Nelson de Moura Martins Gomes e Gustavo Vilela Momenté

UAV navigation using SLAM

São Paulo

2015

Nelson de Moura Martins Gomes e Gustavo Vilela Momenté

UAV navigation using SLAM

Monografia apresentada no Departamento de Engenharia Mecatrônica e Sistemas Mecânicos da Escola Politécnica da Universidade de São Paulo para obtenção do título de Engenheiro. Área de Concentração: Engenharia Mecatrônica.

Universidade de São Paulo – USP

Escola Politécnica

Graduação em Engenharia Mecatrônica

Supervisor: Thiago Martins

São Paulo

2015

Nelson de Moura Martins Gomes e Gustavo Vilela Momenté
UAV navigation using SLAM/ Nelson de Moura Martins Gomes e Gustavo Vilela Momenté. – São Paulo, 2015-
68 p. : il. (algumas color.) ; 30 cm.

Supervisor: Thiago Martins

Trabalho de Conclusão de Curso – Universidade de São Paulo – USP
Escola Politécnica
Graduação em Engenharia Mecatrônica, 2015.

1. SLAM 2. GPU 3. PTAM 4. ROS 5. AR.Drone I. Escola Politécnica. Departamento de Engenharia Mecatrônica e de Sistemas Mecânicos. II. Gomes, Nelson de Moura Martins III. Momenté, Gustavo Vilela

Nelson de Moura Martins Gomes e Gustavo Vilela Momenté

UAV navigation using SLAM

Monografia apresentada no Departamento de Engenharia Mecatrônica e Sistemas Mecânicos da Escola Politécnica da Universidade de São Paulo para obtenção do título de Engenheiro. Área de Concentração: Engenharia Mecatrônica.

Trabalho aprovado. São Paulo, ___ de dezembro de 2015:

Thiago Martins
Orientador

Professor
Convidado 1

Professor
Convidado 2

São Paulo
2015

Este relatório é apresentado como requisito parcial para obtenção do título de Engenheiro na Escola Politécnica da Universidade de São Paulo. É o produto do nosso próprio trabalho, exceto onde indicado no texto. O relatório pode ser livremente copiado e distribuído desde que a fonte seja citada.

Abstract

Given the growing demand for autonomous vehicles, our project proposes a solution to SLAM (Simultaneous Location And Mapping) using heterogeneous parallel computing. Our objective is to study main solutions for monocular-SLAM, i.e, SLAM using a monocular camera. Following, we propose some implementation modifications that adds parallel processing to it. Results and data were collected using two commercially available drones an AR.Drone 1.0 and an AR.Drone 2.0, parallel computations were carried on a CUDA-enabled Nvidia GPU and communication with the drones were done using ROS (Robot Operating System).

Keywords: Drone, SLAM, ROS, GPU, PTAM.

Resumo

Atendendo a crescente demanda por veículos autônomos, nosso projeto propõe uma solução para SLAM (*Simultaneous Location And Mapping*) utilizando processamento heterogêneo. Realizamos um estudo do estado da arte dos principais métodos de resolução do SLAM monocular, ou seja, SLAM usando somente uma câmera monocular. Em seguida, foram propostas modificações à implementação para adicionar processamento em paralelo. Coletou-se dados e resultados através de dois modelos de drones comercialmente disponíveis um AR.Drone 1.0 e um AR.Drone 2.0, os cálculos paralelos foram feitos em uma GPU Nvidia com o uso de CUDA, finalmente o interfaceamento com os drones foi feito utilizando o middleware de robótica ROS (*Robot Operating System*)

Palavras-chave: Drone, SLAM, ROS, GPU.

List of Figures

Figure 1 – Amazon Dash reader 25
Figure 2 – Amazon Prime Air drone 26
Figure 3 – Beam™ presence system 27
Figure 4 – Nvidia TX1 28
Figure 5 – Proposed architecture 30
Figure 6 – AR.Drone 2.0 31
Figure 7 – PTAM window 47
Figure 8 – Mapping thread, taken from (KLEIN; MURRAY, 2007) 49
Figure 9 – Tum package 53
Figure 10 – Feature detection examples for each tested algorithm. 59
Figure 11 – Frames per second versus the number of features for each algorithms. . 59

List of abbreviations and acronyms

SLAM	Simultaneous Location And Mapping.
GPS	Global Positioning System.
SIFT	Scale Invariable Feature Transformation.
SURF	Speeded-Up Robust Features.
BRIEF	Binary Robust Independent Elementary Features.
PCA	Principal Component Analysis.
ORB	Oriented FAST and Rotated BRIEF.
BRISK	Binary Robust Invariant Scalable Keypoints.
NN	Nearest Neighbor.
JCBB	Joint Compatibility Branch and Bound.
MHT	Multiple Hypothesis Tracker.
EKF	Extended Kalman Filter.
UKF	Unscented Kalman Filter.
GPU	Graphics Processor Unit.
CUDA	Compute Unified Device Architecture.
ROS	Robotic Operational System.
IMU	Inertial Measurement Unit.
OpenCV	Open Computer Vision.
SFM	Structure From Motion.
BA	Bundle Adjustment.
SMC	Sequential Monte Carlo.
MCMC	Markov Chain Monte Carlo.
LM	Levenberg-Marquardt.

Contents

1 INTRODUCTION . . . 17
1.1 Navigation systems . . . 17
1.2 SLAM problem formulation . . . 18
1.3 Feature detection . . . 19
1.4 Data association . . . 20
1.5 Sensor data fusion . . . 21
1.6 SLAM solution proposition . . . 23

2 MOTIVATION . . . 25
2.1 Amazon Dash and Prime Air . . . 25
2.2 Car automation . . . 26
2.3 Presence systems . . . 26
2.4 Jetson TX1 . . . 27
2.5 Closing Arguments . . . 28

3 REQUIREMENT ANALYSIS . . . 29
3.1 Design Requirements . . . 29
3.2 Functional Requirements . . . 29
3.3 Non-Functional Requirements . . . 29
3.4 Performance Requirements . . . 30
3.5 Proposed Architecture . . . 30
3.6 Required material . . . 30

4 FEATURE EXTRACTION . . . 33
4.1 Detection and description algorithms . . . 33

5 ROBOT'S PATH ESTIMATION . . . 35
5.1 Particle Filter . . . 36
5.1.1 Posterior distributions . . . 36
5.1.2 Monte Carlo Sampling . . . 37
5.1.3 Sequential Importance Sampling . . . 38
5.1.4 Structure of the FastSLAM algorithm . . . 39
5.1.4.1 Image depth estimation . . . 40
5.1.4.2 Drone odometry and state prediction . . . 41
5.1.4.3 EKF filter . . . 42
5.1.4.4 Sampling the proposal distribution . . . 43

5.1.4.5 Data association method . . . 44
5.1.4.6 Resampling method and estimation of pose . . . 44
5.2 Bundle Adjustment . . . 45
5.3 PTAM framework . . . 46
5.3.1 Tracking . . . 46
5.3.1.1 Map projection points . . . 47
5.3.1.2 Patch Search . . . 48
5.3.1.3 Pose Update . . . 48
5.3.1.4 Repetition of the patch search and pose update . . . 48
5.3.2 Mapping . . . 49
5.3.2.1 Map expansion . . . 49
5.3.2.2 Bundle adjustment and data association refinement . . . 50
5.4 Our choice of algorithm . . . 50

6 IMPLEMENTATION . . . 53
6.1 TUM AR.Drone ROS package . . . 53
6.1.1 Scale information . . . 53
6.1.2 Sensor information fusion . . . 54
6.1.3 Control . . . 56
6.2 Analysis of the TUM package . . . 56

7 TEST AND RESULTS . . . 59
7.1 Comparison between feature detection algorithms . . . 59
7.2 FAST Benchmark . . . 59

8 ANALYSIS . . . 61
8.1 Comparison between feature detection algorithms . . . 61
8.2 PTAM at GPU tests . . . 61

9 CONCLUSION . . . 63

BIBLIOGRAPHY . . . 65

1 Introduction

Our primary intention on this project is to address the localization and mapping problem. The state-of-the-art methods used nowadays to implement the most common and efficient solutions will be studied, altogether with the types of algorithms used in the solution and their possible variations.

Our main goal in this work is to apply the parallel power that GPUs can offer in the resolution of the SLAM problem. Since 2000, Nvidia created a framework to allow programmers that are not familiar with the graphic pipeline inside a GPU to do general programming using its inherent parallelism. Since then, many papers were published documenting speed-ups with the use of GPU in some problems.

The SLAM problem have a parallel structure itself, which can be seen if we consider the landmarks detected in the environment. If some estimation process allow us to estimate each landmark position independently from one to another, they could be updated simultaneously. As it will be seen in the next chapters, the particle filter makes that possible. Other ways to explore parallelism in the SLAM will be studied too.

First, in this chapter, navigation will be defined and its importance will be discussed, then the SLAM problem definition will be explained and one example will be given. The three modules that a typical SLAM solution have will be discussed: feature detection, data association and data filtering.

1.1 Navigation systems

First of all, we need to define what is navigation. Navigation is the operation in which a trajectory is traced considering a point of arrival knew *a priori* and the path made to get there. The trajectory is related with the state vector of the vehicle at a given time and the guidance with the observations made from the sensor used to capture the trajectory. A special category of navigation that will be addressed here is the autonomous navigation, which consists of the implementation of a navigation system that does not depend on external control (HOFMANN-WELLENHOF; LEGAT; WIESER, 2011).

The relevance that navigation systems has in our lives is bigger than we realize. Traveling to another city, finding a store on a map application or driving a boat or a ship, all these tasks use navigations systems. In the 15th century the western world, represented by some European countries (Portugal, Spain and in the 16th England and Netherlands too), started to explore the Atlantic and Indian ocean, at the time surrounded by myths and fantasies. All these countries had experience in navigation in the Mediterranean sea

or in the Northern sea, where ships could follow the coast to reach their destination.

But in the ocean there aren't any landmarks to be followed. The navigation using the stars, already known since the early ages became extremely important, essential in transoceanic expeditions. An analogy between the transoceanic transport and the autonomous robot navigation can be made: in a ocean the stars are used to orientate ship's movement, so some type of environmental characteristic can also be used to orientate a robot in an unknown environment (FEDER, 1999).

For airplanes the same problem exists, to follow landmarks is very difficult or even impossible in the cases of supersonic planes. Before the GPS advent, maps were used to guide commercial airplanes and combat planes flew in low altitudes, therefore they were able to use landmarks. With the GPS, which is today's most used navigation system, the non-autonomous navigation can be considered a solved problem in most cases, since all aerial and aquatic vehicles and most part of the terrestrial vehicles can obtain an accurate and precise trajectory.

But in the domain of the autonomous navigation system in closed unknown environment a robust solution is far from being available. Today's best candidate solution is the family of SLAM algorithms, that maps the environment and locate the robot independently of external agents. In theory SLAM is well defined, as it will be shown in the next section, but its implementation is still under scientific research (DURRANT-WHYTE; BAILEY, 2006).

1.2 SLAM problem formulation

Due to computational limitations of embedded systems and to noises that can arise in sensor readings, a stable and affordable SLAM solution has not been found yet. Noise can be caused, for example, by cheap sensors or environmental influence. If the sensor information is used without any treatment to compensate these errors, the dead reckoning operation, i.e., to calculate the robot's position using a previously determined position and an estimated speed information over an elapsed time, will accumulate a significant amount of imprecision. Normally this situation happens with a robot that uses only an odometer to retrieve information about its position.

To diminish these errors one option is to use another type of observation, one that tries to match the position of previously recorded characteristics of the environment with current observations. This is how SLAM (Simultaneous Location and Mapping) determines the target position, and as a consequence creates a map of the environment, that is, the location of all know environment characteristics (DURRANT-WHYTE; BAILEY, 2006). Together with this structure the probabilistic theory can be used to determine the position and orientation of the robot and of all the environment characteristics, that are usually

called landmarks.

Originally three main directions of the SLAM implementation can be identified: topological, grid-based and feature-based approaches. In the first the world is modeled simply by graphs and nodes, and to detect the location of the robot is necessary to recognize that the robot is in a determined node. This is the method more indicated for large environments, but it suffers from a clear drawback: the environment recognition is rather difficult to be made (JENSFELT; CHRISTENSEN; ZUNINO, 2002).

The grid-based approach consists in the construction of a multidimensional field that maintains stochastic estimates of the occupancy state of each cell (ELFES, 1989). But the inherent problem with this approach is that it is computationally expensive and memory consuming.

Feature-based methods are definitely the most used ones today, because they are less computationally expensive than grid-based and the placement recognition is easier than its topological approach counterpart. More about this approach will be discussed further ahead.

One usual problem that stem from the use of SLAM is high computational usage. Many types of solutions have been proposed over the years to try to diminish this problem. Two types of techniques are: optimal techniques, aiming the reduction of the required computation maintaining the quality of the estimation; and conservative ones that tries to maintain the quality of the estimation while reducing the computational demands (BAILEY; DURRANT-WHYTE, 2006).

1.3 Feature detection

To facilitate the comparison between frames one can decompose the image in keypoints. The ideal keypoints detector identify singularities in the frame in such a way that the same result is obtained repeatably, even with a change of the viewpoint (LEUTENEGGER; CHLI; SIEGWART, 2011). A degree of invariance to rotation and scale transformation, affine transformation, noise and luminosity is also desired (ALAHI; ORTIZ; VANDERGHEYNST, 2012). A keypoint detector with all these characteristics does not exist, the available ones adopt a compromise between these characteristics.

To represent these keypoints detectors two types of descriptors can be used, a binary descriptor or a histogram descriptor (ALAHI; ORTIZ; VANDERGHEYNST, 2012). The combination between the image keypoints and descriptor makes it possible to compare images and subtract the features necessary for the SLAM update. The choice of which image keypoint detector to use is necessary in SLAM because the robot that uses it sometimes does not have the same requirements. For example, for a terrestrial vehicles

rotation of the image is not an important constraint, but for an aerial vehicle it is.

There are many types of image keyframe detector, most prominent ones will be walked through here. Certainly the most popular is SIFT (Scale Invariable Feature Transformation), which is invariant to scale and rotation transformation and robust to luminosity change (LOWE, 2004). Unfortunately, it is too computationally expensive to be executed in real time. Inspired in this algorithm a simplification was made, creating the most used image keypoint descriptor.

The SURF (Speeded-Up Robust Features) is definitely the most used, because it performs similarly to SIFT but it can be implemented for real time execution. This detector focuses in maintaining scale and rotation invariance, moreover, it also shows some resistance to skew, anisotropic and perspective distortion (BAY et al., 2008), despite having a higher sensibility to luminosity changes than SIFT.

Another important image decomposition algorithm is BRIEF(Binary Robust Independent Elementary Features). The high dimensionality of SIFT does not allow real time execution. One option to solve this problem is dimensional-elimination methods, for instance PCA (Principal component analysis), to decrease the dimension of the descriptor. However, it is possible to eliminate the dimensionally reduction step by creating a short binary descriptor, exactly the idea behind BRIEF (CALONDER et al., 2010). It also serves as basis for two others algorithms, ORB (Oriented FAST and Rotated BRIEF) (MUR-ARTAL RAÚL; TARDÓS, 2015) and BRISK (Binary Robust Invariant Scalable Keypoints) (LEUTENEGGER; CHLI; SIEGWART, 2011).

Another feature extraction algorithm that is worth mentioning is the FAST (Features from Accelerated Segment Test), used to detect corners in images. The biggest advantage of this feature detection method is its performance, since it is much lighter than SURF, for example (ROSTEN; DRUMMOND, 2006). This method is used in the PTAM framework (KLEIN; MURRAY, 2007), which today is probably the most successful SLAM solution.

1.4 Data association

Data association consists simply in relating feature readings given by one or more sensors with the elements already present on the map, therefore this operation is responsible for the update of the environment information kept in the map. The complexity is exponential considering the number of sensor measurements, and the solution space for the right association between measurements and features growth with the clutter of the environment and the imprecision of the sensor (NEIRA; TARDÓS, 2001).

The inherent problem with SLAM is the fact that the information given by the

sensors normally is a nonlinear function of the relative difference between the position of the sensor/robot and the position of the feature, thus this combination of unknowns needs a joint estimation scheme (DAVEY, 2007). The data association algorithm is normally divided in two steps: a test to determine the coherence between the observation and the feature given an estimation of the vehicle position; and a selection criteria to choose the best possible match.

The simplest method to associate observations is the gated nearest neighbor. The squared innovation test is used to determine which hypothesis are coherent and the set with the smallest Mahalanobis distance is chosen. The complexity of this algorithm is linear considering the number of features and observations, but it is too permissive, therefore, it allows matches between landmarks and spurious points sometimes.

The JCBB (Joint Compatibility Branch and Search) traverses the information tree, which is the representation of the solution space for the data association, searching the solution with the smallest number of null jointly compatible pairing. The method has a very restrictive criteria, preventing the solution space explosion due to the growth of the robot position error. The most immediate problem presented in its implementation its quadratic complexity (NEIRA; TARDÓS, 2001).

Another data association method that has as objective to reduce the number of mismatches is the Multiple Hypothesis Tracker. It applies the idea of expectation-maximization algorithm to multi-target data association with linear complexity in the number of targets (DAVEY, 2007). Simply put, it considers multiple hypotheses of data association through time, conserving the ones that are most likely to be true.

The nearest neighbor and the JCBB are more commonly used when the vision sensor is a sonar or a radar. In the case of cameras, many SLAM applications uses a region of pixels around the interest point to execute data association. This approach is fairly common with the use of Structure from Motion algorithms to implement the SLAM.

1.5 Sensor data fusion

In an ideal world the odometry data would suffice to determine robot's localization, but fused with this measure there is an inherent noise, that produces an error in the robot's position every time the observation of the odometer is used to update its position. To correct this error propagation, one idea is to use another type of observation, one that tries to match the position of actual landmarks with the old positions. This how SLAM (Simultaneous Location and Mapping) determines the target position, and as a consequence creates a map of the environment considering the landmarks locations (DURRANT-WHYTE; BAILEY, 2006).

However, the landmarks measurements also have noise, so a filter is commonly used to cope with the odometer and other sensors errors. Some different SLAM implementations use different types of filters. The most common filter is the EKF (Extended Kalman Filter), a filter based on the Hidden Markov Model that uses the first order Taylor series to linearize nonlinear observations functions. But it considers that the observation and the state prediction have a Gaussian distribution. On the linear case this filter has an optimal solution but in the nonlinear case the approximation can produce significant errors (WAN; MERWE, 2001).

One alternative of this type of filter is the Unscented Kalman Filter (UKF). It uses a deterministic sampling approach to represent the Gaussian random variable, and it can determine posterior mean and covariance accurately to the third order in the Taylor series without any additional computational effort than that expended by the EKF. It solves the two principal drawbacks of the EKF: nonlinear approximation and the derivation of the Jacobian (JULIER; UHLMANN, 1997).

Nowadays, particle filters can be applied to obtain a more general solution, since it is non-Gaussian and non-linear, using Monte Carlo based methods. Instead of represent the density function by itself, a set of points chosen randomly from the distribution is used (DELLAERT et al., 1999), just like intuitively one filter using Monte Carlo method should work. It is also simpler to implement and in strong nonlinear situation more suitable than Kalman-based filters. Every type of filter considers different hypothesis and demand different quantities of computational processing, and the implementation of the SLAM solution is linked directly with the type of the filter.

Filtering implements the concept of localization proposed by probabilistic robotics. The idea is to represent the uncertainty of the robot's position by a probabilistic distribution, not trying to calculate a discrete position point (THRUN; BURGARD; FOX, 2005). Using the probabilistic approach to determine the robot's position also makes easier to apply the sensor uncertainty in the position calculation.

The concept described above gives the solution the propriety to deal with uncertain information. Two characteristics that also are desirable is the capacity to deal with ambiguities and the possibility to integrate sensor readings from different types of sensors (BURGARD et al., 1996). The filter makes it possible to use sensor readings disregarding the source of the information.

Filtering is normally the usual option for the robot's position estimation. But another approach, more popular in the augmented reality community, can also be applied to our problem. This method is known as bundle adjustment and consists of using batch optimization to solve the position problem. The SLAM problem is the real-time case of the Structure from Motion problem, which represents the problem of estimate 3D structures from 2D image sequences. Moreover, bundle adjustment also sparsifies

the problem differently, defining keyframes of the environment instead of propagating probabilities through the map (STRASDAT; MONTIEL; DAVISON, 2012).

1.6 SLAM solution proposition

In our project we will study the implementation of the SLAM for a aerial vehicle, more specifically the AR.Drone. Our main goal is to use the GPU to implement most algorithms that can be executed using parallelism. This can solve the high demand of the feature detection algorithm and the data association process. All these algorithms will be constructed over the implementation of the SLAM solution proposed by (ENGEL; STURM; CREMERS, 2014). Our objective is to implement the feature detection and the data association algorithms and use the communication and drone's control previously implemented.

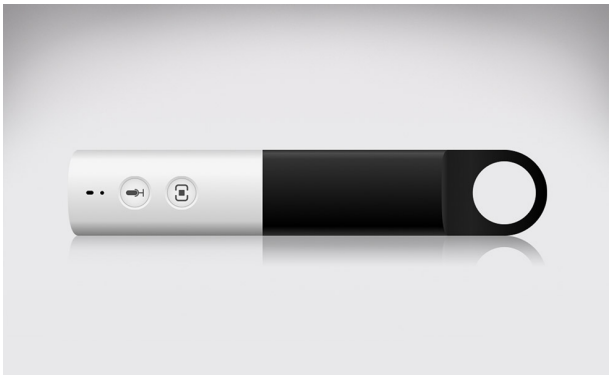
2 Motivation

The SLAM problem is very popular in the scientific community and the reason is very simple: it has commercial applications in many economic sectors. In the following sections three examples of possible applications will be shown.

2.1 Amazon Dash and Prime Air

The Amazon Dash is an optical reader for bar codes. The idea behind it is to use it to scan products that need to be bought, and all the registered barcodes are sent to Amazon Fresh for purchase and delivery. This product is still in test phase, so all the owners were previously invited to test it (AMAZON, 2015a). This technology does not apply in the SLAM solution, but the Prime Air program implements the SLAM solution in an entire autonomous solution to sell and deliver products.

Figure 1 – Amazon Dash reader



The objective of the Prime Air program is use drones to deliver products to costumers hours after the selling (AMAZON, 2015b). In that context the SLAM solution can be used to generate a map for a city and record the route between an amazon deposit and the costumer's hour or a delivery point. Every time that a delivery needs to be done the same route can be used for the drone, without the use of GPS. And with the map generated by all the available drones new routes can be generated off-line, or pieces of existing routes can be used to generate new ones.

As we will see in this project SLAM is too computationally expensive to be executed in a low-cost embedded processor, but recent advancements in GPU embedded hardware can bring a change to this situation. We propose to implement part of the treatment in the graphics unit comes from the possibility that in a near future embedded GPU will be able to execute the parallelized part of the code with low energy consumption and in real

Figure 2 – Amazon Prime Air drone

time. An example of this is the Nvidia Tegra X1, which is already been using in advanced driver assistance (ADAS), computer vision and deep learning (NVIDIA, 2015).

2.2 Car automation

Autonomous cars have always been pictured like a feature from the future in movies. In reality, autonomous navigation systems for terrestrial vehicles are been researched for more than a decade. With the increasing automation in cars it will be possible to create instruments to prevent accidents and optimize the use of cities roads and highways. This technology can cause a revolution in logistics and transportation systems and security.

In this context the SLAM solution would be applied for example to generate paths that are used frequently, e.g., the daily commute path, or the path to a supermarket. This solution could be implemented completely inside the car, without the need for network connection. Actually the solution proposed by (MUR-ARTAL RAÚL; TARDÓS, 2015) seems very robust given its results with KITTI and TUM RGB-D benchmarks.

The solution proposed by the ORB-SLAM team (MUR-ARTAL RAÚL; TARDÓS, 2015) dos not use GPU acceleration, so it can even be improved. There are some details that need to be worked before a real implementation, for example, if a map created *a priori* is used some mechanism to update the map considering changes in the environment must be present, with a way to propagate updates to all other users. But even with pending modifications SLAM is an excellent candidate to integrate the navigation system of future's cars.

2.3 Presence systems

The use of presence systems is growing every day thanks to the globalization of the production chain. The idea is to provide a tool to communicate with people in meetings

and in work situation in real-time, in the most natural way possible and with high image quality together with the capacity of moving depending only on the user. One example of this type of product is the Beam™, produced by Suitable Technologies.

Figure 3 – Beam™ presence system

A SLAM-based solution can be used in this situation to navigate the system from one point to another independent of the user, like for example from an office to a meeting room. This can save a significant amount of time for the user, who can dedicate its time and attention entirely to the communication with the surrounding people, that is the main reason of this product creation.

The Beam™ already have a wide-angle high-resolution camera, but does not have enough processing power to implement the SLAM solution at this moment. And another question is: if the GPU is used to accelerate the implementation its energy footprint must be small, because the system has to use the battery to be able to stream the video and to power the locomotion system, but even so it is a real world solution that can benefit from the implementation of an autonomous navigation system.

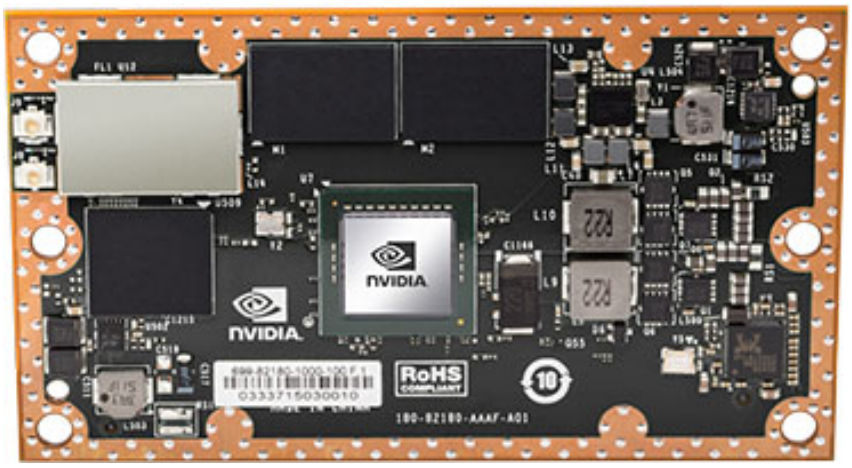
2.4 Jetson TX1

The Jetson TX1 is an embedded GPU system recently launched by Nvidia that illustrates the portability of GPU processing power. It has 256 CUDA processing cores and delivers over 1 TeraFlops of performance. Nvidia also provided a developer kit specially

for this chip. The processing power of this chip is enough to solve many problems of deep learning, computer vision and GPU computing.

The size of this chip is the same of a credit card (Fig. 4), and considering the performance delivered by it, many parallel problems can take advantage of its structure to increase their performance.

Figure 4 – Nvidia TX1



2.5 Closing Arguments

Considering the examples given the SLAM solution can be object of direct implementation in many economic sectors. We do not even consider indirect implementation, i.e., problems that can use parts of the SLAM solution to solve other ones, for instance, the tracking of objects. Normally a CAD model is used to detect and track an object, but with the development of SLAM solutions some discovery or breakthrough can be used to increase the robustness or to decrease the computational requirements, or to detect the object without needing a CAD model. Therefore, the implementation of SLAM solutions is important for many research areas and its study is very actual economically and scientifically.

3 Requirement analysis

To better understand all the requirements of the project and the necessary algorithms and tools that will be used to achieve them, this section will describe then and the expected results. Moreover, a simple system architecture showing communications and requires materials is unveiled.

3.1 Design Requirements

- The AR.Drone used wasn't designed to carry heavy payloads, therefore, a crucial requirement is that no further sensors can be used, hence, only the ones already embedded with it can be used. They are : IMU, ultrasound sensor, wide-angle camera and high-speed camera;
- As we will be using CUDA, ROS and possibly OpenCV, most of the project will be write either in C++ or Python;
- For feature point extraction GPU based algorithms should be used in detriment of CPU based ones;

3.2 Functional Requirements

- Receive video feed from the cameras;
- Map an unknown region while flying (considering static landmarks);
- Perform localization in an unmapped region;
- Send controls to the drone, considering that an off-board approach is adopted in this project;
- Detect feature points using images from the camera, making possible to match new frames with old ones.

3.3 Non-Functional Requirements

- Use parallel algorithms where possible.
- Implement all the parallel algorithms preventing problems like race conditions or starvation in the GPU.

3.4 Performance Requirements

- The system must function as close as possible to real-time;
- The system should work at map of size ld^2 meters, where ld is the distance covered by the drone while flying in a straight line using only one battery charge;

3.5 Proposed Architecture

A simple schematic of the major pieces and communications in our system is shown in Fig. (5). Basically, the AR.Drone will send its measurements and video feeds to a computer through wireless network, the computer by its turn will send control signals to the drone and fulfill the remainder of the requirements by doing itself the computations or delegating to a CUDA GPU when suitable.

Figure 5 – Proposed architecture

3.6 Required material

Given the proposed architecture the following hardware are necessary for the development and deployment of the project:

- An AR.Drone 1.0 or 2.0;
- A network that allows wireless communications;
- A computer with CUDA enabled Nvidia GPU.

Figure 6 – AR.Drone 2.0

4 Feature extraction

Concerning the feature detection and image description, There are dozens of algorithms that can be used to extract interest points (given a criterion) and to describe an image. As discussed in Section 1.3 and in (LOWE, 1999), a good feature detector should be invariant to some degree to illumination, 3D projective transformation and any type of object variation. But at the same time it should have the capacity to detect distinctive object's characteristics.

In a feature detection algorithm two operations dominates: keypoint localization and feature description. Keypoint localization is the identification of distinctive characteristics in a image. In this part the keypoints, that should be detected under different conditions, are localized. To allow scale invariance in detection, normally, feature extraction is made using the same image at different scales. Finally, an affine region detector is added to give the invariance to multiple viewpoints (GRAUMAN; LEIBE, 2010).

The second step in a feature detection algorithm is the feature description. They can be divided in two subgroups, considering how the features descriptors are constructed: detectors based in histogram descriptors and based in binary descriptors. The former one describes the characteristics of adjacent points (BEHLEY; STEINHAGE; CREMERS, 2012) and the latter uses a binary vector to represent the keypoint appearance (CALONDER et al., 2010).

4.1 Detection and description algorithms

The two most popular algorithms are SIFT and SURF. The first one uses a difference of Gaussians detector with a histogram descriptor, but its complexity does not allow real-time execution. To solve this problem SURF was developed as an approximation to allow real-time execution (BAY et al., 2008).

Another popular feature extraction algorithm is ORB. This method was created as another replacement for SIFT, aiming real-time execution, similar matching performance and variance caused by image noise (MUR-ARTAL RAÚL; TARDÓS, 2015). This algorithm uses the FAST detector (ROSTEN; DRUMMOND, 2006) and the BRIEF descriptor (CALONDER et al., 2010) to extract features. But this algorithm is less resistant to scale and rotation invariance than SURF or SIFT.

Many others algorithms exists, for instance, BRIEF, BREAK, KAZE, etc. We will remain discussing only the SURF and ORB because we want a real-time execution and both algorithms have GPU implementation in the OpenCV library. Using the ROS

interface with the AR.Drone some tests with both algorithms were made to decide which one we should use in our program.

The FAST feature detector is a method develop to extract keypoints, like all other ones, but it was conceived to be less computational demanding than the other algorithms. Around a candidate point, some pixels are compared to the intensity of this point and a threshold predefined; if there is a certain number of points outside the interval $[I_p - t]$, I_p being the intensity of the candidate point and t the threshold value, then this candidate is declared a corner, or a keypoint (ROSTEN; DRUMMOND, 2006).

This algorithm have some weakness too, for example it detects multiple features adjacent to other previously detected. This problem is solved by a non-maximum suppression, that suppresses all the point with exception of only the maximum one considering a arbitrary metric in a local region (NEUBECK; GOOL, 2006). The FAST is used for feature extraction inside the PTAM framework.

5 Robot's path estimation

SLAM can be perceived as a particularization of the SfM (Structure From Motion) problem, which refers to extracting 3D representations using 2D image sequences (AZARBAYEJANI; PENTLAND, 1995). But SLAM has one very important constraint: real-time execution. That demands an almost completely new solution approach, since SfM is generally solved offline using batch optimization (STRASDAT; MONTIEL; DAVISON, 2012).

All the identifiable features in the images are, by hypothesis, related to some 3D entity in the space, captured in consecutive video frames. With the rigidity assumption it can be asserted that the feature motion is due purely to the camera, solving the camera and the feature localization problem. Both locations can be further refined with new images and new features. Bundle adjustment is defined by the constant update of the positions due to new measurements (DAVISON, 2003).

In the most popular form of bundle adjustment, the keyframe BA, only part of the past data is maintained, therefore most of it iss simply discarded. The keyframes are defined automatically or heuristically and must represent the features in the environment. In the filtering approach only the features are retained, which causes the map to be heavily connected, since after every new frame new connections between features are created. As one can deduce, the computational cost of propagating the joint distributions scales poorly in comparison with the number of features observed.

STRASDAT; MONTIEL; DAVISON state that bundle adjustment is more efficient than the filtering approach considering the complexity cost and its better precision. But, as JULIER; UHLMANN shows inconsistencies can be produced if the EKF filter is used, due to the linearisation of the process and observation model. Therefore, the comparison between the bundle adjustment and a filtering technique should use another filter.

Another possible approach for the SLAM algorithm implementation is the use of filtering methods (STRASDAT; MONTIEL; DAVISON, 2012). This domain is more applied to sonar and radar sensors, but some applications for cameras also exist. Since filtering considers the correlation between landmarks and observation the tracking and mapping is not separable, so a complete iteration must comply with real-time restrictions.

As the probabilistic method were developed for robotic use, odometry information is always available, that used in data association helps to avoid wrong matches. Without it, data association needs to use complex methods, like covariance-driven gating, joint compatibility branch and bound (JCBB) or RANSAC (RANdom SAmple Consensus), which are very computational demanding (KLEIN; MURRAY, 2007).

In next sections FastSLAM and PTAM will be discussed.The former is the implementation of the particle filter using a Rao-Blackwellized marginalization and the latter uses bundle adjustment for SLAM due to its performance and precision (STRASDAT; MONTIEL; DAVISON, 2012).

5.1 Particle Filter

Since its creation the particle filter has become a popular method to search for the solution of optimal estimation problems in non-linear and non-Gaussian situations (DOUCET; JOHANSEN, 2009). In contrast with the Extended Kalman Filter, which applies a linearisation step, it does not simplify in any means the function estimated, converging to a result that the EKF does not or getting a better estimation. Of course, this comes with a cost: the computational cost is bigger than the EKF's.

Our goal is to achieve the estimation of an unknown quantity using observations. Normally one have already some information about the system, thus the estimation becomes simple applying Bayes theorem. But if one wants to run the inference sequentially and on-line some sort of update mechanism is necessary. Analytical answers for these models are rare. For example, if the system is modeled as a linear Gaussian state-space the Kalman Filter is the optimal estimator, or if it is modeled as partially observed and finite-state Markov Chain the HMM (Hidden Markov Model) filter is the optimal estimator (SMITH et al., 2013).

For the non-linear and non-Gaussian cases the EKF and the Sequential Monte Carlo (SMC) methods are alternatives for the estimation.

5.1.1 Posterior distributions

The main goal of the SMC is to estimate the posterior distribution $p(x_{0:t}|y_{0:t})$ and the following expectation for some function $f_t : \chi^{(t+1)} \rightarrow \mathbb{R}^{n_{f_t}}$:

$$I(f_t) = \mathbb{E}_{p(x_{0:t}|y_{1:t})}[f_t(x_{0:t})] = \int f_t(x_{0:t}p(x_{0:t}|y_{1:t})dx_{0:t} \tag{5.1}$$

The joint distribution $p(x_{0:t}|y_{1:t})$ can be calculated as follows:

$$p(x_{0:t-1|y_{1:t-1}}) = p(x_{0:t}|y_{1:t}) \cdot \frac{p(y_{1:t+1}|x_{0:t+1})p(x_{t+1}|x_t)}{p(y_{1:t+1}|y_{0:t})} \tag{5.2}$$

A recursive relation that is used to calculate the marginal distribution $p(x_t|y_{1:t})$ is the following:

$$p(x_{t+1}|y_{1:t}) = \int p(x_{t+1}|x_t)p(x_t|y_{1:t})dx_t \tag{5.3}$$

$$p(x_{t+1}|y_{1:t+1}) = \frac{p(y_{t+1}|x_{t+1})p(x_{t+1}|y_{1:t})}{\int p(y_{t+1}|x_{t+1})p(x_{t+1}|y_{1:t})dx_{t+1}} \tag{5.4}$$

As one can observe, all the integrals in the equations (5.1), (5.2), (5.3) and (5.4) have a high dimensional order. Equations (5.3) and (5.4) are know as the Bayesian filter, the first equation being the prediction step and the second the update step. In our case, the state variable x_t will have the robot pose and all the features poses, therefore any attempt to calculate these integrals are not feasible.

5.1.2 Monte Carlo Sampling

Consider N independent and identically distributed (now on represented by the abbreviation i.i.d.) random samples $x_{0:t}^{(i)}; i = 1, \ldots, N$ from the conditional distribution $p(x_{0:t}|y_{1:t})$. This distribution can be estimated using the equation:

$$P_N(dx_{0:t}|y_{0:t}) = \frac{1}{N}\sum_{i=1}^{N}\delta_{x_{0:t}^{(i)}}(x_{0:t}) \tag{5.5}$$

Using the equation (5.4) with (5.23) we have the following unbiased estimator:

$$I(f_t) = \int f_t(x_{0:t})P_N(dx_{0:t}|y_{1:t}) = \frac{1}{N}\sum_{i=1}^{N} f_t\left(x_{0:t}^{(i)}\right) \tag{5.6}$$

And considering the *Law of Large Numbers*, shown in equation (5.7), one can say that $I(f_t)$ approaches the real expectation value $\mathbb{E}(f_t)$ when the number of samples grows.

$$\lim_{N\to\infty}\frac{1}{N}\sum_{i=1}^{N} g\left(X_i\right) = \mathbb{E}(g) \tag{5.7}$$

As for the variance, if the (5.8) holds, then the Central Limit Theorem, (5.9) is valid.

$$\sigma_{f_t}^2 = \mathbb{E}_{p(x_{0:t}|y_{1:t})}[f_t^2(x_{0:t})] - I^2(f_t) < +\infty \tag{5.8}$$

$$\sqrt{N}[I_N(f_t) - I(f_t)] \longrightarrow_{N\to\infty} \mathcal{N}(0, \sigma_{f_t}^2) \tag{5.9}$$

However, there is an inherent problem with this method: the samples must be from the conditional distribution that one expect to estimate, and in our case, $p(x_{0:t}|y_{1:t})$. But since this method convergence does not depend upon the dimensionality of the integral

estimated it is a good idea to adapt it to our needs. The Sequential Importance Sampling (SIS) is the modified version of the Monte Carlo sampling that generalizes the MC method for recursive use. The Markov Chain Monte Carlo (MCMC) methods are not considered here as candidate since they are not efficient when used in recursive problems (DOUCET et al., 2000).

5.1.3 Sequential Importance Sampling

The idea of the importance sampling is to draw samples from a proposal distribution and re-weight the integral to target the correct distribution. This proposal distribution is called $\pi(x_{0:t}|y_{1:t})$. As the number of samples becomes larger, it approaches the aimed distribution (ARULAMPALAM et al., 2002). Therefore, the equation (5.6) is transformed into (5.10):

$$I(f_t) = \frac{\int f_t(x_{0:t})\omega(x_{0:t})\pi(x_{0:t}|y_{1:t})dx_{0:t}}{\int \omega(x_{0:t})\pi(x_{0:t}|y_{1:t})dx_{0:t}} \tag{5.10}$$

If we compare both equation one may think that they are not equal, but the denominator in (5.10) shows in (5.5) term $\frac{1}{N}$. The weight used to compensate the samples is defined by the equation (5.11):

$$\omega(x_{0:t}) = \frac{p(x_{0:t}|y_{1:t})}{\pi(x_{0:t}|y_{1:t})} \tag{5.11}$$

Applying the importance sampling to the Monte Carlo estimation we have:

$$\hat{I}_N = \frac{\frac{1}{N}\sum_{i=1}^{N} f_t\left(x_{0:t}^{(i)}\right)\omega\left(x_{0:t}^{(i)}\right)}{\frac{1}{N}\sum_{i=1}^{N}\omega\left(x_{0:t}^{(i)}\right)} = \sum_{i=1}^{N} f_t\left(x_{0:t}^{(i)}\right)\tilde{w}_t^{(i)} \tag{5.12}$$

Where the term $\tilde{w}_t^{(i)}$ represent the normalized weight:

$$\tilde{w}_t^{(i)} = \frac{\omega\left(x_{0:t}^{(i)}\right)}{\sum_{j=1}^{N}\omega\left(x_{0:t}^{(j)}\right)} \tag{5.13}$$

From the comparison between the equation (5.12) and (5.6) one can say that the weight operation is a sampling method defined by the following expression:

$$\hat{P}_N(dx_{0:t}|y_{1:t}) = \sum_{i=1}^{N}\tilde{w}_t^{(i)}\delta_{x_{0:t}^{(i)}}(dx_{0:t}^{(i)}) \tag{5.14}$$

To update the weight sequentially the (5.13) needs to be changed. Using the law of total probability, equation (5.15), one gets the expression (5.16):

$$\pi(x_{0:t}|y_{1:t}) = \pi(x_0) \cdot \Pi_{k=1}^{t} \pi(x_k|x_{0:k-1}, y_{1:k}) \tag{5.15}$$

$$\pi(x_{0:t}|y_{1:t}) = \pi(x_t|x_{0:t-1}, y_{1:t})\pi(x_{0:t-1}|y_{1:t-1}) \tag{5.16}$$

Deducing the new weight equation, we have:

$$\omega_t^{(i)} = \frac{p(y_t|x_t^{(i)})p(x_t^{(i)}|x_{t-1}^{(i)})p(x_{0:t-1}^{(i)}|y_{1:t-1})}{\pi(x_t^{(i)}|x_{0:t-1}^{(i)}, y_{1:t})\pi(x_{0:t-1}^{(i)}|y_{1:t-1})} = \omega_{t-1}^{(i)} \cdot \frac{p(y_t|x_t^{(i)})p(x_t^{(i)}|x_{t-1}^{(i)})}{\pi(x_t^{(i)}|x_{0:t-1}^{(i)}, y_{1:t})} \tag{5.17}$$

The choice of the importance sampling distribution interferes in the convergence of the filter, and it is left for the user. Many types of adaptations can and should be done in the SIS method for a valid and efficient implementation, e.g., adding a re-sampling mechanism and treating the degeneracy problem. But explaining just the idea behind the SIS is enough to introduce the particle filter.

5.1.4 Structure of the FastSLAM algorithm

The particle filter has one distinct drawback: the sampling of high dimensional spaces can be inefficient. But with some models it is possible to use the inherent characteristics to simplify the sampling. This is the definition of the Rao-Blackwellised particle filter (DOUCET et al., 2000). Since the FastSLAM uses the EKF to do this marginalization, it is a type of RBPF (Rao-Blackwellised Particle Filter).

The sequence of operations in the FastSLAM algorithm can be resumed into the following list:

1. Get new sensor measurements.
2. Reconstruct features points using the received image.
3. Sample a new robot position.
4. Associate the features in the image with the existing landmarks.
5. Update landmarks position.
6. Calculate the particle's weight.
7. Resample particles.

The FastSLAM can be used with cameras, radars, sonars or any type of sensor. Since our sensor of choice is the monocular camera, the algorithm will be presented with

the necessary adaptations, notably with some changes in comparison with the FastSLAM structure presented in (THRUN; BURGARD; FOX, 2005). A reconstruction step to estimate the feature's depth using the actual frame and the old one is necessary in this case.

Another change is the data association method, when we suggest the NN (Nearest Neighbor), while the reference uses the likelihood maximization. Lastly, our feature initialization is also different from the method proposed in the book. In the next sections the steps can be seen with more details.

5.1.4.1 Image depth estimation

The depth estimation can use a simple reconstruction algorithm based on least squares, implemented inside the OpenCV function *triangulatePoints*. But before the reconstruction the feature extraction and matching must be done.

The first step of the estimation is the extraction of the images features, which can be done using the SURF algorithm, for example. However, in the case that a heavy feature detector algorithm is used it is advisable to search for its implementation in GPU (SURF and GPU have implementation in GPU inside the OpenCV library).

Then, the descriptors of the detected keypoints are matched with the keypoints descriptors from the precedent frame. For the matching we used the brute force matcher, also from OpenCV and implemented in GPU. It returns the vector with the best matches considering given descriptors. Distance is measured in Euclidian space.

Then, with the keypoints from the new image and the old one the reconstruction can be done. The first frame detected by the drone is considered the initial frame, and all the others are calculated multiplying the actual extrinsic camera projection matrix by the rotation and translation matrix, with parameters obtained with the estimation made by the particle filter.

The equations (5.19) and (5.20) represent the initial extrinsic camera matrix and the equations (5.21) and (5.22) represent the update made at each iteration of the reconstruction. The matrix inside (5.21) is the rotation matrix for Tait-Bryan angles with the notation $Z_1Y_2X_3$. Rotation matrix and translation matrix are used, together with the intrinsic matrix, obtained from the camera calibration procedure, to calculate the projection matrix (5.18) at each iteration.

$$P_t = I \cdot [R_t|t_t] \tag{5.18}$$

$$R_0 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{5.19}$$

$$T_0 = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \tag{5.20}$$

$$R_t = R_{t-1} \cdot \begin{bmatrix} c\psi_t c\theta_t & c\psi_t s\theta_t s\phi_t - c\phi_t s\psi_t & s\psi_t s\phi_t + c\psi_t c\phi_t s\theta_t \\ c\theta_t s\psi_t & c\psi_t c\phi_t + s\psi_t s\theta_t s\phi_t & c\phi_t s\psi_t s\theta_t - c\psi_t s\phi_t \\ -s\theta_t & c\theta_t s\phi_t & c\theta_t c\phi_t \end{bmatrix} \tag{5.21}$$

$$T_t = \begin{bmatrix} x_t \\ y_t \\ z_t \end{bmatrix} \tag{5.22}$$

The equation (5.23) shows one form of the system that is solved for two image points (u_1, v_1) and (u_2, v_2) using their respective camera matrices, where m_1 and m_2 represent the elements of each matrix.

$$\begin{bmatrix} r^1_{31}u_1 - m^1_{11} & r^1_{32}u_1 - m^1_{12} & r^1_{33}u_1 - m^1_{13} \\ r^1_{31}v_1 - m^1_{21} & r^1_{32}v_1 - m^1_{22} & r^1_{33}v_1 - m^1_{23} \\ r^2_{31}u_2 - m^2_{11} & r^2_{32}u_2 - m^2_{12} & r^2_{33}u_2 - m^2_{13} \\ r^2_{31}v_2 - m^2_{21} & r^2_{32}v_2 - m^2_{22} & r^2_{33}v_2 - m^2_{23} \end{bmatrix} \cdot \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} m^1_{14} - r^1_{33}u_1 \\ m^1_{24} - r^1_{33}v_1 \\ m^2_{14} - r^2_{33}u_2 \\ m^2_{24} - r^2_{33}v_2 \end{bmatrix} \tag{5.23}$$

5.1.4.2 Drone odometry and state prediction

Using ROS we can get all te sensors measurements directly from the drone using wi-fi. The package used for this task is the *ardrone_autonomy v1.4.1* (SFU, 2015), getting the image from the frontal camera, the linear acceleration and the angular velocity. For

the odometry, the equation (5.24) can represent the state of the drone.

$$\begin{bmatrix} x_t \\ y_t \\ z_t \\ \dot{x_t} \\ \dot{y_t} \\ \dot{z_t} \\ \Phi_t \\ \Theta_t \\ \Psi_t \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & t & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & t & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & t & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & t & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x_{t-1} \\ y_{t-1} \\ z_{t-1} \\ \dot{x_{t-1}} \\ \dot{y_{t-1}} \\ \dot{z_{t-1}} \\ \Phi_{t-1} \\ \Theta_{t-1} \\ \Psi_{t-1} \end{bmatrix} + \begin{bmatrix} \frac{t^2}{2} & 0 & 0 & 0 & 0 & 0 \\ 0 & \frac{t^2}{2} & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{t^2}{2} & 0 & 0 & 0 \\ t & 0 & 0 & 0 & 0 & 0 \\ 0 & t & 0 & 0 & 0 & 0 \\ 0 & 0 & t & 0 & 0 & 0 \\ 0 & 0 & 0 & t & 0 & 0 \\ 0 & 0 & 0 & 0 & t & 0 \\ 0 & 0 & 0 & 0 & 0 & t \end{bmatrix} \cdot \begin{bmatrix} \ddot{x}_t \\ \ddot{y}_t \\ \ddot{z}_t \\ \dot{\Phi}_t \\ \dot{\Theta}_t \\ \dot{\Psi}_t \end{bmatrix} \quad (5.24)$$

Concerning the estimation of the feature's position, the relation expressed by (5.25) can be used. The deltas in these equations represent the displacement of the drone during iterations.

$$z = \begin{bmatrix} \mu_{x,t} \\ \mu_{y,t} \\ \mu_{z,t} \end{bmatrix} = \begin{bmatrix} \mu_{x,t-1} + \Delta x_t \\ \mu_{x,t-1} + \Delta y_t \\ \mu_{x,t-1} + \Delta z_t \end{bmatrix} \quad (5.25)$$

5.1.4.3 EKF filter

Each feature position (x, y, z) will be approximated by a Gaussian in the FastSLAM, therefore an EKF filter will be used for each one of the detected features. Here we have one important simplification considering the initialization of a brand-new feature: the initialization of the estimation parameters (mean and covariance) is made using the position reconstructed by two images.

According to (EADE, 2009), the distribution of landmark's positions is poorly estimated by Euclidean coordinates. Thus, we have the need to refine the estimation of the depth, made by the reconstruction of the two consecutive images, allowing a better representation of the feature.

One possible approach is presented by (DAVISON et al., 2007), the creation of a one dimension particle filter with particles scattered on the ray connecting the particle and the camera. When this estimate becomes acceptable (small covariance matrix) the feature, with the depth estimated, is added to the map.

But this approach is too heavy to be implemented in real-time. Then (EADE; DRUMMOND, 2006) and many others papers use the inverse depth, which can be estimated used the EKF since it is linear under certain conditions. Specially in (EADE;

DRUMMOND, 2006), the inverse depth in the frame of first observation of the feature is estimated.

Equation (5.26) represents the euclidean estimation given by the image reconstruction. The inverse depth is defined by (5.27). This notation is introduced in (MONTIEL; CIVERA; DAVISON, 2006).

$$x = (x \quad y \quad z)^t \tag{5.26}$$

$$x_{invdepth} = \left(\frac{x}{z} \quad \frac{y}{z} \quad \frac{1}{z}\right)^t = (u \quad v \quad q)^t \tag{5.27}$$

5.1.4.4 Sampling the proposal distribution

The proposal distribution used in the FastSLAM 2.0 is improved considering the FastSLAM 1.0 taking into account the measure z_t in the sampling of the posterior. The equation (5.28) represents the posterior in the integral form. This equation can be approximated by a Gaussian with the parameters given by (5.29), (5.30) and (5.31).

$$p(x_t|x_{1:t-1}^{[k]}, u_{1:t}, z_{1:t}, c_{1:t}) = \eta^{[k]} \int \overbrace{p(z_t|m_{c_t}, x_t, c_t)}^{\sim\mathcal{N}(z_t; g(m_{c_t}, x_t), R_t)} \overbrace{p(m_{c_t}|x_{1:t-1}^{[k]}, z_{1:t-1}, c_{1:t-1})}^{\sim\mathcal{N}(m_{c_t}; \mu_{c_t,t-1}^{[k]}, \Sigma_{c_t,t-1}^{[k]})} dm_{c_t} \overbrace{p(x_t|x_{t-1}^{[k]}, u_t)}^{\sim\mathcal{N}(x_t; h(x_{t-1}^{[k]}, u_t), R_t)} \tag{5.28}$$

$$Q_t^{[k]} = R_t + G_m \Sigma_{c_t,t-1}^{[k]} G_m^T \tag{5.29}$$

$$\Sigma_{x_t}^{[k]} = [G_x^T Q_t^{[k],-1} G_x + R_t^{-1}]^{-1} \tag{5.30}$$

$$\mu_{x_t}^{[k]} = \Sigma_{x_t}^{[k]} G_x^T Q_t^{[k],-1} (z_t - \hat{z}_t^{[k]}) + \hat{x}_t^{[k]} \tag{5.31}$$

To generate the normal distribution with the parameters expressed by the equations (5.30) and (5.31) the polar form of the Box-Muller sampler can be used. It is faster and more robust numerically in comparison with the basic form. The equations (5.32), (5.33), (5.34) shows how a normal random can be generated, with u and v being random uniform number in the interval $[-1, 1]$.

$$w = u^2 + v^2 \tag{5.32}$$

$$z_0 = u \cdot \sqrt{\frac{-2\ln(w)}{w}} \tag{5.33}$$

$$z_1 = v \cdot \sqrt{\frac{-2\ln(w)}{w}} \tag{5.34}$$

Then, with z_0 and z_1 we can sample the posterior distribution using the parameters defined in (5.30) and (5.31).

5.1.4.5 Data association method

The data association method proposed here is the nearest neighbor. This method calculates the Malahanobis distance between one landmark and each detected feature and compares this distance with the chi-squared estimate, considering the degree of freedom of the problem and the confidence interval chosen. The equation (5.35) shows how this distance is calculated, with the variable $Q_{t,j}$ given by (5.29).

$$D^2_{i,j} = \nu^t_{i,j} Q^{[k],-1}_{t,j} \nu_{i,j} < \chi^2_{d,\alpha} \tag{5.35}$$

Normally this method is divided in two steps: first the distance calculated is validated considering the chi-squared parameter with a certain level of confidence. This is the test step. The next one is the selection criterion, which in our case is the smallest distance calculated, $D^2_{i,j}$.

NEIRA; TARDÓS and MONTEMERLO; THRUN tell us that this method can cause divergence in the EKF case, due to errors in data association. Moreover, when the environment is noisy usually we have more than one feature close to a landmark, which can introduce errors.

A more suitable method is the JCBB (Joint Compatibility Branch and Bound), since it explores the joint compatibility that exists during the observation of a batch of features, or in our case, an image. Then this algorithm searches in the interpretation tree (a tree with all the combinations of associations) the combination of association between features and landmarks that are most likely to be true, but as one can assume demands a high computational capacity.

5.1.4.6 Resampling method and estimation of pose

Particle sampling is carried after one complete iteration of the particle filter. The most popular method is the stratified resampling: particles are divided in intervals and sampled from them based on the cumulative weight.

The sampling allows us to filter the particle set searching for particles that are close to reality (THRUN; BURGARD; FOX, 2005). But one must watch for the dominance inside this sampling, a particle cannot occupy the entire set, as diversity is important for the precision of the filter.

Following this step we need to estimate the position of landmarks and the robot using all the particles. This is done using the weights in a weighted mean with all the

positions of landmarks and the estimates of the robot's path in each particle.

5.2 Bundle Adjustment

Simply put, bundle adjustment is a large sparse geometric estimation problem and normally they are formulated as non-linear least-square problems (TRIGGS et al., 2000). Its main goal is to minimize the reprojection error between the observed and predicted points (LOURAKIS; ARGYROS et al., 2005).

Bundle adjustment was born in the context of visual reconstruction, when one tries to extract a 3D model from a set of images. In the 3D model construction the camera position can be recovered, therefore the position of the robot can be determined using this information (TRIGGS et al., 2000).

The problem faced by this method is tackled from the reconstruction perspective, that is, try to match the features from the image with a model. To reconstruct the 3D space using 2D images the features detected must be fitted in the model, which is usually done using a non-linear least square method, searching to minimize the reprojection error (WU et al., 2011).

Being x a vector of parameters and f the cost function, the following expression is minimized by the least square problem:

$$x^* = \arg\min_x \sum_{i=1}^{k} \|f_i(x)\|^2 \tag{5.36}$$

The most used method to solve the nonlinear least square estimation problem is the Levenberg-Marquardt method. It uses a trust-region approach to find a minimum, and when this minimum is found another trust region is build, this time around the point recently found, and the process start again. The LM method can be seen as a blend of the gradient descent method, (5.37), and the Gauss-Newton method, which performs a quadratic approximation.

$$x_{i+1} = x_i - \lambda \nabla f(x_i) \tag{5.37}$$

The gradient descent method presents many problems with the step size, causing convergence issues (ROWEIS, 1996). For this reason the Gauss-Newton method uses second-order information to speed-up convergence. The approximation in the equation (5.39) is applied in the equation (5.38) to obtain (5.40). This approximation is used due to the difficulty of calculate the Hessian matrix. Moreover, the first term of the equation

(5.39) usually dominates the second one (NOCEDAL; WRIGHT, 2006).

$$(x_{i+1} - x_i) \cdot \nabla^2 f(x_i) = -\nabla f(x_i) \tag{5.38}$$

$$\nabla^2 f(x) = J(x)^T J(x) + \sum_{j=1}^{m} f_j(x) \nabla^2 f_j(x) \approx J(x)^T J(x) \tag{5.39}$$

$$J(x)^T J(x)(x_{i+1} - x_i) = -J(x_i)^T \tag{5.40}$$

The equation 5.41 represents the Levenberg-Marquardt method, where the λ factor controls the composition between the Newton-Gauss and the gradient descent method. Bigger it is, the method behaves like the gradient descent, closer to the zero means that he is basically the Newton-Gauss method.

$$(J(x)^T J(x) + \lambda I) \cdot (x_{i+1} - x_i) = -J(x_i)^T \tag{5.41}$$

5.3 PTAM framework

PTAM stands for parallel tracking and mapping and its described in (KLEIN; MURRAY, 2007). Considering the probabilistic techniques, FastSLAM for example, this application has one vital difference, the complete separation between tracking and mapping. This allows a more precise update on the mapping and a more robust tracking.

Other implementations of SLAMs solutions, like the EKF-SLAM (SMITH; CHEESEMAN, 1986) or even the FastSLAM (MONTEMERLO; THRUN, 2007) itself are implemented for the use with robots, which implies precise control of the velocity and odometry feedback. They are not suitable for cellphone cameras, for example, since they don't have a trustworthy odometry feedback and can move much faster that robots. It is exactly for this use case that the PTAM was created (KLEIN; MURRAY, 2007).

The division between tracking and mapping have consequences in both processes. The tracking is free to use the map without having to wait for updates and any type of tracking can now be used. As for the mapping it do not need to be update at every frame, eliminating redundancies that exists in sequential images and giving the mapping process has more time to use a precise refinement and update method. In the figure 7 one can see the PTAM execution with the map points displayed in the camera image.

5.3.1 Tracking

Considering that a map was already been created (map initialization will be treated in the Mapping section), this operation can be divided in the following steps:

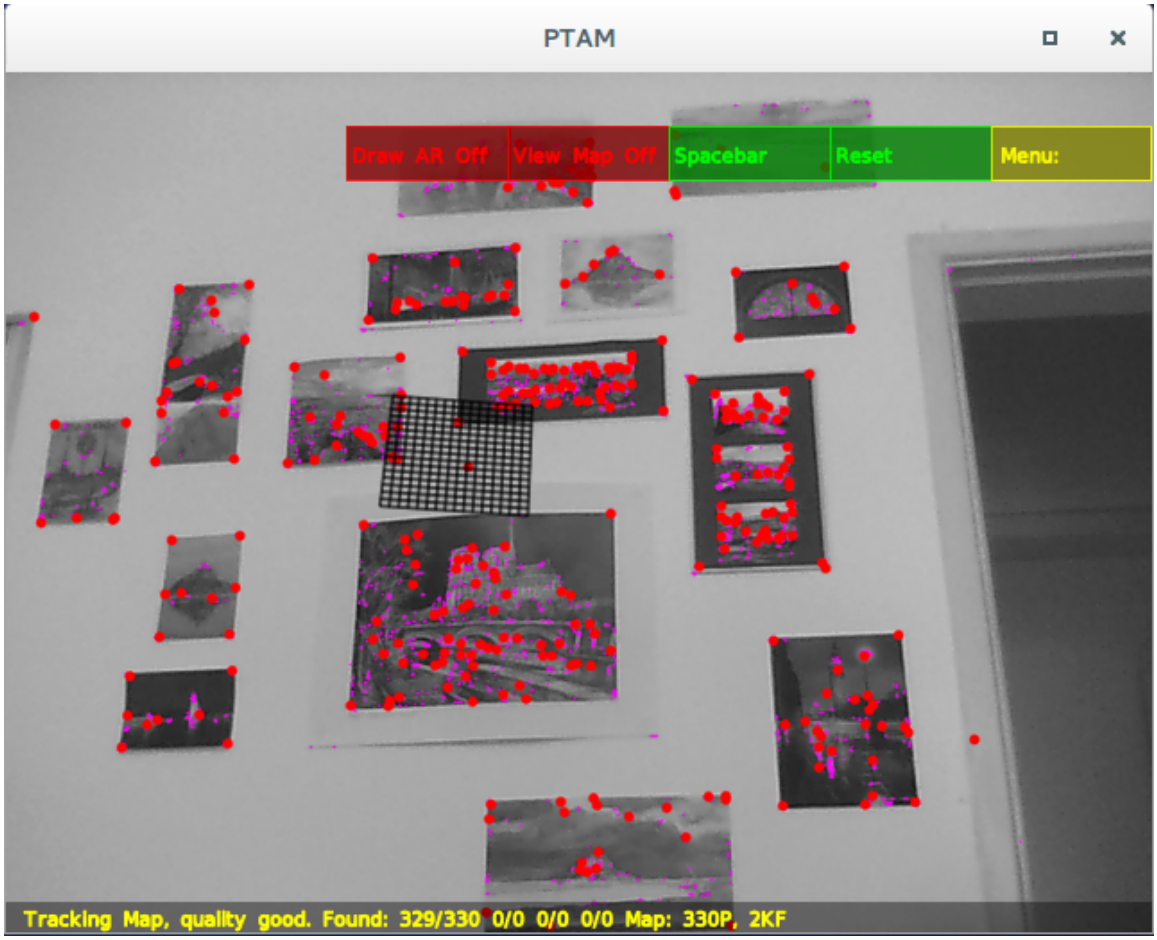


Figure 7 – PTAM window

1. A pose estimation is made with the new frame.
2. All the map points are projected in the image.
3. A few coarsest-scale features are searched in the image.
4. The camera pose estimate is updated using the matches.
5. Many features are again searched in the image.
6. A final pose is estimated.

5.3.1.1 Map projection points

To project the points of the map to the image the coordinate system should be changed and the camera model should be applied. The first is just a multiplication by a homogeneous 4x4 matrix, expressed by (5.42). Then the pin-hole model is applied to the 3D point. The radial distortion is accounted for in (5.43), using (5.44) and (5.45).

$$\boldsymbol{p}_{j\mathcal{C}} = E_{\mathcal{CW}}\boldsymbol{p}_{j\mathcal{W}} \tag{5.42}$$

$$CP(\boldsymbol{p}_{j\mathcal{C}}) = \begin{bmatrix} u_0 \\ v_0 \end{bmatrix} + \begin{bmatrix} f_u & 0 \\ 0 & f_v \end{bmatrix} \cdot \frac{r'}{r} \begin{bmatrix} \frac{x}{z} \\ \frac{y}{z} \end{bmatrix} \tag{5.43}$$

$$r = \sqrt{\frac{x^2 + y^2}{z^2}} \tag{5.44}$$

$$r' = \frac{1}{\omega} \arctan\left(2r \tan\frac{\omega}{2}\right) \tag{5.45}$$

5.3.1.2 Patch Search

The pixels patch stored in each map point is warped to account for viewpoint changes, by multiplication using the matrix 5.46. The point (u_s, v_s) corresponds to horizontal and vertical displacements in the patch's original level and the point (u_c, v_c) in the zeroth level of the current frame.

$$A = \begin{bmatrix} \frac{\delta u_c}{\delta u_s} & \frac{\delta u_c}{\delta v_s} \\ \frac{\delta v_c}{\delta u_s} & \frac{\delta v_c}{\delta v_s} \end{bmatrix} \tag{5.46}$$

The matrix A is calculated using three projections. The first projection made is the unit pixel displacement in the source keyframe into the patch's plane, then this is projected in the current frame. Matching is done evaluating the zero-mean SSD scored for each corner location. If the smallest score is lower than a threshold, the match is positive.

5.3.1.3 Pose Update

Using the set of matched observations the camera pose is calculated minimizing the error of a robust objective function of the reprojection error, equation (5.47), e_j being the reprojection error. We consider the Tukey bi-weight objective function.

$$\boldsymbol{\mu'} = \arg\min_{\boldsymbol{\mu}} \sum_{j \in S} \text{Obj}\left(\frac{|e_j|}{\sigma_j}, \sigma_T\right) \tag{5.47}$$

5.3.1.4 Repetition of the patch search and pose update

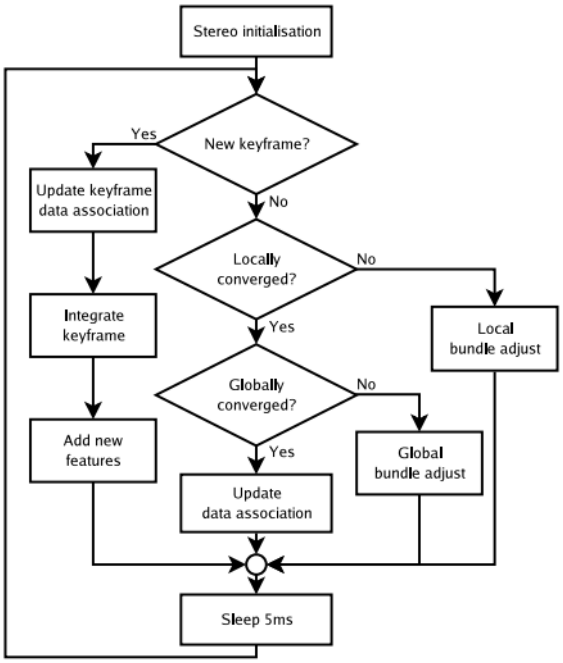
As said in the beginning of section 5.3.1 the patch search and pose update are done twice, so at the final of pose update a bigger set of patches from map points are searched in the image. After that, the final pose is estimated.

The quality of the tracking is evaluated at each frame, and if it is not acceptable, the system do not create more keyframes with low quality. If even so the tracking reference is considered lost, then the initialization procedure must be redone (it will be explained in the mapping section).

5.3.2 Mapping

The mapping procedure is divided in two steps, the initialization and the expansion. All the procedures are explained in the figure 8. The map initialization procedure consists only in the capture of two frames translated from 10 centimeters from one to another.

Figure 8 – Mapping thread, taken from (KLEIN; MURRAY, 2007)



5.3.2.1 Map expansion

The map is expanded, after the initialization, by the addition of keyframes. They are frames used to describe the environment and from them the points of the map are extracted. These keyframes have the following structure:

- A transformation matrix from the frame to the world coordinate system, $E_{\mathcal{C}|\mathcal{W}}$.
- A four level pyramid of greyscale images, dividing the frame dimensions by two at each level.

If three conditions are respected, a common frame is added to the map as a keyframe:

1. Tracking quality is good.
2. Time after the last keyframe capture is bigger than 20 frames.
3. The camera must be at a minimum distance from the closest frame.

Each point of the map, called here a feature, represents a distinct characteristic about the environment. They are represented by the following data structure:

- Coordinates $(x_{j\mathcal{W}}, y_{j\mathcal{W}}, z_{j\mathcal{W}}, 1)$ in the world coordinate system.
- An unit patch normal n_j
- Reference for the patch source pixels.

The process to add information from a keyframe to a map starts by applying the FAST corner detector to each pyramid level of the keyframe. Then non-maximal suppression and thresholding based on Shi-Tomasi score is then applied to detect the most salient points. Non-maximal suppression consists on an algorithm that eliminates points that do not lie in important edges.

From the remaining set the points close to an already detected point map are eliminated. Then the set is a candidate to be added to the map. One parameter remains to be found: the depth. For this a triangulation is necessary. The closest keyframe is chosen for the operation and the correspondences are made using epipolar search. If a match is detected then the triangulation is executed and the point is added to the map.

5.3.2.2 Bundle adjustment and data association refinement

Bundle adjustment refines the map using new measurements. The minimization of the equation 5.48 is done using the Levemberg-Marquardt bundle adjustment method. When the number of keyframes already saved is high and new keyframes are being added, the full bundle adjustment becomes too slow for real time execution. In this case only the local bundle adjustment is executed, just the closest points of the map are updated.

When no keyframes are added, the mapping thread refines the map. To find the depth of a candidate point it is only used two frames, but these points may be present in other keyframes as well. If the measurement is successful, it is also added to the map. Outliers in the form of measurements are also detected, using the Tukey estimator.

$$\{(\boldsymbol{\mu}_1 \cdots \boldsymbol{\mu}_N), (\boldsymbol{p'}_1 \cdots \boldsymbol{p'}_M)\} = \arg\min_{\{\boldsymbol{\mu},\boldsymbol{p}\}} \sum_{i=1}^{N} \sum_{j \in S_i} \text{Obj}\left(\frac{|e_j|}{\sigma_j}, \sigma_T\right) \tag{5.48}$$

5.4 Our choice of algorithm

Numerous factors must be considered to make a correct choice. Considering the complexity of both methods the particle filter has an updating cost of $O(M \log(N))$, M being the number of particles necessaries for convergence and N the number of landmarks or features (ROLLER et al., 2003) but the bundle adjustment has normally a quadratic cost

(TRIGGS et al., 2000). Although, with the help of new methods to reduce the complexity of the BA, particularly in the SLAM case, linear complexity can be achieved (STRASDAT; MONTIEL; DAVISON, 2012).

Due to the use of the Marquardt-Levenberg non-linear estimator the BA is more precise than filtering, in comparison with the EKF implementation (STRASDAT; MONTIEL; DAVISON, 2010). The principal difficulty encountered concerning the comparison and the decision process is the lack of a common benchmark to test all the SLAM solutions. Scientific article results usually are not compatible nor comparable, thus we cannot affirm that one method is better than the other. This situation gets more complex if we consider that the simplifications for the reduction of the BA complexity cited in the previous paragraph can affect the precision.

The articles (MONTEMERLO et al., 2002) and (ROLLER et al., 2003) introduce FastSLAM, which is an implementation of the particle filter for the SLAM problem. In this work one can see the performance of the particle filter to estimate the robot's path, even if one particle is used, we have a good estimation (1 to 2 meters of RMS (Root Mean Square) error in a trajectory that have order of kilometers). And the size of the dataset used shows us that this method does not have problems with the use of many features as the EKF has.

Another great advantage of this method is the possibility of use a simple likelihood method for the data association, instead of other complex methods, like the nearest neighbour (NN) or the joint compatibility brand and bound (JCBB). The article (DAVISON et al., 2007) shows an implementation of the EKF for SLAM, but presents more details about map generation that the FastSLAM articles.

One solution that uses the bundle adjustment that can be pointed out is the PTAM framework (KLEIN; MURRAY, 2007). One of the biggest advantages of the bundle adjustment techniques is the possibility to separate the tracking step from the mapping step, which is exploited in this framework. But before the nonlinear least-square estimation method, one needs to execute data association, normally using the NN or the JCBB method. A parallel structure for these methods are not simple to be obtained as for particle filter, which poses another problem for our implementation.

Considering that we want to implement the SLAM solution with GPU, we need a parallel form of the solution, which the particle filter can offer. We also have just three months to implement it, an interval of time too short to implement an high complex algorithm like the FastSLAM, therefore the adoption of the bundle adjustment, represented by the PTAM framework, makes more sense. Thinking about the performance and precision obtained by this framework, we can affirm that is better than any other SLAM solution presented until this day (not considering DenseSLAM algorithms) (KLEIN; MURRAY, 2007).

6 Implementation

6.1 TUM AR.Drone ROS package

The ROS application created by the Technische Universität München for the AR.Drone uses the frontal monocular camera and the odometry sensors to execute off-board SLAM, communication is carried through Wi-Fi to communicate with computer. This program estimates the scale of the SLAM system, then uses the position calculated by the PTAM framework to feed an EKF filter, taking into account the position of the drone. The map management is done exclusively by PTAM.

The control of the drone is also calculated using the parameters estimated by the EKF using a PID controller. The main features of this program are the scale estimation, the sensor information fusion and the controller implementation. In the figure 9 the interface of the package, with the PTAM screen at the right side below the map.

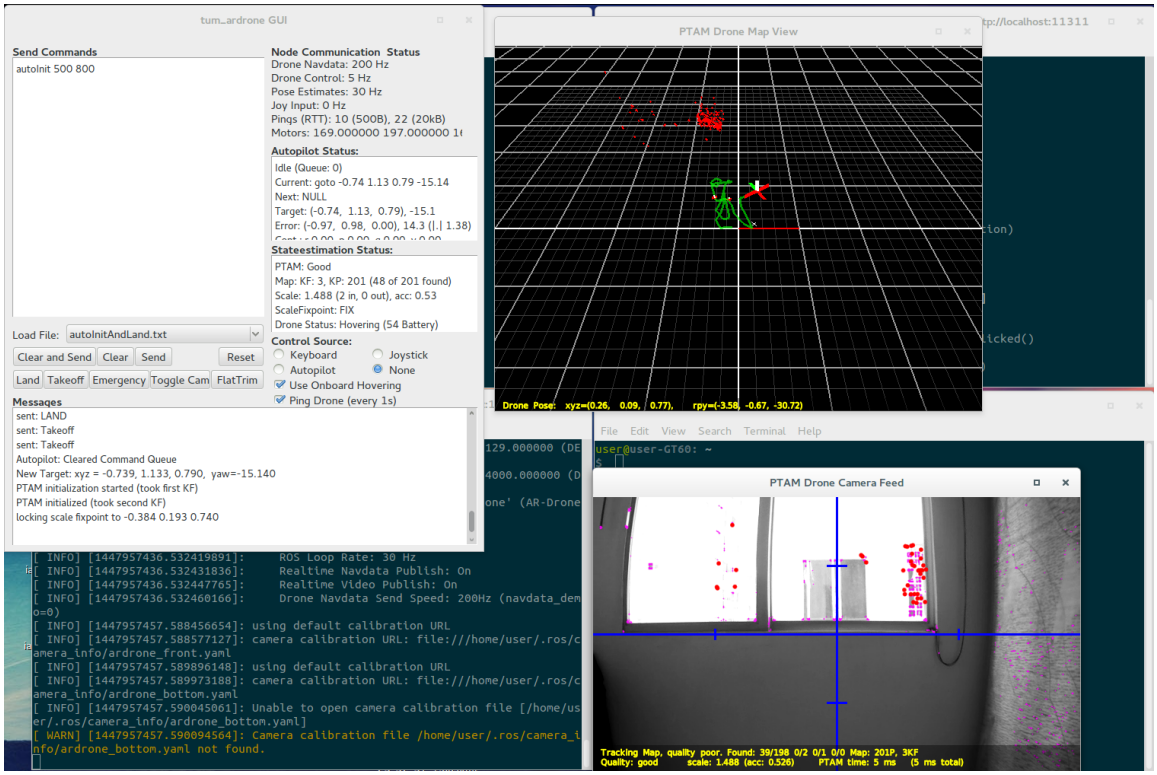


Figure 9 – Tum package

6.1.1 Scale information

At each measurement interval (which is different for the camera and the odometry sensors, but here we will assume that they are equal for the sake of simplicity) the drone

sends for the PC a sample pair (x_i, y_i), x_i being scaled with the visual map (PTAM) and y_i in metric units. Since both are related with the motion of the drone, we can affirm that $x_i \approx \lambda y_i$.

Assuming that these measurement samples are isotropic and with Gaussian white noise, they can be represented by equations (5.48) and (6.1).

$$x_i \sim \mathcal{N}(\lambda\mu_{\rangle}, \sigma^{\in}_{\S}) \tag{6.1}$$

$$y_i \sim \mathcal{N}(\lambda\mu_{\rangle}, \sigma^{\in}_{\dagger}) \tag{6.2}$$

Then a maximum likelihood estimation is used to estimate the mean of the distribution and the scale. Equation (6.3) is the expression of the negative log-likelihood, and by first minimizing over the means and after over λ a global minimum is achieved.

$$\mathcal{L}(\mu_1, \cdots, \mu_n, \lambda) \propto \frac{1}{2} \sum_{n}^{i=1} \left(\frac{\|x_i - \lambda\mu_i\|^2}{\sigma_x^2} + \frac{\|y_i - \mu_i\|^2}{\sigma_y^2} \right) \tag{6.3}$$

Concerning the generation of the samples, the sensors have different sampling frequencies, therefore a compensation mechanism must be implemented to allow the use of compatible sample value. Since the altimeter from the drone is the most precise sensor, the altitude is the measurement used for the scale estimation. The metric sensor is faster so at each visual measure a mean of all the metric values not used is made. Then the distance traveled within some timespan is calculated. This value, in meters and in the video scale, is used for the scale estimation.

6.1.2 Sensor information fusion

The EKF is used to update the estimated pose with all the data collected, from the PTAM and from the sensors. The fusion of all this information occurs in the definition of the prediction and the observation model. Concerning the prediction model, the following

model is used:

$$\begin{bmatrix} x_{t+1} \\ y_{t+1} \\ z_{t+1} \\ \dot{x}_{t+1} \\ \dot{y}_{t+1} \\ \dot{z}_{t+1} \\ \Phi_{t+1} \\ \Theta_{t+1} \\ \Psi_{t+1} \\ \dot{\Psi}_{t+1} \end{bmatrix} \leftarrow \begin{bmatrix} x_t \\ y_t \\ z_t \\ \dot{x}_t \\ \dot{y}_t \\ \dot{z}_t \\ \Phi_t \\ \Theta_t \\ \Psi_t \\ \dot{\Psi}_t \end{bmatrix} + \delta t \cdot \begin{bmatrix} \dot{x}_t \\ \dot{y}_t \\ \dot{z}_t \\ c_1 R(\Phi_t, \Theta_t, \Psi_t)_{1,3} - c_2 \hat{x}_t \\ c_1 R(\Phi_t, \Theta_t, \Psi_t)_{2,3} - c_2 \hat{x}_t \\ c_7 \bar{\dot{z}}_t - c_8 \dot{z}_t \\ c_3 \bar{\Phi}_t - c_4 \Phi_t \\ c_3 \bar{\Theta}_t - c_4 \Theta_t \\ \dot{\Psi}_t \\ c_5 \bar{\dot{\Psi}}_t - c_6 \dot{\Psi}_t \end{bmatrix} \tag{6.4}$$

c_i are constants calibrated before execution, the parameters with a bar are the control effort sent to the drone in the last iteration and the parameters denoted by $R(\Phi_t, \Theta_t, \Psi_t)_{i,3}$ are the elements $i, 3$ from the rotation matrix defined by roll, pitch and yaw.

Then we have the observation model for the odometry and the PTAM. The first are represented by equations (6.5) and (6.6). Since the drone measures the horizontal velocity in its own referential, the yaw must be taken into account in the measurement function. The pitch and roll are given by the direct lecture from the sensor and the yaw and vertical velocity are calculated from the yaw and height directly to account for uneven ground and yaw-drift.

$$h_o(x_t) := \left[\dot{x}_t \cos\Psi_t - \dot{y}_t \sin\Psi_t \dot{x}_t \sin\Psi_t + \dot{y}_t \cos\Psi_t \dot{z}_t \Phi_t \Theta_t \dot{\Psi}_t\right] \tag{6.5}$$

$$z_{o,t} := \left(\hat{v}_{x,t}, \hat{v}_{y,t}, \frac{\hat{h}_t - \hat{h}_{t-1}}{\delta_{t-1}}, \hat{\Phi}_t, \hat{\Theta}_t, \frac{\hat{\Psi}_t - \hat{\Psi}_{t-1}}{\delta_{t-1}}\right)^T \tag{6.6}$$

The scaling factor λ^* is used with the PTAM's pose to transform to the coordinate system of the drone, as showed by equation (6.7). The matrix $E_{\mathcal{C},t}$ refers to the camera pose in SE(3) and $E_{\mathcal{DC}}$ the transformation from the camera referential to the drone's. f is the conversion from SE(3) to the roll-pitch-yaw representation.

$$h_P(x_t) := (x_t, y_t, z_t, \Phi_t, \Theta_t, \Psi_t)^T \tag{6.7}$$

$$z_{P,t} := f\left(E_{\mathcal{DC}} E_{\mathcal{C},t}\right) \tag{6.8}$$

The EKF predicts the pose of the drone also considering the delays involved in the Wi-Fi communication and the time spend by control signal calculation.

6.1.3 Control

The control implemented for the drone will be quickly shown here because it is outside the scope of this work. With the actual position $(\hat{x}, \hat{y}, \hat{z}, \hat{\Psi})^T$ and the predicted point $x_{t+\Delta t_{control}}$, the controller is implemented by the equation (6.9):

$$\begin{bmatrix} \bar{\Phi} \\ \bar{\Theta} \\ \bar{\dot{z}} \\ \bar{\dot{\Psi}} \end{bmatrix} = \begin{bmatrix} R(\Psi) * (0.5(\hat{(}x) - x) + 0.32\dot{x}) \\ R(\Psi) * (0.5(\hat{(}y) - y) + 0.32\dot{y}) \\ 0.6(\hat{z} - z) + 0.2\dot{z} + 0.01 \int(\hat{z} - z) \\ 0.02(\hat{\Psi} - \Psi) \end{bmatrix} \tag{6.9}$$

6.2 Analysis of the TUM package

The objective of this work is to apply parallelism to the SLAM problem. Studying what the TUM package does we can easily see that there isn't much room for parallelism, since the scale estimation, EKF and the control calculation have a sequential structure. That does not mean it is impossible to optimize the code with GPU processing, it just creates a doubt about the level of improvement.

Normally for a code to be accelerated with a GPU it should have big independent loops with rather simple operations or many independent structures to present a significant gain in performance and to compensate the overhead caused by the memory allocation in the GPU and the copy of all the necessary data. This is why the GPU acceleration is nowadays more popular in operations with images, for example. Feature detectors that operates in GPU present are many times faster than ones using CPU, and this can allow real-time execution of computational expensive algorithms like SURF.

Inside the PTAM framework there is many structures that can be parallelized. The feature extraction is an obvious example, but we also have certain operations in the tracking and mapping. Execute the mapping and the tracking thread in the GPU, although, is not a good idea, since the GPU has a lower clock than the CPU and therefore takes more time to deal with complex instructions. In the GPU the processing power is sacrificed in the expense of higher parallelism.

At the start of the keyframe preparation the PTAM uses the FAST corner detection to extract the features from the current frame. This is done four times for each frame, since a 4-level pyramid must be created, and the CVD library is used, where the FAST is implemented in CPU. We changed this function and made necessary adaptations on the code to use the OpenCV implementation in GPU of the same algorithm.

Another area from the PTAM that can be parallelized is the search for patches in the tracking operation. After the calculation of a prior pose and the projection of the map points in the camera centered referential, the map-points are selected to be searched in

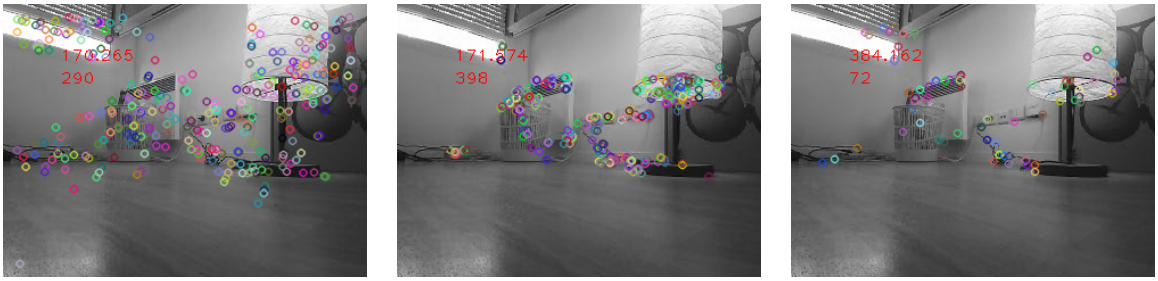
the image. This is another operation that can be parallelized and probably will present a huge gain in performance, since in one step of the tracking the search is done twice, one time with a smaller number of points (in the order of 50) and after with more points (in the order of 1000).

It is important say that functions from the library CVD uses SIMD (Single instruction, multiple data) parallelism in the CPU and tries to optimize as much as possible the CPU code (using the SSE3 instruction set, for example). This can increase the velocity of calculation up to a level superior to the GPU, since the copy overhead delay is significant. Maybe for some GPUs models the FAST implemented in the CPU can be faster. But in the case of the patch search it is almost certain that the GPU will be faster due to the dimension of the loop (order of 1000). We will not enter in the discussion of the differences of implementation of the FAST corner detection in CVD and OpenCV because it goes beyond the scope of this work.

7 Test and results

7.1 Comparison between feature detection algorithms

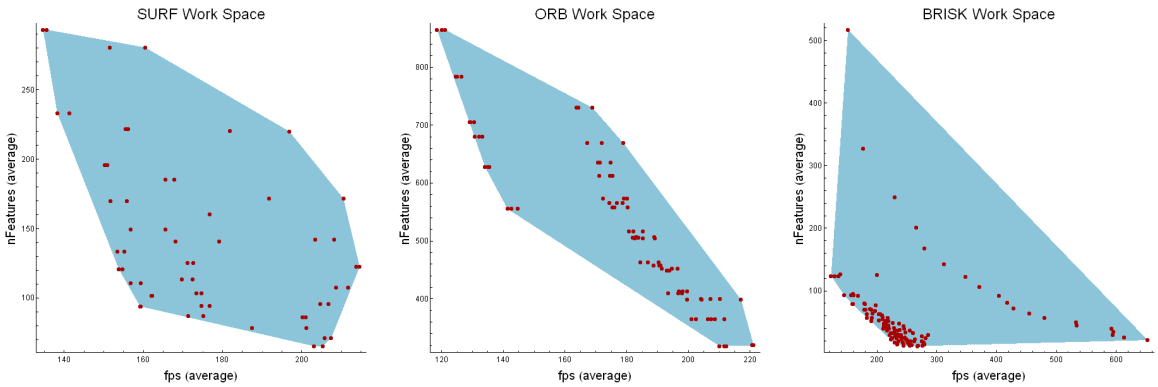
During preliminary stages of the project we wanted to test different feature detection algorithm, namely: SURF, ORB and BRISK, to better assert their usefulness and correspondence to our needs. Examples of the kind of features that each detects are show in Figure 10. This test was carried on an AR.Drone 1.0 using a 65 seconds obtained from the drone's frontal camera (each frame has 320x240 pixels).



(a) SURF (b) ORB (c) BRISK

Figure 10 – Feature detection examples for each tested algorithm.

Moreover, we varied each algorithm's parameters to verify how them influences the number of detected features and execution time. Results are show in Figure 11 and were obtained using the notebook described in next section.



(a) SURF (b) ORB (c) BRISK

Figure 11 – Frames per second versus the number of features for each algorithms.

7.2 FAST Benchmark

We benchmarked the mean execution time for the FAST algorithm implemented both in CPU and GPU. A sequence of video frames captured by the drone were used in

region rich on features. Moreover, we run the benchmark on two different configurations:

1. A notebook using:
   - Intel® Core™ i7-4700MQ CPU @ 2.40GHz
   - Nvidia GeForce GTX 770M
2. A desktop using:
   - Intel® Core™ i7-980X CPU @ 3.33GHz
   - Nvidia GeForce GTX 980

   The following results were obtained:

		CPU	GPU
Level 0	Frequency	1258.3 Hz	578.9 Hz
	N° of features	3386.6	2884.8
Level 1	Frequency	3681.9 Hz	1556.7 Hz
	N° of features	1300.8	1292.0
Level 2	Frequency	8652.3 Hz	3895.2 Hz
	N° of features	648.4	649.2
Level 3	Frequency	17373.6 Hz	5704.6 Hz
	N° of features	386.1	175.1

Table 1 – Results using the desktop

		CPU	GPU
Level 0	Frequency	908.6 Hz	542.5 Hz
	N° of features	3384.6	1786.2
Level 1	Frequency	2619.9 Hz	1057.5 Hz
	N° of features	1299.9	899.6
Level 2	Frequency	6149.4 Hz	2508.0 Hz
	N° of features	647.9	600.7
Level 3	Frequency	11678.0 Hz	3982.8 Hz
	N° of features	385.9	167.9

Table 2 – Results using the notebook

8 Analysis

8.1 Comparison between feature detection algorithms

With this test we wanted to show that feature detection could be done in real-time, that we processed frames faster than the drone send them. As video is received ate $20Hz$, it's clear that all algorithms respected the real-time constrain.

It's important to note that only SURF and ORB were implemented at GPU, while BRISK only had a CPU implantation at the time of tests. The reason for testing BRISK was to assert if all the good results that it showed on literature translated to our use-case and only then expend development time on producing a GPU-based BRISK implementation. And we can see that it is generally the fastest of all tested algorithms, but unfortunately it comes with a cost: fewer features.

Therefore, given BRISK's not so expressive gains in processing time in the workspace where it has a compatible number of detected features with the others algorithms we choose not to port it to GPU.

Finally, after some following project decision we finished by using a GPU-based implementation of the FAST feature detection algorithm that will be discussed in the following section.

8.2 PTAM at GPU tests

Before discussing over the results, it is worth noting that at each level the dimension of the image is divided by two. Starting with a 640x480 image, the dimensions at the following levels are: 320x240, 160x120 and 80x60.

The term *time* refers to the mean time necessary to process a frame. In all times we can see that the parallelization does not compensate the overhead caused by the data transfer from the CPU to the GPU and vice versa, this is more expressive in the small images. In the desktop case the time to process one frame in the level 4 is almost three times slower in the GPU, but in the level 0 it is approximately two times slower. Probably if we were dealing with high-definition images the GPU execution would be faster. Another point that need to be raised is that the FAST detector was created to be lightweight (ROSTEN; DRUMMOND, 2006) and the CVD library optimizes as much as possible the CPU code (it uses SSE2 instruction set).

In the case of the desktop the almost same quantity of features are observed in

both CPU and GPU, since we adjust the threshold value of the FAST (GPU) detector adding an offset of 5. This value is used to classify if a pixel is in a corner or not, which takes in consideration the intensity of the gray-scale image. We the same threshold from the notebook test but a slightly higher offset of 10, and the difference between the quantity of features detected in that case is significant. This can be caused by differences between implementations in the CVD library and the OpenCV or due to some imprecision in the hardware.

It won't be the first time that an algorithm in CPU gives a different result that o the same implemented in GPU. The SURF feature extraction algorithm is also implemented in CPU and GPU inside OpenCV and it is well know that some differences can be observed depending which implementation is used.

9 Conclusion

During this project we had to use many subjects and techniques that aren't seen or taught in our graduation course. Particle filters, nonlinear optimization, heterogeneous programming are only a few examples of the topics that we had to study to complete this project. An extensive research was made about the subjects in question.

In the search for a SLAM solution that is precise and lightweight at the same time, two alternatives standed-out: bundle adjustment or particle filter. Both has advantages and inconveniences, but considering the period that we had to implement this project we chose the BA as solution, even if the particle filter structure has a more parallel structure than the latter. However, the BA method shown more recently a more precise solution splitting the tracking and mapping in two separate operations.

To implement the parallelization we sent some operations that are normally executed in the CPU by the PTAM framework to the GPU. The feature extraction was entirely changed for a GPU-based implementation. Aiming to test this configuration with real use-case, we used a drone. In this context the ROS, Robotic Operation System, a framework that allows the implementation of interfaces between nodes, in our case the PC and the drone, was used, together with a package implemented by the Technische Universität München specifically for the AR.Drone that uses the PTAM to generate and manage the map.

The results of our test shown us that the overhead caused by the memory allocation in the GPU is significant, therefore this instrument should be used only when we have HD images, that has much more pixels than our image. Considering this result it can be seen too the level of optimization and speed of the PTAM framework only with CPU use.

Bibliography

ALAHI, A.; ORTIZ, R.; VANDERGHEYNST, P. Freak: Fast retina keypoint. In: IEEE. *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*. [S.l.], 2012. p. 510–517. Citado na página 19.

AMAZON. *Amazon Dash*. 2015. Disponível em: <https://fresh.amazon.com/dash/>. Citado na página 25.

AMAZON. *Amazon Prime Air*. 2015. Disponível em: <http://www.amazon.com/b?node=8037720011>. Citado na página 25.

ARULAMPALAM, M. S. et al. A tutorial on particle filters for online nonlinear/non-gaussian bayesian tracking. *Signal Processing, IEEE Transactions on*, IEEE, v. 50, n. 2, p. 174–188, 2002. Citado na página 38.

AZARBAYEJANI, A.; PENTLAND, A. P. Recursive estimation of motion, structure, and focal length. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, IEEE, v. 17, n. 6, p. 562–575, 1995. Citado na página 35.

BAILEY, T.; DURRANT-WHYTE, H. Simultaneous localization and mapping (slam): Part ii. *IEEE Robotics & Automation Magazine*, v. 13, n. 3, p. 108–117, 2006. Citado na página 19.

BAY, H. et al. Speeded-up robust features (surf). *Computer vision and image understanding*, Elsevier, v. 110, n. 3, p. 346–359, 2008. Citado 2 vezes nas páginas 20 and 33.

BEHLEY, J.; STEINHAGE, V.; CREMERS, A. B. Performance of histogram descriptors for the classification of 3d laser range data in urban environments. In: IEEE. *Robotics and Automation (ICRA), 2012 IEEE International Conference on*. [S.l.], 2012. p. 4391–4398. Citado na página 33.

BURGARD, W. et al. Estimating the absolute position of a mobile robot using position probabilitygrids. In: *Proceedings of the National Conference on Artificial Intelligence*. [S.l.: s.n.], 1996. Citado na página 22.

CALONDER, M. et al. Brief: Binary robust independent elementary features. In: *Computer Vision–ECCV 2010*. [S.l.]: Springer, 2010. p. 778–792. Citado 2 vezes nas páginas 20 and 33.

DAVEY, S. J. Simultaneous localization and map building using the probabilistic multi-hypothesis tracker. *Robotics, IEEE Transactions on*, IEEE, v. 23, n. 2, p. 271–280, 2007. Citado na página 21.

DAVISON, A. J. Real-time simultaneous localisation and mapping with a single camera. In: IEEE. *Computer Vision, 2003. Proceedings. Ninth IEEE International Conference on*. [S.l.], 2003. p. 1403–1410. Citado na página 35.

DAVISON, A. J. et al. Monoslam: Real-time single camera slam. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, IEEE, v. 29, n. 6, p. 1052–1067, 2007. Citado 2 vezes nas páginas 42 and 51.

DELLAERT, F. et al. Monte carlo localization for mobile robots. In: IEEE. *Robotics and Automation, 1999. Proceedings. 1999 IEEE International Conference on*. [S.l.], 1999. v. 2, p. 1322–1328. Citado na página 22.

DOUCET, A. et al. Rao-blackwellised particle filtering for dynamic bayesian networks. In: MORGAN KAUFMANN PUBLISHERS INC. *Proceedings of the Sixteenth conference on Uncertainty in artificial intelligence*. [S.l.], 2000. p. 176–183. Citado 2 vezes nas páginas 38 and 39.

DOUCET, A.; JOHANSEN, A. M. A tutorial on particle filtering and smoothing: Fifteen years later. *Handbook of Nonlinear Filtering*, Oxford, UK: Oxford University Press, v. 12, p. 656–704, 2009. Citado na página 36.

DURRANT-WHYTE, H.; BAILEY, T. Simultaneous localization and mapping: part i. *Robotics & Automation Magazine, IEEE*, IEEE, v. 13, n. 2, p. 99–110, 2006. Citado 2 vezes nas páginas 18 and 21.

EADE, E. *Monocular simultaneous localisation and mapping*. Tese (Doutorado) — University of Cambridge, 2009. Citado na página 42.

EADE, E.; DRUMMOND, T. Scalable monocular slam. In: IEEE. *Computer Vision and Pattern Recognition, 2006 IEEE Computer Society Conference on*. [S.l.], 2006. v. 1, p. 469–476. Citado 2 vezes nas páginas 42 and 43.

ELFES, A. Using occupancy grids for mobile robot perception and navigation. *Computer*, IEEE, v. 22, n. 6, p. 46–57, 1989. Citado na página 19.

ENGEL, J.; STURM, J.; CREMERS, D. Scale-aware navigation of a low-cost quadrocopter with a monocular camera. *Robotics and Autonomous Systems*, Elsevier, v. 62, n. 11, p. 1646–1656, 2014. Citado na página 23.

FEDER, H. J. S. *Simultaneous Stochastic Mapping and Localization*. Tese (Doutorado) — Massachusetts Institute of Technology, 1999. Citado na página 18.

GRAUMAN, K.; LEIBE, B. *Visual object recognition*. [S.l.]: Morgan & Claypool Publishers, 2010. Citado na página 33.

HOFMANN-WELLENHOF, B.; LEGAT, K.; WIESER, M. *Navigation: principles of positioning and guidance*. [S.l.]: Springer Science & Business Media, 2011. Citado na página 17.

JENSFELT, P.; CHRISTENSEN, H. I.; ZUNINO, G. Integrated systems for mapping and localization. In: *ICRA-02 SLAM Workshop. IEEE*. [S.l.: s.n.], 2002. Citado na página 19.

JULIER, S. J.; UHLMANN, J. K. New extension of the kalman filter to nonlinear systems. In: INTERNATIONAL SOCIETY FOR OPTICS AND PHOTONICS. *AeroSense'97*. [S.l.], 1997. p. 182–193. Citado na página 22.

JULIER, S. J.; UHLMANN, J. K. A counter example to the theory of simultaneous localization and map building. In: IEEE. *Robotics and Automation, 2001. Proceedings 2001 ICRA. IEEE International Conference on*. [S.l.], 2001. v. 4, p. 4238–4243. Citado na página 35.

KLEIN, G.; MURRAY, D. Parallel tracking and mapping for small ar workspaces. In: IEEE. *Mixed and Augmented Reality, 2007. ISMAR 2007. 6th IEEE and ACM International Symposium on*. [S.l.], 2007. p. 225–234. Citado 6 vezes nas páginas 11, 20, 35, 46, 49, and 51.

LEUTENEGGER, S.; CHLI, M.; SIEGWART, R. Y. Brisk: Binary robust invariant scalable keypoints. In: IEEE. *Computer Vision (ICCV), 2011 IEEE International Conference on*. [S.l.], 2011. p. 2548–2555. Citado 2 vezes nas páginas 19 and 20.

LOURAKIS, M. I.; ARGYROS, A. et al. Is levenberg-marquardt the most efficient optimization algorithm for implementing bundle adjustment? In: IEEE. *Computer Vision, 2005. ICCV 2005. Tenth IEEE International Conference on*. [S.l.], 2005. v. 2, p. 1526–1531. Citado na página 45.

LOWE, D. G. Object recognition from local scale-invariant features. In: IEEE. *Computer vision, 1999. The proceedings of the seventh IEEE international conference on*. [S.l.], 1999. v. 2, p. 1150–1157. Citado na página 33.

LOWE, D. G. Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, Springer, v. 60, n. 2, p. 91–110, 2004. Citado na página 20.

MONTEMERLO, M.; THRUN, S. *FastSLAM: A scalable method for the simultaneous localization and mapping problem in robotics*. [S.l.]: Springer, 2007. v. 27. Citado 2 vezes nas páginas 44 and 46.

MONTEMERLO, M. et al. Fastslam: A factored solution to the simultaneous localization and mapping problem. In: *AAAI/IAAI*. [S.l.: s.n.], 2002. p. 593–598. Citado na página 51.

MONTIEL, J.; CIVERA, J.; DAVISON, A. J. Unified inverse depth parametrization for monocular slam. *analysis*, v. 9, p. 1, 2006. Citado na página 43.

MUR-ARTAL RAÚL, M. J. M. M.; TARDÓS, J. D. ORB-SLAM: a versatile and accurate monocular SLAM system. *IEEE Transactions on Robotics*, v. 31, n. 5, p. 1147–1163, 2015. Citado 3 vezes nas páginas 20, 26, and 33.

NEIRA, J.; TARDÓS, J. D. Data association in stochastic mapping using the joint compatibility test. *Robotics and Automation, IEEE Transactions on*, IEEE, v. 17, n. 6, p. 890–897, 2001. Citado 3 vezes nas páginas 20, 21, and 44.

NEUBECK, A.; GOOL, L. V. Efficient non-maximum suppression. In: IEEE. *Pattern Recognition, 2006. ICPR 2006. 18th International Conference on*. [S.l.], 2006. v. 3, p. 850–855. Citado na página 34.

NOCEDAL, J.; WRIGHT, S. *Numerical optimization*. [S.l.]: Springer Science & Business Media, 2006. Citado na página 46.

NVIDIA. *NVIDIA® Tegra® X1*. [S.l.], 2015. Citado na página 26.

ROLLER, D. et al. Fastslam 2.0: an improved particle filtering algorithm for simultaneous localization and mapping that provably converges. In: *Proceedings of the International Joint Conference on Artificial Intelligence*. [S.l.: s.n.], 2003. Citado 2 vezes nas páginas 50 and 51.

ROSTEN, E.; DRUMMOND, T. Machine learning for high-speed corner detection. In: *Computer Vision–ECCV 2006*. [S.l.]: Springer, 2006. p. 430–443. Citado 4 vezes nas páginas 20, 33, 34, and 61.

ROWEIS, S. Levenberg-marquardt optimization. *Notes, University Of Toronto*, 1996. Citado na página 45.

SFU, A. L. at. *Ardrone autonomy a ROS driver for Parrot AR-Drone 1.0 and 2.0 quadrocopters*. 2015. Disponível em: <http://github.com/AutonomyLab/ardrone_autonomy>. Citado na página 41.

SMITH, A. et al. *Sequential Monte Carlo methods in practice*. [S.l.]: Springer Science & Business Media, 2013. Citado na página 36.

SMITH, R. C.; CHEESEMAN, P. On the representation and estimation of spatial uncertainty. *The international journal of Robotics Research*, Sage Publications, v. 5, n. 4, p. 56–68, 1986. Citado na página 46.

STRASDAT, H.; MONTIEL, J.; DAVISON, A. J. Real-time monocular slam: Why filter? In: IEEE. *Robotics and Automation (ICRA), 2010 IEEE International Conference on*. [S.l.], 2010. p. 2657–2664. Citado na página 51.

STRASDAT, H.; MONTIEL, J. M.; DAVISON, A. J. Visual slam: why filter? *Image and Vision Computing*, Elsevier, v. 30, n. 2, p. 65–77, 2012. Citado 4 vezes nas páginas 23, 35, 36, and 51.

THRUN, S.; BURGARD, W.; FOX, D. *Probabilistic Robotics*. MIT Press, 2005. ISBN 9780262201629. Disponível em: <http://books.google.fr/books?id=2Zn6AQAAQBAJ>. Citado 3 vezes nas páginas 22, 40, and 44.

TRIGGS, B. et al. Bundle adjustment—a modern synthesis. In: *Vision algorithms: theory and practice*. [S.l.]: Springer, 2000. p. 298–372. Citado 2 vezes nas páginas 45 and 51.

WAN, E. A.; MERWE, R. V. D. The unscented kalman filter. *Kalman filtering and neural networks*, New York: Wiley, p. 221–280, 2001. Citado na página 22.

WU, C. et al. Multicore bundle adjustment. In: IEEE. *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*. [S.l.], 2011. p. 3057–3064. Citado na página 45.