eman ta zabal zazu
Universidad del País Vasco
Euskal Herriko Unibertsitatea
INFORMATIKA FAKULTATEA
FACULTAD DE INFORMÁTICA

Bachelor Degree in Computer Engineering

Computer Science

Bachelor Thesis

# Analysis of a RGB-D SLAM system using Real-Time Appearance-Based Mapping on the Kbot robot

Author

*Jon Ander Ruiz*

2022

eman ta zabal zazu
Universidad del País Vasco
Euskal Herriko Unibertsitatea
INFORMATIKA FAKULTATEA
FACULTAD DE INFORMÁTICA

Bachelor Degree in Computer Engineering

Computer Science

Bachelor Thesis

# Analysis of a RGB-D SLAM system using Real-Time Appearance-Based Mapping on the Kbot robot

Author

*Jon Ander Ruiz*

Supervisors

Igor Rodriguez & Elena Lazkano

# Abstract

The Simultaneous Localization And Mapping (SLAM) problem has been a matter of great importance and research in the area of intelligent robotics. The ability to map the environment and locate itself on the map simultaneously is an essential tool for mobile robots in an unknown environment. For localization, it is necessary to have maps. To map the surroundings, localization is needed. Very much like a chicken-and-egg problem. SLAM technology solves both the problem of localization as well of mapping together.

Looking for answers to this challenge, different approaches have been developed, i.e. Visual SLAM (vSLAM), which is SLAM using cameras, in the case of this project, a RGB-D camera.

In this Bachelor Project, the literature about robot navigation and the state of the art of SLAM approaches have been reviewed in deep. The system has been setup on the one hand, in simulation using Gazebo, and on the other hand, in a real a environment system; more precisely, using RSAIT's Kbot in the first floor of the Faculty of Informatics (UPV/EHU). Experiments in both configurations revealed the potential of the tool for accurately mapping the environment avoiding odometry error, and allowed to learn the wide set of visualization tools available to ensure map correction and proper adjustment of some parameters. The obtained maps have been used later on to command navigation goals to the robot and to prove the usability of the learned maps.

# Contents

# List of Figures

# List of Tables

# CHAPTER 1

# Introduction

In order to fulfill tasks, mobile robots need proper environmental information to be able to navigate. This information is usually given in form of a map (topological or grid based). The mapping process is affected by motion and sensor uncertainty and thus, the option to simultaneously build a map with localization uncertainty has been deeply investigated in the area of intelligent robotics. Autonomous exploration requires the robot to build a precise map of its environment, while simultaneously localizing itself relative to that map, and plan a viable and optimum path from its position to any goal position. For this, not only the robot must locate itself, but it must also locate the goal position on the map. The task of enabling mobile robots to operate without any kind of human intervention, i.e. to autonomously navigate, can be divided on 3 sub-tasks: mapping, localization and planning. The mission of mapping and estimating the pose of the robot is also known as Simultaneous Localization And Mapping (SLAM).

The area of SLAM [2], has been an area of great interest in robotics research. The generation of a precise map allows the creation of systems that are designed to work on a previously unknown environment for the robot, only using their on-board machinery and tools like sensors and cameras, without the help of GPS. This ability to generate a map and find the current position of the robot on said map is very useful, almost fundamental, for mobile robots, including but not limited to transportation robots, search and rescue robots and automated vacuum cleaners [3].

The main goal of this Bachelor Project is to acquire basic knowledge about robot navigation and visual SLAM by using the Real-Time Appearance-Based Mapping (RTAB-Map)

algorithm. As a result, the project is intended to produce an initial setup of the RTAB-Map ROS package for Kbot, one of the RSAIT's robot team members.

Given this, first the literature about robot navigation and the state of the art of SLAM approaches have been reviewed in Chapters 2, 3 and 4. Then, RTAB-Map has been studied in deep (see Chapter 5) and, an initial setup of this approach has been implemented for Kbot in simulation using Gazebo and also in the real environment system (see Chapter 6). In order to test the RTAB-Map tool capabilities, several experiments has been carried out in both simulated and real environments (see Chapter 7). Finally, the resulting conclusions are presented in Chapter 8.

# CHAPTER 2

# Robot Navigation

As previously stated, the robot navigation problem can be divided in 3 different sub-tasks or phases:

- Mapping
- Localization
- Planning

Mapping and localization are tightly coupled and are the core tasks involved in SLAM. While the more ambitious goal of autonomous exploration requires to choose and plan paths to the goals in order to automatically cover the environment to be mapped, SLAM puts aside the complexity of the planning step during map building and assumes the robot is guided in prefixed trajectories or by teleoperation. It is important to underline that, autonomous exploration is out of the scope of this project.

The following sections aims to explain how the robot navigation works, and how the task can be divided in different sub-tasks to accomplish a precise navigation without any type of human interaction.

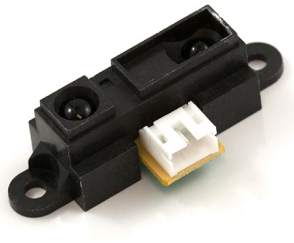
## 2.1 Mapping

The creation of maps is a very important process in robot navigation, since is also commonly used in the localization phase. A map is a spatial reconstruction of the robot's

surroundings [4]. This reconstruction, also referred as model, is created using the sensors and cameras of the robot, since the odometry alone can't be trusted.

These sensors play a significant role in acquiring accurate environmental information for further processing and mapping. Although this work focuses on a SLAM system that uses and RGB-D camera, other sensors like sonar, infrared (IR), laser scanner and LiDAR can be used for mapping the environment. Figure 2.1 shows some examples of widely used sensors for SLAM.

**(a)** Ultrasound sensor



**(b)** Infrared sensor


**(c)** Laser scanner


**(d)** LiDAR

**(e)** RGB-D Camera

**Figure 2.1:** Examples of widely used sensors for SLAM

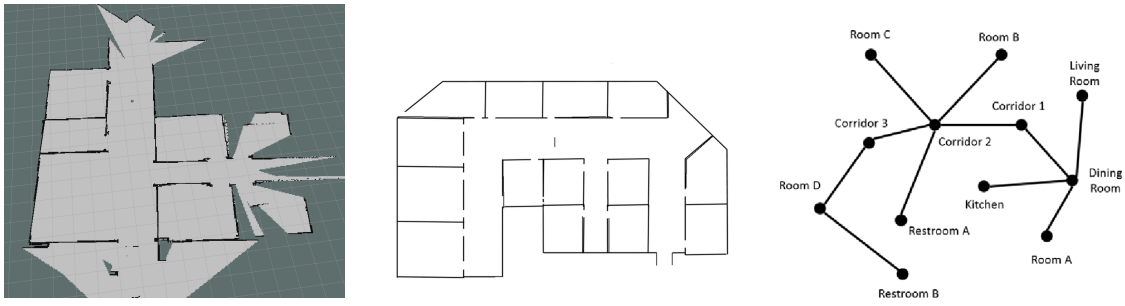
There are three major different types of maps [5]: Occupancy Grid Maps, Feature-Based Maps and Topological maps. The map representation you choose depends on different factors, such as the sensors involved in the creation of the map, the size and characteristics of the environment, and the precision needed, among others [3].

**Occupancy Grid Maps**, originally proposed by A. Elfes [6], were intended to build a configuration of the environment using a probabilistic representation of spatial information. For this kind of map, the best sensors a robot can have are range sensors, but there is a problem with sensors, they have a certain degree of uncertainty. Their responses are affected by environmental conditions.

In Occupancy Grid Maps, the map is divided in a grid. Later, each square of the grid is given a value (this is known as the grid occupancy), that represents the probability that the square is occupied [4]. The squares of the grid can have three different states: occupied, free, or unknown. Each of those states are given a probability margin [5]. Figure 2.2a shows an example of an Occupancy Grid Map. The probability of a grid to be occupied (or free) is represented by a color that ranges from white to black; darker the color higher the occupancy value.

Occupancy Grid Maps usually tend to require a lot of computation power and memory. An alternative to this type of map is the **Feature-Based Map**. This type of map is focused on extracting representative features of the environment. Three commonly used features are point, lines and planes (see figure 2.2b). With this features we can represent walls with lines and planes, which reduces the memory and computational cost [5].

**Topological maps** consist of nodes that are singular places identifiable by concrete features (not necessarily unique), and edges, i.e paths between nodes that can contain useful information like distance between features (see figure 2.2c). Therefore, topological maps are collections of nodes and edges [5]. To give an example, features could be images collected by a camera (later represented as visual words) and an adjacency list containing the links or connections between the images [7].




**(a)** Occupancy Grid Map **(b)** Feature-based map **(c)** Topological map

**Figure 2.2:** Examples of different types of map representations

## 2.2 Localization

Robot localization is is the process of determining where a mobile robot is located with respect to its environment, i.e. provides an answer to the question: “Where is the robot now?”. A mobile robot equipped with sensors to monitor its own motion (e.g., wheel encoders and inertial sensors) can compute an estimate of its location relative to where

it started if a mathematical model of the motion is available. This is known as odometry, and in a planar robot can be summarized as an estimation of the robots position relative to the worlds coordinate system. The pose of the robot at time step $k$ is defined by the 3D vector $(X_k, Y_k, \theta_k)$, where:

- $X_k$ is the $X$ position of the robot, on a certain $k$ moment.
- $Y_k$ is the $Y$ position of the robot, on a certain $k$ moment.
- $\theta_k$ is the direction the robot is facing, on a certain $k$ moment.

This estimation is commonly done by measuring the motion of the robot wheels using encoders. The process is very dependant on physical parameters. Given $v$ and $w$, the measured linear and angular velocities respectively of the robot at time $k$, and $\Delta t$, is the time difference between $k$ and $k+1$, the evolution of the robot pose is given by:

$$\begin{aligned} X_{k+1} &= X_k + v/\Delta t \times \cos(\theta_k) \\ Y_{k+1} &= Y_k + v/\Delta t \times \sin(\theta_k) \\ \theta_{k+1} &= \theta_k + w/\Delta t \end{aligned} \tag{2.1}$$

As it can be seen in equations 2.1, the estimation of the pose relies on the estimation on the previous time step and thus, the error accumulates over time. The robot can be calibrated in order to reduce the systematic error introduced by incorrect parameter values [8][9], but non systematic errors produced by environmental factors such as a slippery floor and excessive accelerations cannot be avoided.

In SLAM, the loop closure procedure is used in order to correct the error of odometry. Loop closure is the problem of detecting if the robot has already seen the area in which the robot is currently. This procedure is done in order to avoid adding redundant information (information of an already seen area) and thus avoiding unnecessary computational cost.

Now we know why we can't only rely on odometry in order to estimate the robots position.

## 2.3 Planning

The planning phase consists on finding the optimal path between the current position and a goal position avoiding any obstacle. Planning can be divided in two categories [10]:

- Off-line path-planning (Global planning): In off-line planning, the environment should be known before the planning. In short, a map of the environment is needed. Some algorithms of this type are Dijkstra algorithm and A* algorithm. In on-line planning, the robot must locate itself [10][11] continuously. To complete this task, LiDAR technology is commonly used.
- On-line path-planning (Local planning): In on-line path-planning, the robot does not have a previously generated map, the map is created while the robot is moving and perceiving the changes of the environment. The on-line path-planning is necessary in order to navigate in dynamic environments in order to avoid obstacles. Vector Field Histogram [12] and the Dynamic Window Approach (DWA) [13] are well known local planners.

Global and local planners are combined in navigation architectures as shown in figure 2.3.

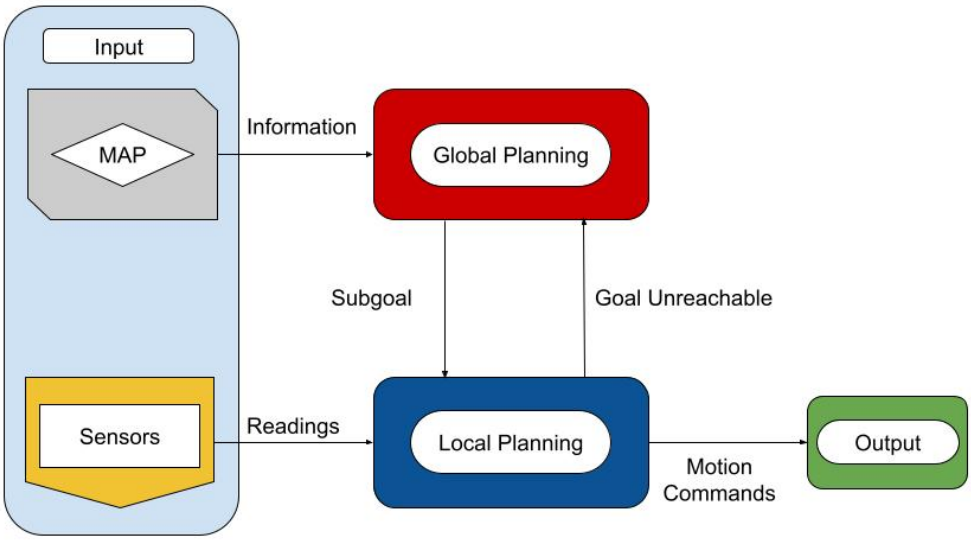



**Figure 2.3:** Combination of planners for navigation

## 2.4 Simultaneous Localization And Mapping

As mentioned before, in order to achieve robot navigation the robot must map, locate and plan according to the position of the robot relative to the map. The task of simultaneously

building a map while estimating the robot's localization in the map is called SLAM.

To solve the SLAM problem, the robot must have sensors that helps the robot to locate all near obstacles and measure the trajectory. Widely used devices in SLAM are cameras or laser range-finders.

In the following sections different approaches on SLAM are going to be analyzed. According to the literature reviewed, these approaches can be distinguished in two categories: Filter-based approaches and Optimization-based approaches [3][14][15].

### 2.4.1 Filter-based SLAM

Between the most popular techniques we find Kalman filters and particle filters which usually are designed as on-line SLAM techniques [14]. This type of approaches represent the problem as an on-line state estimation. The state of the system is the current robot position and the map. Then, as new information of the environment becomes available as the robot moves, the estimate is refined by incorporating this new information [3]. Filtering approaches are applied in two steps: the prediction step and the update step. In the prediction step, sensors like encoders and Inertial Measurement Units (IMU) are used to predict the motion of the robot, also known as odometry, and then in the update step, sensors like cameras are used in order to measure the features on images. This feature measurements and the estimated camera pose are used to update as a likelihood distribution [14].

**The Kalman filter**

The Kalman Filter (KF) [16] is an algorithm used to identify the unknown state of a lineal dynamic system and, optimally, assumes that errors have Gaussian distribution.The KF assumes that sensor readings and motion information are affected by inaccuracies and the uncertainty us expressed explicitly by noise covariance matrices in the algorithm. When either the system state dynamics or the observation dynamics are non-linear, as it frequently occurs, the conditional probability density functions are no longer Gaussian and the KF cannot give an optimal solution. In order to solve this problem, the motion model and the sensor model are linearized using first order Taylor expansion and the derivatives (Jacobian matrices) of the state transition function and observation function are needed in order to apply the KF to the linearized model, giving raise to the Extended Kalman Filter (EKF) [17].

The EKF also requires a Gaussian belief of the robot state and all the measurements taken by the robot have Gaussian noise. In SLAM, the EKF calculates the position of the robot,

as well as its orientation by verifying its state and uncertainty from the measurements taken with sensors, i.e. IMU, cameras, etc. Later, the real information taken with the camera or the sensors is integrated to improve the prediction of the pose. This improved state, as well as the uncertainty is fed back in order to model a new prediction [14].

**Particle filters**

Even though the EKF is one of the most used approaches in Filter-based SLAM due to its efficiency and easy to implement, it is limited to Gaussian processes. Particle filters (PF), are non-parametric filters able to cope with unknown probability density functions. These PFs can be used to approximate the posterior distribution over the positions of the robot. PFs are better handling outliers, but scale worst with respect to the dimensions of the state. In order to reduce the scale problem, the Rao-Blackwellised Particle Filter (RBPF) [18] aims to reduce it by factoring the state variables so that by sampling over a subset of them the remaining ones can be separated.

### 2.4.2 Optimization-based SLAM

Also known as smoothing based SLAM [3][14], these approaches tend to solve the full SLAM problem by estimating the full trajectory of the robot taking into account all of the measures. They are usually based on least-square error minimization techniques like the Graph SLAM [14]. In terms of accuracy, smoothing or optimization approaches are chosen over the filtering approaches, but they can be very memory demanding.

**Graph Based SLAM**

Optimization-based approaches address the full SLAM problem, meaning that it recovers the entire path and map, instead of just the recent pose and map [19]. A very popular way to do this is with the Graph SLAM formulation. The idea of Graph SLAM is intuitive, it consists of creating a graph where the nodes are the robot poses or landmarks and the edges between nodes are the measure constraints [3][14]. This measures can be either a measure between the robot and a feature, or a motion constraint between two robot poses. Then, the graph must be optimized, finding the configuration of the nodes that best satisfies the constraints. This means solving a large error minimization problem. The RTAB-Map algorithm used in this project [19] is a graph based SLAM technique and thus, it will be more deeply covered in chapter 4.

# CHAPTER 3

# Visual SLAM

As the name suggests, Visual SLAM (or vSLAM) uses images acquired from cameras and other image sensors such as monocular, stereo vision, omnidirectional or RGB-D cameras to localize and map the environment. The principle of vSLAM is quite easy to understand. The objective of such a system is to estimate sequentially the camera motions depending on the perceived movements of pixels in the image sequence.

It can be implemented at low cost with relatively inexpensive cameras. In addition, since cameras provide a large volume of information, they can be used to detect landmarks (previously measured positions). Landmark detection can also be combined with graph-based optimization, achieving flexibility in SLAM implementation.

Early vSLAM techniques with monocular cameras identified and mapped feature points, and was named “feature-based approach”. However, vSLAM has evolved to “direct approaches”, i.e algorithms able to use whole images to track the robot. Furthermore, the introduction of RGB-D cameras allowed new techniques that use both monocular images together with depth images [15][20].

Even if vSLAM provides very good results, it is still an emerging technology, and nowadays available solutions are prone to errors because of their sensitivity to light changes or a low textured environment. Moreover, RGB-D based approaches are very sensitive to daylight because they are based on IR light. As a result, they perform well only for indoor scenarios. Another important thing to consider is that image analysis still requires high computational complexity.

Next, Monocular SLAM and RGB-D SLAM will be reviewed.

## 3.1 Monocular SLAM

Two are the main approaches to monocular SLAM. In the early days, filtering methods were chosen over optimization methods. The use of perspective-projection cameras produced new difficulties to the problem. A single camera provided two-dimensional measurements of a three dimensional structure such as the environment, so filtering methods that allow an indirect observation model were very important [21]. To that we must add that, at the time, the optimization approach required high computational power, because it needed a good network of matches and good initial guesses, so it was discarded as it was unaffordable.

But, as time went on, and seeing that optimization techniques, such as Bundle Adjustment (BA), a state estimation technique which is used to estimate the 3D location of points in the environment, were known to provide very precise estimates of camera localization as well as sparse geometrical reconstruction [22], the search for a graph-based affordable Monocular SLAM approach began.

As it is explained in [22], such approach must meet certain criteria, which can be summarized as follows:

1. The approach must avoid unnecessary redundancy as the complexity grows.

2. It has to provide observations of scene features also known as map features, corresponding to keyframes[1].

3. It must have a strong network configuration of those keyframes and points to produce accurate results.

4. It has to provide an initial estimation of the keyframe poses and point locations for non-linear optimization.

5. It has to provide a local map where optimization is focused on scalability and the approach must have the ability to loop closure.

[1] A keyframe is a frame selected over other frames

The first ground-breaking approach that met that criteria was named Parallel Tracking And Mapping (PTAM), proposed by Georg Klein and David Murray [23], originally proposed for a small Augmented Reality (AR) workspace.

PTAM, is a method of estimating camera pose in an unknown space. The core of this approach is the division of tracking and mapping into two separate tasks, that are processed in parallel threads. One thread produces a 3D map of point features and the other deals with the task of tracking hand-held motion. The result is a system that constructs maps with thousands of landmarks, which can be tracked at frame-rate [23].

ORB-SLAM is another versatile and accurate feature-based monocular SLAM solution that operates in real time, in small and large, indoor and outdoor environments [22]. This method expands the versatility of PTAM to environments that are intractable for that system, and improves PTAM by incorporating new ideas and algorithms such as a loop detector, a loop closing procedure and a covisility graph, the optimization framework $g^2o$ (later explained in Section 4.3.2) and ORB features.

## 3.2 RGB-D SLAM

RGB-D SLAM refers to using RGB-D cameras in SLAM. The use of these cameras, provides great information to the system: colored image and depth image at the same time. That depth information has great value, as it can be used to recreate a dense reconstruction easily.

Since the proposal of PTAM many feature-based methods have been introduced for RGB-D reconstruction. RGB-D SLAM can be classified into two groups: (i) direct methods that extract all the geometry or photometric information, such as BAD SLAM [24]; (ii) feature-based methods that extract and match features from color images, such as RGBD-SLAM v2 [25], RTAB-Map [1] and ORB-SLAM2 [26]. Focusing on the second group, it is worth mentioning that these methods, in low textured scenes, they cannot provide reliable constraint because few points are extracted, and many of them are wrongly matched. And in addition, as mentioned before, the result in outdoor scenarios can be poor because the sensitivity to daylight of this type of sensors based on IR light.

It is also worth mentioning that in this kind of approaches, an Iterative Closest Point (ICP) algorithm is usually used to estimate the camera motion [15]. ICP is an algorithm that has the goal to minimize the difference between two point clouds.

# CHAPTER 4

# Graph-Based SLAM

As previously presented, there are many different ways to formulate the SLAM problem, and the problem can be solved either by using filter approaches or by optimization approaches. Among the optimization approaches, a very intuitive way of formulating SLAM is by creating a graph, hence the name Graph SLAM.

This approach consists of building a simplified estimation problem by reading, interpreting and abstracting the raw measurements of the sensor space [3][19]. Later, after the graph has been built, still the problem to find the optimal alignment or configuration of the nodes that is maximally consistent with the constraints presented in the measurements remains to be solved [14]. There are several techniques [27] to cope with it that will be reviewed later in this chapter. Whatever the solution is used, the problem can be divided in two tasks: graph creation and graph optimization. This two tasks are usually called the "Front End" and the "Back End", respectively.

## 4.1 Graph Creation and Optimization

The Front End of Graph-Based SLAM has the goal of building a graph using the information provided by the odometry and the sensors. This task includes collecting and interpreting the information, creating the graph and adding nodes and edges as the robot moves and collects more data. The design of the front end changes depending on the sensors involved. The data collected by a camera is not interpreted nor processed the same

way as laser data. Finally, the front end has to solve the *data association problem* [28], which is the challenge of identifying if two features observed at different points in time belong to the same object in the world. i.e, the front end needs to cope with the well known loop closure problem.

The Back End in Graph SLAM is where the optimization occurs. The input received by the Back End is the complete graph, and the output is the most probable configuration of robot poses and map features[1]. This means that an optimization process must be followed, finally getting the system configuration that produces the smallest error. The design of the Back End does not change as much as the Front End between applications [19].

Front and Back ends can be executed either sequentially or iteratively, by feeding the Back End with the graph to be updated and sent to the Front End at each iteration (see figure 4.1).

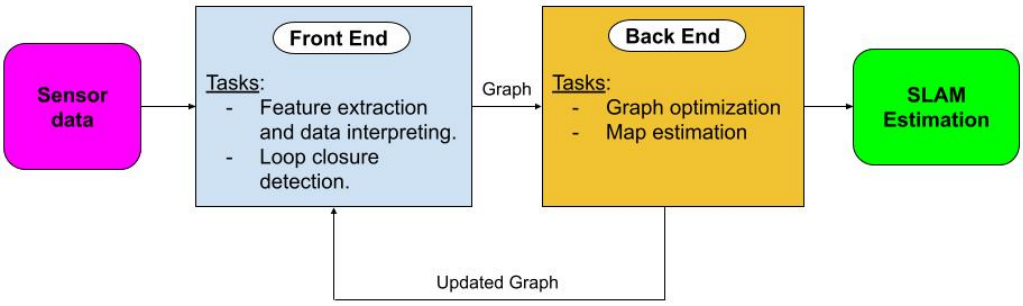



**Figure 4.1:** The tasks can be summarized as feature extraction and data association for the Front End, and Map estimation and Graph optimization for the Back End.

## 4.2 Formulating the Graph

The graph consists of two main elements:

- **Nodes:** The nodes on the graph can represent either a robot pose (pose-graph) as can be seen in figure 4.2 or both, robot poses and landmarks, at different points in time.

[1] In some cases the landmarks are not used, and thus, are not built in the graph. In that case, the representation is often referred as **pose-graph** [29]. Processing landmarks adds accuracy, but it also increases computational cost to the graph optimization, so that may be a reason no to process landmarks.

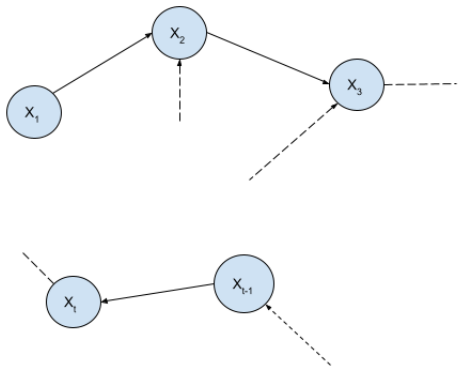



**Figure 4.2:** A pose-graph representation. The nodes correspond to a robot pose and the nearby poses are connected by edges that represent spatial constraints.

- **Edges:** The edges of the graph represent constraints among the poses. Those are obtained from observations of the environment, using for example cameras, or from movements made by the robot, obtained by sensor measurements. Every edge is labeled with a probability distribution over the relative locations of the two poses, conditioned to their mutual measurements.

## 4.3 Graph Optimization

This section aims to review the Maximum Likelihood Estimate technique and several nonlinear optimization approaches.

### 4.3.1 Maximum Likelihood Estimate

Maximum Likelihood Estimate (MLE), is used to estimate the most probable configuration of nodes given the observations of the environment. These observations come from the odometry and the sensors [19].

The measurement update at a time step $t$ is given by:

$$\overline{z^t} = x_t + m_t^{(i)} \tag{4.1}$$

Where:

- $x_t$ : The robot pose at a specific time step $t$.

- $m_t^{(i)}$ : The location of a feature $i$ in the map.

Additionally, the motion update at a time step t is given by:

$$\overline{x^t} = x_{t-1} + \mu_t \tag{4.2}$$

The terms in equation 4.2 correspond to:

- $x_{t-1}$ : Previous location of the robot..
- $\mu_t$ : A certain transformation regarding the robot status at the time step $t$. i.e. distance moved.

The updates are assumed to have Gaussian noise and thus, the corresponding probability distribution are given by:

$$\mathcal{P}_u(x_t) = \frac{1}{\sigma_m\sqrt{2\pi}} e^{-(z_t-\overline{z_t})^2/2\sigma_m^2} \tag{4.3}$$

$$\mathcal{P}_m(z_t) = \frac{1}{\sigma_u\sqrt{2\pi}} e^{-(x_t-\overline{x_t})^2/2\sigma_u^2} \tag{4.4}$$

In some cases it can be useful to convert the target function to the negative log-likelihood form:

$$J_{GraphSLAM} = \sum_t \left(\frac{z_t - \overline{z_t}}{\sigma_m}\right)^2 + \sum_t \left(\frac{x_t - \overline{x_t}}{\sigma_u}\right)^2 \tag{4.5}$$

In real world, most systems are multi-dimensional, hence, the above equations are expressed in matrix form where state and covariance matrices are involved. The constraints are given by:

$$\begin{aligned} v_t &= z_t / h(x_t, m_t) \\ w_t &= x_t / g(x_{t-1}, \mu_t) \end{aligned} \tag{4.6}$$

Where:

- $h()$ : Represents the measurement function.
- $g()$ : Represents the motion function.

Finally, the multi-dimensional formula for the sum of all constraints is given by:

$$J_{GraphSLAM} = x_0^T * \Omega * x_0 + \sum_t \left( w_t^T * R_t^{-1} * w_t + v_t^T * Q_t^{-1} * v_t \right) \tag{4.7}$$

Where the terms correspond to:

- $x_0$ : Initial position of the robot.
- $\Omega$ : Information matrix.
- $v_t$ : Constraint of measurement of the environment.
- $w_t$ : Constraint of movement of the robot.
- $Q_t$ : Covariance of the measurement noise.
- $R_t$ : Covariance of the motion noise.

### 4.3.2 Nonlinear Pose-Graph Optimization Approaches

The goal of pose-graph optimization is to find the configuration of nodes that minimizes the least squares error over all given constraints.

Normally, a nonlinear least squares optimization problem can be defined as follows:

$$x^* = \underset{x}{argmin} F(x) \tag{4.8}$$

Where $F(x)$ is the sum of errors over all constraints in the graph:

$$F(x) = \sum_{(i,j) \in C} e_{ij}^T \Omega_{ij} e_{ij} \tag{4.9}$$

The terms in equation 4.9 represent the following:

- $C$ : Represents the set of index pairs between connected nodes.
- $\Omega_{ij}$ : Represents the information matrix between nodes i and j.
- $e_{ij}$ : A nonlinear error function that models how well the poses $x_i$ and $x_j$ satisfy the constraint imposed by the measurement $z_{ij}$.

Note that each constraint is modeled using the information matrix and the error function.

Conventionally, equation 4.8 is solved by iterative optimization techniques such as Gauss-Newton or Levenberg-Marquardt. The general idea on those approaches is to approximate the error function with its first-order Taylor expansion around the initial guess.

Graph-based approaches have a sparse structure, making computation faster. They are robust to inaccurate initial guess problems, but in contrast, they have some disadvantages: for instance, they are normally not very robust to outliers and do not converge when there are sundry false loop closures to mention some.

A summary of some optimization frameworks based on the nonlinear least squares technique [27] are going to be described below:

**g$^2$o**

g$^2$o is an open-source general framework for (hyper) graph optimization [30]. This framework performs the optimization of Nonlinear Least Squares problems that can be embedded as a graph or a hyper-graph[2].
The purposes of the framework are to provide an easy-to-extend and easy-to-use library, provide an easy-to-read documentation and achieve state of the art performance.
ORB-SLAM uses g$^2$o as a back end.

**CERES**

Ceres Solver [31] is an open source C++ library for solving large optimization problems, such as the Nonlinear Least Squares problems with constraints. Implemented solvers include trust region solvers, such as Levenberg-Marquardt and Powell's Dogleg and line search solvers.
It has been used in production at Google for a few years now. It is extensively optimized and supports GPU acceleration, with CUDA technology.
It is the best performing solver on the NIST problem set.

**TORO**

Tree-based network optimizer, also known as TORO[3] is an optimization approach for constraint-networks. It provides a gradient descent-based error minimization process. In 2006 Olson, Leonard and Teller presented a, by the time, novel approach [32] to solve graph-based SLAM by applying stochastic gradient descent to minimize the error introduced by the constraints. TORO is an extension of this algorithm. It applies a tree

[2] A hyper-graph is given when an edge can join any number of nodes instead of a single pair of nodes.
[3] https://openslam-org.github.io/toro.html

parameterization of the nodes that enables a robot to deal with arbitrary network topologies, allowing the complexity to be bound to the size of the mapped area instead of the trajectory.

**GTSAM**

GTSAM[4] is an open source BSD-licensed C++ library that implements sensor fusion for robotics and computer vision applications. These applications include SLAM, Visual Odometry (VO) and Structure from Motion (SFM). It uses factor graphs[5] to model complex estimation problems.
GTSAM is used coupled with sensors to power many autonomous systems, both in academia and industry.

It is worth mentioning that the RATB-Map implementation in form of ROS package allows to choose among these different optimizers. An optimizer can better adjust to a problem, depending on the robotic platform used and the complexity of the environment it moves in.

[4] https://gtsam.org/

[5] A factor graph is a bipartite graph that represents the factorization of a function. They enable efficient computations such as the computation of marginal distributions using the sum-product algorightm [33].

# CHAPTER 5

## Real-Time Appearance-Based Mapping

Real-Time Appearance-Based Mapping (RTAB-Map) is a Graph-Based SLAM technique that uses information provided by RGB-D cameras, stereo cameras and Lidar sensors including an incremental appearance-based loop closure detector with a memory management approach. The loop closure[1] detector uses a bag-of-words method to determine whether each newly acquired frame of image (or Lidar scan) corresponds to a new location or came from a previous location. A graph optimizer is then applied to minimize the errors based on the frame. In order to ensure acceptable real-time performance when dealing with a large-scale environment, a memory management scheme is applied to limit the number of locations used for the loop closure detection and graph optimization.

Even though in this work we focus precisely on RTAB-Map, it is worth mentioning there are a great variety of open-source SLAM approaches available in ROS. To mention just a few examples, GMapping [34] and Hector SLAM [35] are two popular lidar-based approaches commonly used. Furthermore, ORB-SLAM2 [26] and RGBDSLAMv2 [25] are two other popular examples, in this case, of visual-based approaches.

Focusing again on RTAB-Map, since its initial release as open source library in 2013, RTAB-Map has been extended to a complete Graph-Based SLAM approach, and is currently being used for autonomous vehicles navigation or 3D environment reconstruction [36][37] among others. Furthermore, RTAB-Map has evolved into a cross-platform

[1] Loop closure is the process that focuses on determining if the robot has returned to previously seen location. It is also known as the data association problem.

standalone C++ library and a ROS package, named *rtabmap_ros*, driven by practical requirements such as:

- Online processing: The ability to set a maximum delay after receiving data in order to avoid lag and processing problem as the graph grows in size, end therefore the computational cost of searching for loop closures grows.
- Robust and low-drift odometry: Increase robustness of odometry by using a mix of proprioceptive[2], and exterioceptive[3] sensors, such as lidars and cameras.
- Robust localization: Increase robustness of localization, seeking to solve or minimize problems like illumination changes, and cope with dynamic environment.
- Practical map generation and exploitation: Map generation and usage in a practical way. For example, if the environment is mostly static, it is more practical to map the environment and then switch to localization mode. RTAB-Map gives the option to navigate the environment and localize the robot in the map without adding additional nodes to the graph.
- Multi-session mapping (also known as initial state problem): Solving or mitigating the initial state problem. The initial state problem, occurs when the robot is turned on and it does not know its relative position to a previously created map. This may happen when mapping in more than one session (Multi-session mapping). In order to solve this problem, the system allows the SLAM approach to initialize a new map and when a previously visited location is seen, a transformation between the two maps can take place.

Figure 5.1 describes the main ROS node of the RTAB-Map package, called *rtabmap*, the inputs required by the system and the outputs it generates. This node needs as input: the RGB-D images (or stereo images) of the camera input, odometry from any source (which can be from the camera or from a robot), and the TF[4] that defines the position of the sensors used in relation to the base of the robot. Optional inputs are either a laser scan from a 2D lidar (or laser sensor) or a point cloud from a 3D lidar. On the other hand, the outputs are: the Map Data containing the graph and compressed sensor data,

[2]Proprioception or kinesthesia is the sense of self-movement. In this case this is done using inertial measurement units.

[3]Exterioception relates to the information received from the exterior.

[4]TF maintains the relationship between coordinate frames in a tree structure buffered in time, and lets the user transform points, vectors, etc. between any two coordinate frames at any desired point in time.

the Map Graph with only the graph, and the TF with the odometry correction to derive the robot localization in the map frame. Optionally, an OctoMap, a PointCloud, and a 2D Occupancy Grid can be obtained.

The main *rtabmap* node consists of several elements connected that taking aforementioned input produce the Map Graph. The Synchronization component is in charge of take the input and output all sensor data synchronized. After sensor synchronization, the Short Term Memory (STM) module creates a node memorizing the odometry pose, sensor's raw data and additional information useful for next modules (e.g., visual words for Loop Closure and Proximity Detection, and local occupancy grid for Global Map Assembling). The structure of the MAP Graph is a graph with nodes an links. Nodes are created at a fixed rate (*"Rtabmap/DetectionRate"*) set in milliseconds. Additionally links contain a rigid transformation between two nodes, which can be of three types:

- Neighbor links, which are added in the STM between consecutive nodes with odometry transformations.
- Loop Closure, added trough the loop closure detection procedure.
- Proximity Links, added by proximity detection process.

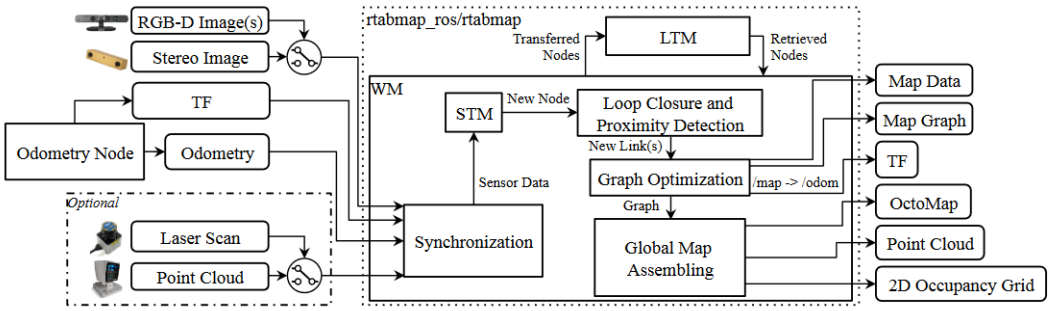



**Figure 5.1:** Block diagram of rtabmap ROS node (image taken from [1]).

These links are used as constraints for graph optimization. With the graph optimized, OctoMaps, Point Clouds and 2D Occupancy Grid outputs can be assembled and published.

## 5.1 Memory Management

The memory management approach [38], proposed by Labbé and Michaud, is used to limit the size of the graph. As the size of the map increases, so does the time required to

compare the current observation with observations stored in memory, in order to compute loop closures. Without this memory management, the time of processing the observations can become greater than the time used to acquire new observations.

This approach divides RTAB-Map's memory in Long Term Memory (LTM) and Working Memory (WM). RTAB-Map has two parameters used to control the time of processing and the size of the memory: *"Rtabmap/TimeThr"* sets a time threshold and *"Rtabmap/MemoryThr"* sets a memory threshold. Whenever one of the thresholds is exceeded, nodes are transferred from WM to LTM. In order to decide which nodes are sent to LTM, a weight value is used. This value determines which nodes are more important compared to others, and it is calculated using a heuristic that defines that the longer a location is being visualized the more important it is and consequently it should be left on WM. When a node is sent to LTM, leaves WM and it becomes unavailable for modules on WM.

The process works as follows: first the node is initialized in the STM, with weight 0, as it is new. Then, this new node is compared with the last node in the graph, looking to the number of visual words. If they are similar enough (if the percentage of corresponding visual words is over the parameter *"Mem/RehersalSimilarity"*) the weight of the last node is added to the new node. It is noteworthy that, whenever the robot is static, in order to avoid unnecessary graph growth and thus computational cost, the last node of the graph is discarded and its weight is set to 0. Note that, in order to optimize the process, the working memory must not surpass either a maximum memory threshold nor a maximum time threshold. Whenever this happens, the oldest node with less weight of the graph is sent to the LTM.

We have seen that nodes are sent from WM to LTM, but the system can also recover back nodes from LTM and send them to WM. This happens whenever a loop closure is detected in a location in the WM. Whenever this occurs, neighbor nodes of that location can be brought back to maximize the number of loop closures detected and to search for proximity detections [11].

### 5.1.1 Graph Optimization

In order to minimize the errors in the map when a loop closure or a proximity detection are identified, or some nodes are retrieved or transferred due to a memory management issue, a graph optimization approach is applied.

The *rtabmap_ros* package provides several graph optimization approaches: *TORO*, *Ceres*,

$g^2o$ and *GTSAM* (previously described in Section 4.3.2). *TORO* is more robust to multi-session mapping and less sensitive to poorly estimated odometry covariance than $g^2o$ and *GTSAM*. On the contrary, the latter converge faster and optimization quality is better than in *TORO*. In addition, as *GTSAM* is is slightly more robust to multi-session than $g^2o$, the strategy used by default in RTAB-Map is *GTSAM*. In this project, as multi-session mapping was not used, $g^2o$ has been used.

# CHAPTER 6

# Setup of RTAB-Map on Kbot

As mentioned before in Chapter 5, *rtabmap* is the main node of the RTAB-Map implementation on ROS. This main node is a wrapper of the RTAB-Map Core library, and its main goal is to create a Graph-Based map which is incrementally built and optimized when a loop closure is detected. Therefore, the online output of this node is a local graph with the latest added data to the map, and it is given in a database (.db) format. By default the database is stored in "$\sim/.ros/rtabmap.db$" and the workspace is also set to "$\sim/.ros$". Once the the database is created, it is possible to get the 3D point cloud or the 2D occupancy grid map by subscribing to the topics *cloud_map* or *grid_map*, respectively.

In the following sections, on the one hand, the main tools employed in this project are described, as well as the real platform used for the experimentation. On the other hand, the package and files created and the parameters used to configure *rtabmap_ros* package in Kbot are enumerated.

## 6.1 ROS and Gazebo

Robot Operating System (ROS) is an open source set of software libraries and tools that helps in the development of robot applications. It is a very popular tool within the robotics research community. Summarising ROS main characteristics, its most useful functionalities are: the computation graph, which allows to see the communication between processes; the file system, which contains packages like RTAB-Map (core package used in

this work); and the extensive amount of tools and algorithms. The main mechanism used by ROS nodes to communicate is by sending and receiving messages. The messages are organized into specific categories called topics. Nodes may publish messages on a particular topic or subscribe to a topic to receive information.

ROS also provides tools such as *Rviz*[1], that helps to visualize information related to the robot (e.g. laser scans or images captured from sensors and cameras), or such as *rqt*[2], a set of Graphical User Interface (GUI) tools in the form of plugins that allow graphical representations of ROS nodes, topics, messages and other information. to see the process tree and how they communicate between them.

Gazebo[3] is an open-source 3D robotics simulator and a toolbox of libraries and loud services that allows simulating real-world physics in high fidelity simulation. It helps robot developers rapidly test algorithms and design robots in digital environments. Moreover, Gazebo brings the opportunity to integrate a multitude of sensors, and it provides the necessary tools to test those sensors and develop own robots to best use them.

In this project, a representation of the first floor of the Faculty of Informatics in San Sebastian (UPV/EHU) together with a simulated model of Kbot, both developed by RSAIT[4], have been used to setup and test the RTAB-Map tool. In figure 6.1, the 3D model of the first floor can be seen.

## 6.2 Kbot

Kbot is a differential drive robot built by Neobotix in 2004, originally used as a tour guide at the Eureka Museum of Science in San Sebastian. In 2006 the robot broke down and kept in a warehouse until 2014, when it was sent to the University of the Basque Country. The RSAIT team repaired the robot and renewed some elements as well as supplied Kbot with an onboard Zotax MiniPC with a NVIDIA graphics card and an Intel Realsense D435[5], as can be seen in figure 6.3. This camera is a stereo solution that consists of a pair of stereo cameras, a RGB sensor, and an infrared projector. It uses stereo vision to calculate depth. The rigid arms where a touch monitor rested were removed and instead a smaller Getich monitor was installed on its back. In the front we can observe several sonars and the laser

[1] https://wiki.ros.org/rviz
[2] https://wiki.ros.org/rqt
[3] https://gazebosim.org/home
[4] http://www.sc.ehu.es/ccwrobot/
[5] https://www.intelrealsense.com/depth-camera-d435/

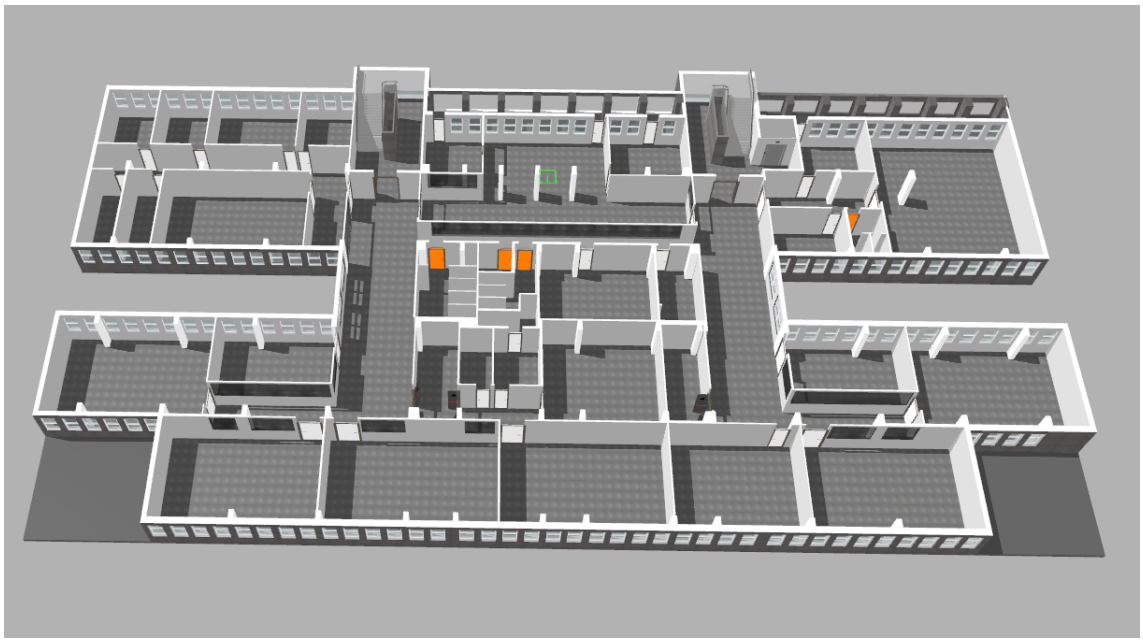
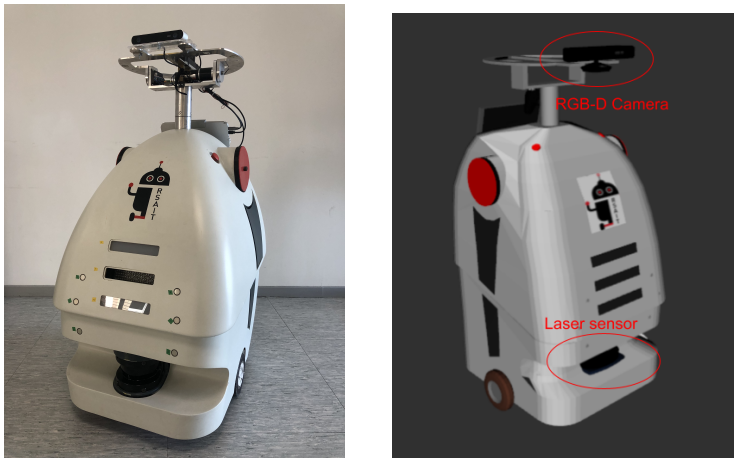


**Figure 6.1:** First floor of the UPV-EHU's Computer Science faculty represented as a Gazebo environment, built by the RSAIT team.




**(a)** Real robot **(b)** Simulated robot

**Figure 6.2:** Kbot's appearance

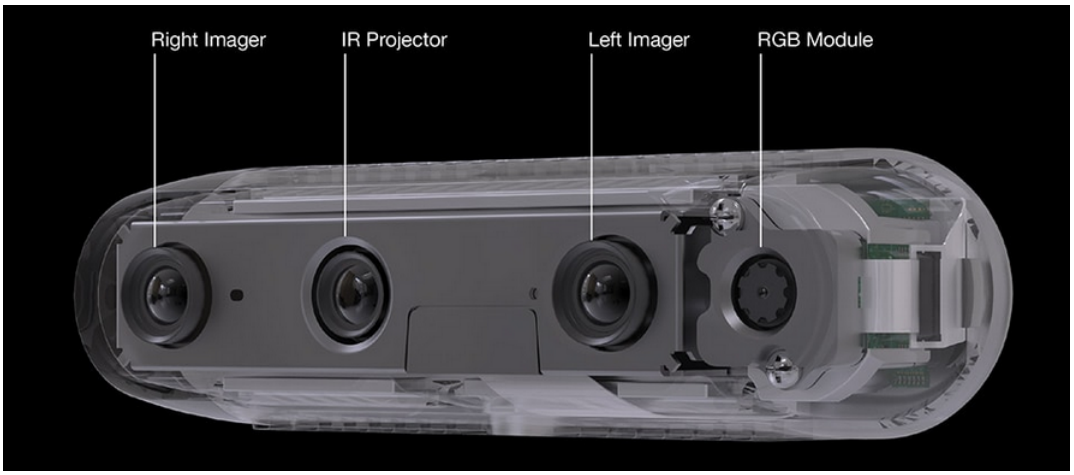



**Figure 6.3:** Intel Realsense D435 Depth Camera. It is equipped with right and left imagers, an IR Projector and a RGB module.

sensor. For safety reasons, the robot is equiped with two emergency stop buttons on each side that when pressed cause an electrical stop. Another safety mechanism of Kbot is that when the laser sensors detects an object dangerously close to the robot it causes to stop any further movement. However, this safety measures are only active on the real robot, not on the simulation.

The robot can be operated using the keyboard when *key_teleop* program is launched both in reality and in the simulation, and with the gamepad using a similar program.

Figures 6.2a and 6.2b show Kbot's real and simulated appearances, respectively.

## 6.3 *rtabmap_kbot* package

In order to start with RTAB-Map ROS package on Kbot (both in the simulation and in the real one), a ROS package, *rtabmap_kbot* was created. This package contains the launch file *rtabmap.launch* that contains the modified parameters of *rtabmap* and the *rviz* configuration file, used to see how the robot builds the environment in real time and to set goals. In order to start the simulation, a package named *gazebo_utils* has been provided by the RSAIT team, as well as the package *gazebo_navigation*. The former contains the robot model and the environment while the later contains the ROS Navigation Stack[6], configured for the Kbot. Additionally, in order to enable Kbot to cross hallways, the parameter that controls the minimum distance allowed between the robot and an obstacle, had to be

[6]https://wiki.ros.org/navigation

modified. Finally, in order to move the robot, the package *key_teleop* package has been used.

### 6.3.1 Parameters and topics of the *rtabmap* node

*rtabmap_ros* package has a well documented ROS wiki page[7] that includes several tutorials and demos showing some examples of how to build a map using RTAB-Map. As described before in Section 6.3, the core of the package is a launch file (*rtabmap.launch*) that executes the *rtabmap* node with specific setup, including RTAB-Map's parameters and the required topics. This launch file is based on a template developed by the ROS community to use *rtabmap* on a Turtlebot2[8].

There are two set of parameters: ROS and RTAB-Map's parameters. The ROS parameters are for connection stuff to interface the RTAB-Map library with ROS, while the RTAB-Map's parameters are those from the RTAB-Map library. Table 6.1 describes in short the main RTAB-Map's parameters used in this work. Although in the Turtlebot's customized launch file you can find several explanations about those parameters and which values can they have, all the configurable parameters are better described in the `Parameters.h` file of the *rtabmap_ros* package.

By default, *rtabmap* is in mapping mode, but this node also provides the option to set it in localization mode with a previously created map. To do so, the memory parameter (*Mem/IncrementalMemory*) should be set as not incremental.

Regarding the topics required by *rtabmap* node, it is necessary to specify the input data that will be used by the node to generate the Map Graph.

The topics that are necessary are *scan*, *rgb/image*, *depth/image* and *rgb/camera_info*. *scan* is the topic of the laser sensor, and in the Kbot corresponds to the topic */scan*. The topics *rgb/image*, *depth/image* and *rgb/camera_info* are the topics of the camera and correspond to the Kbot's */camera/rgb/image_raw*, */camera/depth/image_raw* and */camera/rgb/camera_info*.

[7] https://wiki.ros.org/rtabmap_ros#rtabmap
[8] https://robots.ros.org/turtlebot/

| Parameter | Value | Explanation |
|---|---|---|
| RGBD/ProximityBySpace | true | Local loop closure detection (using estimated position) with locations in Working Memory. |
| RGBD/OptimizeFromGraphEnd | false | Set to false to generate map correction between /map and /odom. |
| Kp/MaxDepth | 8.0 | Filter extracted keypoints by depth. |
| Reg/Strategy | 2 | Loop closure transformation refining with ICP: 0=Visual, 1=ICP, 2=Visual+ICP. |
| Icp/CorrespondenceRatio | 0.3 | Ratio of matching correspondences to accept the transform. |
| Vis/MinInliers | 6 | 3D visual words minimum inliers to accept loop closure. |
| Vis/InliersDistance | 0.1 | 3D visual words correspondence distance. |
| RGBD/AngularUpdate | 0.3 | Update map only if the robot is moving. Minumun rad to update. |
| RGBD/LinearUpdate | 0.3 | Update map only if the robot is moving. Minimun distance moved to update. |
| Rtabmap/TimeThr | 500 | Maximum time allowed for map update. |
| Mem/RehearsalSimillarity | 0.30 | Rehearsal similarity. |
| Reg/Force3DoF | true | Force 3 degrees-of-freedom transform (3DoF: x, y, and yaw). Parameters z, roll and pitch will be set to 0. |
| Optimizer/Strategy | 1 | Graph optimization strategy. 0=TORO, 1=g2o, 2=GTSAM, 3=Ceres. |
| Mem/InitWMWithAllNodes | false | Initialize the Working Memory with all nodes in Long Term Memory. When false it is initialized with nodes of the previous session. |
| Mem/IncrementalMemory | true | true for SLAM mode, false for localization mode. |

**Table 6.1:** Table of relevant parameters changed on *rtabmap.launch file*

## 6.4 Additional Tools

### 6.4.1 *rtabmapviz* node

This is a node that starts the visualization interface of RTAB-Map. It is a wrapper of the RTAB-Map GUI library, and it has the same purpose as *rviz* but with specific options for RTAB-Map. It allows to configure several parameters in real time such as RTAB-MAP update rate or time limit processing, but also shows information about loop closure detection and memory management (see Figure 6.4).

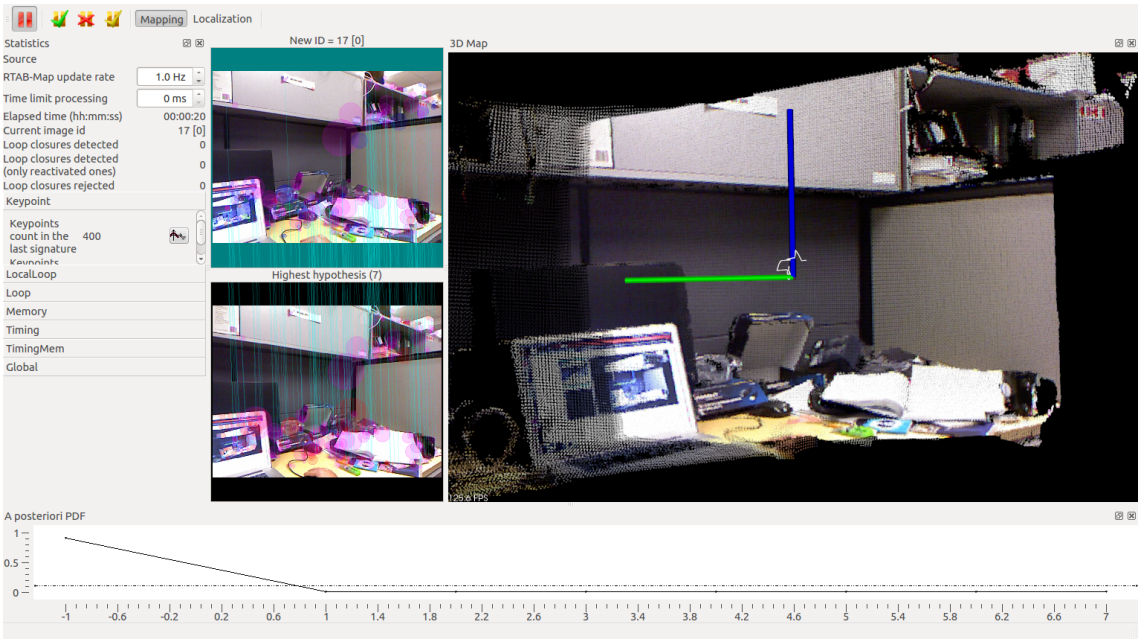



**Figure 6.4:** Database viewer. To the left the Graph view can be seen, and to the right the frames can be seen, showing the similarities (blue lines). These features are only a few.

### 6.4.2 *rtabmap* Database viewer

The Database viewer tool is used to browse the data that is stored in the RTAB-Map databases. With this tool, 3D maps, graphs and 2D occupancy grid maps (among others) can be generated and it is also useful to observe the loop closures.
Regarding loop closures, the user can add or remove them, thus changing the graph.
An example of the GUI can be seen on figure 6.5.

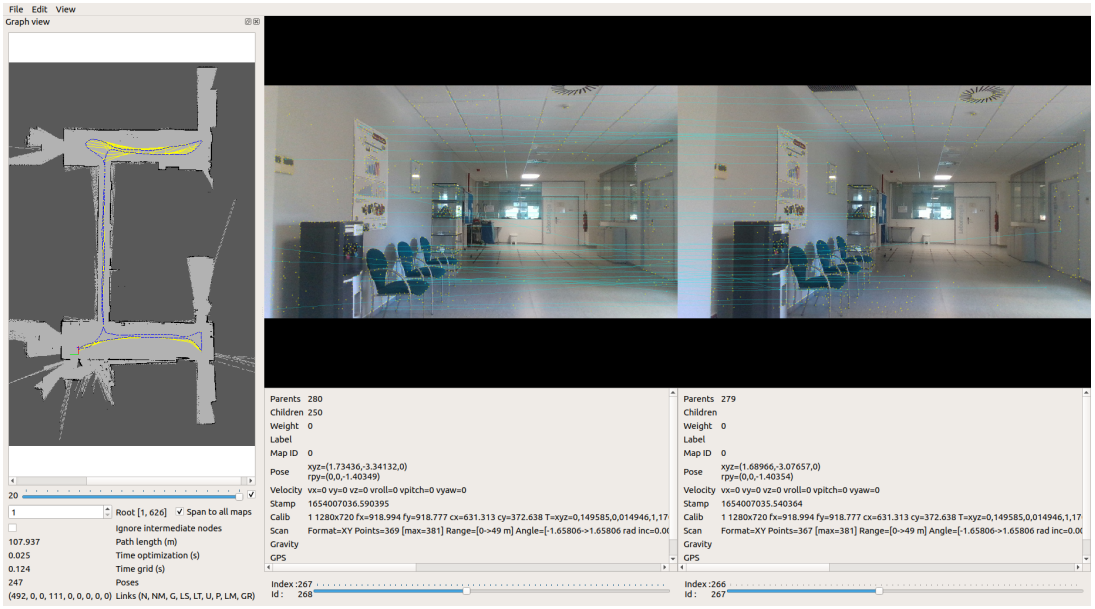



**Figure 6.5:** Database viewer. Left: Graph view. Right: Image frames and correspondences (blue lines).

# CHAPTER 7

# Experiments and Results

The experimentation carried out in this project consists of testing the RTAB-Map tool capabilities on the Kbot robot to build a map of the environment and navigate in it at the same time, using visual information captured from an RGB-D camera.

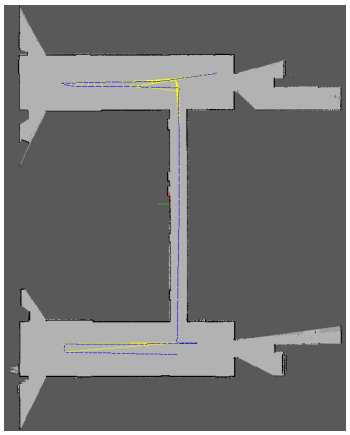
The experiments have been performed both in real and simulated environments. Specifically, the first floor of the Faculty of Informatics have been chosen as testing scenario. Additionally some experiments have also been carried out on the third floor. For the simulated part, a Gazebo world representing the Faculty's first floor developed by RSAIT is employed.

The objective of the experiments is therefore to analyse the functionalities of *rtabmap_ros*. In both simulation and real environment the mapping process has been the same: teleoperate the robot through the environment while acquiring the map. During the mapping process several videos have been recorded to show how the map is built in real time (through *rviz* and *rtabmapviz* tools). After the mapping process a RTAB-Map database (in .db format) is obtained including the 3D point cloud representation, the Graph-Based map and the final 2D occupancy grid map. This map is then used for testing the navigation on Kbot.

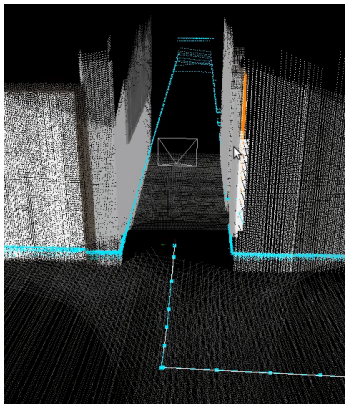
## 7.1 Mapping in simulation

In order to map the environment, the robot has been controlled with the keyboard, using the teleoperation package. The robot trajectory can be seen in figure 7.1a. The robot starts the mapping process at the lower part of the map (close to the dean's office). After going through that area it was directed to the hallway to finally map the space close to Ada Lovelace's room. In figure 7.1b the 3D point cloud can be observed. The process can be seen in the video[1].

The mapping process in the simulated environment was completed resulting a map of 155,2 MB. The graph has 560 links (of which 2 are global loop closures and 126 are local loop closures by space), and 256 poses. The mapping was done in one session of 5 minutes and 7 seconds. Overall the map of the simulation has showed good results, but it must be taken into account that in simulations there is no odometry error.



**(a)** Robot's trajectory and 2D occupancy grid of the First floor in the simulated environment.



**(b)** 3D point cloud of the environment. The graph can also be seen being created, as well as the laser readings showing where the wall is.

**Figure 7.1:** Mapping on the simulator.

## 7.2 Planning in simulation

The goal of this experiment is to measure the adequateness of the previously created map to perform navigation. In order to carry this experiment out, *rtabmap* must be set on lo-

[1] https://drive.google.com/file/d/1Zud7laq_p0z4-oyFPWb0nOHO0SNYRbLi/view?usp=sharing

calization mode, changing setting the argument "localization" to "true", the parameter *"initWMWithAllNodes"* to "true" and the ROS navigation package *move_base* is needed. Changing the localization mode to "true" means that it sets the parameter *"Mem/IncrementalMemory"* to "false".

The *move_base* package provides an implementation of an action that, given a goal in the world, will attempt to reach it with a mobile base. The *move_base* node links together a global and a local planner to accomplish its global navigation task. In order to set a goal in the map, *rviz* provides an utility named “2d nav goal” that just selecting that option and clicking on any point in the map, a path from the robot’s location to the goal location is planned (if any) and sent to the robot in order to achieve the navigation task.

Figure 7.2a shows the global plan calculated by the global planner and to be followed by the robot in red, creating the shortest possible path between the robots position and the goal position. This global plan consists of several subgoals that define the path the robot has to follow. Whenever the robot finds itself unable to reach a subgoal, the local planner makes a variation in that path (the path between the robots position and the subgoal) and creates a deviation in the global path in order to reach that checkpoint. Figure 7.2b shows the local planner modifying the trajectory in order to avoid obstacles.

A video[2] has been recorded to show Kbot’s behaviour while navigating. As can be seen in the video, the robot is able to autonomously navigate to the goal, but sometimes it loses track of its position momentarily, This is particularly noticeable when the environment is featureless, i.e. locations looks very similar. However, when the robot reaches a point in the map where relevant features can be observed, such as a door or a window, the robot makes a loop closure and relocates itself.

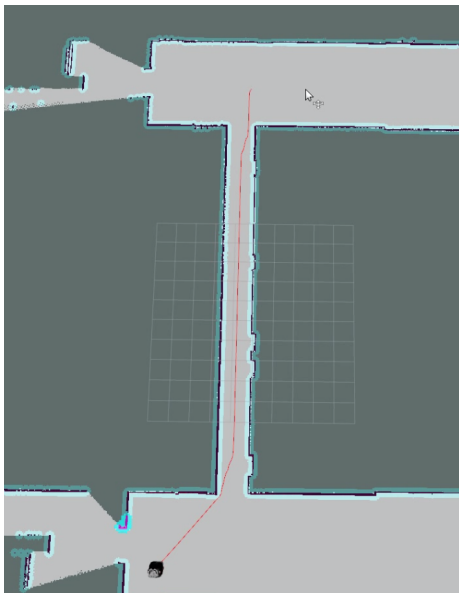
## 7.3 Mapping with Kbot

The process followed in this experiment is essentially the same as in simulation, using the real Kbot in the first floor of the faculty, but this time the robot is teleoperated using a joystick. The process can be seen in the video[3].

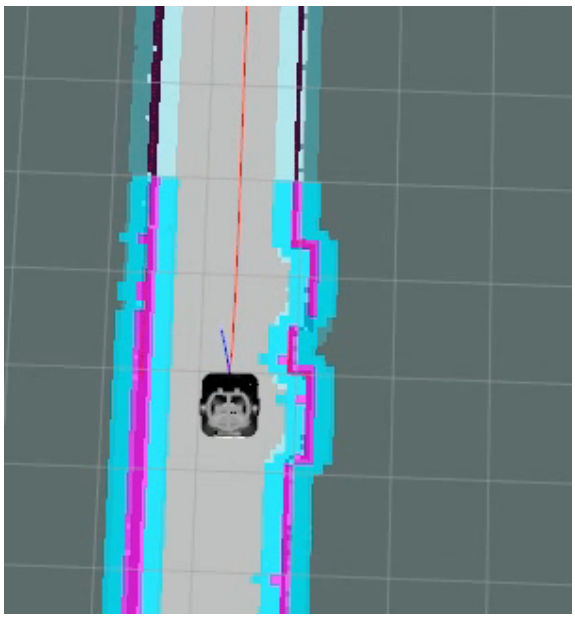
The map of the first floor was built in one session of 15 minutes and 24 seconds and

[2] https://drive.google.com/file/d/1Ha3X5T2vB5_IOyHYatYKSUNnVnfJlnX9/view?usp=sharing

[3] https://drive.google.com/file/d/1sa44TjQ05BEi2jjChYMnMnNlm6dvj7gi/view?usp=sharing



**(a)** Kbot planning. The red lines denote the global planned path.



**(b)** The blue lines mark the local plan.

**Figure 7.2:** Planning on the simulation.

produced a file of 566 MB. The graph has 627 poses and 603 links divided as 492 links, 111 local loop closures of space and, because of the characteristics of the environment and the way the robot was controlled, there is no global loop closures.

During the experimentation with the real Kbot, a problem with the map arose. Although the map is saved correctly, it cannot be visualized due to lack of memory. The robot's memory can't handle the size of the whole map, as a consequence the system only loads the areas close to the robot pose. Whenever the robot moves, the system loads the correspondent area.

In more featured environments, more global loop closures may be computed. For example, we built a map of the third floor of the faculty, also in one session, which is larger than the first floor and more importantly, it's structure is different, and in that case 36 global loop closures were computed (see 7.5 for a more detailed explanation). The results of the mapping can be checked in figure 7.3.

## 7.4 Planning with Kbot

The procedure followed during this experiment is the same as in simulation, but using the real Kbot and in the faculty's first floor. As in simulation, in order to define a goal on the

**Figure 7.3:** Robot's trajectory and 2D occupancy grid of the first floor in the real environment.

map, "2d nav goal" has been used. In the video[4], it can be seen how the robot is sent from its location to the other end of the map, and then back to the initial pose.

A fragment of the global plan can be seen in figure 7.4.

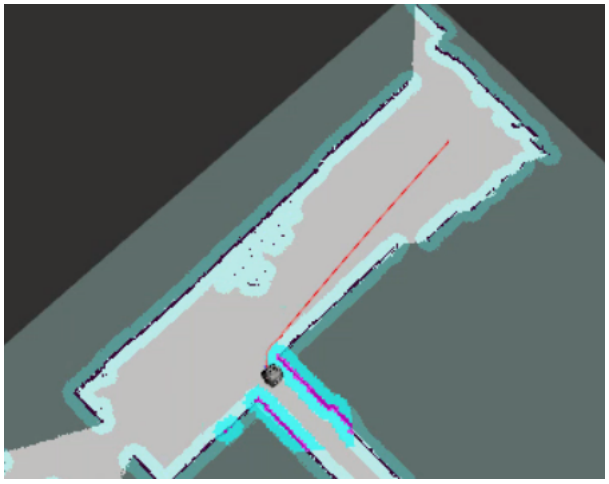


**Figure 7.4:** Kbot on the move with a goal set.

As it happened with the mapping process, the map was not fully loaded neither during the planning, but the map corresponding to a new area was loaded while the robot moved. In the same video it can be seen how the map was updated as the robot approached new areas of the environment. It must be noted that it was not new mapping, as localization mode was enabled, neglecting any change in the database, and thus not building any new map.

## 7.5 Additional Experiments

In order to explore certain functionalities of *rtabmap_ros* more experiments have been done. A mapping session of the third floor of the faculty was conducted, and very interesting results were achieved. As previously introduced, the characteristics of the environment and the user's ability to cover the environment while mapping are also very important for finding loop closures. As it can be appreciated in figure 7.5, the mapped area is larger than the one mapped in the first floor and it has a square form. Nevertheless, the results of the session are very good indeed.

[4] https://drive.google.com/file/d/1dwk8VcCcLT3zN00e91F90lrNOSmeQWbz/view?usp=sharing

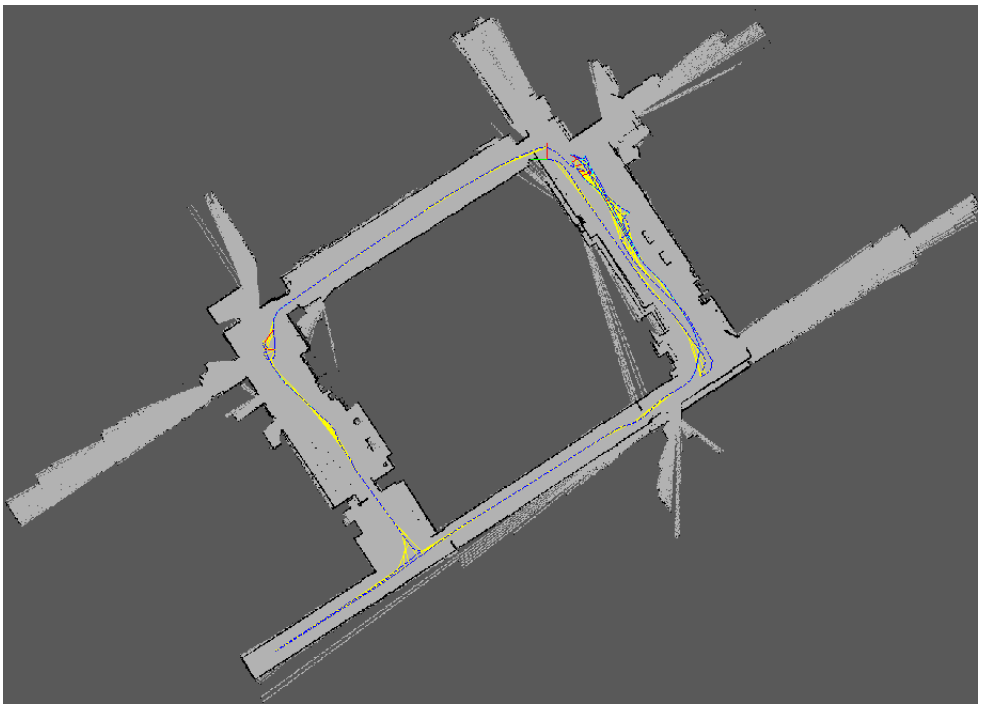


**Figure 7.5:** Map of the third floor. The mapping was done in one session of 13 minutes and 13 seconds. The file is 609,2 MB. The graph has 728 poses and 1281 links. 950 of the are normal links, 36 of them are global loop closures and 295 of them a local loop closures (time)

However, the Database viewer let us perform additional loop closures, as well as to ignore some. Figure 7.6 shows the difference between correcting poses and using loop closures, and also shows the ever present odometry error.

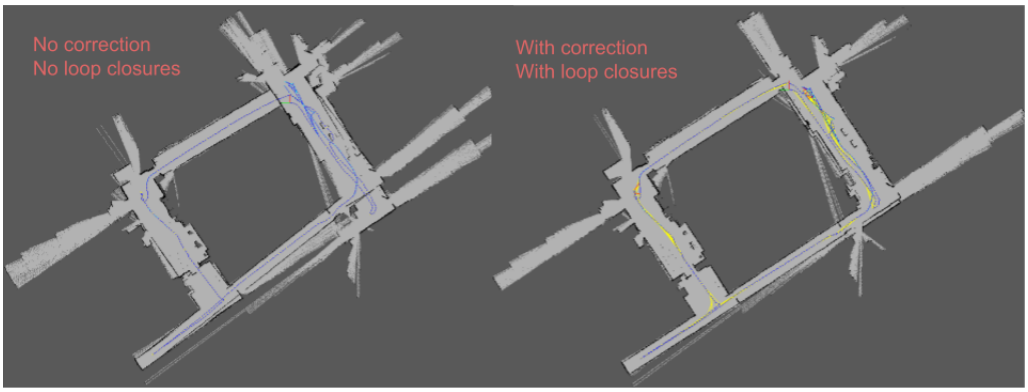



**Figure 7.6:** Importance of loop closures and pose corrections. The left image ignores pose correction, global loop closures and local loop closures. The right image is the map of the session.

In addition to help detecting loop closures, Database viewer allows to modify the loop closure radius, as well as the number of iterations. In figure 7.7 we can see the results of increasing the number of loop closures. After setting the number of iterations to 10

value, the system found 639 new loop closures, improving greatly the precision of the map. Compared to fig 7.5, we can observe that it is much more precise.

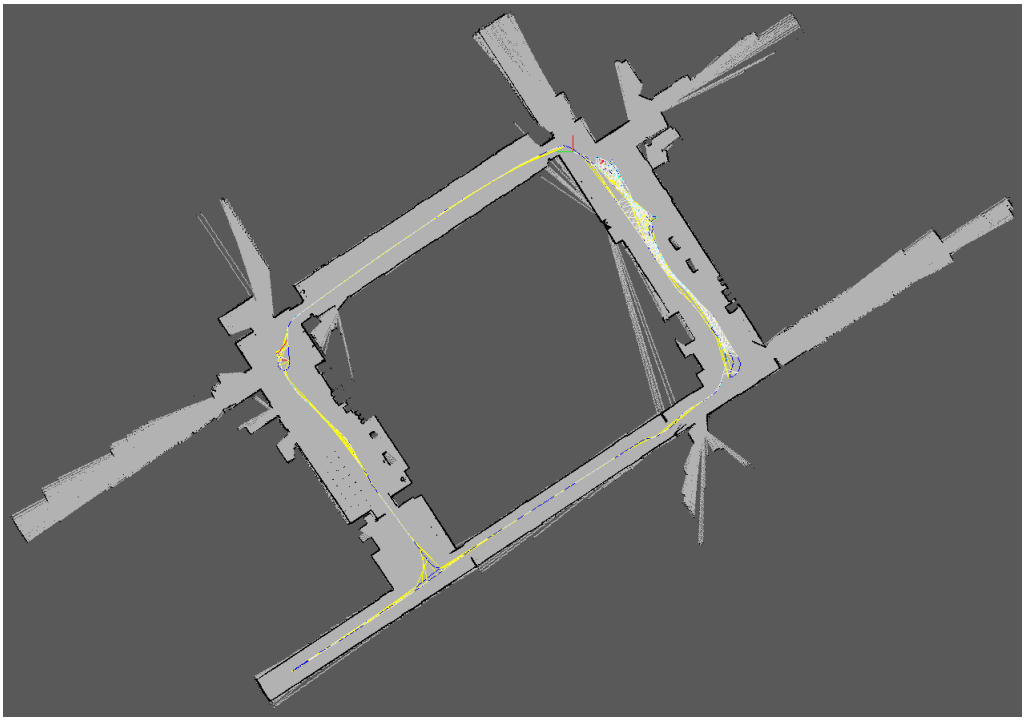


**Figure 7.7:** The map of the third floor after being modified by the option to look for more loop closures. The new white lines represent loop closures added due to the modification.

## 7.6 Identified issues in the application of RTAB-Map

Despite the satisfactory setup of the RTAB-Map in Kbot, there are several issues confronted during the development of this project that are worthwhile mentioning.

Regarding the experimentation conducted on simulation, we found that whenever the robot approaches some corners (and sometimes even in randomly situations), Kbot bent forwards leading to obtain false scan readings from the laser sensor. That issue made the robot to perceive the floor as a downwards slope (see Figure 7.8), and thus the RTAB-Map database had to be discarded.

In the experiments carried out in the first floor of the Faculty of Informatics, we also suffered from some issues. One of the main problems concerns to the odometry error. Certain areas of the Faculty's floor have small flaws that make the robot wheels slip and therefore, as one of the wheels turns faster than the other, the robot interprets it is turning to one side

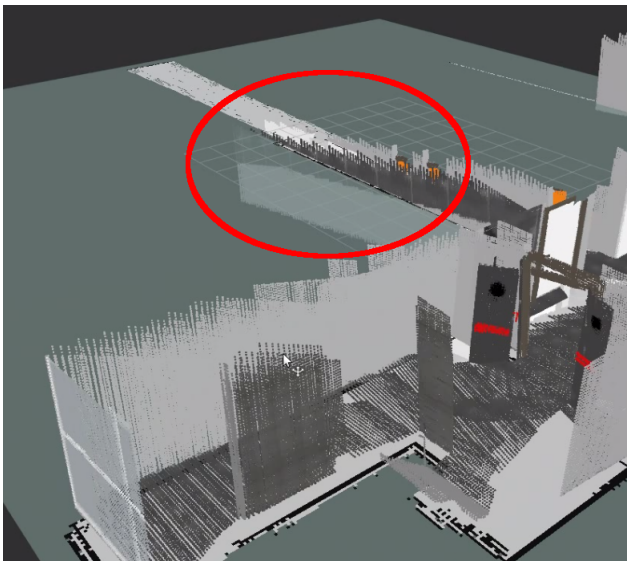


**Figure 7.8:** Snapshot of the moment when Kbot bends forward.

when it is really moving forward. Finally, sometimes due to the safety mechanism of the robot, whenever Kbot approached a wall in order to turn, there where several occasions where the robot faced the wall and stopped, because it found the wall excessively close. This can be solved either by modifying the robot's behavior in those situations, which of course we discarded because of safety reasons, or by modifying the costmap's characteristics, and forbidding the robot to get that close to the walls. The robot would have a much harder time getting through narrow hallways though, or it could simply be unable to access certain areas. Thereby, no modification was made to the navigation stack.

# CHAPTER 8

# Conclusions and Future Work

The goal of this Bachelor Project was to acquire basic knowledge about robot navigation and visual SLAM by using the RTAB-Map ROS package. During its development, the literature about robot navigation and the state of the art of SLAM approaches have been reviewed in deep. The system has been setup in simulation using Gazebo and also in the real robot environment system. Experiments in both configurations revealed the potential of the tool for accurately mapping the environment avoiding odometry error, and allowed to learn the wide set of visualization tools available to ensure map correction and proper adjustment of some parameters. The obtained maps have been used later on to command navigation goals to the robot and to prove the usability of the learned maps.

*rtabmap_ros* is a great way to start studying visual and Graph-Based SLAM, as well as achieving a better understanding of robot navigation in general. The package, *rtabmap_ros*, is easy to use and, as is modern and continuously being updated and improved, there is a lot of literature regarding different experiments carried with the software.

Regarding this work, a comparison of different graph optimization techniques was left out of the scope of the project. This will allow not only to study how the different algorithms work in reality, but also to find the best parameter composition for each situation. It also remains as further work the study of multi-session mapping and trying different kind of environments, such as, bigger environments.

In regard to the future of robot navigation, we are able to see the growing number of optimization approaches, and generally more computationally expensive approaches. Nowa-

days graph-based methods belong to State of The Art techniques because of their accuracy and speed, and they are the center of attention of different research teams that try to optimize even further graph SLAM methods, using different techniques, as neural networks [39].
Neural networks are being used today in order to solve SLAM, as it can be seen in [40], and new concepts are born, such as neuromorphic SLAM.
Moreover, certain State of The Art investigations are underway in the area of autonomous navigation while mapping. Certain systems have the ability to map without a human operator. This is called native SLAM [41].

The development of this project has produced an initial setup of the RTAB-Map algorithm that opens the door for future expansions/development in the real robot-environment system.

# CHAPTER 9

# Project Management

This chapter defines the tasks done in the project, the time spent on those tasks and problems encountered during the procedure of the project.

## 9.1 Work Breakdown

The work load has been divided in five different tasks as it can also be seen in figure 9.1 :

1. T1 Preparation: In order to understand the basics of ROS, the tutorials on the ROS page where completed.
2. T2 Investigation and Study: Deep analysis and research of SLAM, Graph SLAM, Visual SLAM, graph optimization approaches and techniques, RTAB-Map, and *rtabmap_ros*.
3. T3 Implementation: Research of implementations on other robots, as well as research on ROS topics and parameters and adaptation of the package to the Kbot, both in the simulation and in the real one.
4. T4 Experimentation: Performing the experiments, both in the simulation and in the real robot, as well as study the results.
5. T5 Documentation: Write this document, the poster and the presentation, as well as creating the necessary material to present this project.

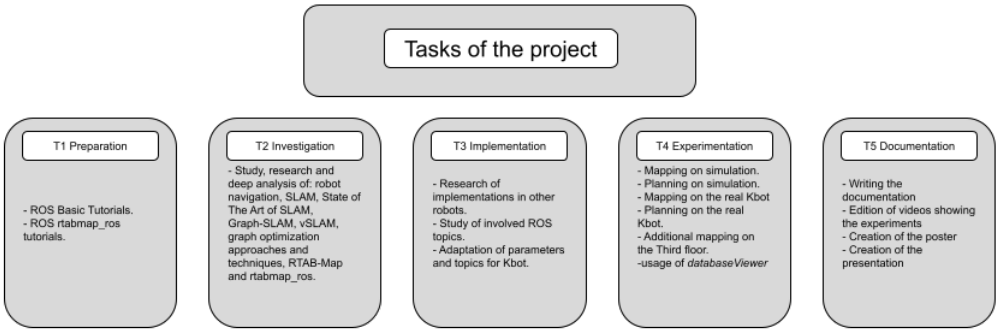



**Figure 9.1:** Tasks involved in the creation of the project

In figure 9.2 the Gantt diagram can be seen. Regarding the estimation of time, the table 9.1 shows the comparison between the estimated time and the employed time

| Task | Estimated time (h) | Final time (h) |
|---|---|---|
| T1 Preparation | 20 | 15 |
| T2 Investigation | 160 | 185 |
| T3 Implementation | 50 | 40 |
| T4 Experimentation | 30 | 25 |
| T5 Documentation | 60 | 80 |
| Total | 320 | 345 |

**Table 9.1:** Time estimated and employed on this project

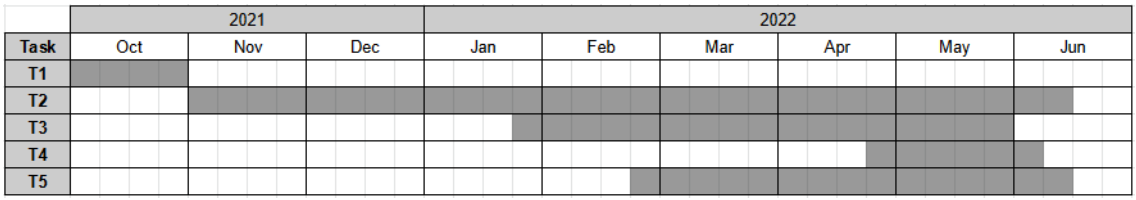



**Figure 9.2:** Gantt Diagram showing the time for each of the tasks.

## 9.2 Risk Management

The following risks have been identified:

- R1: Between October 2021 and June 2022, it is expected the pandemic of COVID-19 will not disappear, and the possibility of contagion is present. As this project is individual, the work can be done in lockdown without any further problems, except

to the experimentation in the real robot. In case of suffering serious symptoms or being infected in just before the experimentation on the real robot phase, the work could be delayed, and thus, the option to extend the date of delivery to September could be considered.

- R2: *rtabmap_ros* is a state of the art package, which is continuously being updated. In the worst case scenario, an update could go wrong and delay the work. In that case, extending the date of delivery to September could be considered.
- R3: In case the real Kbot would be unavailable, either by a software problem, or a hardware problem, the experimentation on the robot could be compromissed. In the worst case scenario, where the robot is not available indefinitely, the experiments on that robot would be omitted.

## 9.3 Evaluation

Regarding the risks, no contemplated risk has occurred. As this work is a work of analysis, a great investigation work was expected, and that is what has happened. However, T2 has posed a greater challenge than anticipated, as the student was not familiar with the proposed subject prior to the start of this project. Regarding T3, at the start of the project, several errors with the simulation delayed the start of the implementation phase. However, the implementation and the adaptation of the parameters for the robot, getting good results, was achieved earlier than expected.

# Bibliography


[1] M. Labbé and F. Michaud, "Rtab-map as an open-source lidar and visual simultaneous localization and mapping library for large-scale and long-term online operation," *Journal of Field Robotics*, vol. 36, no. 2, pp. 416–446, 2019.

[2] H. Durrant-Whyte and T. Bailey, "Simultaneous localization and mapping: part I," *IEEE robotics & automation magazine*, vol. 13, no. 2, pp. 99–110, 2006.

[3] G. Grisetti, R. Kümmerle, C. Stachniss, and W. Burgard, "A tutorial on graph-based slam," *IEEE Intelligent Transportation Systems Magazine*, vol. 2, no. 4, pp. 31–43, 2010.

[4] S. Thrun *et al.*, "Robotic mapping: A survey," *Exploring artificial intelligence in the new millennium*, vol. 1, no. 1-35, p. 1, 2002.

[5] S. Yu, C. Fu, A. K. Gostar, and M. Hu, "A review on map-merging methods for typical map types in multiple-ground-robot slam solutions," *Sensors*, vol. 20, no. 23, p. 6988, 2020.

[6] A. Elfes, "Using occupancy grids for mobile robot perception and navigation," *Computer*, vol. 22, no. 6, pp. 46–57, 1989.

[7] F. Fraundorfer, C. Engels, and D. Nistér, "Topological mapping, localization and navigation using image collections," in *2007 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2007, pp. 3872–3877.

[8] J. Borenstein, "Experimental results from internal odometry error correction with the omnimate mobile robot," *IEEE Transactions on Robotics and Automation*, vol. 14, no. 6, pp. 963–969, 1998.

[9] A. Martinelli, "The odometry error of a mobile robot with a synchronous drive system," *IEEE transactions on robotics and automation*, vol. 18, no. 3, pp. 399–405, 2002.

[10] F. Gul, W. Rahiman, and S. S. Nazli Alhady, "A comprehensive study for robot navigation techniques," *Cogent Engineering*, vol. 6, no. 1, p. 1632046, 2019.

[11] M. Labbé and F. Michaud, "Long-term online multi-session graph-based splam with memory management," *Autonomous Robots*, vol. 42, no. 6, pp. 1133–1150, 2018.

[12] J. Borenstein and Y. Koren, "The vector field histogram-fast obstacle avoidance for mobile robots," *IEEE Transactions on Robotics and Automation*, vol. 7, no. 3, pp. 278—-288, 1991.

[13] D. Fox, W. Burgard, and S. Thrun, "The dynamic window approach to collision avoidance," *IEEE Robotics & Automation Magazine*, vol. 4, no. 1, pp. 23–33, 1997.

[14] B. Alsadik and S. Karam, "The simultaneous localization and mapping (slam)-an overview," *Surv. Geospat. Eng. J*, vol. 2, pp. 34–45, 2021.

[15] T. Taketomi, H. Uchiyama, and S. Ikeda, "Visual slam algorithms: A survey from 2010 to 2016," *IPSJ Transactions on Computer Vision and Applications*, vol. 9, no. 1, pp. 1–11, 2017.

[16] R. E. Kalman, "A new approach to linear filtering and prediction problems," *Transactions of the ASME - Journal of Basic Engineering*, vol. 82, pp. 35–45, 1960.

[17] M. I. Ribeiro, "Kalman and extended kalman filters: Concept, derivation and properties," *Institute for Systems and Robotics*, vol. 43, p. 46, 2004.

[18] R. Sim, P. Elinas, M. Griffin, J. J. Little *et al.*, "Vision-based slam using the rao-blackwellised particle filter," in *IJCAI Workshop on Reasoning with Uncertainty in Robotics*, vol. 14, 2005, pp. 9–16.

[19] S. Das, "Simultaneous localization and mapping (slam) using rtab-map," *arXiv preprint arXiv:1809.02989*, 2018.

[20] B. Tang and S. Cao, "A review of vslam technology applied in augmented reality," in *IOP Conference Series: Materials Science and Engineering*, vol. 782. IOP Publishing, 2020, p. 042014.

[21] E. Eade and T. Drummond, "Scalable monocular slam," in *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*, vol. 1. IEEE, 2006, pp. 469–476.

[22] R. Mur-Artal, J. M. M. Montiel, and J. D. Tardos, "Orb-slam: a versatile and accurate monocular slam system," *IEEE transactions on robotics*, vol. 31, no. 5, pp. 1147–1163, 2015.

[23] G. Klein and D. Murray, "Parallel tracking and mapping for small ar workspaces," in *2007 6th IEEE and ACM international symposium on mixed and augmented reality*. IEEE, 2007, pp. 225–234.

[24] T. Schops, T. Sattler, and M. Pollefeys, "Bad slam: Bundle adjusted direct rgb-d slam," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 134–144.

[25] F. Endres, J. Hess, N. Engelhard, J. Sturm, D. Cremers, and W. Burgard, "An evaluation of the rgb-d slam system," in *2012 IEEE international conference on robotics and automation*. IEEE, 2012, pp. 1691–1696.

[26] R. Mur-Artal and J. D. Tardós, "Orb-slam2: An open-source slam system for monocular, stereo, and rgb-d cameras," *IEEE transactions on robotics*, vol. 33, no. 5, pp. 1255–1262, 2017.

[27] A. Jurić, F. Kendeš, I. Marković, and I. Petrović, "A comparison of graph optimization approaches for pose estimation in slam," in *2021 44th International Convention on Information, Communication and Electronic Technology (MIPRO)*. IEEE, 2021, pp. 1113–1118.

[28] D. Hähnel, S. Thrun, B. Wegbreit, and W. Burgard, "Towards lazy data association in slam," in *Robotics Research. The Eleventh International Symposium*. Springer, 2005, pp. 421–431.

[29] N. Sünderhauf and P. Protzel, "Towards a robust back-end for pose graph slam," in *2012 IEEE international conference on robotics and automation*. IEEE, 2012, pp. 1254–1261.

[30] G. Grisetti, R. Kümmerle, H. Strasdat, and K. Konolige, "g2o: A general framework for (hyper) graph optimization," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2011, pp. 9–13.

[31] S. Agarwal, K. Mierle, and T. C. S. Team, "Ceres Solver," 3 2022. [Online]. Available: https://github.com/ceres-solver/ceres-solver

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

[33] Wikipedia contributors, "Factor graph — Wikipedia, the free encyclopedia," 2021, [Online; accessed 7-June-2022]. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Factor_graph&oldid=1028331726

[34] G. Grisetti, C. Stachniss, and W. Burgard, "Improved techniques for grid mapping with rao-blackwellized particle filters," *IEEE transactions on Robotics*, vol. 23, no. 1, pp. 34–46, 2007.

[35] S. Kohlbrecher, O. Von Stryk, J. Meyer, and U. Klingauf, "A flexible and scalable slam system with full 3d motion estimation," in *2011 IEEE international symposium on safety, security, and rescue robotics*. IEEE, 2011, pp. 155–160.

[36] Y. Chen, F. Wu, N. Wang, K. Tang, M. Cheng, and X. Chen, "Kejia-lc: a low-cost mobile robot platform—champion of demo challenge on benchmarking service robots at robocup 2015," in *Robot Soccer World Cup*. Springer, 2015, pp. 60–71.

[37] I. Caminal, J. R. Casas, and S. Royo, "Slam-based 3d outdoor reconstructions from lidar data," in *2018 International Conference on 3D Immersion (IC3D)*. IEEE, 2018, pp. 1–8.

[38] M. Labbe and F. Michaud, "Appearance-based loop closure detection for online large-scale and long-term operation," *IEEE Transactions on Robotics*, vol. 29, no. 3, pp. 734–745, 2013.

[39] R. Azzam, F. H. Kong, T. Taha, and Y. Zweiri, "Pose-graph neural network classifier for global optimality prediction in 2d slam," *IEEE Access*, vol. 9, pp. 80 466–80 477, 2021.

[40] R. Kreiser, A. Renner, Y. Sandamirskaya, and P. Pienroj, "Pose estimation and map formation with spiking neural networks: towards neuromorphic slam," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2018, pp. 2159–2166.

[41] I. Lluvia, E. Lazkano, and A. Ansuategi, “Active mapping and robot exploration: A survey,” *Sensors*, vol. 21, no. 7, p. 2445, 2021.