POLITECNICO DI MILANO
Corso di Laurea Magistrale in Ingegneria Informatica
Dipartimento di Elettronica, Informazione e Bioingegneria

Monocular Autonomous Exploration in Unknown Environment with Low-Cost Quadrotor

AI & R Lab
Laboratorio di Intelligenza Artificiale
e Robotica del Politecnico di Milano

Relatore: Prof. Andrea Bonarini

Tesi di Laurea di:
Daniele Iasella, matricola 755506
Stefano Fossati, matricola 755149

Anno Accademico 2012-2013

Abstract

In this thesis, we develop a system that enables a low-cost quadrotor to localize and to explore autonomously a previously unknown and GPS-denied environments without requiring artificial markers or external sensors.

We develop a unique exploration strategy for a mobile robot moving in a three-dimensional space through a monocular, keyframe-based simultaneous localization and mapping (SLAM) system and an Extended Kalman Filter, to fuse and synchronize all available sensors measurements. The quadrotor using the exploration algorithm can move around autonomously in an unknown room of a limited size by exploiting a two-dimensional map to avoid obstacles, and to plan its next moves.

The novelty of the presented system is the use of only a single camera-based quadrotor, without any laser rangefinder, along without a target-driven exploration strategy, rather seeking new unseen areas, in order to reconstruct an obstacle map of the drone surroundings. The exploration strategy takes into account the limitations of the low-cost device.

We developed our approach on a Parrot AR.Drone, demonstrating what can be achieved with modern, low-cost, and commercially available hardware platforms as tool for robotics research. In our approach, information processing and exploration strategy evaluation are performed on a ground station, which is connected to the drone via wireless LAN.

The results for various environment situations are discussed demonstrating the accuracy of the system in an indoor environment exploration.

Sommario

In questa tesi è sviluppato un sistema che permette ad un elicottero quadrirotore a basso costo di localizzarsi e di esplorare autonomamente un ambiente totalmente sconosciuto a priori ove non è possibile utilizzare sensori GPS e senza l'utilizzo di marker artificiali e sensori esterni.

È sviluppata una nuova e unica strategia di esplorazione che permette ad un robot mobile di muoversi in uno spazio tridimensionale sfruttando un sistema SLAM (Simultaneos Localization And Mapping) monoculare basato su keyframe e un filtro di Kalman esteso che permette di fondere e sincronizzare tutte le rilevazioni provenienti dai sensori disponibili. Inoltre, il quadricottero, utilizzando l'algoritmo di esplorazione studiato, potrà muoversi in una ambiente di dimensione limitata fruttando una ricostruzione bidimensionale evitando ostacoli e pianificando di volta in volta il prossimo movimento.

La novità del sistema presentato riguarda la possibilità di usare un quadricottero dotato di una unica videocamera, sprovvisto di sensori di prossimità o telemetro laser, e senza una strategia di esplorazione basata su target prestabiliti, per ricostruire una mappa degli ostacoli presenti attorno al drone. Inoltre, la strategia proposta tiene in considerazione le limitazioni del drone a basso costo e del sistema di SLAM scelto.

Il sistema proposto è stato sviluppato basandosi su un AR.Drone della Parrot, dimostrando come possa essere possibile raggiungere l'obiettivo oggetto di questa tesi attraverso una piattaforma hardware economica, moderna e disponibile in commercio usata come strumento per la ricerca robotica. L'elaborazione dei dati ricevuti dal quadrirotore e il calcolo della strategia di esplorazione si svolge su un computer connesso via wireless LAN al drone.

Si riportano infine i risultati di test effettuati in vari ambienti dimostrando l'accuratezza del sistema nell'esplorazione di un ambiente al chiuso.

Contents

Abstract **I**

Sommario **II**

Acknowledgments **V**

1 Introduction **1**
1.1 Thesis Overview 4

2 State of art **5**
2.1 Example of applications 6
2.2 Methodologies and algorithms 9
2.2.1 Three-dimensional reconstruction 10
2.2.2 Pose estimation 15
2.2.3 Exploration 16

3 Problem analysis and proposed solution **18**
3.1 3D reconstruction and pose estimation 19
3.1.1 PTAM 19
3.2 Pose estimation 27
3.2.1 Extended Kalman Filter 30
3.3 Autonomous exploration 32
3.3.1 Knowledge extraction 34
3.3.2 Knowledge increase methods 40

4 System architecture **46**
4.1 Hardware component 46
4.1.1 AR.drone 46
4.2 Software 51
4.2.1 Communication services between AR.Drone and devices 53

5 Implementation and evaluation **58**
5.1 SLAM manager 58
5.1.1 tum_ardrone package modifications 59
5.2 ROS messaging 61
5.2.1 Message standardization 61
5.2.2 Occupancy Grid 63
5.2.3 Visited Grid 67
5.2.4 Scale adaptation 68
5.3 Exploration manager 69
5.3.1 Chosen strategy 72
5.3.2 Custom message 82
5.4 Graphical User Interface 83
5.5 Evaluation 86

6 Conclusions and future research **92**
6.1 Conclusions 92
6.2 Future works 93

Bibliography **95**

Acknowledgments

To our supervisor, Prof. Andrea Bonarini, for all the patience, teachings and time devoted to help us developing this thesis.

To AIRlab guys for all the advices, the helps (in particular during the video editing) and for all funny moments gone.

Chapter 1

Introduction

"We shall not cease from exploration,
and the end of all our exploring
will be to arrive where we started
and know the place for the first time."

T. S. Eliot

Robotics is an engineering discipline that designs and develops robots and methods useful to make them performing specific tasks. A major goal of robotics is to develop mobile robots that can operate autonomously in real world situations. As described by Murphy [1], an intelligent robot is a mechanical creature that can function autonomously. These intelligent robots can be used for a wide range of applications; for example: cleaning, inspection, transportation tasks, medical and construction assistance or also operating in dangerous environments without risking human lives.

An important prerequisite of an intelligent autonomous robot is the ability to know where it is located. To estimate the position of a robot, methods that rely on typically adopted sensors (e.g., velocity, acceleration or GPS sensors) have been developed. More sophisticated methods, which depend only on visual or laser rangefinder sensors, allow to estimate the position of the robot and to simultaneously extract a description of the surrounding environment without the need of an a priori model.

In specific applications, such as victims research and rescue, exploration of the environment has to be performed trying to cover the maximum possible area.

The aim of this thesis is to develop an autonomous exploration system based on a robot with navigation capabilities (Parrot AR.Drone quadro-

tor), with the only help of the onboard sensors (i.e. Inertial Measurement Unit an ultrasound altimeter and a single camera) and a computer. Already developed autonomous systems for navigation and exploration (e.g., those used in Robocup Rescue League, DARPA, and by TUM-MIT quadrotor) are mainly based on expensive hardware. The mapping of the environment is always performed by a laser rangefinder. In our system, we have opted for a visual mapping and localization system (using only low cost hardware) to demonstrate that results can be compared to those used on the aforementioned approaches.

The localization of a robot can be performed by adopting sensors like GPS, which can be seen as a black box that gives a direct observation of the position, or making an indirect estimate through the use of velocity and accelerometer sensors. However, GPS localization is inaccurate (typical error is estimated around 2m) and it is impossible in indoor environment, and velocity and acceleration sensors suffer from drifts and the unavoidable need of approximation to obtain information about position.

More accurate approaches are obtained merging sensors measurements by computing an estimate of the localization and filtering sensors readings. The most widely adopted method in robotics applications is the Extended Kalman Filter [2]. This method usually uses sensors readings (such as accelerometer and gyroscope data) to estimate the current state of the system (i.e., robot pose) starting from the initial one. This method could be fused with a direct estimate of the state to reduce drifts over time (due to sensor measurement errors).

In case where the state is represented by the robot position, the direct estimate could be observed with localization methods. Accurate methods, that can be also used in GPS-denied environments, are based on image analysis algorithms with multi-camera configuration. An example of this kind of system is represented by the OptiTrack [3] that is usually used to reconstruct movements of tracked objects in a virtual three-dimensional space by movie and videogame industries. Moreover, it is considered as a very reliable system and it used by researchers as ground truth estimate of three-dimensional position to test localization algorithms. This multi-camera systems require installation and a complex configuration in order to work and, moreover, they are really expensive. Out of the box approaches that use a single sensor camera have been studied by many starting from the seminal work of A. J. Davison [4]. In addition, these approaches can also to

map the environment and for this reason they are called Visual Simultaneous Localization And Mapping (SLAM) algorithms.

To achieve the aim of this thesis, we use an existent system developed by researchers at Technische Universitat Munchen (TUM) [5], based on a low-cost quadrotor and a ground-based computer to perform localization and mapping of the drone surroundings. To perform its task TUM system uses three main components: a monocular keyframe-based SLAM system (PTAM) [6] for pose estimation, an Extended Kalman Filter, to fuse all sensor measurements to give a better estimate of the quadrotor position, and a PID controller to control the position and orientation of the drone. We choose this system as localization and mapping core taking into account, firstly, the good quality of its drone pose estimate, its environment description accuracy and the possibility to run in real-time. Moreover, the package is developed to work as a native Robot Operating System (ROS) package allowing fast development and transparent interaction with other packages. However, the original package performs only basic autonomous navigation (point to point without obstacle avoidance). So, we have modified it to match our exploration goals.

To make the low-cost quadrotor able to explore, we have studied the most used method, such as Frontier-based method [7] that focuses on approaching to frontiers of the so far achieved map to increase its knowledge. However, this method, is not designed to work with limited field of view mapping sensor. The View-improvement method [8] describes a possible solution to increase the knowledge of the environment in system with limited field of view sensor. This method explains how the robot has to orient its visual sensor in order to find new knowledge.

For our thesis, we present a novel exploration strategy that takes inspiration from these two methods. We explain how the robot autonomously decides a target position and orientation and how it can perform obstacle avoidance.

In summary, our system consists of:

- a localization and mapping system able to store information about the environment using a monocular visual sensor, an IMU and an ultrasound altimeter

- a strategy to perform obstacle free navigation with a drone

- an exploration algorithm that maximises the improvement in knowledge about the environment through an strategy designed for PTAM algorithm.

1.1 Thesis Overview

Our thesis is composed as follows:

In Chapter 2, the state of art is introduced with example applications used in exploration scenarios, technologies and methodologies used to achieve this aim.

In Chapter 3, the exploration problem with a monocular low-cost quadrotor is analyzed in detail, and the chosen solutions for navigation and exploration are presented.

In Chapter 4, are illustrated the hardware and software platform used.

In Chapter 5, possible strategies are evaluated and the implementation details of the chosen one are described.

In Chapter 6, conclusions about our work are drawn, and possible future works are outlined.

Chapter 2

State of art

"We're charging our battery
And now we're full of energy
We are the robots ...
We're functioning automatic
And we are dancing mechanic
We are the robots"

We are the robots, Kraftwerk

Exploration of an unknown environment is a challenging problem for robotics.

Our aim is to develop a system capable of autonomous exploration of an environment without any a priori model description of it.

The exploration problem in robotics consists in the use of a robot to maximize the knowledge over a particular area, allowing to understand how is the environment around the robot and to visit previously unseen areas.

Considering a dangerous scenario such as a city after an earthquake the search and rescue of victims could be a really risky task if performed by persons independently of their qualification. The use of robots could help to save lives without sacrifice rescuers. The Robocup Rescue League, which has the aim of simulate this kind of situation, is thereafter described.

In addition, we want to reach our goal of autonomous exploration through the use of a low cost hardware platform. In this chapter, we present also how expensive can be a real world navigation that consists only in path planning with obstacle avoidance citing as an example the typical equipment used in the DARPA Grand Challenge.

These two first examples of applications use robots designed to work on ground surfaces. Nonetheless, terrain conformation could be an obstacle to the navigation of a robot. The use of Unmanned Aerial Vehicle (UAV) can bypass this problem since movements of a flying object are not conditioned by uneven ground. Thereafter we show a possible solution to the autonomous exploration problem developed with a quadrotor by a collaboration between Massachusetts Institute of Technology (MIT) and Technische Universitat Munchen (TUM).

A last example of application, presented by the Vision group of TUM, shows how autonomous navigation capabilities can be achieved by the use of low cost quadrotor.

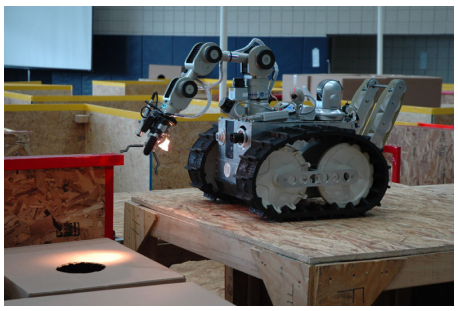
Finally, in this chapter, we present methodologies and algorithms, studied in the last decades, that can be useful in the solution of the exploration problem together with all the activities that are correlated to it such as the localization of the robot and mapping of the environment.

2.1 Example of applications

An example of where autonomous robots exploration can be used is in rescue applications, where they have to perform multiple tasks simultaneously: they have to estimate their position, build a map of the surroundings and decide where to go later. Implementation of this kind of systems is the main aim of the Robcup Rescue Simulation [9].

The RoboCup Rescue Robot League was started in 2001 [10] and takes the research on the competitive level, giving the birth to an international competition for urban search and rescue robots, in which robots compete to find victims in a simulated earthquake environment in test arenas.

(a)



(b)

Figure 2.1: a) RoboCup Rescue 2008 German open test arena b) A robot inspecting a hole in the arena

RoboCup Rescue League teams have to develop robots capable of mobility, sensory perception, mobile manipulation and assistive autonomous behaviors. As in many other applications, there is one basic chore that is of highest importance, in Robocup Rescue this task is to ensure the coverage of the entire environment by the robot in order to localize the maximum number of victims. However, exploration is not possible without the skill to determine the position of the robot and of the surroundings. This knowledge allows to plan the movements to make further inspections.

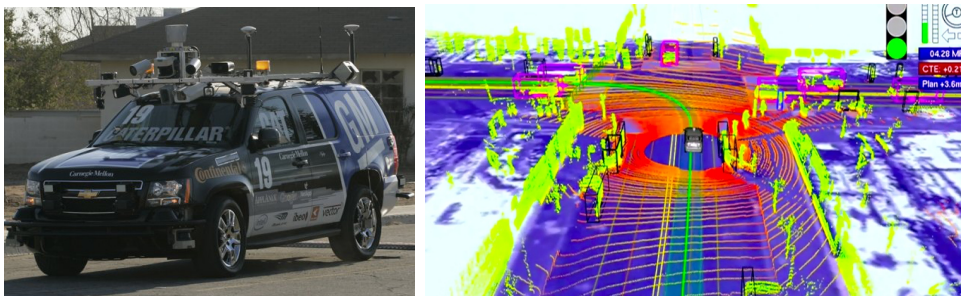
A strategy to perform robot localization without any a priori knowledge (e.g. a map) is to use expensive sensors. Adopting GPS and IMU sensors combined with a laser range scanner system it is possible to rapidly determine a dense and precise three-dimensional map of the environment.

Very reliable systems to localize and map environments have been used in DARPA Grand Challenge [11].

DARPA Challenge was a competition for American driverless vehicles funded by Defense Advanced Research Project Agency (DARPA), an organization of United States Department of Defense. The Grand Challenge was the first long distance competition for driverless vehicles in the world.

In 2012, the team announced that they have completed over 300,000 autonomous-driving accident-free miles. The adoption of $150,000 in equipment including a $70,000 LIDAR[1] is an example of how expansive can be an application of an autonomous wheeled robot in real environments.

[1]Lidar is a portmanteau of *light* and *radar* In late 1970s it was addressed also as an acronym of "LIght Detection And Ranging".



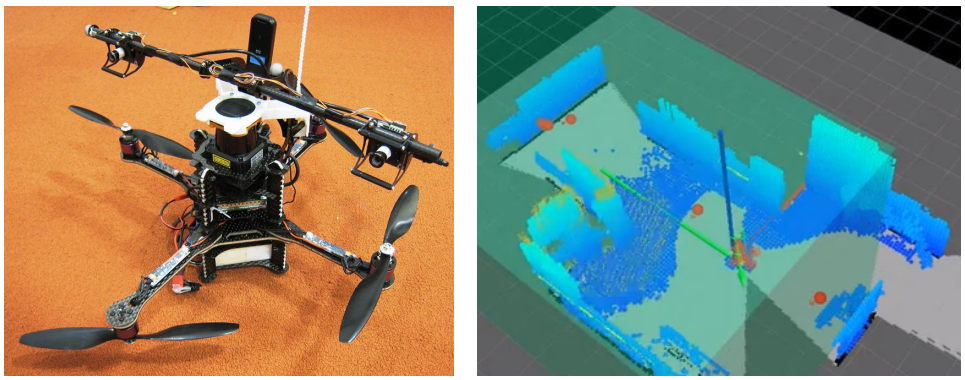
(a) (b)

Figure 2.2: a) Stanley, 2005 DARPA Grand Challenge winner. b) 3D representation of laser data from a driverless car

A limitation of ground vehicles such those used in RoboCup Rescue Robot League and DARPA Grand Challenge is represented by the inability to face particular situations such as crossing water or passing a gully without looking for a feasible ground path. Moreover, ground vehicles have great difficulties traveling on uneven grounds, difficulties that slow down, or even stop, operations. These problems can be bypassed by adopting a typology of vehicle that is not linked directly to the ground. If surface vehicles are well suited for water environments, but not for ground ones, the universal vehicles, in terms of exploration capability, are the flying ones. This kind of vehicles can explore the environment in a wide range of situations and, in case of helicopters, they not need a large runway to takeoff and land. Moreover, GPS localization, as used in DARPA's vehicles, is possible only in outdoor environments denying the exploration of indoor areas.

Concerning exploration ability of a quadrotor in unstructured and unknown indoor environments a team of MIT and TUM presented a solution that let the robot autonomously navigate, explore and locate objects of interest [12].

To achieve this goal an Ascending Technologies Pelican [13](figure 2.3 a) was used. It was equipped with a custom stereo-camera rig providing mounts for two grayscale uEye cameras[14], a Lippert CoreExpress 1.6Ghz Intel Atom board with a wireless link to the ground control station and the lightweight Hokuyo UTM laser rangefinder[15] (price around $5,000). From the combination of data passed by laser and other onboard sensors (figure 2.3 b) they have been able to localize the robot and to reconstruct in three-dimensional space objects and obstacles from environment with a good quality.



(a) (b)

Figure 2.3: a) Ascending Technologies Pelican modified by MIT and Technische Universitat Munchen with stereo camera and laser sensor. b) Laser data reconstructed in a virtual 3D space.

As mentioned, localization and mapping of the environment have been made possible by the use of really expensive hardware.

An example of autonomous navigation without the use of this kind of hardware is presented by TUM-Vision group [16]. The aim of TUM-Vision group system is to make a low-cost quadrotor able to autonomously navigate in previously unknown and GPS-denied indoor and outdoor environments without requiring artificial markers or external sensors. This has been possible using a monocular visual SLAM system and an Extended Kalman Filter for data fusion and state estimation that are thereafter described. In particular, this system allows the quadrotor to accurately fly different figures taking as a reference a single plane located in front of the quadrotor camera. The autonomous navigation system developed by TUM Vision group does not care about collisions making possible to use it only in obstacle-free zone.

2.2 Methodologies and algorithms

In this section, we describe the main approaches used to resolve the localization, mapping and exploration problems available in literature. We show how the environment in which a robot is located can be reconstructed, how can its position be estimated and the main strategies to improve knowledge with exploration techniques.

2.2.1 Three-dimensional reconstruction

Three-dimensional structure of environment can be easily reconstructed with LIDAR systems. This kind of systems were firstly developed in 1960s shortly after the invention of laser. They are based on reflection of laser upon surfaces. By measuring the time it takes for a laser impulse to come back to a sensor situated along with the emitter is possible to compute the exact distance of a reflecting object [2]. First applications for this system were in Meteorology (e.g. to measure the distance of clouds), but rapidly evolved in more complex systems which allowed, by using rotating lasers and sensors [17], to reconstruct three-dimensional environments, represented by point clouds.

Another approach to obtain information about the environment takes inspiration from animals visual perception. Stereopsis[3] is possible in humans and other animals by perceiving multi ocular (typically binocular in mammals) translated vision of the same environment [18].

Structure from Motion (SfM) is an approach that takes the same assumptions of stereo images and applies them to a single moving camera. In this case, we have to consider images sensed over time instead of images taken at the same time from different positions.

Since for our thesis we used a monocular camera system, from now on we focus the description of three-dimensional structure estimation using SfM.

To find relations (i.e. translation) between images, at first, particular features are searched in each image. A feature of an image is a piece of information which is relevant for solving some problem. The simplest features present in an image are represented by corners, edges and blobs. One of the earliest algorithm to detect important features was presented by H. Moravec [19]. This algorithm marks as important features the pixels for which their neighbourhood pixels have very different values (in terms of color or intensity). Moravec approach was then improved by C. Harris and M. Stephens [20] which, with their algorithm, give more importance to the direction of the patch[4]. Further studies to this detector have been carried on by J. Shi and C. Tomasi, who made a slight variation on the computation of the "goodness" of a feature resulting on much better results compared

[2]Distance can be computed with equation $d = \frac{v}{2t}$ where d is the distance and t is speed of light in air.

[3]From *stereo* meaning "solid" or "three-dimensional", and *opsis* meaning appearance or sight.

[4]A patch of a candidate feature in an image is its pixels neighbourhood

with those of original Harris detector [21].

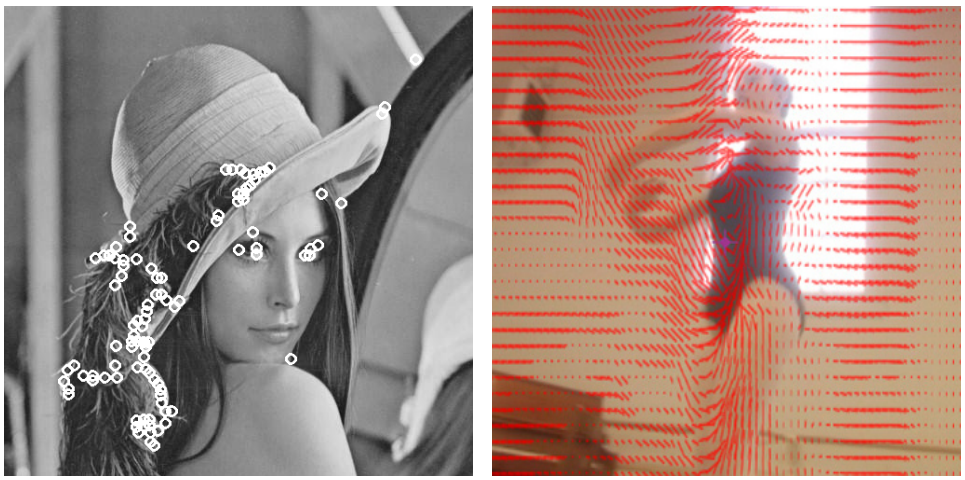
A slightly different approach to detect features in images has been shown by E. Rosten and T. Drummond with their FAST[5] algorithm [22]. This algorithm classifies important features by evaluating a circle of pixels around the inspected pixel[6]. The pixel considered is classified as an important feature if a set of contiguous pixel in the circle contains exclusively values darker (or brighter) than the circle center.

Features extracted from an image represent a sort of description of the scene. By comparing features found in different frames, a correspondence between them can be established. These correspondences can be found either via optical flow or via features matching that we describe in the following.

The optical flow concept was firstly introduced in 1940 by J.J. Gibson who tried to explain how living creatures perceive stimulus from the visual world [24]. D.H. Warren and E.R. Strelow [25] then thought that this approach could be reused in electronic vision to help blind humans in detecting relative motion of objects by evaluating the velocity field which warps one image into another. The main methods used for computing the optical flow are Lucas-Kanade [26] and Horn-Schunck [27] methods. The first algorithm is the most widely adopted method for computing optical flow given its low computational complexity. It is a local optical flow estimator which takes the assumption that the flow is constant in the neighbourhood of a point and solves the problem of estimating the vector of flow by looking for a similar patch in the neighbourhood of the given pixel by means of least squares criterion. On the other hand, the algorithm presented by B.K.P. Horn and B.G. Schunck assumes that the flow over the image cannot contain discontinuities, so it tries to minimize an energy function computed over the whole image. This method allows global estimates of the optical flow and has the advantage to give more precise information of the flow in homogeneous objects (the flow of a homogeneous area results as a mean of the region boundary flow), but it is highly susceptible to image noise.

[5]Features from Accelerated Segment Test.

[6]The circle is computed with mid-point circle algorithm [23].



(a) (b)

Figure 2.4: Example of harris corner detection (a) and Horn Schunck optical flow (b).

Feature matching is a different approach to solve the tracking problem. As the name suggests, this approach is based upon the seek of similar features between images rather than trying to understand where pixels have moved on subsequent frames. This method firstly looks for features in both images with one of the aforementioned methods, it then assigns a descriptor to each of them, and then makes a comparison between descriptors to detect the corresponding ones. Descriptors of features can be computed with several methods, the most used is SIFT[7] algorithm [28]. D.G. Lowe originally designed SIFT to detect objects in different images after an initial training phase, the key-point descriptor step of the algorithm can be however used to match only corner features between images. Other common features descriptor algorithms are SURF[8] [29], BRIEF[9] [30] and ORB[10] [31][32]. These descriptors use a patch around a pixel to make a description of the feature. SIFT for example uses a set of orientation histograms which then is normalised (to enhance invariance to illumination changes), SURF instead computes description as sum of the Haar wavelet response around the point of interest[11], BRIEF classifies features on the basis of a "small" number of pairwise intensity comparisons in the feature neighbour-

[7] Scale-invariant feature transform.

[8] Speeded Up Robust Features.

[9] Binary Robust Independent Elementary Features.

[10] Oriented FAST and Rotated Brief.

[11] Haar wavelet are similar to Fourier transform, but it uses "square shaped" functions instead of trigonometric ones

hood , ORB takes inspiration from the BRIEF descriptor, but it also relies on orientation. After the computation of the descriptors of the features the comparison phase allows to match the points by computing L^2-norm (for the descriptors which store description in an array as SIFT or SURF) or Hamming distance (for string descriptors such as BRIEF or ORB).

Figure 2.5: Feature matching using SURF algorithm

Correct correspondences of features between image frames are used to understand movement that a camera has done in terms of rotation and translation between them. The method described by H.C. Longuet-Higgins [33] allows to extract the fundamental matrix, a 3×3 matrix which relates corresponding points in stereo images, from the pairs of corresponding points in two images with the normalised eight-point algorithm [34].

Knowing the intrinsic parameters of the cameras used to take the images of the scene it is possible to extract the essential matrix [35], which can then be decomposed to extract rotation and translation of a camera with respect to the other. Moreover, with the essential matrix, it is possible to estimate the three-dimensional position of the feature points making a triangulation of them[12].

Rotation, translation and structure reconstruction between two image frames is the basis of visual SLAM[13] works. Studies to reconstruct both the three-dimensional structure of environment and the trajectory of a camera, have been firstly held A. J. Davison [4]. The early MRTP[14] works enabled to track points and remember the position of out-of-sight objects enabling a robot to continuously frame an object and then to change orientation of the camera to frame a previously seen point without the need of markers.

[12] A process that can be solved with the already computed essential matrix.
[13] Simultaneous Localization And Mapping.
[14] Monocular SLAM and Real-Time Perception.

An improvement to Davison's technique has been made by G. Klein and D. Murray in their PTAM[15] algorithm [6]. The main purpose of PTAM is the estimation of an handheld camera pose in an unknown scene by combining different steps such as: visual odometry, bundle adjustment and loop closure detection. By splitting tracking and mapping processes in parallel threads they obtain real-time performances and detailed three-dimensional maps of the environment[16]. PTAM is essentially based upon constant frame inspection and feature extraction; it allows to store camera frames and world points (called *keyframes* and *keypoints*) only if they are really needed (redundant frames can be ignored if another frame has been taken in a near position and outliers points are removed with a statistical process).

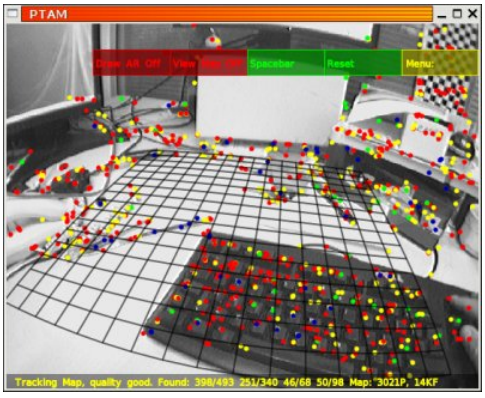


Figure 2.6: PTAM graphical interface

When tracking and maintaining three-dimensional structure calculation, errors and imprecisions increase at each iteration. These errors result in unfound correspondences of features after the camera moves along a loop. This problem, known as loop closure problem, can be resolved by considering a probabilistic feature-based map in which are stored, together with the best estimates of three-dimensional points, their uncertainty described by deviations from these values [36].

Further results by Newcombe and Davison enable the creation of a live dense reconstruction system of the environment called Dense Tracking And Mapping (DTAM) [37]. The system uses dense track of all pixels in the *keyframes* to rebuild a fine representation of the world.

For our exploration system we have chosen the PTAM algorithm, described in Chapter 3, due to its real-time performances and its few require-

[15]Parallel Tracking and Mapping.
[16]Result point cloud contains thousands of points.

ments (it needs only a single camera to perform localization and mapping).

2.2.2 Pose estimation

The first approach for three-dimensional space pose estimation appeared in late fifties [38] during the Cold War. Space race brought the competitors (USA and URSS) to make great investments on aerospace research. In 1958 R.E. Kalman and R.S. Bucy [2] started performing basic research on estimation and control of aerospace vehicles. They reformulated the problem of optimal estimation methods, developed by N. Wiener [39] and A. N. Kolmogorov [40], in the time domain using a generalised state-space form of a linear differential equation introduced in 1908 by P. Langevin to model Brownian motion[17] [41]. The equation of the Kalman filter resulted then equivalent to a particular case of the Wiener filter which was already recasted from a non-linear differential equation to a system of linear equations following Riccati's reduction [42].

A non-linear version of the Kalman filter is known as the Extended Kalman Filter [43]. This version has been developed to solve most engineering problems given their implicit non-linear nature (e.g. speed and pose estimation of aerovehicles). The Extended Kalman Filter gives reasonable performance, and it is presently a well established standard in pose estimation systems [44].

The Kalman Filter algorithm takes as input series of sensor measurements observed over time, containing noise and other inaccuracies (random variations and measurement errors), and produces as output the estimation of unknown variables (e.g. pose estimation) that tend to be more precise than those estimated based on a single measurement. The operation is split in two phases: state prediction and state update. Prediction consists of estimating the next state by observing the last one and projecting it using the last information received from the sensors. Update is applied when new information is available form the sensors and adjusts, using a weighted average[18], the last prediction with the new measurements. Since the algorithm is not complex and it needs no information other than new measurements and last state variables, it can be run in real time.

[17] The motion of small particles contained in fluids.

[18] More weight being given to estimates with higher certainty.

2.2.3 Exploration

Exploration of unknown environment by robots is a problem that can be described as the maximization of the knowledge over the environment itself.

Before real exploration algorithms were presented, a similar result of exploration (three-dimensional map building, localization and safe travel) could be achieved by systems guided by markers [45]. This kind of systems, however, implement only navigation, since the target of the exploration was given to human agents (e.g. markers, data input). Since the main target of the activity was to reach markers and not to explicitly increase the knowledge over the environment we cannot talk of exploration. The right strategy to take when exploring is not directly goal driven, the robots rather have to seek new unseen areas to view and explore.

One of the first approaches that focuses over knowledge gain is the so called frontier-based algorithm presented by B. Yamauchi. Behaving in this way, a robot sets its path towards regions that are adjacent to unmodeled regions. By applying this behaviour the robot could reach positions that let it increase its knowledge. Frontier-based algorithms are adaptable to large and narrow spaces without any shape restriction allowing the robot to increase its knowledge over all the areas reachable from starting position [7].

Another approach to exploration is called view-improvement strategy [8]. This kind of strategy is designed for limited field-of-view sensors (e.g. RGB cameras) and focuses on increasing the knowledge by turning around already known obstacles to understand how is the environment behind them.

Since a room can be visited multiple times causing a loss of time, segmentation of the environment can be applied to classify environment. In this way it is possible to divide the environment in subsets (e.g. rooms) with an enhanced Voronoi diagram. This approach better suites real-world cases since it can avoid reiteration of exploration of the same room [46].

Once computed the target position to improve the robot's knowledge a path that avoids obstacles and minimizes the cost (represented by the travel distance) must be planned. This can be done by the Dijkstra [47], A* [48] or Fast Marching [49] algorithms. The path computed in this way may lead the robot to travel very near the obstacles. A very basic approach to travel in adequately free areas (e.g. far from walls) is to dilate and to consider as occupied all the areas around obstacles. Other approaches deal with the concept of Extended Voronoi distance transform [50]. Distance transform is a transformation of obstacle map which returns a map of the same size in

which each cell is populated with the distance between it and the nearest occupied cell in the map. By inspecting the distance transform for each movement it is possible to travel only in the safe areas (e.g. those more distant form the walls).

In this thesis we present a novel quadrotor autonomous exploration system using approaches similar to those described before (such as using visual SLAM) paying attention to use hardware that is as cheap as possible and dealing with the limitations related to this restriction.

Chapter 3

Problem analysis and proposed solution

"Hot dog, let's play games.
You catch me and I catch you;
no love can cut our knife in two"

Speedy - Runaround, Isaac Asimov

As we have described previously, our aim is to create a system to explore an unknown environment using a low-cost quadrotor. This choice has entailed a study on algorithms and strategies able to work with few sensors such as camera, IMU and altimeter.

We have divided the achievement of our goal in two principal tasks: the first one concerns the study of three-dimensional surroundings map construction and pose estimation, the second one regards the study of an artificial intelligence able to drive safely the drone in a room without collisions with objects or walls.

These tasks are strictly connected: we can not explore anything without knowledge about quadrotor position and surroundings as well as we cannot increase our environment knowledge with the help of only pose estimation.

The first task is assigned to a SLAM manager whereas the second is assigned to an Exploration manager (figure 3.1).

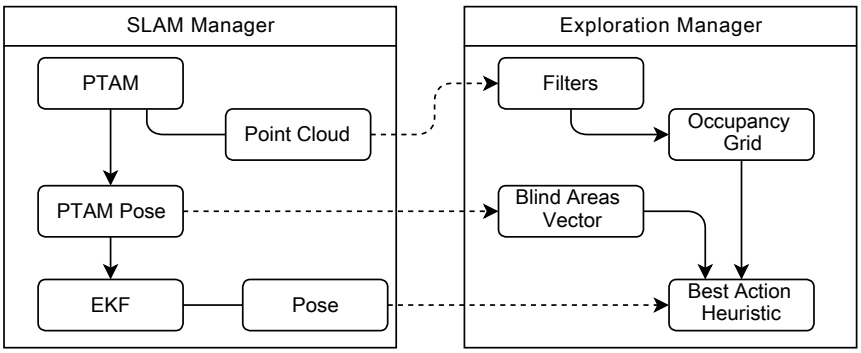


Figure 3.1: Scheme of system life-cycle

3.1 3D reconstruction and pose estimation

SLAM manager makes it possible to localize the quadrotor in the three-dimensional space and, in parallel, to reconstruct a map of the surroundings of the robot.

The selection of the SLAM algorithm has been influenced by our choice to use affordable hardware. As mentioned in chapter 2, LIDAR systems are really expensive, so we have opted for a quadrotor equipped with a single, low-cost camera.

To localize the quadrotor and to reconstruct a map of an unknown environment, we have chosen, between SLAM algorithms mentioned in chapter 2, to use PTAM [6], which is a monocular visual algorithm.

This algorithm has also been chosen for its real time performance and its reliability.

3.1.1 PTAM

PTAM algorithm has been originally designed to provide augmented reality with a hand-held single camera. Moreover the authors of PTAM decided to provide tracking without prior model of world (e.g. markers or environment structure) and dedicated particular attention to speed, accuracy and robustness of the algorithm.

To estimate the pose of the camera the algorithm relies on features tracking and registering.

The algorithm have threads to compute independently the map (represented by three-dimensional registration of tracked features) and the pose of the camera. The parallel execution of these threads possible to have a real

time estimation of the camera pose without being affected by the "heavy" work of building and maintaining a map.

The map of PTAM consists of a set S of N keyframes (snapshot taken by the camera), along with the corresponding estimated position of the camera in the world $\boldsymbol{\mu}$ with respect to a given reference. Moreover each keyframe has its own set of points $\mathbf{p}$, called keypoints, which are three-dimensional representation of features in the world. All M keypoints are stored along with their position coordinates and a patch extracted from nearby pixels in the image frame which they belong to.

The mapping thread performs the task of finding three-dimensional points of the world, filtering and registering them by importance.

The first step for almost each image analysis application is to remove lens distortion and calibrate the camera to understand its camera model.

Removal of lens distortion is made by 5 intrinsic parameters k_1, k_2, k_3, p_1, p_2. k_1, k_2 and k_3 are parameters used to correct radial distortion caused by the lense morphology (the more noticeable effect of radial distortion is the so called "fish-eye" effect).

$$\begin{aligned} x_k &= x(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) \\ y_k &= y(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) \end{aligned} \tag{3.1}$$

In 3.1 x_k and y_k are the correct coordinates after removing radial distortion.

p_1 and p_2 are used instead to correct tangential distortion that is due to the non-perfect parallel alignment of the lens with the sensor.

$$\begin{aligned} x_p &= x + [2p_1 xy + p_2(r^2 + 2x^2)] \\ y_p &= y + [p_1(r^2 + 2y^2) + 2p_2 xy] \end{aligned} \tag{3.2}$$

In 3.2 x_p and y_p are the correct coordinates after removing tangential distortion.

The intrinsic matrix (3.3) contains parameters which encompass focal lengths which are the distances, in term of pixels, from the center of the lens to the focal point (f_x, f_y) and image principal point that is the intersection between the optical axis of the lens and the sensor plane (c_x, c_y).

$$\begin{pmatrix} x \\ y \\ w \end{pmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{pmatrix} X \\ Y \\ Z \end{pmatrix} \tag{3.3}$$

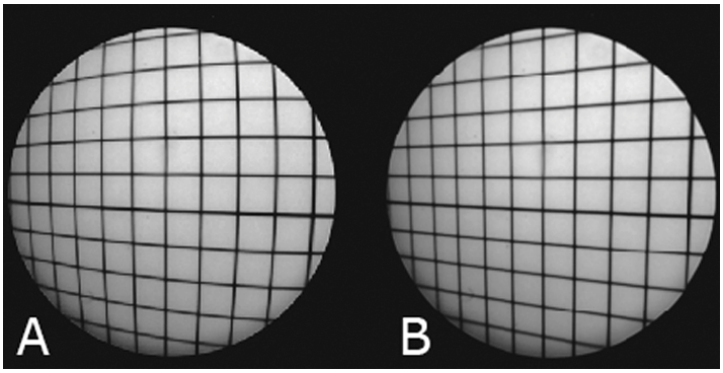


Figure 3.2: An example of lens radial distortion correction.

These parameters (distortion and intrinsic ones) are identified with a camera calibration, done once for all, when the camera is fixed on its reference frame with the algorithm described by Z. Zhang in [51].

With undistorted images it is possible to initialise PTAM system by taking two translated frames from the camera (this step is called stereo-initialisation).

Stereo initialization generates the first map population. This is done by looking for important features on a taken keyframe and then tracking these features, with Lucas-Kanade algorithm, on the smooth movement of the camera until a second keyframe is taken. Feature detection is made by using FAST algorithm which is based on machine learning and is very fast compared to other approaches such as Harris detector. FAST algorithm uses a circle of 16 pixels to classify whether a candidate point is actually a corner (figure 3.3). This is done by counting how many contiguous pixels on that circle are brighter or darker than the center pixel. When a point is classified as more than 12, the point is marked as a feature. The features are searched on four pyramid levels which are extracted from the original frame by repeatedly smoothing and subsampling it resulting in downscaled images of the original frame. Features found on lower pyramid levels are the most robust ones because they are trackable on frames very far from each other. The explanation of this robustness is due to the fact that features on lower pyramid levels are relative to bigger patch of the frame with respect to the patches taken on the original one.

After having tracked these important features PTAM, computes Shi-Tomasi score for each extracted point. If the score is sufficiently high then the point is added as a keypoint to the map.

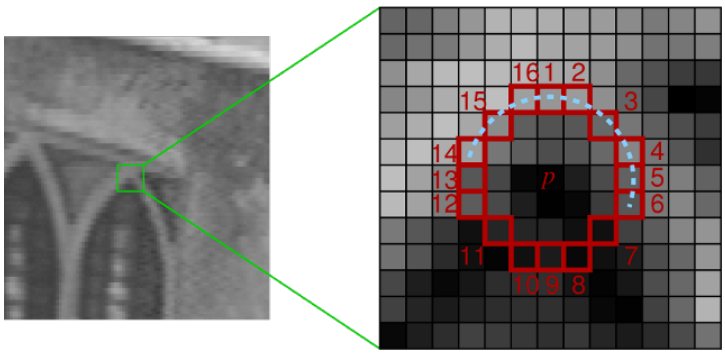


Figure 3.3: Fast Features Detection.

The described approach is used to decrement time needed to extract features (with the low computational complexity of the FAST algorithm) guaranteeing a good tracking quality (with the accuracy of Shi-Tomasi algorithm).

Shi-Tomasi score is computed starting from the matrix representation of the image intensity[1] I of area (u, v) with x and y variations over the patch.

$$S(x,y) = \sum_u \sum_v w(u,v)(I(u+x, v+y) - I(u,v))^2 \tag{3.4}$$

$w(u, v)$ is the weight function over the considered window of the feature (w could be of different shapes and types: most used are circular and Gaussian windows).

Considering the image gradients over the two axes represented by I_x and I_y we can approximate the intensity of a near pixel as

$$I(u+x, v+y) \approx I(u,v) + I_x(u,v)x + I_y(u,v)y \tag{3.5}$$

So the sum of squared distance can be approximated as

$$S(x,y) \approx \begin{pmatrix} x & y \end{pmatrix} \mathbf{A} \begin{pmatrix} x \\ y \end{pmatrix} \tag{3.6}$$

where the matrix $\mathbf{A}$ (in Eq. 3.7) is called the structure tensor (angle brackets denote weighted averaging over (u, v) with w function).

$$\mathbf{A} = \sum_u \sum_v w(u,v) \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix} = \begin{bmatrix} \langle I_x^2 \rangle & \langle I_x I_y \rangle \\ \langle I_x I_y \rangle & \langle I_y^2 \rangle \end{bmatrix} \tag{3.7}$$

[1] image intensity is computed as an average of the channels of the image

The Shi-Tomasi score is represented by the minimum of the eigenvalues of the structure tensor (Harris algorithm choose the bigger between the two).

$$min(\lambda_1, \lambda_2) \text{ with } \lambda_1, \lambda_2 = eig(\mathbf{A}) \tag{3.8}$$

If the Shi-Tomasi score is sufficiently high the point is added to keypoint set.

Now, the initialisation generates the first population of keypoints in the map of PTAM by applying structure from motion principles to the tracked points.

To understand the process we introduce the fundamental matrix (3.9) which is a matrix that relates corresponding points in stereo images.

$$\mathbf{x}'^{\mathrm{T}}\mathbf{F}\mathbf{x} = 0 \tag{3.9}$$

Nevertheless, the fundamental matrix maps the points between uncalibrated camera frames; if we have calibrated image frames it is better to use the essential matrix which is related to the fundamental by Eq. 3.10 and satisfies Eq. 3.11, where y and y' are image calibrated coordinates.

$$\mathbf{E} = \mathbf{K}'^{\mathrm{T}}\mathbf{F}\mathbf{K} \tag{3.10}$$

$$\mathbf{y}'^{\mathrm{T}}\mathbf{E}\mathbf{y} = 0 \tag{3.11}$$

The essential matrix $\mathbf{E}$ is found by applying the five-point algorithm presented by H. Stewénius, C. Engels and D. Nistér [52] and performing RANSAC to remove outliers points.

The proof of Eq. 3.14 is in Eq. 3.12 given x and x' the three-dimensional coordinates of the same point relatively to each frame, y and y' are the relatives projections on the image plane (homogeneous coordinates)

$$\begin{pmatrix} y_1 \\ y_2 \\ 1 \end{pmatrix} = \frac{1}{x_3}\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} \text{ and } \begin{pmatrix} y'_1 \\ y'_2 \end{pmatrix} = \frac{1}{x'_3}\begin{pmatrix} x'_1 \\ x'_2 \\ x'_3 \end{pmatrix} \tag{3.12}$$

$$\mathbf{x}' = \mathbf{R}(\mathbf{x} - \mathbf{t}) \tag{3.13}$$

$\mathbf{E}$ can be defined as the multiplication of rotation matrix $\mathbf{R}$ and the matrix representation of the cross product with $\mathbf{t}$ ($[\mathbf{t}]_\times$) as in Eq. 3.14.

$$\mathbf{E} = \mathbf{R}[\mathbf{t}]_\times \tag{3.14}$$

In 3.16 we show that this definition of $\mathbf{E}$ describes a constraint on corresponding image coordinates.

$$\mathbf{x}'^{\mathrm{T}}\mathbf{E}\mathbf{x} = (\mathbf{x}-\mathbf{t})^{\mathrm{T}}\mathbf{R}^{\mathrm{T}}\mathbf{R}[\mathbf{t}]_{\times}\mathbf{x} = (\mathbf{x}-\mathbf{t})^{\mathrm{T}}[\mathbf{t}]_{\times}\mathbf{x} = 0 \tag{3.15}$$

Now we can show that the previous relation implies a relation between image points:

$$0 = \mathbf{x}'^{\mathrm{T}}\mathbf{E}\mathbf{x} = \left(\frac{1}{x'_3}\right)\mathbf{x}'^{\mathrm{T}}\mathbf{E}\left(\frac{1}{x_3}\right)\mathbf{x} = \mathbf{y}'^{\mathrm{T}}\mathbf{E}\mathbf{y} \tag{3.16}$$

Rotation and translation can therefore be computed from $\mathbf{E}$ determining the rotation and translation (up to a scaling) between the two camera's coordinate systems.

After having computed the singular value decomposition of the essential matrix $\mathbf{E} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^{\mathrm{T}}$ translation and rotation are found then as in Eq. 3.17, Eq. 3.18 and Eq. 3.19.

$$\mathbf{W} := \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \text{ with } \mathbf{W^{-1}} = \mathbf{W}^{\mathrm{T}} = \begin{bmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.17}$$

$$\mathbf{R} = \mathbf{U}\mathbf{W}^{-1}\mathbf{V}^{\mathrm{T}} \tag{3.18}$$

$$[\mathbf{t}]_{\times} = \mathbf{V}\mathbf{W}\mathbf{\Sigma}\mathbf{V}^{\mathrm{T}} \tag{3.19}$$

These results are valid because of Eq. 3.20, but the rotation and translation found are not the only found with this process.

$$\mathbf{R}[\mathbf{t}]_{\times} = \mathbf{U}\mathbf{W}^{-1}\mathbf{V}^{\mathrm{T}}\mathbf{V}\mathbf{W}\mathbf{\Sigma}\mathbf{V}^{\mathrm{T}} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^{\mathrm{T}} = \mathbf{E} \tag{3.20}$$

Other solution could be found by changing the sign of the scale of $\mathbf{t}$ or changing $\mathbf{W}$ with $\mathbf{W}^{-1}$ for a total of four possible solutions. Nevertheless, only one of these is feasible since the other three solutions have a translation vector which lies behind at least one of the two camera poses.

However, the problem of the scale remains and it can be solved by knowing the initial translation of the cameras and then by multiplying the translation computed before by the correct scale factor.

After having initialised PTAM mapping process the map has to be continuously updated, adding, if necessary, new keyframes and keypoints, updating keypoints (removing outliers) and adjusting measurements.

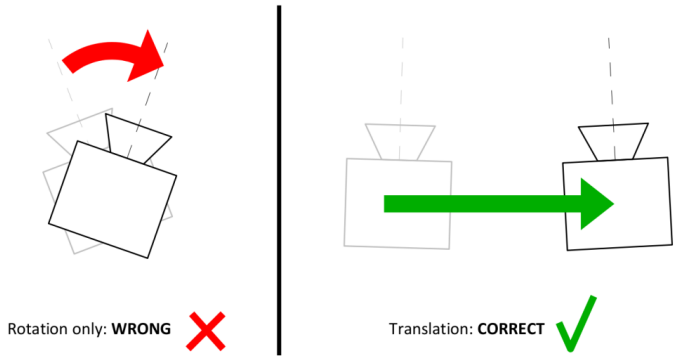


Figure 3.4: PTAM initialization movement.

A new keyframe is added at anytime the condition of good tracking (expressed by ratio between the number of expected keypoints in current frame and currently viewed ones) is respected and the position of the camera is sufficiently different from an already existent keyframe (a translation is needed both to not influence the system too much with the same frame and to ensure stereo baseline for feature triangulation). Moreover, the distance from currently seen points is taken into account; if they are very close to the current estimated pose the translation from a previously registered keyframe needed to take another keyframe is smaller than the distance needed in a very depth scene (if the camera is very far from seen points it is needed a big translation to have a change in the framed scene).

$$\begin{pmatrix} u_i \\ v_i \end{pmatrix} = \text{CamProj}(\mathbf{p}_{i\mathcal{C}}) \tag{3.21}$$

The pose estimation of the keyframe is assumed to be the output of the tracking thread. Taking into account the camera pose, all keypoints are reprojected on the keyframe (in Eq. 3.21 CamProj() is a $\mathbb{R}^3 \rightarrow \mathbb{R}^2$ function that considers Eq. 3.1 and Eq. 3.2, and u_i and v_i are the coordinates of the point p_i reprojected on the image plane), then a score measure is computed for each feature to check if there are outliers to be removed. After that, features are extracted with FAST from the taken keyframe, these features are filtered then by Shi-Tomasi score and added to the map only if they are not too close to already existent keypoints. To add depth information to the newly added keypoints triangulation with the nearest keyframe is done (this step is similar to the initialisation one).

Every keyframe i of the map besides its keypoints has a set of positions of

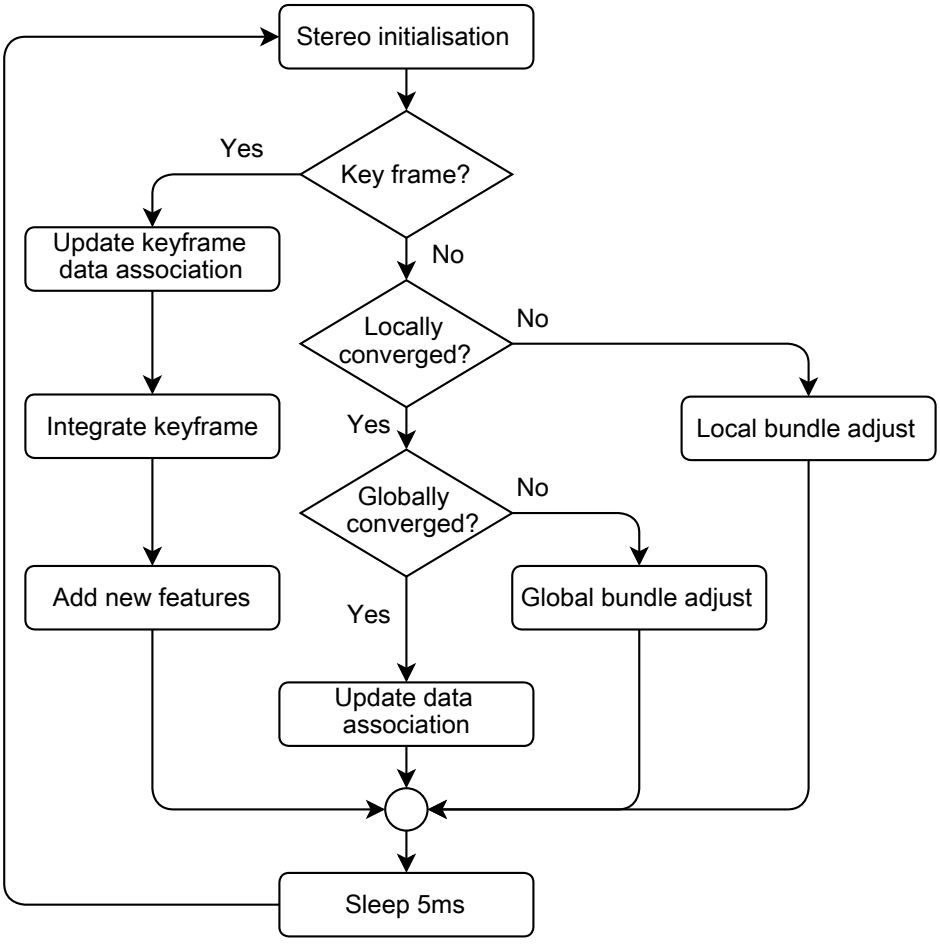


Figure 3.5: Mapping thread

keypoints taken in other keyframes. The resulting measurement is composed by the average of the three-dimensional position of the keypoint and its standard deviation. Bundle adjustment (3.22) iteratively adjusts the map in order to minimise the error using Tukey's biweight objective function Obj() [53].

$$\{\{\boldsymbol{\mu}_2 \dots \boldsymbol{\mu}_N\}, \{\mathbf{p}'_1 \dots \mathbf{p}'_M\}\} = \underset{\{\{\boldsymbol{\mu}\},\{\mathbf{p}'\}\}}{\operatorname{argmin}} \sum_{i=1}^{N} \sum_{j \in S_i} \operatorname{Obj}\left(\frac{|\mathbf{e}_{ij}|}{\sigma_{ji}}, \sigma_T\right) \tag{3.22}$$

The optimisation is done locally, minimising the estimator only for the last N keyframes[2] and globally using all map keyframes.

To track the movement of the camera PTAM does not use optical flow or features matching between frames, instead it makes an early and rough camera position with a decaying velocity model. A coarse pose update of the estimate is therefore done by reprojecting over the frame 50 keypoints. A search in the neighbourhood of the projection is done after applying an affine transformation to projected patches of pixels to better match the current frame ones (position in the frame are amended). In this way PTAM refines the previously estimated pose (with the motion model) to a better one. The last step is now to make a fine pose estimation reprojecting all the known keypoints in the current frame and making another patch search to update the actual pose to the correct one. The pose update is done by minimizing reprojection error as in Eq. 3.23.

$$\boldsymbol{\mu}' = \underset{\boldsymbol{\mu}}{\operatorname{argmin}} \sum_{j \in S} \operatorname{Obj}\left(\frac{|\mathbf{e}_j|}{\sigma_j}, \sigma_T\right) \tag{3.23}$$

If PTAM loses tracking of the pose an initial guess of it is made by image matching with all the stored keyframes in the map. The camera position of the keyframe which best fits the current one is used to begin the coarse phase.

3.2 Pose estimation

As mentioned in the previous section, it is possible to estimate the pose of the camera through a visual SLAM system as PTAM.

To achieve our aim (autonomous exploration using a low-cost quadrotor) the pose estimation can not be computed only by the PTAM system since

[2] three frames are used in PTAM implementation

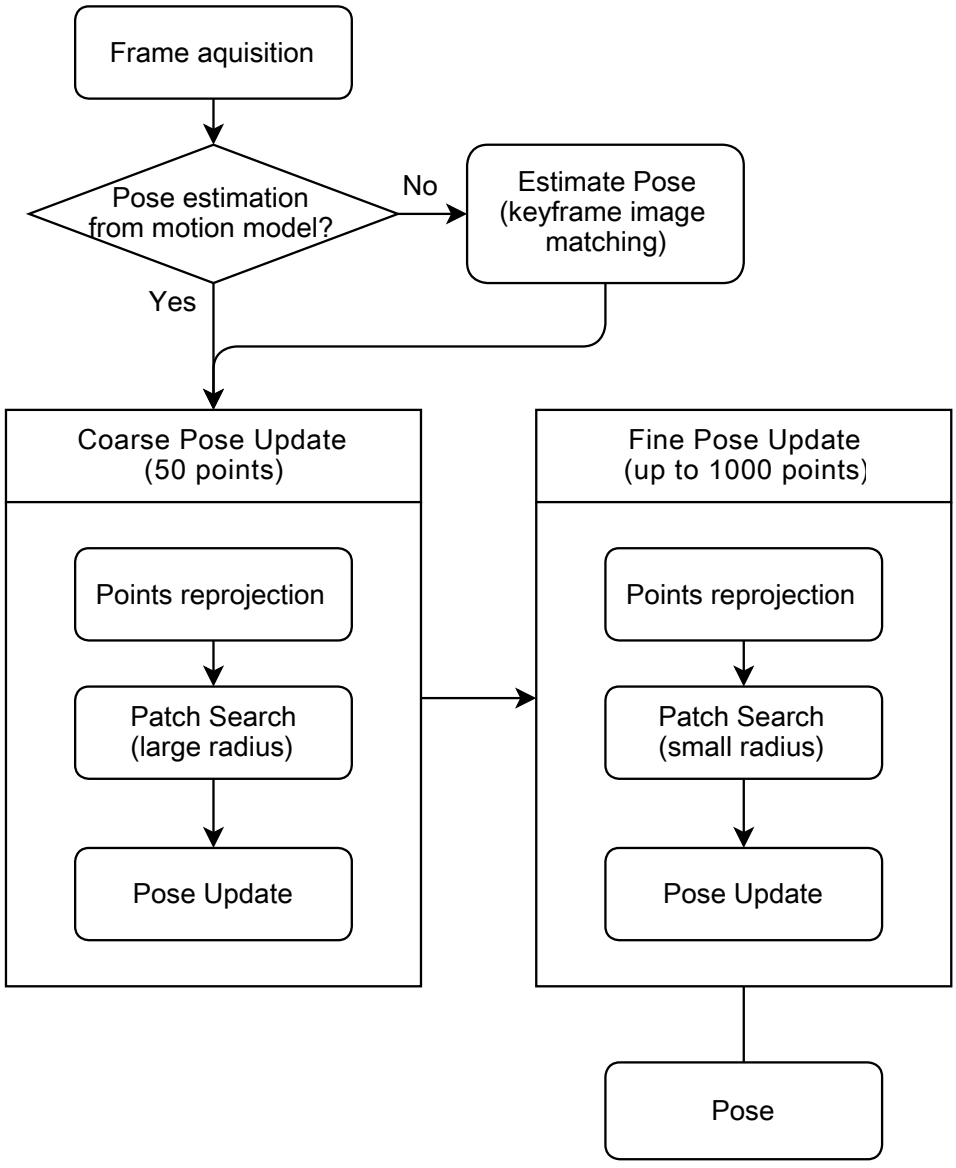


Figure 3.6: Tracking thread

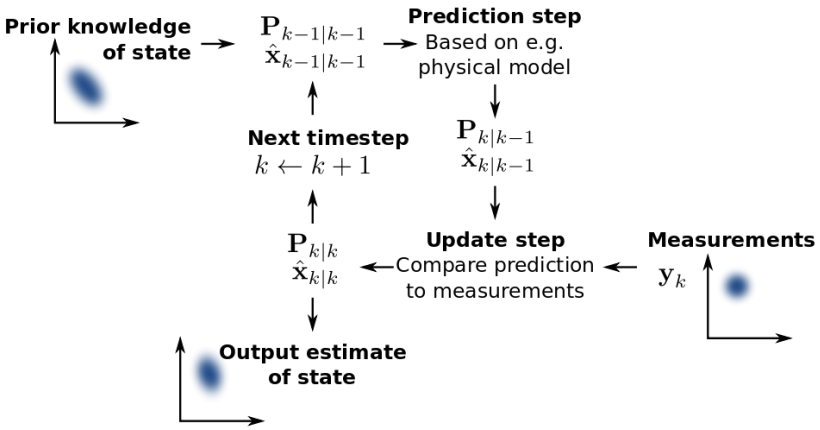


Figure 3.7: A synthetic representation of how a Kalman filter works.

it is not developed for an UAV. The original aim of PTAM developers is the creation of a system useful in augmented reality applications using a hand-held camera technique.

A limitation of PTAM system consists in possible loss of the tracking of the found features due to various situations. Changes of light intensity, camera occlusion or fast movements of the camera could lead to a sudden change of the acquired frame (with respect to the previous one) avoiding the correct estimation of the pose of the camera.

In augmented reality (AR) applications, the loss of features tracking is not critical because, in this case, the result of this failure does not condition the application flow (the rendering of the AR layer is locked and hidden until the tracking is restored and a new pose is estimated). The restoring of the features in original implementation can be considered automatic because it is a consequence of the instinct of the human who is controlling the camera.

In our case, the system has to be updated on the estimate of the pose, also in cases of blindness (when the features tracking fails) because there is not any automatism typical of human behavior. The system has to be aware of the quadrotor pose in order to be able to restore the visual tracking.

To estimate the pose of the camera without any visual reference, we have chosen to use a method, seen in literature, that relies on sensors (IMU and altimeter) readings. This method uses an Extended Kalman Filter to produce a good estimation of the pose starting from sensors measurements.

3.2.1 Extended Kalman Filter

The Extended Kalman filter is the nonlinear version of the Kalman filter. It is better suited in all the occasions for which it is not possible to represent a problem (possibly through an approximation) in a linear form. In our case, it is used to initialize the PTAM system (scale factor) and for dead reckoning when the PTAM pose estimation is not available.

The algorithm works similarly to the Kalman filter working in a discrete time-domain, but rather than having a linear transformation to represent the state transition model it has non linear, but differentiable functions.

$$\mathbf{x}_k = f(\mathbf{x}_{k-1}, \mathbf{u}_{k-1}) + \mathbf{w}_{k-1} \tag{3.24}$$

The state of the system at step k is represented by vector $\mathbf{x}$. The state is updated with the state transition function f that takes, as arguments, the previous state and control commands sent to the drone (represented by $\mathbf{u}$) to which is added process noise $\mathbf{w}$. Process noise is modeled by $\mathbf{w} \sim \mathcal{N}(0, \mathbf{Q}_k)$ where $\mathbf{Q}_k$ is the covariance of the process noise.

In our case the complete state function f is represented by

$$\begin{pmatrix} x \\ y \\ z \\ \dot{x} \\ \dot{y} \\ \dot{z} \\ \phi \\ \theta \\ \psi \\ \dot{\psi} \end{pmatrix}_k = \begin{pmatrix} x \\ y \\ z \\ \dot{x} \\ \dot{y} \\ \dot{z} \\ \phi \\ \theta \\ \psi \\ \dot{\psi} \end{pmatrix}_{k-1} + \delta_k \begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \\ \ddot{x} \\ \ddot{y} \\ \ddot{z} \\ \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \\ \ddot{\psi} \end{pmatrix}_{k-1} \tag{3.25}$$

where x, y and z is the position of the quadrotor relative to the position of take off; $\dot{x}$, $\dot{y}$ and $\dot{z}$ are the linear velocities; $\ddot{x}$, $\ddot{y}$ and $\ddot{z}$ are the linear accelerations; ϕ, θ and ψ are roll, pitch and yaw angles (rotations over x, y and z axes); $\dot{\phi}$, $\dot{\theta}$ and $\dot{\psi}$ are the angular velocities and $\ddot{\psi}$ is the angular acceleration over the z axis. δ_k is the time between step k and step $k-1$.

At time k a new observation of the state is done as $\mathbf{z}$

$$\mathbf{z}_k = h(\mathbf{x}_k) + \mathbf{v}_k \tag{3.26}$$

where h is the observation model which maps the true state space into the observed space and $\mathbf{v}$ is the observation noise which is assumed to be distributed as $\mathbf{v} \sim \mathcal{N}(0, \mathbf{R}_k)$ where $\mathbf{R}_k$ is the covariance of the observation noise.

In our case the observations are both made by PTAM (h_{PTAM}) and by IMU (h_{IMU}) (Eq. 3.27).

$$h_{\text{PTAM}}(\mathbf{x}) = \begin{pmatrix} x \\ y \\ z \\ \phi \\ \theta \\ \psi \end{pmatrix} \quad , \quad h_{\text{IMU}}(\mathbf{x}) = \begin{pmatrix} \cos(\psi)\dot{x} - \sin(\psi)\dot{y} \\ \sin(\psi)\dot{x} - \cos(\psi)\dot{y} \\ z \\ \phi \\ \theta \\ \psi \end{pmatrix} \tag{3.27}$$

While PTAM returns direct observations of the pose of the quadrotor with the IMU it is possible to compute x and y relative movements by multiplying rotation matrix of the yaw θ by the linear velocities.

The Kalman filter process can be split in two phases: the predict and the update phase. The predict phase produces an estimation of the state $\hat{\mathbf{x}}_{k|k-1}$ at the step k.

Since f and h are differentiable, as required for an Extended Kalman Filter, the Jacobians $\mathbf{F}$ and $\mathbf{H}$ of those non linear functions are used to update the linearized model.

$$\begin{aligned} \mathbf{F}_{k-1} &= \left.\frac{\delta f}{\delta \mathbf{x}}\right|_{\hat{\mathbf{x}}_{k-1|k-1}, \mathbf{u}_{k-1}} \\ \mathbf{H}_k &= \left.\frac{\delta h}{\delta \mathbf{x}}\right|_{\hat{\mathbf{x}}_{k|k-1}} \end{aligned} \tag{3.28}$$

Prediction is evaluated with the following linear system.

$$\begin{aligned} \hat{\mathbf{x}}_{k|k-1} &= f(\hat{\mathbf{x}}_{k-1|k-1}, \mathbf{u}_{k-1}) \\ \mathbf{P}_{k|k-1} &= \mathbf{F}_{k-1}\mathbf{P}_{k-1|k-1}\mathbf{F}_{k-1}^{\mathrm{T}} + \mathbf{Q}_{k-1} \end{aligned} \tag{3.29}$$

Update allows then to combine the current observation to adjust the previously computed state estimation

$$\begin{aligned}
\tilde{\mathbf{y}}_k &= \mathbf{z}_k - h(\hat{\mathbf{x}}_{k|k-1}) \\
\mathbf{S}_k &= \mathbf{H}_k \mathbf{P}_{k|k-1} \mathbf{H}_k^{\mathrm{T}} + \mathbf{R}_k \\
\mathbf{K}_k &= \mathbf{P}_{k|k-1} \mathbf{H}_k^{\mathrm{T}} \mathbf{S}_k^{-1} \\
\hat{\mathbf{x}}_{k|k} &= \hat{\mathbf{x}}_{k|k-1} + \mathbf{K}_k \tilde{\mathbf{y}}_k \\
\mathbf{P}_{k|k} &= (I - \mathbf{K}_k \mathbf{H}_k) \mathbf{P}_{k|k-1}
\end{aligned} \tag{3.30}$$

where $\tilde{\mathbf{y}}_k$ is the innovation of the measurement, $\mathbf{S}_k$ is the covariance of the innovation, $\mathbf{K}_k$ is the Kalman gain that expresses how much the innovation influences the updated state, $\hat{\mathbf{x}}_{k|k}$ is the updated state and $\mathbf{P}_{k|k}$ is the updated estimated covariance.

3.3 Autonomous exploration

Once we have knowledge on the pose of the drone, we can consider the second phase of our project that is the study of a solution for exploration problem. Taking into account the goal of our project we have absolute absence of information about shape and size of the scene and we have no possibility of human intervention after quadrotor takes off. Moreover, we have not used way-markers or custom markers to guide the drone and we have not assumed that all internal areas or object are visible or accessible.

The main idea is to make the drone able to explore a room in an intelligent way, behaving like a human being, considering the nature of PTAM system that is developed to work using a handheld camera as data source. The drone has to face problematic situations, as aforementioned, such as lost PTAM tracking due to sudden changes in the framed scene.

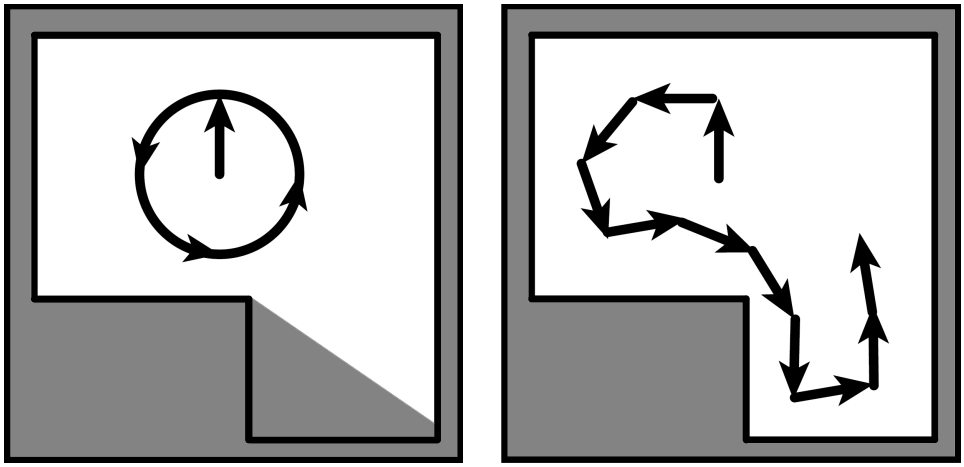
Concerning exploration, the drone has to direct to areas that increase knowledge of the system, but which are not too close to objects or obstacles because we have to consider the nature of the hardware (e.g., it drifts).

In addition, we have to make sure that PTAM maps as many parts of a room as possible finding the greatest number of points possible.

As previously mentioned, the PTAM algorithm needs always movements composed of only translation or translation and rotation, in order to increase the number of keyframes catched (and consequently visual knowledge).

Considering this requirement, we have devised two principal approaches to the exploration problem.

The first approach consists in the execution of default pose sequences to reconstruct an approximate environment. Each sequence has been composed



(a) (b)

Figure 3.8: Example of a generic: a) static path approach. b) dynamic path approach.

of commands which generate a combination of translations and rotations or translation-only in a range of few decimeters.

Once the drone has completed the sequence, it elaborates a possible pose target in order to increase the chances of finding new features and to expand knowledge.

This first static approach has been chosen for its simplicity taking into account the aforementioned requirements, but it is not reliable enough to be used in our system.

Considering a general scenario the result of this approach is unsatisfactory because it depends on many factors such as the starting position or shape and size of the room (figure 3.8). These factors condition the reconstruction of real world point in virtual space because PTAM maps features and it gets keyframes from different distance surfaces. Moreover, we can not predict if the quadrotor completes all steps of the static sequence.

Therefore we have considered a second approach that we have called dynamic. This method does not use any default sequence of commands but it determines, after the initialization process, the next command and target to reach dynamically.

Static exploration strategies are explained and compared to dynamic one in Chapter 5.

3.3.1 Knowledge extraction

Once a visual SLAM system (PTAM) and a strategy (between static and dynamic) of exploration are chosen we have to describe how they have to collaborate to extract knowledge from surroundings. In fact, exploration is impossible if the quadrotor position and environment description are not known to Exploration manager, moreover, knowledge increment can not be expected without exploration.

Furthermore, considering SLAM manager and Exploration manager in figure 3.1 as those which have respectively the task of localization/mapping and exploration strategy computation, the main connection between them is represented by the communication, from SLAM manager to Exploration manager, of the drone pose and the positions of features (organized in a point cloud). We just need this information to extract knowledge of the quadrotor surroundings.

Firstly, some considerations about the data coming from PTAM have to be done. In fact, PTAM algorithm, during feature extraction, does not consider their real nature (that can represent a specific object or, for example, a tile on the ground), but it searches and tracks features which are only patches of the framed scene that satisfy certain requirements described in Chapter 3.2. Therefore, in pose estimation and three-dimensional reconstruction of the environment, PTAM algorithm has to consider all found features with the same weight in order to make more accurate estimates. So, in one hand, we can not remove the found features from inside of the PTAM system and, in the other hand, we need to consider only useful features coming from the exploring process. To make this possible, the Exploration manager filters the feature positions sent by the SLAM manager. This filtering process makes the Exploration manager able to consider only reliable points, which removing isolated and unfitting points. In particular, we have chosen to remove, firstly, isolated points since obstacles, represented by objects, are very unlikely to be represented by only a feature. Subsequently, we have considered the points located in areas where the quadrotor can not fly (e.g. under fixed altitude) as unfitting and they have been removed. Unfitting points (e.g. points found on the ground) do not have to be considered as obstacles because the drone fly-zone is over them.

Once a dependable point cloud, composed of reliable points sent by PTAM, is ready we have studied a possible representation of data obtained from the cloud in order to have a solid basis to begin exploration.

In details, from point cloud information we did have to got robust and useful spatial descriptions of the quadrotor surroundings to be able to use these descriptions in appropriate short-term and long-term planning and decision-making activities.

This is possible with the discretization of all the information about surroundings.

One approach to discretize point clouds is called *voxelization*. It consists of a simplification process of a three-dimensional point cloud into another one which consists of different units called voxels[3] representing a feature in a three-dimensional space. Voxelization is often used to discretize very dense point clouds. This process allows to reduce point clouds in better organised structures, in fact the space is divided into voxels and only one three-dimensional point per voxel is represented in the resulting output.

Nevertheless, voxelization is not very useful for map exploration since it does not give information about the density of the points in the space and, as thereafter described, we rely on the density distribution of points to describe obstacles.

In contrast, another approach to discretize point clouds is using occupancy grids. The main idea of the occupancy grid method is to use a probabilistic tessellated representation of spatial information. It is possible to consider an occupancy grid as a two-dimensional representation describing a slice of three-dimensional world.

In our case, we have used an occupancy grid to describe the world that the drone can explore without obstacles collisions because, unlike the voxelization method, the density of the points in three-dimensional space is considered.

The representation of occupancy grid is a two-dimensional tessellation of environment into cells (like a grid pattern) where each cell contains a stochastic estimate of its occupancy state.

We can consider the following equation that expresses posterior probability of every cell in the map computed from occupancy grid algorithm:

$$p(m|z_{1:t}, x_{1:t}), \quad p \in [0, 1] \tag{3.31}$$

In this representation, m is the map, $z_{1:t}$ is the set of sensor data during time and $x_{1:t}$ is the set of robot, or drone, poses from time 1 to time t. We can denote m_{ij} as the grid cell and $p(m_{ij})$ as the probability that cell (i, j) is occupied.

[3] VOlumetric piXEL.

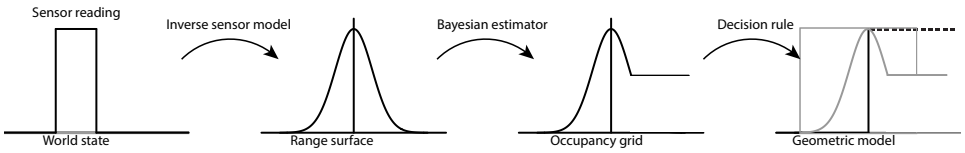


Figure 3.9: Occupancy grid algorithm steps.

Traditional approach to grid estimation consists in interpretation of the range data obtained from a sensing device through stochastic sensor model defined by probability density function $p(m|z_{1:t}, x_{1:t})$ before mentioned (figure 3.10). The probability of each cell is determined by Eq. 3.32.

$$p(m_{ij}|z_{1:t}, x_{1:t}) = \frac{e^{l_{t,ij}}}{1 + e^{l_{t,ij}}} \tag{3.32}$$

$$l_{t,ij} = l_{t-1,ij} + \text{logit}(p(m_{ij}|z_t, x_t)) - l_0 \tag{3.33}$$

$$l_0 = \text{logit}(p(m_{ij}) \tag{3.34}$$

$$\text{logit}(p) = \log \frac{p}{1-p} \tag{3.35}$$

The use of the logit function avoids numerical instabilities for probabilities near zero or one values. $p(m_{ij}|z_t, x_t)$ in Eq. 3.33 is called *inverse sensor model* and it is distributed, in case of Gaussian sensor, as $N(\mu; \sigma^2)$ where μ and σ are values returned by the sensor and they represent, respectively, the discrete observation of the reflecting surface (obstacle) and the standard deviation that is proportional to the distance between the obstacle and the sensor. The *inverse sensor model* is used through Bayesian filter procedure to incremental update the occupancy grid cell state probabilities (Eq. 3.33).

Finally, a deterministic world model is obtained using optimal estimators such as the maximum a posteriori (MAP) decision rule to assign discrete states to the cells, labeling them *empty, occupied and unknown*.

In contrast with the traditional approach we have opted for an heuristic approach.

The main reason for our choice is represented by the information given to create of the grid. Instead of the set of sensor data, which is continuous, we have used a filtered set composed of points, organised in a point cloud, obtained from features found by PTAM. For this reason, we have not to make any probabilistic sensor model or sensor integration since all information is already filtered and merged (figure 3.11). Moreover, to estimate the occupancy probability of every cell in the grid, we have considered the

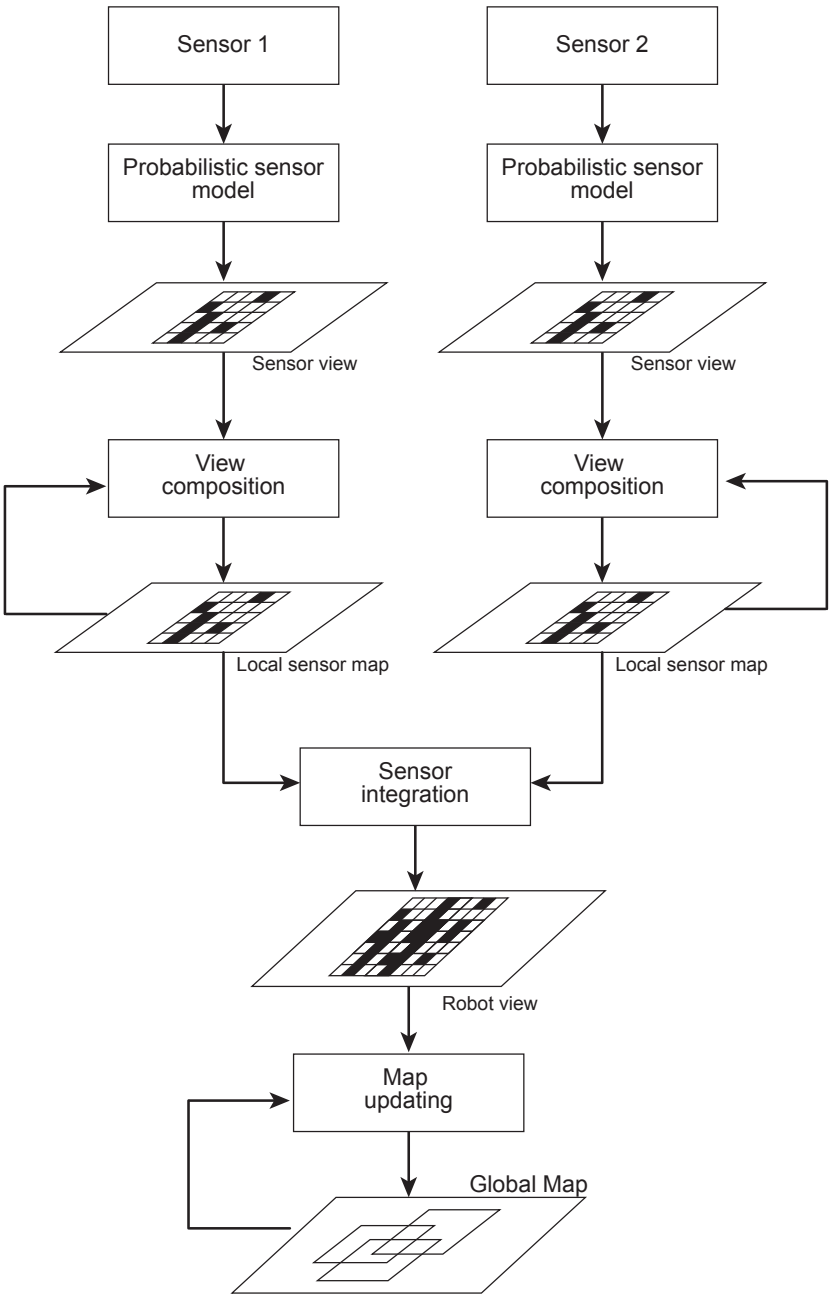


Figure 3.10: Original occupancy grid algorithm.

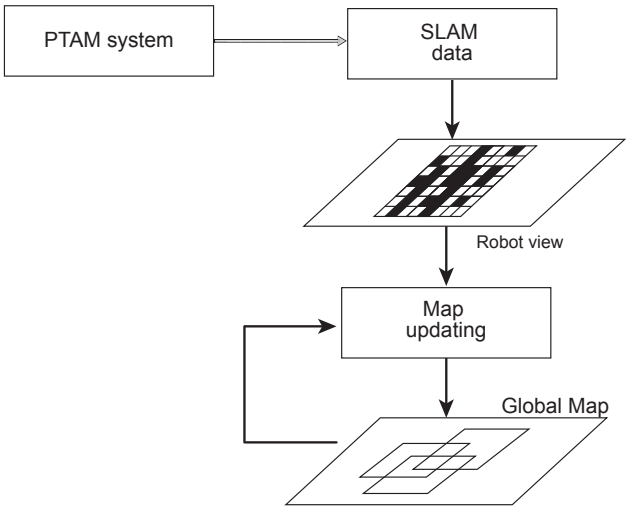


Figure 3.11: Our occupancy grid algorithm.

density projected points in each cell taking into account that points position estimate is already the optimal one (see section 3.1.1).

We chose to classify the cells on the grid depending on the estimated number of points present in each cell as it is described in Chapter 5.

Safe flight exploration

One of the main differences between a wheeled robot and an UAV concerns motion. While a wheeled robot reliably responds to control commands an UAV has difficulties in completing a movement without significant drifts. This happens because the static friction between ground and robot wheels is significantly higher than fluid friction of a flying object in the air medium resulting in a more responsive control of a ground vehicle. Moreover, an UAV has no reliable sensors in order to close the control loop since. Unlike sensors on wheels that could be used to recover reliable movements data from wheel speed, the UAV rotor sensors could not be used because they return angular velocities that change roll, pitch and yaw angles of the drone conditioning UAV change of position (the movement is not directly observed).

Concerning this problem we have been forced to study a system that made the quadrotor able to fly safely preventing the impact with objects or walls.

Therefore, we have made the drone fly at a distance of at least 1.2 meters from any known point (feature).

To make our system able to keep the drone at a safe distance from obstacles (e.g. walls and other objects), a distance map has to be computed. This distance map consists in a map with the same size of the obstacle (occupancy) map in which is stored in every cell the distance between the cell and the nearest obstacle.

Distance calculation can be done extensively by computing, for each point, all distances (metric can be chosen arbitrarily) from each occupied point in the map and keeping the minimum of these. Nevertheless, this process is very expensive in terms of computation complexity, given a $n \times n$ binary grid the time complexity results in $\mathcal{O}(n^4)$ (for all the point in the grid all distances to occupied cell are computed keeping the minimum one). This complexity can be reduced to $\mathcal{O}(n^2 * m)$ where m is the number of occupied cells in the grid, by initially inspecting the grid and extracting the occupied cells in a vector (this passage allows to compute distances without iterate continuously on the grid, but on the vector of occupied cells).

A possible solution of decreasing this time complexity is to use Voronoi diagram. A Voronoi diagram is a particular diagram that, given a set of m points called seeds, decomposes the space in regions (called cells), which consist of the set of points that are nearer to the cell seed rather than all other seeds.

The problem to build the Voronoi diagram is solved with Fortune's algorithm [54] with the time complexity of $\mathcal{O}(m \log m)$. Fortune's algorithm computes (with the same time complexity) also the Euclidean distance grid (called distance transform) of the given map.

In robotics the distance transform (distance metric can be chosen arbitrarily between for example Euclidean, Manhattan or Chebychev) is also called the Extended Voronoi Transform. Fortune's algorithm cannot be applied in a grid of obstacles because it requires a set of punctual elements. In fact, in an obstacle grid, it is possible to have contiguous occupied cells. To compute the exact distance transform of an occupancy grid, extensive search for all distances has to be done. Nevertheless, different algorithms that can compute an approximation of the distance transform have been considered. These algorithms compute the Extended Voronoi Transform.

Considering a binary matrix where elements occupied are represented by ones and free space is represented by zeros, it is possible to consider some

algorithms [55] such as CDA[4] [56], Chessboard[5], City Block[6] or DRA[7]. In our application, where we have to be really careful to stay away obstacles, we need a good approximation of Euclidean distance which is only given by CDA or DRA. To evaluate the distance transform these algorithms consider a window of cells. Thus, the distance value of each cell in the map depends on distance value of the neighbour cells located inside the window that usually is 3x3 or 7x7.

For our purposes, we have chosen DRA, that is a straightforward modification to the CDA, since it, employing only a 3x3 window, typically produces more accurate results than CDA with a 7x7 window with similar execution time [55].

3.3.2 Knowledge increase methods

Once we have solved problem about environment information filtering and discretization of the scene, we have faced the knowledge increase problem.

Starting from PTAM system it is necessary to make some considerations on the heuristic used to take new key-frames. The heuristics require a translation of the drone and not only a rotation on itself. Without translation movement, the PTAM algorithm returns incorrectly estimated points that distribute on a conic shape (like in figure 3.13). Moreover, as mentioned in section 3.1.1, the PTAM algorithm always needs a good feature tracking to catch a new keyframe and, consequently, to increase the knowledge.

The heuristic to take a new keyframe considers also the depth of the scene. This evaluation can be done by estimating the average depth of reprojected key points in the current frame. If points are on average far from the camera position, a short change of this one does not affect too much the framed scene so a new keyframe is not needed. By contrast, when the camera is closer to reprojected key points also small movements drastically change the currently seen world, therefore a new keyframe is saved.

[4]Chamfer Distance Algorithm.
[5]CDA variation that uses Chebychev distance metric.
[6]CDA variation that uses City block distance or Manhattan distance.
[7]Dead Reckoning Algorithm.

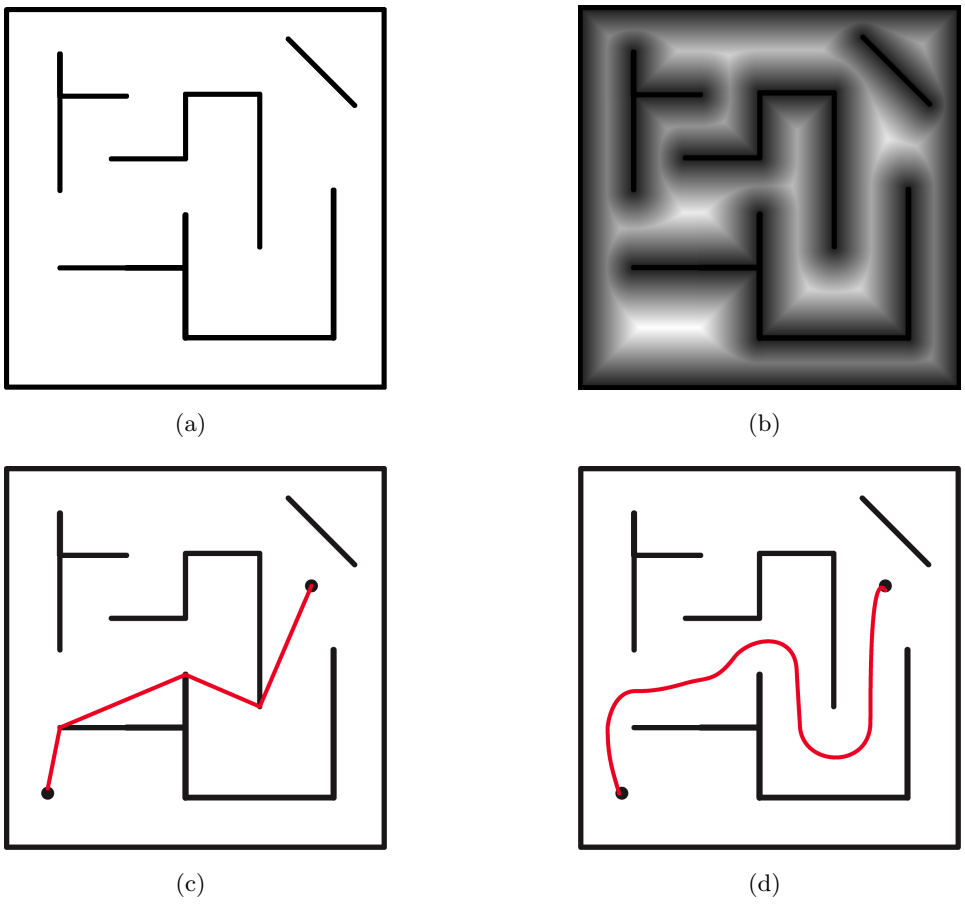


Figure 3.12: a) Example of a map to explore. b) Distance map compute with distance transform. c) Path calculated with fast marching algorithm. d) Path calculated with fast marching over distance map.

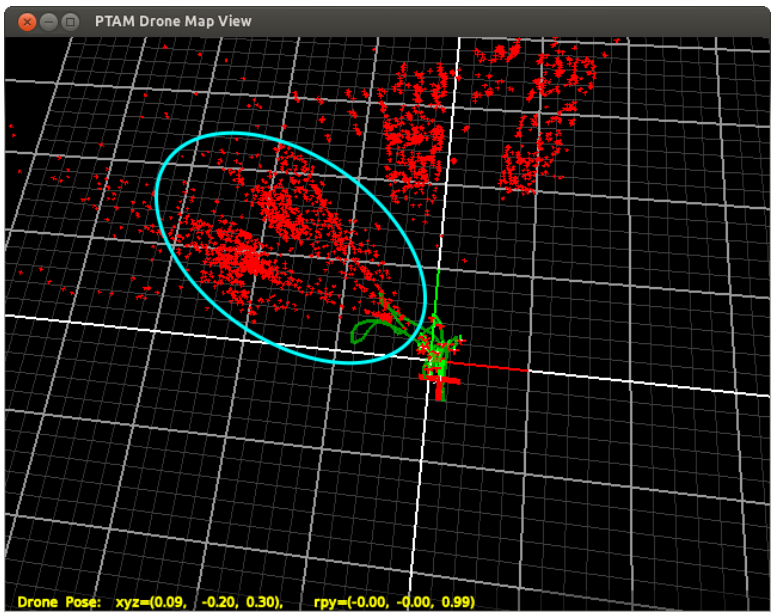


Figure 3.13: PTAM error due to only rotation.

Taking into account these considerations we have chosen to develop a mixed exploration method. This union consists in the *frontier based* and the *view improvement* methods completed with a random strategy method used when our strategy remains locked on a particular state.

Frontier based

The frontier based is an approach that uses concept of regions, called frontiers, on the unknown space fringe. The central idea behind this approach is to drive the robot always to boundary between visitable region and unexplored space in order to get more and more information about the environment.

Being more precise, it is possible to divide this process into steps: frontier detection and navigation to frontier. As aforementioned, once the occupancy grid is created, each grid cell is categorised by its probability to be occupied and labeled as free, occupied or unknown.

Frontier detection is possible through a process similar to blob extraction and edge detection in computer vision. From a set of occupancy grid cells regions that differ in properties are detected. Then, each cell adjacent to an unknown cell, is labeled as frontier region cell. The union of adjacent frontier region cells is called frontier region. At the end, the set of regions

are filtered by a fixed minimum size that, in our project, is around 10-15 cm larger than quadrotor width.

Once the set of frontiers has been built, the quadrotor is sent to the nearest accessible frontier. A target position is computed choosing, in the set of frontiers (0,1,2 in figure 3.14 (c)), the one with the shortest obstacle free path between it and the current position of the robot. An example of the method used for path computation is the Fast Marching algorithm (figure 3.12 (c)).

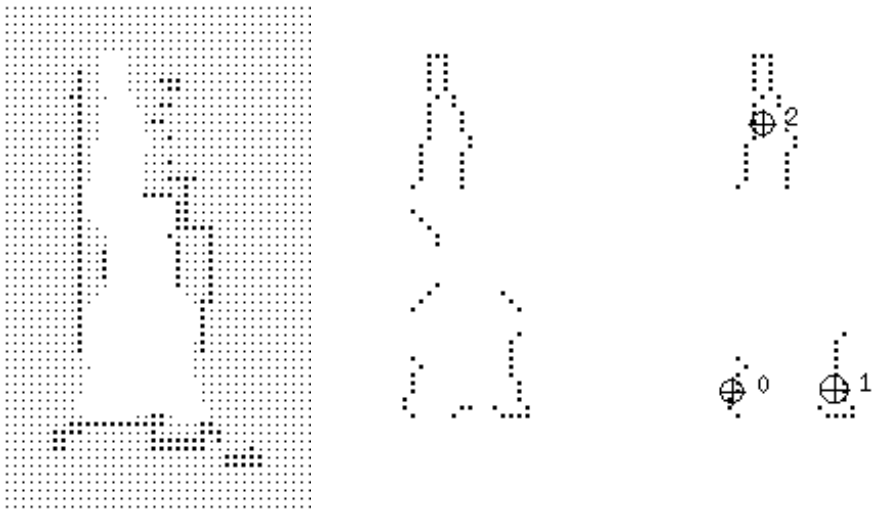


Figure 3.14: Frontier-based strategy steps. (a) occupancy grid, (b) frontier edge segments, (c) frontier regions.

A frontier based approach is used, in our project, to decide the next target position that the robot has to reach. This position is computed taking into account two distances: the one between obstacles and the target position, and the other one between the frontiers and the target position. The implementation details of this approach are described in Chapter 5.

The frontier-based method returns only a two-dimensional position in the grid, but we have to decide, due to limited field of view of the quadrotor camera, also the orientation angle of the drone.

View improvement

Another strategy to make a mobile robot able to perform an exploration in a 2D plane is called view improvement. This strategy fits well in concave room or furnished rooms and it is designed for limited field-of-view vision systems.

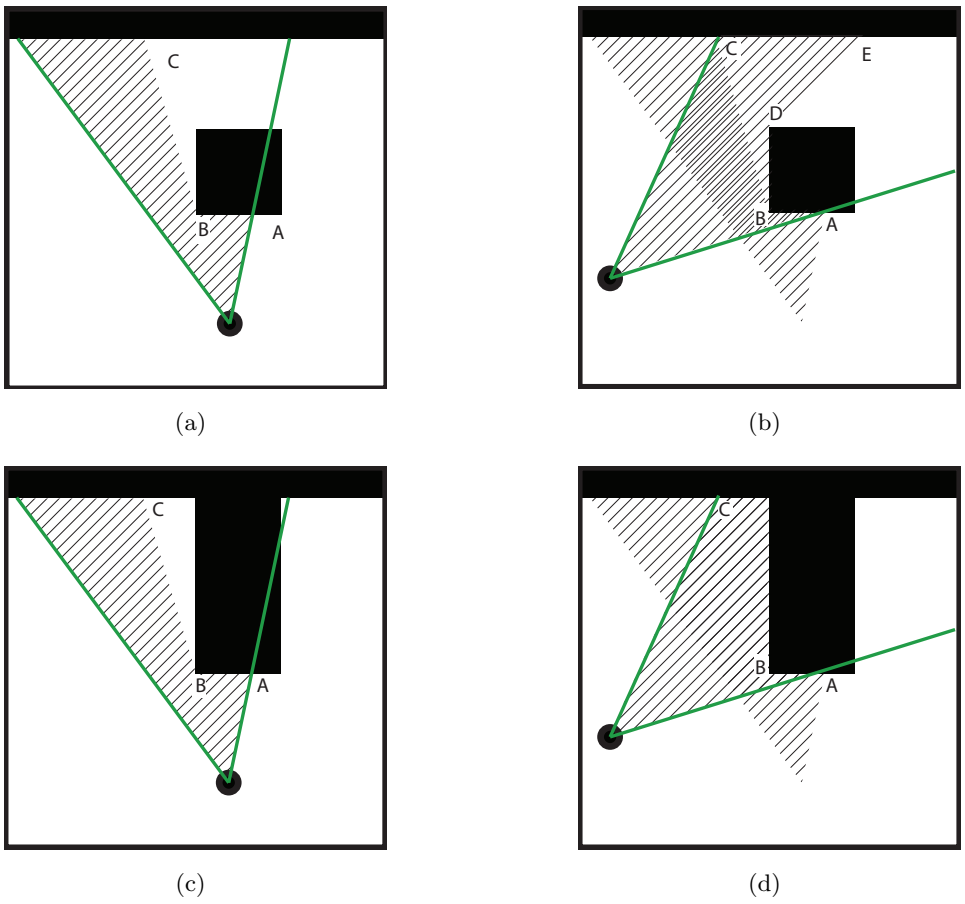


Figure 3.15: Example of a view improvement strategy approach.

Assuming that in real world the presence of obstacles or pillars can occlude unseen features, it is possible to define a robot behavior to increase its knowledge by trying to look over these objects. Seen objects are not necessarily a boundary for exploration, they may only hide some areas in current field of view. Nonetheless, it is possible to expand knowledge by turning around the obstacle.

Taking two found objects that are distant in the real world but near in the view of the robot, the algorithm returns a strategy that consists in

moving to an area equidistant from those obstacles taking into account the robot field of view in order to frame both obstacles.

In figure 3.15 (a) is shown a sensor field of view and (visual boundaries) features (B and C) that are near the sensor projection but far in two dimensional plane. Considering that between B and C features there could be unseen explorable areas, the next target position and orientation of the robot is chosen between the positions on the axis of B-C segment so that B and C features lie in the field of view of the robot.

This movement could reveal new features and areas to be explored (like in figure 3.15 (b)) or enclosed space that completes the region exploration (figure 3.15 (c)).

In our case, we have taken the cue from this exploration method since the camera of the quadrotor has a limited field of view. Considering the position computed with a frontier based approach, our system elaborates the orientation by pointing at known features detected from the target position as visual boundary.

In our project, while the quadrotor moves toward its target destination, the behavior system can reactively detect obstacles in order to prevent collisions with any object not found during the occupancy grid construction. These behaviors are strictly necessary in a dynamic scenario to avoid collisions with objects lacking features like monochromatic tables or chairs.

Chapter 4

System architecture

"Buy it, use it, break it, fix it,
Trash it, change it, melt - upgrade it,
Charge it, pawn it, zoom it, press it,
Snap it, work it, quick - erase it,
Write it, cut it, paste it, save it,
Load it, check it, quick - rewrite it"

Technologic, Daft Punk

In this chapter we describe hardware and software components used to develop our project.

4.1 Hardware component

The hardware platform consists of two components. The first component, a personal computer[1], communicates with a second one, a quadrotor UAV[2] through wireless connection.

4.1.1 AR.drone

Quad-rotor also called quad-copter or quadrocopter, is an helicopter-like aircraft that is lifted and propelled by four rotors [57]. We have used AR.Drone

[1] We have used a laptop computer equipped with 2.4GHz Intel Core 2 Duo running Ubuntu 12.04 and ROS Groovy.

[2] An unmanned aerial vehicle (UAV), commonly known as a *drone*, is an aircraft without a human pilot on board. Its flight is controlled either autonomously by computers in the vehicle, or under the remote control of a pilot on the ground or in another vehicle

2.0 made by Parrot as MAV[3] quadrotor. Compared with other MAV, AscTec Hummingbird or AscTec Pelican, its main advantage is the very low price, its robustness to crashes and the fact that it can safely be used indoor and close to people. AR.Drone 2.0 was unveiled at CES Las Vegas 2012 and it's commercially available from 2012.

Designed as a toy with indoor and outdoor flight capability, AR.Drone has become very interesting for electronics hobbyists and researchers. The main reason of this consideration has been ascribed to its hardware equipment (see next section) combined with its automated actions like take-off, land and stay fairly stable in the same position (hover) while flying that allows users to focus on navigation or image analysis implementations. These features joined with a relatively low price (around €299) and with ability to fly safety in indoor environment make it a valuable research object.

Drone Operation

The mechanical structure (made up carbon-fiber and plastic air-frame) consists of four rotors attached to the four ends of a crossing to which the battery and the RF hardware are attached. One pair of opposite rotors turns clockwise and other pair turns anti-clockwise as show in figure 4.1.

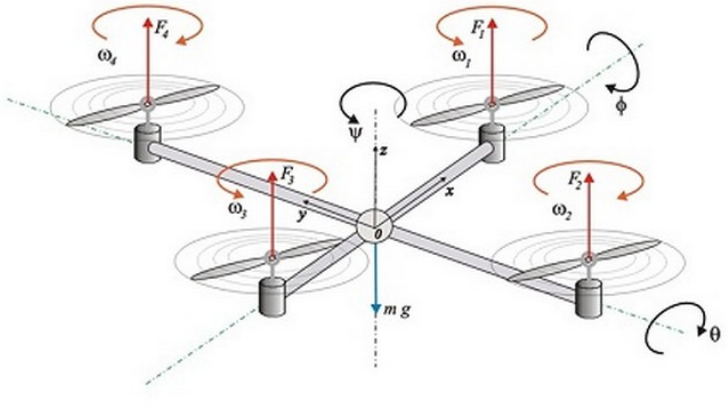
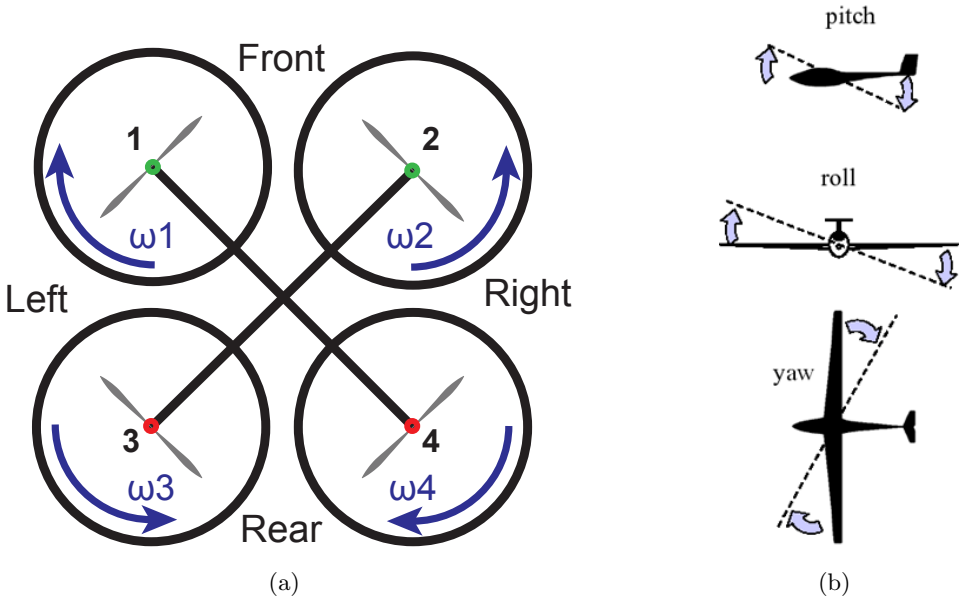


Figure 4.1: Rotors of drone turning

Consider ω_1, ω_2, ω_3 and ω_4 as the angular speeds of each motor (figure 4.2 a). To stay flying in the same position (hovering), the drone has to keep all the angular speeds equal. To go in the direction of axis **x**, **y** and **z**, the

[3]A micro air vehicle (MAV), or micro aerial vehicle, is a class of unmanned aerial vehicles (UAV) that has a size restriction and may be autonomous.

drone changes the angular speed of rotors. To make movements as front, rear, left and right translations and rotations the drone has to change pitch, roll and yaw angle. To understand what these angles mean, see the roll, pitch and yaw angles in a plane (figure 4.2 b).



(a) (b)

Figure 4.2: a) Drone movements scheme. b) Explanation of pitch, roll and yaw angles in a plane.

To change the pitch angle, the drone has to modify angular speeds of front pair motors (4.1). To increase or decrease roll angle drone has to alter angular speed of left pair and right pair motors (4.2). To modify yaw angle in order to rotate clockwise or anticlockwise on itself drone has to change angular speed of pair of each pair of opposite motors by the same amount (4.3). Finally, to go upward and downward, the drone has to increase or decrease all angular speeds of the drone by the same amount (4.4).

Pitch angle variations:

$$\begin{cases} \omega_1 = \omega + \Delta_A \\ \omega_2 = \omega + \Delta_A \\ \omega_3 = \omega - \Delta_B \\ \omega_4 = \omega - \Delta_B \end{cases} \tag{4.1}$$

Front (rear) translation with $\Delta_A, \Delta_B \in \mathbb{R}^+$ ($\Delta_A, \Delta_B \in \mathbb{R}^-$)

Roll angle variations:

$$\begin{cases} \omega_1 = \omega - \Delta_B \\ \omega_2 = \omega + \Delta_A \\ \omega_3 = \omega - \Delta_B \\ \omega_4 = \omega + \Delta_A \end{cases} \tag{4.2}$$

Left (right) shift with $\Delta_A, \Delta_B \in \mathbb{R}^+$ ($\Delta_A, \Delta_B \in \mathbb{R}^-$)

Yaw angle variations:

$$\begin{cases} \omega_1 = \omega + \Delta_A \\ \omega_2 = \omega - \Delta_B \\ \omega_3 = \omega - \Delta_B \\ \omega_4 = \omega + \Delta_A \end{cases} \tag{4.3}$$

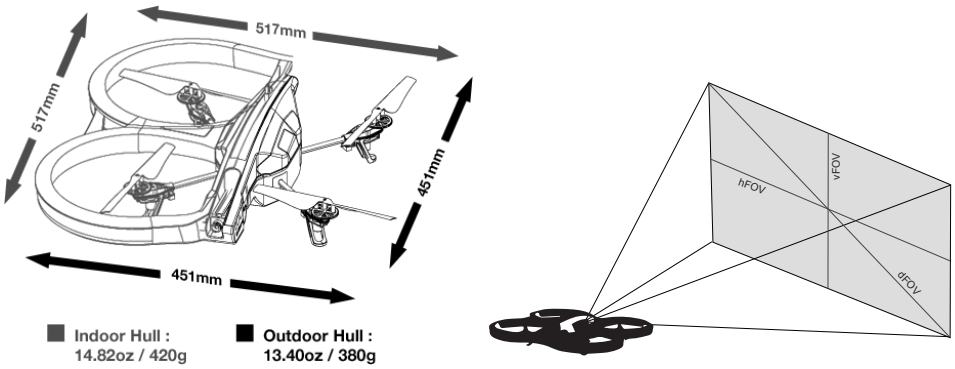
Clockwise (anticlockwise) rotation with $\Delta_A, \Delta_B \in \mathbb{R}^+$ ($\Delta_A, \Delta_B \in \mathbb{R}^-$)

Gaz[4] variations:

$$\begin{cases} \omega_1 = \omega + \Delta_G \\ \omega_2 = \omega + \Delta_G \\ \omega_3 = \omega + \Delta_G \\ \omega_4 = \omega + \Delta_G \end{cases} \tag{4.4}$$

Upward (downward) translation with $\Delta_G \in \mathbb{R}^+$ ($\Delta_G \in \mathbb{R}^-$)

The drone uses a 3 cell 1000 mAH LiPo rechargeable battery designed by Parrot. A fully charged battery gives power for about 10-14 minutes of continuous fly.



(a) (b)

Figure 4.3: a) Dimensions of Parrot AR.Drone with outdoor and indoor hull. b) Outline of drone camera field of view.

[4]The vertical speed (called by Parrot "gaz") argument is a percentage of the maximum vertical speed as defined here. A positive value makes the drone rise in the air. A negative value makes it go down [58].

Cameras and sensors

AR.Drone includes some components useful for the project such as:

- Front camera: Parrot equips its drone with a sensor with 93° diagonal field of view (dFOV), 81° horizontal field of view (hFOV) and 45° vertical field of view (vFOV) (Figure 4.3). Exposure compensation is automatic as well white balance and camera can recorder at 720p but stream video only at 640x360 resolution (nHD[5]) up to 30fps.

- Vertical bottom camera is a QVGA sensor equipped with 64 degree diagonal field of view, 60fps

- Ultrasound altimeter enhanced with additional air pressure sensor

- IMU (Inertial Measurement Unit): reports velocity and orientation, using a combination of accelerometers and gyroscopes allowing for more stable flight and hovering

- On board computer, running Linux 2.6.32, equipped with 1GHz 32 bit ARM Cortex A8 processor

- Interfaces: USB, SD memory card, and Wi-Fi 802.11n

- Indoor hull: 517mm x 517mm hull made of EPP (Expanded Polypropylene) (figure 4.3 a)

Altitude measures, used for automatic altitude stabilisation and assisted vertical speed control, are provided by ultrasound telemeter ($0 \sim 6$ meters) and air pressure sensor (over 6 meters). Bottom (vertical) camera is needed by AR.Drone 2.0 for ground speed measurement and automatic hovering and trimming through optical flow estimation[6].

The two most important components for realization of our project are front camera and IMU. We have used Inertial Measurement Unit data to reconstruct real world movements of drone to our virtual environments as previously described. Moreover, technical characteristics of the front camera are relevant to analyze reasons behind exploration strategy choice.

[5]Ninth of FullHD resolution

[6]Detailed description of what *optical flow* algorithm is and how it works can be found in chapter 3.

Figure 4.4: AR.Drone components

4.2 Software

The software developed in this thesis consists of independently running processes. These processes communicate with each other in an event driven architecture based on message passing.

This kind of modular architecture is worth during and after system development. In fact, with this approach it is possible to break a complex problem (such as exploration problem) into simpler tasks making easier to update, debug and modify every single part of the system. Moreover, it allows interoperability with other packages and easier future enhancements.

We have chosen Robot Operating System (ROS)[59] as software platform in line with our idea to develop a system as modular as possible. ROS is a framework and a toolbox for the development of robot applications. The framework has a lot of features including hardware abstraction, device drivers, libraries, visualisers and the aforementioned message-passing. The toolbox is very extensive and we have used many utilities from it. Some

important ROS tools and packages for our work are *RViz*, *rosbag*, *pcl_ros*, *ardrone_driver* and *tum_ardrone*.

RViz is a 3D visualisation tool. The strong point of this tool is that it can display many kind of messages, like pose or point positions, as soon as they are published. In figure 4.5 it is possible to see an empty view of the visualiser.

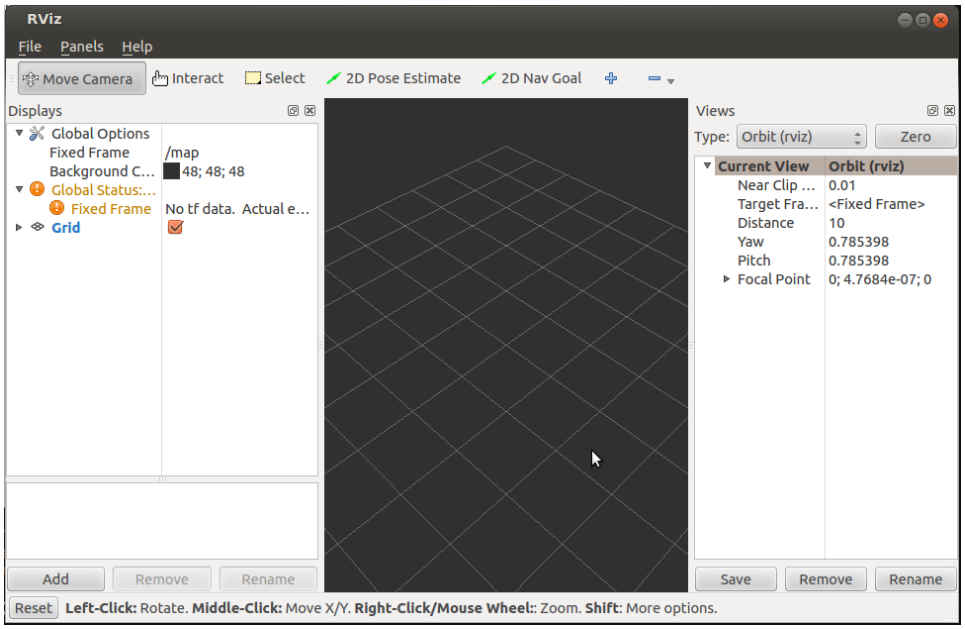


Figure 4.5: RViz window

rosbag is a package of tools for recording messages from and playing back them to ROS topics. It is an useful debugging instrument during the development life cycle of a project. It allows to record a pack of all the needed messages for the correct operation of the developed system and to make tests in safety conditions in a different afterwards.

PCL[60] is a standalone open-source framework developed by Willow Garage[7]. The library is composed by different modules which implement state-of-the art algorithms to manipulate three-dimensional point clouds[8]. Some relevant modules for the purposes of the project are *filters* and *kdTree* which are helpful to remove noise or outliers point. Storing point cloud in kdTree structure allows to decrease complexity of algorithms. In order

[7]Willow Garage is a company of researchers and engineers around the world devoted to developing hardware and open source software for personal robotics applications

[8]A point cloud is a set of data points in some coordinate system.

to take advantage of PCL in ROS based applications, developers have to utilise *pcl_ros*. This package allows to translate PCL data from and to ROS messages.

4.2.1 Communication services between AR.Drone and devices

Management of AR.Drone is done via three main communication services.

All information data such as battery level, its status, engine rotation speed or IMU data, are sent by the drone to its client on UDP port 5554 [9]. This information, called *navdata*, also include tags detection result that can be used by developer to create applications as augmented reality games. They are sent by default approximately 30 times per second.

Video stream is sent by the AR.Drone to client device on port UDP 5555. It is encoded with P264 algorithm[10].

Controlling and configuring the Parrot quadrotor takes place through sending *AT commands* (usually 30 times per second) on UDP 5556. AT commands compose the Hayes command set, a specific command language originally developed for the Hayes Smartmodem 300 baud modem in 1981 [61].

ardrone_autonomy package

To simplify communication between computer and drone we have employed a ROS package developed in Autonomy Lab of Simon Fraser University by Mani Monajjemi called *ardrone_autonomy*[62].

ardrone_autonomy, fork of *AR-Drone Brown driver* is a ROS driver for Parrot AR.Drone. The real advantage is all client-drone communications are translate in ROS topic and ROS service. To read data from drone developer can implement a subscriber to `ardrone/navdata` topic. In this topic, `ardrone_autonomy` publishes an on purpose message[11] containing navdata of drone. Sending commands to AR-Drone may be categorised in two classes. In order to allow the drone to take off, land or emergency stop/reset we have to publish an empty[12] message to `ardrone/takeoff`, `ardrone/land`

[9] Devices can connect to the drone through a 802.11g connection.

[10] The encoding concepts involved comes from the H264 specification. Like H264, P264 makes a spatial prediction for each macro-block based on the neighbouring pixels. Please refer to the *Recommendation ITU-T H.264* for further details

[11] Message type: `ardrone_autonomy::Navdata`.

[12] Message type: `std_msgs::Empty`.

and `ardrone/reset` topic respectively. Once the quadrotor is flying we can publish a message[13] to the `cmd_vel` topic to move it. Messages published to `cmd_vel` are translated in terms of pitch, roll and yaw angles by driver of AutonomyLab.

tum_ardrone package

tum_ardrone[5] is a package developed for Parrot AR.Drone and AR.Drone 2.0. This system enables a low-cost quadrocopter coupled with a laptop to navigate autonomously, after fixed coordinates, in previously unknown environments without GPS sensor.

The package consists of three components: a monocular SLAM system based on PTAM[6] algorithm, an extended Kalman filter for position estimation and data fusion and a steering commands generator based on PID controller (*control_node*).

Concerning the operation of the package under consideration it is possible to control the drone through a graphical user interface (shown in figure 4.6). The interface allows users to create custom scripts containing positions in terms of three dimensional coordinates and angle of rotation[14] that describe the path the drone has to follow. As mentioned in chapter 3, to localize the camera of the drone and to map the environment we choose to use PTAM system coupled with an Extended Kalman Filter. Therefore, we have chosen to use this package as base of our SLAM system since it implements the original implementation of PTAM system suited to work with the AR.Drone camera and an Extended Kalman Filter that computes the reliable position of the drone in case of PTAM failures.

[13] Message type: `geometry_msgs::Twist`.

[14] Ready to use scripts are available to show simple performance. They command the drone which draws shapes such as vertical/horizontal rectangle or simple house.

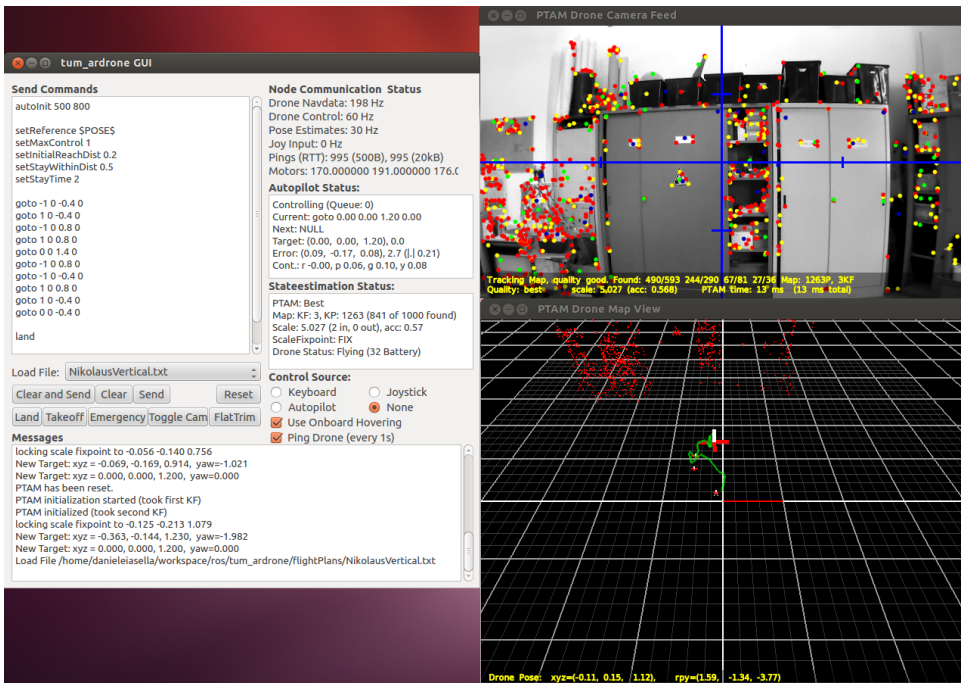


Figure 4.6: Tum ardrone interface.

Our package

Our ROS package consists in two principal nodes: *communication_node* and *exploration_node*.

The main task of *communication_node* is to manage states in which the AR.Drone is and to communicate with *control_node* in order to send commands and to receive information about poses of the drone.

The flight of the drone has to necessarily be scanned into phases in order to know always what the quadrotor is doing and not only its position and its orientation. We have defined four phases: HOVERING, MOVING, PTAM_TRACKING_LOST and PTAM_TRACKING_RECOVERING.

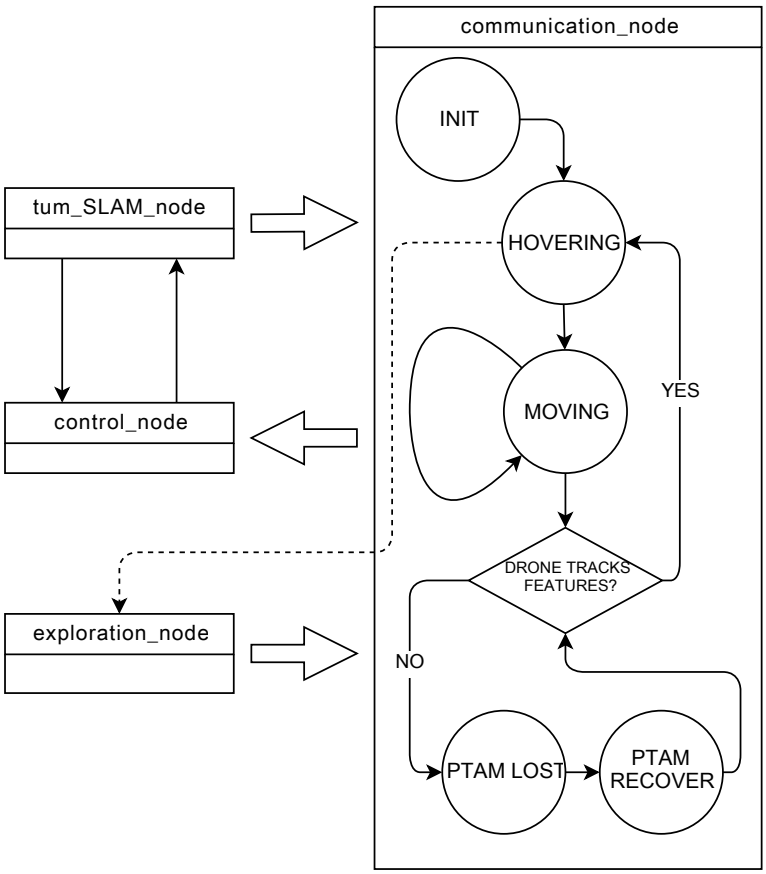


Figure 4.7: Diagram of states of the drone.

Once the initialization routine, which includes drone take off and PTAM initialization, is completed, the drone enters in HOVERING state. It waits in this state until a new task is received. Then it starts to move entering in MOVING state until the task is completed and the target position is reached.

Traditionally in presence of constant knowledge about the quadrotor position is possible to consider only these two states, but in case of position data loss due to, for example, a GPS signal loss or a vision tracking loss, we have to consider also a system that is able to delete received tasks when a position is not provided, preventing to fly without reliable data. When the drone loses PTAM tracking of features trying to reach a target position it enters in PTAM_TRACKING_LOST state. When it happens a warning message containing current position is published to enable other nodes to use this information loss, marking, for example, that position as unvisitable.

PTAM_TRACKING_RECOVERING is strictly consequent to PTAM_TRACK-

ING_LOST. In this state the drone forgets any task that it ought to perform and it moves itself to the nearest position where it is sure to find features. Once the position with known features is reached the drone enters again in HOVERING state.

exploration_node has the task of analyzing data and computing the best strategy, in terms of consecutive target positions and orientations, for autonomous exploration of unknown environment in order to increase every time the knowledge about the room.

exploration_node collaborates with *communication_node* and it starts to work only when the drone is ready for a new target.

As shown in next chapter, *exploration_node* evaluates at each step the best action that is thought to mainly increase the knowledge about the scene. The computation of the best action takes into account the limits of visual SLAM algorithm (such as, for example, high latency map construction).

Chapter 5

Implementation and evaluation

"Brava, brava Caterina,
ti si sono accese le luci!
Stai elaborando, vero?!
Non ti stancare sai cara?!"

Enrico Melotti - Io e Caterina, Alberto Sordi

In this chapter, we describe the communication through ROS messages and the implementation of SLAM and Exploration manager. Furthermore, we illustrate the user interface and how it can be used.

In conclusion, we show some results of the implemented system in real-world trials compared to ground truth planimetry.

5.1 SLAM manager

As mentioned in chapter 3, all data, such as the positions of the drone and of the objects around it, are managed by SLAM manager (figure 5.1). It consists in a localization and mapping system and a command control system. As mentioned in Chapter 4, we have chosen to use the SLAM manager implemented into *tum_ardrone* package. Nevertheless, it does not provide the ability to extract PTAM information such as pose and three-dimensional positions of features, which could allow us to re-use them for our exploration purposes.

Information about characteristic positions and poses of the drone must be extracted from PTAM registrations and EKF. Then, they are transmitted

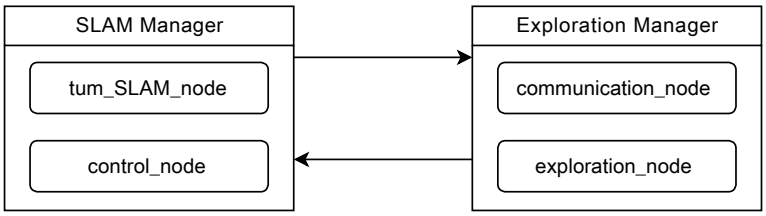


Figure 5.1: Scheme of SLAM and Exploration managers

on ROS topics in order to be read by other nodes. To make this possible some changes to *tum_ardrone* package were needed.

5.1.1 tum_ardrone package modifications

The quadrotor, with the use of *tum_ardrone* package, is not able to perform an autonomous exploration in a three-dimensional environment, therefore we improved this package to allow the machine to autonomously move in three-dimensional space.

This package underwent three main important changes, which concerned the extraction of all SLAM data via message standardization (which is described in the section *5.2*), the communication of the pose of the drone related to a target position, and the heuristics behind the choice of a new keyframe in PTAM system.

Firstly, analyzing the original pose communication system, we found a significant gap consisting of a missing reached target message. A reached target message is a message used to make other ROS packages able to understand when the drone reaches the given target position. Without this kind of message only the *tum_ardrone* knows when a target is achieved.

Therefore, we added the message (containing "u t Target reached" [1]), which is only published when both the pose of the AR.Drone is sufficiently close to the target position, and PTAM pose estimation is good which means that is tracking a sufficiently high number of features as described in Chapter 2.

The second main adjustment regards, in details, the PTAM system implementation that, as mentioned in section *4.2*, is only slightly modified in *tum_ardrone* package, in order to work with the camera of the AR.Drone.

The original PTAM system was created for augmented reality applications in order to not require any markers, pre-made maps or inertial sensors.

[1] The message structure is modeled following the structure used in *tum_ardrone* package.

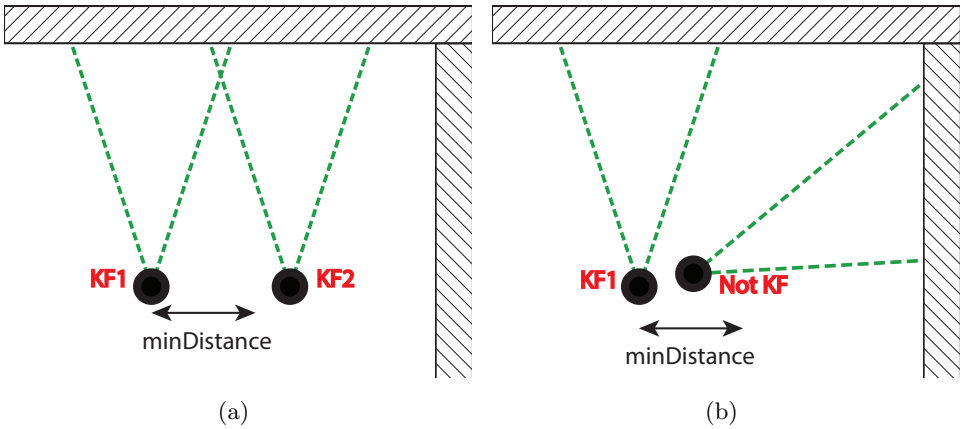


Figure 5.2: Example of failure of original PTAM implementation in our system.

This kind of applications needs single horizontal (or vertical) plane where the PTAM system can detect features in order to draw the three-dimensional models.

The standard implementation of the PTAM system, to acquire a new keyframe, only requires the distance between every keyframe to be higher than a fixed number (20 cm by default). Basically, when a keyframe is recorded at a certain position, every other frame, that could be registered in a radius of 20 cm from that point in the space, is not eligible as keyframe (as shown in figure 5.2 (a)). Hence, using this implementation, the heuristic does not consider any frame in a short radius from a keyframe independently of the orientation at which the new frame is acquired (figure 5.2 (b)). Since we want the AR.Drone to move in a three-dimensional environment, we need to consider as keyframes also those which are close to other keyframes but are recorded at a significantly different orientation.

We, therefore, have developed a novel heuristic that is able to record several keyframes at different directions and at the same distance from a starting point, creating a sphere of information around it. Applying this modified heuristic, every time a new frame is taken, it is only evaluated in comparison with other keyframes that point towards the same direction. The improved PTAM system, thus, only compares a new frame with saved keyframes owning similar orientations (those which differ by $\pm$ 30°) and, if it can not find any nearest keyframe, the system saves it as new keyframe.

5.2 ROS messaging

All communications between nodes in our package are made possible using ROS message-passing. This convention let us focus on our main problem, the exploration problem, leaving to *ros_core* all the assignments such as creation and maintenance of communication channels.

5.2.1 Message standardization

Particular care has been taken regarding standardization of all messaging, thus allowing an easier further development.

Given that *tum_ardrone* package does not manage the PTAM feature position extraction, but it only displays them on a dedicated visualizer, and it publishes all data, such as the drone pose, as an unsuitable String message. All messages sent by *tum_ardrone* PTAM wrapper have been converted to standard ROS messages and new messages have been written following the ROS guidelines [59].

Every ROS message is composed by three main fields: a **header** field, which stores information about the message sequence (all the sent messages are marked with a number that expresses the message count), a **timestamp**, which represents the time of dispatch, and a **frame identifier**, which brings the information regarding the particular frame the message is relative to.

`geometry_msgs/PoseStamped` can be used to communicate the position of the AR.Drone. This message is composed, in addition to the header, of a `geometry_msgs/Pose`, which represents the position and orientation of the drone. In `geometry_msgs/Pose` the position is represented by a `geometry_msgs/Point` which has 3 floating point attributes representing respectively x, y and z coordinates in the cartesian coordinates system. Eventually, the orientation of the robot is identified by a `geometry_msgs/Quaternion` in the `geometry_msgs/Pose` message. Many operations on orientation vectors can be defined using quaternions, therefore they are widely adopted in theoretical and applied mathematics and, in particular, for calculations involving three-dimensional rotations such as in three-dimensional computer graphics and computer vision. In the basic *tum_ardrone* package rotations are expressed in terms of euler angles, therefore they have to be converted to quaternions in order to publish a ROS compliant message.

$$\begin{aligned}
\mathbf{q}_0 &= \cos(\phi/2)\cos(\theta/2)\cos(\psi/2) + \sin(\phi/2)\sin(\theta/2)\sin(\psi/2) \\
\mathbf{q}_1 &= \sin(\phi/2)\cos(\theta/2)\cos(\psi/2) - \cos(\phi/2)\sin(\theta/2)\sin(\psi/2) \\
\mathbf{q}_2 &= \cos(\phi/2)\sin(\theta/2)\cos(\psi/2) + \sin(\phi/2)\cos(\theta/2)\sin(\psi/2) \\
\mathbf{q}_3 &= \cos(\phi/2)\cos(\theta/2)\sin(\psi/2) - \sin(\phi/2)\sin(\theta/2)\cos(\psi/2)
\end{aligned} \tag{5.1}$$

$$\begin{aligned}
\phi &= \text{atan2}(2(q_0 q_1 + q_2 q_3), 1 - 2(q_1^2 + q_2^2)) \\
\theta &= \arcsin(2(q_0 q_2 - q_1 q_3)) \\
\psi &= \text{atan2}(2(q_0 q_3 + q_1 q_2), 1 - 2(q_2^2 + q_3^2))
\end{aligned} \tag{5.2}$$

A trail of the robot is also sent through a `nav_msgs/Path`; this message is a vector which includes all the history of the poses of the robot.

On account of the nature of this project, further information such as PTAM features positions, in particular to map the environment, were required from the AR.Drone system. The coordinates of every features have been sent through ROS messages from *tum_ardrone* package. Two possible message types can be applied at sending a set of points through ROS, which are: `sensor_msgs/PointCloud` and `sensor_msgs/PointCloud2`. `sensor_msgs/PointCloud` is the oldest and easiest way to send point cloud information, but it is not well suited for PCL, thus use `sensor_msgs/PointCloud2` is better choice since a wrapper to convert the ROS message to `PCL PointCloud` type by ROS API. PointCloud2 message is not a simple array of three-dimensional positions, it also allows to transmit other information in the message (e.g. rgba encoded color of a point). Nevertheless, all coordinates and attributes of the points have to be serialized in a data object before sending the message (each point must be serialized in a variable). PointCloud2 message is also very helpful since it allows interaction with PCL. This is possible on account of ROS API bridge which is able to convert ROS message datatype to PCL one.

ROS messages have also been used for making available to all running ROS nodes the occupancy map of the environment (creation of the map is described in 5.2.2). This has been accomplished using `nav_msgs/OccupancyGrid` message, which represents a $n \times m$ matrix (where n and m are height and width of the map). The value stored in each element is the probabilistic value of the occupancy in that particular cell (0-100); if there is no information concerning cell availability the value stored is -1.

Furthermore, `nav_msgs/GridCells` message has been used for debugging purposes such as to display cells already visited and evaluated, and

to visualize the current target, which can be described as the best position chosen among the evaluated ones.

Standardization of messages has allowed us to use existent plugins to develop a better graphic user interface. ROS visualization standalone tool *RViz*, as an example, allows to render several display message types such as those used in our project. Additionally, a QT based plugin for graphical user interfaces is available to embed a *RViz* visualization frame into a custom interface.

5.2.2 Occupancy Grid

As described in Chapter 3, to make the drone able to autonomously fly and explore in the real world, we have to obtain a reasonable amount of reliable information by the PTAM system in order to reconstruct a discretized (occupancy) map of the surrounding environment.

Therefore, the three-dimensional projection of each point catched by PTAM system has to be classified as reliable or unreliable. To classify each point of the Point Cloud coming from PTAM system, a filtering process has to been applied.

As mentioned in Chapter 3, we have considered two kinds of unreliable points: those with inaccurate position estimate and those which are outside the fly-zone of the drone. Hence, we filter the point cloud by firstly deleting inaccurate points and then removing irrelevant points.

In the first step, we have taken advantage of the sparse outlier removal filter implementation in PCL (Algorithm 1), that is based on the computation of the distribution of points to neighbour distances.

Initially, the average distance from each point to all its neighbours is computed. The resulting distribution of averaged points is, then, assumed as a Gaussian characterized by a standard deviation and a mean. Therefore, all the points whose mean distance is outside an interval defined by the distribution global distance standard deviation and mean can be considered as outliers and deleted from the point cloud.

Algorithm 1 is the pseudo code of the first filtering step where **P** is a point array that represents the input point cloud and **FP** is the corresponding filtered point cloud.

Moreover, the **getPointsInRadius** function implements **FLANN** algorithm, which is already known [63] for fast approximate the nearest neighbour searches. The result of this function is a set of all the points within a

Algorithm 1 Point Cloud Filtering algorithm

```
1: procedure RadiusOutlierRemoval(P, radius, min_neighbour)
2:     i ← 0
3:     j ← 0
4:     while i < P.size do
5:         B ← getPointsInRadius(P_i, radius)                ▷
6:         if B.size < min_neighbour + 1 then
7:             FP.append(P_i)
8:         end if
9:     end while
10:    return FP
11: end procedure
```

fixed radius from the point P_i. An example or result of this filtering process is shown in figure 5.3. The points removed with this filter are represented in red.

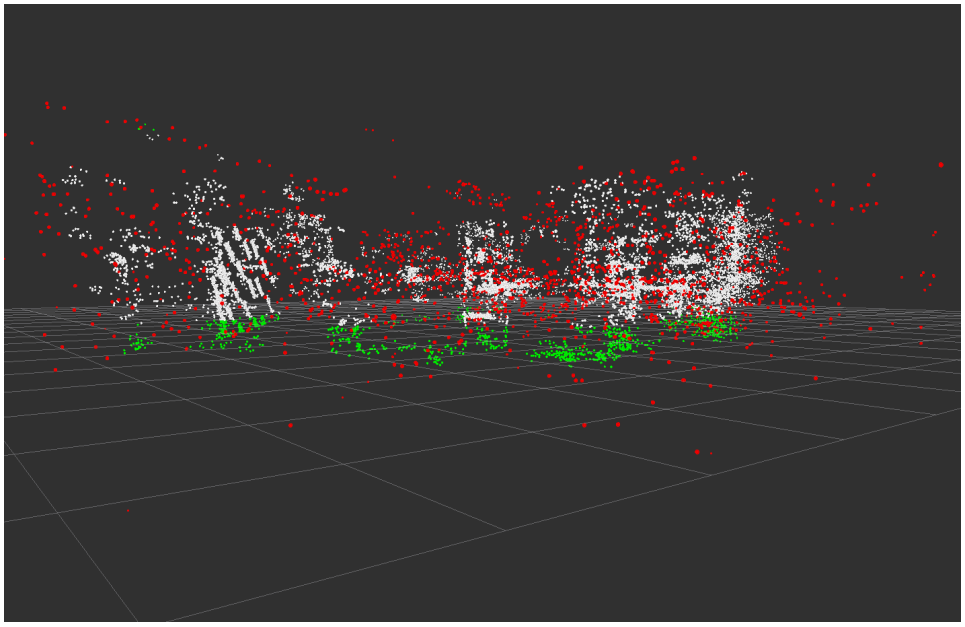
As already mentioned, a second filtering step is needed to prevent errors in estimation of visitable cells. Thus, to remove irrelevant points from point cloud a *PassThrough filter* is used. This filter deletes every point of the cloud with z-value under 0.2 meters and over 3 meters (green points in figure 5.3) making the system able to detect obstacles only in areas where the drone can fly.

Just after filtering the point cloud, we have introduced an actual step of *occupancy map* creation. At the beginning, the occupancy grid is totally initialized with -1 values (all cells occupancies are unknown) and, subsequently, it is populated by setting each cell with a representativeness heuristic (equation 5.3).

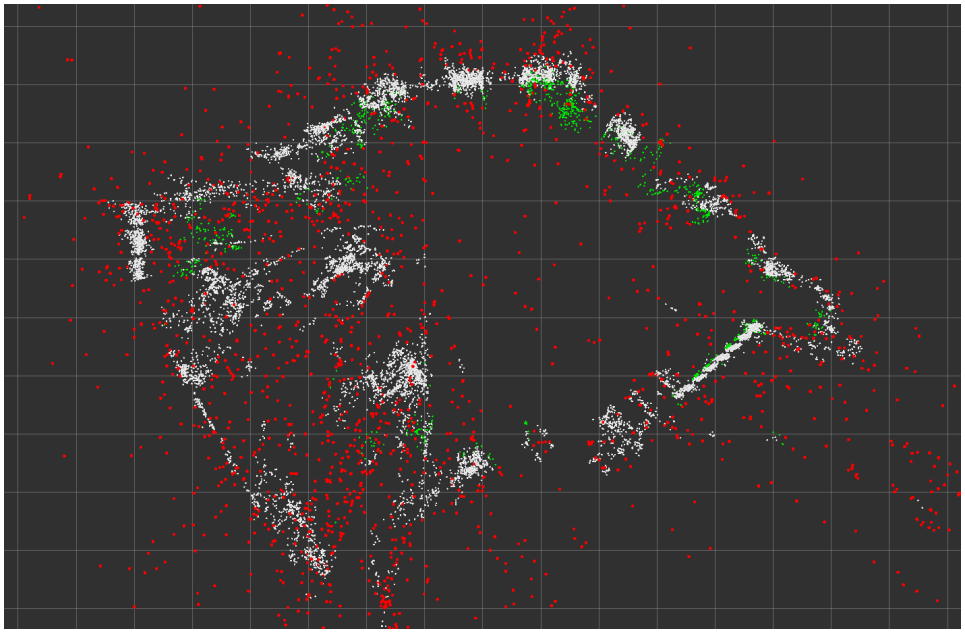
$$\rho(x,y) = \begin{cases} 1/10 & \text{if } |P|_{x,y} = 1 \\ 2/5 & \text{if } |P|_{x,y} = 2 \\ 7/10 & \text{if } |P|_{x,y} = 3 \\ 1 & \text{if } |P|_{x,y} > 3 \end{cases} \tag{5.3}$$

where $P_{x,y}$ is the set of points of the point cloud that lies in cell (x, y) and ρ is the occupancy probability given to cell (x, y).

To build an occupancy map, the drone position and orientation (provided by the `geometry_msgs/PoseStamped` message) are used to understand which cell represented in the occupancy grid is free and which is not. A process



(a)



(b)

Figure 5.3: Examples of a filtered point cloud. White points represent the final filtered point cloud.

of ray tracing is done from drone position on a field of view of 80° [2] (from 40° left to 40° right from the orientation of the robot) to inspect the map for possible obstacles.

Bresenham line algorithm (Algorithm 2) has been used to ray trace from the position of the drone on all its lines of sight due to its low computational impact. Bresenham lines are drawn from the drone position as steps of 1 degree each in its field of view [64].

Algorithm 2 Bresenham line algorithm

```
1: procedure LINE(x_0, x_1, y_0, y_1)
2:     dx ← x_1 − x_0
3:     dy ← y_1 − y_0
4:     D ← 2 ∗ dy − dx
5:     line ← {(x_0, y_0)}
6:     y ← y_0
7:     for x_0 + 1 < x_1 do
8:         if D > 0 then
9:             y ← y + 1
10:            line ← line ∪ {(x, y)}
11:            D ← D + (2 ∗ dy − 2 ∗ dx)
12:        else
13:            line ← line ∪ {(x, y)}
14:            D ← D + (2 ∗ dy)
15:        end if
16:    end for
17: end procedure
```

If an obstacle is found on the grid its cell is registered as occupied, while all the cells between the drone and that one are marked as free tiles (in the line of sight). We set, in our system, that an obstacle is detected if the probability for a cell to be occupied is over 1/3 (the probability of each cell is computed by equation 5.3). When no obstacles are found (tracing the ray) until the end of the grid, no information related to occupancy of the line of sight can be guessed since PTAM does not have continuous information about objects, but sparse ones. As shown in figure 5.4, a clean wall has no features on it, therefore it is not recognised as an obstacle by the system, but it is one, indeed.

By this approach we can populate the map adding new information regarding free cells to the previous map populated only with occupied ones.

[2] 80° is the horizontal field of view of AR.drone front camera

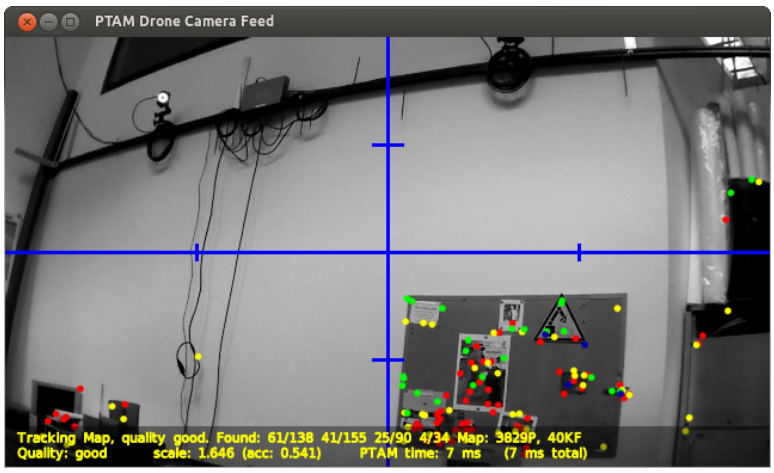


Figure 5.4: Lack of features on walls.

5.2.3 Visited Grid

In order to increase the efficiency of the exploration strategy, a map called *visited map* together with the occupancy map is used. The knowledge related to visited positions could improve exploration efficiency in terms of time spent by the drone to explore all the areas of an environment.

The first step of visited map creation is the initialization of all cells with -1 values.

As in the occupancy map construction, in order to build a visited map, the position of the drone (received by manager in the form of `geometry_msgs/PoseStamped` message) is used to detect which cells have been visited and which have been not. As shown in figure 5.5, every time that a pose message is received, a set of cells is marked as visited (green color in figure). This set is built considering a square of side equal to the drone size (70cm), which is centered in the current pose of the drone.

Once the set of visited cells is constructed, a `nav_msgs/GridCells` message (defined by the size of the visited map and the position coordinates of the visited cells in the map) is sent to the graphic interface in order to be represented on the exploration map.

Figure 5.5: Visited grid construction sample.

5.2.4 Scale adaptation

tum_ardrone continuously adjusts the scale factor of the map features in order to have consistent measurements between PTAM points and sensors data (by means of an ultrasound altimeter).

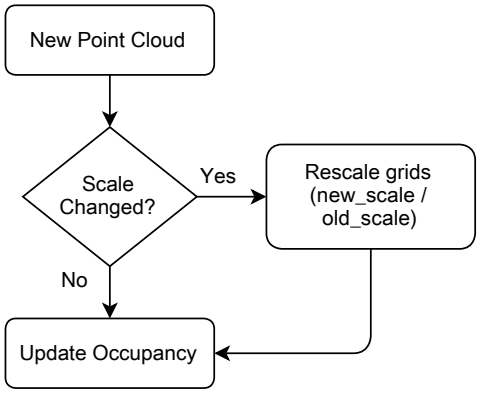


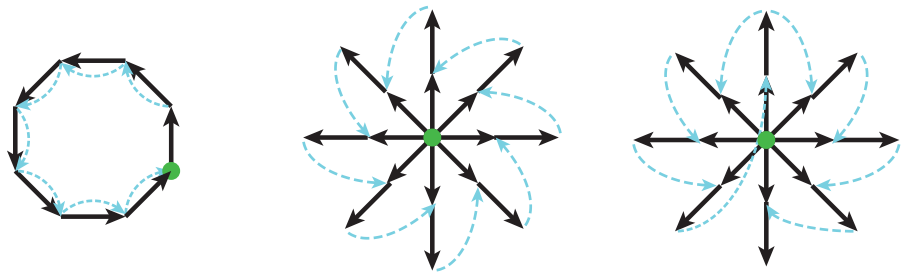
Figure 5.6: Diagram of Point Cloud scale procedure.

Since the map is built in a node different with respect to the one that publishes the point cloud and the scale factor, every time that the scale factor changes, the occupancy grid has to be rescaled according to both the previous and the current scale factor before any map update. This is accomplished by, firstly, locking any unwanted update by other functions and, then, releasing the lock only when the occupancy grid has been rescaled. This operation has been inserted to prevent from any accidental rewrite or update during the rescale mechanism.

5.3 Exploration manager

As mentioned in Chapter 3, two approaches towards the initialization of the exploration were considered: a static and a dynamic approach. We have evaluated three main static heuristics named circle, star and double half star path (figure 5.7).

The first static path evaluated is called circle path. As shown in figure 5.7 (a), this path consists in a sequence of tasks which drives the drone to move on a circular path.



(a) (b) (c)

Figure 5.7: (a) Circle path heuristic. (b) Star path heuristic (c) Double half star heuristic

Each step consists of an α rotation of 45° from the previous direction pose plus a translation of a given step, which is possible to configure before the application launch, in an α direction. The experimental result of this static path expresses a limit due to shape and size of the scene. The presence of obstacles (e.g. tables or pillars) can, indeed, prevent the quadrotor from completing a full circle on account of a loss of PTAM tracking during movements. Moreover, applying this path, every step the quadrotor leads to a movement to an unknown position, without any guarantee that the new space slot will really be available.

Star path (figure 5.7 (b)) has been implemented to decrease the working area. This path is composed of a couple of steps that draw a star-like trail. Each couple consists of a forward translation and rotation (-45°) step paired with a backward translation step. This allows the robot to return every two steps at the starting point, staring always at known features previously taken. The main problem is caused by an error in the point cloud estimation process, which increases the turning angle at each step thus ending to a complete rotation over 360° .

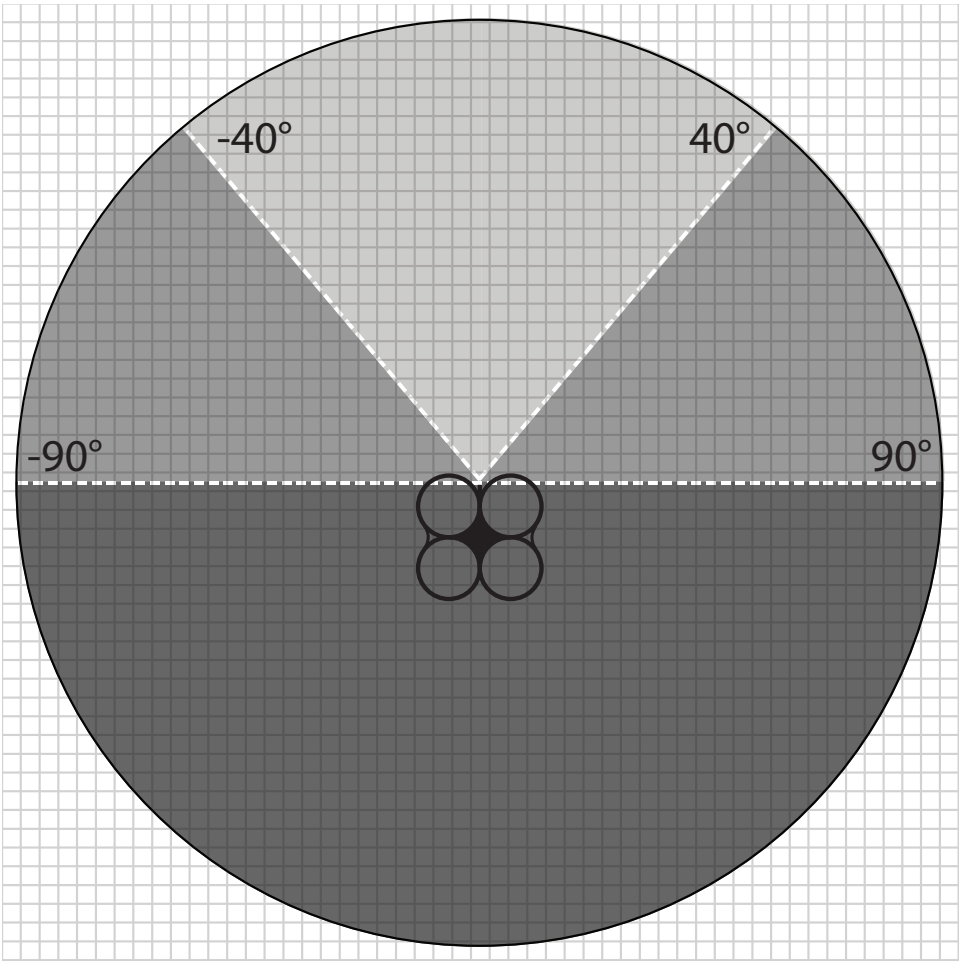


Figure 5.8: Order of the utility function computation.

In order to reduce the described errors, we tried to divide the previous sequence in two sections creating a double half star path static strategy that is shown in figure 5.7 (c). The first half of the sequence is, therefore, performed through a left rotation path and, after bringing the robot back to the starting position, the second part is performed through a right rotation path.

All of the static paths studied are not always feasible for different rooms. Changing the starting base of the drone from a room to another, a static path needs many modifications in order to fit well a new scene. This makes the previous strategies useless for previous environments.

Therefore, we needed to implement a strategy able to adapt itself to different scenarios without any human modifications or predetermined rules.

Taking into account that the main piece of information derived from the surrounding environment is acquired by the visual sensor, our goal was to develop a strategy fully based on utility functions computed following an order of importance (from light gray to dark gray in figure 5.8). In our system, indeed, higher importance is given to the current by observed partition of space, in order to make the AR.Drone able to perform movements the less suddenly as possible, and to avoid fast changes from the current orientation.

Towards developing a good strategy for the exploration, we had to implement some utility functions. These functions take as input the position and orientation of a possible next target of the exploration and return a score relative to the possible increase of environment knowledge with the given target.

Before any utility function evaluation, the difference between the altitude estimate by SLAM system and the altitude estimate of the ultrasound altimeter is considered. If the difference between the two is too big (we have inserted a maximum difference equal to 0.3 meters in order to keep the drone in flight safety), a routine will be called to recalibrate the estimated altitude.

As for as the dynamic exploration strategies are concerned, initially, we decided for an approach based on consecutive positions at a fixed distance.

In this strategy, to evaluate the next position in which the robot will be sent, the exploration manager initially builds a set of cells based on a perimeter of radius equal to a predefined distance centered on the current drone position. We set, by default, the radius at 50cm (the width of the AR.Drone). Afterwards, the set is filtered by removing all the cells that are too close to obstacles (we decided for a distance under 1.2 meters taking into account drifts and scale errors) and the utility of each cell is evaluated by computing the distance between the cell and the nearest neighbour obstacle. Finally, the exploration manager chooses, between all evaluated cells, the one which is the nearest to an object and it evaluates how the drone has to be oriented. The orientation is computed by equation 5.4 through the $atan2$ function.

$$orientation = \text{atan2}\frac{x_d - x_c}{y_d - y_c} \tag{5.4}$$

where x_d and y_d are the target coordinates and x_c and y_c the current position coordinates.

One limit of this strategy is that the drone has not to start orthogonal to a wall in order to avoid a forward-backward movement loop. In addition, this

approach results inappropriate for a rapid exploration since visited positions are not considered and the drone can go to these position more than once.

5.3.1 Chosen strategy

To make the exploration process faster, we have developed a strategy based on several utility functions, thus achieving a ready available room map and, moreover, saving as much battery as possible since a fully charged one allows the drone to fly continuously for a maximum of 10 minutes. Therefore, the time scale of the exploration process has to be minimised without losing efficiency.

The first utility function, in order of importance, is called `forward_UF`. This function is computed for a set of grid cells in the light gray map partition shown in figure 5.8. To decrease computational complexity the set is populated by a function (`FFC` [3]) based on Bresenham algorithm as described above. This function evaluates each cell in a line starting from the drone pose and it, then, returns the nearest cell that satisfies the following constraint: it is inside the grid, not already visited and at least 120 centimeters from an object. The last feature was introduced to let the drone fly safely, taking into account possible scale errors and drifts. The distance between a cell and the nearest neighbour object is computed through a function called `getDistanceFromObject`. We, at first, developed this function using a *midpoint circle algorithm* (Algorithm 3).

This algorithm consists of iteratively looking for obstacles on circumferences centered on the considered cell. It starts inspecting circumference of radius 1 and, in case it does not find any obstacles, it increments the radius. The algorithm stops when the current inspected circumference contains at least an occupied cell.

Subsequently, to increase the performance of `getDistanceFromObject` we have implemented it with a distance transform algorithm. To the extent of decreasing the time taken to evaluate cells distances we have used the *"dead reckoning" signed distance transform* (DRA) [55].

In contrast with *midpoint circle algorithm* based method, which evaluates distance value every time for each cell, DRA computes the distance map (that is the map of distances from each cell to the nearest neighbour occupied cell) only when the point cloud changes. Thus, DRA makes the

[3] First Fitting Cell

Algorithm 3 Midpoint circle algorithm

```
1: procedure CIRCLE(x_0, y_0, r)
2:     f ← 1 − r
3:     dx ← 1
4:     dy ← (−2) ∗ r
5:     x ← 0
6:     y ← r
7:     circle ← ∅
8:     circle ← circle ∪ {(x_0, y_0 + r), (x_0, y_0 − r)}
9:     circle ← circle ∪ {(x_0 + r, y_0), (x_0 − r, y_0)}
10:    while x < y do
11:        if f >= 0 then
12:            y ← y − 1
13:            dy ← dy + 2
14:            f ← f + dy
15:        end if
16:        x ← x + 1
17:        dx ← dx + 2
18:        f ← f + dx
19:        circle ← circle ∪ {(x_0 + x, y_0 + y), (x_0 − x, y_0 + y)}
20:        circle ← circle ∪ {(x_0 + x, y_0 − y), (x_0 − x, y_0 − y)}
21:        circle ← circle ∪ {(x_0 + y, y_0 + x), (x_0 − y, y_0 + x)}
22:        circle ← circle ∪ {(x_0 + y, y_0 − x), (x_0 − y, y_0 − x)}
23:    end while
24: end procedure
```

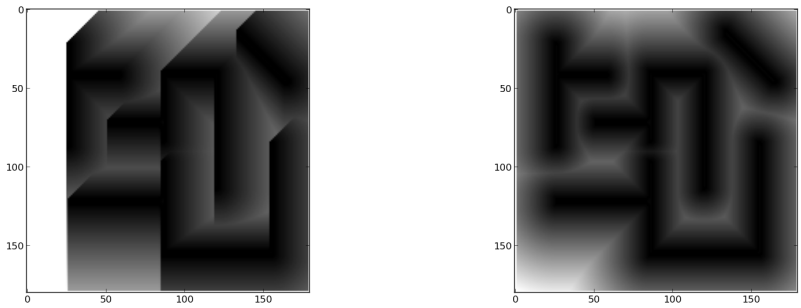


Figure 5.9: Results after first pass (a), second pass (b) in DRA.

distance value of all cells in the grid always updated and ready to use and it can be returned by `getDistanceFromObject` without other computations.

As mentioned in Chapter 2, DRA evaluates the distance transform (also known as distance map or distance field). In the pseudo code (Algorithm 4), the two passes used to evaluate the distance map are shown.

Considering I as a 2D map of size X by Y, d as the distance map of size X by Y and p as the corresponding border (occupied) cell for each cell in the grid, DRA evaluates the distance value of each cell from the distance value of the surrounding cells in a 3x3 window [4]. This is possible by scanning firstly the occupancy map I from left to right, top to bottom, and then, in the second pass, from bottom to top and right to left. In figure 5.9 the results of the two phases are shown. As it is possible to see in figure, the quality of the approximation of the distance value of every cell with only two passes is undeniable.

As mentioned, with DRA implementation we have obtained a significant increase of performance, due to the smaller complexity of the algorithm. In fact, the extensive computation of the distance map with the *midpoint circle algorithm* based implementation has a time complexity of $\mathcal{O}(n^4)$, whereas the time complexity of DRA, on the same task, is $\mathcal{O}(n^2)$, considering n as the number of cells from which the distance value has to be evaluated and a n x n occupancy map.

Next target position estimation

Once a set of possible cells is obtained, the utility of every cell in the set is computed by `forward_UF` function (5.5).

[4] The 3x3 square of the neighbourhood of the considered cell

Algorithm 4 Dead Reckoning Distance Transform Algorithm

```
1: procedure DRA(I)
2:     d1 = 1
3:     d2 ← sqrt(2)
4:     for y = 1 → Y do                                   ▷ d initialization
5:         for x = 1 → X do
6:             p(x, y) = −1
7:             d(x, y) = ∞
8:         end for
9:     end for
10:    for y = 1 → Y do                                   ▷ first pass
11:        for x = 1 → X do
12:            if d(x − 1, y − 1) + d2 < d(x, y) then
13:                p(x, y) = p(x − 1, y − 1)
14:                d(x, y) = sqrt((x − p(x, y).x)^2 + (y − p(x, y).y)^2)
15:            end if
16:            if d(x, y − 1) + d1 < d(x, y) then
17:                p(x, y) = p(x, y − 1)
18:                d(x, y) = sqrt((x − p(x, y).x)^2 + (y − p(x, y).y)^2)
19:            end if
20:            if d(x + 1, y − 1) + d2 < d(x, y) then
21:                p(x, y) = p(x + 1, y − 1)
22:                d(x, y) = sqrt((x − p(x, y).x)^2 + (y − p(x, y).y)^2)
23:            end if
24:            if d(x − 1, y) + d1 < d(x, y) then
25:                p(x, y) = p(x − 1, y)
26:                d(x, y) = sqrt((x − p(x, y).x)^2 + (y − p(x, y).y)^2)
27:            end if
28:        end for
29:    end for
30:    for y = Y → 1 do                                   ▷ second pass
31:        for x = X → 1 do
32:            if d(x + 1, y) + d1 < d(x, y) then
33:                p(x, y) = p(x + 1, y)
34:                d(x, y) = sqrt((x − p(x, y).x)^2 + (y − p(x, y).y)^2)
35:            end if
36:            if d(x − 1, y + 1) + d2 < d(x, y) then
37:                p(x, y) = p(x − 1, y + 1)
38:                d(x, y) = sqrt((x − p(x, y).x)^2 + (y − p(x, y).y)^2)
39:            end if
40:            if d(x, y + 1) + d1 < d(x, y) then
41:                p(x, y) = p(x, y + 1)
42:                d(x, y) = sqrt((x − p(x, y).x)^2 + (y − p(x, y).y)^2)
43:            end if
44:            if d(x + 1, y + 1) + d2 < d(x, y) then
45:                p(x, y) = p(x + 1, y + 1)
46:                d(x, y) = sqrt((x − p(x, y).x)^2 + (y − p(x, y).y)^2)
47:            end if
48:        end for
49:    end for
50:    for y = Y → 1 do
51:        for x = X → 1 do
52:            if I(x, y) == 0 then
53:                d(x, y) = 0
54:            end if
55:        end for
56:    end for
57: end procedure
```

`forward_UF` initially evaluates δ_{obj} distance from nearest neighbour object, δ_{unk} distance from nearest neighbour unknown cell through the aforementioned functions, and consequently, the difference between δ_x and θ_x.

θ_{obj} and θ_{unk} are preset parameters which represent respectively minimum distance from object and unknown space in order to prevent collisions due to drift and external agents.

$$\begin{aligned} \Delta_{obj}(i,j) &= |\delta_{obj}(i,j) - \theta_{obj}| \\ \Delta_{unk}(i,j) &= |\delta_{unk}(i,j) - \theta_{unk}| \end{aligned} \tag{5.5}$$

Then it computes the utility u of each cell as a function of $\Delta_{obj}(i,j)$ and $\Delta_{unk}(i,j)$ that are shown as Δ_{obj} and Δ_{unk} for simplicity:

$$u(i,j) = \begin{cases} \dfrac{\mu}{|\Delta_{obj}\Delta_{unk}^2|} & \text{if } \Delta_{obj},\ \Delta_{unk} \neq 0 \\ \mu & \text{if } \Delta_{obj},\ \Delta_{unk} = 0 \end{cases} \tag{5.6}$$

$$\forall\ \text{cell}(i,j) \notin\ \text{Grid} : \mu = -1 \tag{5.7}$$

$1/\Delta_{obj}(i,j)$ is used to try to increase number of found features since it increases the utility when getting close to an object (and consequently to possible features). In contrast $1/\Delta_{unk}(i,j)$ represents the frontier based strategy since it increases utility getting close to a frontier.

If `forward_UF` returns all negative values for grid cells previously considered, the utilities of cells in gray partition of map (figure 5.8) grid are evaluated with `forward_UF`. We have computed utilities of cells in front of the drone with the same function in two steps, in order to give more importance to cells in the field of view partition of the space, and to decrease the computational time complexity in case of valid target cell in that partition.

When all computed forward utilities are negatives, the utilities of a different set of cells are evaluated with `backward_UF` (5.8). This set consists of cells in dark gray partition of map grid (as shown in figure 5.8) through the same algorithm (`FFC`) used for the population of the forward set of cells.

$$u(i,j) = \begin{cases} \mu|\delta_{obj}| & \text{if } \delta_{obj} \neq 0 \\ \mu & \text{if } \delta_{obj} = 0 \end{cases} \tag{5.8}$$

Unlike `forward_UF`, `backward_UF` is in function of only one variable $\delta_{obj}(i,j)$ that, like in `forward_UF`, represents distance from nearest neighbour object. The utility of each cell is much greater when target position is far from objects.

Next target orientation estimation

Once a safe target position is evaluated, we have to decide where the drone will look at. Since an overlapping area is always needed due to the PTAM algorithm nature (new knowledge starts only from already known references), we have to choose the target angle of view in order to always have in one of the halves of the field of view enough features and in the other half few or none features.

The direction of the drone is computed with `getOrientation` function taking inspiration from view-improvement method. `getOrientation` initially inspects all directions from the target position to detect on which angles we have a reference (that means that there is an obstacle on that line of sight) and it labels them as valid.

Consequently, valid angles are filtered by `getGoodOrientations` in order to consider only angles with particular characteristics.

As shown in algorithm 5, `getGoodOrientations` checks for every angle if left (or right) hemisphere has a number of occlusions at least five times greater that the right (or left) hemisphere.

Finally, the `getOrientation` evaluates the utility u_α of each good angle with respect to current angle ψ. Angle orientation with greatest utility is chosen and a command with the target position and angle evaluated is sent to the drone.

$$u_\alpha = |\alpha - \psi| \tag{5.9}$$

Once target position and orientation are computed, exploration module has to choose which rotation, if clockwise or anticlockwise, the drone has to perform. When the drone has to move from an orientation to another, it establishes to turn on z axis in order to perform the shorter rotation. For example, if the drone is looking at 20° and it has to move looking at 50°, it will choose to perform a clockwise rotation. Problems arise when the drone can not track any feature between 20° and 50°.

In order to make the drone able to operate always in presence of tracked features, we have to detect a lack of features before the drone starts the rotation. To make this possible we have implemented the `getDirection` function (algorithm 6).

`getDirection`, starting from current position and orientation of the drone, checks the presence of trackable features, computing the number of directions for which some known features are available.

Initially, `getDirection` evaluates r and $foundFeatures_{right}$ increasing

Algorithm 5 Algorithm used to classify orientations

```
1: procedure getGoodOrientations(cell(i, j))
2:     α ← 0
3:     count_l ← 0
4:     count_r ← 0
5:     while α < 2π do
6:         γ = −π/6
7:         while γ <= π/6 do
8:             β ← α + γ
9:             if γ <= 0 and β is valid then
10:                count_l + +
11:            end if
12:            if γ >= 0 and β is valid then
13:                count_r + +
14:            end if
15:            if count_l/count_r > 5 or count_r/count_l > 5 then
16:                α is good angle for cell(i,j)
17:            end if
18:            γ ← γ + 1
19:        end while
20:        x ← x + 1
21:    end while
22: end procedure
```

Algorithm 6 Algorithm

```
1: procedure getDirection(cur_cell(i, j), dest_cell(i, j))
2:     l ← 0
3:     r ← 0
4:     foundFeatures_left ← 0
5:     foundFeatures_right ← 0
6:     tmp ← orientation_cur_cell
7:     while tmp ≠ orientation_dest_cell do
8:         if featuresOn(tmp) then
9:             foundFeatures_left ← foundFeatures_left + 1
10:        end if
11:        tmp ← tmp − 1
12:        l ← l + 1
13:    end while
14:    tmp ← orientation_cur_cell
15:    while tmp ≠ orientation_dest_cell do
16:        if featuresOn(tmp) then
17:            foundFeatures_right ← foundFeatures_right + 1
18:        end if
19:        tmp ← tmp + 1
20:        r ← r + 1
21:    end while
22:    if foundFeatures_left / l > foundFeatures_right / r then
23:        if orientation_dest_cell − orientation_cur_cell > 0 then
24:            return "ANTICLOCKWISE"
25:        end if
26:    else
27:        if orientation_dest_cell − orientation_cur_cell < 0 then
28:            return "CLOCKWISE"
29:        end if
30:    end if
31: end procedure
```

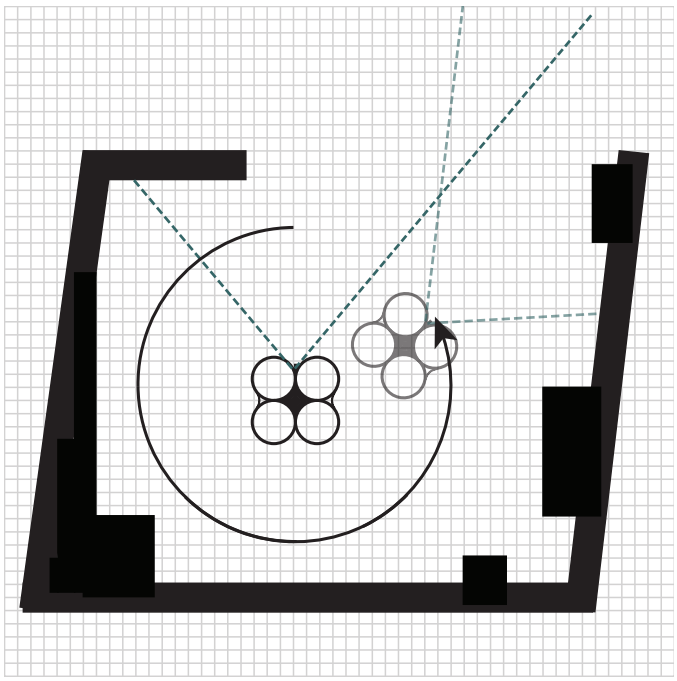


Figure 5.10: Rotation anticlockwise evaluated with getDirection.

the current orientation angle until it is equal to the target orientation angle. r represents the number of possible orientations between current and target angle rotating clockwise. $foundFeatures_{right}$ is incremented every time that known features are in the line of sight. The same process is done rotating anticlockwise computing l and $foundFeatures_{left}$.

Finally, the presence density of features is computed in order to evaluate which rotation, clockwise or anticlockwise, is better.

If the ratio between $foundFeatures_{right}$ and r is greater than the ratio between $foundFeatures_{left}$ and l the drone has to rotate clockwise. Otherwise, the drone has to rotate anticlockwise.

In case that all utilities computed by `forward_UF` and `backward_UF` are negative the exploration manager uses a fixed strategy that consists in evaluating only the utility of cells on a circle of fixed radius. The farthest cell from obstacles is chosen as cell with greater utility values. Associated to the position of this cell, this fixed strategy returns also an orientation equal to the sum of the current orientation and $\pi/4$ (or when this orientation frames any known features, the sum of the current orientation and $-\pi/4$).

This strategy is necessary in cases like the one in figure 5.11. In this kind of situations (such as dead-end) the drone explores the room or corridor

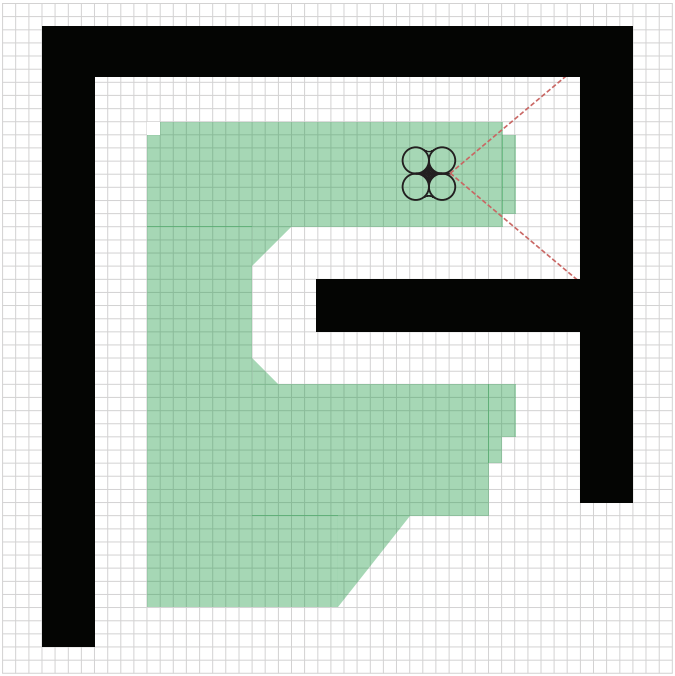


Figure 5.11: Example of scenario that needs different strategy.

evaluating utilities of cells which have not yet been visited. Once it reaches a limit position (such as position at 1.2 meters from an obstacle), there is not any eligible target because all cells in front of the drone are too close to obstacles and all cells behind the drone are already visited. Therefore, the drone has to move on an already visited cell rotating itself in order to return to unvisited areas.

Emergency thread

As mentioned, we have presented our exploration strategy taking into account the possible failure of feature tracking, which derives from limitation of the PTAM system. To prevent any collision, we have implemented a thread, which works in parallel with the exploration and controller threads, acting in the case of blind navigation due to PTAM tracking loss. In fact, this thread starts when the drone is in a PTAM_TRACKING_LOST state (figure 4.7 Chapter 4) and it blocks the execution of any other command regarding the exploration (e.g., the achievement of a target position) sending to the drone a sudden hover command. After that, it drives the drone to the nearest visited position where some found features are certainly present considering

the pose computed only with the help of EKF. Once this pose is reached the thread ends and the exploration can continue.

5.3.2 Custom message

In order to simplify communication between the Exploration manager nodes (*communication_node* and *exploration_node* in figure 5.1) communications we have to create a new custom ROS message called `Strategy`.

The most important variable in message structure is *droneState*. By changing *droneState*, Exploration manager can perform different tasks with respect to the current state of the drone. When the drone reaches a preset target, it publishes, on the `arsec/strategy` topic, a message containing, as *droneState*, DRONE_HOVERING, x, y, z, *yaw*, *altitude* equal to evaluated pose values and real altimeter readings; after this, it starts listening for a message on the same topic.

Like in case of *droneState* equal to DRONE_HOVERING, *communication_node* publishes on `arsec/strategy` a message with *droneState* equal to DRONE_MOVING or DRONE_AVOIDING respectively when the drone is moving to a target or if it loses PTAM tracking. In addition the message is completed with pose variables and in case of PTAM tracking loss, exploration manager saves that pose as no-explorable position in order to prevent to return to this position during following tasks.

Once a message of drone hovering is published on `arsec/strategy`, the exploration manager starts to compute the best next position to reach in order to increase its knowledge. After that, manager publishes a message which contains *droneState* equal to SENDING_ACTION associated to x, y, z of the target, a *direction* such as FORWARD, BACKWARD or SHIFT (depending on type of translation to do), a *degree* which represents orientation of the drone and a *rot* which represents in which direction the drone has to rotate (either clockwise or anticlockwise).

A schematic representation of `Strategy` is shown below.

ROS STRATEGY MESSAGE TYPE

— constants —

uint32	DRONE_HOVERING	= 0
uint32	DRONE_MOVING	= 1
uint32	DRONE_AVOIDING	= 2
uint32	SENDING_ACTION	= 3

— header —

Header	header
— pose variables —	
float32	x
float32	y
float32	z
float32	yaw
float32	altitude
— best action strategy —	
string	direction
int32	degree
string	rot
uint32	droneState

5.4 Graphical User Interface

Graphical user interface is a tool used for debugging and controlling the application. A good developed GUI makes faster the understanding of what the application is doing. In our project, the graphical user interface (figure 5.12) allowed us to debug during the implementation of the strategies in order to simplify and make it easier the identification of target that the drone has to reach. The target identification is not the only reason for which we have developed a GUI.

Moreover, we have studied and implemented the structure of the graphical user interface in order to have always under control some significant data about the drone such as IMU data, altimeter readings, state of the drone (taking off, landing, flying, etc.) and battery level. In addition to this data, some significant information about PTAM and Exploration Manager such as the current position and orientation of the drone, the target pose, the state of the PTAM system (in terms of number of founded features) and a map representation of the knowledge of the environment are displayed.

The GUI implementation has been possible thanks to the attention paid on the standardization of ROS messages and on the resources made available by the ROS framework.

In particular, we have used QT4 libraries to represent all information. QT is a cross-platform application framework used to develop application software with a graphical user interface and it is classified as a widget toolkit.

As shown in figure 5.13, the structure of GUI is divided in three blocks. In the red block, the drone and SLAM manager information are displayed on

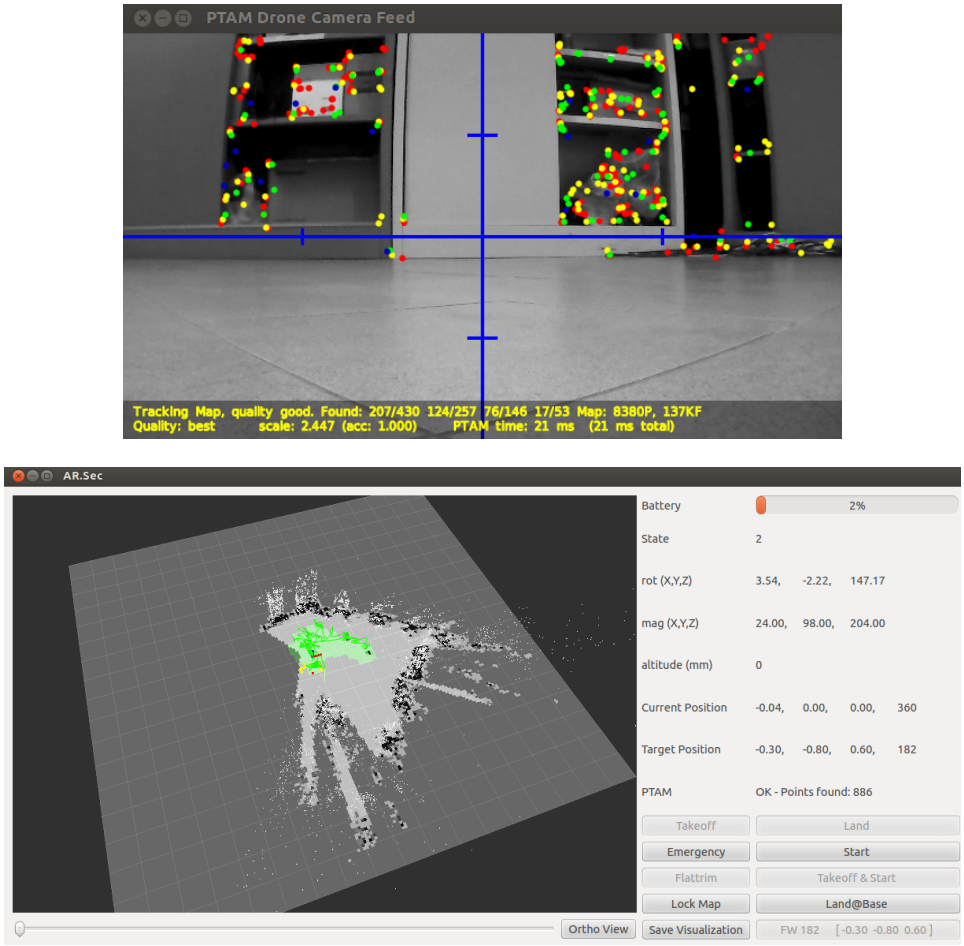


Figure 5.12: Graphical User Interface.

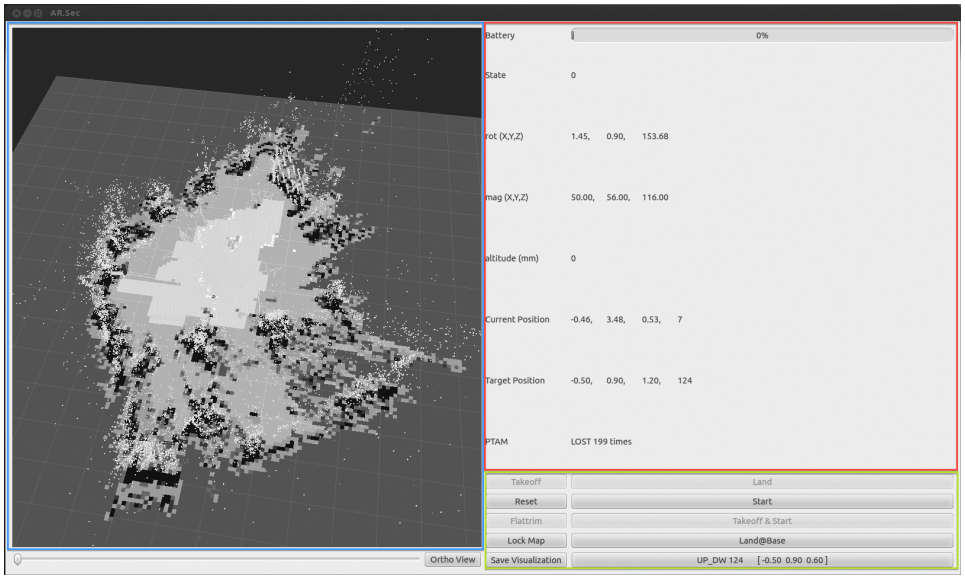


Figure 5.13: Information representation structure.

a grid. The user can interact with the drone by sending initial commands ("TakeOff" and "Start") through dedicated buttons in order to start the exploration process. Users can also act in case of danger by pressing the "Land" button (in order to make the drone to land immediately) or by pressing the "Emergency/Reset" button which shuts down all motors of the drone. All these buttons are in the green block. The last block, the blue one, is used to display a map of known objects, the visited path, and, in general, all information about the environment.

To display this information we have used RViz library. So we have embedded a visualizer (rviz::VisualizationFrame) in our GUI as a Qt widget. Thanks to this library we did not have to manage every single piece of information, but through the creation of a custom configuration file the widget programmatically loads data published on ROS topic directly on the GUI.

Moreover, the use of a ROS visualization frame integrated in our interface has enabled the removal of the original PTAM GL visualizer resulting in unexpected performance improvements (original PTAM implementation of the point cloud renderer has a busy wait in the render process that pushes the CPU at almost 100% usage, while we avoided this step, by decreasing the memory use to render the cloud).

5.5 Evaluation

To evaluate performance and usability of the system we have tested our package in some different scenarios.

The first evaluated scenario is an indoor room at the Artificial Intelligence and Robotics Laboratory at the Department of Electronics, Information and Bioengineering of Politecnico di Milano (AIRlab). As shown in figure 5.14, the shape of the room is irregular and it has some peculiarities that influenced the exploration operation.

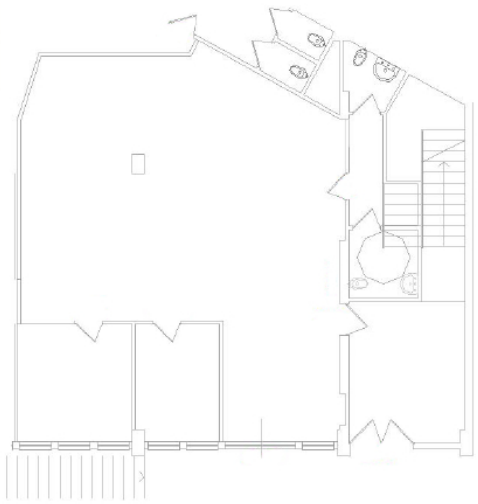
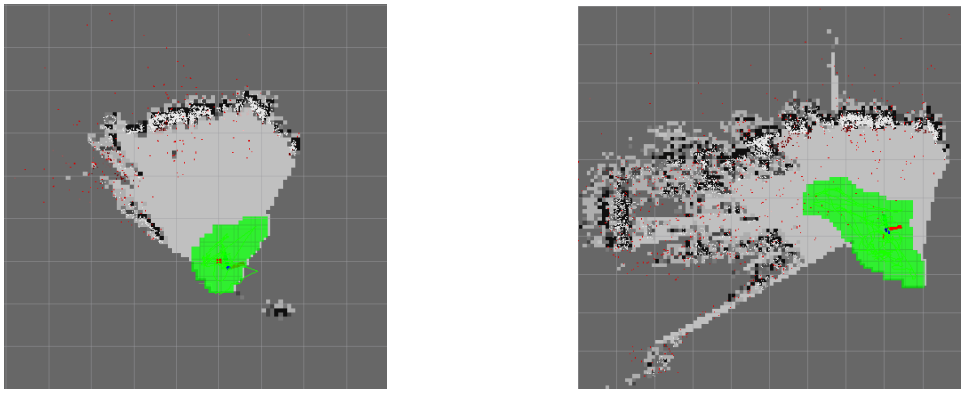


Figure 5.14: AIRlab room real map.

To better understand how these characteristics can influence the exploration, firstly we have tested our system using the static exploration paths.

As earlier discussed, these exploration strategies are too limited and they are influenced by the shape of the room. In this specific case, the room is not rectangular and, in addition, it shows a convex part. These peculiarities, in conjunction with the presence of a pillar (shown as black rectangle in the middle of the room map in figure 5.14), make it impossible for the drone to complete the exploration. In particular, applying the circle strategy, the drone performed a route that was too close to an obstacle (represented by a desk) and the feature tracking system performed by PTAM failed, because the field of view of the drone camera was covered by the mentioned object (figure 5.15 (a)).

We, thus, tried to adopt the star strategy, but it failed as well. In this case the failure of the exploration strategy is due to a column present in the



(a) (b)

Figure 5.15: Static exploration results. (a) AIRlab room reconstruction with circle static strategy. (b) AIRlab room reconstruction with start static strategy.

room. Indeed, PTAM can not detect the features of the pillar, depending on the position of the drone camera, thus obtaining a frame where only the right half is populated by PTAM keypoints. Later, during the next movement (which is composed of an orientation rotation of $-\pi/4$ and a translation as previously described), PTAM system lost tracked features, since the only features visible in the previous frame are now out of the field of view. This PTAM tracking failure and the nature of the static strategy made the drone unable to complete the static path, performing, every time, the movement that induces the PTAM system to lose the tracked features.

Considering the result of star strategy and the nature of the half star strategy (that consists in the same movements in different order), we have demonstrated that also the half star strategy can not be used to explore this environment since it gives the same results.

Subsequently, we tested our dynamic exploration strategy in the AIRlab room. Figure 5.16 shows the overlapping of the planimetry of the room on the map constructed by our system. In this figure, we have represented walls and door with orange color and furniture objects, like desks and cabinets, with yellow rectangles. As shown, the results of this exploration strategy are impressive in comparison with the static strategies and also from a general point of view, considering the absence, in our system, of any kind of laser scanner. The map, reconstructed in real time during the autonomous exploration, fits almost perfectly on the planimetry of the AIRlab.

To better understand, we have to underline the final result related to the concept of what represents a group of features. In fact, as aforementioned,

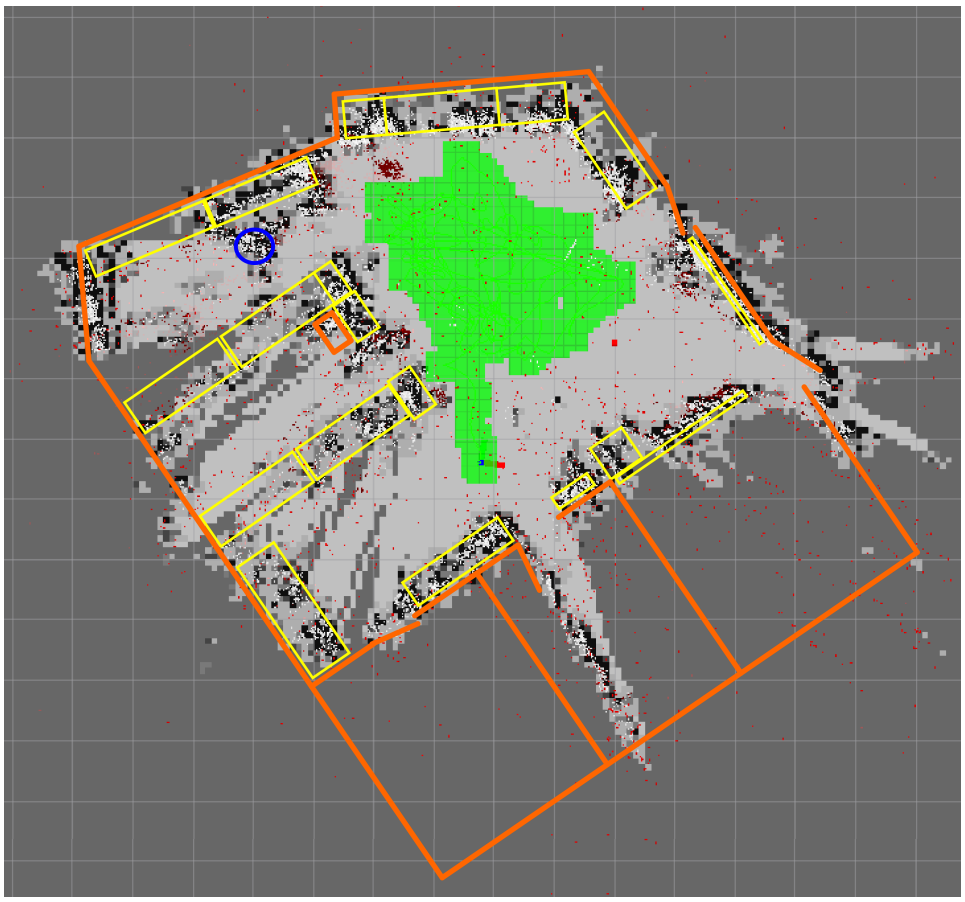


Figure 5.16: Result of dynamic exploration strategy over AIRlab room.

the correlation between an object and its features is very strong and it is also clear in the built map. The majority of the boundaries, indeed, are not a representation of walls (excluding the cases where something, such as a poster, is present) but they represent obstructions, drawn with yellow rectangles, such as cabinets, desks or even people (that are symbolized by a blue circle in this case).

The second evaluated scenario is an outdoor limited environment. This environment is a courtyard shaped like a rectangle. Despite the simple concave shape, the dynamic exploration of this space has not been successful. This failure can not be addressed to obstacles presence or a difficult environment shape, but it is likely related to a variable light intensity during the test that heavily conditioned the camera exposure. We noticed indeed that, in case of transition between a frame of a low-intensity lighted scene

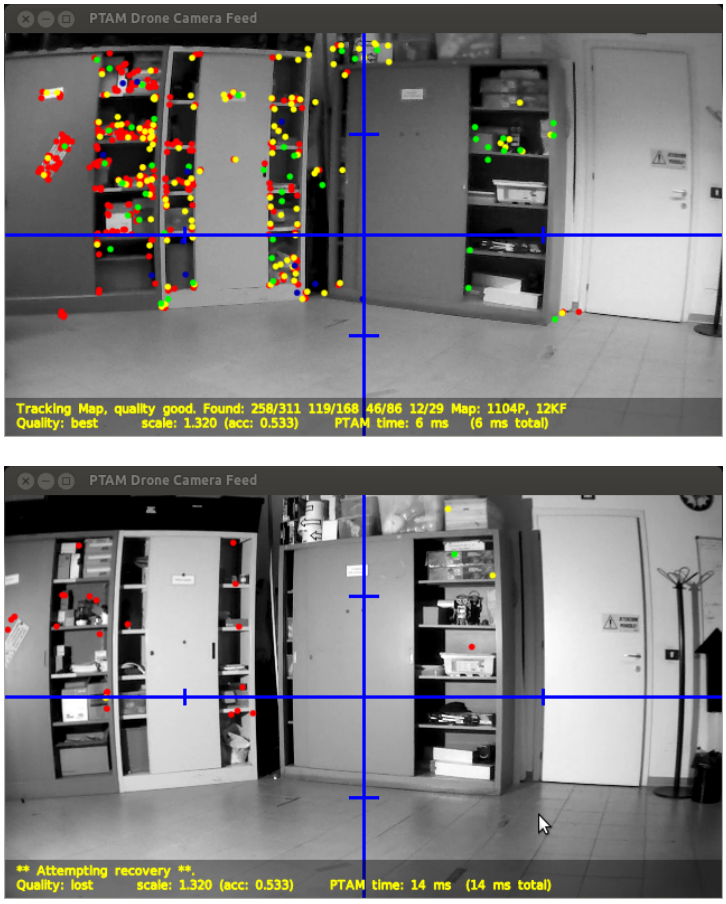


Figure 5.17: Effect of a sudden variation of light intensity in PTAM system.

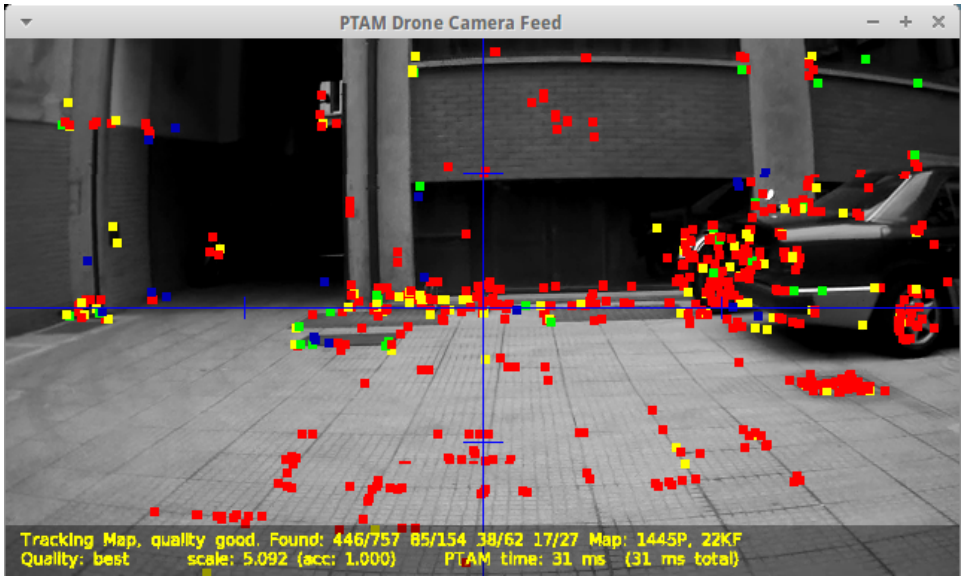
and a different one with a high-intensity light, the scene framed resulted very different with respect to the recorded features. Hence, a feature found in the first frame can not be detected by the system in the second one, since its characteristics in terms of intensity of light are different.

Moreover, the camera of the AR.Drone 2.0 has an automatic exposure (AE) mode that calculates and adjusts exposure settings autonomously to match (as closely as possible) the mid-tone of the considered subject with the mid-tone of the scene. This AE mode prevents us to avoid problems derived from variations of light intensity, on account of the impossibility to distinguish between a closer object (that obstructs the camera) and a sudden change of exposure.

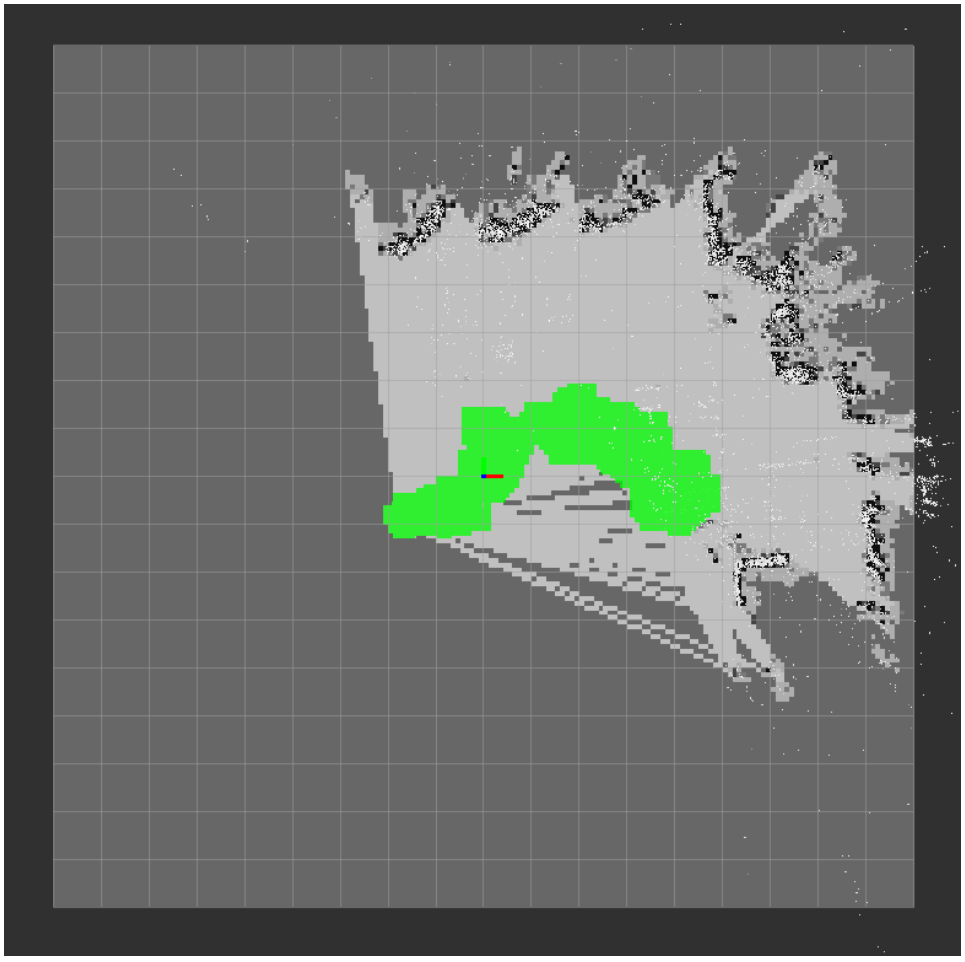
In contrast, this test allowed us to underline the importance of the point cloud filtering. As shown in figure 5.18, during this test PTAM system found many features located on the floor. Considering our concept of relation

between features and obstacles, described in previous chapters, without a filter on the position of the features the drone would not able to fly over them. In fact, as mentioned, the points used to map the environment have not a correspondence to specific objects, but they represent only image features of the scene framed. Therefore, since the drone flies between 20 cm (ground altitude) and 210 cm (doors height), these feature will not disturb the flight of the drone and, for this reason, they were correctly deleted by filtering processes.

In conclusion, we have established that our system works well in indoor environments where a lot of features are present because without them and without the use of any proximity sensor the system can not detect obstacles. Moreover, outdoor scenarios are advised against because some parameter such as the presence of wind, are not be considered in EKF system (as shown in Chapter 3.2) and they make the pose estimation unreliable.



(a)



(b)

Figure 5.18: Outdoor vision. (a) Example of features found on the ground. (b) Outdoor recreated map.

Chapter 6

Conclusions and future research

"Aimless extension of knowledge, however, which is what I think you really mean by the term curiosity, is merely inefficiency. I am designed to avoid inefficiency."

R. Daneel Olivaw - The Caves of Steel, Isaac Asimov

6.1 Conclusions

In this thesis we presented a novel system that enables a low-cost quadrotor to localize itself, navigate and explore autonomously an unknown environment without any positioning sensor such as GPS-sensor. To hold down the cost of the hardware, our approach uses only sensors available on the Parrot AR.Drone 2.0 and, particularly it uses the monocular camera without requiring any artificial marker.

In order to achieve our aim, we have used PTAM, a real-time, monocular localization and mapping algorithm based on keyframes, to reconstruct a discretized map of the surroundings of the quadrotor and to evaluate the position and orientation within this map. We have used a novel method from TUM Vision group to estimate the scale of the map from inertial and altitude measurements by formulating the problem statistically, and by deriving a closed-form solution for the maximum likelihood estimator of the unknown scaling factor. Furthermore, the TUM method uses PTAM algorithm together with an Extended Kalman Filter to join visual extracted

information and data received from other sensors in order to estimate the pose of the drone and a three dimensional description of the surrounding world.

We have used information from PTAM and the Extended Kalman Filter as the base of the exploration. To make this possible we have developed the system that extracts knowledge from environment description, obtained by PTAM, by filtering it opportunely and creating an occupancy grid which describes the location of obstacles and free space in a two-dimensional map.

Autonomous exploration is performed with a novel strategy influenced by Frontier-based and View-improvement methods. This strategy has been developed taking into consideration the nature of the localization and mapping system chosen and its constraints. Given the limited field of view of the sensor used to localize the quadrotor, our strategy favors movements that consist in small variations of the orientation angle.

The developed autonomous exploration system provides safe navigation with obstacle avoidance without the need of specific sensors such as proximity ones.

In summary, we showed in our experiments that a low cost quadrotor, based only on a monocular view, can navigate and explore an unknown environment building a three-dimensional representation of the environment and a two-dimensional map of the obstacles without the use of an expensive laser rangefinder. On one hand, the system resulted very reliable and accurate in typical indoor environment, on the other hand we showed that particular scenarios, such as unfurnished room or outdoor environments with sudden light intensity changes, do not fit well with the PTAM algorithm visual nature.

6.2 Future works

Considering the work of this thesis there are several aspects that can be analysed and improved.

First of all, the Extended Kalman Filter can be improved by adding new sensor observations used to estimate the pose of the quadrotor. The Parrot AR.Drone is equipped with a magnetometer, a barometer and a wind speed and angle direction sensor. Taking into account these sensors it might be possible to build a better EKF model in order to stabilize the navigation control and make exploration possible where it is now difficult (e.g., in windy outdoors environments or surfaces where ultrasounds altimeter give incorrect

estimate).

Another possible field of improvement could be the visual SLAM system. To increase the tracking performance during sudden movements, it is possible to use the edgelet-enhanced version of PTAM [65] that tracks also edge features that are more resilient to motion blur.

To increase the definition of the reconstructed three-dimensional map a possible approach is Dense Tracking and Mapping (DTAM) [66]. DTAM is a system for real-time camera tracking and reconstruction which does not rely on feature extraction like PTAM, but on dense methods, relying on every pixel of the image of the scene framed. This approach allows to build very precise three-dimensional meshes of the environment extracting information also about featureless objects. Moreover, DTAM gives better performance on camera position tracking.

Considering the exploration area a possible improvement is represented by multi-room exploration. To achieve this aim it is not possible to use PTAM algorithm, in fact, keyframes taken in different rooms, framing two sides of the same wall, could lead to conflict between features. As mentioned in Chapter 3, the tracking process of PTAM reprojects the features that lie in the current field of view on the image plane. If these features have been originally taken in another room, there is no possibility to found them in the current room resulting in a failure of the tracking that blocks exploration. A possible solution to multi-room problem could be the adoption of Parallel Tracking and Multiple Mapping system (PTAMM) [67] which is based on PTAM. PTAMM adds multi-map feature on PTAM system. This allows to bypass the problem described above because it treats features as belonging only to their map. However, PTAMM needs user interaction because it does not have the capability of understanding when a new map is needed (e.g. when entering a new room). To automate this process a segmentation of the environment could be performed to allow the system to understand what keyframes belongs to a room or another one.

Bibliography

[1] R. R. Murphy, *Introduction to AI Robotics*. Cambridge, MA, USA: MIT Press, 1st ed., 2000.

[2] R. E. Kalman, "A new approach to linear filtering and prediction problems," *Transactions of the ASME–Journal of Basic Engineering*, vol. 82, no. Series D, pp. 35–45, 1960.

[3] OptiTrack, "http://www.naturalpoint.com/optitrack/."

[4] A. J. Davison, W. W. Mayol, and D. W. Murray, "Real-time localisation and mapping with wearable active vision," in *Proceedings of the 2nd IEEE/ACM International Symposium on Mixed and Augmented Reality*, ISMAR '03, (Washington, DC, USA), pp. 18–, IEEE Computer Society, 2003.

[5] TUM, "https://github.com/tum-vision/tum_ardrone."

[6] G. Klein and D. Murray, "Parallel tracking and mapping for small ar workspaces," in *Proceedings of the 2007 6th IEEE and ACM International Symposium on Mixed and Augmented Reality*, ISMAR '07, (Washington, DC, USA), pp. 1–10, IEEE Computer Society, 2007.

[7] B. Yamauchi, "A frontier-based approach for autonomous exploration," in *Proceedings of the 1997 IEEE International Symposium on Computational Intelligence in Robotics and Automation*, CIRA '97, (Washington, DC, USA), pp. 146–, IEEE Computer Society, 1997.

[8] R. Gartshore, P. Palmer, and J. Illingworth, "A novel exploration algorithm based on a view-improvement strategy," 2005.

[9] Robocup, "http://www.robocuprescue.org/."

[10] Robocup, "http://wiki.robocup.org/wiki/robot_league."

[11] DARPA, "http://archive.darpa.mil/grandchallenge/."

[12] M. Achtelik, A. Bachrach, R. He, S. Prentice, and N. Roy, "Autonomous Navigation and Exploration of a Quadrotor Helicopter in GPS-denied Indoor Environments," in *Robotics: Science and Systems*.

[13] A. Technologies, "http://www.asctec.de/uav-applications/."

[14] I. I. D. S. GmbH, "http://en.ids-imaging.com/."

[15] Hokuyo, "http://www.hokuyo-aut.jp/02sensor/07scanner/utm_-30lx.html."

[16] J. Engel, J. Sturm, and D. Cremers, "Camera-based navigation of a low-cost quadrocopter," in *Proc. of the International Conference on Intelligent Robot Systems (IROS)*, Oct. 2012.

[17] B. Schwarz, "Lidar: Mapping the world in 3D," *Nature Photonics*, vol. 4, pp. 429–430, 2010.

[18] G. Stockman and L. G. Shapiro, *Computer Vision*. Upper Saddle River, NJ, USA: Prentice Hall PTR, 1st ed., 2001.

[19] H. P. Moravec, *Obstacle avoidance and navigation in the real world by a seeing robot rover*. PhD thesis, Stanford University, Stanford, CA, USA, 1980. AAI8024717.

[20] C. Harris and M. Stephens, "A combined corner and edge detector," in *Proceedings of the 4th Alvey Vision Conference*, pp. 147–151, 1988.

[21] J. Shi and C. Tomasi, "Good Features to Track," *Proceedings of the Conference on Computer Vision and Pattern Recognition*, pp. 593–600, June 1994.

[22] E. Rosten and T. Drummond, "Machine learning for high-speed corner detection," in *Proceedings of the 9th European conference on Computer Vision - Volume Part I*, ECCV'06, (Berlin, Heidelberg), pp. 430–443, Springer-Verlag, 2006.

[23] M. L. V. Pitteway, "Algorithm for drawing ellipses or hyperbolae with a digital plotter," *The Computer Journal*, vol. 10, pp. 282–289, 1967.

[24] J. Gibson, *The perception of the visual world*. Houghton Mifflin, 1950.

[25] D. Warren and E. Strelow, *Electronic Spatial Sensing for the Blind: Contributions from Perception, Rehabilitation, and Computer Vision*. NATO ASI Series. Series E: Applied Sciences, Springer, 1985.

[26] B. D. Lucas and T. Kanade, "An iterative image registration technique with an application to stereo vision.," in *IJCAI* (P. J. Hayes, ed.), pp. 674–679, William Kaufmann, 1981.

[27] B. Horn and B. Schunck, "Determining optical flow," *Artificial Intelligence*, vol. 17, no. 1-3, pp. 185–203, 1981.

[28] D. G. Lowe, "Object recognition from local scale-invariant features," in *Proceedings of the International Conference on Computer Vision-Volume 2 - Volume 2*, ICCV '99, (Washington, DC, USA), pp. 1150–, IEEE Computer Society, 1999.

[29] H. Bay, A. Ess, T. Tuytelaars, and L. Van Gool, "Speeded-up robust features (surf)," *Comput. Vis. Image Underst.*, vol. 110, pp. 346–359, June 2008.

[30] M. Calonder, V. Lepetit, C. Strecha, and P. Fua, "Brief: binary robust independent elementary features," in *Proceedings of the 11th European conference on Computer vision: Part IV*, ECCV'10, (Berlin, Heidelberg), pp. 778–792, Springer-Verlag, 2010.

[31] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski, "Orb: An efficient alternative to sift or surf," in *Proceedings of the 2011 International Conference on Computer Vision*, ICCV '11, (Washington, DC, USA), pp. 2564–2571, IEEE Computer Society, 2011.

[32] O. Miksik and K. Mikolajczyk, "Evaluation of local detectors and descriptors for fast feature matching," 2012.

[33] H. C. Longuet-Higgins, "Readings in computer vision: issues, problems, principles, and paradigms," ch. A computer algorithm for reconstructing a scene from two projections, pp. 61–62, San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1987.

[34] R. I. Hartley, "In defense of the eight-point algorithm," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 19, pp. 580–593, June 1997.

[35] R. I. Hartley and A. Zisserman, *Multiple View Geometry in Computer Vision*. Cambridge University Press, ISBN: 0521540518, second ed., 2004.

[36] A. J. Davison, I. D. Reid, N. D. Molton, and O. Stasse, "Monoslam: Real-time single camera slam," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 29, pp. 1052–1067, June 2007.

[37] R. A. Newcombe, S. J. Lovegrove, and A. J. Davison, "DTAM: Dense tracking and mapping in real-time," in *Computer Vision (ICCV), 2011 IEEE International Conference on*, pp. 2320–2327, IEEE, Nov. 2011.

[38] M. Grewal and A. Andrews, "Applications of kalman filtering in aerospace 1960 to the present [historical perspectives]," *IEEE Control Systems Magazine*, vol. 30, pp. 69–78, June 2010.

[39] N. Wiener, *Extrapolation, Interpolation, and Smoothing of Stationary Time Series*. The MIT Press, 1964.

[40] A. Kolmogorov, "Stationary sequences in Hilbert's space.," *Bull. Mosk. Gos. Univ. Mat.*, vol. 2, no. 6, pp. 1–40, 1941.

[41] D. S. Lemons and A. Gythiel, "Paul Langevin's 1908 paper On the theory of Brownian motion [Sur la théorie du mouvement brownien, Comptes-rendus de l'Académie des Sciences (Paris) 146, 530-533 (1908)]," *American Journal of Physics*, vol. 65, pp. 1079–1081, Nov. 1997.

[42] J. Riccati, "Animadversiones in aequationes differentiales secundi gradus," *Actorum Eruditorum, quae Lipsiae publicantur, Supplementa*, vol. 8, pp. 66–73, 1724.

[43] G. Smith, S. Schmidt, L. McGee, U. S. N. Aeronautics, and S. Administration, *Application of statistical filter theory to the optimal estimation of position and velocity on board a circumlunar vehicle*. NASA technical report, National Aeronautics and Space Administration, 1962.

[44] A. Vaughan, M. I. of Technology. Dept. of Aeronautics, and Astronautics, *A Monte-Carlo Performance Analysis of Kalman Filter and Targeting Algorithms for Autonomous Orbital Rendezvous*. Massachusetts Institute of Technology, Department of Aeronautics and Astronautics, 2004.

[45] M. J. Mataric, "Integration of representation into goal-driven behavior-based robots," *IEEE Transactions on Robotics and Automation*, vol. 8, pp. 304–312, June 1992.

[46] D. Holz, N. Basilico, F. Amigoni, and S. Behnke, "Evaluating the efficiency of frontier-based exploration strategies.," in *ISR/ROBOTIK*, pp. 1–8, VDE Verlag, 2010.

[47] M. L. Fredman and R. Tarjan, "Fibonacci heaps and their uses in improved network optimization algorithms," in *Foundations of Computer Science, 1984. 25th Annual Symposium on*, pp. 338–346, 1984.

[48] P. Hart, N. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *Systems Science and Cybernetics, IEEE Transactions on*, vol. 4, no. 2, pp. 100–107, 1968.

[49] J. Tsitsiklis, "Efficient algorithms for globally optimal trajectories," *Automatic Control, IEEE Transactions on*, vol. 40, no. 9, pp. 1528–1538, 1995.

[50] S. Garrido, L. Moreno, D. Blanco, and F. Martin, "Exploratory navigation based on voronoi transform and fast marching," in *Intelligent Signal Processing, 2007. WISP 2007. IEEE International Symposium on*, pp. 1–6, 2007.

[51] Z. Zhang, "A flexible new technique for camera calibration," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 22, no. 11, pp. 1330–1334, 2000.

[52] H. Stewénius, C. Engels, and D. Nistér, "Recent developments on direct relative orientation," *ISPRS Journal of Photogrammetry and Remote Sensing*, vol. 60, pp. 284–294, June 2006.

[53] A. E. Beaton and J. W. Tukey, "The fitting of power series, meaning polynomials, illustrated on band-spectroscopic data," *Technometrics*, vol. 16, no. 2, pp. 147–185, 1974.

[54] S. Fortune, "A sweepline algorithm for voronoi diagrams," in *Proceedings of the second annual symposium on Computational geometry*, SCG '86, (New York, NY, USA), pp. 313–322, ACM, 1986.

[55] G. J. Grevera, "The "dead reckoning" signed distance transform," *Comput. Vis. Image Underst.*, vol. 95, pp. 317–333, Sept. 2004.

[56] G. Borgefors, "Distance transformations in arbitrary dimensions," *Computer Vision, Graphics, and Image Processing*, vol. 27, pp. 321–345, Sept. 1984.

[57] Wikipedia, "Quadrotor — wikipedia, the free encyclopedia," 2013.

[58] P. A. D. guide, "https://projects.ardrone.org/."

[59] ROS, "Robot operating system, http://www.ros.org."

[60] PCL, "Point cloud library, http://pointclouds.org."

[61] Wikipedia, "Hayes command set — wikipedia, the free encyclopedia," 2013.

[62] AutonomyLab, "https://github.com/autonomylab/ardrone_autonomy."

[63] M. Muja and D. G. Lowe, "Fast approximate nearest neighbors with automatic algorithm configuration," in *In VISAPP International Conference on Computer Vision Theory and Applications*, pp. 331–340, 2009.

[64] J. E. Bresenham, "Algorithm for computer control of a digital plotter," *IBM Systems Journal*, vol. 4, no. 1, pp. 25–30, 1965.

[65] G. Klein and D. Murray, "Improving the agility of keyframe-based SLAM," in *Proc. 10th European Conference on Computer Vision (ECCV'08)*, (Marseille), pp. 802–815, October 2008.

[66] R. Newcombe, S. Lovegrove, and A. Davison, "Dtam: Dense tracking and mapping in real-time," in *Proc. of the Intl. Conf. on Computer Vision (ICCV), Barcelona, Spain*, vol. 1, 2011.

[67] R. O. Castle, G. Klein, and D. W. Murray, "Video-rate localization in multiple maps for wearable augmented reality," in *Proc 12th IEEE Int Symp on Wearable Computers, Pittsburgh PA, Sept 28 - Oct 1, 2008*, pp. 15–22, 2008.