DEPARTAMENTO DE INGENIERÍA DE SISTEMAS Y AUTOMÁTICA

UNIVERSIDAD CARLOS III DE MADRID

ESCUELA POLITÉCNICA SUPERIOR

PROYECTO FIN DE CARRERA

RGB-D SLAM

Author: Jorge García Bueno

Tutor: Dr. Luis Moreno Lorente

LEGANÉS, MADRID

OCTOBER 2011

"Stay hungry, stay foolish"

"Se ambicioso, se inquieto"

Steve **Jobs**, 1955-2011.

Dedicated to Rosalía, Jorge, Luis and Raquel,

who understand my craziness ...

... and also to Alex,

boosting it everyday

Contents

List of Figures

List of Tables

Abstract

UNIVERSIDAD CARLOS III DE MADRID

Department of Systems and Automation

by Jorge García Bueno

This project has been developed as an implementation of a SLAM technique called GraphSLAM. This technique applies the theory of graphs to create an on-line optimization system that allows robots to map the scenario and locate themselves using a Time of Flight camera as the input source. To do that, a RGB-D system has been calibrated and used to create color 3D point clouds. With this information, the feature detector module estimates, as a first approximation, the pose of the camera. Therefore, a ICP pose refinement completes the graph structure. Finally, a HogMAN graph optimizer close the loop on each iteration using a hierarchical manifold optimization. As a result, 3D color maps are created containing, at the same time, the exact position of the robot over the map.

Resumen

UNIVERSIDAD CARLOS III DE MADRID
Departamento de Sistemas y Automática
por Jorge García Bueno

Este proyecto ha sido desarrollado como una propuesta para implementación de una de las técnicas de SLAM denominada GraphSLAM. Esta técnica aplica la teoría de grafos para crear un sistema de optimización en tiempo real que permite a los robots mapear un escenario y localizarse utilizando una cámara de tiempo de vuelo como fuente de información. Para llevarlo a cabo, ha sido desarrollado y calibrado un sistema RGB-D que tiene como finalidad crear una nube de puntos 3D. Con esta información, el detector de características estima, como primera aproximación, la posición de la cámara. A continuación, mediante ICP se realiza una corrección más fina de la estructura del grafo. Finalmente, mediante un optimizador global de grafos denominado HogMAN se cierra el bucle en cada iteración basándose en *manifolds* jerárquicos. Como resultado, se generan mapas 3D a color que contien, al mismo tiempo, la posición exacta del robot dentro del mapa.

Abbreviations

RGB-D **R**ed **G**reen **B**lue **D**epth

SLAM **S**imultaneous **L**ocation **A**nd **M**apping

DOF **D**egrees **O**f **F**reedom

CPU **C**ontrol **P**rocess **U**nit

GPS **G**lobal **P**ositioning **S**ystem

EKF **E**xtended **K**alman **F**ilter

ROS **R**obot **O**perative **S**ystem

TOF **T**ime **O**f **F**light

FOV **F**ield **O**f **V**ision

CUDA **C**ompute **U**nified **D**evice **A**rchitecture

CMOS **C**omplementary **M**etal **O**xide **S**emiconductor

CCD **C**harge **C**oupled **D**evice

SV **S**tereo **V**ision

PM **P**ulse **M**odulation

CWM **C**ontiuous **W**ave **M**odulation

NAR **N**on **A**mbiguity **R**ange

SIFT **S**cale **I**nvariant **F**eature **T**ransform

SURF **S**peeded **U**p **R**obust **F**eatures

NIR **N**ear **I**nfra **R**ed

DoG **D**ifference **o**f **G**aussians

ICP **I**terative **C**losest **P**oint

VFM **V**oronoi **F**ast **M**arching

ROS **R**obot **O**perative **S**ystem

RANSAC **R**andom **S**ample **C**onsensus

K-NN **K** **N**earest **N**eighbours

pdf **p**robability **d**ensity **f**unction

LCS **L**ocal **C**oordinate **S**ystem

HOG-Man **H**ierarchical **O**ptimization **G**raphs on **M**anifolds

RSF **R**obotic **S**oftware **F**ramework

PCL **P**oint **C**loud **L**ibrary

Chapter 1

Introduction

Robotics is considered nowadays as one of the most challenging and demanded fields of research for engineers, mathematicians and physicists. It covers a large number of branches: mechanics design, control, perception, human-robot interaction, machine learning, actuators, sensing, manipulation, training or even medicine among others, creating not only a complete line of research but also a way for living. Robotics is expected to take part of our lives gradually and fulfill the human needs by doing their main task: ***make human beings's live easier***

1.1 Story of robotics

Robotics is the art of perceiving and take actions though devices controlled by a computer or machine. That is, to make actuators interact with real life by means of sensors that react to external stimulus. The term **robot** was originally assigned by Karel Capck in a book named *Rossum's Universal* published in 1921. In this work, the word appears as a successor of the Czech expression *robota* which means servitude: those machines with limited intelligence designed to perform the hard jobs. Nowadays this term has progressed into a wide description of any autonomous system able to perform a work by itself reacting to external incitations.

Figure 1.1: *Example of robots* From left to right: Toyota Partner Robot (*Toyota Inc.*), ABB Robot IRB 2400 (*ABB Robots Inc.*), Asimo (*Honda Motor Co., Ltd.*), Maggie (*Robotics Lab, UC3M*), Da Vinci Surgical System(*Intuitive Surgical Inc.*), Roomba (*iRobot Co.*), Robotic Fish (*Essex University*), quadcopter, Nasa Mars Rover (*Jet Propulsion Lab. NASA*)

1.2 Classification of robots

Furthermore, robots can be classified following several patterns. Depending on the target or the point of interest, the following list present a classic classification methods:

1. *Arm configuration:* Rectangular, cylindrical, polar coordinates. Jointed arm
2. *Shape of workspace:* Limited sequence, Point-to-Point, continuous path
3. *Locomotion and kinematics:* Stationary, wheeled, legged, swimming or flying motion, ...
4. *Type of controller:* Distributed or centralized
5. *Type of power:* Electrical, pneumatic, hydraulic
6. *Size:* nano, small, big or huge
7. *Type and number of joints:* rotary or linear

8. *Type of technology:* Low, medium or high technology level

9. *Task being performed:* Industrial, domestic, medical, service, military, entertainment, exploration

10. *Generation of design:* $1^{st}, 1.5, 2^{nd}, 2.5$ or 3^{rd} generation

11. *Type of motion:* slew motion, joint-interpolation, straight-line or circular interpolation

1.3 The aim of this project

In this project, a SLAM introduction is presented in Chapter 2. Therein, different alternatives and ways to solve the problem are stated. Afterwards, a GraphSLAM solution is presented in Chapter 3, where each section introduce the steps followed to perform the complete system; from the input sensor, a Time of Flight camera, to the final graph optimization strategy passing through the feature detection and the pose estimation procedures. A mobile manipulator robot called Manfred has been selected to accomplish this work (more information about the complete system is included in chapter 4).

The complete GraphSLAM algorithm has been implemented for this robot using ROS framework. As it will be explained lately, thanks to this algorithm the robot will be able to navigate and recognize its own location and path in unknown maps.

To create the graph, each node will represent a scan point cloud, and each edge connecting two nodes will correspond to the camera pose transformation between those scans. Those edges will be set with the pose estimation given by an ICP cloud matching algorithm. The initial estimation of the pose transformation required by ICP will be done by means of features extraction over the color map. This graph will be optimized using HOGMan in order to decrease the accumulated error on each iteration.

Chapter 2

SLAM Principles

In this chapter the theoretical concepts of SLAM will be explained. Furthermore, a list of the most used methods will be explained briefly to fully understand the problem and the tackled solutions until now. SLAM stands for *Simultaneous Location And Mapping* and it is a technique commonly used on mobile robotics to determine the position of the robot in an unknown place by means of a probabilistic map of the surroundings, tracking the path generated by the robot.

2.1 Definition

One of the most important problems that arise when researching on mobile manipulators field is to know on each moment the exact position of the robot in the space (*6 DOF*). As [1] defines, SLAM answers to two common questions in relation with this problem:

1. *Where am I?*
2. *How is the world around me?*

A precise robot's location is only valid if a robust reference is taken, and therefore, a robust map of the obstacles and environment is generated. Any sensor which provides

confidence to the system is welcomed, with the objective of completing the information of odometry sensors. As it's known, odometry sensor measurements are usually poor and imprecise, suggesting any different source of information to be provided in order to improve the robot's location.

SLAM problem is still not solved. A large number of research centers and labs are working on it, trying to obtain a fast, on-line and robust method for SLAM [2] [3] [4] [5]. The reason why it is still unsolved falls on the measurements noise in the sensors and the limitations in resolution and accuracy (See figure 2.1). The most remarkable factors which prevent the process to be easier are:

1. Observations are taken with respect to the robot's reference system, which its position is affected by the uncertainty of the odometry. So, not only position but measurements are exposed to errors, getting error minimization problem even more difficult.

2. In most cases, a big map is required to be obtained, moving on extra computational cost and more imprecision in the odometry as robot moves.

3. Surroundings are normally dynamic, specially in mobile manipulator robots or humanoids which are designed to collaborate with humans or uncontrolled workspaces. Those transition objects can be treated as noise, and be removed by means of probabilities.

4. The association of observations with real map objects can be complicated as long as objects are similar between them geometrically or in texture. Normally, a deterministic correspondence does not make sense, turning into a probabilistic correspondence.

5. Sensors can be simplified to have a planar conception working perfectly on 2D environments but having difficult the transition to 3D environments, increasing the complexity and initial statements.

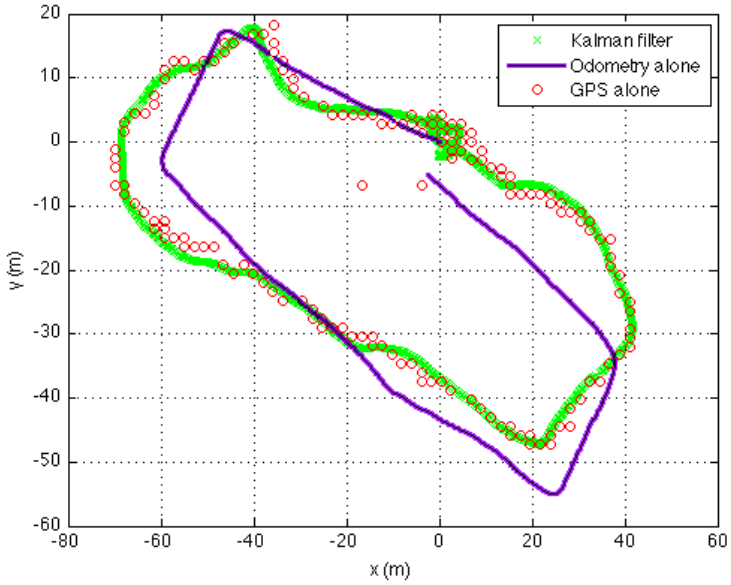

Figure 2.1: *SLAM basics.* Kalman Filter, odometry and GPS signals represented for the same path

2.2 Facing SLAM problem: Taxonomy

SLAM problem has been treated from different points of view, sharing all of them a final purpose of establishing a position over a spatial map.

- **Volumetric Versus Feature-Based** In volumetric SLAM, the map is reconstructed at a resolution high enough to be able to build a map with photographic resolution. On the other hand, feature-based SLAM only highlighted environmental features are extracted from the sensors. Those features are the ones used to create the map, much more efficient but inaccurate due to the data reduction.

- **Topological Versus Metric** Other mapping techniques only take into account a qualitative description of the environment that resumes the basic locations. This is the topological point of view. A topological map can be defined as a set of different places with several relations in common (such as A is close to B). Metric methods, in the other hand, only gather metric information about the relations

between places. During the last years, topological methods have been displace on a second row, even realizing that human beings usually use this topological information to locate themselves in new unknown places.

- **Known Versus Unknown Correspondence** One of the most difficult issues when performing SLAM is data association. The problem consist on relating the identity of the objects observed in instant t with the ones observed during instant $t-1$. Some alternatives assume that the identity of the landmarks is done while others do not. Those last stipulate special mechanism to estimate the correspondences for the observed features stored in the map.

- **Static Versus Dynamic** Static SLAM algorithms assumes that environment does not change with time. However, dynamic methods believes that changes in the environments can exist. Most part of the bibliography treat the problem as static, removing dynamic effects converting them into noise. Methods that integrate the movement of the environment are more complex, but tend to be more robust in most of the applications.

- **Small Versus Large Uncertainty** Depending on the level of uncertainly that robots are able to handle during location, SLAM algorithms are also classified. Simplest can only deal with small errors to succeed while complex systems might withstand with paths with intersections or large changes in direction. In case the robot can reach the target using multiple alternatives system uncertainly will get highly increased. Furthermore, the well-known loop closing problem takes an important part in this kind of classification. The ability to detect and fuse the data when closing a loop is one of the challenging research lines right now.

- **Active Versus Passive** Depending on the decision-making of the SLAM algo-

rithm over the movement of the robot, the strategy can be active in case it does, or passive in case the designer decides where the robot goes to. Almost all alternatives offer freedom of movement to the robot. Active SLAM uses to explore actively the environment trying to create a precise map in the fewest time.

- **Single-Robot Versus Multi-robot** Multi-robot platforms are a new alternative where multiple robots explore the environment and share the information with the surrounding colleagues. Communication and synchronization problems, BW limitation and delays turn this technique into a hard question to solve.

So, as it has been listed above, there exists a large quantity of strategies to solve the location and mapping problem. Depending on the ultimate application, sensors, conditions and statements,

2.3 Three ways to solve the same problem

During this section, the three most common alternatives to solve the SLAM problem are described. From those main three, some other alternatives have been developed afterwards refining the results and improving the details.

The first one, very well known as *Extended Kalman Filter (EKF)* was the premier historically, but its use has been decreased due to the limitations this method posses. Even though, a version named *EKF-SLAM* was created proposing a single state vector to estimate the location of the robot and a set of features in the environment. A covariance matrix representing the uncertainty in these estimates and including the correlations between the vehicle and and feature state estimates.

The second method, based on graphic representations applies the non-linear optimization called *Sparse Non-Linear Optimization Methods* is nowadays the most widely used strategy to solve the problem.

Last but not least, the third method uses *Particle Filters* to solve the entire problem by means of non-parametric statistical filtering. This alternative is quite common in online-SLAM and introduces a new way to answer the association job.

2.4 Graph-Based SLAM Optimization Technique

Before explaining the GraphSLAM architecture, a resumed mathematical explanation of the SLAM concept will be written. As it has been mentioned before, it relies on the extended Kalman Filter (EKF) for representing the robotâĂŹs best estimate.

2.4.1 Mathematical statements

SLAM is formally described in probabilistic terminology due to the large uncertainly attached to the problem. Let say the location of the robot is x_t at time t. If the problem is two-dimensional $\{x_t\} = \{x, y, \theta\}$ contains the position coordinates x, y and the orientation θ for each instant t. The group of location measurements (path) is given as

$$X_T = \{x_0, x_1, x_2, \dots x_T\} \tag{2.1}$$

where T can move from time zero to infinite and the only known position is the initial x_0. As in other similar location problems, odometry takes an important role: introduce relative location information between two consecutive instants of time. If the motion estimated by means of odometry between time $t-1$ and time t is labeled as u_t, the relative motion of the robot will be denoted as

$$U_T = \{u_0, u_1, u_2, \dots u_T\} \tag{2.2}$$

where empirically is well known that U_T is not enough to obtain the exact position of the robot in any moment due to the lack of accuracy and noise of the wheel encoders.

The map of the environment the robot is exploring is denoted by m. This static map contains all the landmarks and features within their locations. What the robot

will measure in each point of time is the information between the features of m and the robot location x_t. This sequence of measurements is given as

$$Z_T = \{z_1, z_2, z_3, \dots u_T\} \tag{2.3}$$

where index starts by 1 because it contains the relative location information between u_0 and u_1. Figure 2.2 represents the previous assumptions graphically. It is really useful to understand the variables involved in the SLAM problem and their relationships.

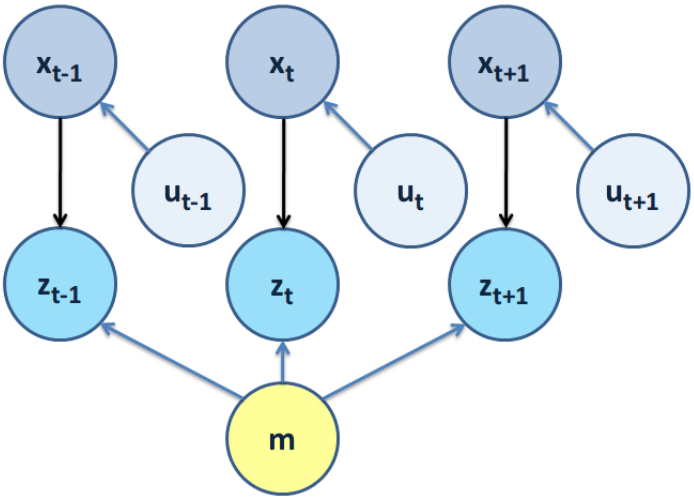

Figure 2.2: *SLAM basics.* Involved variables represented graphically. The location of the robot on each instant x_t is estimated using the odometry u_t, creating measurements contrasted with the map z_t

So, the target is to recover the world m and the path X_T from the odometry and measurements. There are two main alternatives to solve the same problem:

1. **Full SLAM problem:** This first method aims to solve the entire robot path with the map. That is, to calculate the joint posterior probability over X_T and m from the available data. Logically, this alternative is highly time consuming and requires of batch processing where the whole system is optimized at once, forcing this mentioned problem to be solved *off-line*.

$$p(X_T, m|Z_T, U_T) \tag{2.4}$$

2. **Local SLAM problem:** The second method, contrary to the previous one, tries to obtain uniquely the present robot location x_t and not the full path. This choice can be solved incrementally *on-line* processing one scan at a time. Those algorithms are usually called *filters* due to their time dependency. Therefore, the problem is described as

$$p(x_t, m|Z_T, U_T) \tag{2.5}$$

To find the solution to the SLAM problem, two models have to be taken into account. In one hand, a mathematical model which links the odometry u_t with the robot locations in two consecutive points x_{t-1} and x_t. In the other hand, a model that connects measurements z_t to the environment m and robot location x_t. Those models are the arrows in previous figure 2.3. Thanks to Bayes, it is possible to obtain probability distributions from measured data transforming the probability distributions

- $p(x_t|x_{t-1}, u_t)$ Probability of location x_t expecting robot to be in position x_{t-1} and with a measured odometry u_t), that is, the **motion model**.
- $p(z_t|x_t, m)$ Probability of measuring z_t from location x_t in a known map m, that is, the **measurement model**.

into a proper form.

The motion model g comes from the kinematics model of the robot. Using the location vector x_{t-1} and the motion vector u_t, function $g(x_{t-1}, u_t)$ calculates with only kinematics equations x_t. Therefore

$$x_t = g(x_{t-1}, u_t) \tag{2.6}$$

This model can be represented as a normal distribution centered at $g(x_{t-1}, u_t)$ with Gaussian noise, with a covariance matrix R_t

$$p(x_t|x_{t-1}, u_t) \backsim \mathcal{G}(g(x_{t-1}, u_t), R_t) \tag{2.7}$$

Covariance size is 3×3 since as it was mentioned before location vector is $\{x_t\} = \{x, y, \theta\}$. The measurement function h defines the information acquired by the sensors. Supposing that the sensors are *noise-free*, measurements can be obtained using environment information and robot location

$$z_t = h(x_t, m) \tag{2.8}$$

As before, the previous model can be represented as a normal distribution centered at $h(x_t, m)$ with Gaussian noise, with a covariance matrix Q_t

$$p(z_t|x_t, m) \backsim \mathcal{G}(h(x_t, m), Q_t) \tag{2.9}$$

2.4.2 GraphSLAM

As it has been mentioned before, this kind of techniques solve the problem through non-linear sparse optimization [6]. Their purpose claims to focus the problem from a graph representation way.

2.4.2.1 Mathematical relation

Taking map landmarks and robot locations as nodes in a graph, each pair of consecutive locations x_{t-1} and x_t are connected by an arrow that represents the information acquired by the odometry u_t. Moreover, the links between positions x_t and landmarks m_i are called *soft constraints*. These special constrains are relaxed to let the robot estimate the optimal map and full path X_T. So, as it is explained in figure 2.3, the constraints graph grows linearly with the elapsed time and the number of nodes in the

graph, creating a sparse matrix due to the poor connectivity between the nodes.

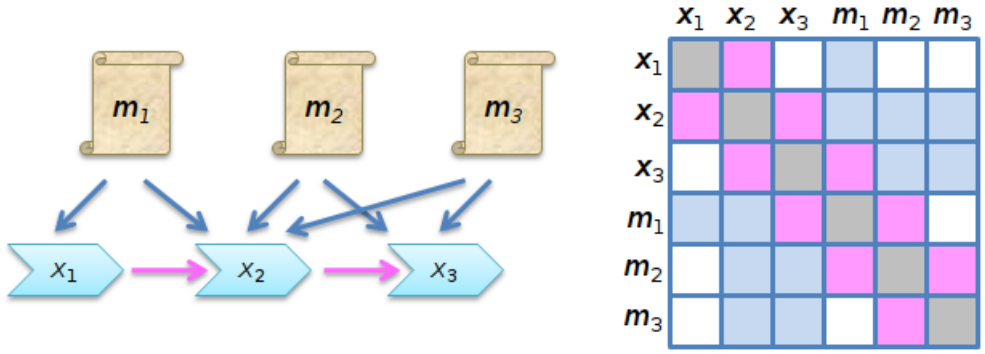

Figure 2.3: *GraphSLAM representation and constraints matrix.* Representation of nodes 1, 2 and 3 with the relations between them (left) and the generated sparse constraint matrix with those relations (right)

Analyzing the graph as a spring-mass model, SLAM solution seems equivalent to computing the state of minimal energy of this model. To see this, note that the graph corresponds to the log-posterior of the full SLAM problem denoted in 2.4.

$$p(X_T, m|Z_T, U_T) = k \cdot \prod_{t=1}^{T} p(x_t|x_{t-1}, u_t) \cdot p(z_t|x_t, m) \tag{2.10}$$

$$\log p(X_T, m|Z_T, U_T) = k + \log \sum_{t=1}^{T} p(x_t|x_{t-1}, u_t) + \log \sum_{t=1}^{T} p(z_t|x_t, m) \tag{2.11}$$

where k is a constant representing the initial conditions (t = 0). So, the target is to maximize the expression

$$\{X_t^*, m^*\} = \arg\max_{X_T, m} \{\log p(X_T, m|Z_T, U_T)\} \tag{2.12}$$

And assuming the previous Gaussian estimation,

$$\begin{aligned} \log p(X_T, m|Z_T, U_T) = k + \\ \sum_{t=1}^{T} [x_t - g(x_{t-1}, u_t)]^T \cdot R_t^{-1} \cdot [x_t - g(x_{t-1}, u_t)] + \\ \sum_{t=1}^{T} [z_t - h(x_t, m)]^T \cdot Q_t^{-1} \cdot [z_t - h(x_t, m)] \end{aligned} \tag{2.13}$$

Now, this sparse function has to be optimized. To do that, several options exist as *gradient descend* or *conjugate gradient*[1].

2.4.2.2 Linearizing the cost function

Usually functions g and h are linearized obtaining a pure quadratic function. Those equations are nonlinear due to effects of robot orientation. To linearize the position model it is possible to say that $x_t = F|_x + J|_x \Delta x$ using a column vector $F|_x$ and matrix $J|_x$ representing the Jacobian of the constraint equations with respect to the state. A single rigid-body constraint will provide three constraint equations filling a block-row of the Jacobian. Naming the search direction $d = \Delta x$ and defining the error residual $r = u - g(x_{t-1}, u_t).F|_x$

$$\begin{aligned} \log p(X_T, m|Z_T, U_T) &\propto (J|_x \cdot d - r)^T \cdot R_t^{-1} \cdot (J|_x \cdot d - r) \\ cost &= d^T \cdot J|_x^T \cdot R_t^{-1} \cdot J|_x \cdot d - 2d^T \cdot J^T \cdot R_t^{-1} \cdot r + r^T R_t^{-1} \cdot r \end{aligned} \tag{2.14}$$

Minimizing it means, it this case, to differentiate the cost with respect to d and setting the expression to zero. That is,

$$\frac{\partial cost}{\partial d} = \frac{\partial \left(d^T \cdot J|_x^T \cdot R_t^{-1} \cdot J|_x \cdot d\right)}{\partial d} - 2 \cdot \frac{\partial \left(d^T \cdot J|_x^T \cdot R_t^{-1} \cdot r\right)}{\partial d} + \frac{\partial \left(r^T R_t^{-1} \cdot r\right)}{\partial d} = 0 \tag{2.15}$$

$$\begin{aligned} 2 \cdot J|_x^T \cdot R_t^{-1} \cdot J|_x \cdot d - 2 \cdot J|_x^T \cdot R_t^{-1} \cdot r &= 0 \\ \underbrace{\left(J|_x^T \cdot R_t^{-1} \cdot J|_x\right)}_{A} \cdot d &= \underbrace{J|_x^T \cdot R_t^{-1} \cdot r}_{b} \end{aligned} \tag{2.16}$$

Now the problem is a typical linear algebra problem with the information matrix A. Solving this last equation 2.16 for d several times by re-evaluating $J|_x$ around the state estimate each time the method of nonlinear least squares is yielded. Typically, as it has been mentioned before, this problem is solved locally using well-known minimization methods such as Newton–Raphson, Gradient Descent or Conjugate Gradient Descent.

[1]Optimization classic definition: $A\mathbf{x} - \mathbf{b} = 0 \Rightarrow f(x) = \|Ax - b\|^2 \Rightarrow \nabla_x f = 2A^\top (Ax - b) = 0$

2.4.2.3 Soft Constraints

One of the advantages of graph representation is that the problem can be easily handled to include data associations. That feature allows, for instance, the addition of soft constraints between nodes. Those soft constraints can improve or include more details about the whole system. For instance, if landmarks m_q and m_r are exactly the same pose but they are obtained in different moments, it is possible to inform the system about this boundary condition by saying that

$$[m_q - m_r]^T \cdot \Gamma \cdot [m_q - m_r] \tag{2.17}$$

That is, if both landmarks represent the same information, one of the nodes can be removed and all the edges attached to that node can be shifted to the remaining one. In equation 2.17 Γ is a diagonal matrix which represents the penalty of not choosing m_q as the same node as m_r ($|\Gamma| \gg 1$)

2.4.2.4 Comparing GraphSLAM with EKF

Graphical SLAM methods includes a really useful advantage over the EKF method: *scalability*. Thanks to graph concept it is possible to scale to much higher dimensional maps. Contrary to graph methods, EKF SLAM increases covariance matrix quadratically with the number of observations (that is, with the map size). Furthermore, the number of optimizations is also affected proportionally with the map size, requiring more computational time to be done.

In the other hand, the update time of graph methods is constant, while the amount of memory required is considered linear if some considerations are taken. But, it is true that the optimization of the log function 2.13 is never inexpensive. The resolution increases with the number of close loops and size of those loops inside the map. As [5] explains, the effectiveness of the EKF approaches comes from the fact that they estimate a fully correlated posterior about landmark maps and robot poses. Their weakness lies in the strong assumptions that have to be made on both, the robot motion model and

the sensor noise. Moreover, the landmarks are assumed to be uniquely identifiable. There exist techniques to deal with unknown data association in the SLAM context, however, if certain assumptions are violated the filter is likely to diverge.

Chapter 3

GraphSLAM Architecture

3.1 Architecture of the proposed system

In this chapter, the description of the proposed GraphSLAM architecture will be detailed. Firstly, a brief description of the whole system will be presented and afterwards each part of the developed architecture will be explained deeply. In figure 3.1 the complete processing steps are represented. This approach will generate a full 3D color map from a set of RGB-D images.

All the beyond steps have been done using ROS architecture as it will be described later on. In a very first place, the GraphSLAM algorithm will need a RGB-D feed. For this requirement a ToF camera with a color webcam attached has been provided. It will be input source of information, with the environment information and the color for each pixel. However, not only the raw cloud point is needed, also the calibration information of the camera and its distortion factors are required to do the inverse transform and obtain 2D/3D projections for any pixel. In this case, as it will be explained later on, some alternatives have been implemented.

Once the 3D cloud point is received, its time to introduce this new information in the system. To do that, a feature extraction step is required. This process will get the

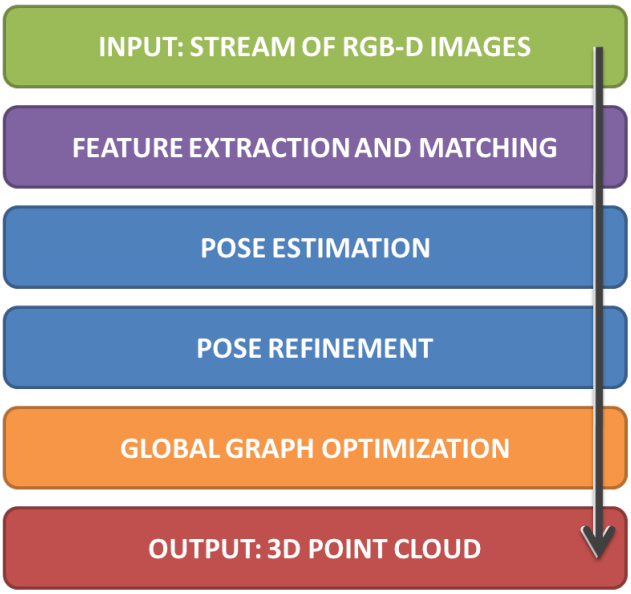

Figure 3.1: *System flow* The proposed GraphSLAM architecture constructs a full 3D color map from a set of individual RGB-D images

relation between some of the points of the last scan and the actual one. With that relation, is possible to compute in next step so called pose estimation the movement of the camera (and in fact, the robot) during the last scan. With the world camera's transformation (R, t) both scans can be matched up. However, this last step is normally not enough to obtain a clear and nice matching. Feature extraction is not completely accurate and propagate some errors into the global pose estimation algorithm.

For this reason, after the "first approach", another one is performed. This time, it is denominated a refinement because its main target is to reduce the error between matches as much as possible. There exist several techniques to solve this cloud matching and here ICP is used as it is explained during the next sections. Straightaway, a graph-based optimization is performed to relax the nodes and diminish the global system energy.

3.2 Environment acquisition. ToF technology

Just because humans are living in a three–dimensional world, they are provided with an adequate set of tools to describe and locate different objects in any surrounding scene. These given features include motion, relative position, size and evolution of perceived objects. The demand of spatial perception has been satisfied by nature providing animals and humans with at least two eyes. This stereo vision ability allow humans to process an image flow inside the brain and compute precisely depth measures of the observed environment.

3.2.1 Previous alternatives

Before ToF technology was introduced, there were several methods to acquire and estimate 3D point clouds: those classical stereo vision algorithms which are based on correspondence matching such as dense depth maps generated with Dynamic Programming [7], block-matching approach [8] or even improved methods based on various consecutive frames to enhance the results [9]. Besides, instead of passive systems like stereo vision, active sensors came up removing problems such as illumination conditions, unfocused scenes or image artifacts. This alternative is essentially a laser scanning line that delivers an specific signal and measure the received answer. Those methods have been widely implemented in location problems, SLAM, environment modeling, surgery or industrial applications.

3.2.1.1 Stereo Vision System

Stereo Vision systems comprise two perspective cameras with limited *Field Of Vision* (FOV). Any physical point is found in the observed 3D-space using both cameras. For each pixel in one image, the appropriate location in the other view must be found. Assuming that both cameras are perfectly calibrated, undistorted and rectified, image planes for both images are coplanar if optical axis are exactly parallel. In that case,

both cameras would have equal focal lengths $f_l = f_r$ and also equal *principal points* [1] ${c_x}^l = {c_x}^r$ as the following Figure 3.2 describes.

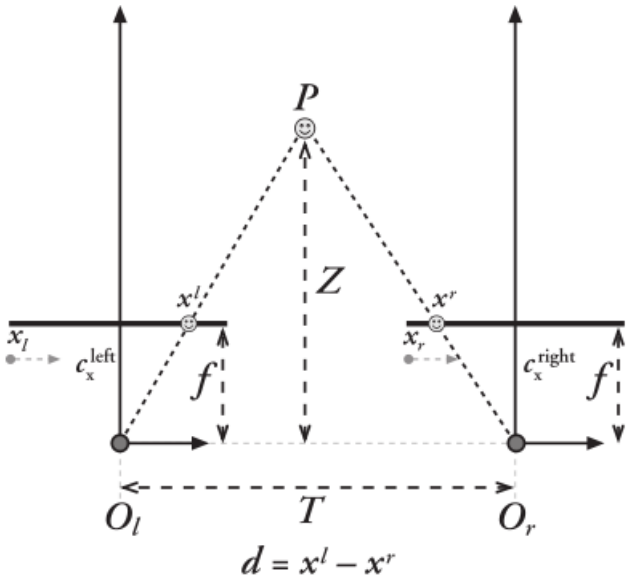

Figure 3.2: *Stereo Vision.* Perfect model of a stereo-pair for depth acquisition. (Bradski and Kaehler, 2008)

In order to do that, pinhole model can be easily imposed in both cameras, giving Equation 3.1 the corresponding relation between depth and disparity on pixels location using the triangulation principle.

$$\frac{T - (x^l - x^r)}{Z - f} = \frac{T}{Z} \rightarrow Z = \frac{fT}{x^l - x^r} \tag{3.1}$$

where $x^l - x^r$ is defined as *disparity* For this action, the most obvious drawback found is the *correspondence problem*, that is to match the *pixel-wise* pairs in both images. This operation requires a large consumption process in terms of computing resources and time to achieve good results due to the fact that pixels are not easy to find. Several applications have been developed during the last years in order to decrease computing times using new concepts such as parallel computing using CUDA [10] , dynamic programming or Parallel cells such as [11].

[1] A principal point is where the principal ray intersects the camera plane. This intersection depends just on the optical axis of the lens

3.2.2 Time-Of-Flight technology

Once again, nature has beaten humans when talking about intelligence. Several thousands of years have passed since bats or dolphins were able to see without proper eyes. Those species use this sensor to both navigate and object tracking. This feature make them possible to detect and locate external actions or events in order to escape or attack. Humans have applied time-of-flight measurement systems later on, for instance when measuring the unknown depth of a well listening to the returned echo after a stone was thrown. These ToF methods are based on the propagation time of sound instead of light. It was in the 17^{th} century when Galileo Galilei performed an experiment to estimate the speed of light [12]. To do that, he took two people handling a torch and placed them at the top of two mountains one kilometer far. If one of them turned it on, the other would do the same and *viceversa*. With this simple experiment, he tried to measure the light speed neglecting the time of reaction of the contributors.

There are two mainly approaches currently employed in ToF technology [13]. The first one uses modulated, incoherent light and it is based on a phase measurement that is possible to be implemented in standard CMOS or CCD technology. The second solution is based on an optical shutter technology having first used in studio cameras and later on miniaturized cameras. In Figure 3.3 are represented the most notable ToF cameras and their respective manufacturers.

Figure 3.3: *ToF Cameras.* PMDTec CamCube, MESA SR4000 and Canesta sensors from left to right

The basic principle of a ToF camera is represented in Figure 3.4 and it is resumed

by [14] as follows: A source emits a light pulse and starts a highly accurate stopwatch. The light pulse travels to the target and back. Reception of the light pulse by the detector mechanism stops the stopwatch, which now shows the time of flight of the light pulse. Considering the fact that the light pulse travels the distance twice (forth and back) and that the speed of light is *299.792.458 m/s*, then a measured time of 6.67 ns corresponds to a distance of one meter. As it is logical, the hardest problem here is to create a high accuracy time measurement system able to deal with nano and pico seconds. For instance, a resolution of 1 cm requires a time interval of 70 pico seconds. Because ToF cameras are highly compact devices, the active light source and receiver are located very closely avoiding shadowing effects. That is, illumination and observation directions are collinear [15]. Furthermore, one of the most important conditions of ToF sensors in general is that emitter and detector are operated synchronously, extracting the time of flight as accurate as possible.

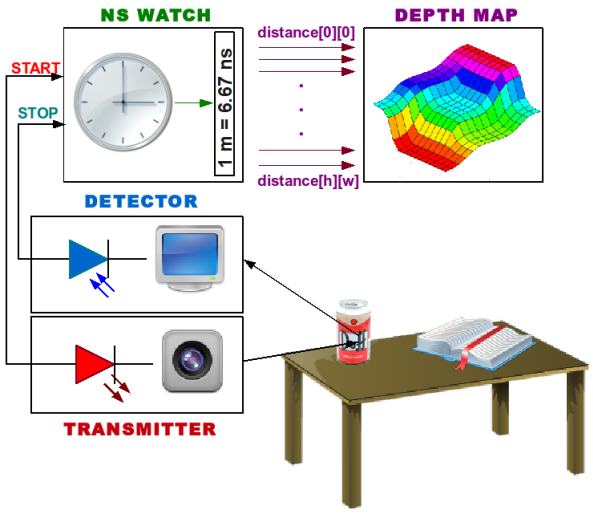

Figure 3.4: *ToF concept.* Schema of how ToF camera works.

ToF cameras are able to return three different sources of information. Firstly, a range map with a resolution of 204×204 pixels `float` precision in centimeters. This information it is obviously used to acquire object distances, scene segmentation and object modeling (Figure 3.5.a). Secondly, an intensity image that reveals the texture and brightness for each item inside the scene. That information becomes crucial for

pattern recognition and camera calibration (Figure 3.5.b). Finally, an amplitude image that contains an estimation of the committed error measuring the time of flight for every pixel (Figure 3.5.c). Exploiting this information may increase the reliability of distance for each pixel, as it would considered subsequently with the discussion of the shading constraint [16]. In Figure 3.5.d it is displayed the original scenario to obtain a better understanding.

Figure 3.5: *ToF outputs.* Depth map, intensity image, amplitude map of the PMD camera and original scenario.

3.2.3 General Image processing chain

3.2.3.1 Common arising problems

There exist several problems when processing images of captured objects. *Aperture problem* leads the list exhibiting the limited aperture angle of the camera optics giving information of a partial environment. *FOV problem* arises when fixing the field to an specific application. 3D information get lost when projecting information into a planar sensor, that is, the *optical projection problem*. Finally, the detection of fast object move-

ments cannot be sampled due to the sampling rate limit given by the camera calling to this situation *sampling problem*. To all those problems, it has to be added the error committed measuring gray level changes in pixels (*divergence problem*).

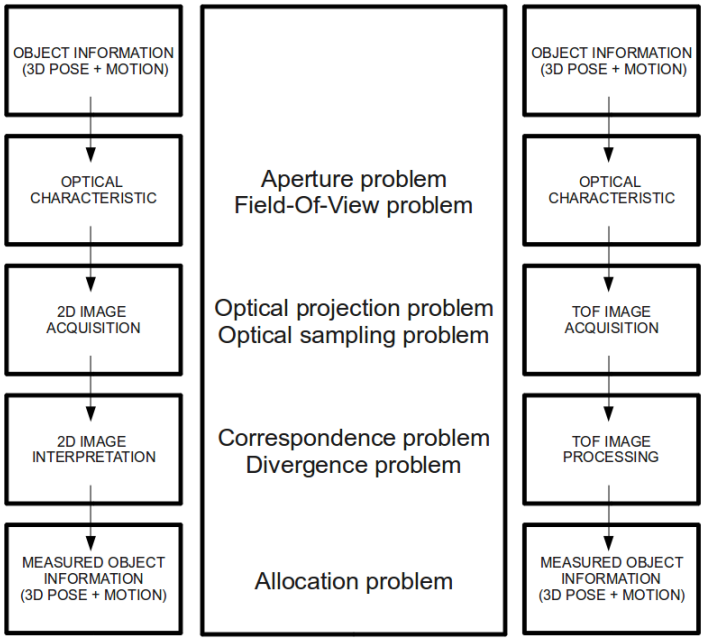

Figure 3.6: *Functional block* Image processing chain comparison between SV systems and ToF (Hussmann and Ringbeck, 2008)

ToF cameras do not contain every of those problems as Figure 3.6 shows. Only *sampling* and *aperture* problems appear. Other issues such as FOV problem does not affect just because hardware setup can be changed easily. There is not correspondence, divergence or allocation problem due to the fact that each pixel of sensor calculates a range value so no object information is lost by the optical projection on a planar sensor.

3.2.4 Field of View problem

ToF cameras do not depend on geometrical parameters counter to SV systems where the distance between cameras implies different triangulation possibilities and therefore a range of depth resolutions. The usage of active modulated light source makes ToF systems more effective and reliable.

Two different approaches are the most common modulation techniques used for depth measurement. The first one is *pulsed modulation*, introduced 20 years ago by [17] and not too much used. The alternative is *continuous wave modulation (CWM)* explained by [18]. CWM is nowadays used on ToF systems because this method do not required high rise and fall times allowing then several sources of light to be used. Usually, square waves or sinusoidal waveforms are applied for modulation. The idea is simply to measure the phase between sent signal and received signal instead of measuring the time to go and return of a single light source. Once modulation frequency f_{mod} is established, the measured corresponding phase means directly the time-of-flight [14]. Using Equation 3.9 and replacing ϕ by the frequency response of the modulation

$$\phi = (\frac{\varphi_0}{360^o} + N \cdot 360^o) \; with \; N = 0, 1, 2, 3, ... \tag{3.2}$$

In case of modulation, $2 \cdot \pi\omega$ in 3.9 equals to f_{mod} and therefore range of the camera can be expressed as

$$R = (\frac{c}{2 \cdot f_{mod}}) \cdot (\frac{\varphi_0}{360^o} + N \cdot 360^o) \; with \; N = 0, 1, 2, 3, ... \tag{3.3}$$

Studying the above Equation 3.3, if f_{mod} is set to 20Mhz as [19] recommends for typical PMD cameras, the *Non Ambiguity Range (NAR)* turns to

$$NAR = \max(R) = \frac{c}{2 \cdot f_{mod}} = \frac{3 \cdot 10^8 m/s}{2 \cdot 2 \cdot 10^7 s^{-1}} = 7.5\, m \tag{3.4}$$

giving an idea of the spaciousness those devices can take in. Equations 3.3 and 3.4 demonstrates that NAR depends only on the frequency f_{mod} applied to obtain a larger or shorter distance region.

3.2.5 Correspondence problem

As it has been mentioned before, one of the most relevant problems that emerge in ToF technology is the correspondence problem also known as by *correspondenceless*

[20]. The idea is quite simple, SV systems need of rich textured frames to offer good reliability. That is because disparity has to be found between equivalent pixels in both images. If gray values are quite similar along the *epipolar lines*, disparity levels would be erroneous inducing into bad disparity maps.

The following Figure 3.7 proves this problem. A small jug have been captured in the laboratory (Figure 3.7.a) and computed with a SV system (Figure 3.7.b) and then with a ToF system (Figure 3.7.c). It is trivial to find out how background is computed erroneously when it is poorly textured. Contrary to ToF cameras, SV systems require several settings such as disparity window size or maximum disparity levels to be defined beforehand, otherwise depth maps will not correspond to observed systems. Furthermore, due to SV systems run without active lighting they generate shadows creating false positives and hence, they can not estimate the 3D information of the objects due to the correspondence problem.

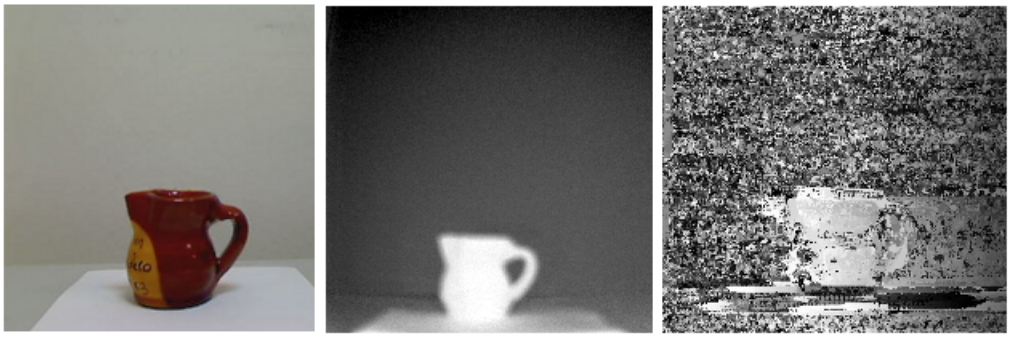

Figure 3.7: *Correspondence problem* Real example comparing SV and ToF technologies

3.2.6 Intensity Modulation Principle

Most important companies have focused out their prototypes following this kind of ToF-principle such as **Mesa Imaging** [2] (Figure 3.3.a), **PMDTech electronics** [3] (Figure

[2]Mesa Imaging – `http://www.mesa-imaging.ch`
[3]PMDTech electronics – `http://www.pmdtec.com`

3.3.b) and **CanestaVision Camera Modules** [4](Figure 3.3.c). The intensity modulation principle is based on the on-chip correlation of the incident optical signal s, which comes from a modulated NIR *near infra-red* source and reflected by the objects inside the scenario, with a reference signal g, which posses an internal offset τ:

$$c(\tau) = s \otimes g = \lim_{T \to \infty} \int_{-T/2}^{+T/2} s(t) \cdot g(t + \tau) dt \tag{3.5}$$

Choosing s and g as sinusoidal signals:

$$g(t) = \cos(\omega \cdot t), \qquad s(t) = b + a \cdot \cos(\omega \cdot t + \phi) \tag{3.6}$$

where ω represents the modulation frequency, a is the amplitude of the incident optical signal, b corresponds to the correlation bias and ϕ is the phase offset due to the incident object distance. The convolution of those signals yields

$$c(\tau) = \frac{A}{2} \cos(\omega \cdot \tau + \phi) + b \tag{3.7}$$

Every pixel of the sensor samples the amount of modulated light reflected by any object four times every period at equal intervals m_1 to m_4. These four values are sufficient to recover the sinusoidal signal easily. The phase offset between the emitted light and received signal is

$$\phi = \arctan(\frac{m_4 - m_2}{m_3 - m_1}), \qquad m_i = c\,(i, \frac{\pi}{2}), i = 1, \ldots, 4. \tag{3.8}$$

and this value determines the range of the object in the scene

$$R = \frac{c}{4\pi\omega} \cdot \phi, \; c \approx 299.792.458 m/s \tag{3.9}$$

The intensity of the objects in the image can be recovered from the average light reflected as

$$I = \frac{m_1 + m_2 + m_3 + m_4}{4} \tag{3.10}$$

The amplitude of the measured sinusoidal can be expressed as

[4]Canesta Vision – `http://canesta.com/`

$$A = \frac{\sqrt{(m_3 - m_1)^2 + (m_4 + m_2)^2}}{2} \tag{3.11}$$

and therefore it allows to predict the quality of the measurement ΔR as

$$\Delta R = \frac{c}{2\omega\sqrt{8}} \frac{\sqrt{I}}{2A} \tag{3.12}$$

With all this information, it is possible to obtain in real time not only depth values for each pixel but also the reliability or estimated error for each pixel.

3.2.7 Time-Of-Flight Applications

ToF cameras are a young family of devices which are progressing continuously. Last years the Time-of-Flight imaging became more attractive to a growing research community [21]. Nowadays 3D matrix cameras can be manufactured and be applied for many application such as robotics [22], automotive [23], industrial [24], medical [25] and multimedia [26] applications. The fast advances in ToF-camera market will grow during the next years. [19] expects the unit price of these systems in the mass production to drop down to 100 euros. Figure 3.8 shows an integrated system which captures hand gestures of the vehicle's driver and perform different activities such as *phone mode* activation or the name of the actual street taking the information of the GPS coordinates. All those actions are interpreted from 3D data captured with an embedded camera on the frontal zone of the car.

3.3 Texture feature extracion: SIFT

This algorithm was proposed by [27] as a revolutionary solution to achieve a large quantity of important points from bustling scenarios. This method has been used not only for points extraction but also to correlate photos of the same scenario from different points of view. The correct correlation of several clouds of points might be used for building a complete 3D scene from a batch of 2D samples [28], [29].

Figure 3.8: *ToF application* Hand gesture recognition inside a car using an embedded ToF camera (Hussmann and Ringbeck, 2008)

SIFT algorithm is divided mainly in four steps, each one extracts a different feature from the image giving at the end a list of points with a complete description representing the most important data fields for each frame [30].

1. *Maximums and minimums detection in the space-scale:* First step consist on the search of possible candidates to represent *keypoints*. This search is done using Gaussian Functions differences in order to identify points invariant to scale and orientation.

2. *Keypoints's localization:* After localize candidates, scale and localization is computed. Afterwards, those keypoints which are more stable are selected.

3. *Orientation assignment:* To each selected point, its orientation (or principal orientations) is selected. Those orientations are defined based on local gradients around the keypoint.

4. *Keypoints's descriptors:* Local gradients are measured and transformed into a descriptor vector. This representation allows to describe the distortion levels around keypoints and changes in illumination.

As it is going to be explained afterwards, the number of keypoints extracted from any image will be dependent of the number of objects, textures and edges of those objects. These keypoints can become very useful to have an idea of the important regions of the image and then to pay more attention to find out objects in these areas. Next sections will cover the four steps in detail, to get a global idea of how this method works.

3.3.1 Maximums and minimums detection in the space-scale

As it has been commented before, the first stage of SIFT algorithm corresponds to the detection of possible keypoints. To do that, stable features are searched along the frame on different scales. Once this requirement is fulfilled, a Gaussian function is in charge of space-scale changes.

Indeed, $L(x, y, \sigma)$ is defined as a space-scale function in an image. It corresponds to a convolution between a Gaussian of an scalar variable $G(x, y, \sigma)$ and the original image $I(x, y)$ in this way:

$$L(x, y, \sigma) = G(x, y, \sigma) * I(x, y) \tag{3.13}$$

where Gaussian function is defined as

$$G(x, y, \sigma) = \frac{1}{2\pi\sigma^2} e^{\frac{-(x^2+y^2)}{2\sigma^2}} \tag{3.14}$$

To perform an efficient calculation of stable keypoints, in scale and space, it is proposed the usage of maximums and minimums of $D(x, y, \sigma)$ function, that corresponds to the difference of Gaussians convoluted with the image. Difference of Gaussians can be obtained from two contiguous scales separated by a multiplicative constant k as

$$\begin{aligned} D(x, y, \sigma) &= [G(x, y, k \cdot \sigma) - G(x, y, \sigma)] * I(x, y) \\ &= L(x, y, k \cdot \sigma) - L(x, y, \sigma) \end{aligned} \tag{3.15}$$

This approximation is quite fast and easy to compute in a computer. Furthermore, it represents a good representation of the normalized Laplacian $\sigma^2\nabla^2 G$. That is important due to σ^2 factor makes the transformation scale invariant. It is also demonstrated that maximums and minimums of the normalized Laplacian function are more stable than gradient, Hessian function or even Harris corners.

The relation between D and $\sigma^2\nabla^2 G$ can be compared with the heat diffusion equation, where temporal parameter is $\sigma^2(t = \sigma^2)$

$$\frac{\partial G}{\partial \sigma} = \sigma^2\nabla^2 G \tag{3.16}$$

where in case of close scales $k\sigma$ and σ it can be approximated

$$\sigma^2\nabla^2 G = \frac{\partial G}{\partial \sigma} \approx \frac{G(x, y, k \cdot \sigma) - G(x, y, \sigma)}{k\sigma - \sigma} \tag{3.17}$$

therefore,

$$G(x, y, k \cdot \sigma) - G(x, y, \sigma) \approx (k - 1)\sigma^2\nabla^2 G \tag{3.18}$$

This result demonstrates that DoG function differs from normalized function just by factor $(k - 1)$. Because this factor affects on every scale, it does not change the maximums and minimums location. Next Figure 3.9 displays an optimal way to construct $D(x, y, \sigma)$ function.

Initial image is increasingly convoluted with Gaussians to produce separated images by a constant scale factor k (left row). Each initial image forms an octave. Each octave is divided a number of intervals s due to $k = 2^{1/s}$. Afterwards it has to be produced $s + 3$ images by octave in the group of fuzzy images so detection of maximums and minimums covers a complete octave. Finally adjacent images are subtracted to obtain a DoG image. After finishing the process for one octave, images have to be sampled again, but this time with a σ value double to initial value, taken the second pixel for each column and row. Precision is reduced in each iteration. Figure 3.10 shows some Gaussians of the same octave and how they get blurred.

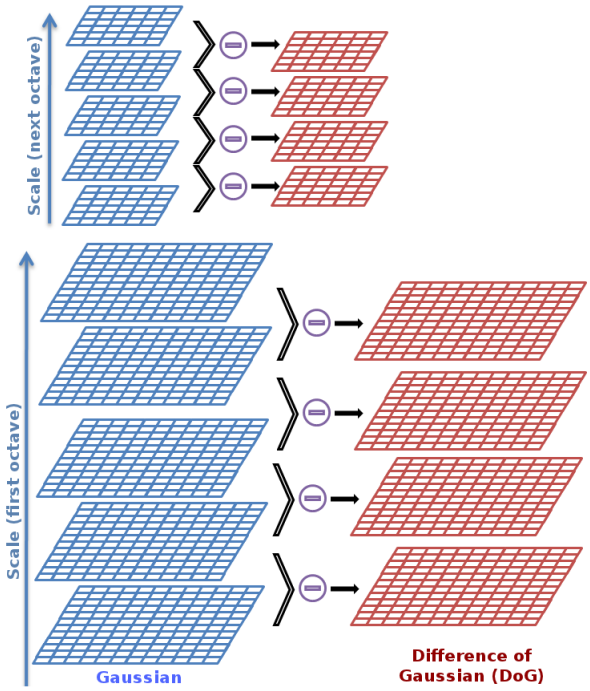

Figure 3.9: *Difference of Gaussians* How SIFT works. Each octave generates several DoG images. On each iteration, image is reduced and blurred again

To obtain the local maximums and minimums, not only the greater and closer pixels from the neighbors pixels are good candidates, but they also those pixels that are maximums and minimums on previous and next images in the same column as Figure 3.11 displays. The computational cost of this maximum and minimum obtaining is small due to most of the points are removed during the initial checking.

A small spatial sampling does not ensure a large number of stable points. It has to be chosen experimentally the number of scales per octave: if a large number of scales is chosen, a lot of instable points would appear being less repetitive.

3.3.2 Keypoints's localization

Once the keypoints are found, next step is to adjust their location, scale and brightness in the neighborhood. This information can reject points due to a low contrast or

Figure 3.10: *Difference of Gaussians* Three Gaussians of the same octave get deformed as fast as process iterates

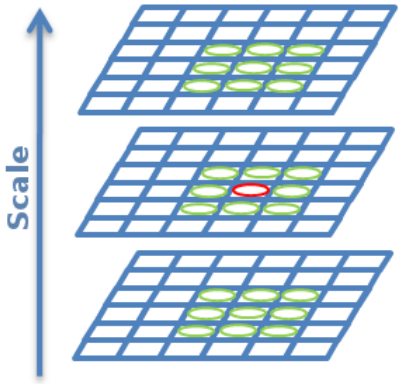

Figure 3.11: *Difference of Gaussians* Finding maximums and minimums not only in the same scale but also in upper and bottom scales

poor localization near an edge. Again, another approximation has been done to improve calculus and better computational times. In this case, DoG function has been approximated to a Taylor series as

$$D(x) = D + \frac{\partial D^T}{\partial x}x + \frac{1}{2}x^T\frac{\partial^2 D}{\partial x^2}x \tag{3.19}$$

where D and its derivatives are evaluated in the sampling point and $x = (x, y, \sigma)^T$ represents the offset in that point. The location if the critical point $\widehat{x}$ equaling Equation 3.19 to zero. Then, it is obtained that

$$\widehat{x} = \frac{\partial^2 D^{-1}}{\partial x^2}\frac{\partial D}{\partial x} \tag{3.20}$$

and it can be demonstrated that Hessian and derivative are approximated by the

use of difference of points around near samples. If the offset $\widehat{x}$ is greater than 0.5 in any dimension, that means that there is a critic point very close, and a simple interpolation between both points is performed. Using Equations 3.19 and 3.20 the critical point $D(\widehat{x})$ is obtained as:

$$D(\widehat{x}) = D + \frac{1}{2}\frac{\partial D^T}{\partial x}\widehat{x} \tag{3.21}$$

According to [27], those points where $\|D(\widehat{x})\| < 0.03$ can be discarded for normalizes pixels between 0 and 1. And that removes all those points with a low contrast.

Nonetheless, there is another condition required for keypoints to became real candidates: those points with a low location along an edge, as DoG posses a high response along the edges. The principal curvature of a point can be processed from a 2x2 Hessian matrix evaluated in the keypoint

$$H = \begin{pmatrix} D_{x,x} & D_{x,y} \\ D_{y,x} & D_{y,y} \end{pmatrix} \tag{3.22}$$

The required derivatives to obtain H matrix have been computed taking differences with neighbors in the sampling point. The H eigenvalues are proportional to principal curvatures of D. Taking α as the greatest eigenvalue and β as the smallest, the summation of eigenvalues can be done from the trace as

$$Tr(H) = D_{x,x} + D_{y,y} = \alpha + \beta \tag{3.23}$$

and the determinant as the product

$$Det(H) = D_{x,x} \cdot D_{y,y} - (D_{x,y})^2 = \alpha \cdot \beta \tag{3.24}$$

Taking $\boldsymbol{R}$ as the ratio between those eigenvalues ($\alpha = r\beta$), it is obtained

$$\frac{Tr(H)^2}{Det(H)} = \frac{(\alpha+\beta)^2}{\alpha \cdot \beta} = \frac{(r+1)^2}{r} \tag{3.25}$$

that only depends on the relation between eigenvalues. So, according to [27], a reasonable threshold is $r = 10$ for Equation 3.26 to reject those instable points because of a poor location in an edge.

$$\frac{Tr(H)^2}{Det(H)} < \frac{(r+1)^2}{r} \tag{3.26}$$

3.3.3 Orientation assignment

The orientation of keypoints is quite important. A good assignment can make the point invariant to rotation. The approach here exposed is based on local image properties. This has a disadvantage, the number of descriptors selected is reduced, cropping areas of the image.

The selection of the orientation is based on local gradients around the keypoints. For this, image is blurred with the highest Gaussian in that point. In this way, calculus are done over information invariant to scale. For each sampling image, $L(x, y)$, gradient magnitude $m(x, y)$ and orientation $\theta(x, y)$ are calculated using pixel's differences:

$$\begin{aligned} m(x,y) &= \sqrt{(L(x+1,y) - L(x-1,y))^2 + (L(x,y+1) - L(x,y-1))^2} \\ \theta(x,y) &= arctg\left(\frac{L(x,y+1) - L(x,y-1)}{L(x+1,y) - L(x-1,y)}\right) \end{aligned} \tag{3.27}$$

An histogram of a keypoint is formed with the orientation of the sampling point's gradients around the keypoint's neighborhood. The orientation histogram is formed by 36 divisions that covers 360^o. Each sample added to the histogram has a weight times the gradient magnitude times a Gaussian mask with a 1.5 times the value of the keypoint. Strong directions in the orientation histogram correspond to dominant directions in local gradients. The maximum in the histogram is recorded and compared with the second maximum. If there exist maximums above 80% the largest one, those will be used to create new keypoints with different orientations, creating a keypoint set with the same location but different orientation.

3.3.4 Keypoints's descriptors

Once all keypoints are found, processed and segmented, keypoint's neighborhood is divided into 4×4 regions of 4×4 pixels (See Figure 3.12). Then a gradient orientation histogram is generated for each region with a weight Gaussian function with $\sigma = 4$ pixels. To decrease conflicts made by small displacements, each pixel's contribution is multiplied by a weight $1 - d$ where d represents the distance to the neighborhood's center. Because orientation histograms for each region are divided on 8 columns, for each neighborhood a three dimensional histogram of $4 \times 4 \times 8(128)$ values each.

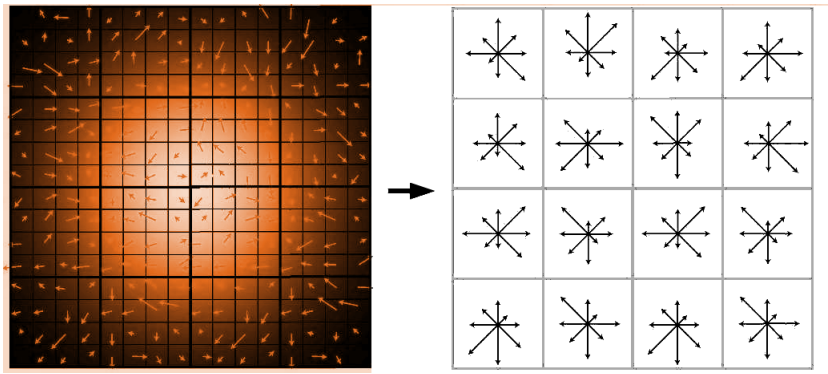

Figure 3.12: *Keypoints orientation* Divisions performed to create a gradient orientation histogram

3.3.5 Keypoints's matching between different subsets

Once feature extraction is done, it takes a very interesting point to match keypoints from different scans or images. This matching can determine the similarity between both views. In case there exist a large number of equalities or matches between two images, it is possible to determine the physical relation between both views, and what is more, the world transformation between them. Figure 3.13 shows an example of this matching. If the camera is well defined (it is calibrated and all its parameters are known) it is possible to obtain extrinsic parameters: Rotation matrix $\boldsymbol{R}$ and translation

vector $\boldsymbol{t}$.

To do this matching, a K-NN search is done over the data base of descriptors using the euclidean distance for the measurements. Once the distance of all the keypoints is done for each keypoint, the relation between the two closest is done. If this relation is lower than a certain threshold [5], its possible to confirm that there is a relation between the keypoint in the first image and the second one.

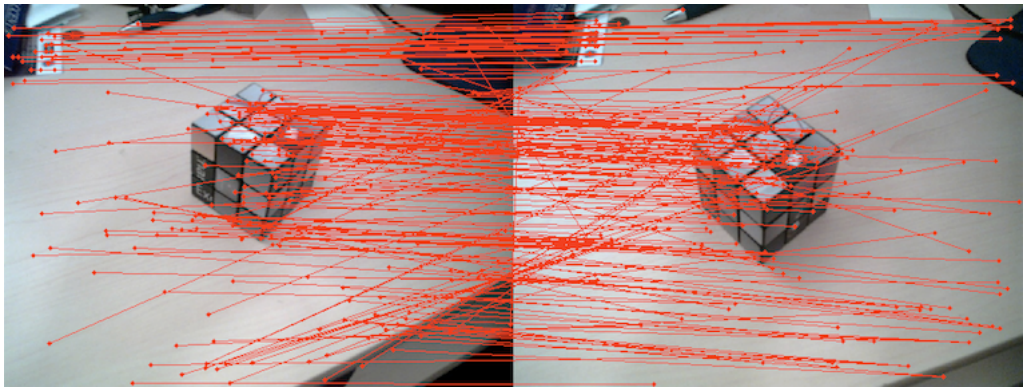

Figure 3.13: *Keypoints matching* Matching between keypoints is done using K-NN search over the 128 descriptor

3.4 Estimating 6 DOF pose though 3D cloud points

Successive frames are aligned by jointly optimizing over both appearance and shape matching. Appearance-based alignment is done with RANSAC (*Random Samples Consensus*, originally proposed by [31]) over SIFT features annotated with 3D position (3D SIFT) proposed by [32]. This method permits to estimate the transformation between two consecutive frames projecting and un-projecting each matched pair-feature and optimizing the transformation matrices recursively. RANSAC is an iterative optimization method whose task is to pick up the best observations and reject the outliers affecting to the whole system.

[5]Empirically, this value is set between 0.8 and 0.9 to avoid false positives during matching process

3.4.1 Definition of outliers

The definition of an outlier is not easy, as [33] annotates:

A datum is considered to be an outlier if it will not fit the "true" model instantiated by the "true" set of parameters within some error threshold that defines the maximum deviation attributable to the effects of noise.

That is, it is possible to assume that some of the points in a distribution are *correct* and the rest are not. This last subset is called outliers. In this case, for the estimation of the camera transformation between two consecutive scans, outliers are detected over the SIFT matching results.

In fact, the rejection system proposed by [27] and explained in subsection 3.3.5 is not perfect. For this reason, RANSAC can select which of those matching pairs are wrong and remove them from the initial guesses, improving the pair-wise pose estimation after all. With this process, and after the algorithm converges, a selection of the best subset of matchings will be done, with the certain of choosing the optimal transformation. Applying this technique to the initial matching set represented in figure 3.13 the result is displayed in figure 3.14.

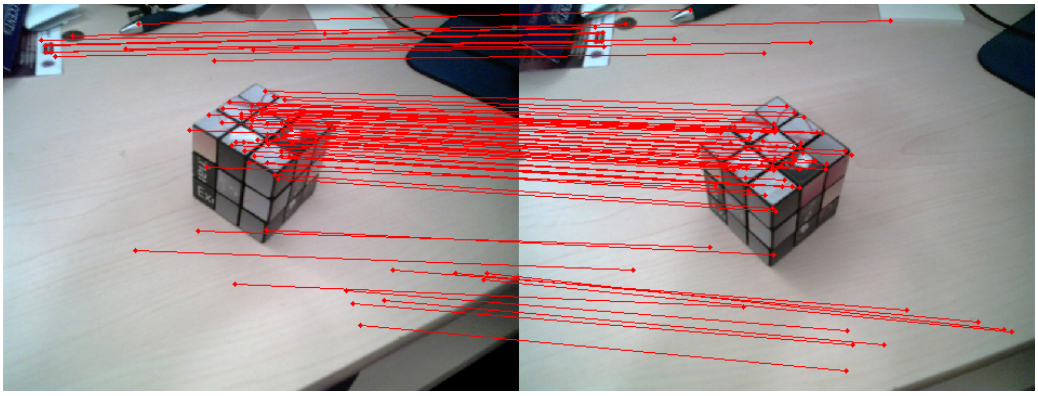

Figure 3.14: *RANSAC over SIFT points* RANSAC removes the outliers of the initial distribution improving the resulting matching

As it can be shown easily, those matching pairs which did not follow the same orientation as the majority do are removed (outliers), remaining the correct ones (inliers).

3.4.2 RANSAC Algorithm

The algorithm is quite simple to implement. The basic steps are as follows:

Algorithm 1 RANSAC pseudo-code

1: Select randomly the minimum number of points required to determine the model parameters.
2: Solve for the parameters of the model.
3: Determine how many points from the set of all points fit with a predefined tolerance ϵ
4: If the fraction of the number of inliers over the total number points in the set exceeds a predefined threshold τ, re-estimate the model parameters using all the identified inliers and terminate.
5: Otherwise, repeat steps 1 through 4 (maximum of N times).

3.5 Local refinement and matching with ICP

However, this pose estimation is not sufficient to obtain the best matching. Some of the reasons are

1. SIFT is not always exact while extracting the features: it depends on light conditions, movement of the camera, abrupt changes of light or occlusions.

2. Pair matching can fail. Furthermore, the number of matching pairs can be insufficient enough to estimate a wrong pose.

3. The fact that there is not a minimum number of pairs to estimate the pose (with three is sufficient) can cause incoherences.

For this reason, after the initial guess, a second step will refine and improve this 3D cloud matching. The state of art in clouds matching is large, but the most famous algorithm for cloud matching is called ICP (*Iterative Closest Point*). It is important to remember that now the cloud point is matched using directly the 3D points ($P(x, y, z)$) while in RANSAC, the classification was done using the pair descriptor's matching orientation.

3.5.1 ICP method

ICP was originally presented by [34] and proposed for point matching for free-form curves and surfaces, but it can be easily extended to three-dimensional problems or any N-dimensional system. As it has been mentioned before, ICP is in charge of refining the 3D point cloud focusing on the position of each point. It iteratively revises the transformation (translation, rotation) needed to minimize the distance between the points of two raw scans.

The original algorithm is presented in Algorithm 2. It's complexity is low enough to work in real time even with 3D cloud points. The key concept of the standard ICP algorithm can be summarized in two steps:

1. Compute correspondences between the two scans.
2. Compute a transformation which minimizes distance between corresponding points.

For this project, a modified version of the algorithm proposed by [35] called *Generalized ICP* has been used. Generalized-ICP is based on attaching a probabilistic model to the minimization step on line 10 of Algorithm 2. This option does extract planes from both clouds, doing a plane matching instead of point matching. This probabilistic approach mix the simplicity of the original proposal over the advantages of other fully probabilistic techniques: speed and simplicity.

Algorithm 2 Iterative Closest Point pseudo-code

Require: Two point clouds: $A = \{a_i\}$ and $B = \{b_i\}$

Require: Initial transformation : T_0

Ensure: The correct transformation $\boldsymbol{t}$ which aligns A and B

$T \leftarrow T_0$
while not converged **do**
  for $i \leftarrow 1$ **to** N **do**
    $m_i \leftarrow \texttt{FindNearestPointInA}(T \cdot b_i)$
    if $||m_i - T \cdot b_i|| \leq d_{max}$ **then**
      $w_i \leftarrow 1$
    else
      $w_i \leftarrow 0$
    end if
    $T \leftarrow \underset{T}{\arg\min} \{\sum_i w_i \cdot ||T \cdot b_i - m_i||^2\}$
  end for
end while

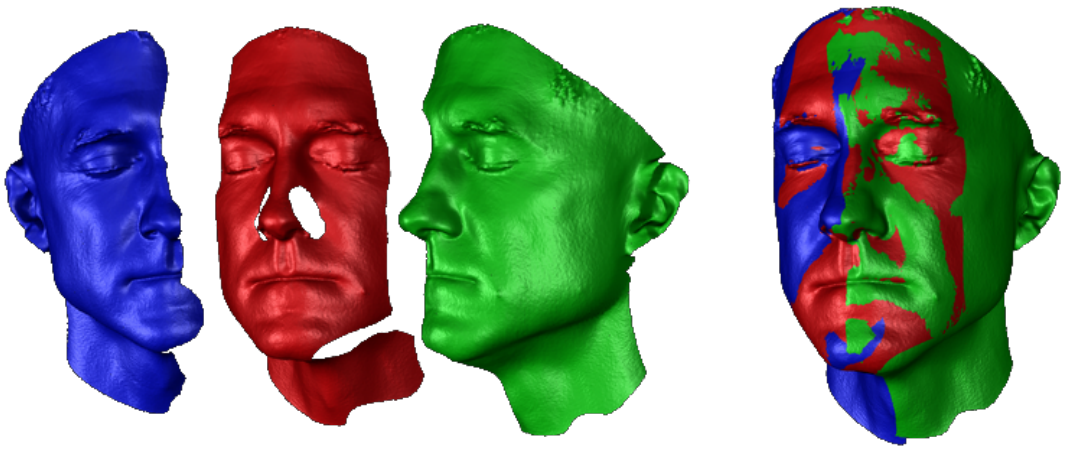

Figure 3.15: *ICP matching example.* Matching of three 3D scans of a human face. Left: three scans. Right: Results after ICP minimization refinement (Matt Chiang, NTU)

3.5.2 Optimization function

For the 3D cloud matching problem, the cost function which ICP has to deal with is the following: Given the previously defined two independent sets of 3D points A (model set, $|A| = N$) and B (data set, $|B| = N$) [6] which correspond to a single shape, it is aimed to find the transformation consisting of a rotation $\boldsymbol{R}$ and a translation $\boldsymbol{t}$ which minimizes the following cost function:

$$E(\boldsymbol{R}, \boldsymbol{t}) = \sum_{i=1}^{N} \sum_{j=1}^{N} w_{i,j} \cdot ||m_i - (R \cdot d_j + t)||^2 \tag{3.28}$$

$w_{i,j}$ is assigned 1 if the i-th point of A describes the same point in space as the j-th point of B. Otherwise $w_{i,j}$ is 0.

3.6 Global pose graph optimization

The last step of the proposed Graph-SLAM algorithm is the global graph optimization. This final action includes a very specific target: to improve the matches between nodes by relaxing the whole system optimizing the edges between all the nodes. Furthermore, the system has to detect loop closures while running and adapt the nodes properly.

3.6.1 Data association: χ^2 test

Problems such as loop closure or cluttered environments difficult the measurements of the joint compatibility between nodes, requiring of techniques to determine the best solution to data association. For this reason, the χ^2 test is done for each pair of nodes to determine the independence and variance estimation. The probability density function (pdf) of this distribution is

[6] N is supposed to be the same for both clouds simply because the sensor is the same

$$f(x;\, k) = \begin{cases} \frac{1}{2^{k/2}\Gamma(k/2)}\, x^{k/2-1} e^{-x/2}, & x \geq 0; \\ 0, & \text{otherwise.} \end{cases} \tag{3.29}$$

where $\Gamma(k/2)$ denotes the *Gamma function*. So, the like-hood between two nodes will be done using this distribution. The P-value is the probability of observing a test statistic at least as extreme in a χ^2 distribution. A P-value of 0.05 or less is usually regarded as *statistically significant*.

3.6.2 Relaxation on a mesh to localize the robot and build the map

Minimizing the error in the constraint network is the way maps are relaxed. To understand the concept of relaxing a graph, the spring–mass example is normally explained: In this view, the nodes are regarded as masses and the constraints as springs connected to the masses. The minimal energy configuration of the springs and masses describes a solution to the mapping problem [4]. Figure 3.16 shows an example of an uncorrected constraint network and the corresponding corrected one.

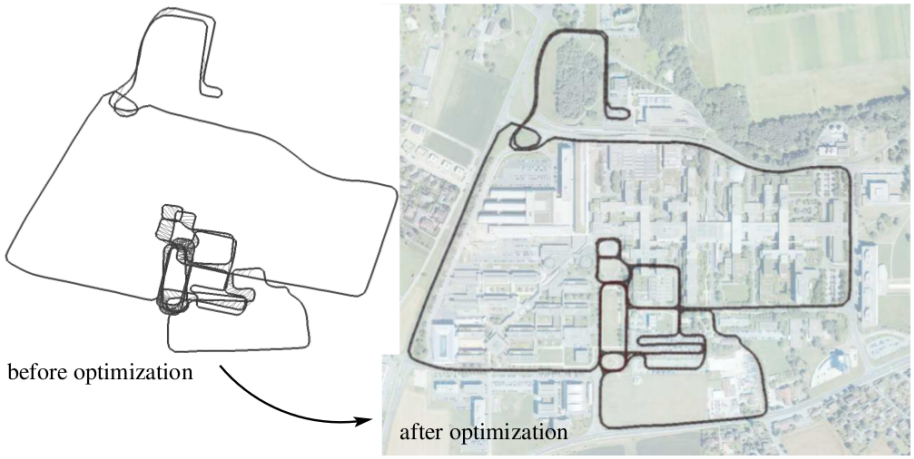

Figure 3.16: *Constraint network.* Example of an constraint network corresponding to a raw dataset (before optimization) and the corresponding corrected one (after optimization)

Let's consider a pair of masses (or rigid bodies) p and q that are connected by a spring. If x_p represents the pose of p from an arbitrary global coordinate system and x_q the pose in the same global coordinate system. Let z_p contains the pose of p in a *local coordinate system* (LCS) and z_q denote the pose of the object in LCS of their respective bodies. The energy U_{pq} of this spring is given by

$$U_{pq} = \frac{1}{2} k_{pq} (\xi_{x_p} z_p - \xi_{x_q} z_q)^2 \tag{3.30}$$

where ξ is a coordinate transform operator that maps points from local to global coordinate systems. The spring constant is typically set to $1/\sigma^2$, where σ is the uncertainty in the measurement represented by this spring. Therefore, if exist a mesh with several rigid bodies, the total energy of the mesh can be calculated as

$$U = \frac{1}{2} \sum_{pq} U_{pq} \tag{3.31}$$

The goal here is to find out the set of poses $\{x_p, x_q, ..\}$ that minimizes this energy. In a real scenario, springs and masses will get into equilibrium immediately if the system is released. It is possible to emulate the physical equations of this system and apply them to this simile. This is done in two steps as follows. It is repeated until the global energy reach a minimum threshold (ideally this value might be zero)

1. For each body p, compute the total force acting on it:

$$F_p = \sum_q F_{pq} = -\sum_q \nabla_{x_p} U_{pq} \tag{3.32}$$

   where F_{pq} is the force generated by the spring U_{pq} and ∇_{x_p} denotes the gradient with respect to x_p.

2. For each body p refresh the pose x_p using the equation

$$x_p \leftarrow x_p + \frac{1}{2} \Delta t^2 F_p \tag{3.33}$$

   where Δt is a time constant that regulates the rate of convergence.

3.6.3 Hierarchical optimization solution to the GraphSLAM

In this project, the optimization has been done using *HOG-Man* optimizer (Hierarchical Optimization on Manifolds[7] proposed initially proposed by [5]). It splits up the problem in two different slides:

1. Constraints need to be extracted from sensor data to construct the abstract graph representation. This is referred to as the SLAM **front-end**.

2. Given the constraints, the most likely configuration of the poses as well as the pose uncertainty need to be computed. This is referred to as the SLAM **back-end**.

3.6.3.1 Front-end problem

The first problem can be addressed solving the data association problem: detecting if the robot is perceiving the same information as the measurements in previous observations. It is not necessary to cover the whole graph each iteration to perform the exploration, the search area can be resized based on the current uncertainty estimate of the robot. For this, an optimization based on Gauss-Newton with sparse Cholesky factorization is used on the previously explained Equation 2.16, and χ^2 test is applied to perform the comparisons.

The method proposed by [5] not only use an optimized versio of Gauss-Newton with sparse Cholesky factorization but also linearize on a Manifold. The reason why this strategy is done falls on the fact that normally the space of parameters X_t is considered to be Euclidean, which is not valid for SLAM (this may lead to sub-optimal solutions). The explanation of the linearization on a Manifold is out of the limits of this project, but can be further read in detail in [5].

[7] A manifold is a topological space that can be assumed as Euclidean space of a specific dimension in a small scale

3.6.3.2 Back-end problem

The second problem is the optimization engine. That one is solved using a hierarchical approach where in each step instead of repeatedly optimizing all nodes of the graph, it computes a solution to a simplified problem. The levels in the hierarchy represent the levels of abstraction. For instance, if a high level node is modified, only those nodes of the middle layer which are connected to that node have to be updated. With this method, large computational cost can be saved. Figure 3.17 represents the same graph at different levels of abstraction.

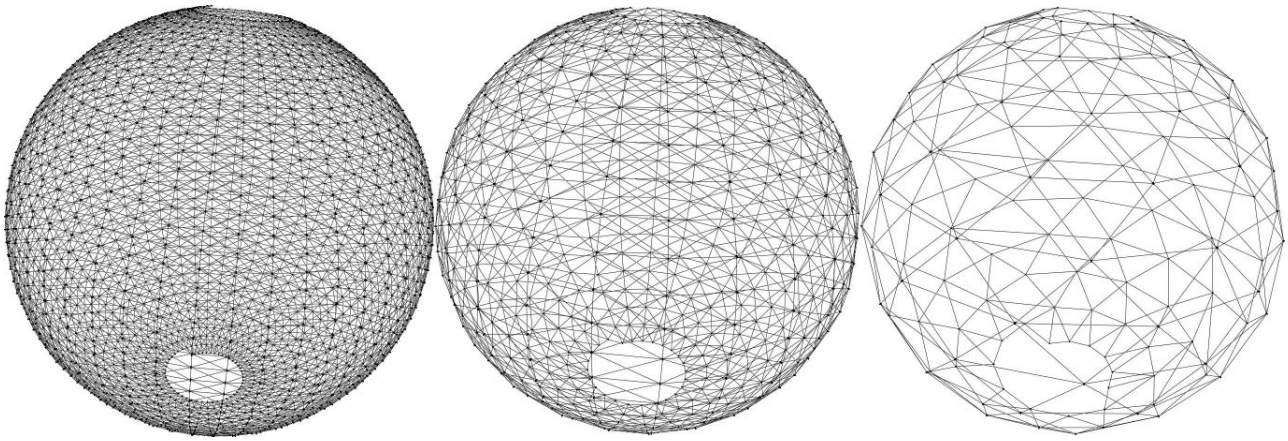

Figure 3.17: *Hierarchical levels in a graph.* Representation of different levels of abstraction of a hierarchical graph. Left sphere represents the graph at level $k = 0$ while the right sphere contains the nodes of the last layer $k = 2$

Mathematical definition

The idea of a hierarchical pose-graph claims on represent different levels of abstraction: each level is a pose-graph with connections modeling correspondences between levels of abstraction. The lowest level represents the original input in $k = 0$. Each node at level $k > 0$ is a graph at level $k - 1$. The number of parameters describing the environment decreases with the level of abstraction. Therefore, the highest level (k = K) represents the lowest quality but at the same time is the faster to be optimized. Thus,

if there exist K levels, $\mathcal{G}^{[k]}$ represents the graph at level k. Graph $\mathcal{G}^{[k]}$ consist of a group of nodes $\{x_i^{[k]}\}$ and a set of edges between them $\{e_{ij}^{[k]}\}$.

Each node $x_i^{[k]}$ at level k is linked to

i) A "representative" node $x_i^{[k-1]}$ at level $k-1$

ii) A connected sub-graph $\mathcal{G}_i^{[k-1]}$ at level $k-1$

There will be an edge $e_{ij}^{[k]}$ between $x_i^{[k]}$ and $x_j^{[k]}$ at level $k > 0$ if two sub-graphs $\mathcal{G}_i^{[k-1]}$ and $\mathcal{G}_j^{[k-1]}$ are already linked. Therefore, lower level graphs are split in local maps $\{\mathcal{G}_i^{[k-1]}\}$ creating high level graphs (see figure 3.18). Each local graph is represented by a node at the higher level. Edges between nodes contains the information about the relations between local maps.

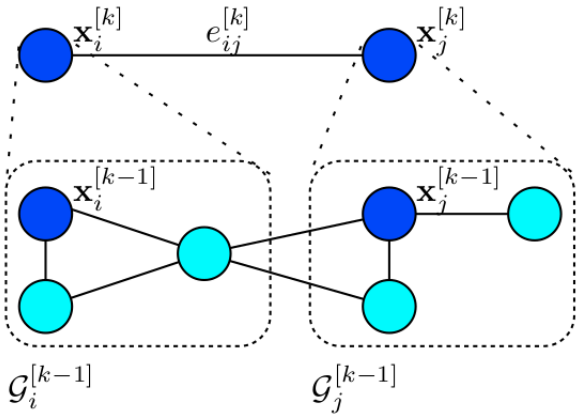

Figure 3.18: *Hierarchical Graph.* Definition and relation between sub-graphs in different levels of abstraction and edges linking them

Construction of the graph

The rule followed to build graph $\mathcal{G}^{[k]}$ from the graph $\mathcal{G}^{[k-1]}$ is based on the distance on the graph. That means that nodes are grouped at level $k-1$ when they are closer than a certain threshold τ. Those groups are named $\{\mathcal{G}_i^{[k-1]}\}$ and a representative node $x_i^{[k-1]}$ has to be selected. This representative becomes the node x at level k. An edge will

be attached between $x_i^{[k]}$ and $x_j^{[k]}$ at level k if the corresponding sub-graphs are already connected. This edge will contain the information included in all edges of $\mathcal{G}_i^{[k-1]}$ and $\mathcal{G}_j^{[k-1]}$ as well as the edges connecting both.

Propagating the hierarchy to upper levels

As soon as the robots moves and a new node is added to the bottom level, the new node is added to a previously created group or it becomes the representative of a new one at level $k = 0$. This process is done recursively until there are no need for new groups.

Refreshing the graph

When the hierarchical pose-graph is modified, an optimization is done from the top level. Only if there are sufficient changes in one level, it is propagated to lower levels. Those changes can be seen comparing the distance between nodes $x_i^{[k]}$ and its representative in level $k-1$, $x_i^{[k-1]}$ as they are supposed to be solid rigid [8].

[8]Due to both nodes represent the same pose, if an upper layer change, the restriction $x_i^{[k]} = x_i^{[k-1]}$ has to be applied to update the lower layer and validates the equation

Chapter 4

Development

4.1 Software

This software has been developed, with some technical modifications, based on a GraphSLAM application offered by [36]. All the source is implemented in ROS platform, making easy its integration with other parts of the robot: odometry, lasers, cameras, GPS, IMU's or any other source of information. In this section, it is explained and analyzed the different parts of the application, how are they connected among them and how the whole system works.

4.1.1 Introduction to ROS platform

ROS (Robot Operative System [1]) was originally developed in 2007 by the Stanford AI Lab with the name *switch yard*. After a while, in 2009, the development was assigned to Willow Garage who has continue developing and updating the libraries for free. It can be downloaded and modified in `https://code.ros.org/gf/project/ros` (last visit, Oct. 2011).

As their own creators say, ROS can be defined as:

[1] Also, the acronym stands for Robot Open Software

ROS (Robot Operating System) provides libraries and tools to help software developers create robot applications. It provides hardware abstraction, device drivers, libraries, visualizers, message-passing, package management, and more under an open source, BSD license.

ROS can be resumed in three fundamental things:

1. Meta- Operative System
2. Robotic Software Framework
3. Distributed Robotic Architecture

It covers most of the low-level programming (threads, communication protocols, semaphores, memory maps, etc.) giving the opportunity to their users to develop fast and specific code (see figure 4.1) with a community in steady growth with 100+ packages, approximately 200 stacks and 50+ updated repositories around the world.

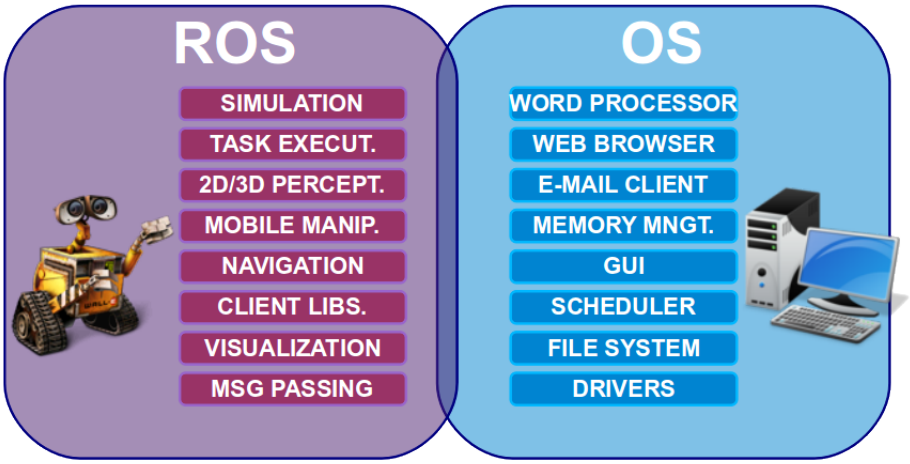

Figure 4.1: *ROS task Vs OS task.* ROS is designed to work at low level in harmony with the OS. Low level operations are handled by ROS to let users think only in high-level applications

One of the most important features of ROS is the amplitude of their researches. It covers almost every branch in robotics: path planning, object recognition, machine

learning, perception, localization, SLAM, tele-operation, manipulation, etc (some snapshots are shown in figure 4.2) and it is being used in 50+ robots nowadays.

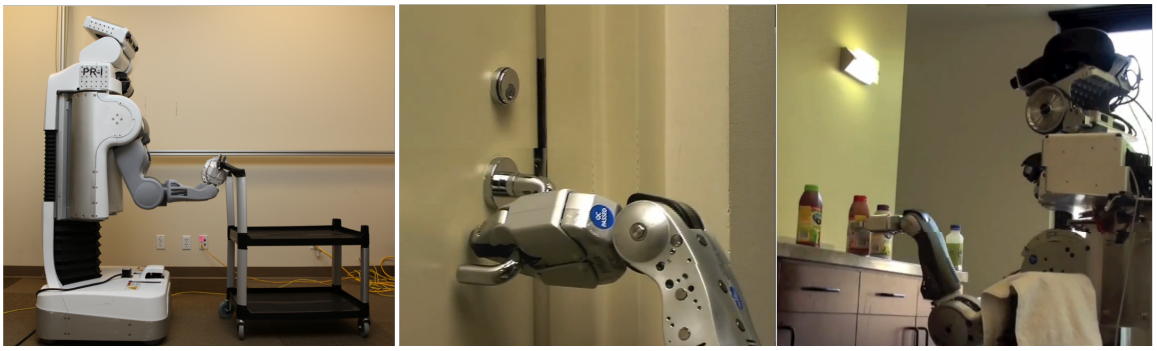

Figure 4.2: *Examples of PR2 robot using ROS* From left to right: PR2 moving a trolley, PR2 opening a door and PR2 recognizing and grasping a bottle in a kitchen.

4.1.1.1 Advantages and disadvantages

As any solution, ROS offers a great variety of advantages but also leaks in some points. The following list highlights its pros and finds the cons.

☺ Advantages

- Focus the problem on high level problems, reducing the low-level processing
- Wide compatibility with "standard" robotics structures (laser scans, images, location messages, etc.)
- Growth learning curve: approximately six months are needed to understand the structure
- Easy to share and learn: Subversions and Git servers allow to share and improve algorithms though any open source platform
- Research in robotics community accept and support it!

☹ Disadvantages

- Its philosophy is to reach every robot and to be able to control it, and this means a great dispersion
- It requires some experience in low level programming to understand how to adapt the code to the platform
- Focused on Unix OS, it does not support Windows/Mac (nowadays)

4.1.1.2 Internal structure

ROS is a distributed system which allows to execute portions of code in different machines distributed in a network and connected via TCP/IP. Furthermore, this connections are designed as a *Peer-to-peer* configuration, allowing the system to distribute the payload and processing task. The platform accepts different programming languages such as C, C++, Phyton, LISP, Octave. Furthermore, some of the most famous libraries are easily integrated:

- **PCL**: Library specialized in 3D cloud management: filtering, transformations, clustering or meshing.
- **OpenCV**: Library specialized in image processing: image filtering, pattern recognition, machine learning or features extraction.
- **OpenRAVE**: Library specialized in developing, testing and deploying motion planning algorithms.
- **Player**: Library specialized in robot control interface and simulator.

Nodes

Nodes are each sub-process or task that the robot is able to do: odometry, planning, object recognition, etc. That is, each node is able to publish some information based on other nodes. For instance (see figure 4.3), if a node is in charge of recognize objects in

a table, probably will need firstly a node which acquire 3D cloud points, another that get the support plane, then another one to cluster the objects in the table and finally a last one saying which object is seeing[2].

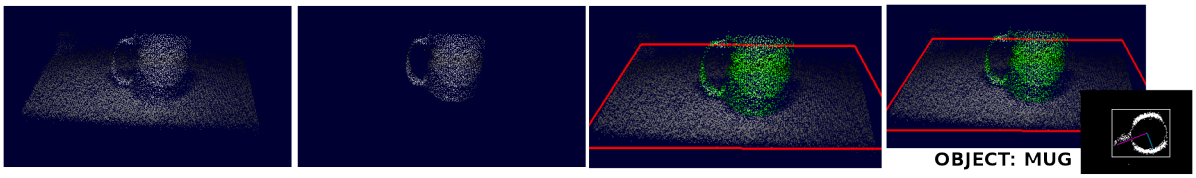

Figure 4.3: *Explanation of how nodes work in ROS.* From left to right: first node is in charge of 3D Point cloud extraction from the camera. Afterwards second node extracts the supporting plane (table). Then third node segments the objects laying on the table. Finally, last node recognize the object successfully as a mug.

ROS connects and disconnects nodes in real time. Those messages are transported by a main node called **roscore**. Any node can get subscribed to any other node that publish information whenever it needs it. The only aspect which must be reminded is how the information is passed through them. That is in charge of messages.

Messages

Two nodes can communicate by means of messages. There exist different structures for this messages depending on the information to be transmitted. For instance, if a camera node wants to publish 3D cloud point information gathered from the Kinect camera, it will construct a RGB-D message which contains, among other fields:

- Red, green and blue color channels [`unsigned char (uint8_t)`]
- Height and width of the color frame [`unsigned char (uint8_t)`]
- Depth map [`float `]
- Height and width of the depth frame [`unsigned char (uint8_t)`]

[2]This example has been extracted from a previous work of the author. More information in [37]

- Time stamp of the frame [`ros::Time`]
- Extrinsic parameters matrix [`3 x 3 float`]
- Intrinsic parameters matrix [`3 x 3 float`]
- Distortion factors [`1 x 3 float`]

Topics

The vehicle to transmit the information between nodes are the messages, but the way a node inform that is publishing a new data is with topics. Therefore, nodes work with a *publish-subscriber* messaging pattern. A node publish an specific topic and any node interested on this information will subscribe to it.

4.1.2 Application components

The ROS application developed named GraphSLAM is divided on different modules. Each one has a specific task that will be described in following table 4.1. The next figure 4.4 represents how modules are inter-connected.

4.1.3 Application GUI

Next figure 4.5 shows the GUI's (Graphical User Interface) for the application. Because the program has been done in Qt all the graphical objects are standard components (`QButton`,`QMessageBox`, `QTextEdit`, etc. [3]) It has been tested in **Ubuntu Linux x86_64 (2.6.35-30)** in an **Intel(R) Core(TM)2 Duo CPU P8700@2.53GHz 4GB RAM** and a nVidia graphics card **GeForce GT 240M**.

[3]More info at `http://doc.qt.nokia.com/latest/qtgui.html`

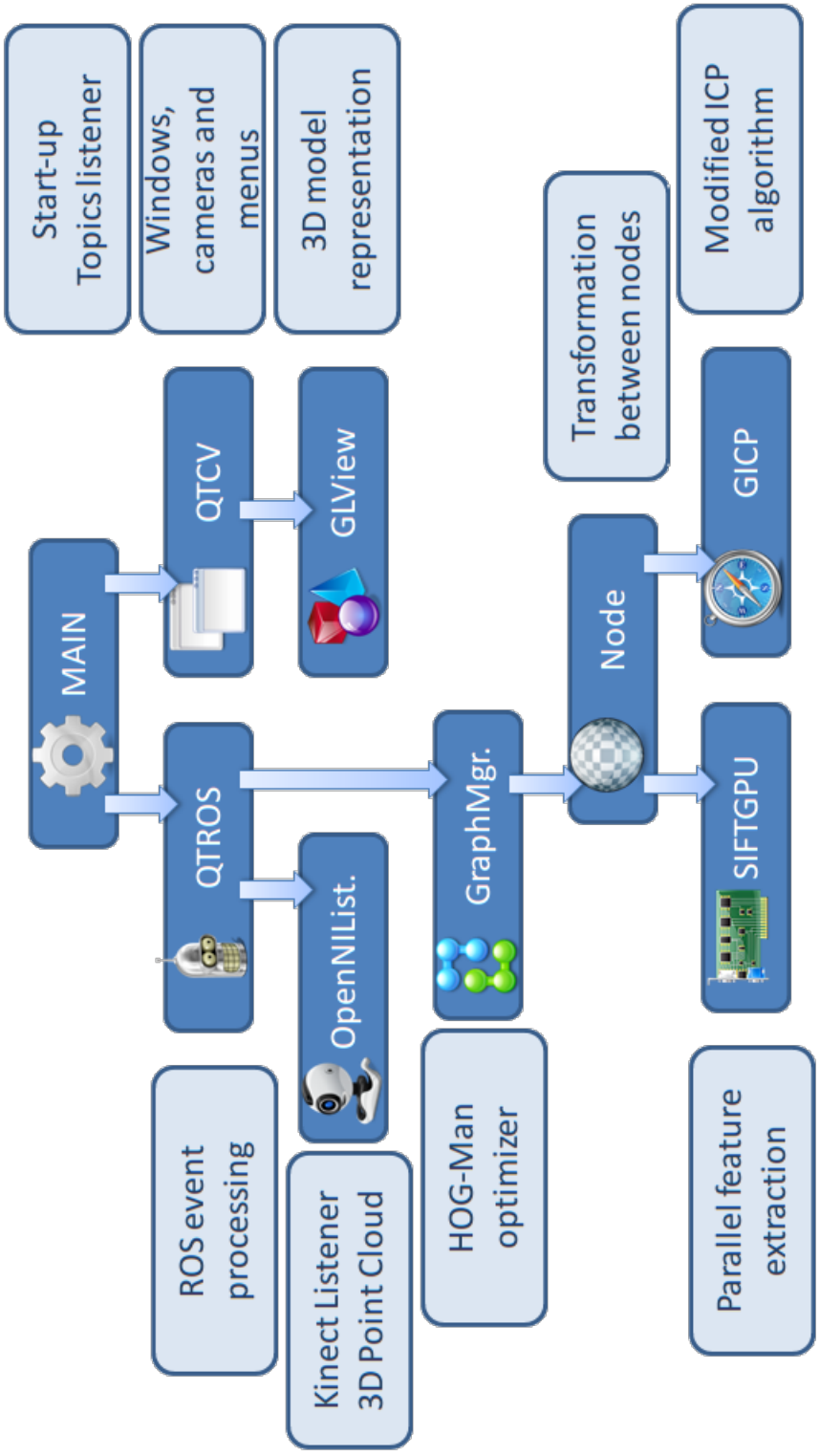

Figure 4.4: *Flowchart of the application.* Connection between the modules and representation of how information is tranfered along the application

Module Name	Description
GICP	Module in charge of obtaining the refine transformation of the cloud point, that is, from two RGB-D messages, it returns the best rotation and displacement.
GLViewer	Module in charge of displaying the 3D model in OpenGL. Points clouds are represented using single points with color.
GraphManager	Module in charge of applying the HOG-man transformation between nodes and refresh globally the whole graph on each iteration.
Main	Module in charge of starting the application and receiving the neccessary topics and establishing the topics which are going to be published.
Node	Module in charge of holding the data for one graph node and providing functionality to compute relative transformations to other nodes.
Parameters Server	Module in charge of storing the default values of the application, global constants and manage all publishing topics.
OpenNIListener	Module in charge of most of the ROS-based communication and Kinect feed synchorinzation.
QTCV	Module in charge of the GUI of the application: menus, messages, file management, frames and buttons.
QTROS	Module in charge of setting up a thread for ROS event processing: external events, etc.
SiftGPU	Module in charge of computing the SIFT feature extraction in GPU (only valid for *nVidia CUDA* technology cards.)

Table 4.1: *Modules of the application* Description of the different parts of the GraphSLAM algorithm. All of them are implemented for ROS platform.

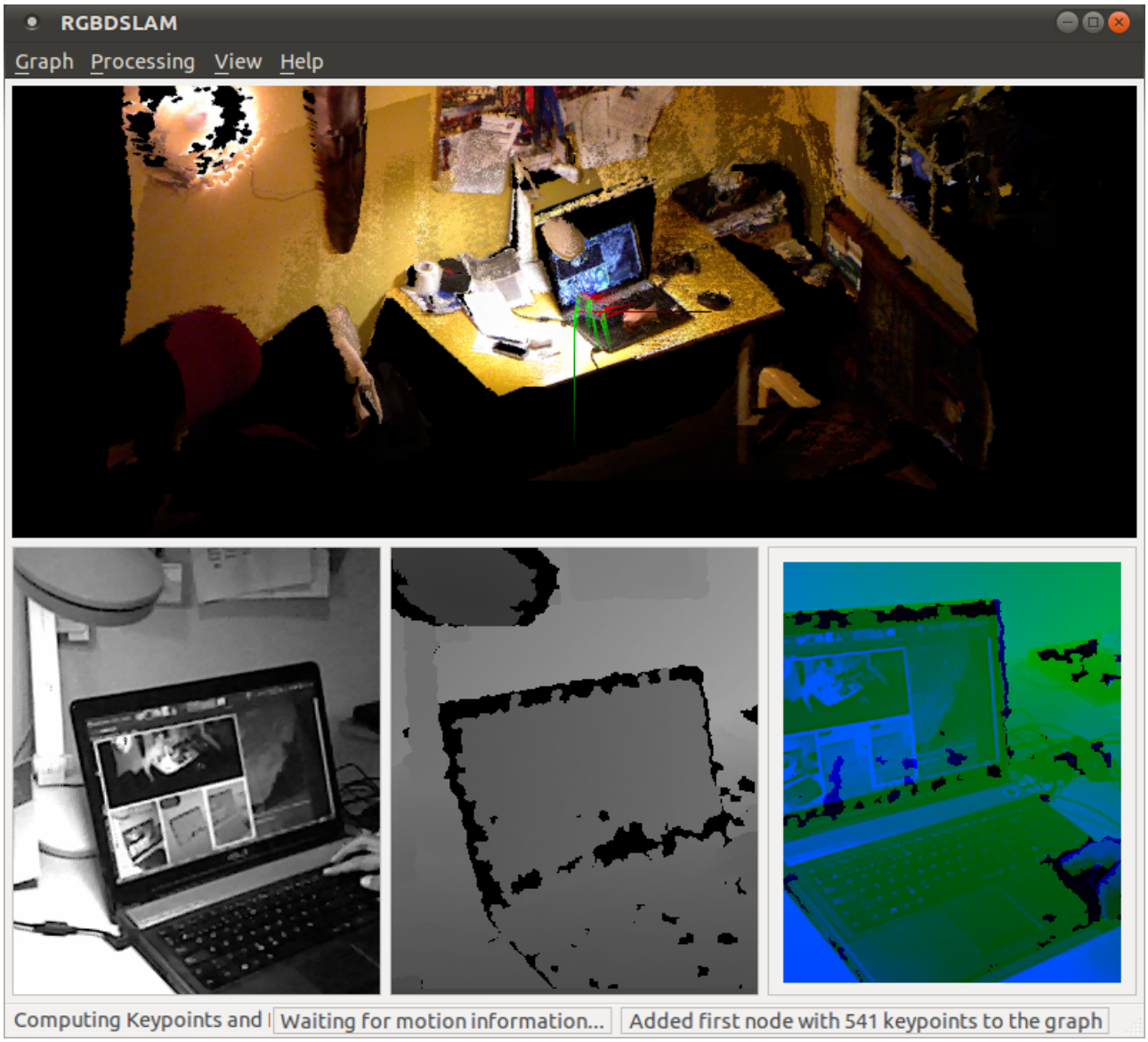

Figure 4.5: *Graphical User Interface* The main window is divided on two parts. The openGL view with the 3D processed cloud on the top and the three views (grayscale frames, depht frames, feature matching frames) for processing tasks.

4.2 Hardware

As it has been mentioned before, this project has been developed using Manfred robot as a platform. It is presented in the next section. The perception system used for the demos and trials is a Kinect Camera (instead of the presented RGB-D system) because of several outward things:

- Kinect camera is cheaper than the PMD-Tech. Its usage is less risky and gives an option to be broken
- Kinect camera integrates color on the information channel. That means that no transformation is needed before playing with the data
- Kinect depth resolution is lower than the PMD-Tech one, but spatial resolution is greater. This represents a disadvantage for small workspace applications such as object recognition, object modeling or object grasping but offer some advantages in large environments such as localization or SLAM
- The weight of the PMD-Tech camera is greater than the Kinect's. That is an important feature in terms of integration in the robot structure.
- Kinect camera is commonly used between research groups, and it has received lot of support during last months.

4.2.1 Manfred manipulator

Manfred is an advance mobile manipulator designed and developed entirely in Robotics Lab (UC3M). It has been used as one of the research lines of the group, focused on manipulation, navigation, in-hand movements, grasping, perception and control. The most important features and sensors of this robot are stated below:

- 6 DOF carbon fiber arm
- Arm weight: 18 Kg.

- Arm length: 1205mm
- Maximum Load: 4.5 Kg
- Control: PMAC 8 axis (2 base + 6 arm)
- Harmonic drive gearings
- Brush-less motors (C.C)
- Torque sensor
- Differential wheeled base
- Motorized SICK
- Motorized Hokuyo
- RGB-D camera

and there is a large list of task it is able to perform:

- Switch the light on/off
- Open doors
- Local planification based on Voronoi Diagrams (updates global planning)
- Cooperation with humans
- SLAM using laser data
- Machine Learning for grasping

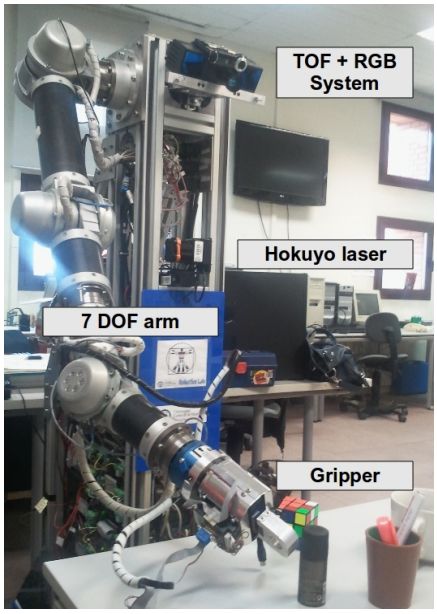

Figure 4.6: *Hardware specifications.* Mobile manipulator Manfred has been used for the presented research. Some of its connected devices are a RGB-D camera for perception, lasers for navigation and a gripper for manipulation.

4.2.1.1 Manipulation skills

His development trajectory has been very interesting. Robotics Lab started developing a first version built using commercial elements (the robot was named *Otilio*). Afterwards, it was developed the mobile manipulator named Manfred-1 and also the lightweight carbon fibber robot `UC3M-LWR1`. Last version is called Manfred-2 maintaining the original arm `UC3M-LWR1`.

On each one of the designs, it has been looked for a new progresses and new capacities to make the robot more versatile. In this sense, nowadays it looks for an evolution for `UC3M-LWR1` into a more anthropomorphic capacities, that is 7 DOF, a weight between 10-11 Kg and 4-5 Kg. load capacity. Also, as a new feature it would have attached an anthropomorphic hand. Theoretical studies and simulations performed until this moment revealed that this suggested arm is achievable.

4.2.1.2 Planning based on sensors

In planning terms, during last years it has been developed researching focused on methods based on sensors from the Level Set Methods proposed by [38]. Based on the Fast Marching technique, there had been developed the methods VFM and FM2 for planning based on trajectories's sensors in 2D, 2+1/2 D and 3D environments. In the following figure 4.7, there are three examples of this method. In the first one, the dynamic re-planning of trajectories in indoor 2D environments, where it can be shown how trajectory to the target point changes as long as new obstacles are found. Therefore, new obstacles that initially were not considered in an a priori static environment map are taken into account.

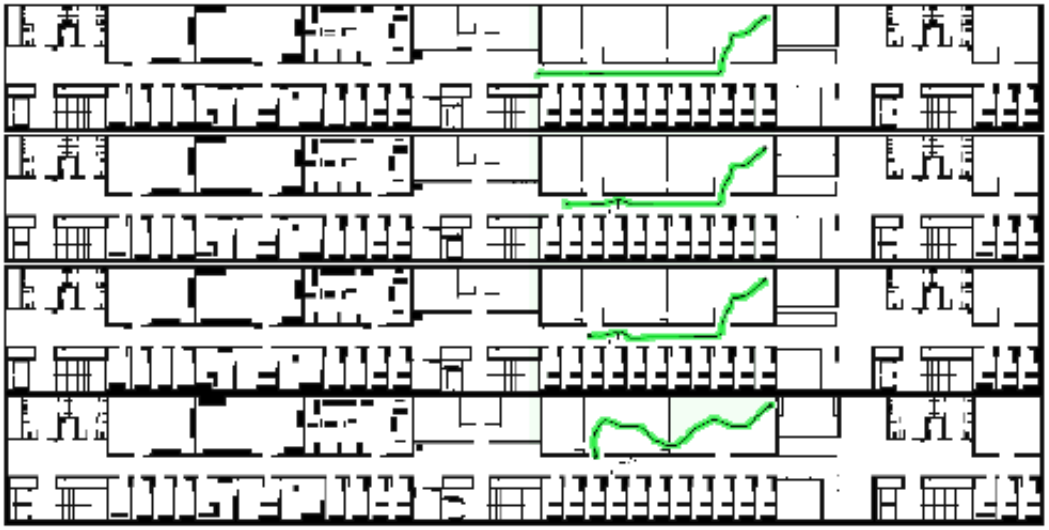

Figure 4.7: *Global localization* VFM method for dynamic calculation of trajectories in 2D environments

On the second example, a recent version of the planning VFM method for outdoor environments is shown. Here trajectories are calculated based on a 2+1/2 D elevation terrain map.

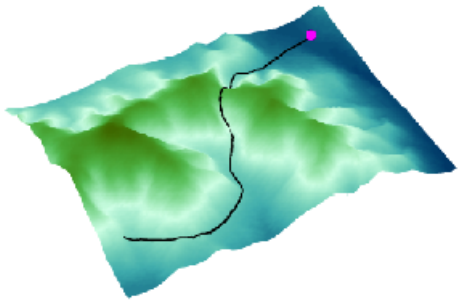

Figure 4.8: *Global localization* VFM algorithm used for trajectories calculation in 3D outdoor maps.

4.2.1.3 Evolutionary-based methods applied to optimization and learning

When applying different evolutionary-based methods in mobile manipulators, there have been developed evolutionary-based methods not only for the global location problem but also for SLAM in 2D environments based on information obtained with 2D laser telemetry. It has been developed also an exploration method, learning and autonomous location in a 2D environment. (Observe in the next figure 4.9 how the mobile manipulator explores its environment, re-plans and modify its trajectory dynamically as long as it moves along the map environment maintaining its location).

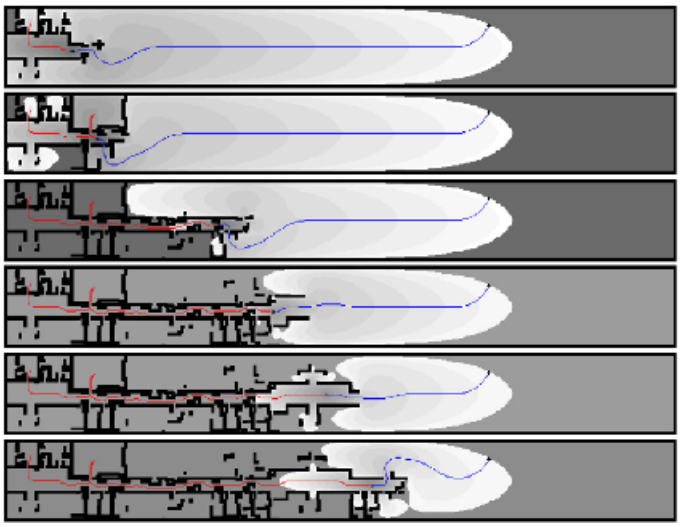

Figure 4.9: *Autonomous exploration* Autonomous exploration and environment learning for indoor applications.

Chapter 5

Results

During this chapter, some of the results of the complete algorithm will be spotted. The experiments have been classified into four different categories: speed tests, feature extraction tests, pose estimation tests, global optimization tests and dynamic environment tests.

5.1 Experiment 1: Speed tests

5.1.1 Description of the experiment

The goal of this experiment is to understand the computer resources that demand each part of the process and compare them. Figure 5.1 represents the time-line of a graph experiment in a room. The experiment consists on 114 nodes. For each node, the time required by the camera callback, features extraction and matching, pose estimation and graph optimization has been logged. With this information, it is expected to study which parts of the process create bottlenecks and how to fix them.

Furthermore, this experiment is very handy and useful to see the relationship between the number of nodes of the system and the speed of the process.

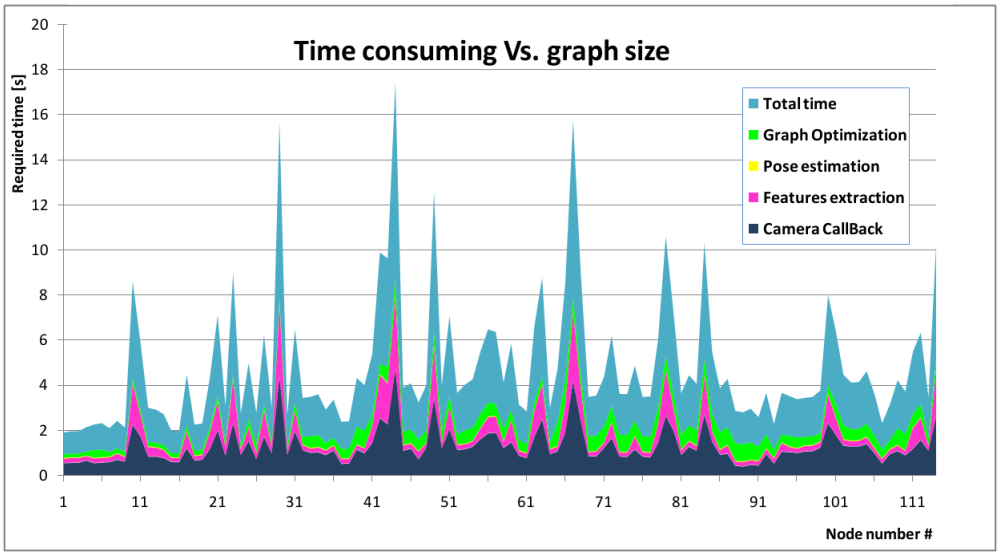

Figure 5.1: *Time consuming vs graph size* Time for each part of the process for a set of 114 nodes. Also, this graph shows the relation between the number of nodes of the graph and the total consuming time

5.1.2 Results

As it can be seen in the previous figure, most of the time is consumed by the camera callback. This makes a lot of sense, the camera is connected to the computer via an USB port which is reducing the flow of data from the camera to the application. However, the problem can be attached not only to the physical connection but also to the driver that manages the information source and receiver.

In figure 5.2 it is plotted, in average, the time required of each part of the process. Clearly, Camera callback requires the most time (55%), followed by HOG-man (23%) which makes most sense because of the complex algorithm and sparse optimization process and then features extraction (20%) and finally pose estimation (2%) which is quite fast due to the Flann KDD Tree optimized for ROS.

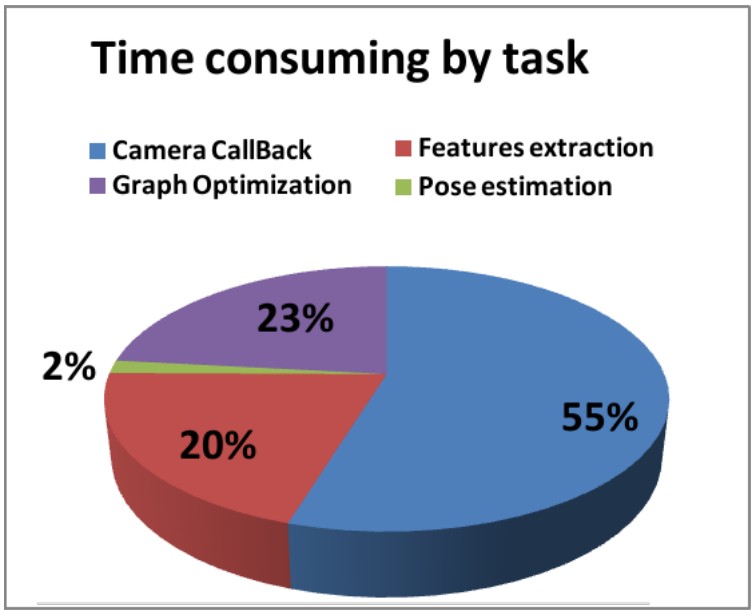

Figure 5.2: *Time consuming by task* Time required in average for each task. Camera callback requires the most time (55%), followed by HOG-man (23%) and then features extraction (20%) and finally pose estimation (2%)

5.2 Experiment 2: Feature detector tests

5.2.1 Description of the experiment

The goal of this experiment is to compare the performance of the complete system using different feature detectors. The most common feature extractors are SIFT, SURF, FAST, MSER, GFTT and STAR. Each of them extract different kind of points according to their specifications. Furthermore, the number of points extracted by each method varies.

The minimum number of matching points required to evaluate if there is a new node in the graph is **16** to make the process more accurate and restrictive [1]. Next table 5.1 gives the average number of features detected by node.

[1] As it is widely demonstrated, with 3 pairs of matched points is possible to determine the inverse transform of the camera. The more number of pairs, the better estimation.

Feature extractor	Number of features (avg)	Time [s] (avg)	Detector speed [feat./s]
STAR	34.9808	0.02822	1239.392705
FAST	84.657	0.0943	897.6670201
GFTT	31.51	0.04924	639.9268887
MSER	53.98	0.1241	434.9717969
SURF	199.789	0.6736	296.5988717
SIFT	113	1.5022	75.22189349

Table 5.1: *Feature extractor speed.* Study of the different feature extractors and their values

5.2.2 Results

In a first impression, it would seem better to choose STAR feature extractor due to the number of features per second is able to extract. This speed data can be confused if it is not taken with care. As in any others on-line problems, the time is a key variable. In this case, analyzing all the extractors, it is important not only the speed but the quality of the features. This quality is measured with the number of features per node (first column of table 5.1).

In this way, depending on the problem, the user will take care of the number of features per frame in order to estimate a consistent pose transform choosing then SURF or SIFT feature extractor. Or, if it is primordial to perform the problem as fast as possible with a lack of accuracy, FAST or MSER feature extractors must be chosen (STAR or GFTT's number features might be insufficient).

5.2.3 Description of the experiment

To get a more precise discussion of the previous experiment, the statistical χ^2 has been compared for all the feature extractors in the same environment over similar conditions. Figure 5.3 describes the evolution of χ^2 with the growth of the graph.

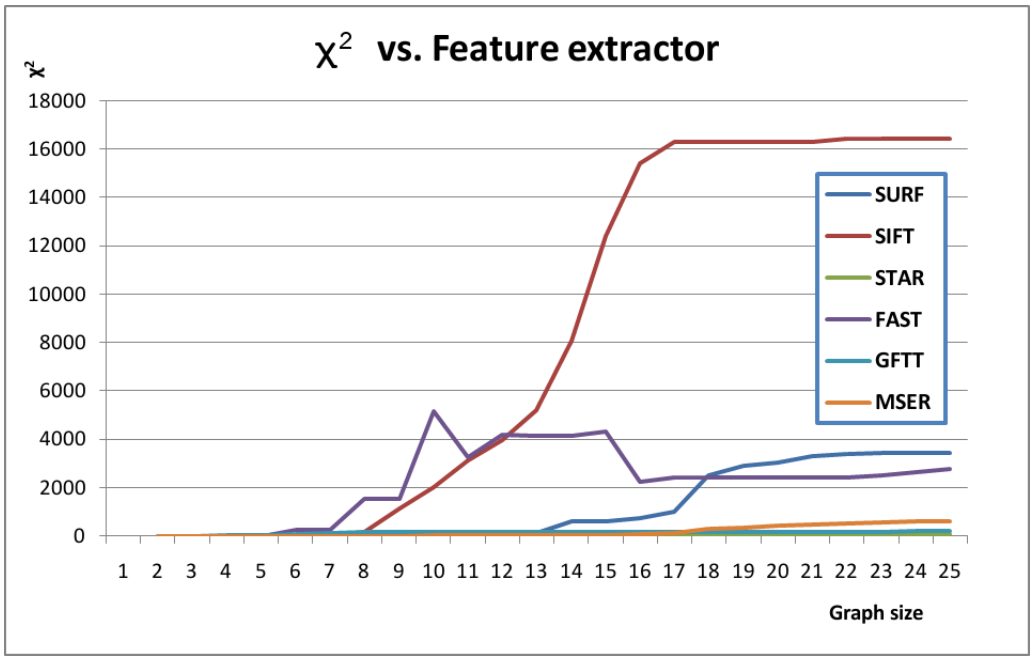

Figure 5.3: *Evolution of* χ^2 Representation of the evolution of χ^2 with the increase of the graph nodes. SIFT gives the best performance followed by SURF

5.2.4 Results

The previous figure 5.3 gives SIFT as a winner after the eighth node in the graph. The grow slope is very high due to the number of features this extractor retrieves. Taking into account both experiments, SIFT might be the best feature extractor followed by SURF (taking into account not only the graph performance but also the extraction speed).

5.3 Experiment 2: Pose estimation tests

5.3.1 Description of the experiment

With this experiment, the pose estimation and refinement algorithm will be studied. For this reason, some of the parameters of the algorithm has been measured. In particular, it has been payed attention to four:

- **Number of iterations:** How many iterations have been done before obtaining a valid pose.
- **Percentage of inliers:** Proportion of inliers over the whole number of matching points.
- **Quantity of inliers:** Size of the complete matching points vector.
- **Error:** Error committed after the matching (euclidean distance between clouds)

All this experiment has been done using SURF feature extractor. Next figure 5.4 shows the error explained above. As it is shown, the error remains constant with the number of nodes in the graph. That is, the size of the graph does not alter the refinement and pose estimation of the node.

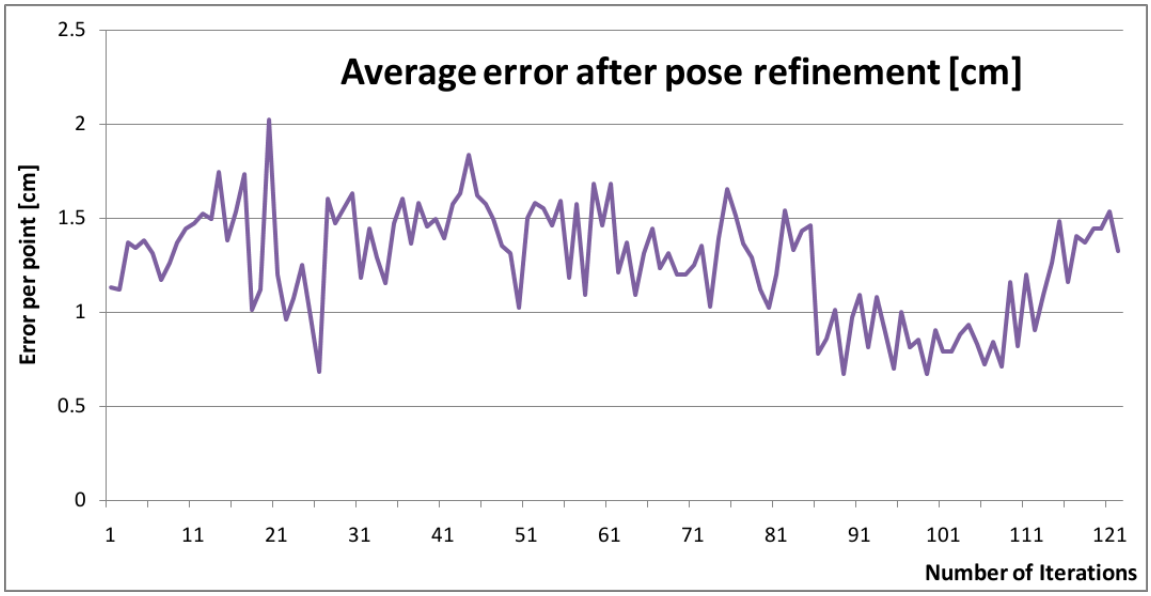

Figure 5.4: *Refinement error vs size of the graph* The error in the pose refinement remains constant with the size of the graph. The error value moves between 0.7 and 1.7 cm, that is, the euclidean distance error.

The following table 5.2 shows the overall average values for the presented variables in the ICP algorithm.

# iterations	% inliers	# inliers	Error [cm]
82.93	50.52%	167.06	1.26

Table 5.2: *Iterative Closest Point variables*. Analysis of the number of iterations, amount of inliers and error committed by the pose refinement in the room experiment.

5.3.2 Results

The error given by ICP moves between 0.7 and 1.7 cm. This data maintains constant with the number of graph nodes, and explains the kind of environment that has been mapped. During the first 10 nodes it is almost constant. Then the camera moves into a dark or untextured zone (frame 20) and then come back to a bright or previous zone (frame 26). Exactly the same can explains the tendency during the frames 80-110 where the error diminishes due to the same factors.

With relation with table 5.2, it resumes the variables of the ICP process. The number of inliers may appear slightly low, but it must be taken into account the fact that the environment light conditions can be bad and textures can be flat in some parts of the room scenario.

5.4 Experiment 3: Loop Closure

5.4.1 Description of the experiment

In this experiment, some environments have been mapped and the top view of the room are plotted. There is not a feasible way of measuring the performance of the algorithm with this information but the reader can, at least, understand the quality of the mapping technique and the loop closure of the map.

Figure 5.5 represents the top view of some of the scenarios the algorithm has been applied on. The first map correspond to the room experiment, the second one to the

Figure 5.5: *Top view of some scenarios* The top view of the scenes represent the quality of the matching and graph optimization.

kitchen experiment while the third one represents the office experiment and the last one the laboratory.

5.4.2 Results

As it can be seen in figure 5.5, the walls match up great and there are not big incoherences in the scans. However, as it can be seen in figure 5.6, sometimes the nodes are not perfectly fitted.

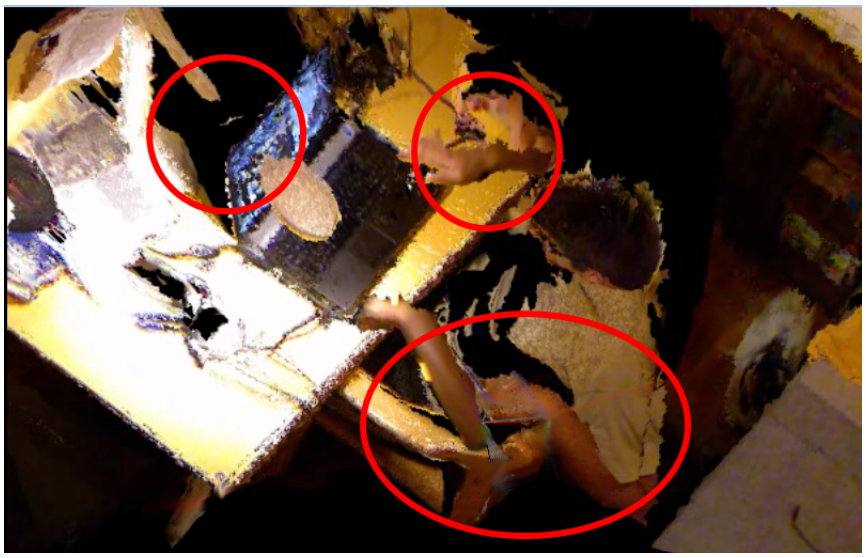

Figure 5.6: *Errors in matching* Red marks point out some of the problems found during mapping. The lack of features or the movement of the objects during the mapping create wrong matches.

There are cases where the GraphSLAM algorithm does not work properly. Some of the possible reasons of this failures are:

- There are not enough features in the area
- Dynamic objects are not supported by the algorithm. There is not a strategy for that.
- Thin objects are not taken into account by the ICP matching

- Similar objects can be wrongly matched mistaken

In figure 5.7 another scenario is shown. This time, it was selected on purpose a place with too many similar textures in order to find out the problems stated before. As it can be seen, the close loop fails due to the lack of features and significant nodes.

Figure 5.7: *Errors in matching* Top view of a scenario with similar walls and roof. The loop closure fails mostly because of the lack of new information in the graph.

Finally, last figure 5.8 shows the top view of a room where the loop closure is not well done. After the analysis of the environment, the reason for this problem was found in the texture of the zone. Due to the flat color of the walls, the number of

matching features in this area decreases (low keypoints), obtaining a low pose (R, t) estimation. In this circumstances, the number of iterations of the ICP algorithm is over exceeded and the final solution is poor.

Figure 5.8: *Errors in matching* Top view of a scenario with a failure loop closure. Flat textures in the walls reduce the number of keypoints and therefore the (R, t) initial estimation

5.5 Some examples

To sum up the actual project, next figures show some of the results obtained during the experiments. In those figures, the reader can see different environments and how the location and mapping are performed taking into account the problems experienced before.

Figure 5.9: *Experiment: room* Real image of the room scenario and snapshots from different points of view of the mapped room.

Figure 5.10: *Experiment: office* Real image of the office 1 scenario and snapshots from different points of view of the mapped office.

Figure 5.11: *Experiment: office2* Real image of the office 2 scenario and snapshots from different points of view of the mapped office.

Figure 5.12: *Experiment: office3* Real image of the office 3 scenario and snapshots from different points of view of the mapped office.

Chapter 6

Conclusions

In this project a complete Graph-SLAM architecture is presented. It has been programmed in C++ and uses ROS platform to get integrated with other sensors and devices. The application has been included in Manfred, a mobile manipulator robot designed and developed in Robotics Lab. A TOF camera has been used to acquire the information from the environment: a 3D color point cloud of 307200 elements at 15 frames per second.

Next list resumes the most important features and the goals achieved:

- A complete study of the basic concepts of SLAM algorithm
- Mathematical explanation of Graph SLAM
- Development and integration of a Graph SLAM algorithm in ROS platform
- Calibration and set-up of a RGB-D camera
- Kick off the complete system and performing of several experiments
- Analysis of the optimal feature extractor for the GraphSLAM
- Description of the most relevant issues and fails

Chapter 7

Future Works

As in any project, the most it is researched, the most ideas and works undone appear. SLAM is one of the most amazing an interesting branches in robotics nowadays. Thanks to the new TOF sensors and the fast increase on computer performance, researchers pay more attention to this world. The following list gathers some of the ideas that the author propose as the immediately future tasks.

- Prepare the graph to be modified in dynamic scenarios
- Propose a new graph optimizer based on Differential Evolution
- Take advantage of the IR camera of the Kinect and create maps of rooms with the light off
- Include a Bayesian filter to improve the node fusion creating more stable and concise maps
- Add new constraints when adding new nodes to the graph such as the quality of the color frame to reject fuzzy images
- Improve the performance programming SURF feature detector in parallel programming

Bibliography

[1] S. Thrun, W. Burgard, and D. Fox. *Probabilistic robotics*. MIT Press, 2001.

[2] P. Cheeseman, P. Smith, and M. Self. Estimating uncertain spatial relationships in robotics. *Autonomous robot vehicles*, pages 167–193, 1990.

[3] HF Durrant-Whyte and JJ Leonard. Mobile robot localization by tracking geometric beacons. *IEEE Transactions on Robotics and Automation*, 7:376–382, 1991.

[4] A. Howard, M.J. Mataric, and G. Sukhatme. Relaxation on a mesh: a formalism for generalized localization. In *Intelligent Robots and Systems, 2001. Proceedings. 2001 IEEE/RSJ International Conference on*, volume 2, pages 1055–1060. IEEE, 2001.

[5] G. Grisetti, R. Kummerle, C. Stachniss, U. Frese, and C. Hertzberg. Hierarchical optimization on manifolds for online 2d and 3d mapping. In *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, pages 273–278. IEEE, 2010.

[6] E. Olson, J. Leonard, and S. Teller. Fast iterative alignment of pose graphs with poor initial estimates. In *Robotics and Automation, 2006. ICRA 2006. Proceedings 2006 IEEE International Conference on*, pages 2262–2269. Ieee, 2006.

[7] M. Gong and Yang Yee-Hong. *Fast stereo matching using reliability-based dynamic programming and consistency constraints*. IEEE, 2003.

[8] Hu Wu-Chih. *Adaptive Template Block-Based Block Matching for Object Tracking*. IEEE, 2008.

[9] J. Davis, D. Nehab, R. Ramamoorthi, and S. Rusinkiewicz. Spacetime stereo: a unifying framework for depth from triangulation. *IEEE transactions on pattern analysis and machine intelligence*, 27(2):296–302, 2005.

[10] Clemente Martín Alejandro Iván. Generación de Mapas de Disparidad usando CUDA. 2009.

[11] Jiaju Liu, Yanyan Xu, Reinhard Klette, Hui Chen, and Tobi Vaudrey. Disparity Map Computation on a Cell Processor. *Architecture*, page 2009.

[12] Carl B. Boyer. Early Estimates of the Velocity of Light. *The University Of Chicago Press On Behalf Of The History Of Science*, 33(1):24–40, 1941.

[13] Andreas Kolb, Erhardt Barth, and Reinhard Koch. ToF-sensors: New dimensions for realism and interactivity. *2008 IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*, pages 1–6, June 2008.

[14] Robert Lange. *3D Time-of-Flight Distance Measurement with Custom Solid-State Image Sensors in CMOS/CCD-Technology*. PhD thesis, University of Siegen, 2000.

[15] Stephan Hussmann and Thorsten Liepert. Robot Vision System based on a 3D-TOF Camera. *2007 IEEE Instrumentation & Measurement Technology Conference IMTC 2007*, pages 1–5, May 2007.

[16] Martin Bohme, Martin Haker, Thomas Martinetz, and Erhardt Barth. Shading constraint improves accuracy of time-of-flight measurements. *2008 IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*, pages 1–6, June 2008.

[17] I. Moring, T. Heikkinen, R. Myllyla, and A. Kilpela. Acquisition of three- dimensional image data by a scanning laser range finder. *Optical Engineering*, 28(8):897–902, 1989.

[18] G Beheim and K Fritsch. Range finding using frequency-modulated laser diode. *Applied optics*, 25(9):1439, May 1986.

[19] Pmdtechnologies Gmbh and D-Siegen. A performance review of 3D TOF vision systems in comparison to stereo vision systems. 2006.

[20] Ronald Chung. Correspondenceless Stereo Vision under General Stereo Camera Configuration. *Signal Processing*, (October):405–410, 2003.

[21] Rudolf Schwarte, Zhanping Xu, Horst-Guenther Heinol, Joachim Olk, and Bernd Buxbaum. New optical four-quadrant phase detector integrated into a photogate array for small and precise 3D cameras. In Richard N. Ellson and Joseph H. Nurre, editors, *Three-Dimensional Image Capture*, volume 3023, pages 119–128, San Jose, CA, USA, March 1997. SPIE.

[22] J. U. Kuehnle, Z. Xue, M. Stotz, J. M. Zoellner, A. Verl, and R. Dillmann. *Grasping in Depth maps of time-of-flight cameras*. IEEE, October 2008.

[23] S. Hussmann and H. Hess. Dreidimensionale Umwelterfassung. *Elektronik automotive, WEKA Publisher House*, (8):55–59, 2006.

[24] Thorsten Ringbeck and B U Systems. A 3D Time Of Flight Camera for Object Detection. *Measurement*, 2007.

[25] Stefan Soutschek, Jochen Penne, Joachim Hornegger, and Johannes Kornhuber. 3-D Gesture-Based Scene Navigation in Medical Imaging Applications Using Time-Of-Flight Cameras Chair of Pattern Recognition , Department of Computer Science. *Gesture*, pages 2–7, 2008.

[26] Simon Meers and Koren Ward. Face Recognition using a Time-of-Flight Camera. 2009.

[27] D.G. Lowe. Object recognition from local scale-invariant features. In *Computer Vision, 1999. The Proceedings of the Seventh IEEE International Conference on*, volume 2, pages 1150–1157. Ieee, 1999.

[28] D. Schleicher, L.M. Bergasa, R. Barea, E. López, M. Ocaña, and J. Nuevo. Real-time wide-angle stereo visual slam on large environments using sift features correction.

In *Intelligent Robots and Systems, 2007. IROS 2007. IEEE/RSJ International Conference on*, pages 3878–3883. IEEE, 2007.

[29] N. Zhang, M. Li, and B. Hong. Active mobile robot simultaneous localization and mapping. In *Robotics and Biomimetics, 2006. ROBIO'06. IEEE International Conference on*, pages 1676–1681. IEEE, 2006.

[30] Abel Alguacil Gomez. Aplicaciones del operador sift al reconocimiento de objetos. Master's thesis, Universidad Carlos III Madrid, 2009.

[31] M.A. Fischler and R.C. Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6):381–395, 1981.

[32] P. Scovanner, S. Ali, and M. Shah. A 3-dimensional sift descriptor and its application to action recognition. In *Proceedings of the 15th international conference on Multimedia*, pages 357–360. ACM, 2007.

[33] Marco Zuliani. *RANSAC for Dummies*, volume 1. University of California Santa Barbara, 08 2011.

[34] Z. Zhang. Iterative point matching for registration of free-form curves and surfaces. *International Journal of Computer Vision*, 13:2:119–152, 1994.

[35] A. Segal, D. Haehnel, and S. Thrun. Generalized-icp. In *Proc. of Robotics: Science and Systems (RSS)*, 2009.

[36] Nikolas Engelhard, Jürgen Hess, Jürgen Sturm, Wolfram Burgard, and Felix Endres. Real-time 3d visual slam with a hand-held rgb-d camera. *European Robotics Research Network*, 2011.

[37] Jorge Garcia Bueno, Piotr Jurewicz Slupska, Nicolas Burrus, and Luis Moreno Lorente. Textureless object recognition and arm planning for a mobile manipulator. *53rd International Symposium ELMAR*, 1(1):59–62, September 2011.

[38] J.A. Sethian. *Level set methods and fast marching methods: evolving interfaces in computational geometry, fluid mechanics, computer vision, and materials science*. Number 3. Cambridge Univ Pr, 1999.