estimation whenever it is available.

As already stated, the x and y estimations are already available as position increments provided by the integration of the x and y velocities. This is convenient since it already permits to

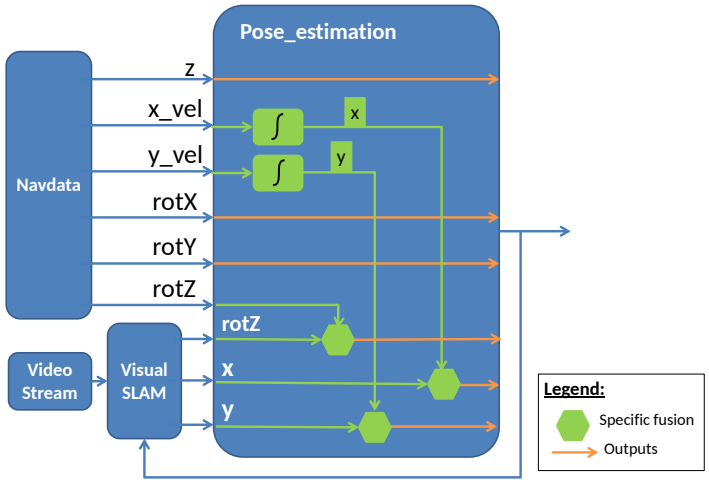


Figure 4.26: Summary of the inputs and outputs of the `Pose_estimation` node

adopt this strategy. At the opposite, for the yaw measurement, the increments must be deduced and kept in memory.

Another problem that has not yet been addressed is the delay between the moment a picture is taken and the moment the associated visual pose is fed to the `pose_estimation` node. It is clear that if the visual estimation arrives 300 ms late while the drone is flying at 3 m/s, an error of about 1 m would be introduced through this method. Since the drone velocity is unknown and the delay is non-deterministic, the only way we could think of to counter this, is to keep all increments in memory along with a corresponding timestamp. Then, a timestamp was added to the visual pose estimation messages too, corresponding to the instant the picture was taken. When a visual pose message arrives, all the increments that were added since the current visual correction are summed-up again and added to the visual estimation. Then, the list of older increments (the ones that were already too old for this iteration) is flushed from memory. This list of increments and timestamps was implemented using queues, and more practically, using the `std::queue`[13] dedicated queue management library. This is illustrated at Figure 4.27.

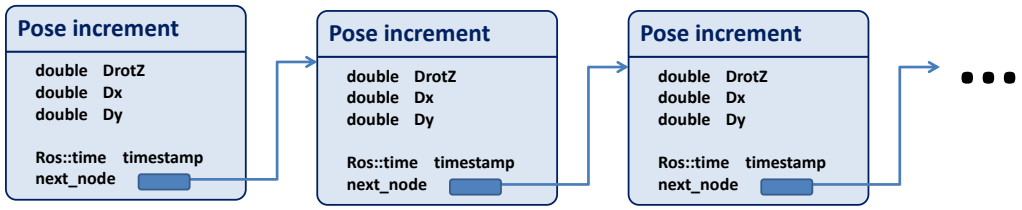


Figure 4.27: Example of a queue for the pose estimation increments that must be added when a visual correction is done.

Figure 4.28 compares a hand-drawn prediction of what should be expected from our estimator behaviour and reality. During the experiment, the drone was kept static. The red curve clearly

[13] see `http://www.cplusplus.com/reference/queue/queue/`

shows how the x-displacement estimation wrongly drifts away, while the yellow curve (visual estimation) remains zero-centered. It also shows how the correction mechanism induces noise on the fused pose (blue curve), while the red curve seems very smooth and high-frequency-noise-free. Finally, the delay at which the `navdata` pose is corrected makes it difficult to observe the sawtooth behaviour that was predicted.

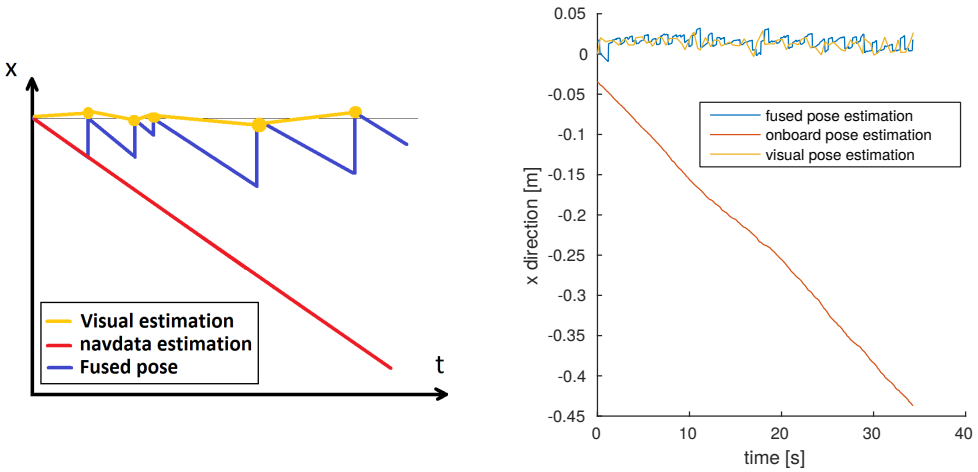


Figure 4.28: Effect of the re-calibration of the `navdata` pose estimation with the SLAM pose estimation. (a) Hand-drawn expected behaviour; (b) Measurement when the drone is kept static.

During another experiment, the drone was left flying in "hover" mode (i.e. Parrot controller stabilizing around $(x, y, z, rotX, rotY, rotZ) = (0, 0, 0.5, 0, 0, 0)$). In Figure 4.29, the effect of the delay on the propagation of information was drawn (in fuchsia). Note that the red curve seems to be the right one here, but it only denotes the fact that it is this quantity that is used for position control. In fact, the drone was spinning during the experiment.

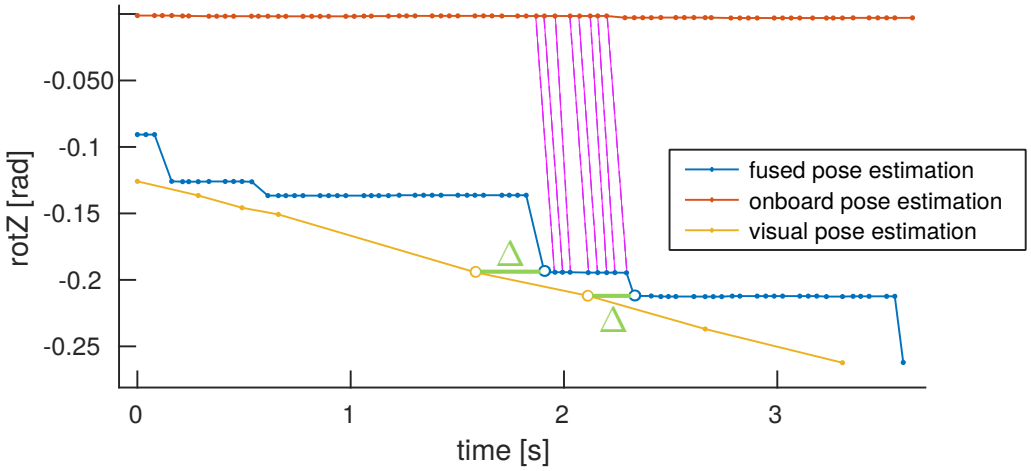


Figure 4.29: Effect of the re-calibration of the `navdata` pose estimation with the SLAM pose estimation. MATLAB® post-edited experiment results.

The choice could also have been made to memorize increments for the altitude and the roll and pitch measurements, and then to re-calibrate these quantities when the visual pose

estimation is available. But for the reasons listed above, the onboard measurements are slightly more efficient. Indeed, they already provide an absolute measure, while having smoother slopes (less subject to high frequency noise), and higher and constant update rates. All these properties are desirable for the controller, for instance. Note that in Figure 4.28 and Figure 4.29, this noise happened to be not particularly visible in comparison to results obtained during other experiments such as in Figure 5.10.

Finally, this combination of sensors is used in the mapping process as well as in the controller. Indeed, as already mentioned, a perfect coherence between the visual odometry based on the map and the position in the real world is necessary to be able to switch to pure `navdata` quantities when the "tracking lost" problem occurs.

Future improvements

An example of improvement of this method would be to include the IMU information, making our system a VINS (see Subsection 2.2.4). Indeed, the IMU information is at present disregarded, except maybe in the Parrot internal fusion algorithm, to which we have no access.

Another possible improvement would be to implement a Kalman filter. If so, several existing solutions should be envisaged:

- The `Kalman_CPP` library: a "Kalman filter for ARDRONE (Parrot) 2.0"[14].
- The implementation of the *TooN Algorithm Library*[15].
- The `robot_localization` ROS node[16].
- An OpenCV implementation[17].
- Inspire from the `tum_ardrone` Kalman filter.

However, it should be kept in mind that the different information sources that are available all have different update frequencies. Moreover, even the optical estimator has a variable rate. Then, no efficient way exist to filter non-deterministic drift on sensors. For example, Kalman filters usually assume zero-centered noise. Finally, the raw sensor readings are not available, preventing to use directly the unfiltered ultrasonic sensor readings for example. This implies non-linear and difficult-to-model sensor behaviours. According to us, the idea of putting a Kalman filter on top of an unknown pre-existing filter could lead to lots of design troubles for a very hypothetical improvement. This is why our sensor fusion technique, which is an effective technique, has been used until now.

If a Kalman filter was to be implemented in future developments, a change of hardware should be envisaged to a more open-source hardware, where lower-level functions are also available, like raw sensor readings.

[14] `https://github.com/PBRT/Kalman_CPP`
[15] `https://www.edwardrosten.com/cvd/tag/html/group__kalmanfiltergroup.html`
[16] `http://wiki.ros.org/robot_localization`
[17] `http://docs.opencv.org/trunk/dd/d6a/classcv_1_1KalmanFilter.html`

4.7 Code architecture

The aim of this section is to sum up all the methods we implemented, and explains how it was actually coded, following a typical "ROS nodes" scheme. This ensures parallelism during execution and also granularity, which is desirable for maintenance and future development of the code. The principal interactions between nodes and their purposes are resumed in the representation of the whole code architecture in item 4.30 (along with the nodes implemented by our colleagues). The emphasis will also be put on some details linked to the use of ROS.

ROS provides an elegant way to launch several nodes and to pass parameters to the latter. These instructions are written in a `.launch` file. Several examples of these files are provided with different configurations into the code archive. Let us review all the nodes called in these files.

First of all, the `ardrone_autonomy` node is used as a driver to communicate with the drone. It internally uses the Parrot SDK and translates the sensor readings and the commands into the standard ROS messages format. In fact, it acts like a wrapper for the Parrot SDK, so that only a simple drone driver is visible from the ROS side. Please read the author's documentation[18] for the explanation of each `.launch` parameters for this driver node. The two main outgoing topics used are the video stream (front OR bottom camera) and the onboard sensor fusion readings (`navdata`). The ingoing topics are velocity commands `cmdvel` and `takeoff` and `land` commands.

The node `strategy` has the role of the brain in the architecture and determines which is the current behaviour to engage according to the selected mission. The node `controller` is a position controller and consists mainly in a simple PID. It is the only node which sends velocity commands to the drone. The `path_planning` determines a set of via points according to the behaviour selected by the strategy, the current state of the quadcopter and the detection of a target. These via points are streamed one at a time to the controller. For more details about the implementation of these three nodes, please refer to the report of the other group [153].

These three nodes implemented by the other team need the following informations:

- The estimated position of the drone relative to its starting point (in world coordinates).
- A message stating if the target is detected, and if so, its position in world coordinates.

Let us now enumerate the nodes put in place by our team to make this possible.

The images of the video topic are first processed by the `image_proc` node for the lens distortion correction (as illustrated on Figure 4.6). It was omitted on item 4.30 for readability and also because this node was already implemented in ROS.

The `computer_vision` node performs the keypoints detection and tracking, and computes their feature description (see Subsection 4.2.2). It also detects the predefined target and removes its keypoints from the candidate list which will be used to localize the quadcopter and build the map (see Section 4.3).

[18] `ardrone_autonomy`: `http://wiki.ros.org/ardrone_autonomy`

The mapping node contains our implementation of the Keyframe-Based SLAM technique. Using the keypoints received from the computer_vision node and based on the previously built map, a visual estimation of the pose is computed (see Section 4.4). Keypoints not previously mapped are also added to the map if needed (see Section 4.5). If the visual pose estimation fails, due to the loss of visual tracking, the pose from the pose_estimation node is used to add the last new keypoints to the map.

In the pose_estimation node, the final pose estimation of the drone is computed following a simple fusion approach (see Section 4.6), then published to be used by several other nodes, like the controller, the path_planning and the strategy. The published 6 DOF pose estimation is the result of the fusion between the visual estimation based on the map and the onboard estimation readings.

A last node that is not represented here is the swarm_initialisation node. This node is used for a mission which involves several drones. Depending on the predetermined kind of mission, this node broadcasts messages containing a list of available drones on the mission and a role assignment for each quadcopter. This node can either be used only at initialisation.In this case, it just configures some mission settings. Or it can be used as a "master" node, managing the whole swarm like a common brain. If it is embedded on a single drone, this would imply the existence of a chief drone, endorsing more responsibility than the others. This node being still not very elaborate, this design choice is let for future developers.

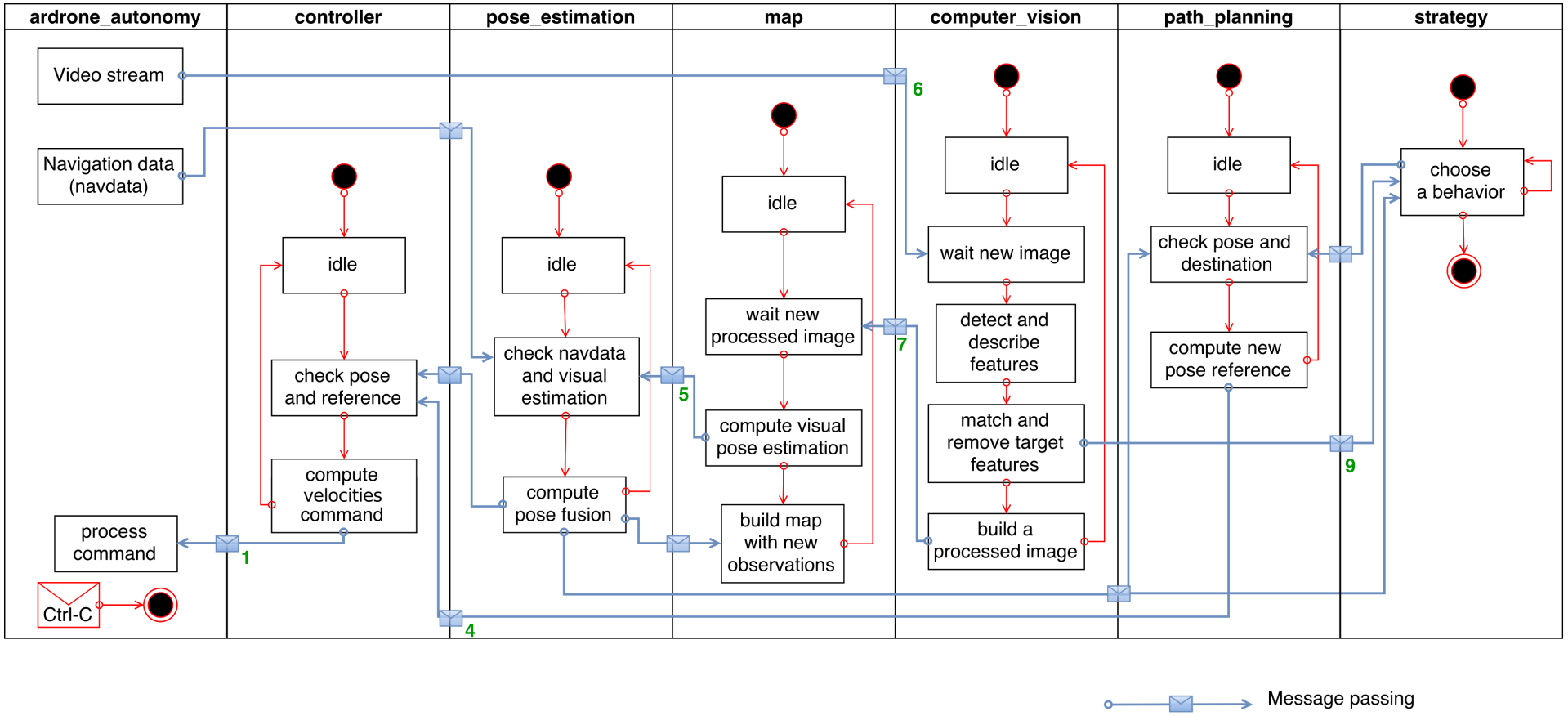


Figure 4.30: Code architecture summary

Messages Legend:

1. Velocities commands (`Twist`)
2. Navigation data (`Navdata`)
   contains onboard pose estimation
3. 6 DOF Estimated pose (`Pose3D`)
4. Next destination (`PoseRef`)
   next via point in world coordinates
5. 6 DOF Visual pose estimation (`Pose3D`)
6. ROS `Image`
7. `ProcessedImageMsg`
8. Behaviour (`StrategyMsg`)
   contains a number to identify the predefined behaviour
9. `TargetDetected`
   published only when the target is detected, contains the current pose of the drone, the image coordinates and the world coordinates of the target

Chapter 5

Performances and results

In this section, some main results achieved by our home-made SLAM algorithm will be presented. They include the precision and absoluteness of the final pose estimator, and code execution timings.

5.1 Precision of the visual odometry for pure x-y translational displacements

5.1.1 Description of the experiment

Since our lab is not equipped with any motion capture system or other ground based measurement system, we have no ground truth to correctly compare with our estimation in flight. For this reason, the drone is maintained perfectly flat on a rolling table, and manually moved perfectly parallel along a wall such that an ideal translational motion is executed (see Figure 5.1). In addition to this system, a small desk lamp is added to reduce the effect of the neon lights, which constantly flash light at a frequency of 100HZ, reducing the robustness of the image descriptors we track.

The reason we plotted only the visual pose estimation and not the total fused pose is that, when not in flight, the *Parrot* software prevents us from having access to a large part of the sensor readings. The reason for this is that a kind of filtering is done on-board, which does not function right if no commands are sent to the rotors. For example, in the case of a Kalman filter, this would interfere with the prediction step of the filter. We do not know precisely which sort of filter is used (this is not communicated by *Parrot*). This observation is very important to note, since it was a very annoying constraint during the whole development and validation process. This highlights the need for more open-source and tunable embedded electronics (autopilot and board computer), and, even better, for a motion capture system.

Figure 5.1: Description of the experiment for measuring the visual pose estimation for pure x-y translations

5.1.2 Results on a well textured environment

For the first experiment, some printed images were put on the ground, providing good visual texture. In this case, the visual SLAM technique is the most robust, since a lot of easily detectable features are present on the patterns. As seen on Figure 5.2, on a 2 meter displacement, an error of 0.194 m has been measured in the x direction (thus 9.71%), and 0.028 m in the y direction (thus 1.41%).

It should be noted that in this experiment setting, the y direction corresponds to the large side of camera pictures. Figure 5.2 illustrates that the error made on the movement in this direction is smaller. This corresponds to the intuition because the visual range is bigger and more features are visible at a time. This leads to a more accurate estimation.

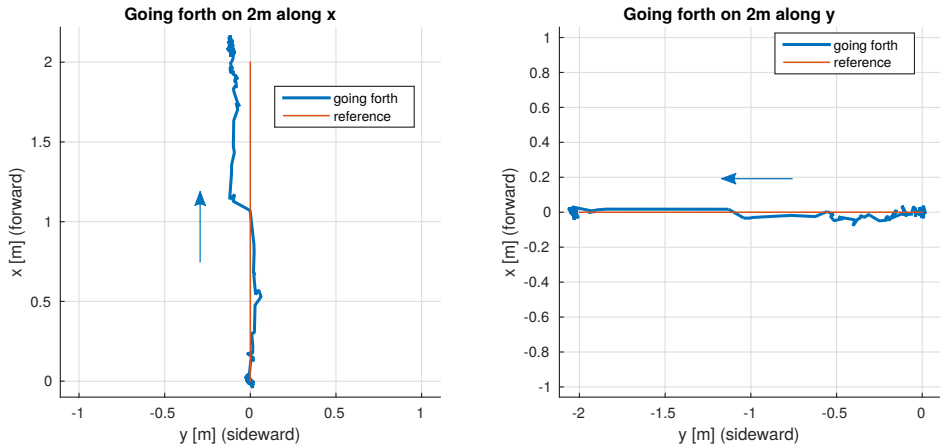


Figure 5.2: Measurement of the precision of the visual pose estimation for a pure translational displacement in the x direction (a); and for a pure translational displacement in the y direction (b)

Concerning the map, 15 keyframes were taken and successfully mapped during the y displacement. The last keyframe to be taken was estimated to be 0.194 m further than it actually was, so the mismatch between the world and the map is thus a scale factor of 1.0971. This means that, if put in front of each other, the world and the map should respectively look like in Figure 5.3. However, as it is stated in the SLAM problem, the estimated position keeps absolute with respect to the map, not to the world. This means an overall capacity to return with absolute accuracy to a known place in the world. This point is illustrated in Section 5.2.

Figure 5.3: Comparison of scale of the map versus the real scene

5.1.3 Results on a cluttered environment

In Figure 5.4, it is shown that the SLAM algorithm still works in an environment in which none of the previous images are used, provided that the environment still presents enough "visual texture". This is achieved by putting random objects on the ground (see Figure 5.5). This kind of environment is well known in the visual SLAM literature as a *cluttered environment*. On a 2 meter displacement, an error of 0.4383 m has been measured in the y direction (thus 21.91%). This is less precise than with textured environment but it still works better than with pure onboard sensors readings. However, the less visual texture, the less robustness and precision should be expected from the visual SLAM algorithm (see results of Section 5.2).

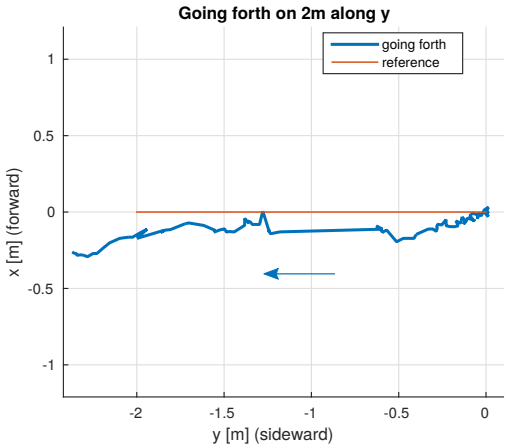


Figure 5.4: Measurement of the precision of the visual pose estimation for a pure translational displacement in a cluttered environment

As for the map, in Figure 5.5 it is clear that the correspondence between the scene and the map is more difficult for a human to interpret since the locus of the detected features is not easy to anticipate.

Figure 5.5: Cluttered environment used in Subsection 5.1.3, and associated map. The map is difficult to interpret

5.2 Absolute positioning

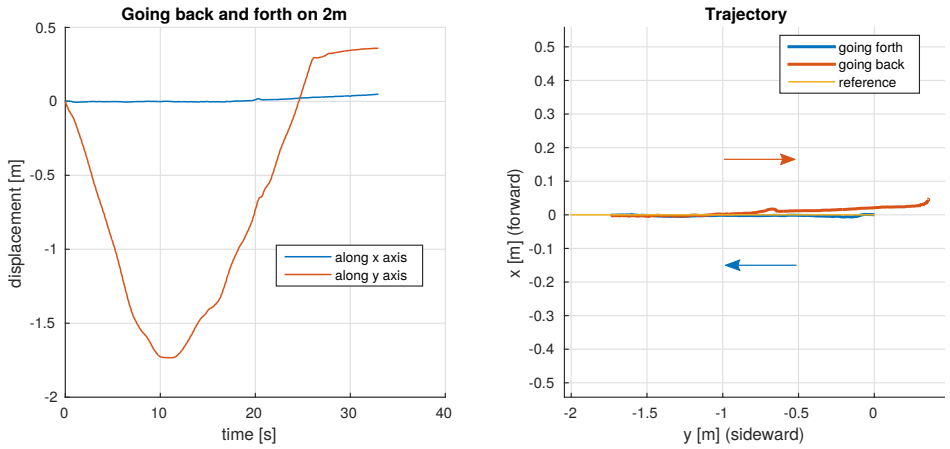
Due to technical limitations, it is difficult to gather more measurements characterising the precision of the pose estimation, as explained in Subsection 5.1.1. For this reason, this section will concentrate on showing another crucial aspect of our SLAM technique: the absoluteness of the estimated pose with respect to the constructed map.

Towards this end, the same experiment as in Subsection 5.1.2 was repeated, but this time the one-way translation was replaced by a back and forward movement of 2 meter. Results are shown in Figure 5.6 and Figure 5.7 for an y and an x displacement respectively.

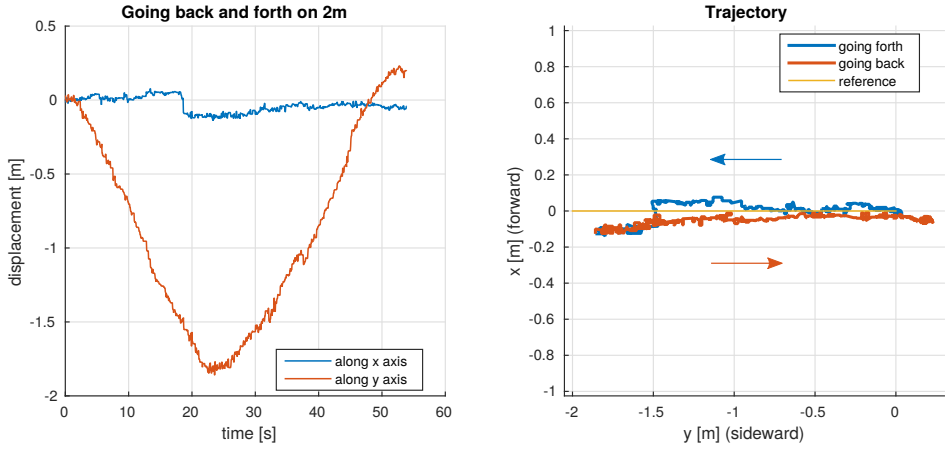
In both cases, one can notice that, when based only upon the original sensor readings, the estimated trajectory does not return to the starting point. Indeed, due to the sensor drift, while the drone has returned to its exact starting place (0;0), the estimated pose is mistaken by several centimeters. In case visual odometry is used, this error decreases significantly, but the closure of the loop is still not achieved and the pose estimation is not absolute w.r.t the map. Finally, when we include the search among older keyframes in the map, if the tracking of one particular keyframe is lost, the method becomes true SLAM, as defined in [38], and the loop closes well, with a sub-centimeter precision. From this, we conclude to the absoluteness of our pose estimation in the operational conditions of the experiment.

However, even if the visual odometry is less precise than full SLAM in the case of a loop-closure in the trajectory, it can still be envisaged as a lighter solution for pose estimation. Indeed, disabling the search among older keyframes in the map highly reduces the overall computational overhead (see Table A.1), while also preventing some odd situations, like when two very similar patterns are observable in different places, which can cause "teleportations" in the pose estimation, thus parasitic jumps in the drone displacements.

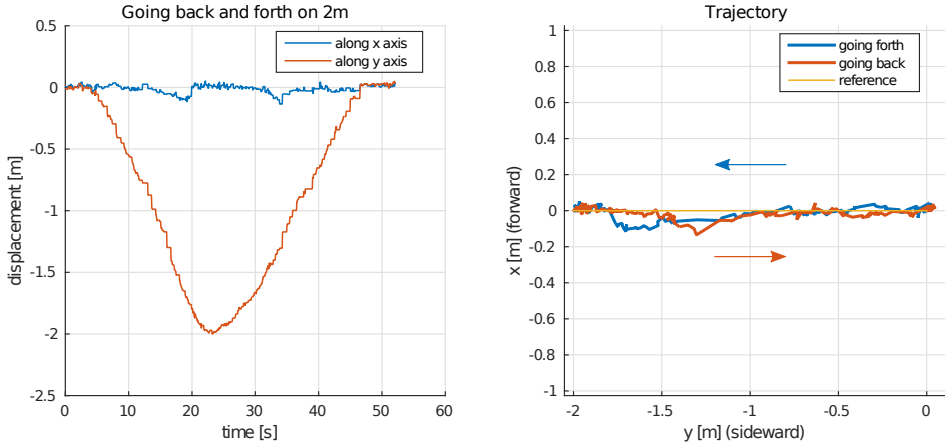
Figure 5.6: Illustration of the absoluteness of the pose estimation with respect to the world (y displacement). The comparison between (a) and (b) shows that the visual estimation improves the displacement estimation, and on (c) our SLAM implementation performs accurate backtrack.



(a) estimation based on sensors

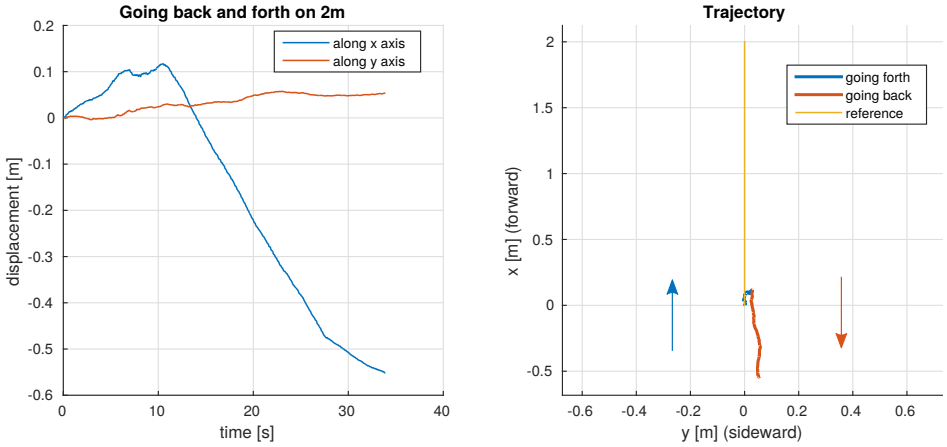


(b) estimation based on visual odometry

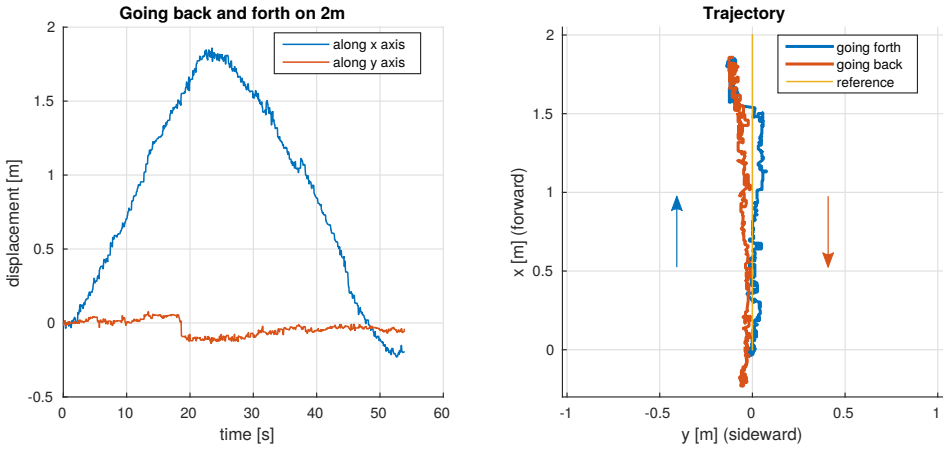


(c) estimation based on visual odometry and map (SLAM)

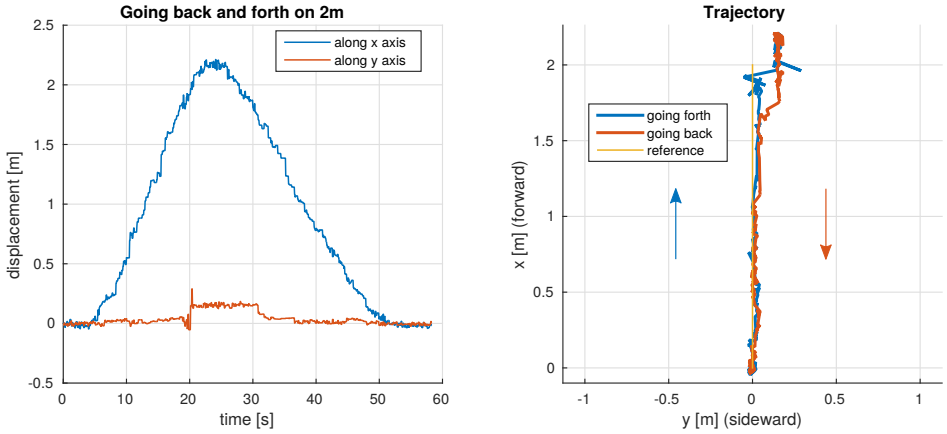
Figure 5.7: Illustration of the absoluteness of the pose estimation with respect to the world (x displacement). The comparison between (a) and (b) shows that the visual estimation improves the displacement estimation, and on (c) our SLAM implementation performs accurate backtrack.



(a) estimation based on sensors



(b) estimation based on visual odometry



(c) estimation based on visual odometry and map (SLAM)

In Figure 5.8, an illustration of the map created with only visual odometry is provided. It illustrates well the situation in which the loop closure does not operate ideally. Indeed, the last keyframe added to the map should correspond exactly to the first one. Instead, the same keyframe is repeated by superimposition in the map with about 4 cm error. Regarding the total length of the loop, this can be acceptable depending on the application targeted.

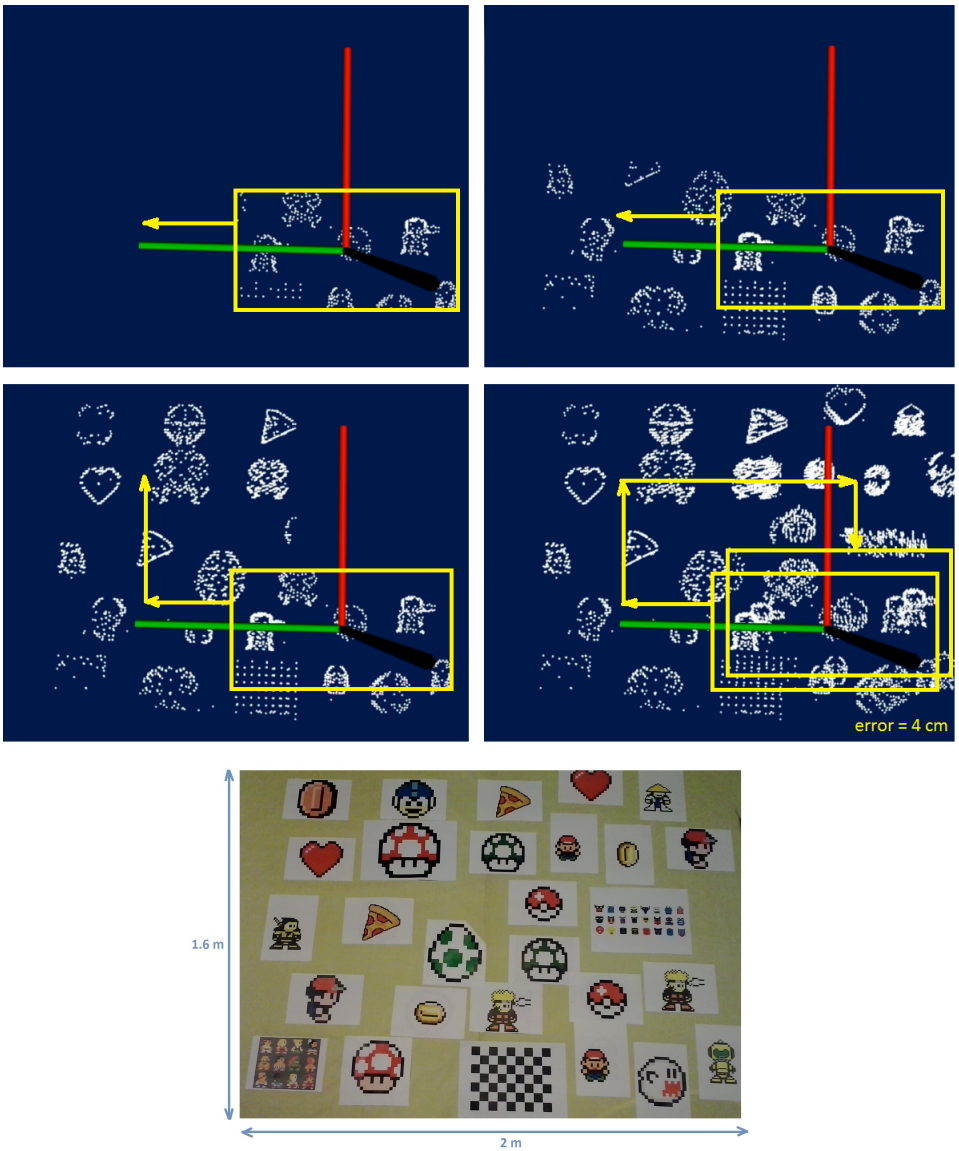


Figure 5.8: Illustration of the bad loop closure in the map while only using visual odometry

In contrast, the Figure B.1 shows the map created when using only the basic sensor readings. In this situation, the drone does not even move. But even so, the result is clearly not acceptable.

Finally, let us note that the search amongst previous keyframes is not sufficient to perform a re-adjustment of all previously mapped points when performing a loop closure. Indeed, as explained in Chapter 4, bundle adjustement is needed to perform a pose correction of several already mapped keyframes.

5.3 Results of the 6 DOF pose fusion

In this section will be discussed the main results of our simple sensor fusion strategy presented in Chapter 4.

5.3.1 Experiment with the drone flying while stabilised by the Parrot controller

In Figure 5.9, the drone is flying and controlled by the *Parrot* on board controller. Thus, it uses as pose feedback only the on-board pose estimation (in red), and its reference signal is $(x, y, z, rotX, rotY, rotZ) = (0, 0, 0.5, 0, 0, 0)$. Let us analyze each component of the pose.

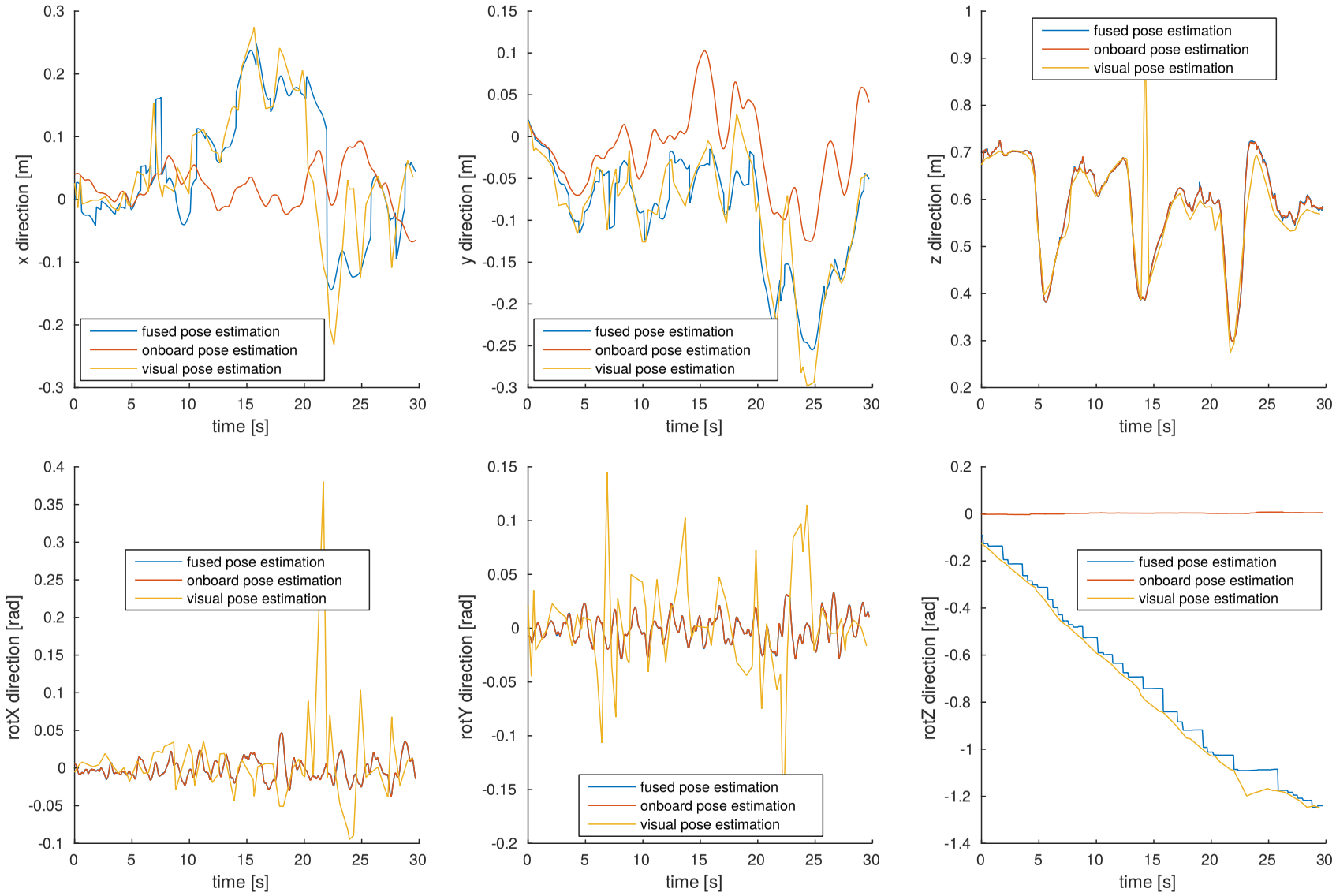
x and y While the red curve is indeed centered around zero (which means the controller has no static error), the visual odometry curve (yellow) indicates that the drone is drifting regarding its absolute visual reference of the moment. The fused pose takes this information into account to correct the pose estimation (blue curve). Unfortunately, during this particular flight, no constant drift has been recorded, making it difficult to interpret whether this drift physically occurred. Indeed, the amplitude of the presumed drift is not significantly larger than the amplitude of noisy displacements on this particular measure. For more explicit examples of drift corrections, see Chapter 4 or the behaviour in the rotZ direction, commented below.

Altitude Here, only the *Parrot* estimation of altitude is used. This is why the blue and red curves superimpose. Indeed, it is based on an ultrasonic sensor and a closed source on-board fusion algorithm (see Appendix B). These sensors already have a zero static error by nature. This is why the visual pose estimation (yellow curve) and the ultrasonic sensor signal (red curve) globally match. Since the visual pose estimation is available at lower frequency, and that it can present some noise (like what seems to be an outlier at second 15), it is natural to rely on the ultrasonic sensor reading. For what concerns the large fluctuations of altitude, this revealed to be a typical behaviour of the *Parrot* altitude controller.

rotX and rotZ Similarly to the altitude estimation, these sensor readings (in red) are fused on-board, by closed-source software (see Appendix B). Therefore, since these curves are well centered around the same mean as our absolute visual pose estimation, it was opted for the pure closed-source on-board estimation. Indeed, it has a better update rate and presents less noise.

rotZ Here, as for the x and y directions, it is absolutely necessary to correct the pose deduced from the sensor readings by a visual estimator. Indeed, the presence of a totally non-deterministic (and sometimes very large, see Appendix B) drift can be observed. Thus, in the compromise between drift cancellation and both an increase of noise and a decrease of refreshing rate, the drift cancellation has been chosen. The typical staircase effect on the fused pose (blue curve) is due to the periodic corrections brought by the visual pose estimator, which is updated at lower frequency, as explained in Chapter 4. Indeed, it is only updated at about 2.3 Hz, due to computation time limitations, and is only used to re-calibrate the blue curve (see Chapter 4). Finally, let us note the successful fusion, in which the re-calibration process unfortunately introduces abrupt slopes, which must be treated by the controller (for example for the derivative term of the PID, to avoid too large values).

Figure 5.9: Comparison of the pose estimation based upon on-board sensors, the pose estimation based on the SLAM algorithm, and the fused pose. For this experiment, no ground truth positioning is available, the drone is controlled on board by the closed source Parrot software at $(x, y, z, rotX, rotY, rotZ) = (0, 0, 0.5, 0, 0, 0)$ (using onboard pose estimation in red).



5.3.2 Experiment with the drone flying but maintained strictly fixed

To have a better idea of the ground-truth pose of the drone, since our lab is not equipped with any motion capture system or equivalent, the following experiment was imagined.

To ensure all the sensor readings are available, the drone must think it is flying. So, by turning on the rotors while maintaining the drone at a known position and orientation, one can observe all the components of the pose (see Figure 5.10). In this situation, the drone is virtually controlled around

$$(x, y, z, rotX, rotY, rotZ) = (0, 0, 0.81, 0, 0, 0)$$

This time, an important drift in the x and y direction has been recorded, and the usefulness of visual-based pose corrections is better illustrated. It can also be noted that in this experiment, it is the blue and yellow curves that are stabilized around their reference values, as can be expected from absolute measurements. Now it is the red curve that drifts away, giving better insight in what is truly happening.

Next, the effect of the on-board closed-source sensor fusion can be observed within the altitude curves. Indeed, an offset of about 15 cm between the true value and the estimation can be observed. This is due to the fact that the drone is firmly kept in place while flying. This causes a distortion of the normal drone behaviour, in the sense that the rotors must be less solicited to rest at a higher altitude than the reference signal, which is miss-interpreted by the fusion algorithm.

Finally, let us once more highlight the compromise that has to be done between a noisy and slow rated estimator, and a fast rated and clean, but drifted estimator.

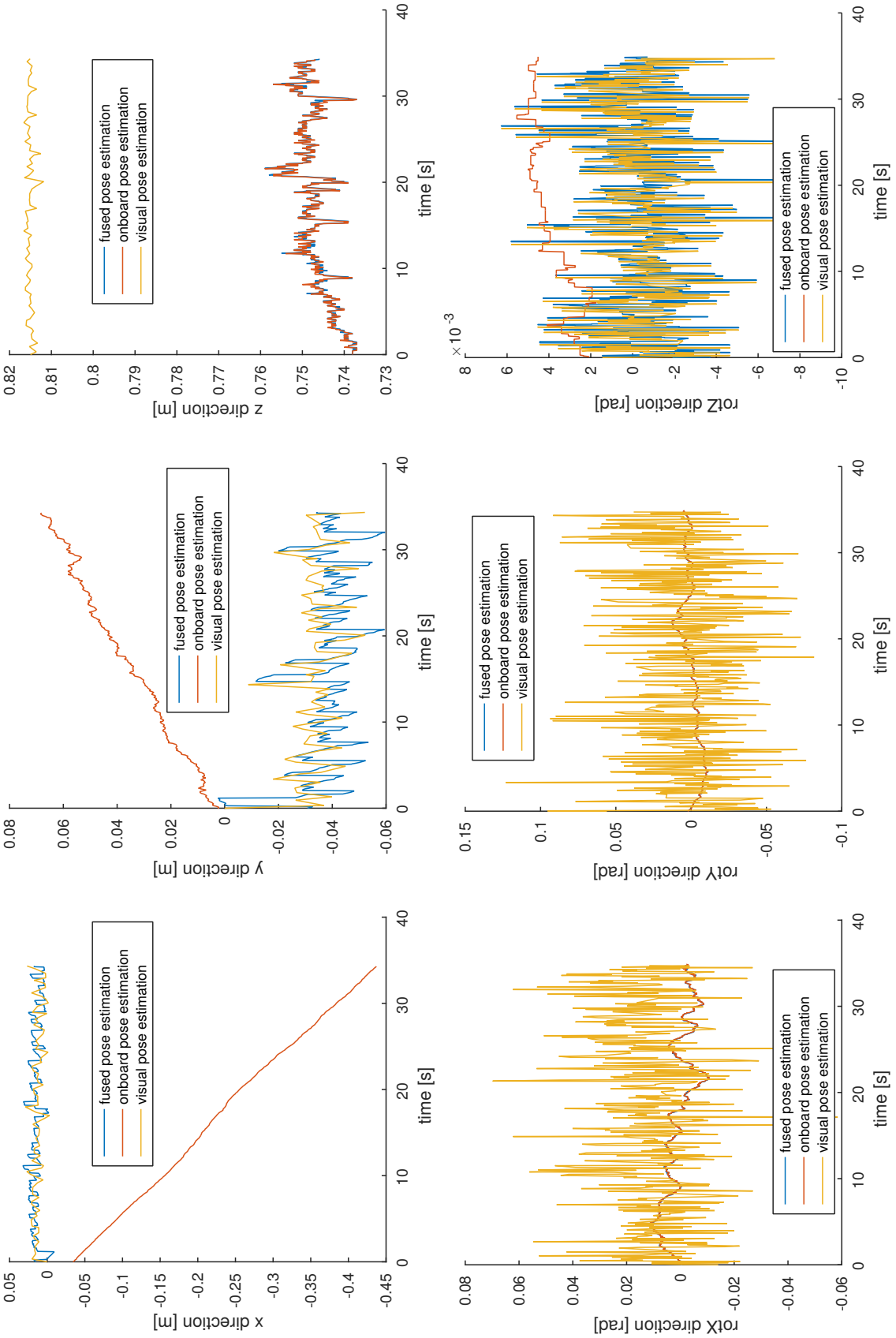


Figure 5.10: Comparison of the pose estimation based upon on-board sensors, the pose estimation based on the SLAM algorithm, and the fused pose. The ground truth positioning is known as the drone is fixed at $(x, y, z, rotX, rotY, rotZ) = (0, 0, 0.81, 0, 0, 0)$.

5.3.3 What happens when the visual odometry loses track

In this section, no measurement can really be taken, but the behaviour of the drone in such situations is an important feature achieved in our algorithm.

When the drone loses track of where it is w.r.t. the map, due to a lack of visually recognisable elements on the ground, the drone just keeps flying normally, relying on the basic sensor readings. Then it directly begins mapping what it sees in its presumed area of the map, based upon the pose estimated with the basic sensors. Whenever the environment presents enough texture to be able to rely on the created map again, then the map is used for pose estimation, otherwise it is disregarded.

In other words, everything that has been coded can theoretically, in the best case, improve the pose estimation by providing an absolute internal reference (the map), and, in the worst case (i.e. a poorly textured environment), have no effect at all on the basic pose estimation provided by a direct reading of the sensors.

5.4 Comparison of robustness with and without neon lighting

A simple experiment was put in place to quantify the negative effect of neon lightning on our visual SLAM technique. The drone was put at rest in front of one particular image and the lightning conditions were changed between the normal neon lightning of our lab, and one obtained with only a desktop lamp. In both cases, various measurements were taken: the number of detected keypoints, the number of keypoint matches between two consecutive images, and the number of inliers of the RANSAC method for selecting the best matches. Results are shown in Table 5.1.

	neons	desktop lamp
detected keypoints	160 to 249	175
matched keypoints	128 to 180	174 to 175
RANSAC inliers	96 to 136	163 to 170

Table 5.1: Characteristics of the keypoint detection under different lightning conditions

From these results, it is clear that the stroboscopic effect of neons substantially reduces the robustness of keypoints detection. The only observation that could seem contradictory is the total number of detected keypoints, which is slightly larger in the case of neons. This is explained by the overall inferior light intensity obtained with the desktop lamp that was used.

5.5 Timing measurements

Since the code execution timings are critical for on-board real-time computation of the SLAM algorithm, some code optimisation has been done. At present, everything works fine in real-time, but not on-board. Indeed, a pc running Ubuntu 14.04, with an Intel® Core™ i5 CPU 650 @ 3.2 Ghz x 4, and an 8 Gb RAM has been used until now for the development phase. Even so, the code execution timing lacks some performance. Thus, this point must remain at the center of attention for the future thesis. In Figure 5.11, the duration of the node execution were measured as a mean on a typical flight of one minute. The execution timings have been compared with the ones obtained last year, but unfortunately, the analysis led last year did not cover the same code aptitudes. Moreover, the code was organised following a slightly different architecture. Finally, the figure included in their report is not clear about the quantity that is measured. For these reasons, it is difficult to establish any comparison.

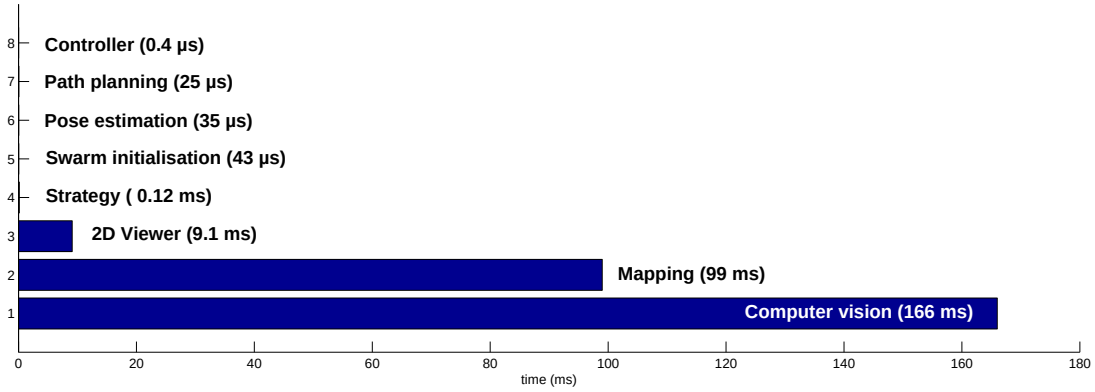


Figure 5.11: Code execution timings

Let us note that ROS being not a real-time environment (it is a program emulating a pseudo-real-time OS running on top of a non-real time Linux kernel), a compromise had to be done between the code modularity and the timing performance. Indeed, the code is organised into nodes that execute in a multi-threaded fashion corresponding to the tasks of a real-time system (see Subsection 3.2.2). Thus, respecting this node principle for the sake of modularity implies to let the hand to the ROS system, over which we have no control. Moreover, the communication between nodes for shared variables also increase the overall computation time, since MUTEX (mutual exclusion) and synchronising systems have to be put in place in the ROS master. This way of organising the code was done to best respect the ROS philosophy, and to be easier to transpose to a truly real-time system.

For more complete timing analysis, refer to Appendix A.

Chapter 6

Conclusion

This year, we have achieved the different objectives that were announced in Section 1.7:

- Test state-of-the-art implementations of SLAM.
- Propose an appropriate code architecture with ROS.
- Re-implement the past years benchmark mission with our own techniques, on the ROS platform.
- Implement a visual feature-based target recognition, and provide a target pose estimation.
- Implement our own feature-and-keyframe-based visual SLAM algorithm with full 6 DOF (3D) visual pose estimation and architecturally ready for 3D reconstruction and map interpretation.

This report covered the design choices, a description of our implementation, the underlying theory, and the main results. Besides, it also contained an analysis of the current state-of-the-art in the domain of autonomous navigation for drones.

In terms of functionalities, the improvements that were made regarding last years level, are:

- The possibility of tracking a moving target (see Section 4.3).
- A truly absolute positioning with respect to the map, implying the ability of recovering a known place with absolute accuracy (true SLAM) (see Section 4.4, Section 4.5, Section 4.6 and Chapter 5).
- A 3D conception of the environment (see Section 4.4 and Section 4.5).
- The use of libraries like PCL, that easily permit to implement higher level behaviours using pre-existing solutions that are open-source and well-maintained (see Subsection 3.2.3 and Section 4.5).
- An improved development environment (see Appendix D).

This being achieved, since the code is destined to be re-used and improved by the students of next year, good documentation was provided. This includes *readme* files informing on how to launch the program, the software versions, the files hierarchy, and so on. A Doxygen-generated documentation was also given, and the code has been cleaned up and well commented. The

whole `ucl_drone` package was already available on the ROS web page since June, and has been upgraded with the totality of this years functionalities, so that everybody can use it, comment it, and even collaborate on it.

Finally, let us point out the fruitful collaboration between our two teams of students. In spite of some additional organisational complexity, a good communication led us to achieve good results in combining our different code parts.

6.1 Difficulties encountered

The results are quite satisfying, after spending lots of hours on debugging the hardware, repairing drones, and solving practical issues that are proper to thesis subjects with technical aspects. Indeed, every single experimental measure is a puzzle when it comes to imagining reproducible results with limited laboratory installations and closed source hardware and software.

For example, in order to be able to measure the ground-truth pose of the drone without any motion capture system, the drone had to remain fix, but by doing so, the internal-sensors were misled. Moreover, if the drone was not flying, certain sensor readings were not available. On the six drones that are currently used in the lab, the sensors concerned were - of course - different from a drone to another, as if they all had their own personality. This, even though the same version of the firmware had been re-uploaded on the motherboards. This does not ease any debug process, since too much random factors can influence the outcome of a test.

Another very time consuming task was to quickly assimilate a lot of theory in multiple domains. Indeed, the drone field of research involves a great number of aspects of mobile robotics sciences, which is a highly interdisciplinary field. Months were spent on understanding concepts that were totally new to us, like the computer vision domain, the ROS environment, and the positioning problem for drones, that still remains a challenge for the coming years. The multiple code documentations as well as Parrot documentation were also a pain, or let us rather say, the lack of documentation and the errors contained in it.

Finally, taking into account the speed at which novelties occur in the world of drones, it is not hard to imagine how difficult it is to observe the "technological watch" principle. Indeed, this would represent a nearly full-time work. Fortunately, a large community spreads interesting news very fast on dedicated forums. These include the ROS forum, frequented by roboticists, drone aficionados and academics, but also the DroneCode project, and many others.

All and all, one main difficulty was thus the vastness of possible tasks to do. Indeed, the main labs that are shining by their impressive results are all composed of large permanent teams of researchers, technicians, assistants and students. Their studies cover several models of drones, from which several are either open-source or hacked. Consequently, it was difficult to limit ourselves to some specific features running on our limited hardware, by lack of time resources and human resources. However, we are happy of the specific features we decided to implement, since they permit to improve the drone pose estimation, which remains, according to us, the next great technology-unlocking challenge.

6.2 Future challenges

Room for improvements in the existing solution

As discussed in Chapter 4, some more advanced techniques can be envisaged for some parts of our implementation. We fixed a lot of issues and did not have much time left to implement all of them, but our search in the literature leads us to note the following possible improvements:

- Some pieces of code need to be optimized for better speed performances, for instance:
  - The keypoint detection is really time consuming.
  - The computation of the visual pose needs RANSAC, which calls P3P lots of time (PROSAC could be an alternative).
- An extended Kalman filter would be an interesting feature to improve the sensor fusion. The main difficulties to tackle are the large delay of the visual estimation and the fact that sensors are multi-rated. These need non-trivial adaptations of the classical solution.
- Implement the triangulation module to get rid of the last 2D hypothesis of the code and enable full 3D reconstruction. It seems to us that all the concepts needed to reach this feature are stated in our text and the code architecture of our mapping was designed with this objective in mind.
- Use a bundle adjustment library. This feature can be envisaged thanks to the data structures we implemented, but some important adaptations are probably needed. Indeed, the optimization to refine anterior pose estimation of the drone and the position of keypoints in the map needs to be executed in parallel with the other mapping tasks and it is computer intensive.
- Enable map sharing between drones. This feature is probably the most easy one, since inter-drone communication has already been put in place, as well as conversions between PCL point cloud formats and ROS messages. This would permit to have two or more drones collaborating on a same map, and most importantly, to localise relatively in an absolute fashion.
- Use a Voxel approach to interpret the map (with PCL) to be able to perform path planning algorithms. The 3D path planning part could be developed on simulator for saving time in solving troubles with the hardware.

Feasibility of long term objectives

The long term objective is to enable multiple agent systems flight. With the design choices made until now, this could naturally lead to autonomous multi-quadcopter mapping. As stated in the "Related work" Chapter 2, this is an active topic of research and even if some proposals of map communication are available in the literature, we did not find any effective project where multi-quadcopter mapping is achieved yet.

However, knowing all the possible applications for autonomous drones and the number of research teams working on it, it makes no doubt to us that applications such as large-scale collaborative mapping, assisted indoor inspection, drone-swarms rescue teams, etc. will be available with good robustness in no time.

The main problem actually remains the full 6 DOF pose estimation of the drone, independently of any visual texture, external beacons or gps. The most promising improvements of these last few years in the domain of computer vision aided navigation systems are the apparition (or the weight diminution) of sensors like RGB-D cameras, DVS, or stereo cameras. Nevertheless, we are convinced that a robust pose estimation in the future will result of the summed up effect of little improvements all along the system, like:

- Computer vision processors which appeared this last year.
- General computation performances of on-board processors.
- Clever combinations of lots of state-of-the-art fusion and inter-sensor calibration algorithms.
- New combinations of sensors (like wind-speed sensors combined with aero-dynamical models of displacement similar to the one of the Parrot AR.Drone).
- The miniaturisation and decreasing cost of sensors, permitting more and more redundancy.
- The quality of on-chip sensors (IMU, gyroscopes, magnetometers, ultrasound, etc.).
- Generalized theory for drone state model estimation.
- Active mapping integrated to SLAM.
- Inclusion of machine learning features.
- etc.

Indeed, as we have seen, even though the angle measurements deduced from gyroscopes are supposed to drift, the Parrot AR.Drone is able to stabilize remarkably, only through the usage of fusion techniques. This advance is already very promising.

Our perception for the way ahead

Looking back on a ten months experience with drones, we would like to give our advice on what we think could help the students for next year:

- If the project goes in a direction that favours a hardware change (advanced control techniques, Kalman filtering, more payload to embed custom electronics, ...), then focus on this hardware change. Indeed, we think that the choice of hardware, the mounting, the interfacing with ROS, the calibration, etc. are crucial enough to have one person work full time on it. Then a modular platform should be chosen based on open hardware and open software.
- Build a simple device with only cameras and IMU sensors to develop a low level sensor fusion.
- A stereo camera will simplify the development for 3D visual reconstruction. Indeed, the scale ambiguity from Section 4.4 is removed as the baseline length joining each camera center is fixed and not subject to uncertainty. RGB-D cameras can also be envisaged.
- Develop a quadcopter motion model, easy to identify, to be used for prediction in a Kalman filter for example, or for simulation purposes. Simulations can include the low level position control but also the swarm behaviour.
- Develop multi-robot strategy and mapping on wheeled robots to begin: this is less constrained by the real-time necessity for all computations, and the motion is less prone to disturbances.

These proposals are only advices that can be discussed with the future contributors to this project alongside other ways to achieve the long term goal.

Bibliography

[1] P. Fitzpatrick, *Perception and Perspective in Robotics*, MIT Artificial Intelligence Laboratory, Cambridge, USA, 2003.

[2] S. Waharte and N. Trigoni, *Supporting Search and Rescue Operations with UAVs*, University of Oxford, Computing Laboratory, 2010.

[3] S. P. Yeong, L. M. King, and S. S. Dol, *A Review on Marine Search and Rescue Operations Using Unmanned Aerial Vehicles*, World Academy of Science, Engineering and Technology. International Journal of Mechanical, Aerospace, Industrial, Mechatronic and Manufacturing Engineering Vol:9, No:2, 2015.

[4] H. L. Andersen, *Path Planning for Search and Rescue Mission using Multicopters*, Norwegian University of Science and Technology, Department of Engineering Cybernetics, June 2014.

[5] *S.W.A.R.M. Search With Aerial RC Multirotor, Volunteer Search & Rescue Network*, `http://sardrones.org/`, Consulted 22 May 2016.

[6] A. Huppertz, *UCL : ils veulent développer des drones bâtisseurs*, L'Actualité de Ottignies-Louvain-la-Neuve, `http://www.lavenir.net/cnt/dmf20160222_00784192/des-chercheurs-de-l-ucl-developpent-des-drones-batisseurs`, 22 February 2016.

[7] P. Latteur, S. Goessens, J.-S. Breton, J. Leplat, Z. Ma, and C. Mueller, *Drone-Based Additive Manufacturing of Architectural Structures*, Proceedings of the International Association for Shell and Spatial Structures (IASS), Symposium 2015, Amsterdam, 2015.

[8] SNCF, *Quel usage pour les drones ? 1/2 La maintenance*, BREVE - SNCF, `http://www.sncf.com/fr/content/brevepresseusage-drone09092015`, Published 9 September 2015.

[9] SNCF, *Quel usage pour les drones ? 2/2 La sûreté*, BREVE - SNCF, `http://www.sncf.com/fr/content/brevepresseusage-drone209092015`, Published 9 September 2015.

[10] *Amazon Prime Air*, `http://www.amazon.com/b?node=8037720011`, Consulted 19 May 2016.

[11] Intel, *A World Record: 100 Dancing Drones*, `http://www.intel.eu/content/www/eu/en/technology-innovation/aerial-technology-overview.html`, Consulted 19 May 2016.

[12] Microdrones, *Aerial photography with drones*, `https://www.microdrones.com/en/applications/areas-of-application/aerial-photography/`, Consulted 19 May 2016.

[13] Propellerheads, *Aerial Photography*, `http://www.propheadsphoto.com/`, Consulted 19 May 2016.

[14] K. Li, *Fukushima Radiation Disaster News: Japan Uses Drone to Check Rad Levels at Nuclear Site*, Website of the *LATIN POST*, http://www.latinpost.com/articles/6442/20140127/fukushima-radiation-disaster-news-japan-uses-drone-check-rad-levels.htm, 27 January 2014.

[15] *Ambulance Drone*, TU Delft, https://youtu.be/y-rEI4bezWc, Consulted 19 May 2016.

[16] N. Rowier, *Interconnexion de multiples drones AR Parrot. Technical report, for UCL/ICTEAM/INMA*, 8 December 2014.

[17] L. Hillen, *Behavior-based Robotics*, Seminar "Embodied Cognition", AG Technische Informatik Universität Bielefeld, 7 June 2011.

[18] T. Balch and R C. Arkin, *Behavior-based Formation Control for Multirobot Teams"*, IEEE Transactions on Robotics and Automation, 1999.

[19] Y. Deville, *AI - Course Introduction*, Artificial Intelligence (LINGI2261) Lecture notes, lecture given at UCL, 2015.

[20] *Glossary of Robotics Terms*, http://www.motoman.com/glossary.php, Consulted 19 May 2016.

[21] *Hydrogen-powered drone takes flight*, BBC website, http://www.bbc.com/news/technology-35890486, 25 March 2016.

[22] KMEL Robotics, *Flying Robot Rockstars*, https://youtu.be/Qlqe1DXnJKQ, Consulted 19 May 2016.

[23] GRASP Lab, University of Pennsylvania, *A Swarm of Nano Quadrotors-Crazy must watch HD*, https://youtu.be/6lCUGPixEnk, Consulted 19 May 2016.

[24] A. Kushleyev, D. Mellinger and V. Kumar, *Towards A Swarm of Agile Micro Quadrotors*, GRASP Lab, University of Pennsylvania, in Robotics: Science and Systems, 2012.

[25] R. D'Andrea, *The Astounding Athletic Power of Quadcopters*, TED Talks, https://youtu.be/w2itwFJCgFQ, Consulted 19 May 2016.

[26] Q. Lindsey, D. Mellinger and V. Kumar at the GRASP Lab, University of Pennsylvania, *Construction with Quadrotor Teams*, https://youtu.be/W18Z3UnnS_0, Consulted 19 May 2016.

[27] Q. Lindsey, D. Mellinger and V. Kumar, *Construction of Cubic Structures with Quadrotor Teams*, Mechanical Engineering and Applied Mathematics, University of Pennsylvania, Philadelphia, PA 19104, September 2012.

[28] F. Gramazio, M. Kohler and R. D'Andrea, *Flight Assembled Architecture*, https://youtu.be/JnkMyfQ5YfY, Consulted 19 May 2016.

[29] D. Scaramuzza, M. C. Achtelik, L. Doitsidis, F. Fraundorfer, E. Kosmatopoulos, A. Martinelli, M. W. Achtelik, M. Chli, S. Chatzichristofis, L. Kneip, D. Gurdan, L. Heng, G. H. Lee, S. Lynen, L. Meier, M. Pollefeys, A. Renzaglia, R. Siegwart, J. C. Stumpf, P. Tanskanen, C. Troiani, and Stephan Weiss, *Vision-controlled micro flying robots: from system design to autonomous navigation and mapping in GPS-denied environments*, IEEE Robotics & Automation Magazine 21 (3), 1-10, 20 August 2014.

[30] R. D'Andrea, *Meet the dazzling flying machines of the future*, TED Talks, https://www.ted.com/talks/raffaello_d_andrea_meet_the_dazzling_flying_machines_of_the_future, Consulted 19 May 2016.

[31] H. Liu, H. Darabi, P. Banerjee and J. Liu, *Survey of Wireless Indoor Positioning Techniques and Systems*, IEEE Transactions on systems, Man, And Cybernetics—PART C: Applications and reviews, VOL. 37, NO. 6, November 2007.

[32] A. Ledergerber, M. Hamer and R. D'Andrea, *A Robot Self-Localization System using One-Way Ultra-Wideband Communication*, 2015.

[33] Decawave, 'DWM1000 UWB modules', Available online at `http://www.decawave.com/products/overview`, Consulted 19 May 2016.

[34] Z. Sahinoglu, S. Gezici, and I. Guvenc, *Ultra-wideband Positioning Systems. Theoretical Limits, Ranging Algorithms, and Protocols*, Cambridge University Press, 2008.

[35] D. Fox, W. Burgard, H. Kruppa, and S. Thrun, *A probabilistic approach to collaborative multi-robot localization*, Autonomous Robots, 8:325–344, 2000.

[36] A. Prorok, P. Tome and A. Martinoli, *A Framework for NLOS Ultra-Wideband Ranging in Collaborative Mobile Robot Systems*, International Conference on Indoor Positioning and Indoor Navigation (IPIN), GUIMARAES, PORTUGAL, 21-23 September 2011.

[37] T. J. Broida, S. Chandrashekhar, and R. Chellappa, *Recursive 3-D Motion Estimation from a Monocular Image Sequence*, IEEE Transactions on Aerospace and Electronic Systems Vol. 26. NO. 4, July 1990.

[38] B. Siciliano and O. Khatib, *Springer Handbook of Robotics*, Springer-Verlag New York, Inc., 2007.

[39] M. Montemerlo, S. Thrun, D. Koller and B. Wegbreit, *FastSLAM: A Factored Solution to the Simultaneous Localization and Mapping Problem*, In Proceedings of the AAAI National Conference on Artificial Intelligence, pages 593–598, AAAI, 2002.

[40] C. Nafouki, *Advanced Seminar - Spatial Navigation Algorithms for Autonomous Robotics*, Neuroscientific System Theory, Technische Universität München, 2015.

[41] D. Ribas, P. Ridao, J. D. Tardos, and J. Neira, *Underwater SLAM in a marina environment*, in Intelligent Robots and Systems (IROS) 2007, IEEE/RSJ International Conference on, pp. 1455–1460, 2007.

[42] D. Ribas, P. Ridao and J. Neira, *Underwater SLAM for Structured Environments Using an Imaging Sonar*, Springer Publishing Company, Incorporated, 2010.

[43] J. Levinson, M. Montemerlo, and S. Thrun, *Map-based precision vehicle localization in urban environments*, Proc. Robot.: Sci. & Syst. Conf., Atlanta, GA, June 2007.

[44] T. N. Yap, Jr. and C. R. Shelton, *SLAM in Large Indoor Environments with Low-Cost, Noisy, and Sparse Sonars*, Proceedings of the IEEE International Conference on Robotics and Automation (pp. 1395-1401), 2009.

[45] A. J. Davison, I. D. Reid, N. D. Molton, and O. Stasse, *MonoSLAM: Real-time single camera SLAM*, IEEE Trans. Pattern Anal. Machine Intell., vol. 29, no. 6, 2007.

[46] G. Klein and D. Murray, *Parallel tracking and mapping for small ar workspaces*, In Mixed and Augmented Reality, 2007. ISMAR 2007. 6th IEEE and ACM International Symposium on, pages 225–234. IEEE, 2007.

[47] B. Triggs, P. F. McLauchlan, R. I. Hartley, and A. W. Fitzibbon, *Bundle adjustment - a modern synthesis. In Vision Algorithms: Theory and Practice*, LNCS, pages 298–375. Springer Verlag, 2000.

[48] A. Harmat, M. Trentini, and I. Sharf, *Multi-Camera Tracking and Mapping for Unmanned Aerial Vehicles in Unstructured Environments*, Journal of Intelligent and Robotic Systems, vol. 78, no. 2, pp. 291–317, May 2015.

[49] R. O. Castle, G. Klein, and D. W. Murray, *Video-rate Localization in Multiple Maps for Wearable Augmented Reality*, Proc 12th IEEE International Symposium on Wearable Computers, Pittsburgh PA, 2008.

[50] R. Newcombe, S. Lovegrove and A. Davison, *DTAM: Dense tracking and mapping in real-time*, Intl. Conf. on Computer Vision (ICCV), 2011.

[51] J. Engel and J. Sturm and D. Cremers, *Semi-Dense Visual Odometry for a Monocular Camera*, "IEEE International Conference on Computer Vision (ICCV)", Sydney, Australia, 2013.

[52] J. Engel and T. Schöps and D. Cremers, *LSD-SLAM: Large-Scale Direct Monocular SLAM*, "European Conference on Computer Vision (ECCV)", 2014.

[53] J. Engel et al., *LSD-SLAM: Large-Scale Direct Monocular SLAM (ECCV '14)*, Youtube video, `https://youtu.be/GnuQzP3gty4`, Consulted 24 May 2015.

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

[55] A. B. Chatfield, *Fundamentals of High Accuracy Inertial Navigation*, Progress in astronautics and aeronautics, Volume 174, 1997.

[56] J. A Hesch, D. G. Kottas, S. L. Bowman and S. I. Roumeliotis, *Camera-IMU-based localization: Observability analysis and consistency improvement*, International Journal of Robotics Research archive Volume 33 Issue 1, Pages 182-201, January 2014.

[57] D. W. Strelow, *Motion estimation from image and inertial measurements*, Thesis defended at the School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, November 2004.

[58] A. I. Mourikis and S. I. Roumeliotis, *A Multi-State Constraint Kalman Filter for Vision-aided Inertial Navigation*, IEEE International Conference on Robotics and Automation, April 2007.

[59] A. D. Wu, E. N. Johnson, *Methods for Localization and Mapping Using Vision and Inertial Sensors*, AIAA Guidance, Navigation and Control Conference and Exhibit 18 - 21 August 2008, Honolulu, Hawaii, August 2008.

[60] J. W. Langelaan, *State estimation for autonomous flight in cluttered environments*, Standford University, March 2006.

[61] M. Li, A. I. Mourikis, *Optimization-Based Estimator Design for Vision-Aided Inertial Navigation*, Proceedings of the Robotics: Science and Systems Conference, Sydney, Australia, 9-13 July 2012.

[62] G. Qian, R. Chellappa, and Q. Zheng, *Bayesian Structure from motion using inertial information*, In International Conference on Image Processing, Rochester, New York, USA, 2002.

[63] G. Qian, R. Chellappa, and Q. Zheng, *Robust structure from motion estimation using inertial data*, J. Opt. Soc. Am. A/Vol. 18, No. 12, December 2001.

[64] S. I. Roumeliotis, A. E. Johnson and J. F. Montgomery, *Augmenting Inertial Navigation with Image-Based Motion Estimation*, IEEE International Conference on Robotics and Automation, 2002, pp. 4326-4333 vol.4., 2002.

[65] A. Martinelli, *Vision and IMU Data Fusion: Closed-Form Solutions for Attitude, Speed, Absolute Scale, and Bias Determination*, Transactions on Robotics, vol. 28, no. 1, pp. 44–60, 2012.

[66] A. I. Mourikis, N. Trawny, S. I. Roumeliotis, A. E. Johnson, A. Ansar, and L. Matthies, *Vision-Aided Inertial Navigation for Spacecraft Entry, Descent, and Landing*, IEEE TRANSACTIONS ON ROBOTICS, VOL. 25, NO. 2, April 2009.

[67] D. G. Kottas, J. A. Hesch, S. L. Bowman, and S. I. Roumeliotis, *On the consistency of Vision-aided Inertial Navigation*, Dept. of Computer Science and Engineering, Univ. of Minnesota, Minneapolis, MN 55455, USA, 2013.

[68] Y. Bar-Shalom, X. R. Li, and T. Kirubarajan, *Estimation with Applications to Tracking and Navigation*, John Wiley & Sons, New York, NY, 2001.

[69] V. Grabe, H.H. Bulthoff, D. Scaramuzza, and P.R. Giordano, *Nonlinear Ego-Motion Estimation from Optical Flow for Online Control of a Quadrotor UAV*, International Journal of Robotics Research, 2015.

[70] M.A.R.S. Lab (Multiple Autonomous Robotic Systems), *Navigation, Localization, and Mapping Algorithms Overview*, `http://mars.cs.umn.edu/research/vins_overview.php#CM`, Consulted 22 May 2016.

[71] Xsens, *AHRS Attitude Heading Reference System*, `https://www.xsens.com/tags/ahrs/`, Consulted 23 May 2016.

[72] R. Mahony, T. Hamel, and J.-M. Pflimlin, *Nonlinear complementary filters on the special orthogonal group*, Automatic Control, IEEE Transactions on, 53(5):1203, June 2008.

[73] S. O. H. Madgwick, *An efficient orientation filter for inertial and inertial/magnetic sensor arrays*, Technical report, University of. Bristol University, UK, 2010.

[74] X-io technologies, *Open source IMU and AHRS algorithms*, `http://www.x-io.co.uk/open-source-imu-and-ahrs-algorithms/`, 31 July 2012.

[75] O. Bastelseiten, *IMU Data Fusing: Complementary, Kalman, and Mahony Filter*, `http://www.olliw.eu/2013/imu-data-fusing/`, 16 January 2015.

[76] *Navio2 for developers - AHRS*, `https://docs.emlid.com/navio2/Navio-dev/ahrs/`, Consulted 22 May 2016.

[77] G.H. Lee, M. Achtelik, F. Fraundorfer, M. Pollefeys, and R. Siegwart, *A Benchmarking Tool for MAV Visual Pose Estimation*, International Conference on Control, Automation, Robotics and Vision (ICARCV'10), Singapore, December 2010.

[78] Vicon, "Motion capture systems," Available online at `www.vicon.com`, Consulted 19 May 2016.

[79] S. Lupashin, M. Hehn, M. W. Mueller, A. P. Schoellig, M. Sherback, and R. D'Andrea, *A platform for aerial robotics research and demonstration: The Flying Machine Arena*, Mechatronics, vol. 24, no. 1, pp. 41–54, February 2014.

[80] M.Y.I. Idris, H. Arof, E.M. Tamil, N.M. Noor and Z. Razak, *Review of Feature Detection Techniques for Simultaneous Localization and Mapping and System on Chip Approach*, Information Technology Journal, 8: 250–262, 2009.

[81] J. Li and N. M. Allinson, *A comprehensive review of current local features for computer vision*, Neurocomputing, Volume 71, Issues 10–12, Pages 1771–1787, June 2008.

[82] C. Harris, M. Stephens, *A combined corner and edge detector*, Alvey Vision Conference, pp. 147–151, 1988.

[83] D. G. Lowe, *Object recognition from local scale-invariant features*, Proceedings of the International Conference on Computer Vision. pp. 1150–1157, 1999.

[84] K. Mikolajczyk, C. Schmid, *Scale & affine invariant interest point detectors*, Int. J. Comput. Vis. 60 (1) 63–86. 2004.

[85] E. Rosten and T. Drummond, *Fusing points and lines for high performance tracking*, IEEE International Conference on Computer Vision 2: 1508–1511, 2005.

[86] H. Bay, A. Ess, T. Tuytelaars, and L. Van Goo, *Speeded-Up Robust Features (SURF)*, Comput. Vis. Image Underst. 110, 3 (June 2008), 346-359, June 2008.

[87] P. F. Alcantarilla and A. Bartoli and A. J. Davison, *KAZE Features*, Eur. Conf. on Computer Vision (ECCV), Fiorenze, Italy, 2012.

[88] D. G. Lowe, *Distinctive Image Features from Scale-Invariant Keypoints*, International Journal of Computer Vision 60 (2): 91–110, 2004.

[89] Y. Ke, and R. Sukthankar, *PCA-SIFT: A more distinctive representation for local image descriptors*, Comput. Vision Pattern Recognition, 2: II-506-II-513, 2004.

[90] M. Calonder, V. Lepetit, C. Strecha, and P. Fua, *BRIEF: binary robust independent elementary features*, Proceedings of the 11th European conference on Computer vision: Part IV (ECCV'10), Kostas Daniilidis, Petros Maragos, and Nikos Paragios (Eds.). Springer-Verlag, Berlin, Heidelberg, pages 778–792, 2010.

[91] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski, *ORB: An efficient alternative to SIFT or SURF*, Proceedings of the 2011 International Conference on Computer Vision (ICCV '11), IEEE Computer Society, Washington, DC, USA, 2564-2571, 2011.

[92] Z. Zhang, H. Rebecq, C. Forster, and D. Scaramuzza, *Benefit of Large Field-of-View Cameras for Visual Odometry*, IEEE International Conference on Robotics and Automation (ICRA), Stockholm, 2016.

[93] E. Mueggler, B. Huber, and D. Scaramuzza, *Event-based, 6-DOF Pose Tracking for High-Speed Maneuvers*, IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Chicago, 2014.

[94] *Event-based, 6-DOF Pose Tracking for High-Speed Maneuvers using a Dynamic Vision Sensor*, `https://youtu.be/LauQ6LWTkxM`, Robotics and Perception Group, University of Zurich, 2014, Consulted 20 May 2016.

[95] I. Dryanovski, W. Morris, R. Kaushik, and J. Xiao, *Real-Time Pose Estimation with RGB-D Camera*, IEEE Conference on Multisensor Fusion and Integration for Intelligent Systems (MFI), 2012.

[96] T. Deyle, *Low-Cost Depth Cameras (aka Ranging Cameras or RGB-D Cameras) to Emerge in 2010?*, Hizook: Robotics news for academics and professionals, `http://www.hizook.com/blog/2010/03/28/low-cost-depth-cameras-aka-ranging-cameras-or-rgb-d-cameras-emerge-2010`, 29 March 2010.

[97] C. Kerl, J. Sturm, and D. Cremers, *Dense Visual SLAM for RGB-D Cameras*, In Proc. of the Int. Conf. on Intelligent Robot Systems (IROS), 2013.

[98] F. Endres, J. Hess, N. Engelhard, J. Sturm, D. Cremers, and W. Burgard, *An Evaluation of the RGB-D SLAM System*, Proc. of the IEEE Int. Conf. on Robotics and Automation (ICRA), 2012.

[99] R. A. Newcombe, S. Izadi, O. Hilliges, D. Molyneaux, D. Kim, A. J. Davison, P. Kohli, J. Shotton, S. Hodges, and A. Fitzgibbon, *KinectFusion: Real-Time Dense Surface Mapping and Tracking*, Proceedings of the 10th IEEE International Symposium on Mixed and Augmented Reality, Pages 127-136, IEEE Computer Society Washington, DC, USA, October 2011.

[100] S. Colaner and M. Humrick, *A Third Type Of Processor For VR/AR: Movidius' Myriad 2 VPU*, `http://www.tomshardware.com/news/movidiud-myriad2-vpu-vision-processing-vr,30850.html`, 3 January 2016.

[101] D. Swaraj, and G.L. Madhumati, *FPGA Implementation of SIFT Algorithm Using Xilinx System Generator*, International Journal of Emerging Trends in Electrical and Electronics (IJETEE – ISSN: 2320-9569) Vol. 10, Issue 10, October 2014.

[102] D. Bouris, A. Nikitakis, and I. Papaefstathiou, *Fast and Efficient FPGA-Based Feature Detection Employing the SURF Algorithm*, IEEE Annual International Symposium on Field-Programmable Custom Computing Machines, FCCM 2010, Charlotte, North Carolina, 2-4 May 2010.

[103] M. Faessler, F. Fontana, C. Forster, E. Mueggler, M. Pizzoli, and D. Scaramuzza, *Autonomous, Vision-based Flight and Live Dense 3D Mapping with a Quadrotor Micro Aerial Vehicle*, Journal of Field Robotics (JFR), 2015.

[104] C. Forster, M. Pizzoli, and D. Scaramuzza, *Air-Ground Localization and Map Augmentation Using Monocular Dense Reconstruction*, IEEE International Conference on Intelligent Robots and Systems, 2013.

[105] C. Forster, M. Pizzoli, and D. Scaramuzza, *Appearance-based Active, Monocular, Dense Reconstruction for Micro Aerial Vehicles*, Robotics: Science and Systems (RSS), Berkeley, 2014.

[106] C. Mostegel, A. Wendel, and H. Bischof, *Active Monocular Localization: Towards Autonomous Monocular Exploration for Multirotor MAVs*, IEEE International Conference on Robotics and Automation (ICRA), 2014.

[107] *3D SLAM on multirotor UAV*, `https://youtu.be/dTTrCQuamPk`, SwarmLab, Maastricht University, Consulted 24 May 2016.

[108] *3D Mapping and Navigation using Octrees on a Quadrotor*, `https://youtu.be/7epTZM0yHA4`, Michigan Autonomous Aerial Vehicle Team, University of Michigan, Consulted 24 May 2016.

[109] C. Forster, M. Faessler, F. Fontana, M. Werlberger, and D. Scaramuzza, *Continuous On-Board Monocular-Vision-based Elevation Mapping Applied to Autonomous Landing of Micro Aerial Vehicles*, IEEE International Conference on Robotics and Automation (ICRA), Seattle, 2015.

[110] C. Forster, M. Faessler, F. Fontana, M. Werlberger, and D. Scaramuzza, *Autonomous Quadrotor Landing using Continuous On-Board Monocular-Vision-based Elevation Mapping*, https://youtu.be/phaBKFwfcJ4, Consulted 24 May 2016.

[111] G. Costante, C. Forster, J. Delmerico, P. Valigi, and D. Scaramuzza, *Perception-aware Path Planning*, Conditionally accepted for IEEE Transactions on Robotics, May 2016.

[112] *Plane model segmentation*, http://pointclouds.org/documentation/tutorials/planar_segmentation.php, Consulted 24 May 2016.

[113] G. Mohanarajah, V. Usenko, M. Singh, R. D'Andrea, and M. Waibel, *Cloud-based Collaborative 3D Mapping in Real-Time with Low-Cost Robots*, Automation Science and Engineering, IEEE Transactions on 12 (2), 423-431, 2015.

[114] K. Konolige, D. Fox, B. Limketkai, J. Ko, and B. Stewart, *Map Merging for Distributed Robot Navigation*, IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2003.

[115] T. Cieslewski, S. Lynen, M. Dymczyk, S. Magnenat, and R. Y. Siegwart, *Map API - Scalable Decentralized Map Building for Robots*, ETH-Zürich, 2015.

[116] D. Zou and P. Tan, *Coslam: Collaborative visual slam in dynamic environments*, IEEE Tran. Pattern Analysis and Machine Intelligence, vol. 35, no. 2, pp. 354–366, 2013.

[117] C. Forster, S. Lynen, L. Kneip, and D. Scaramuzza, *Collaborative Monocular SLAM with Multiple Micro Aerial Vehicles*, IEEE International Conference on Intelligent Robots and Systems, 2013.

[118] Flyability SA, *Introducing Gimball, the collision-tolerant drone*, http://www.flyability.com/product/, Consulted 24 May 2016.

[119] Fleye, *Fleye - Your personal Flying Robot*, http://www.gofleye.com/, Consulted 24 May 2016.

[120] Staff, *An Affordable Open Source Quadcopter: EconoCopter*, http://fsmultirotors.com/econocopter.html, 11 October 2014.

[121] AeroQuad, *AeroQuad - The Open Source Multicopter*, http://aeroquad.com/content.php?s=4b4e5f4c18bb45ab6a682132839ab725, Consulted 24 May 2016.

[122] DJI Store, *Matrice 100*, http://store.dji.com/product/matrice-100, Consulted 24 May 2016.

[123] Parrot, *Parrot: AR.Drone2.0*, http://www.parrot.com/fr/produits/ardrone-2/, Consulted 24 May 2016.

[124] Bitcraze, *Crazyflie 2.0*, https://www.bitcraze.io/crazyflie-2/, Consulted 24 May 2016.

[125] *ArduPilot Autopilot Suite*, http://ardupilot.org/ardupilot/index.html, Consulted 25 May 2016.

[126] PX4 Autopilot, *Software Choice for Pixhawk Hardware*, https://pixhawk.org/choice, Consulted 25 May 2016.

[127] Autoquad, *Autonomous multi rotor vehicle controller*, http://autoquad.org/, Consulted 25 May 2016.

[128] Paparazzi UAV - The free autopilot, *Autopilots*, http://wiki.paparazziuav.org/wiki/Autopilots, Consulted 28 May 2016.

[129] PX4 Autopilot, *Pixhawk Autopilot*, `https://pixhawk.org/modules/pixhawk`, Consulted 25 May 2016.

[130] *ArduPilot Mega, the Open Source Autopilot*, `http://www.ardupilot.co.uk/`, Consulted 25 May 2016.

[131] *AeroQuad32 Flight Control Board v2*, `http://aeroquad.com/showwiki.php?title=AeroQuad32-Flight-Control-Board-v2`, Consulted 25 May 2016.

[132] Emlid, *NAVIO2 - Linux autopilot on Raspberry Pi*, `https://emlid.com/`, Consulted 25 May 2016.

[133] Erlerobotics, *Erle-Brain 2 - All-in-one Linux brain for robots and drones*, `http://erlerobotics.com/blog/erle-brain-2/`, Consulted 25 May 2016.

[134] Alex (Unmanned Tech), *DroneTest - Which RaspberryPi Autopilot - Navio2 vs ERLE Brain2?*, `http://www.dronetrest.com/t/which-raspberrypi-autopilot-navio2-vs-erle-brain2/1545`, Consulted 25 May 2016.

[135] C. Bills, J. Chen and A. Saxena, *Autonomous MAV Flight in Indoor Environments using Single Image Perspective Cues*, International Conference on Robotics and Automation (ICRA), 2011.

[136] M. Achtelik, A. Bachrach, R. He, S. Prentice, and N. Roy, *Stereo vision and laser odometry for autonomous helicopters in gps-denied indoor environments*, SPIE Unmanned Systems Technology XI, 2009.

[137] C. Troiani, A. Martinelli, C. Laugier, D. Scaramuzza, *Low Computational-Complexity Algorithms for Vision-aided Inertial Navigation of Micro Aerial Vehicles*, Robotics and Autonomous Systems Journal, Vol. 69, pp. 80-97, 2015.

[138] A. Bachrach, R. He and N. Roy, *Autonomous flight in unknown indoor environments*, International Journal of Micro Air Vehicles, pages 217–228, 2009.

[139] S. Shen, N. Michael and V. Kumar, *Autonomous Multi-Floor Indoor Navigation with a Computationally Constrained MAV*, Robotics and automation (ICRA), 2011, IEEE international conference, pages 2–25. IEEE, 2011.

[140] S. Shen, N. Michael and V. Kumar, *Autonomous Aerial Navigation in Confined Indoor Environments*, Youtube video, `https://youtu.be/IMSozUpFFkU`, Consulted 25 May 2016.

[141] J. Engel, J. Sturm and D. Cremers, *Camera-Based Navigation of a Low-Cost Quadrocopter*, Proc. of the International Conference on Intelligent Robot Systems (IROS), 2012.

[142] J. Engel, J. Sturm and D. Cremers, *Accurate Figure Flying with a Quadrocopter Using Onboard Visual and Inertial Sensing*, Proc. of the Workshop on Visual Control of Mobile Robots (ViCoMoR) at the IEEE/RJS International Conference on Intelligent Robot Systems (IROS), 2012.

[143] Y. Fan and M. Aramrattana, *Exploration and Mapping of Warehouses Using Quadrotor Helicopters*, Master Thesis, Halmstad, November 2013.

[144] N. Dijkshoorn, Supervised by Dr. Arnoud Visser, *Simultaneous localization and mapping with the AR.Drone*, Universiteit van Amsterdam, 2013.

[145] J. Nyman, Supervised by Prof. Dr. Ir. Benjamin Schrauwen, *Designing an Autonomous Exploration Architecture for an Indoor Quadcopter*, Ghent University, 2012.

[146] B. Steder, G. Grisetti, C. Stachniss and W. Burgard, *Visual SLAM for flying vehicles*, Robotics, IEEE Transactions on, volume 24(5), pp. 1088–1093, 2008.

[147] Parrot AR.Dronewebsite, http://www.parrot.com/fr/produits/ardrone-2/, Consulted on June 2016.

[148] P. Bristeau, F. Callou, D. Vissière and N. Petit, *The Navigation and Control technology inside the AR. Drone micro UAV*, in "World Congress", volume 18, pp. 1477–1484, 2011.

[149] *What is ROS?*, http://www.ros.org/, ROS official website, Consulted 2 August 2016.

[150] J. Kay, *Introduction to Real-time Systems*, http://design.ros2.org/articles/realtime_background.html, Consulted 22 June 2015.

[151] D. Thomas, *ROS 2.0 Roadmap*, https://github.com/ros2/ros2/wiki/Roadmap, Consulted 22 June 2015.

[152] *Orocos RTT and ROS integrated*, http://www.willowgarage.com/blog/2009/06/10/orocos-rtt-and-ros-integrated, Official Willow Garage website, Consulted 22 June 2015.

[153] J. Gérardy and F. Schiltz, *Suivi d'une cible mobile par des drones autonomes*, Master Thesis, UCL, 2016.

[154] J. J. Engel, *Difference to keypoint-based methods*, http://vision.in.tum.de/research/vslam/lsdslam, TUM Computer Vision Group, 15 July 2016.

[155] OpenCV 2.4.13.0 documentation, *OpenFABMAP*, http://docs.opencv.org/2.4/modules/contrib/doc/openfabmap.html, Consulted 3 August 2016.

[156] R. Szeliski, *Computer Vision: Algorithms and Applications (1st ed.)*, Springer-Verlag New York, Inc., New York, USA, 2010.

[157] J. Sturm, *Lecture Notes: Visual Navigation for Flying Robots*, Technische Universität München, Germany, 2013.

[158] Camera calibration With OpenCV, http://docs.opencv.org/2.4/doc/tutorials/calib3d/camera_calibration/camera_calibration.html, OpenCV 2.4.13.0 documentation, 3 Augustus 2016.

[159] M. Brown and D. G. Lowe, *Automatic Panoramic Image Stitching using Invariant Features*, University of British Columbia, Canada, 2007.

[160] OpenCV API Reference, *Feature Detection and Description: SURF*, http://docs.opencv.org/2.4/modules/nonfree/doc/feature_detection.html#{}surf-surf, Consulted on June 2016.

[161] OpenCV API Reference, *Motion Analysis and Object Tracking: optical flow using the iterative Lucas-Kanade method with pyramids (calcOpticalFlowPyrLK)*, http://docs.opencv.org/2.4/modules/video/doc/motion_analysis_and_object_tracking.html#{}calcopticalflowpyrlk, Consulted on June 2016.

[162] M. Hwangbo, J.-S. Kim and T. Kanade, *IMU-KLT: IMU-Aided KLT Feature Tracking*, http://www.cs.cmu.edu/~myung/IMU_KLT/index.html, Consulted on June 2016.

[163] J.-Y. Bouguet, *Pyramidal implementation of the Lucas Kanade feature tracker*, Intel Corporation, Microprocessor Research Labs, 2000.

[164] OpenCV API Reference, *Camera Calibration and 3D Reconstruction: findHomography*, http://docs.opencv.org/2.4/modules/calib3d/doc/camera_calibration_and_3d_reconstruction.html#findhomography, Consulted on June 2016.

[165] Wikipedia, *Levenberg-Marquardt Algorithm*, https://en.wikipedia.org/wiki/Levenberg%E2%80%93Marquardt_algorithm, Consulted on June 2016.

[166] R. I. Hartley and A. Zisserman, *Multiple View Geometry in Computer Vision*, Cambridge University Press, 2004.

[167] N. Livet, *The P3P (Perspective-Three-Point) Principle*, http://iplimage.com/blog/p3p-perspective-point-overview/, 2012, Consulted on June 2016.

[168] M. Zuliani, *RANSAC for Dummies*, University of California Santa Barbara, 2014.

[169] O. Enqvist, *Correspondence Problems In Geometric Vision*, Lund University, Sweden, 2009.

[170] V. Lepetit and F. Moreno-Noguer and P. Fua, *EPnP: An Accurate O(n) Solution to the PnP Problem*, International Journal Computer Vision, 2009.

[171] H. Alismail, B. Browing and M. B. Dias, *Evaluating Pose Estimation Methods for Stereo Visual Odometry on Robots*, Published In proceedings of the 11th International Conference on Intelligent Autonomous Systems (IAS-11), 2010.

[172] OpenCV API Reference, *Camera Calibration and 3D Reconstruction: solvePnPRansac*, http://docs.opencv.org/2.4/modules/calib3d/doc/camera_calibration_and_3d_reconstruction.html#solvepnpransac, Consulted on June 2016.

[173] K. Kanatani, Y. Sugaya and H. Niitsuma, *Triangulation from Two Views Revisited: Hartley-Sturm vs. Optimal Correction*, Okayama University and Toyohashi University of Technology, Japan, 2008.

[174] N. Beghin and T. Martin, *Localisation et Cartographie Simultanées (SLAM) par un Quadricoptère AR.Drone Autonome*, Master Thesis, UCL, 2013.

[175] B. De Radigues and F. Van Hijfte, *Simultaneous localisation and mapping (SLAM) and investigation of multi-drone capacities*, Master Thesis, UCL, 2015.

Appendices

Appendix A

Code execution timings

For all measurements, the strictly necessary code has been launched at a time to prevent interferences between code modules. Indeed, ROS being a non real-time kernel, the thread priority attribution by the ROS master (kernel) is non deterministic, leading to uninterpretable timing measurements.

		Detailed execution time [ms]	Total execution time [ms]	Average rate [Hz]	Std. dev. of total execution time
Computer Vision	Optical flow tracking	5.4	284.3 no o.f./ 14.8 with o.f.	3.8 no o.f./ 11.9 with o.f.	8.7 no o.f./ 4.2 with o.f.
	Keypoint detection	31.9			
	Keypoint description	246.1			
	Target detection and removing	6.4			
	Publish processed image	0.3			

Table A.1: Timing of the computer vision node and sub modules of the code

All the measurements of Table A.1 have been made using SIFT descriptors and a SURF detector. The mention "no o.f." refers to the situation where the code only uses keypoint detection and description to solve the matching problem between frames. At the opposite, "with o.f." refers to the situation where only optical flow is sufficient to match frames. All measures are an average on 50 measurements at rest (no drone motion with respect to the scene). When the drone gradually begins to move, the total execution time must be expected to tend towards the "no o.f." situation.

In Table A.2, the standard deviation of the total execution time of the node is huge. This is because the execution depends on whether the current reference keyframe is not sufficient anymore to track the displacement. In this case, before deciding to create a new one based on the current camera frame, a search is done amongst older keyframes (see Subsection 4.5.2). This highly increases the execution time of the mapping node at some times, leading to a high

		Detailed execution time [ms]	Total execution time [ms]	Average rate [Hz]	Std. dev. of total execution time
Mapping	Scene to camera projection	0.3	60 static / 118 moving	11 static / 6 moving	15.5 static / 246.5 moving
	Search among old keyframes	497			
	Perspective n Point (PnP)	30.2			
	Frame-keyframe matching	1.7			
	3D Viewer	12			
	Publish visual pose estimation	0.13			
	Publish target position	0.005			

Table A.2: Timing of the mapping node and sub modules of the code

standard deviation. For this reason, when the drone is moving, the execution time is increased due to the process of choosing a new reference keyframe and adding new keyframes to the map.

Second, note that for the 3D viewer, a 10ms delay is fixed to enable the handling of the map by the user (change of viewpoint and zoom). Thus, this delay is purely added to the total execution time.

	Average Rate [Hz]	Fixed rate [Hz]
Computer vision	5,998	6
2D Viewer	N/A*	12
Mapping	2.249	2.3
Pose estimation	20	20
Controller	6613	∞
Path planning	19.9	20
Strategy	20	20
Swarm initialisation	0.2	0.2

Table A.3: Timing of all ROS nodes

*N/A here, for there is no direct way to measure this rate with ROS, since this node does not publish any message

In Table A.3, the drone was put in a situation of typical flight of 1 minute over a well textured environment. All timings were monitored at once, unlike in the above situations.

As a conclusion, one can observe that the variance on all measurements is pretty large. This is because of the non deterministic characteristic of the ROS system, which is, as already mentioned, not real-time. Second, since the controller has no fixed rate, it executes the most

often possible, leading to huge variance of the execution rate. On 16 measurements of 1000 executions, the controller executed at minimum 3864 Hz and maximum value of 10148 Hz, leading to a standard deviation of 0.00128s in total. This point was communicated to the group responsible for the controller design at the time of writing this analysis.

Moreover, for both computer vision and mapping nodes, the execution time depends on whether the drone is moving or not. This implies that the controller should not be too aggressive. Indeed, the more aggressive the flight, the more computations must be done on the received images, thus decreasing the available resources for the controller execution. So, the controller executes less often when the drone moves faster, finally leading to possible stability issues.

Appendix B

Drift on angle and position measurements

Pitch and roll (rotations around the y and x axis)

The on-board angle sensors are physical devices (gyroscopes) that actually measure an angular velocity. This means that the output signal must be integrated in the *Parrot* software in order to obtain an angle information. The problem is that the angular velocity systematic error is also integrated. The summation of the error over time introduces a drift in the angle estimation. Since we have no access to the initial angular velocity measure (to fit into a Kalman filter for example), we have to rely on the *Parrot* drifted angular pose estimation.

It has been observed that when the drone is lying flat on the ground, the angular drift in the roll and pitch directions is about zero. But when the drone is tilted, the angle estimation begins to drift over time. When the drone is put back in its lying position, the drift stops but the error due to the drift remains.

In the following experience, we tried to verify if a positive angle introduces a drift of the same amplitude but opposite sign as the drift obtained with a negative angle. If that is the case, that assumption can be done that the negative drift during the negative angle could globally compensate the positive drift during positive angles. This would reduce the overall effect of the drift over time, since the drone statistically remains flat during a full flight.

The experience procedure can be described by the following steps:

1. Put the drone on flat ground and send a "flat trim" command to the drone.
2. Measure its angular position estimate in the roll direction.
3. Put the drone at an angle of 18° in the roll direction.
4. Wait a minute.
5. Measure the angular position estimate in the roll direction.
6. Put the drone on flat ground and wait until sensor reading stabilization.
7. Measure the angular position estimate in the roll direction.
8. Put the drone at an angle of -18° in the roll direction.
9. Wait a minute.
10. Measure the angular position estimate in the roll direction.
11. Put the drone on flat ground and wait until the sensor reading stabilization.
12. Measure the angular position estimate in the roll direction.

The results are given in the table below (see Table B.1).

step in procedure	roll estimation $[^\circ]$	roll measured $[^\circ]$
2	0.3438	0
5	21.4916	18
7	2.5210	0
10	-19.2055	-18
12	1.4324	0

Table B.1: Measure of the roll estimation $[^\circ]$ for different angles

The conclusion of the experience is that we cannot make the assumption that the drift will be auto-corrected during a normal flight. This implies that we have to rely on a measure that drifts over time. Fortunately, it has been observed that this phenomenon completely vanishes when the drone is controlled with the *Parrot* internal controller. Thus, the closed-source software must contain a sensor fusion that takes into account the commands sent to the rotors to set up a prediction for motion estimation.

Consequently, this drift has no influence on the flight stability. But for development purposes, this leads to a huge handicap. Indeed, as it was explained before, some sensor readings are not available when the drone is not in flight. Now, we observe that even in flight, we cannot manipulate the drone for any test, since this distorts the sensor measurements. In other words, we cannot use the on-board sensors to compare the results of our pose-estimation and the actual known pose when the drone is kept at hand at a fix known position. This problem could be solved by a motion capture system providing a ground-truth measurement, as explained in Subsection 2.2.5. To be able to quantify the performance of our pose estimation methods, we had to find several workarounds (see Chapter 5).

Yaw (rotations around the z axis)

In opposition to the roll and pitch sensor measurements, the yaw component is not corrected during flight. Drifts of about 30 $[\mathrm{deg}/s]$ have been observed. Unfortunately, there is no way to compensate for that drift since it is totally non-deterministic. Indeed, it depends on the execution of the code.

x and y position

Here too, the position is obtained from integrating a velocity information obtained with an optical flow sensor. This yields the same drift problem as for the orientation estimation. However, the raw sensor readings are now available, and the integration is done at our side of the code, in the pose estimation node. Thus, we have better control over what is done on this measurement. Unfortunately, there is no way to compensate for that drift since it is totally non-deterministic. Values of $1cm/s$ and $6cm/s$ were observed in the x and y directions, respectively. The effect of this drift on our pose fusion is shown in the Section 4.6 and Chapter 5.

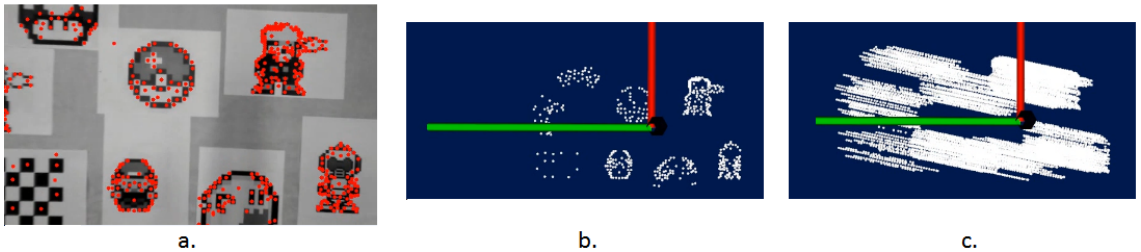


Figure B.1: Effect of the drift for the keypoints mapping if only the integration of on-board measurement is used. The drone is kept static, no SLAM is used (each new image keypoints are mapped). (a) keypoints superimposed to what the drone sees (b) initial keyframe put in the map (c) after a few seconds, all added keyframe superimpose in the map in an odd fashion

In Figure B.1, a very insightful illustration is given, showing the effect of relying on a drifted pose estimation to map feature points. This confirms the need for a corrected pose estimation, based on visual egomotion. So, it is indeed this corrected pose that is used for mapping, in order to ensure coherence in the map, as explained in Section 4.6.

Altitude (z position)

As for the roll and pitch measurements, the altitude is fused inside the *Parrot* software. It is initially provided by an ultrasonic sensor, which directly provides a distance-to-flat-ground information. But when an important slope of altitude is detected without the IMU detecting any vertical acceleration, this software concludes that no movement is achieved. Moreover, when keeping the drone at hand during flight, altitude measurements reveal to be very erratic, which must be due to internal motion prediction models based on motor commands, as for the roll and pitch information. These are additional reasons for which we must be very careful when imagining experiences. This reduces the development opportunities since we have no insight and no real control over the sensor behaviour.

Appendix C

Measurement of the precision of the perspective projection

The first step towards 3D mapping is to test if the algorithm is able to correctly map in 2D. Indeed, many modules of code are necessary to implement 3D mapping. Thus, a good way to test all modules individually is to make assumptions on the output of other modules in an ideal, restricted case.

The module that will be tested here corresponds to the "workspace transformation" of Subsection 4.5.3. Its function is to perform the "2D projection" from the image plane to the flat ground plane.

It is worthwile noting that this module had already been implemented in the first master thesis on the subject (BEGHIN and MARTIN, 2013 [174]), and corrected for more accuracy in the last master thesis (DE RADIGUES and VAN HIJFTE, 2015 [175]). However, our attempts to import this module "as is" showed to be unsuccessful. The reasons are multiple:

- Their mathematical expressions assumed implicitly an altitude of h = 1 m, which meant that no flight was possible at any other altitude.
- Their mathematical expressions seemed to differ from their bibliographic reference in the matter (’`wikipedia`’), which did not explain very well some terms of the expressions.
- Some other bugs related to their implementation prevented us from purely reusing it, even after several corrections from our side.

For these reasons, a new implementation over which we have better control was imagined. Indeed, the mathematical background for the exact solution of the geometrical problem has a simpler formulation, thus easier to verify.

In order to test the accuracy of the mapping of a textured flat ground, the following experiment has been conceived.

C.1 Description of the experiment

To validate the accuracy of the projection, a predefined picture was used. It will be referred to as the target. Its real dimensions are known. Then, this target is described with visual features. Finally, the distances between these features are computed both on the known target picture and on the map. The absolute and relative distance errors are our performance measures.

The procedure is the following:

1. The drone is fixed at 80 cm of altitude with a known orientation.
2. The target is placed on the ground in the field of view.
3. Keypoint detection and features description is performed.
4. The correspondences with the known target picture are computed.
5. The projection is computed with the known orientation of the quadcopter and keypoints are added to the map.
6. The Euclidean distance between all pairs in the set of keypoints is computed on the map and on the target picture.
7. The absolute and relative error are then computed for each pair.
8. The average is finally computed.

C.2 Results

The results are graphically shown on Figure C.1 and Figure C.2 for the drone lying flat and then inclined at 15° around the x-axis, respectively.

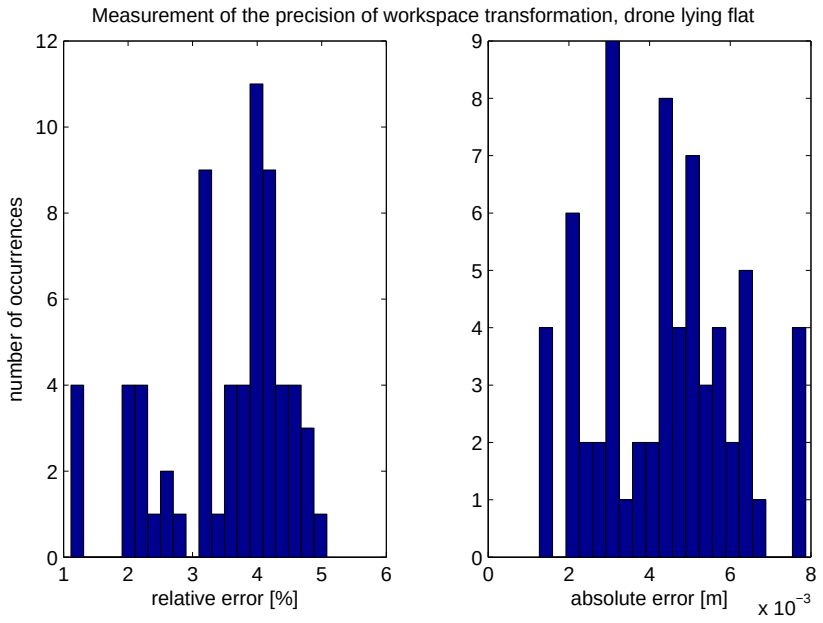


Figure C.1: Measurement of the precision of the perspective projection with the drone lying flat

For Figure C.1, which corresponds to the situation of a flat drone, a dataset of 66 compared distances was used, while the second test (inclined drone) led to a dataset of 325 compared distances (on Figure C.2).

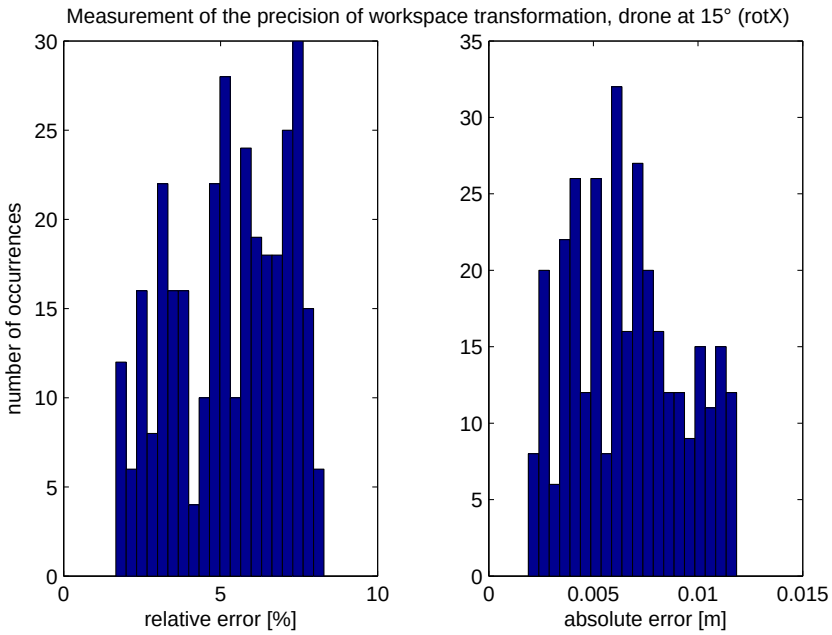


Figure C.2: Measurement of the precision of the perspective projection with the drone inclined at 15°

The results are also synthetized in Table C.1, where they have also been compared to the last years results. However, since the last years code manifestly induced abnormally huge errors in the map during our tests (about 32% of distance mismatches), it was decided to present the mean errors of the results given in their report [175], at table 4.1 and 4.2. Unfortunately, this represents a much smaller dataset (8 points, in both cases). These results were also obtained for slightly different angles: 11° around the x-axis combined with 5° around the pitch axis. Finally, the distances that were measured last year were all obtained from an arbitrary reference point in the image plane, by contrast of our technique where all distances between all points are compared, which is more correct. Indeed, this avoids the possibility of taking an outlier as reference point.

		last year	this year
drone flat	mean relative error [%]	3.5	3.49
	mean absolute error [mm]	6.775	4.4
drone inclined	mean relative error [%]	8.5	5.27
	mean absolute error [mm]	11.612	6.6

Table C.1: Summary of experimental results and comparison with last years results for the "workspace transformation"

All and all, even better results were obtained, confirming that there were probably still some problems with the last years implementation.

Appendix D

Improvements to the development environment

In this appendix are described some improvements that are not directly linked with the principal objectives of our master thesis but related to our contribution to the development environment.

`autoconfarparrot`

The work of Nicolas Rowier during the last academic year was to provide the tools to use several Parrot AR.Drone at the same time. To achieve this, a `bash` script named `autoconfarparrot` was written in order to configure the wireless connection at each reboot of a quadcopter. This script performs the task for which it is designed but a lot of complementary repetitive manual actions are needed to complete the configuration of the PC. Moreover, based on our experience of thousands of drones connections, we determined several situations which are prone to lead to a failure of the script. Then, we added some functionalities to the original script:

- Check situations that lead to a failure (and display an error message).
- Automation of the PC configuration.
- Management of multiple drone connections (only one run of the script is needed to connect multiple drones).
- Avoid to leave unnecessary traces
  (example: remove connections in network manager history to avoid strange errors when the list is full).
- Check if the router and the drones are reachable after the reconfiguration.
- Display simple and clear success and error messages (with colors to improve readability, see Figure D.1).

```
ERROR: Network Manager is disabled
use this command line:  nmcli nm enable true

...

» Drone ardrone2_009941 successfully configured with new ip 192.168.1.5
```

Figure D.1: Examples of `autoconfarparrot` messages

Graphical interfaces

Some useful interfaces were set up in order to ease the interpretation of several pieces of code.

First, the 2D realtime viewer displays the video stream. The keypoints detected by the `computer_vision` node are overprinted, and when the target is detected, a green rectangle is drawn.

The 3D realtime viewer displays the keypoints in the map and allows to visualize and navigate into it in a 3D fashion.

Finally, we implemented a plug-in for the ROS `rqt` node. This one displays several buttons like 'emergency' or 'land' for taking back control in situations during the testing phase of the development. It is also designed to manage several drones using the message from `swarm_initialisation` node.

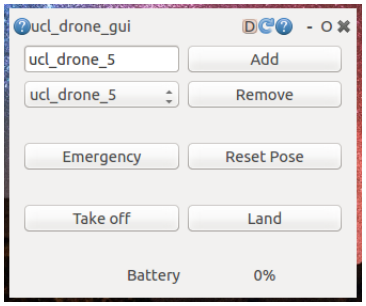


Figure D.2: ROS `rqt` plugin

Documentation

ROS comes with the package `rosdoc_lite` which is a wrapper for the Doxygen tool. Doxygen is an auto-generating API documentation. This enables rich comments in the code to be parsed and compiled into a single well formatted output. We spent considerable time to build this document in order to ease the understanding of our pieces of software for future students.

Moreover, a lot of information messages are displayed at the runtime to indicate the state of the software. Debug messages can be activated at the compilation for further code analysis. These are enabled by `ROS_INFO` and `ROS_DEBUG` commands.

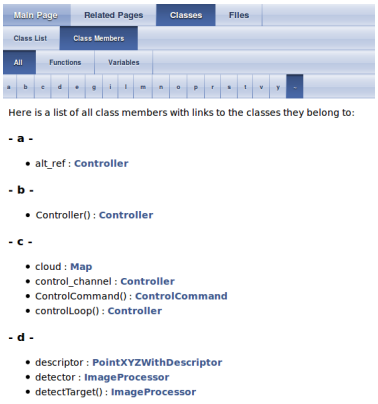


Figure D.3: `rosdoc` output

Appendix E

Synthetic table of possible upgrades for the bottom camera

	Option 1	Option 2	Option 3	Option 4
	Camera Axis	Camera « type GoPro »	Via carte PixHawk	2ème carte AR.Drone
Résolution	HDTV 720p/1MP quality	[640x480 : 4K]	752*480	720p
Modèle	AXIS M1004-W	Spycam, Arlo, Relee RL102, HMNC500, SJCAM, Gopro	Aptina MT9V032	Caméra Frontale AR.Drone 2.0
Matériel Requis	Code ROS disponible sur GitHub et un système d'attache au drone.	Système d'attache au drone.	PixHawk et « Odroïde C1 »	- Carte mère pour AR.Drone 2.0 - Structure inférieure AR.Drone 2.0
Complexité	- Poids et taille assez conséquents (110g + batterie + système d'attache). - Code à adapter par rapport à la nouvelle caméra.	- Hacker la caméra pour la rendre client et non hôte du réseau. - Savoir récupérer les données envoyées sous ROS. - Bande passante inconnue. - Poids peut être conséquent en fonction de la caméra.	- Très peu de documentation. - Fonctionnement assez flou.	- Aucun dispositif WI-FI à ajouter. - Dimensions de la carte entrent dans l'espace intérieur de la carène. - Alimentation identique à celle présente (un « T » suffit). - Interfaçage ROS déjà fait.
Prix	+- 225 €	De 70$ à 300€	+- 250 €	148,99 €

Table E.1: Synthetic table of possible upgrades for the bottom camera

Rue Archimède, 1 bte L6.11.01, 1348 Louvain-la-Neuve **www.uclouvain.be/epl**