Alma Mater Studiorum · University of Bologna

SCHOOL OF SCIENCE

Master Degree in Computer Science

Simultaneous Localization And Mapping for robots: an experimental analysis using ROS

Supervisor:
Prof.
Davide Maltoni

Candidate:
Matteo Scucchia

Academic year 2020-2021
Session IIII

Abstract

The field of robotics has seen major improvements in the past few decades. One of the most important problem researchers all around the world tried to solve is how to make a mobile robot completely autonomous. One important step to achieve this goal is to create robots that can navigate an unknown environment and, using several sensors, build a map of it, locating themselves on the map. This particular problem takes the name of *Simultaneous Localization And Mapping* (SLAM) and it is very important for different scenarios, such as a mobile robot that navigates an indoor environment, where GPS location cannot perform well. Theoretically, this problem can be considered solved [12], since several solutions have been proposed in the literature, but in practice these solutions perform sufficiently well only under particular condition, such as when the environment is static and its dimension is limited. In real world instead, the environment can change, objects can be moved, and external factors can modify the appearance of a place, making the localization of the robot very uncertain. Therefore, in practice, a long-term SLAM is an unsolved problem and it is an open field of research. A practical problem for which a definitive solution hasn't been proposed yet is the *Loop Closure Detection* (LCD) issue, that is necessary to achieve a real long-term SLAM, and it is the ability of the robot to recognize places previously visited. There are many solutions proposed in the literature, but it is very challenging for a robot to recognize the same place at different time in the day, or in different seasons, or again when the particular location is not visited for long time. During the years, several practice SLAM solutions have been implemented, but a really long-term SLAM hasn't been reached yet. In this thesis a comparison is made between two mature SLAM approaches, highlighting their criticalities and possible improvements in view of a long-term SLAM.

Acknowledgements

I thank my supervisor, Professor Davide Maltoni, for helping me during the last year, in drafting the thesis and for introducing me to the SLAM problem, a topic that really fascinated me and that I found really interesting.

I thank my family, for the support that has never made me lack during these years.

I thank my friends, for the good times spent together, the experiences, the shared knowledge.

I thank my colleagues with which i made a lot of project during these years, in particular thanks to Enrico Gnagnarella.

I thank my Scout group for making me the person I am now.

Finally, I thank all the BioLab team for the support during my time in the laboratory, to all the Professors who have accompanied me during these years and transmitted their knowledge and their passion for knowledge and research.

Contents

1 SLAM: an overview **3**

1.1 Introduction to SLAM 3

1.1.1 Related concepts 4

1.1.2 Why we need SLAM 6

1.1.3 Loop closure detection problem 7

1.2 Formal concepts 9

1.2.1 Mathematical formalization 9

1.2.2 Proposed solutions 11

1.3 Types of SLAM 15

1.3.1 Online SLAM vs Full SLAM 15

1.3.2 Sensor-based classification 16

1.4 Mapping 19

1.4.1 Occupancy grid map 20

1.5 State-of-the-art and future goals 22

1.5.1 A common SLAM framework 22

1.5.2 Loop closure detection: what's next? 24

1.5.3 Lifelong SLAM 24

2 Metrics and Benchmarks for SLAM **27**

2.1 Metrics for SLAM evaluation 27

2.1.1 Traditional visual SLAM metrics 28

2.1.2 Lifelong SLAM metrics 30

2.1.3 Loop closure detection metrics 32

2.2 Classical SLAM datasets and benchmarks 33

2.3 OpenLORIS-Scene dataset 34
2.3.1 Lifelong SLAM challenges 34
2.3.2 Scenes and sequences 35

3 SLAM tools: a comparison between RTABMap and GMapping using ROS 37

3.1 ROS 37
3.1.1 ROS Computation Graph 38
3.1.2 Launch files 40
3.1.3 Rviz 41
3.2 RTABMap 41
3.2.1 How RTABMap works 42
3.2.2 RTABMap memory management 43
3.2.3 RTABMap_ROS 45
3.3 GMapping 46
3.3.1 Mapping with Rao-Blackwellized Particle Filters 46
3.3.2 Rao-Blackwellized SIR filter 47

4 The experiments 49

4.1 The robot 50
4.1.1 Intel Realsense D435 depth camera 51
4.1.2 Xaxxon OpenLidar 52
4.2 RTABMap_ROS registration strategies 54
4.2.1 Visual odometry 54
4.2.2 Lidar odometry 55
4.2.3 Wheel odometry integration 56
4.3 ROS packages 57
4.4 The sequences 57
4.4.1 Sequence 01 57
4.4.2 Sequence 02 58
4.4.3 Sequence 03 59
4.4.4 Sequence 04 60

4.5 Results and evaluation 60
4.5.1 Sequence 01 60
4.5.2 Sequence 02 62
4.5.3 Sequence 03 64
4.5.4 Sequence 04 65

5 Conclusions and future work **67**
5.1 Conclusions 67
5.2 Future work 68

A Technical details of the experiments **71**
A.1 Robot configuration 71
A.2 ROS packages and nodes 72
A.2.1 Sensors activation 72
A.2.2 Robot movement 74
A.2.3 Images acquisition 74
A.2.4 Registration and reproduction of ROS bags 75
A.2.5 Mapping 75
A.2.6 Visualization 76
A.2.7 Map saving 76
A.3 RTABMap_ROS launching parameters 76

List of Figures

1.1 Loop closure detection illustration. 8

1.2 The essential SLAM problem. In the image the symbol k stands for the time instant t. 10

1.3 Graph-based SLAM problem formulation. The robot is in $\mathbf{x}_i$. It moves and it estimates to be in $\mathbf{z}_{ij}$, while the true pose estimated by sensing the world is $\mathbf{x}_j$. The difference between the true pose and the estimated one $\mathbf{e}_{ij}(\mathbf{x}_i, \mathbf{x}_j)$ is the error. The goal is to minimize the error for each node in the graph by using $\hat{\mathbf{z}}_{ij}$ as a constraint. 14

1.4 Comparison between online and full SLAM. In the online SLAM only the map $\mathbf{m}$ and the posterior probability of the last position $\mathbf{x_{t+1}}$ are estimated, given the previous state $\mathbf{x_t}$, and the new control vector $\mathbf{u_{t+1}}$ and the new observation $\mathbf{z_{t+1}}$. In the full SLAM instead, the map $\mathbf{m}$ and the full trajectory of the robot $\mathbf{x_{t-1}}, \mathbf{x_t}, \mathbf{x_{t+1}}$ are estimated, given all the observations and the state vectors. 16

1.5 Time-of-flight technique. A transmitter shots an infrared beam, which bounces on a surface and it is captured by a receiver. Using an electronic clock it is possible to calculate the distance of the surface. . . . 18

1.6 Different types of maps: 2D grid map, 2D topological map, 3D point clouds and 3D meshes [17]. 19

1.7 How occupancy grid map is created. The robot, based on a depth sensor, can perceive and locate the obstacles in the environment. Basing on this information, it can create an occupancy grid map [33]. 21

1.8 Example of an occupancy grid map of an indoor environment. The black pixels of the images are those cells of the environment that are occupied, while the white ones are those that are not occupied. The gray pixels are the unexplored part of the map. 21
1.9 Front-end and back-end SLAM framework. 23

2.1 Graphic representation of the absolute trajectory error [54]. The estimated trajectory is first aligned with the ground-truth. 29
2.2 Graphic representation of the relative pose error [54]. In this case, it is not necessary to align the estimated trajectory with the true one. . 30
2.3 Examples of images of the OpenLORIS-Scene datasets [41]. Each column shows images captured in the same place but in different conditions. 36

3.1 The ROS master-nodes communication. When a new node is instantiated, it registers itself to the master, which responds with the information about the rest of the Computation Graph. From now on, the new node will be able to communicate directly with the others, without going through the master. 39
3.2 The ROS Service communication. It's a classic request/response protocol. 40
3.3 Rviz window. On the left all the topics that Rviz is subscribed to are shown, so that it can receive the information from them. It is possible to subscribe to new topic to receive different types of messages. The right side is a visualization of the 2D occupancy grid map, received from the topic `/map`. 41
3.4 The RTABMap structure [24]. The front-end performs the odometry task, while the back-end is designated to the pose graph optimization and the map assembling. 43

3.5 The RTABMap computational cost of loop closure detection [24]. The time grows linearly with the number of nodes in the map graph, until the Working Memory is full. Then, the computational cost becomes constant, because always the same number of nodes are kept in the Working Memory. 45

4.1 Images of the Cesena campus building. 49
4.2 Xaxxon autocrawler. 51
4.3 Intel Realsense D435 depth camera. 51
4.4 Model of the Xaxxon OpenLidar rotating lidar. 53
4.5 Positioning of the OpenLidar inside the autocrawler. 53
4.6 Sensors and algorithms used by RTABMap to perform visual odometry [25]. It uses the PnP algorithm to estimate the motion of the robot. In the legend, frame-to-frame (F2F) and frame-to-map (F2M) are highlighted. F2F registers the new frame against the last keyframe, while F2M registers the new frame against a local map of features created from past keyframes. 55
4.7 Sensors and algorithms used by RTABMap to perform lidar odometry [25]. It uses the ICP algorithm to register successive laser acquisitions. The legend highlights the S2S and the S2M methods, that are analogous to the F2F and the F2M explained in figure 4.6. 56
4.8 Planimetry of the corridor of the sequence 01. 58
4.9 Planimetry of the corridor of the sequence 02. 59
4.10 Planimetry of the corridor of the sequence 03. 59
4.11 Occupancy grid map of the sequence 01 obtained with GMapping. . . 61
4.12 Sequence 01 occupancy grid map obtained with RTABMap using the registration strategy based on visual features only. 61
4.13 Sequence 01 occupancy grid map obtained with RTABMap using the ICP registration. 62
4.14 Sequence 02 occupancy grid map obtained with GMapping. 63

4.15 Sequence 02 occupancy grid map obtained with RTABMap using the registration strategy based on visual features only. 63
4.16 Sequence 02 occupancy grid map obtained with RTABMap using the ICP registration. 63
4.17 Sequence 03 occupancy grid map obtained with GMapping. 64
4.18 Sequence 03 occupancy grid map obtained with RTABMap using the registration strategy based on visual features only. 64
4.19 Sequence 03 occupancy grid map obtained with RTABMap using the ICP registration. 65
4.20 Sequence 04 occupancy grid map obtained with GMapping. 66
4.21 Sequence 04 occupancy grid map obtained with RTABMap. 66

Abbreviations

SLAM	**S**imultaneous **L**ocalization **A**nd **M**apping
LCD	**L**oop **C**losure **D**etection
RMSE	**R**oot **M**ean **S**quared **E**rror
ATE	**A**bsolute **T**rajectory **E**rror
AOE	**A**bsolute **O**rientation **E**rror
RPE	**R**elative **P**ose **E**rror
RGB	**R**ed **G**reen **B**lue
RGB-D	**R**ed **G**reen **B**lue-**D**epth
VSLAM	**V**isual **S**LAM
VIO	**V**isual **I**nertial **O**dometry
VINS	**V**isual **I**nertial **N**avigation **S**ystem
LiDAR	**L**aser **I**maging **D**etection **A**nd **R**anging
ANN	**A**rtificial **N**eural **N**etwork

Introduction

Although robotics and artificial intelligence have great media coverage today, these fields were born many decades ago, in the '50s. From the beginning, scientists imagined that one day humanity would create fully autonomous robots, capable of acting and perceiving the world as a human being. This ideas is obviously fascinating, not only for the insiders, but for the general public too. Despite the efforts made so far, we are still a long way from this type of technology. To address this purpose, a fundamental requirement is that robots must be able to navigate and understand the world in the same way as humans. "In the same way as humans" means at least as good as humans can do, under the same conditions. If a person is in an unfamiliar place, he is able to navigate there and automatically memorizes the environment during navigation, the topological structure of the place and what is the path to return to an exact point previously visited, being able to recognize known places. To do this, it took millions of years of evolution. In robotics this problem is called SLAM and has only existed for thirty years. This thesis is a dissertation of the SLAM problem, which aims to theoretically describe the problem and shows the results of practical experiments with two well-known SLAM approaches. At the end, some of the most recent advances and the possible future works are discussed, to highlight how the most recent artificial intelligence techniques can bring a great improvement in this field of research.

The chapter 1 introduces the SLAM problem, giving an overall explanation of the arguments and formalizing its mathematical formulation. The chapter 2 introduces some well-known SLAM metrics and datasets for benchmarks, to provide objective methods for evaluating SLAM algorithms. The chapter 3 and 4 focus on the experimental part of this thesis, in which a comparison is made between

two SLAM algorithms. The chapter 5 is the last and discusses the conclusions and considerations on the experiments and possible future work in this research field.

Chapter 1

SLAM: an overview

In this chapter the focus is on the SLAM problem. The first part aims to explain the basic concepts of SLAM, starting from a merely description of them. Then, the mathematical formulation of the problem and a brief summary of the different typologies of SLAM are presented. The final part is a discussion about the state-of-the-art solutions and the future goals.

1.1 Introduction to SLAM

In robotics, SLAM is the problem in which a mobile robot navigating an unknown environment and sensing the world, has to incrementally build a map of the environment in real time, locating itself on the map [12].

The problem is composed of two simultaneous tasks:

- **mapping**: is the task of building a model of the environment, perceiving it with the robot's sensors;
- **localization**: is the task in which the robot has to localize itself on the map.

These two tasks are closely dependent on each other and it is one of the causes that makes SLAM such a difficult problem. In fact, to build a map the robot must know its location, and to know its location it must have a map.

To perceive the environment, the robot could have different sensors such as laser, camera, depth camera and others. At any time the robot senses the environment and

compares the new sensors acquisitions with the previous, determining its movement in the environment and incrementally building the map. The robot's movements and its sensors measurements are affected by noise, so the entire process can become quite inaccurate as the robot moves. Based on the sensors used, there are different types of SLAM and several research fields are involved, such as computer vision, graph theory, information theory and others.

1.1.1 Related concepts

To provide a deeper understanding of how SLAM works, several closely related concepts need to be discussed.

Odometry

The word "odometry" is made up of the Greek words "odos", that which "route" and "metron", which means "measure". It is the use of motion sensor data to estimate the change in position over time [47]. Based on the sensor, there are several types of odometry [1]:

- **wheel odometry**, if the motion is estimated by the encoders that measure the wheel revolutions;
- **visual odometry**, when data are images, typically acquired through a camera;
- **LiDAR odometry**, when data are acquired thanks to the use of laser;
- **inertial odometry**, if the mobile robot has inertial sensors such as gyroscope or accelerometer.

There are also other types of odometry and these types can be combined each other in order to improve the accuracy of motion estimation, such as visual-inertial odometry (VIO). It is important to notice that odometry and SLAM are not the same thing. Odometry can be considered as a part of the SLAM process, in fact its purpose is to estimate the motion between two data acquisitions, while SLAM aims to construct

a consistent global map, so also other processes are involved. As said before, sensors measurements are affected by noise, therefore SLAM would be really inaccurate if in each step only the current odometry information was taken into account and the system didn't have any kind of association with places previously visited.

Landmarks

Landmarks, as the name says, represent important points in the map. The formal definition of "landmark" depends on the sensors used by robot. Imagining a robot that uses only a camera, SLAM is based only on the images that camera captures at any discrete time interval, called *frames*. Thus in this specific case, the definition of what a landmark is, resides in the field of computer vision. In visual SLAM, that is based on images, landmarks are points in the image that are easily recognizable and highly discriminating [15], in order to not confuse different landmarks, causing erroneous localization of the robot on the map. At the beginning of the SLAM process, landmarks, as the map, are typically unknown. As the robot starts mapping the environment, incrementally detects new landmarks. The main concept behind landmarks is that easily recognizable points can be observed from different perspectives, so when robot moves, is really likely that, a few moments later, it will observe the same landmarks again. Under certain assumptions, the more the same landmarks are re-observed, the more accurate the pose estimation becomes. Furthermore, landmarks can be re-observed after a long time enabling the loop closure detection problem, described in the following.

Data association problem

Data association is a recognition process, in which the robot must associate the same observations taken at different time. It is an important task both for the short-term part of SLAM, the odometry, both for the long-term part, the loop closure detection. For what concern odometry, since motion estimation uses the information perceived from the environment, if the robot recognizes that a particular observation made in the previous pose is encountered again, comparing the two observations it is able to estimate the motion performed. For the long-term part, the situation becomes

more complicated. Assuming to have a robot equipped only with a laser sensor, for this robot the world representation is formed by its 2D, or possibly 3D structure. In some cases it is hard for it to understand if, visiting a place, it can be considering a new place or a known location. For example, in an indoor environment that has long corridors, all the same, based on laser acquisition only it is not feasible for the robot to associate a location to a precise point of the map, due to the ambiguity of the environment. Another example is a robot which disposes of a camera and navigates an indoor environment that changes continually over time, due to human interactions. Objects are moved, some objects can be dynamically added and removed, perhaps the lighting conditions change. Thus, associating a particular observation to a place on the map, in a real world case scenario, is very tricky.

1.1.2 Why we need SLAM

In a general sense, the utility of SLAM relies on the specific application's purpose [6]. In an outdoor environment where GPS location performs well enough, if the aim of the application is path planning, given a known map the process is really accurate, just think to Google Maps. On the other hand, when the environment is unknown, or if the robot finds its application in an indoor environment, this particular solution is no more sufficiently accurate. So in a merely practical sense, SLAM is necessary for those applications that require a certain autonomy of the robot, which moves in an unknown environment, typically indoor, or the purpose is exactly to provide an accurate map of an unknown place. As said before, for indoor exploration, SLAM represents the consolidated technique, and it's the starting point for other tasks, such as indoor path planning. From a more philosophical perspective, SLAM provides the robot with a deeper knowledge of the environment than odometry. The robot's ability to recognize previously visited places, reached by following other paths, gives to the robot a topological notion of the world, that is not an "infinite corridor". In addition, odometry alone isn't able to reach appreciable solutions.

1.1.3 Loop closure detection problem

Loop Closure Detection is the process of determining whether an agent is returned to a previously visited location by analyzing data from its sensors [34], that is closely related to data association. In the SLAM problem, it's a fundamental part because thanks to loop closure detection, the accuracy of the map and the localization increases greatly. In fact if a mobile robot recognizes a location previously visited, the map can be adjusted in order to increment its consistency.

As explained earlier, the sensors of a robot are affected by noise, making the motion estimation error prone. This error is accumulated during the exploration and the map building process, causing a discrepancy between the robot estimated pose and the true pose, that grows as the robot moves. This difference is called *drift*. When the robot locates on a known, previously visited place, the map can be adjusted, reducing the localization error. Therefore, effective loop closure detection is essential for the realization of mobile robotic systems capable of long-term autonomy. Figure 1.1 shows the different poses of a mobile robot during its movements, at different time instants, represented by the triangles. The stars, on the other hand, represent the places the robot recognizes as the same place, thanks to data association. On the left there is the true path of the robot, on the right the estimated path. Due to the drift, this second path is quite inaccurate. When in this path the robot revisits the location of the starting point, since data association process is successful, the path can be corrected in order to match the current location with the previous one associated to the same observations. So during the mapping, the real pose of the robot and the estimated one diverge, but when a loop is detected, the pose of the robot can be corrected, causing it to collapse on the same pose previously visited. This process allows to correct the map too, by adapting it to the new path estimate.

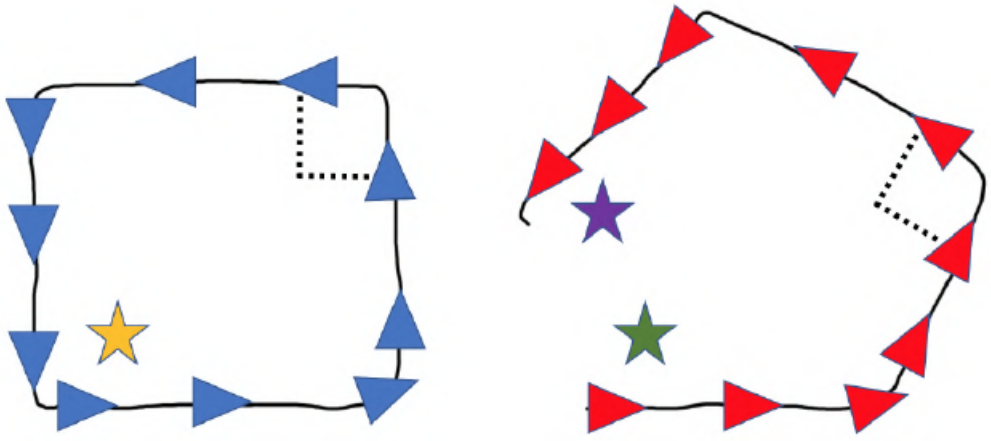


Figure 1.1: Loop closure detection illustration.

Hence, loop closure detection is a powerful strategy for increasing the robustness of SLAM process, but if it's not accurate enough, the consequences could be disastrous. As said before, loop closure detection is strictly related to data association. Inaccurate data association can lead to two scenarios:

- the robot returns to a previously visited place and does not recognize it, so the loop closure detection is missed and the real pose and estimated pose still continue to diverge;

- the robot is in a new location but recognizes it as a previously visited one and the loop closure detection is done, so the position of the robot collapses to the position of the place associated to it [4]. This is obviously the worst condition, because typically when the detection is done it cannot be restored and the future exploration will lead to a wrong mapping and localization.

1.2 Formal concepts

The above description of SLAM concepts is done in high level terms. In order to solve SLAM, a mathematical formalization is needed. It is usually defined in a probabilistic form, as a state estimation problem.

1.2.1 Mathematical formalization

Moving and sensing the world, the robot at any discrete time instant t, changes his state and takes observations of an unknown number of landmarks. All these concepts have the following mathematical formulation:

- $\mathbf{x}_t$, the state vector, represents the pose of the robot. The pose consists of a position and an orientation;
- $\mathbf{u}_t$ is the control vector. Applied at time $t-1$, it moves the robot from state $\mathbf{x}_{t-1}$ to the state $\mathbf{x}_t$;
- $\mathbf{z}_t$ is the observations vector. It corresponds to the sensors measurements. The observation of the i^{th} landmark is $\mathbf{z}_t^i$;
- $\mathbf{m}$ represents the map of the environment. The location of the i^{th} landmark on the map is $\mathbf{m}_i$.

It's possible to define:

- $\mathbf{X}_{0:t} = \{\mathbf{x}_0, \mathbf{x}_1, ..., \mathbf{x}_t\} = \{\mathbf{X}_{0:t-1}, \mathbf{x}_t\}$ the history of robot locations;
- $\mathbf{U}_{0:t} = \{\mathbf{u}_0, \mathbf{u}_1, ..., \mathbf{u}_t\} = \{\mathbf{U}_{0:t-1}, \mathbf{u}_t\}$ the history of robot control inputs;
- $\mathbf{Z}_{0:t} = \{\mathbf{z}_0, \mathbf{z}_1, ..., \mathbf{z}_t\} = \{\mathbf{Z}_{0:t-1}, \mathbf{z}_t\}$ the history of environment observations.

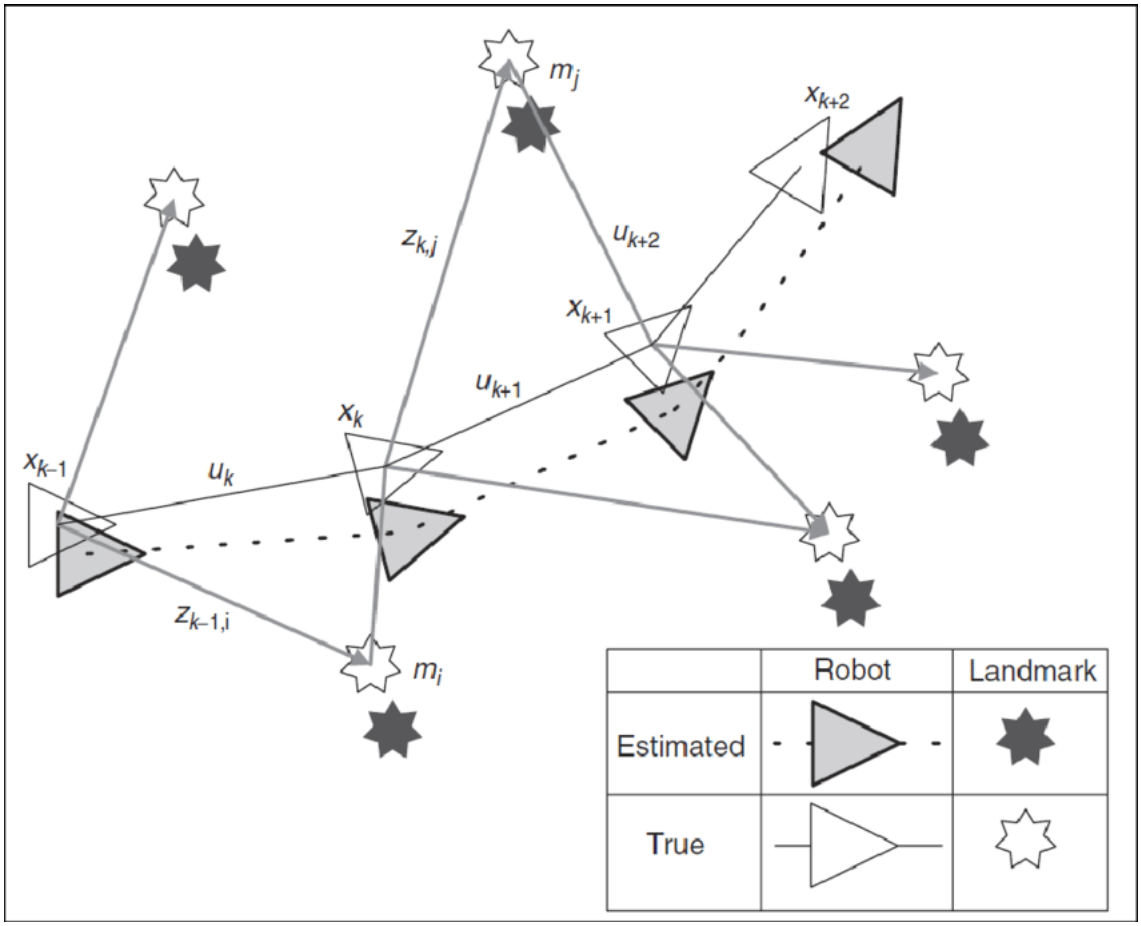


Figure 1.2: The essential SLAM problem. In the image the symbol k stands for the time instant t.

The poses $\mathbf{X}_{0:t}$ and the controls $\mathbf{U}_{0:t}$ are typically defined as 3D euclidean transformations in $SE(3)$ [20], but they can also be a 2D transformation in $SE(2)$ in case of 2D SLAM. A *special Euclidean group* $SE(3)$ is defined as a transformation matrix $\mathbf{T}$, formed by a *special orthogonal group* $SO(3)$, that is a rotation matrix $\mathbf{R}$, and a translation vector $\mathbf{t}$.

$$SO(3) = \{\mathbf{R} \in \mathbb{R}^{3\times3} | \mathbf{R}\mathbf{R}^T = \mathbf{I}, det(\mathbf{R}) = 1\}. \tag{1.1}$$

$$SE(3) = \left\{\mathbf{T} = \begin{bmatrix} \mathbf{R} & \mathbf{t} \\ \mathbf{0}^T & 1 \end{bmatrix} \in \mathbb{R}^{4\times4} | \mathbf{R} \in SO(3), \mathbf{t} \in \mathbb{R}^3 \right\}. \tag{1.2}$$

The initial pose $\mathbf{x}_0$ is the origin and defines the pose of the map. As said before, $\mathbf{u}_t$ applied at time $t-1$ moves the robot from state $\mathbf{x}_{t-1}$ to the state $\mathbf{x}_t$. So it

represents the relative transformation of the robot, that is the rotation and the translation done by the robot at time $t-1$, to evolve from $\mathbf{x}_{t-1}$ to $\mathbf{x}_t$. Instead, $\mathbf{x}_t$ is the pose at time t obtained by applying in cascade to $\mathbf{x}_0$ all the relative transformations from time 0 to t. Hence, in case of 3D SLAM, the pose of the robot is composed by a 3D rotation matrix, which represents the orientation, and a point in the 3D space, which represents the location, with respect to the origin. The map $\mathbf{m}$ can be represented in different ways, depending on several factors, such as the sensors used or the estimation algorithm.

1.2.2 Proposed solutions

Several solutions have been proposed in the literature, depending on the approach used. They can be classified in two main groups: *filtering* and *smoothing* [21].

Filter-based approach

Filtering approaches are the oldest ones and they model the problem as an online state estimation where the state of the system consists in the current robot position and the map. The estimate is augmented and refined by incorporating the new measurements as they become available. This type of approach is so called because it is solved using recursive filters, such as the Kalman filter. In this perspective, the objective is to compute the probability distribution

$$P(\mathbf{x}_t, \mathbf{m} | \mathbf{Z}_{0:t}, \mathbf{U}_{0:t}, \mathbf{x}_0). \tag{1.3}$$

This is a joint posterior density of the state of the robot and the map, given the set of observations and controls and the initial state. Typically, to solve this, a prediction step and a correction step are done recursively. At time $t-1$, starting with an estimate for the probability distribution $P(\mathbf{x}_{t-1}, \mathbf{m} | \mathbf{Z}_{0:t-1}, \mathbf{U}_{0:t-1})$, and given the control vector $\mathbf{u}_t$ and the observations vector $\mathbf{z}_t$, the joint probability at time t can be computed applying Bayes rule. To do this, an *observation model* and a *motion model* are defined.

The observation model represents the probability of making an observation given

the state of the robot and the map, in the form

$$P(\mathbf{z}_t|\mathbf{x}_t, \mathbf{m}). \tag{1.4}$$

The motion model is the probability distribution on state transition, which is the probability of moving in a specific state given the previous state and a command vector, in the form

$$P(\mathbf{x}_t|\mathbf{x}_{t-1}, \mathbf{u}_t). \tag{1.5}$$

In this formulation, the state at time t depends only on the state at time $t-1$, so the process has no memory of the previous states, but only of the previous one and this is called Markov assumption. Instead, $\mathbf{u}_t$ depends neither on observations nor on the state. Now the two step of prediction and correction can be computed as follow.

Prediction step:

$$\begin{aligned} &P(\mathbf{x}_t, \mathbf{m}|\mathbf{Z}_{0:t-1}, \mathbf{U}_{0:t}, \mathbf{x}_0) = \int P(\mathbf{x}_t|\mathbf{x}_{t-1}, \mathbf{u}_t) \\ &\times P(\mathbf{x}_{t-1}, \mathbf{m}|\mathbf{Z}_{0:t-1}, \mathbf{U}_{0:t-1}, \mathbf{x}_0)\, d\mathbf{x}_{t-1}. \end{aligned} \tag{1.6}$$

Correction step:

$$P(\mathbf{x}_t, \mathbf{m}|\mathbf{Z}_{0:t}, \mathbf{U}_{0:t}, \mathbf{x}_0) = \frac{P(\mathbf{z}_t|\mathbf{x}_t, \mathbf{m}) \times P(\mathbf{x}_t, \mathbf{m}|\mathbf{Z}_{0:t-1}, \mathbf{U}_{0:t}, \mathbf{x}_0)}{P(\mathbf{z}_t|\mathbf{Z}_{0:t-1}, \mathbf{U}_{0:t})}. \tag{1.7}$$

It is essential to understand the concept behind these mathematical formulas. At any time, the robot moves. The prediction step tries to estimate where the robot will be after the movement. Obviously, due to the noise, this prediction is very uncertain. In the correction step, the robot senses the environment and thanks to landmarks observations and an accurate data association can correct the prediction, making the new pose estimation more precise.

Smooth-based approach

Smoothing approaches aim to estimate the full trajectory of the robot from the entire set of measurements, so these approaches fall into the full SLAM problem, and they are typically solved with least-square error minimization. Smooth-based SLAM algorithms construct a graph of the map, and for this reason this type of SLAM is commonly called graph-based SLAM. Each node in the graph is a robot pose, while an edge between two nodes corresponds to a spatial constraint relating the two robot poses.

In graph-based SLAM the posterior probability takes into account all the state vectors

$$P(\mathbf{X}_{0:t}, \mathbf{m} | \mathbf{Z}_{0:t}, \mathbf{U}_{0:t}, \mathbf{x}_0). \tag{1.8}$$

Once the graph is constructed the goal is to find the best nodes configuration that minimizes the errors introduced by the constraints. Thus, in this formulation the problem is decoupled in two tasks: graph construction and graph optimization.

The objective of the problem is to determine the most likely constraint given an observation and it depends on the probability distribution over the robot poses. This is exactly the short-term data association problem. In general, it is considered that the observations are affected by Gaussian noise and for this reason the goal is to compute a Gaussian approximation of the posterior probability over the robot trajectory. To do this, it's necessary to compute the mean of the Gaussian, as the configuration of the nodes that maximizes the likelihood of the observations.

In mathematical terms, $\mathbf{z}_{ij}$ and $\mathbf{\Omega}_{ij}$ are respectively the mean and the information matrix of the "virtual measurement" between the node i and the node j. This virtual measurement is a transformation that makes the observations acquired from i maximally overlap with the observation acquired from j, that is in the form of the transformation matrix $\mathbf{T} \in SE(3)$ defined in equation 1.2. So, given a configuration of the nodes $\mathbf{x}_i$ and $\mathbf{x}_j$ and a prediction of the virtual measurement $\hat{\mathbf{z}}_{ij}(\mathbf{x}_i, \mathbf{x}_j)$, the log-likelihood of $\mathbf{z}_{ij}$ is

$$l_{ij} \propto [\mathbf{z}_{ij} - \hat{\mathbf{z}}_{ij}(\mathbf{x}_i, \mathbf{x}_j)]^T \mathbf{\Omega}_{ij} [\mathbf{z}_{ij} - \hat{\mathbf{z}}_{ij}(\mathbf{x}_i, \mathbf{x}_j)]. \tag{1.9}$$

It is possible to define an error function that computes the difference between real observation and expected observation as follow

$$\mathbf{e}_{ij}(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{z}_{ij} - \hat{\mathbf{z}}_{ij}(\mathbf{x}_i, \mathbf{x}_j). \tag{1.10}$$

Now defining C as the set of pairs of indices for which a constraint exists, the goal of the maximum likelihood is to find the optimal configuration of the nodes $\mathbf{x}^*$, in the sense that minimizes the negative log-likelihood $\mathbf{F}(\mathbf{x})$ of all the observations

$$\mathbf{F}(\mathbf{x}) = \sum_{\langle i,j \rangle \in C} \mathbf{e}_{ij}^T \mathbf{\Omega}_{ij} \mathbf{e}_{ij}, \tag{1.11}$$

and therefore the goal is to solve

$$\mathbf{x}^* = \underset{\mathbf{x}}{\operatorname{argmin}} \, \mathbf{F}(\mathbf{x}). \tag{1.12}$$

This equation can be solved with traditional optimization algorithms such as Gauss-Newton or Levenberg-Marquardt.

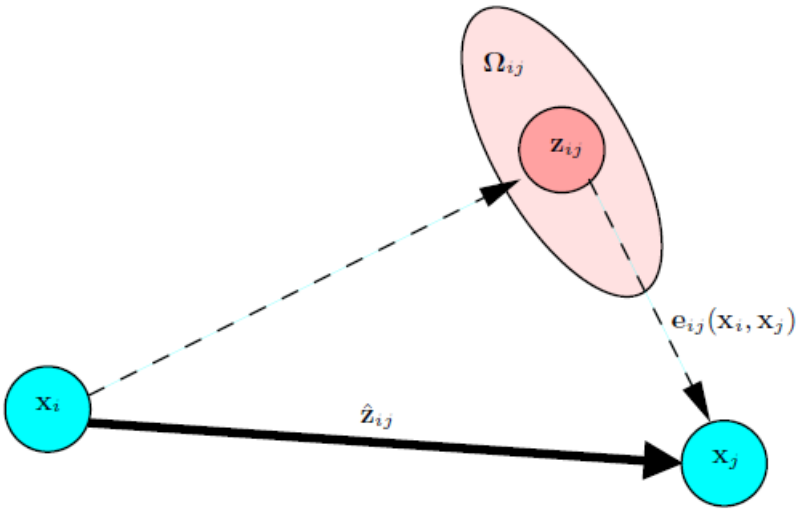


Figure 1.3: Graph-based SLAM problem formulation. The robot is in $\mathbf{x}_i$. It moves and it estimates to be in $\mathbf{z}_{ij}$, while the true pose estimated by sensing the world is $\mathbf{x}_j$. The difference between the true pose and the estimated one $\mathbf{e}_{ij}(\mathbf{x}_i, \mathbf{x}_j)$ is the error. The goal is to minimize the error for each node in the graph by using $\hat{\mathbf{z}}_{ij}$ as a constraint.

Here too, it's important to understand the concepts that lie under the formulas. A robot moves in the environment, it is in the location $\mathbf{x}_i$ and it makes an observation. Then, it moves and it estimates to arrive in $\mathbf{x}_j$, thanks to a motion model that could be uniform rectilinear motion. It makes another observation here and thanks to short-term data association, it recognizes the same observation from different perspective in the two positions and the edge $\hat{\mathbf{z}}_{ij}$ is added to the graph. It can now compute the best transformation in order to overlap the observations in the two poses and applying this transformation to $\mathbf{x}_i$ it results in $\mathbf{z}_{ij}$. So $\mathbf{z}_{ij}$ is the location of $\mathbf{x}_j$ seen from $\mathbf{x}_i$, according to the virtual measurement. The error $\mathbf{e}_{ij}(\mathbf{x}_i, \mathbf{x}_j)$ encodes how much the expected pose differs from the real one. The optimization problem aims to find the best configuration of the graph by minimizing this difference over the entire set of nodes. A motion model is used to estimate the next position of the robot because there may be outliers in the observations that make the transformation estimate very inaccurate and the optimization problem also leads to inaccurate solution. Having an initial guess of the next pose based on a motion model mitigates this bad situation.

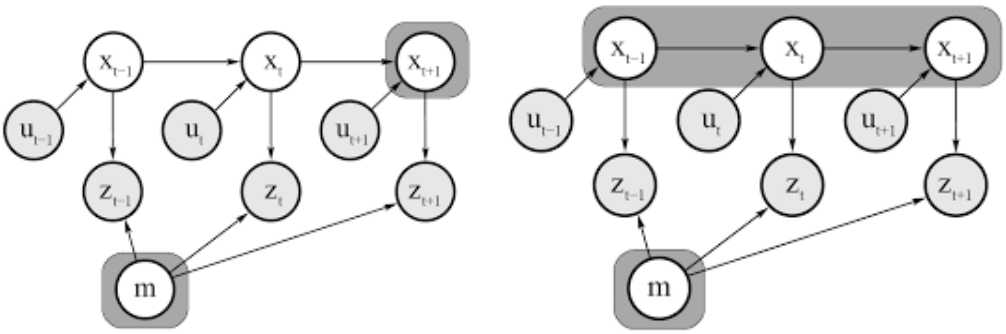
1.3 Types of SLAM

During the years, depending on the problem formulation and the sensors of the robot used, several categories of SLAM have been proposed. The differences between filtering and smoothing solutions have been already explained in 1.2.2, so they won't be discussed again.

1.3.1 Online SLAM vs Full SLAM

Two categories of SLAM problem can be defined, depending on the information computed in SLAM process: *online* and *full* SLAM. The former calculates the posterior probability of the robot's last position only, as in equation 1.3, while the latter estimates the full trajectory of the robot, as in equation 1.8. It is closely related to filtering and smoothing, because the choice of the approach depends on the type of SLAM being considered and vice versa. In the image 1.4 shown below,

the nodes highlighted with a gray padding are the information calculated in the posterior probability.



(a) Online SLAM (b) Full SLAM

Figure 1.4: Comparison between online and full SLAM. In the online SLAM only the map $\mathbf{m}$ and the posterior probability of the last position $\mathbf{x_{t+1}}$ are estimated, given the previous state $\mathbf{x_t}$, and the new control vector $\mathbf{u_{t+1}}$ and the new observation $\mathbf{z_{t+1}}$. In the full SLAM instead, the map $\mathbf{m}$ and the full trajectory of the robot $\mathbf{x_{t-1}}, \mathbf{x_t}, \mathbf{x_{t+1}}$ are estimated, given all the observations and the state vectors.

1.3.2 Sensor-based classification

Another SLAM classification can be made based on the type of odometry used, exposed in 1.1.1. More and more types of sensors exists and can be combined in order to achieve a good SLAM process. Only the most common are presented below.

Visual SLAM

In visual SLAM (VSLAM) the robot is equipped with at least a camera. The odometry between two frames is computed as the transformation needed to best align the two images, and depending on the method used, it can be computed in different ways [39]. In *sparse methods*, the purpose is to find keypoints in the two frames and the odometry represents the transformation needed to overlap the same keypoints belonging to both the images. In *direct methods*, pixels are directly tracked in the two images. Using a monocular camera the entire process relies only on the 2D information of the images. For this reason the process has several limitations, in fact not always the odometry can be accurate without any other information, such as depth information. With a stereo camera, thanks to a binocular vision of

the world and algorithms such as triangulation, 3D information can be obtained, making the process more accurate. Nowadays, RGB-D SLAM is really common and it relies on depth cameras, also called RGB-D cameras. They have three channels for an RGB color image and an additional channel for the depth. The depth channel is typically a 16bit image that is obtained through a laser sensor. Depth information can be measured with two different techniques:

- time of flight (TOF), in which a laser transmitter sends and impulse and a laser receiver captures the impulse after bouncing off an obstacle. The depth is computed on the basis of the time that elapses between the emission and the reception of the impulse;
- structured light, in which a laser projects a known pattern such as a grid on a scene and depending on the deformation of the pattern the 3D structure of the scene can be computed.

The image reconstruction obtained by the fusion of these two information forms the RGB-D images, RGB plus **D**epth, and the SLAM process is typically more accurate than the previous ones. Obviously in some scenario RGB-D is not the best solution. In an outdoor environment with large distances between the objects, the depth information could be not available, due to the maximum range of the laser sensor.

LiDAR SLAM

LiDAR SLAM uses LiDAR odometry to compute the path of the robot. Thanks to laser sensors it reconstructs the spatial structure of the world, using the time-of-flight techniques shown in figure 1.5 and explained in 1.3.2. As for visual SLAM, having the 3D information of two consecutive acquisitions, a transformation can be computed in order to align the same points in the two reconstructions. An advantage of LiDAR based approaches is the fact that it can produce a 2D representation of the environment known as occupancy grid map, which in general is not available from the visual SLAM counterpart, except for the RGB-D SLAM. Occupancy grid

is essential for robot navigation, since path planning algorithms use it as one of the input data along with the starting and final positions [43].

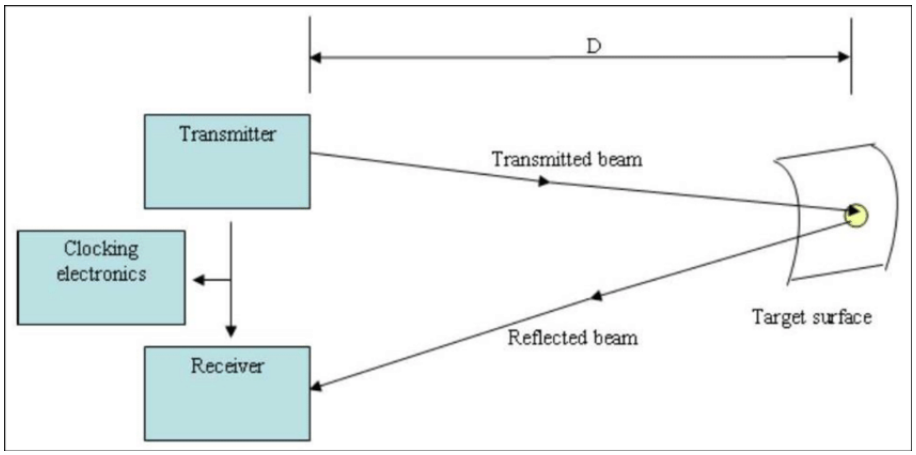


Figure 1.5: Time-of-flight technique. A transmitter shots an infrared beam, which bounces on a surface and it is captured by a receiver. Using an electronic clock it is possible to calculate the distance of the surface.

Visual-Inertial SLAM

Visual-Inertial Navigation Systems (VINS) use the fusion of inertial measurement with the standard visual odometry. Equipping a robot with inertial sensors allows to calculate the transformation between two poses not only relying on external information acquired by sensing the world, but also on internal information, making the process more accurate. Thinking about humans, we obviously have the sight to perceive the world, but we have inertial sensors too. We have a biologic accelerometer that allow us to perceive acceleration, and it's the thing that makes roller coaster so funny. We have some sort of biologic gyroscope too, in fact if we close our eyes and start spinning on ourself, we can easily recognize that we are spinning. But using either biologic inertial sensors and sight, as humans, we can perfectly solve the SLAM problem. This is a good insight to understand that merging more sensory information, obviously in an intelligent way, can lead to more accurate solutions.

1.4 Mapping

Mapping is the process of building a map, that is a representation of the environment. Depending on the SLAM algorithm and the sensors used, different types of maps can be obtained. Some of them are shown in figure 1.6. Maps can be divided into two categories:

- metric maps, that emphasize the exact metrical location of the objects in the map. These can be sparse or dense, depending on the construction of a sparse landmarks representation of the map or on a precise construction of the entire map, respectively. The dense type can be used for navigation;
- topological maps, that emphasize the relationship among map elements. These are graphs composed of nodes and edges, only considering the connectivity between nodes, so they highlight that a first node is connected with a second node, regardless of how to travel from one to the other;

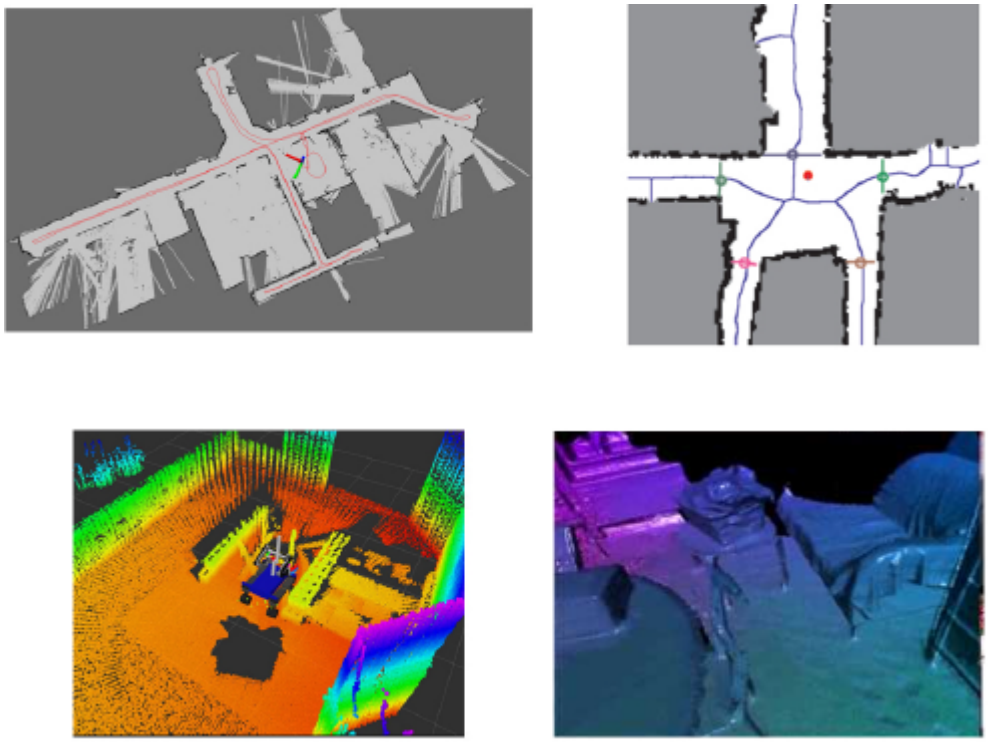


Figure 1.6: Different types of maps: 2D grid map, 2D topological map, 3D point clouds and 3D meshes [17].

1.4.1 Occupancy grid map

As introduced in 1.3.2, a very common type of map obtainable from depth information, such as that provided by a lidar or a depth camera, is the occupancy grid map. This is the type of map built in the experimental part, explained in chapter 4. It's a very important type of map because it is the starting point for the path planning task. An occupancy grid map represents the environment as a block of cells, each one either occupied, so that the robot cannot pass through it, or unoccupied, so that the robot can traverse it [32]. The basic idea of the occupancy grid is to represent a map of the environment as an evenly spaced field of binary random variables each representing the presence of an obstacle at that location in the environment [46]. Depending on the resolution of the grid cells, the map can be more accurate, but also consume more memory. An occupancy grid map can be a 2D or a 3D map, depending on the type of laser sensor used. The most common type of occupancy grid map is the 2D type, because it's one input of the path planning task, and also because it consumes less memory and it is easier to obtain. As explained in section 1.3.2, the LiDAR SLAM is based on a technique called time-of-flight, in which an infrared transmitter shots an impulse and a receiver captures the impulse after it bounced on an obstacle and depending on the time of flight it can compute the distance of the obstacle. The occupancy grid map can be created thanks to this. The environment is composed of discrete cells, the laser shots an infrared beam and depending on the time of flight it discovers the obstacles and their location, as shown in figure 1.7. Since depth-cameras are based on the same technology, they too can be used to obtain a fake laser scan, but they are obviously less accurate and the range is lower than with laser sensors. In this way the cells of the environment in which the obstacles are estimated to be can be set as occupied, while the other are unoccupied. The 2D occupancy grid mapping creates a sort of planimetry of the environment. Having this type of map, a path planning task can be performed, because the robot knows the structure of the environment in which it operates and knows at any moment in which point of the map it is located. The figure 1.8 shows an example of map resulting from the occupancy grid mapping of an indoor environment.

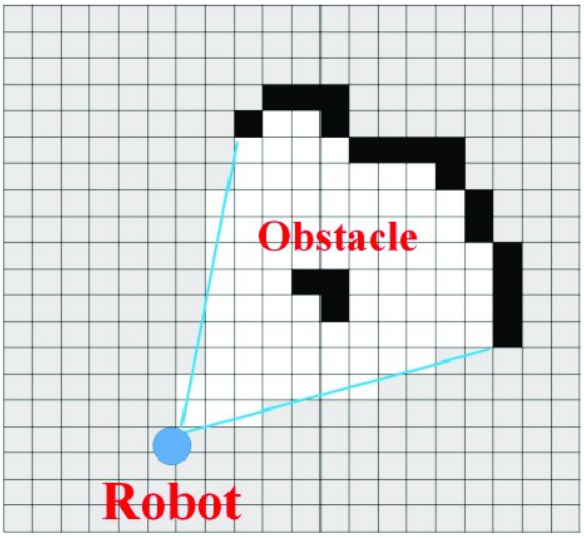


Figure 1.7: How occupancy grid map is created. The robot, based on a depth sensor, can perceive and locate the obstacles in the environment. Basing on this information, it can create an occupancy grid map [33].

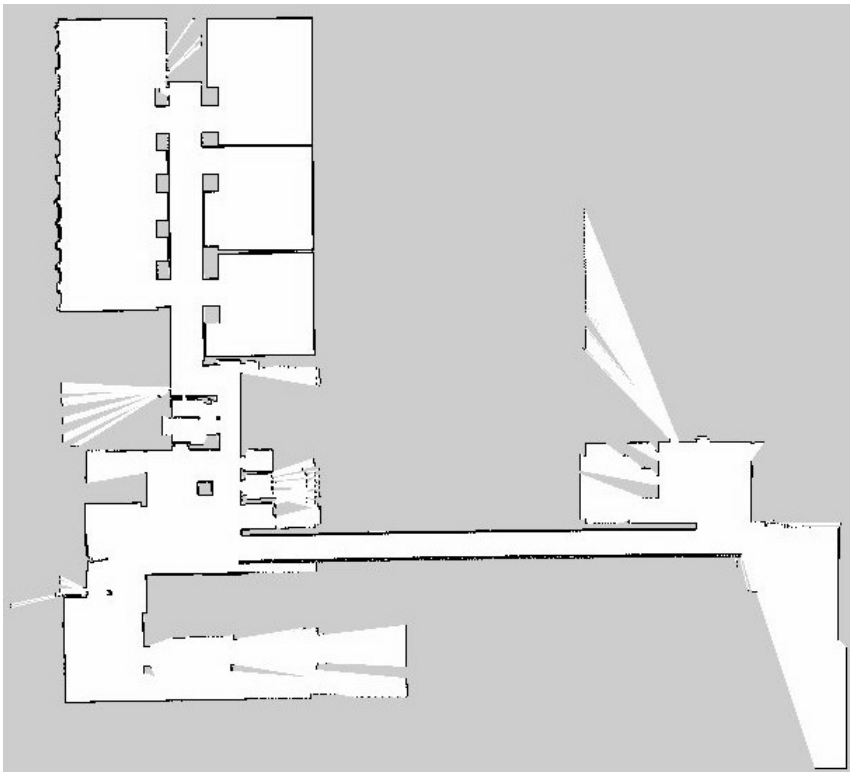


Figure 1.8: Example of an occupancy grid map of an indoor environment. The black pixels of the images are those cells of the environment that are occupied, while the white ones are those that are not occupied. The gray pixels are the unexplored part of the map.

1.5 State-of-the-art and future goals

The past decade has been a good time for SLAM. Several open source solutions have been implemented and the community was able to study and compare them. The state-of-the-art of long-term SLAM typically relies on the fusion of different sensors and visual or LiDAR SLAM based on graph optimization explained in 1.2.2 are very common starting points nowadays. In order to improve the exchange of ideas and test the actual SLAM state-of-the-art, several competitions were born around the world. In 2019 in the International Conference on Intelligent Robots and Systems (IROS), an international forum for the robotics research community [35], took place a long-term SLAM competition where the winner solution was based on visual-inertial SLAM with a wheel odometry enhancing [27]. In the following, a very common framework for graph-based SLAM will be discussed, because it is used in many solutions that are the actual state-of-the-art and, presenting it, a deeper insight about the SLAM process is provided, putting together all the concepts explained so far. Finally, a brief summary about loop closure detection problem and possible future improvements are presented and the lifelong SLAM will be introduced as it is the main future goals of this field of research.

1.5.1 A common SLAM framework

Depending on the sensors and the approach used, several differences arise between SLAM frameworks. A proposed framework for graph-based SLAM plans to split the process into two parts: the *front-end* and the *back-end* [31], as shown in figure 1.9. The first one is strictly dependent on the sensors employed and its main purpose is to build the graph of the map incrementally, creating the virtual measurements for the second one, which is sensor-agnostic and has the task of optimizing the graph. More precisely, the front-end computes the odometry performed by the robot, thanks to the short-term data association, builds an high level representation of the map, depending on the sensors available and the application's purpose, and detects loops with the long-term data association. In case the robot is equipped with multiple sensors, the information acquired are synchronized and fused together

to create more accurate virtual measurements. The front-end should be very fast to do it in real time and without losing any acquisition from the sensors. The back-end continually receives the virtual measurements computed by the front-end and incrementally optimizes the graph. It runs simultaneously with the front-end but it is slower, because the optimization part requires more time for the computation. Once the optimization is done, the front-end estimated pose and the high level map are adjusted according to the back-end optimization in order to increase the accuracy of the localization and mapping.

The aim is therefore to separate the sub-processes that form the SLAM in such a way that the fast, sensor-dependent operations are divided from the slower optimization processes that do not depend on the sensors. Hence, this framework is very flexible and adaptable, and the graph-based approach ensures that it is also efficient and accurate enough. For these reasons, it's the basic framework for several solutions that are the state-of-the-art of SLAM nowadays, typically based on visual SLAM.

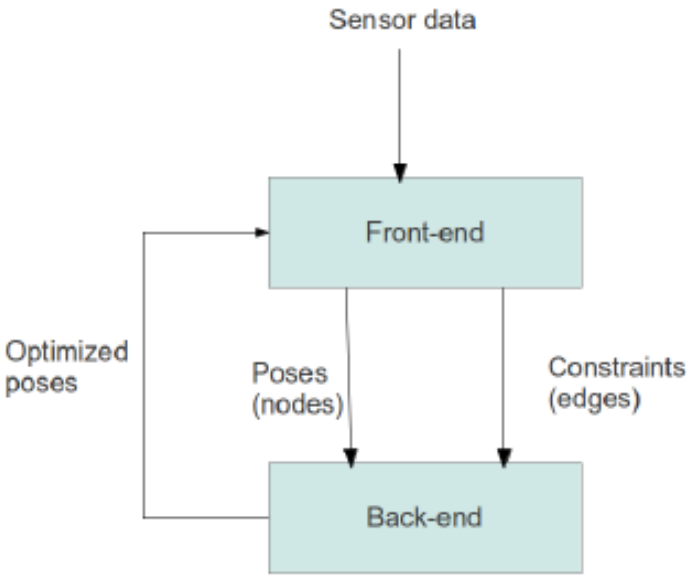


Figure 1.9: Front-end and back-end SLAM framework.

1.5.2 Loop closure detection: what's next?

Loop closure detection is only partially solved in practice, because the robustness of data association isn't perfect and the real time constraint prevents a brute-force matching with the entire set of all the previous observations. Typically two possibilities are followed in order to deal with these problems:

- the system tries to detect loops taking into account all the previous observations, but with a large environment the real time constraint is violated;
- the system tries to detect loops in a probabilistic way, taking into account only a set of previous observations, and the accuracy of the detection decrease depending on the environment and the robot exploration.

Hence, there is a strict trade off between the robustness and the scalability of the process. In the state-of-the-art implementations, loop closure detection is quite accurate on medium size environments, and this accuracy depends on the sensors accuracy. Visual SLAM is, in the most of the cases, more robust to loop closure detection, because visual features are clearly more discriminant and recognizable than spatial features, but often it depends on the environmental conditions. In these solutions, the loop closure detection relies on the correspondence of some types of observations with the same observations previously done and stored in efficient data structures, in order to speed up the comparison. These techniques could be improved in future, taking into account more information, for example not only the observations acquired from the environment but also odometry information, or by the application of artificial intelligence techniques such as artificial neural networks to improve the efficiency of features memorization and the spatial and semantic knowledge of the environment. These two future works are discussed in chapter 5.

1.5.3 Lifelong SLAM

Lifelong SLAM is a term coined to indicate a long-term SLAM, capable of facing a huge map, with an environment that changes over time, in order to give a mobile robot some autonomy. It is an open field of research, because a definitive solution in

a practical sense has not yet been reached. In literature are described two properties that characterize lifelong SLAM which have already been introduced: *robustness* and *scalability*.

Robustness

It refers to the ability of deal with failures. It can be linked to the algorithms, the software part, and to the sensors, the hardware part, in long-term scenario. For what concerns the algorithmic part, one of the most crucial problem is data association. In 1.1.1 is explained what it is and thinking about data association problem in an high level sense, it is easy to understand its complexity in a long-term situation. As explained just before, there are situations in which if the robot bases its knowledge of the environment only on a set of fixed visual landmarks, it's obvious that the algorithm isn't robust enough to face real world conditions. In these cases, it could happen that the robot isn't able to associate itself in a previously visited location, or worse it confuses two different places, localizing itself on the wrong location.

Hence, robustness is guaranteed when the data association is truly accurate or, if not, errors can be handled correctly, for example making data association a reversible process, where if an error is detected the map can be restored and corrected. For these reasons, robustness is essential to build a consistent map in a long-term scenario.

Scalability

In general, it is defined as the property of a system to handle a growing amount of work by adding resources to the system [48]. In SLAM, it is the ability of the algorithms to map a huge environment for a long time, without performance degradation. Thinking about the construction of the map, it is obvious that it grows as the robot navigates new places. In a life-long scenario, the growth of the map is unbounded, in the sense that it can grow indefinitely. Thus, the memory used for the map, the landmarks and the poses of the robot increases over time, and also increases the time to compare the previous observations with new ones, to detect loops. In order to make SLAM solutions more scalable, several approaches have been

proposed, such as efficient representations of the map, that reduce the memory used for the storage, or better optimization algorithms to address larger map representations, or take advantage of parallel programming for the calculation. A related open question for long-term mapping is how often to update the information contained in the map and how to decide when this information becomes outdated and can be discarded. So it is essential to learn the map, but also forget the old configurations of the environment, which can be changed in the meantime. Furthermore, it is clear that keeping the entire map loaded in working memory at any time is a waste of resources, so an intelligent way to offload parts of the map and then reload them when needed has to be found.

Chapter 2

Metrics and Benchmarks for SLAM

Evaluating the SLAM process is not easy. To provide an objective measure of the performance of a SLAM algorithm, formal metrics are needed. Typically a ground-truth is required, that is, the correct pose of the robot at all time, which is obtained by some higher precision system. Benchmarks are computed by aligning the timing of the ground-truth sequence and the estimated sequence and then comparing the estimate pose with the ground-truth, but there are methods to evaluate SLAM also when a ground-truth is not provided [13]. For this purpose, different datasets have been collected and different metrics have been proposed in the literature, depending on the type of SLAM. This chapter introduces traditional SLAM metrics and new metrics specifically designed to assess long-term SLAM. Next, the most common traditional SLAM datasets for benchmarks are presented and the OpenLORIS-Scene dataset is explained, as it was created specifically for evaluating lifelong SLAM algorithms.

2.1 Metrics for SLAM evaluation

The metrics originally introduced for visual SLAM are explained below, which today find application in many different solutions and are the starting point for new proposals for lifelong SLAM. Very common metrics are then described for evaluating loop closure detection .

2.1.1 Traditional visual SLAM metrics

Two traditional metrics have been proposed in the literature to evaluate visual SLAM systems: the *absolute trajectory error*, and the *relative pose error*. There are useful tools for SLAM benchmarks that calculate these metrics given the estimated trajectory and the ground-truth trajectory, as for example the scripts created by Computer Vision Group of the Department of Informatics of Munich [9] for the RGB-D SLAM benchmark.

Two sequences of poses are provided for the evaluation, the estimated trajectory $\mathbf{P}_1, ..., \mathbf{P}_n \in SE(3)$ and the ground-truth trajectory $\mathbf{Q}_1, ..., \mathbf{Q}_n \in SE(3)$.

Absolute trajectory error

The absolute trajectory error (ATE) measures the global consistency of the estimated trajectory of the robot [44]. It is evaluated by comparing the absolute distances between the estimated and the ground-truth trajectory. To determine a correct ATE, it is necessary to align the coordinate system of the estimated trajectory and the ground-truth using a rigid body transformation $\mathbf{S}$, corresponding to the least-squares solution that maps the estimated trajectory $\mathbf{P}_{1:n}$ onto the ground-truth trajectory $\mathbf{Q}_{1:n}$. At time step k the ATE can be calculated as

$$\mathbf{F}_k := \mathbf{Q}_k^{-1}\mathbf{S}\mathbf{P}_k. \tag{2.1}$$

In visual SLAM, the absolute trajectory error on the entire path is given by the RMSE over all time indices of the translational components as

$$RMSE(\mathbf{F}_{1:n}) = \left(\frac{1}{n} \sum_{k=1}^{n} ||trans(\mathbf{F}_k)||^2 \right)^{1/2}. \tag{2.2}$$

The figure 2.1 shows a visual representation of the absolute trajectory error.

In the most recent metrics, presented in 2.1.2, also the *absolute orientation error* (AOE) is considered, and it can be calculated in the same way as the ATE,

considering the orientational components, as shown in equation 2.3.

$$RMSE(\mathbf{F}_{1:n}) = \left(\frac{1}{n} \sum_{k=1}^{n} ||orientation(\mathbf{F}_k)||^2 \right)^{1/2}. \tag{2.3}$$

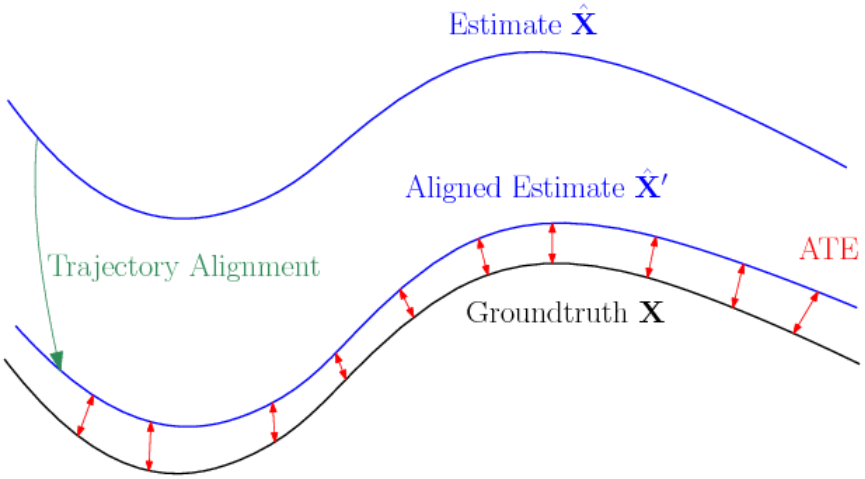


Figure 2.1: Graphic representation of the absolute trajectory error [54]. The estimated trajectory is first aligned with the ground-truth.

Relative pose error

The relative pose error (RPE) measures the local accuracy of the trajectory over a fixed time interval Δ [44]. Thus, the relative pose error corresponds to the drift of the trajectory. Contrary to the ATE, it doesn't require the global rigid alignment, because the discrepancy between corresponding poses in the estimated trajectory and the ground-truth is the error itself. The RPE at time step k is defined as

$$\mathbf{E}_k := \left(\mathbf{Q}_k^{-1} \mathbf{Q}_{k+\Delta} \right)^{-1} \left(\mathbf{P}_k^{-1} \mathbf{P}_{k+\Delta} \right). \tag{2.4}$$

If the ground truth part and the estimated part are identical, $\mathbf{E}_k$ is the identity matrix. So, given a sequence of n poses, $m = n - \Delta$ are the individual relative pose errors along the sequence and as for the ATE, the RMSE is computed on the translational components as follows

$$RMSE(\mathbf{E}_{1:n}, \Delta) = \left(\frac{1}{m} \sum_{k=1}^{m} ||trans(\mathbf{E}_k)||^2 \right)^{1/2}. \tag{2.5}$$

It is worthy to notice that changing the Δ value leads to compute the drift over a different number of poses. For example with $\Delta = 1$ the error measures the drift per frame.

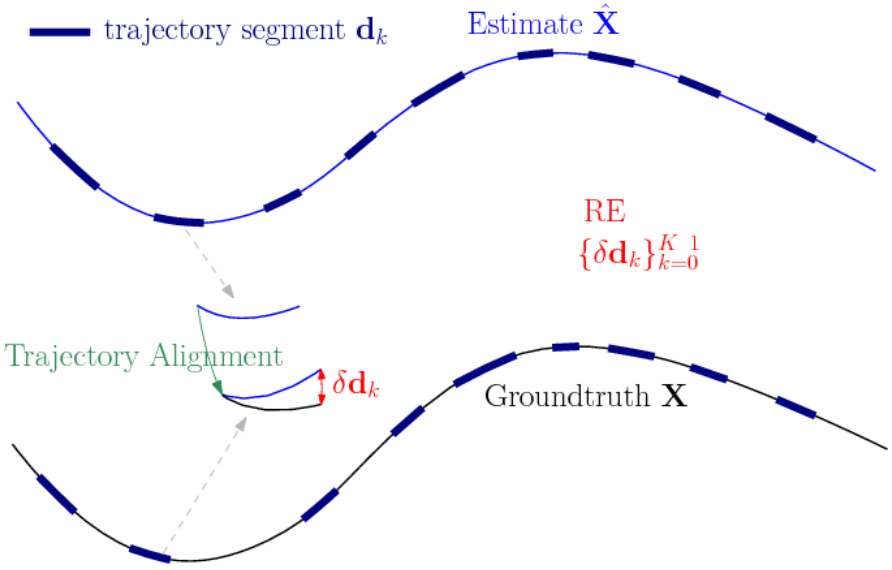


Figure 2.2: Graphic representation of the relative pose error [54]. In this case, it is not necessary to align the estimated trajectory with the true one.

2.1.2 Lifelong SLAM metrics

As explained in 1.5.3, lifelong SLAM is theoretically characterized by two important properties: robustness and scalability. The second one is not strictly related to the classic SLAM metrics, in the sense that it is based more on memory occupation and time complexity than on the accuracy of SLAM process. Two types of long-term SLAM metrics have recently been proposed in the literature: robustness metrics and accuracy metrics [41].

Robustness metrics

To evaluate the robustness, several measures are defined.

Correctness

For each estimated pose $\mathbf{p}_k$ at time $\mathbf{t}_k$, given the ground-truth pose at that time, the correctness of the estimate is computed from its ATE and AOE as follow

$$c^{\varepsilon,\phi} = \begin{cases} 1, & \text{if ATE}(\mathbf{p}_k) \leq \varepsilon \text{ and AOE}(\mathbf{p}_k) \leq \phi \\ 0, & \text{otherwise} \end{cases} \tag{2.6}$$

where ε is the ATE threshold and ϕ is the AOE threshold. They should be choosen according to the area of the scene and the expected drift of the algorithm. So the correctness is an on-off indicator that evaluates the precision of a single pose, comparing the errors with the two thresholds.

Correct Rate (CR) and Correct Rate of Tracking (CR-T)

These measures evaluate the overall robustness over one or more data sequences, that is the correct rate over the whole time span of data. Given a sequence from $\mathbf{t}_{min}$ to $\mathbf{t}_{max}$ and the estimated-true trajectory $\{\mathbf{t}_k, \mathbf{p}_k\}_{k=0,\dots,N}$, it is possible to define

$$\text{CR}^{\varepsilon,\phi} = \frac{\sum_{k=0}^{N}(min(\mathbf{t}_{k+1} - \mathbf{t}_k, \delta) \cdot c^{\varepsilon,\phi}(\mathbf{p}_k))}{\mathbf{t}_{max} - \mathbf{t}_{min}}, \tag{2.7}$$

$$\text{CR}^{\varepsilon,\phi}\text{-T} = \frac{\sum_{k=0}^{N}(min(\mathbf{t}_{k+1} - \mathbf{t}_k, \delta) \cdot c^{\varepsilon,\phi}(\mathbf{p}_k))}{\mathbf{t}_{max} - \mathbf{t}_0}, \tag{2.8}$$

where $\mathbf{t}_{N+1} \doteq \mathbf{t}_{max}$ while δ is a parameter to determine how long a correct pose estimation is valid for and it should be set larger than the pose estimation time, but much smaller than the time span of the sequence. Typically it is set to one second. The insight here is that robustness is calculated as the sum of the correctness of each pose multiplied by the time the pose $\mathbf{p}_k$ is correct in the sequence, that is $min(\mathbf{t}_{k+1} - \mathbf{t}_k, \delta)$, normalized by the total time span of the sequence. In $\text{CR}^{\varepsilon,\phi}$-T the time for re-localization and the initialization $(\mathbf{t}_0 - \mathbf{t}_{min})$ is excluded, because tracking does not work during that time.

Correctness Score of Re-localization (CS-R)

This measure evaluates the re-localization score of the robot. Re-localization is the task in which the robot must recognize its location on the map. This metric takes

into account also how much time the re-localization takes and it's defined as follow

$$\mathrm{C}^{\varepsilon,\phi}\mathrm{S}^{\tau}\text{-R} = e^{-(\mathbf{t}_0 - \mathbf{t}_{min})/\tau} \cdot c^{\varepsilon,\phi}(\mathbf{p}_0) \tag{2.9}$$

where τ is a scaling factor, typically set to 60 seconds. It can be noticed that if the re-localization is instantaneous, so that $\mathbf{t}_0 = \mathbf{t}_{min}$, CS-R = 1.

Accuracy metrics

Regarding accuracy, the proposal is to use ATE and RPE statistics on one or more sequences taking into account only correct estimations of the pose. For example, $\mathrm{C}^{0.1}$-RPE RMSE is the root mean square error of RPE of correct pose estimates selected from an ATE threshold of 0.1 meter.

2.1.3 Loop closure detection metrics

Accuracy alone is not sufficient to evaluate loop closure detection, because, depending on the path and the environment, this metric could not provide representative values. There could be the case in which a particular solution never detect a loop, and another case in which there are several similar but distinct places in the environment and the algorithm always detect a loop when moving between these places. For these reasons, the metric used to evaluate loop closure detection is *Precision-Recall*. Precision-Recall is a useful measure of success of prediction when the classes are very imbalanced [36]. In fact in an environment there could be a lot of loop to detect, in another there aren't loop closure, but only very similar places. Thanks to this metric, the results are more meaningful with respect to the accuracy alone, because it takes into account not only the true positive and the false positive detection, but also the true negative and the false negative.

Precision

The Precision is a measure of result relevancy. It identifies how many true positive there are between all the positive detected.

Precision (P) is defined as the number of true positives (T_p) over the number of true positives plus the number of false positives (F_p).

$P = \frac{T_p}{T_p + F_p}$.

For loop closure detection, it is the proportion of the detection returned that are true loops.

Recall

The Recall is a measure of how many truly relevant results are detected. It identifies how many true positive are detected among all the positive to detect.

Recall (R) is defined as the number of true positives (T_p) over the number of true positives plus the number of false negatives (F_n).

$R = \frac{T_p}{T_p + F_n}$.

For loop closure detection, it measures how many true loop closures are detected among all those present in the explored environment.

2.2 Classical SLAM datasets and benchmarks

There are several known datasets for SLAM benchmarking in literature. Most of them are specific for a type of SLAM, for example the TUM RGB-D dataset [44] is specific for RGB-D SLAM. Three of the best known and largely used datasets are:

- the *KITTI odometry dataset* [18] which consists of 22 stereo sequences of road scenes, captured with two synchronized cameras installed on the roof of a car. For the first 11 sequences the ground-truth is provided, while the last 11 sequences have no ground-truth and are used for evaluation. For this benchmark it is possible to use monocular or stereo visual odometry, laser-based SLAM or algorithms that combine visual and lidar information;

- the *EuRoC MAV dataset* [5] which contains 11 sequences of 20 Hz stereo images collected on-board a Micro Aerial Vehicle (MAV) in small interior rooms and

in a machine room. Camera-synchronized IMU data is also available and an accurate ground-truth is provided;

- the *TUM RGB-D dataset* which consists of 39 sequences that have been recorded in two different indoor environments. Each sequence contains the color and depth images, as well as the ground-truth captured thanks to an high-accuracy motion-capture system with eight high-speed tracking cameras at 100 Hz. Images were recorded at 30 Hz using handheld Kinect v1.

Method	Real data	RGB-D	Stereo	Global shutter	Synched cameras	IMU	Accurate GT	Benchmark
TUM RGB-D	X	X				(1)	X	(2)
EuRoC MAV	X		X	X	X	X	X	
KITTI	X	(3)	X	X	X	X		X

Table 2.1: comparison of the three SLAM datasets [40]. Notes on numbered entries: (1) Accelerometer but not gyroscope measurements are available. (2) While this dataset has a test set, it is not well suited for benchmarking since it shows the same scenes as the training set, and there is no online leaderboard. (3) Sparse measurements of a spinning laser scanner are available.

2.3 OpenLORIS-Scene dataset

The OpenLORIS-Scene dataset was presented in 2019 at the annual International Conference on Intelligent Robots and Systems (IROS). This dataset was created specifically for the research of lifelong SLAM for service robots. The lifelong SLAM problem has already been described in 1.5.3. Here, a more in-depth explanation of the challenges and the reasons behind the creation of the OpenLORIS-Scene dataset are discussed.

2.3.1 Lifelong SLAM challenges

Contrary to a classical concept of SLAM, in which robots create the map of the environment and it is evaluated as it is, in the lifelong SLAM the robots need to navigate the map day after day, with the requirement of reusing and modifying the map if needs. Real-life scene changes and other uncontrolled factors in a long-term scenario pose significant challenges to SLAM algorithms.

"For a robot that needs to operate around a particular region over an extended period of time, the capability of lifelong SLAM aims to build and maintain a persistent map of this region and to continuously locate the robot itself in the map during its operations." [41]

The major challenges for a lifelong SLAM can be summarized as follow:

- changed viewpoints, so the robot may see the same objects or scene from different directions;
- changed things, because many objects may have been changed when the robot re-observe a previously visited area;
- changed illumination, because depending on the time of the day the illumination of a scene may change a lot;
- dynamic objects, because objects may be moved in a scene;
- degraded sensors, because they may be out-of-calibration due to mechanical stress, temperature change, dirty or wet lens, or just have unpredictable noise.

To address all these challenges, OpenLORIS-Scene dataset was captured in a dynamic environment, with people in it, and there are multiple sequences for each scene, with changes in lighting and viewpoints and dynamic objects due to human activities.

2.3.2 Scenes and sequences

To capture the sequences, a robot equipped with a rich combination of sensors was used, including RGB-D, stereo fisheyes, inertial sensors, wheel odometry and lidar, which can facilitate comparison between algorithms with different types of inputs. There are five scenes in the dataset and for each there are 2-7 data sequences recorded at different times. The scenes are summarized as follow:

- *Office*, consisting of seven sequences captured in an university office;
- *Corridor*, consisting of five sequences captured in a long corridor with a lobby in the middle and the above office at one end;

- *Home*, consisting of five sequences in two bedroom apartments;
- *Cafe*, consisting of two sequences in an open café;
- *Market*, consisting of three sequences in an open supermarket;

Figure 2.3 shows an example of the scenes of the dataset.

Figure 2.3: Examples of images of the OpenLORIS-Scene datasets [41]. Each column shows images captured in the same place but in different conditions.

For each sequence of each scene the ground-truth of the robot poses is provided. It was obtained using more accurate systems, such as a Motion Capture System (MCS) or with other SLAM method such as 2D lidar SLAM.

Chapter 3

SLAM tools: a comparison between RTABMap and GMapping using ROS

During the years, several SLAM solutions have been developed by the community as open source software. These tools run different types of SLAM, such as LiDAR and Visual SLAM and more complete solutions combine multiple types of SLAM. This chapter presents an in-depth explanation of two well-known SLAM solutions, RTAB-Map and GMapping. The goal of this chapter is to explain the technologies used in the experiments and in particular these two well-known software that run different SLAM algorithms because they represent the actual state-of-the-art of open source SLAM solutions and for this reason they are used in the experiment presented in the next chapter.

3.1 ROS

The Robot Operating System (ROS) 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 [37]. ROS is a distributed framework of processes called nodes that enables executables to be individually designed and loosely coupled at runtime. Nodes can run on differ-

ent machines and the communication is based on message exchange. The choice to implement ROS as a distributed framework, composed by nodes, bring several advantages:

- code reusability, in fact the same node can be used by different application;
- modularity, because a node should solve a single problem, so each of them is potentially independent from the others;
- interoperability, because it's not important in which language a node is written or on which machine it is running, it can interoperate with other nodes, written in a different language and running on a different machine.

Of course, this design also has its drawbacks. In particular, the message exchange mechanism is slower than memory sharing.

3.1.1 ROS Computation Graph

The Computation Graph is the peer-to-peer network of ROS processes which manage data and computation together. The basic concepts of the Computation Graph are introduced below.

Master

Since ROS is a distributed framework, it needs a node that initiates and coordinates the communication between the other nodes. The ROS master is a node that provides name registration and lookup to the rest of the Computation Graph. Without the master, the nodes would not be able to find each other, exchange messages, or invoke services. There can be only one master node in a peer-to-peer network.

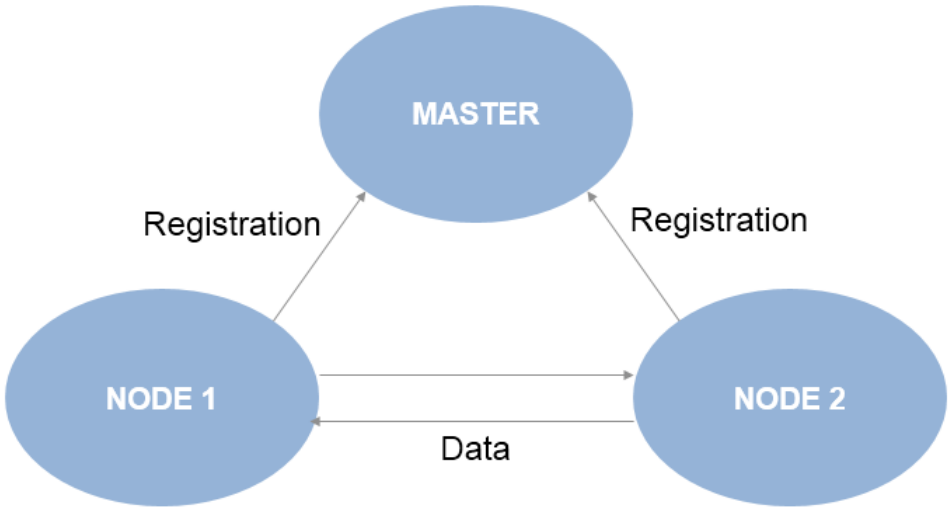


Figure 3.1: The ROS master-nodes communication. When a new node is instantiated, it registers itself to the master, which responds with the information about the rest of the Computation Graph. From now on, the new node will be able to communicate directly with the others, without going through the master.

Nodes

Nodes are processes that perform computation. They should be fine-grained and specific to solving a single task. For example, in a simple SLAM application, a node can be used for the visual odometry, a node for the LiDAR odometry, a node for the fusion of the two odometries, a node for assembling the map and so on.

Messages

Nodes communicate with each other by passing messages. A message is simply a data structure, comprising typed fields, with .msg extension. Complex messages are composed hierarchically from fixed types, such as primitive types and arrays. There exists a lot of message types already implemented in ROS, because they are the most commonly used.

Topics

Messages are routed via a transport system with publish/subscribe semantics. A node sends out a message by publishing it to a given topic. The topic is a name

that is used to identify the content of the message. A node that is interested in a certain type of data can subscribe to the appropriate topic.

Services

Services are structures appropriate for request/response interactions. Services are defined by a pair of message structures: one for the request and one for the reply, with extension .srv.

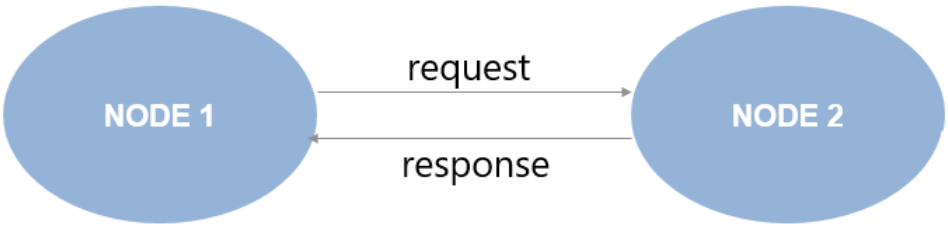


Figure 3.2: The ROS Service communication. It's a classic request/response protocol.

ROS Bags

Bags are a format for saving and playing back ROS message data. They are an important mechanism for storing data, such as sensor data, which can be difficult to collect but are needed for developing and testing algorithms. It's a very fundamental concept of ROS, because ROS bags allow to store sensor data and reuse them in a second moment to test the algorithms. In this way different algorithms can be tested on the same exact data, without any condition variation, making the evaluation very accurate and faithful. ROS bags are used in the experiment to acquire data from the robot's sensors.

3.1.2 Launch files

A very useful ROS package is the *roslaunch* package. It is used to manage the .launch files of a package, which are files written in XML format used to launch multiple nodes in a single shot without repeating the same ROS command. It is also possible to structure the launch files in a hierarchical way, so that a launch file can run other launch files and here it is possible to set the parameters for the ROS

nodes, so it is very useful for the customization of the algorithms. An example of .launch file used in the experiment is shown in appendix A.2.1.

3.1.3 Rviz

Rviz is a built-in ROS package that provides a graphical interface to visualize the map and the topics involved in the process. Figure 3.3 shows the Rviz window. From the graphical window it is possible to choose which topics to subscribe to, in order to receive messages from them and view information. In appendix A.2.6 the Rviz configuration used in the experiment is explained.

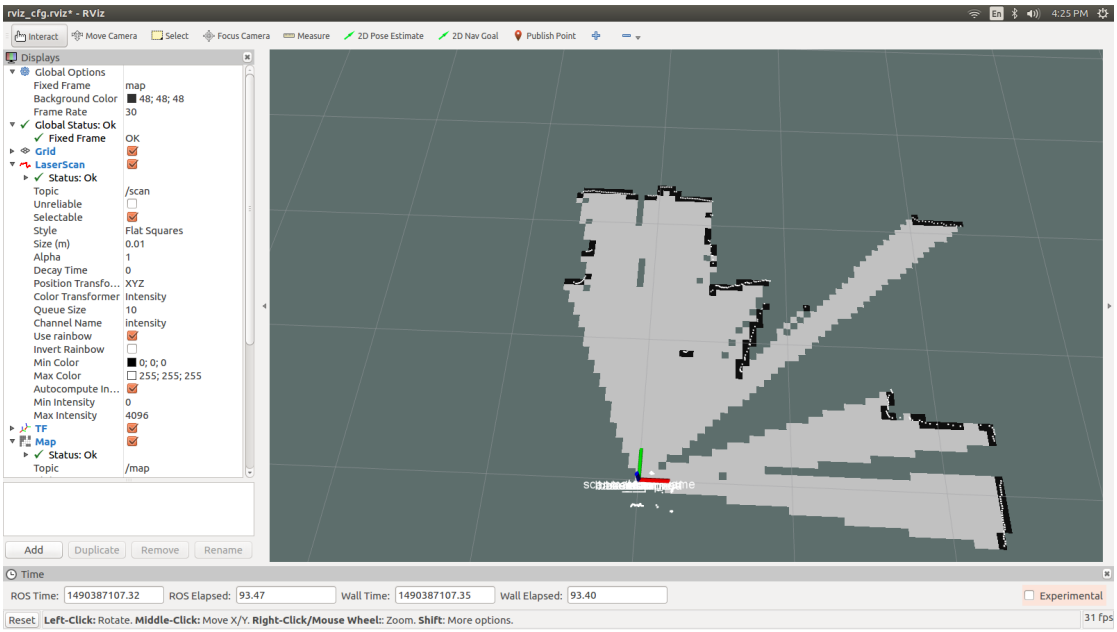


Figure 3.3: Rviz window. On the left all the topics that Rviz is subscribed to are shown, so that it can receive the information from them. It is possible to subscribe to new topic to receive different types of messages. The right side is a visualization of the 2D occupancy grid map, received from the topic `/map`.

3.2 RTABMap

RTABMap (Real-Time Appearance-Based Mapping) is a RGB-D, Stereo and Lidar Graph-Based SLAM approach based on an incremental appearance-based loop closure detector [23]. It has been choosen as a tool for this experiment for three reasons:

- it is equipped with an efficient memory management which makes it robust and effective in the loop closure detection task;
- it exists a ROS version of RTABMap, that has been used in the experiment because it exploits ROS functionalities bringing several advantages explained in section 3.2.3;
- it is highly customizable, well known, well documented and kept updated.

3.2.1 How RTABMap works

RTABMap SLAM relies on a smoothed SLAM approach, so the goal of the algorithm is to create a graph of the map. As explained in section 1.5.1 it is composed of a front-end part and a back-end part, that work simultaneously for the creation of the map. The former performs odometry, fusion of sensor information and a local bundle adjustment to make more accurate the local pose of the robot, while the latter performs graph optimization and a global map assembling. The figure 3.4 shows the division between the front-end and the back-end of RTABMap. RTABMap can create different maps, depending on the sensor used. If an RGB-D camera is used, it is possible to create a dense point cloud map, in which the 3D structure of the environment is reconstructed. If a 2D laser sensor is used, a 2D occupancy grid map is obtained, that explained in section 1.4.1 and it is used in the experiment. Furthermore, RTABMap is equipped with a very effective memory management for the detection of loop closures. When a loop closure is detected, a graph optimization routine starts, in order to correct the map, reducing the accumulated drift.

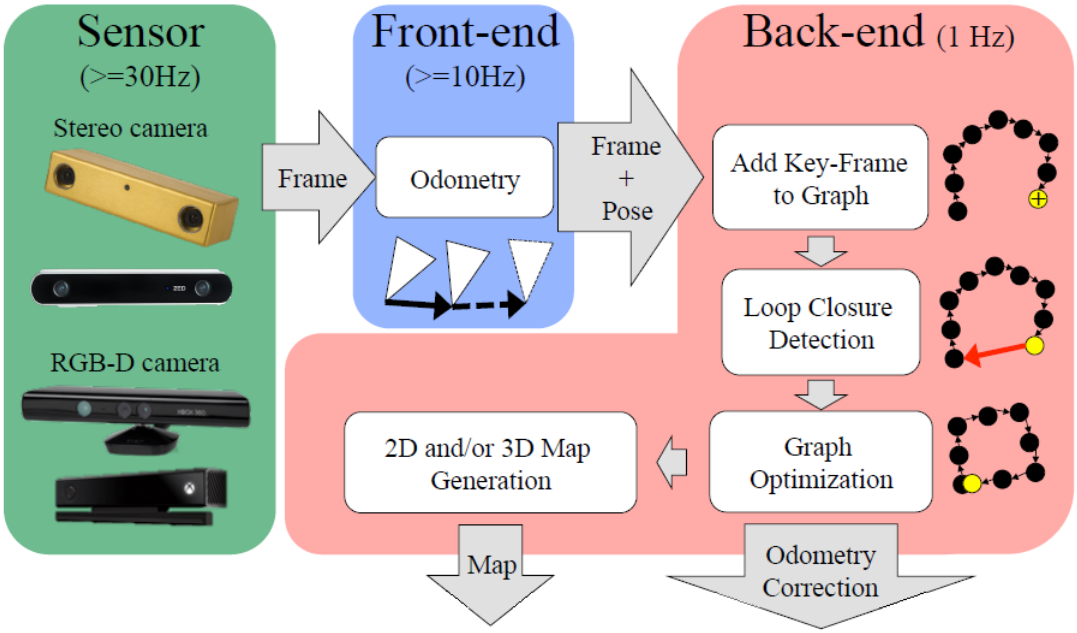


Figure 3.4: The RTABMap structure [24]. The front-end performs the odometry task, while the back-end is designated to the pose graph optimization and the map assembling.

3.2.2 RTABMap memory management

RTABMap is not so different from other SLAM open source tools, except for its memory management system, that allows large-scale environment exploration with effective loop closure detection [30]. This system implements three types of memory:

- *Short Term Memory* (STM), that is a buffer of customizable dimension in which new frames are inserted;
- *Working Memory* (WM), that is a working area used for the map optimization and for the loop closure detection task. Here, the most recent frames and the frames that are more similar to the new ones are kept. The former are used to perform the local map optimization in the back-end, while the latter are used to check if a loop closure is present. Due to the online constraint, this memory can't be too large because otherwise it could not detect the loop closure in real time and optimize the map accordingly;
- *Long Term Memory* (LTM), that is a database in which all the knowledge

about the map is maintained. Depending on the portion of the map in which the robot is more likely to be, a continuous transfer between the WM and the LTM is done, in order to keep in the WM the old frames that are more similar to the new ones observed by the robot;

At the beginning of the process all these memories are empty. When the SLAM process starts, the sensor information is synchronized and fused at each time instant and a new frame is inserted in the STM and a new node in the graph is added. When the STM reaches the maximum size, the less recent nodes begin to be transferred to the WM. In the WM are kept the most recent nodes and those nodes that are more similar to the new ones. A continuous transfer takes place between the WM and the LTM. When a new node is put in the WM, an efficient search in the LTM is performed to find the most similar nodes to this one and they are transferred back to the WM. This process is called *retrieve*. Since the WM has a fixed size, other nodes need to be transferred to the LTM, and they are the less recent or the less similar to the new one. This process is called *transfer*. The loop closure detection is performed by comparing the new node with the nodes that are in the WM only, so there is a constant number of comparison regardless of the size of the environment, and the memory management system ensure to have in the WM always node very similar to the new one. In this way the loop closure detection, even on large maps, is always performed in constant time, but the drawback is that if the right node isn't retrieved from the LTM to the WM, the loop closure is not detected. Figure 3.5 shows the computational complexity of the loop closure detection process.

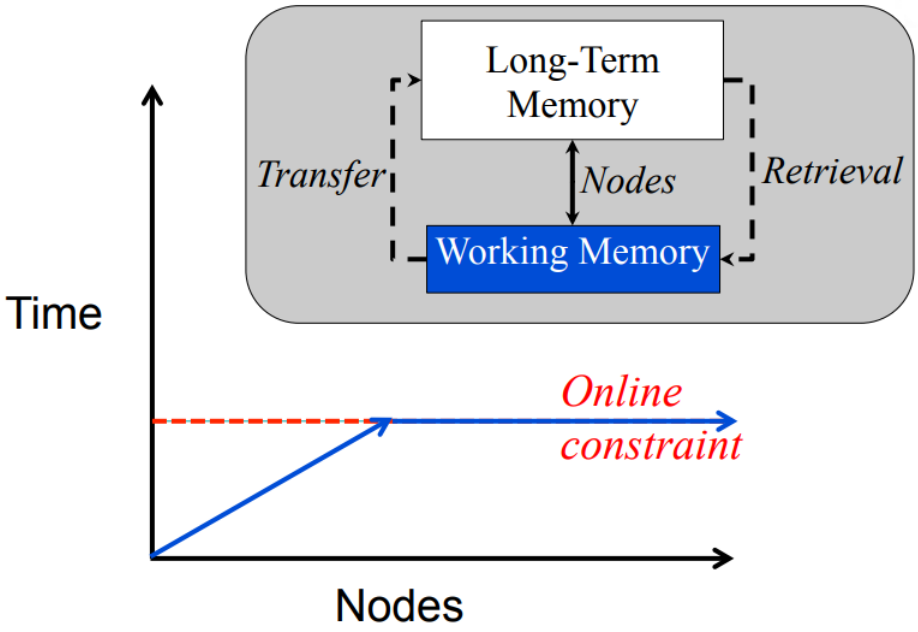


Figure 3.5: The RTABMap computational cost of loop closure detection [24]. The time grows linearly with the number of nodes in the map graph, until the Working Memory is full. Then, the computational cost becomes constant, because always the same number of nodes are kept in the Working Memory.

3.2.3 RTABMap_ROS

RTABMap_ROS is a ROS bridge between the desktop application of RTABMap and ROS. This solution is structured as a ROS application for the front-end and the map assembler, with nodes, services, messages, and it uses the back-end and the memory management of RTABMap. It also extends RTABMap with new functionalities, it can in fact combine visual SLAM with input data from other sensors such as IMU, lidar, wheel odometry and emulated laser scanner from RGB-D data. In the experiment RTABMap_ROS was used instead of the desktop application, for the following reasons:

- the sequences of the environment were captured as ROS bags, so a ROS application was needed to create the maps;
- the rosbags contain also the transformation tree of the sensors of the robot, that is how the sensors are positioned with respect to the baseline of the robot, that is an important information to have an accurate map building process;

- the ROS version greatly simplifies the customization of the algorithm by setting the parameters, thanks to its launch files. The launch command used is explained in A.3.

3.3 GMapping

GMapping is a 2D SLAM algorithm that relies on a filter-based approach and it uses laser sensor information to create the map of the environment as an occupancy grid map. It solves the SLAM problem by using a Rao-Blackwellized Particle Filter [19] and it is implemented as a ROS package.

3.3.1 Mapping with Rao-Blackwellized Particle Filters

The key idea behind the Rao-Blackwellized particle filter is to estimate the joint posterior probability of the robot trajectory $x_{1:t} = x_1, ..., x_t$ and the map m, given the observation $z_{1:t} = z_1, ..., z_t$ and the odometry measurements $u_{1:t-1} = u_1, ..., u_{t-1}$. The Rao-Blackwellized particle filter can be formalized as follow:

$$p(x_{1:t}, m|z_{1:t}, u_{1:t-1}) = p(m|x_{1:t}, z_{1:t}) \cdot p(x_{1:t}|z_{1:t}, u_{1:t-1}). \tag{3.1}$$

To estimate the posterior $p(x_{1:t}|z_{1:t}, u_{1:t-1})$ a particle filter can be applied. Each particle carries an individual map of the environment. In fact, in a particle filter, hypothesis for the robot pose and the map are kept in a set of particles. Each particle $<x^i, m^i, w^i>$ of the set $\{<x^1, m^1, w^1>, ...,<x^N, m^N, w^N>\}$ maintains the history of the past robot poses x^i, a map computed from this history m^i and a weight w^i [42]. The weight of a particle is a probability value that inform how likely the particle represents the pose history and the map, given odometry and sensor measurements. The improvement made by GMapping, is that not only the odometry measurements, but also the observation model $p(z_t|m, x_t)$ is taken into account when weights are calculated. It makes the process much more accurate when the sensor information is more accurate than the motion estimated by the odometry, such as when the robot is equipped with a laser sensor. Furthermore, GMapping uses also an adaptive

resampling technique, to perform the resampling of particles. During resampling, particles with a low importance weight $w^{(i)}$ are replaced by samples with a high weight. The adaptive resampling ensures that the risk of replacing good particles is reduced, because the number of resampling operations is reduced and they are only performed when needed.

3.3.2 Rao-Blackwellized SIR filter

One of the most common particle filtering algorithms is the *Sampling Importance Resampling* (SIR) filter. GMapping uses a Rao-Blackwellized SIR filter to process the sensor observations and the odometry information as they are available and it updates the set of samples that represents the posterior probability of the map and the trajectory of the robot. The entire process can be summarized by the following four steps:

1. *Sampling*: the next generation of particles $\{x_t^{(i)}\}$ is obtained from the generation $\{x_{t-1}^{(i)}\}$ by sampling from the proposal distribution π.

2. *Importance Weighting*: An individual importance weight $w_t^{(i)}$ is assigned to each particle according to the importance sampling principle

$$w_t^{(i)} = \frac{p(x_{1:t}^{(i)}|z_{1:t}, u_{1:t-1})}{\pi(x_{1:t}^{(i)}|z_{1:t}, u_{1:t-1})}. \tag{3.2}$$

   The weights take into account the fact that the proposal distribution π is in general not equal to the target distribution of successor states. The improvement of GMapping is that the importance weights are computed according to the observation model making the proposed distribution suboptimal when the sensor information is more precise than motion estimation

$$w_t^{(i)} \propto w_{t-1}^{(i)} \cdot p(z_t|m_{t-1}^{(i)}, x_t^{(i)}). \tag{3.3}$$

3. *Resampling*: particles are drawn with replacement proportional to their importance weight. This step is necessary since only a finite number of particles

is used to approximate a continuous distribution. Furthermore, resampling allows to apply a particle filter in situations in which the target distribution differs from the proposal. After resampling, all the particles have the same weight. As said before, GMapping use an adaptive resampling technique to ensure that good particles are not replaced.

4. *Map Estimation*: for each particle, the corresponding map estimate $p(m^{(i)}|x_{1:t}^{(i)}, z_{1:t})$ is computed based on the trajectory $x_{1:t}^{(i)}$ of that sample and the history of observations $z_{1:t}$.

Chapter 4

The experiments

Several experiments were conducted with the aim of comparing the 2D occupancy grid map obtained by two SLAM algorithms: RTABMap and GMapping, discussed in sections 3.2 and 3.3 respectively. To do this, a real robot was used, to record the rosbags in the Cesena campus of the University of Engineering and Architecture of Bologna. The building is shown in figure 4.1. Four sequences of three corridors in which there are loop closures were recorded and they are shown in this chapter. Once the sequences were captured, the two algorithms were launched and, exploiting the ROS functionalities, the maps were produced and saved.

(a) Photo of the Cesena campus (b) 3D model of the Cesena campus

Figure 4.1: Images of the Cesena campus building.

4.1 The robot

The robot used for recording the sequences is the Xaxxon autocrawler [51], shown in figure 4.2. Linux Ubuntu 18.04 64-bit OS and ROS Melodic were already installed on it and based on the Xaxxon documentation it has been configured to communicate with a PC, in order to use ROS remotely. A brief explanation of the configuration is given in the appendix A.1. The system hardware of the robot consists of the following components:

- motherboard ASRock J5040-ITX with Quad-Core Pentium J5040 CPU 3.2GHz;
- 16GB DDR4 RAM;
- SSD 120GB;
- M.2 Wifi/Bluetooth mini-pcie and a pigtail/antenna pair;
- Xaxxon ATX Power Kit with wide input regulator;
- expansion USB 2.0 hub 4-port unpowered.

It is equipped with multiple sensors that are now listed, the most important ones for the SLAM experiment are then described in more detail:

- an Intel Realsense D435 depth camera;
- a Xaxxon OpenLidar rotational laser scanner;
- a Pololu L3GD20 3-Axis gyroscope;
- four single wheel quadrature hall-effects high resolution as wheel encoders;
- a microphone;
- two RGB camera, front and rear.

Figure 4.2: Xaxxon autocrawler.

4.1.1 Intel Realsense D435 depth camera

The robot is equipped with an Intel Realsense D435 depth camera [10], consisting of an RGB module to capture RGB images and a depth module, composed of an infra-red beam emitter and two stereo receivers, left and right, for reconstruct the depth of the scenes. The figure 4.3 shows the camera.

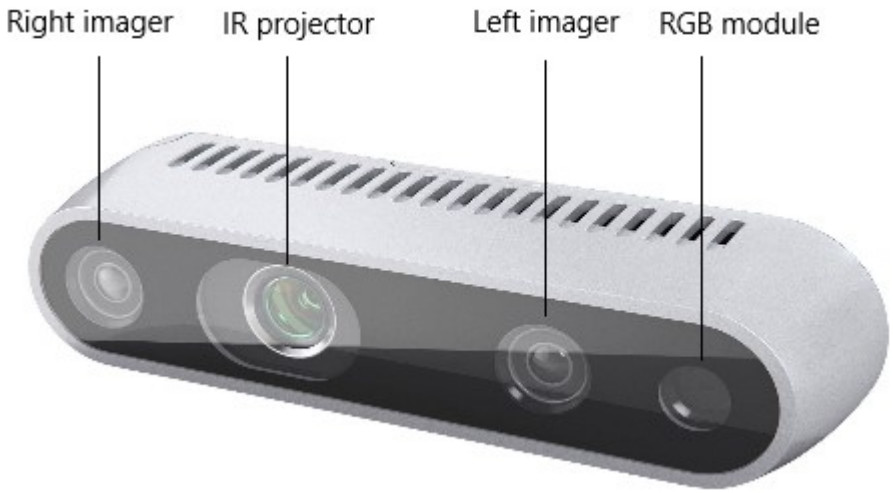


Figure 4.3: Intel Realsense D435 depth camera.

The RGB and the depth module can capture images up to 30 and 90 fps respectively and the laser operating range of the depth camera ranges from ∼0.11m to ∼10m. The depth images reconstructed by the RGB-D camera are 16-bit images in which each pixel value represents the distance in millimeters from the obstacles.

The original robot was equipped with an Intel Realsense D415 depth camera, but Xaxxon changed it for us upon request. The choice to equip the autocrawler with an Intel Realsense D435 instead of an Intel Realsense D415 was dictated by two reasons:

- the field of view (FOV) of the D435 is wider than the D415. It is 85° versus 65° respectively;
- the D415 has a rolling shutter while the D435 has a global shutter [11]. Cameras with a rolling shutter record all the pixels in a scene by rapidly scanning either left and right or vertically. This will usually happen over the course of a couple of frames, but the data will be saved as a single frame. Global shutter cameras operate differently in that they snapshot the whole scene in a single frame, so every pixel of the image is captured simultaneously and this makes the acquisition less sensible to rapid movements.

4.1.2 Xaxxon OpenLidar

A Xaxxon OpenLidar sensor [52] is a rotating lidar developed by Xaxxon. The sensor has a simple mechanical design, using the proven Garmin LIDAR-Litev3 laser distance measurement sensor, wired through a rotational slip ring, with stepper motor drive, two 3D printed frame parts, and an Arduino compatible PCB. The laser sensor has a sample rate up to 750Hz, the scanning speed is up to 250 RPM and the maximum range is 40m. It is positioned inside the robot and it can scan thanks to the lateral openings on the autocrawler. The OpenLidar and its positioning inside the robot are shown in figure 4.4 and 4.5 respectively.

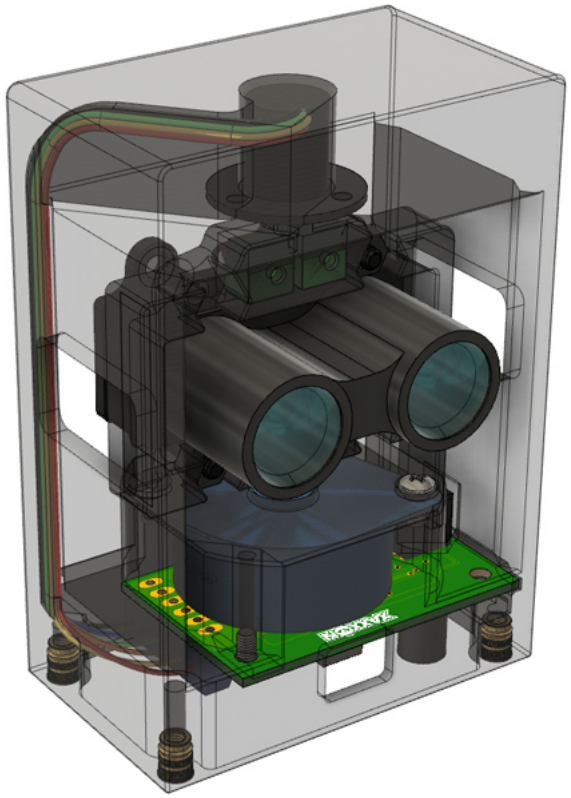


Figure 4.4: Model of the Xaxxon OpenLidar rotating lidar.

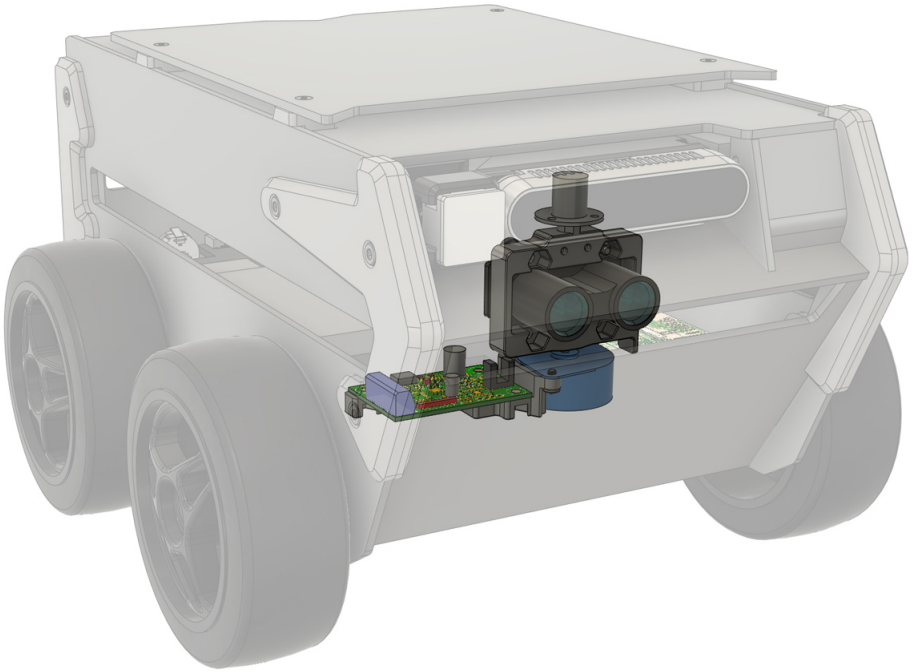


Figure 4.5: Positioning of the OpenLidar inside the autocrawler.

4.2 RTABMap_ROS registration strategies

As explained in subsection 3.2.3, RTABMap_ROS allows a great customization of the algorithm by setting the parameters in the ROS launch files. In the appendix A.3 the launching parameters of RTABMap_ROS used in the experiments are reported. In the following the parameter *registration_strategy* is highlighted, because depending on its value, different combinations of odometry algorithms are used. In the experiments, two different values were assigned to this parameter:

- value 0, that is the default value: in this setting the odometry is computed using only visual features from the RGB-D camera, using the Perspective-n-Point (PnP) algorithm;
- value 1: in this setting the odometry and loop closures are computed and refined using depth scan information from the robot 2D lidar with the Iterative Closest Point (ICP) algorithm.

A comparison was also made between the map obtained with the two methods, to understand some criticalities of visual odometry. For example, university corridors are formed by white walls, with few visual features. Extracting relevant odometry information from the images could be critical. Having depth information, thanks to the laser sensor, the use of ICP algorithm could lead to better solutions. In the absence of a laser, the depth camera can also be used as a fake laser scan, as explained in 1.4.1.

4.2.1 Visual odometry

RTABMap implements a fairly standard visual odometry algorithm. From each sensor acquisition, features are detected and tracked from the previous acquisitions and the motion of the RGB-D camera is estimated. The Perspective-n-Point (PnP) algorithm is used for the motion estimation, with the RANSAC algorithm for outlier rejection. PnP computes the transformation done by the camera given the 3D structure of the last frame and the 2D image feature correspondences in the new frame. Having the depth information, the keypoints of the last frame are projected

into the 3D space, while the keypoints of the new frame, which are tracked from the previous one, are in the 2D image coordinates. The algorithm aims to find the transformation that minimizes the image re-projection error, that is the error obtained by re-projecting the 3D keypoints in the 2D image coordinates [39]. Figure 4.6 shows more in-depth the visual odometry process performed by RTABMap.

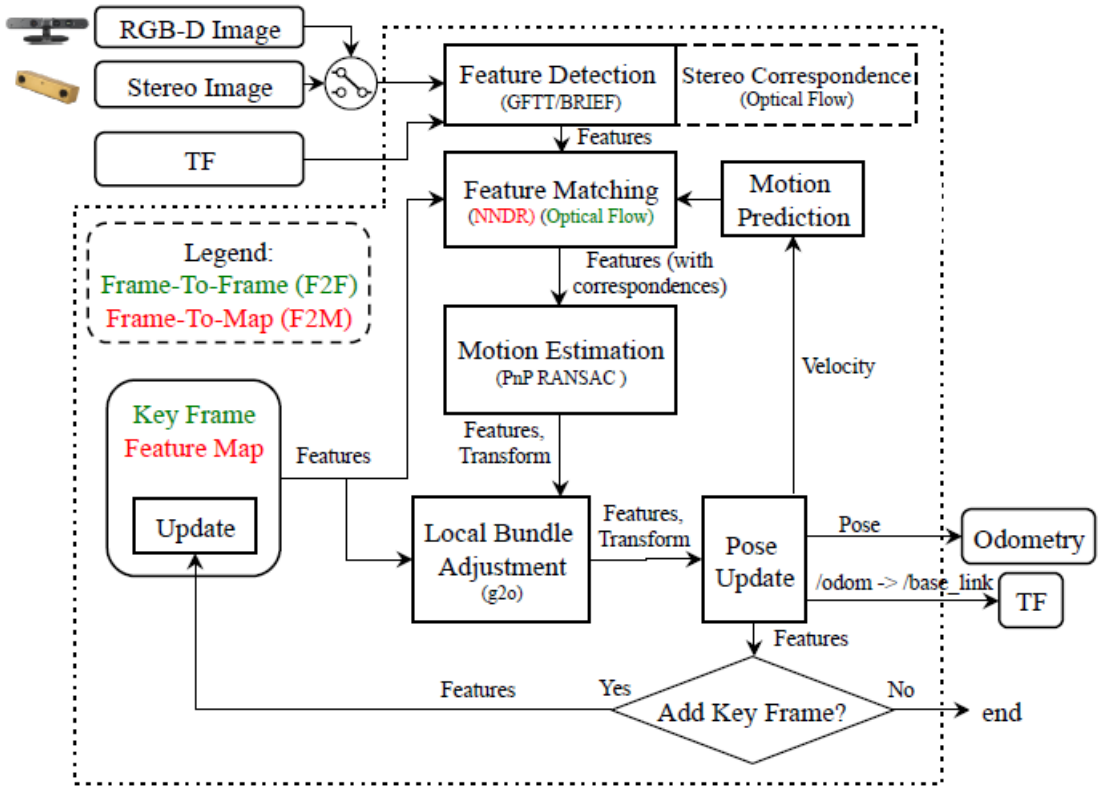


Figure 4.6: Sensors and algorithms used by RTABMap to perform visual odometry [25]. It uses the PnP algorithm to estimate the motion of the robot. In the legend, frame-to-frame (F2F) and frame-to-map (F2M) are highlighted. F2F registers the new frame against the last keyframe, while F2M registers the new frame against a local map of features created from past keyframes.

4.2.2 Lidar odometry

RTABMap allows to use depth information, acquired from a lidar or from a depth camera, to perform odometry. In this case the algorithm used is the classical ICP, that is an iterative algorithm to find the best transformation to align the corresponding 2D or 3D keypoints in two successive frame. In RTABMap, when depth information are available, by setting the $registration\_strategy = 1$, the odometry is

computed by the ICP algorithm on laser keypoints, while loop closure is enhanced with visual features. Figure 4.7 shows more in-depth the lidar odometry process performed by RTABMap.

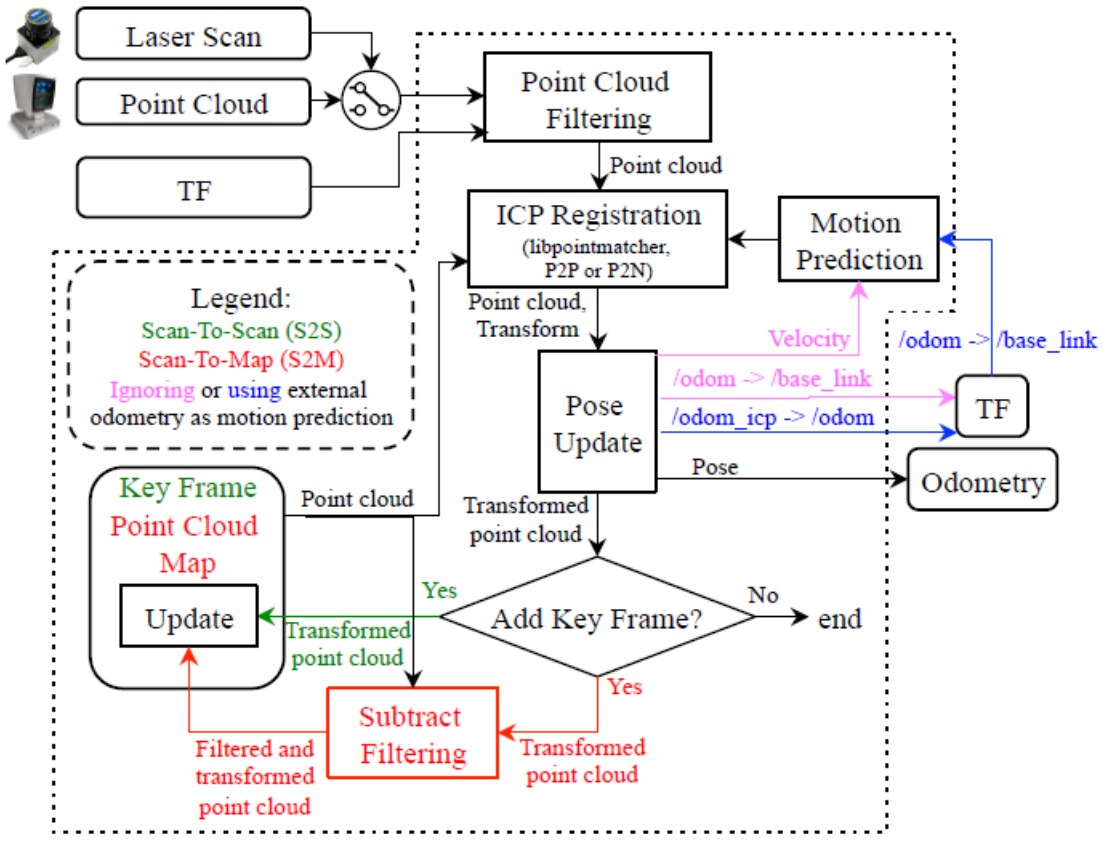


Figure 4.7: Sensors and algorithms used by RTABMap to perform lidar odometry [25]. It uses the ICP algorithm to register successive laser acquisitions. The legend highlights the S2S and the S2M methods, that are analogous to the F2F and the F2M explained in figure 4.6.

4.2.3 Wheel odometry integration

In RTABMap_ROS wheel odometry is used as initial guess for the odometry. When a robot performs a movement, and the command is for example to go straight for one meter, the robot never go straight for one meter, due to the noise, the friction and other causes, so it always encounters drift. In particular, the rotation of the robot is more critical, because if the robot rotates 89° instead of 90°, then the drift with respect to the correct trajectory increases more and more over time. For this reason, sensor information are used to make the odometry estimation more precise,

using the wheel odometry as initial hypothesis. On the other hand, using only sensor data to estimate the odometry can lead to bad results due the presence of outliers. In this way, combining the wheel odometry as initial guess with a refinement step through sensor acquisitions helps to avoid bad data association.

4.3 ROS packages

Thanks to ROS, it was very easy to move the robot and enable all its sensors to record the sequences. In the appendix A.2, all the packages and nodes used in the experiment are explained in-depth, to give a greater clarity of the work done.

4.4 The sequences

For the experiment, four sequences were captured in three different environments.

All the bags were captured in a static environment without the presence of humans, with good light condition. The images were recorded at 10 fps and compressed. The environments in which the sequences were captured are university corridors, in which loop closures are present. The walls of the corridors are orthogonal each other.

Below are the plans of the corridors mapped, to provide a 2D view of the rooms. A red line has been drawn to give an idea about the paths approximately followed by the robot, the black arrows indicate its direction of travel, while dots of different colors are used to mark salient places in the map. The green points and the red points respectively indicate the start and end points, the blue points indicate where loop closures are detected.

4.4.1 Sequence 01

The first sequence presents an orthogonal corridor. The autocrawler started moving and recording from the bottom left corner of the map and moved clockwise. It has traveled all the corridor, returning back at the starting point, so there is a loop

closure here, then it moved inside a lab room and explored it. Finally, it returned back to the corridor and the sequence end with the robot in the left up corner.

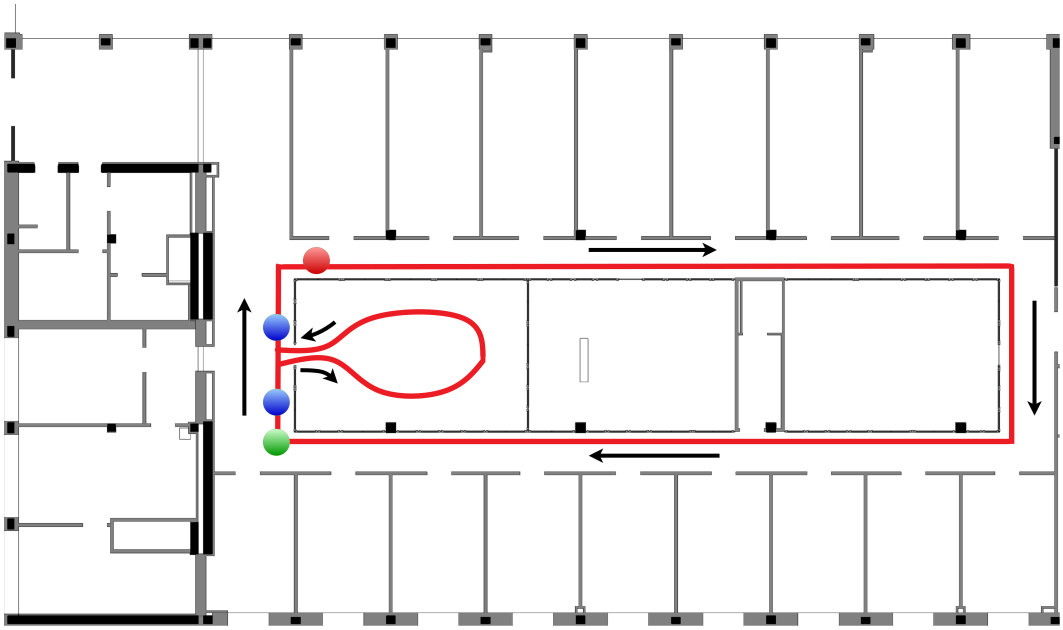


Figure 4.8: Planimetry of the corridor of the sequence 01.

4.4.2 Sequence 02

The second sequence presents two orthogonal corridors. The robot started moving from the center of the map, i.e. the bottom right corner of the left block and moved counter-clockwise. It returned back to the starting point, where there is the first loop closure. Then it explored the right block moving clockwise and returning to the starting point, performing a second loop closure.

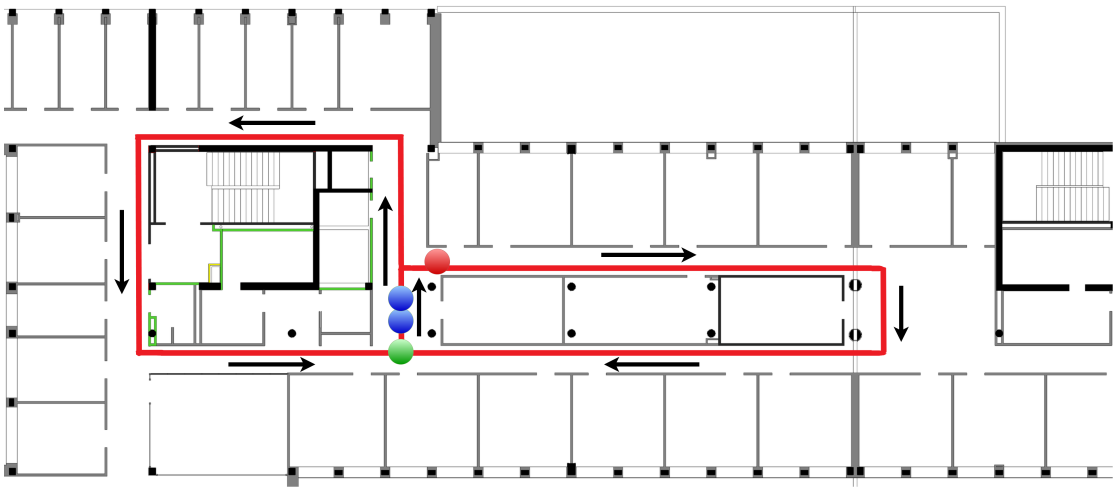


Figure 4.9: Planimetry of the corridor of the sequence 02.

4.4.3 Sequence 03

The third sequence presents two orthogonal corridors. The robot starts from the top right corner of the left block and performs a circle in the center of the two blocks, arriving in the top left corner of the right block. Here it started moving clockwise in the right block, until it returned to the upper left corner of the right block, performing the first loop closure. Then it turned left and started moving counterclockwise into the left block. Finally it returned where the first loop closure was performed and a second loop closure was detected.

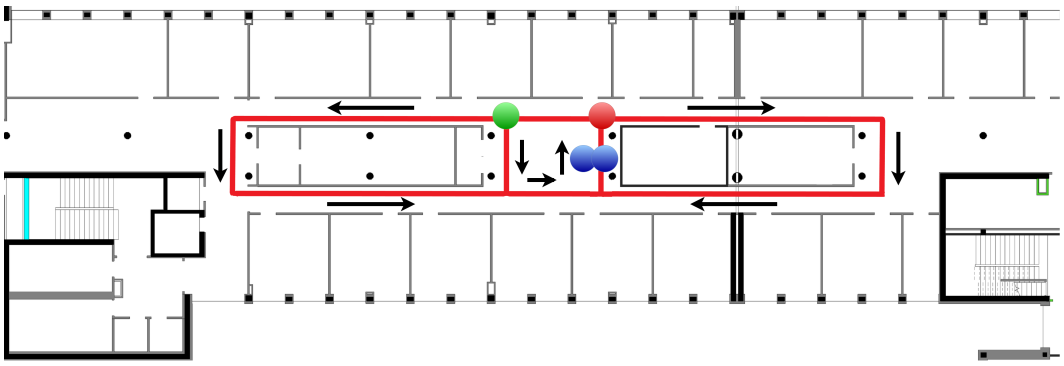


Figure 4.10: Planimetry of the corridor of the sequence 03.

4.4.4 Sequence 04

The fourth sequence is the exact same corridor of the first sequence, as the path followed by the robot. In this sequence the robot moved to an higher velocity with respect to the first one, and it has been turned on itself several times, in order to make the odometry computation more difficult.

4.5 Results and evaluation

In chapter 2, an overview of formal SLAM metrics has been done and they all need the ground truth-trajectory to be computed. In this experiment there isn't a ground-truth, so the comparison between the results obtained can be only qualitative, by evaluating the 2D occupancy grid maps.

The maps created by playing the rosbags with the two different algorithm of RTABMap_ROS and GMapping are presented in the following.

4.5.1 Sequence 01

The image 4.11 shows the occupancy grid map obtained with GMapping, while the images 4.12 and 4.13 show respectively the the occupancy grid maps obtained with RTABMap_ROS using the default and the ICP as registration strategies.

It is possible to notice that GMapping has created a very accurate map, with a good orthogonality of the walls. Despite GMapping isn't equipped with a loop closure detector module, it adjusted the map reducing the accumulated drift when the robot returned back to the starting point, thanks to the Rao-Blackwellized particle filter, that kept the most likely particles. The map created by RTABMap_ROS using the default registration strategy isn't as accurate as the previous one, the walls aren't orthogonal each other and in general the map has a lower quality. The map obtained with the ICP as registration strategy is much better, there is a good orthogonality of the walls and it's really similar to those one created by GMapping. The loop closure detector module of RTABMap_ROS recognized the loop closure in the bottom left angle of the map and adjusted the map reducing the drift, that

was very high, regardless of the registration strategy.

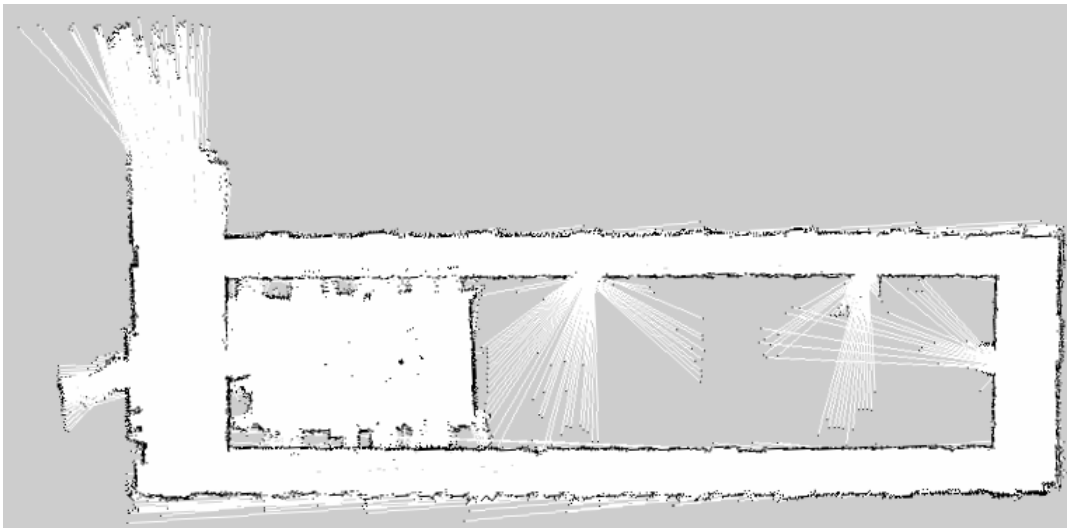


Figure 4.11: Occupancy grid map of the sequence 01 obtained with GMapping.

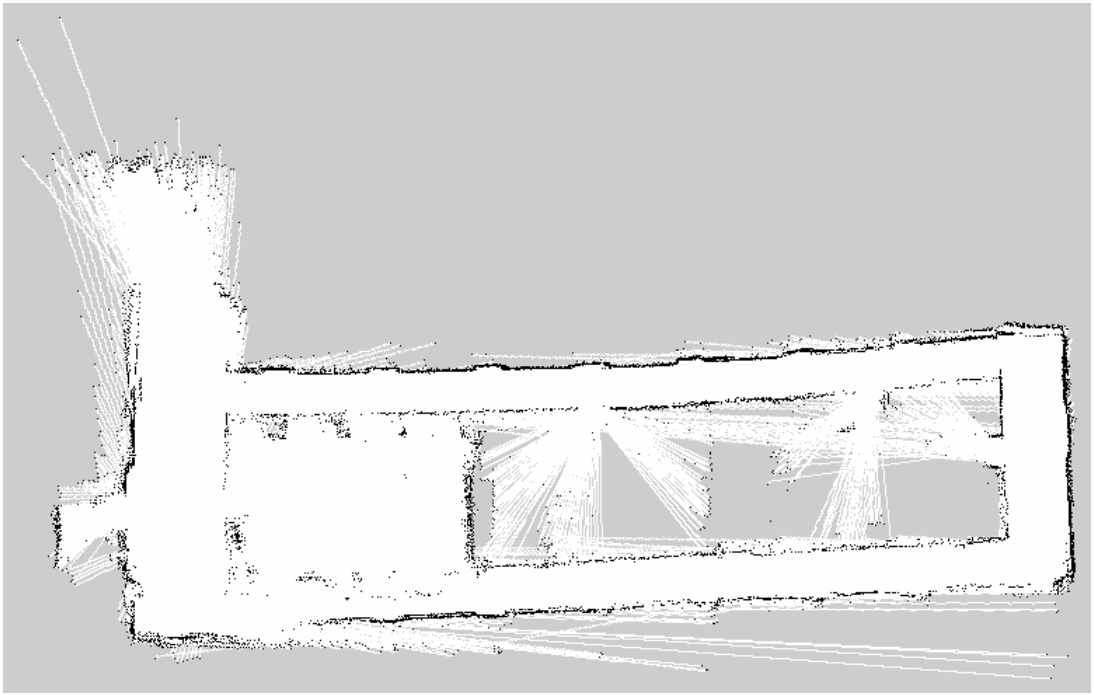


Figure 4.12: Sequence 01 occupancy grid map obtained with RTABMap using the registration strategy based on visual features only.

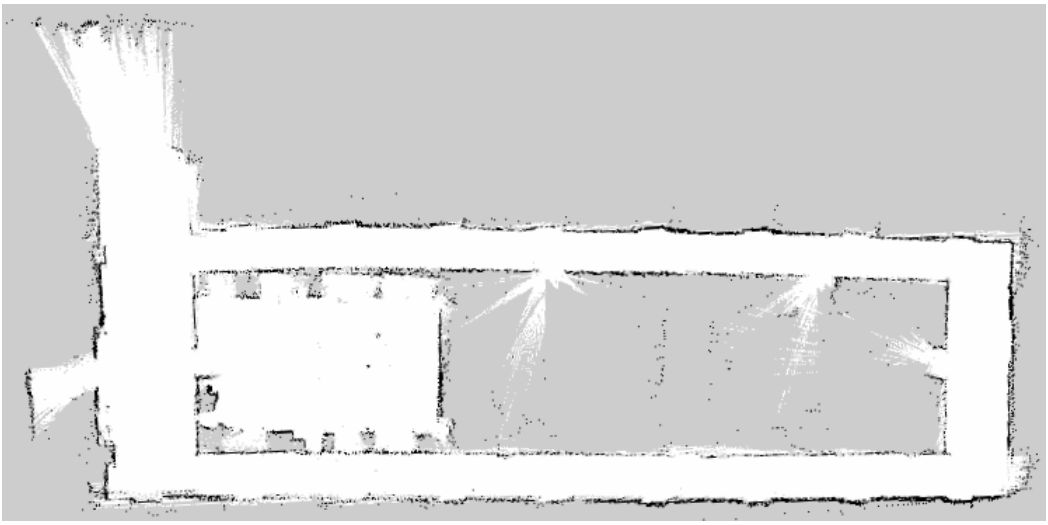


Figure 4.13: Sequence 01 occupancy grid map obtained with RTABMap using the ICP registration.

4.5.2 Sequence 02

The image 4.14 shows the occupancy grid map obtained with GMapping, while the images 4.15 and 4.16 show respectively the the occupancy grid maps obtained with RTABMap_ROS using the default and the ICP as registration strategies.

As for the sequence 01, the maps obtained with GMapping and RTABMap_ROS using the ICP as registration method are similar and they are more accurate than that obtained with RTABMap_ROS using visual features only. The latter in fact presents a lack of orthogonality between the walls in the right block, contrary to the other two maps. The map obtained with RTABMap_ROS using the ICP registration is even better than that obtained with GMapping, as it has a slight curvature in the corridor of the lower part of the right block, unlike the first. As before, both RTABMap and GMapping was able to adjust the map after each loop closure, greatly reducing the accumulated drift.

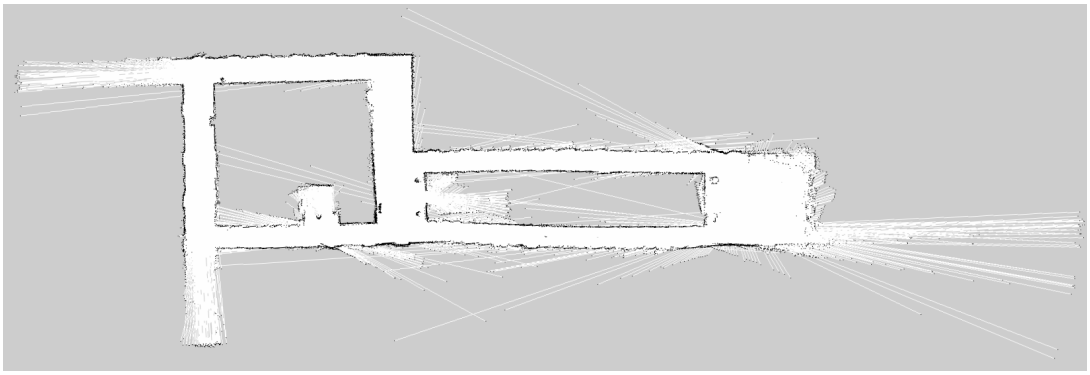


Figure 4.14: Sequence 02 occupancy grid map obtained with GMapping.

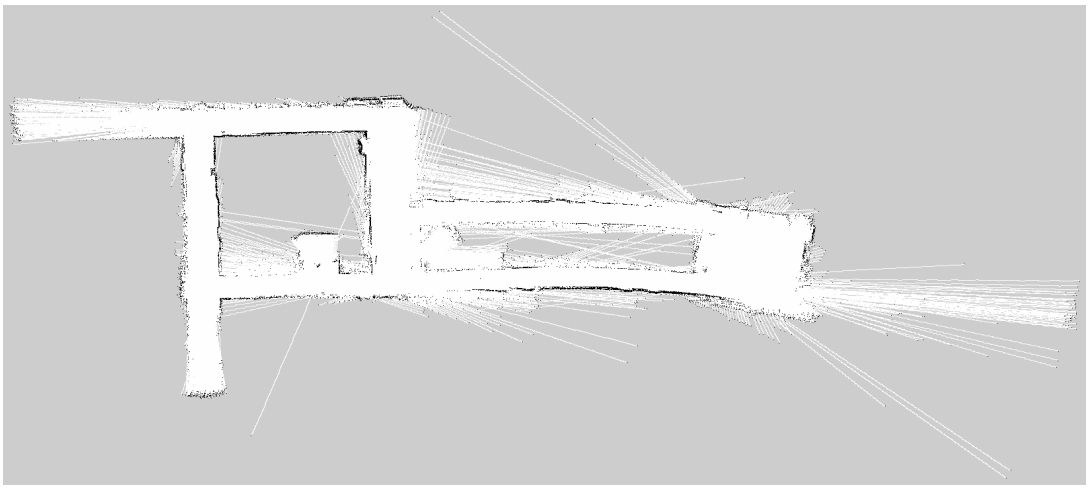


Figure 4.15: Sequence 02 occupancy grid map obtained with RTABMap using the registration strategy based on visual features only.

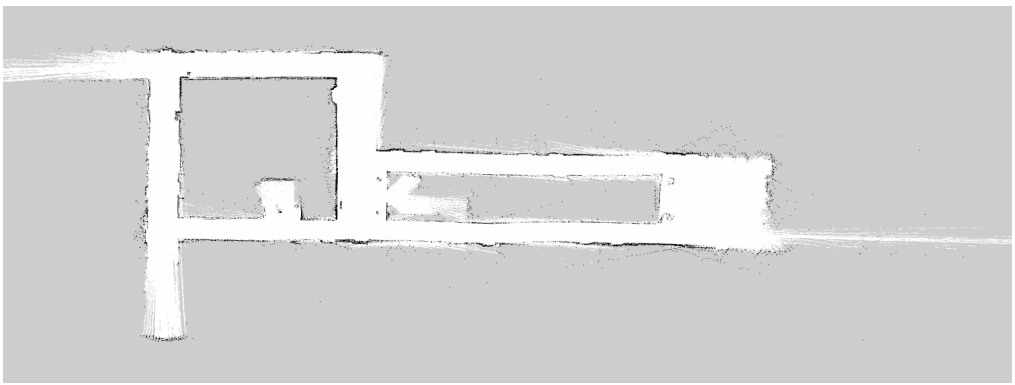


Figure 4.16: Sequence 02 occupancy grid map obtained with RTABMap using the ICP registration.

4.5.3 Sequence 03

The image 4.17 shows the occupancy grid map obtained with GMapping, while the images 4.18 and 4.19 show respectively the the occupancy grid maps obtained with RTABMap_ROS using the default and the ICP as registration strategies.

It's really difficult to say what is the most accurate occupancy grid map for this sequence. It is possible to notice that the map that presents the best orthogonality of the walls is that obtained with RTABMap_ROS using the default registration strategy, contrary of the previous sequences, but it presents also some inaccurate laser scans, for which cells seem to be unoccupied while they are. In general, the three maps obtained for this sequence are very similar and both the algorithms corrected the drift after each loop closures.

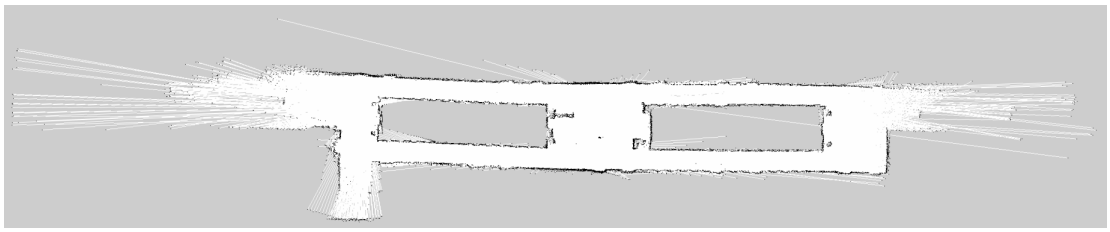


Figure 4.17: Sequence 03 occupancy grid map obtained with GMapping.

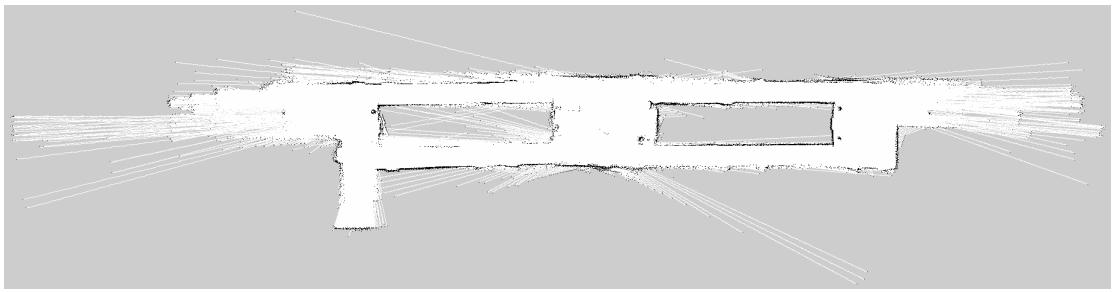


Figure 4.18: Sequence 03 occupancy grid map obtained with RTABMap using the registration strategy based on visual features only.

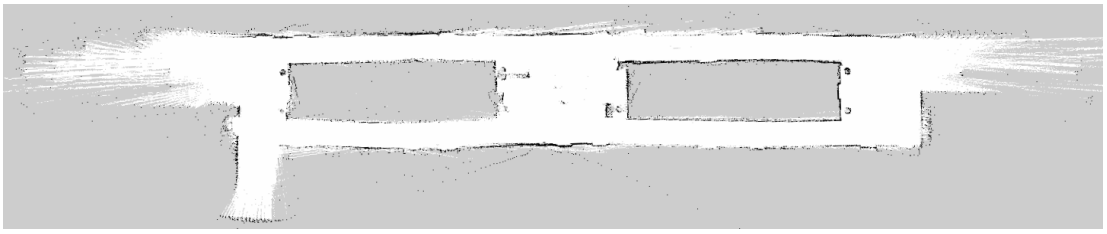


Figure 4.19: Sequence 03 occupancy grid map obtained with RTABMap using the ICP registration.

4.5.4 Sequence 04

The images 4.20 and 4.21 show respectively the occupancy grid map obtained with GMapping and RTABMap_ROS.

This sequence was captured to demonstrate the criticality of these methods when the conditions are not optimal. The corridor corresponds to that of the sequence 01, but the robot moved faster and was turned on itself several times during the mapping, to make the odometry computation much more difficult and less accurate.

It is possible to notice that the map obtained with GMapping is very inaccurate, since the particle filter is unable to sample the correct particles and the drift accumulated is not reduced. This led to the absence of orthogonality in the walls and more in general to a map that does not reflect reality and can't be used as input in a path planning task.

In the map obtained with RTABMap_ROS there is the worst case of loop closure detection: a false positive. When the robot returns to the starting point, the loop closure is correctly detected, but when the robot leaves the lab room, RTABMap_ROS recognizes a loop closure with the top left corner and the map is adjusted so that the entrance of the room coincides with this corner. This leads to a totally wrong map, with unoccupied cells of the occupancy grid map located in places where they doesn't exists.

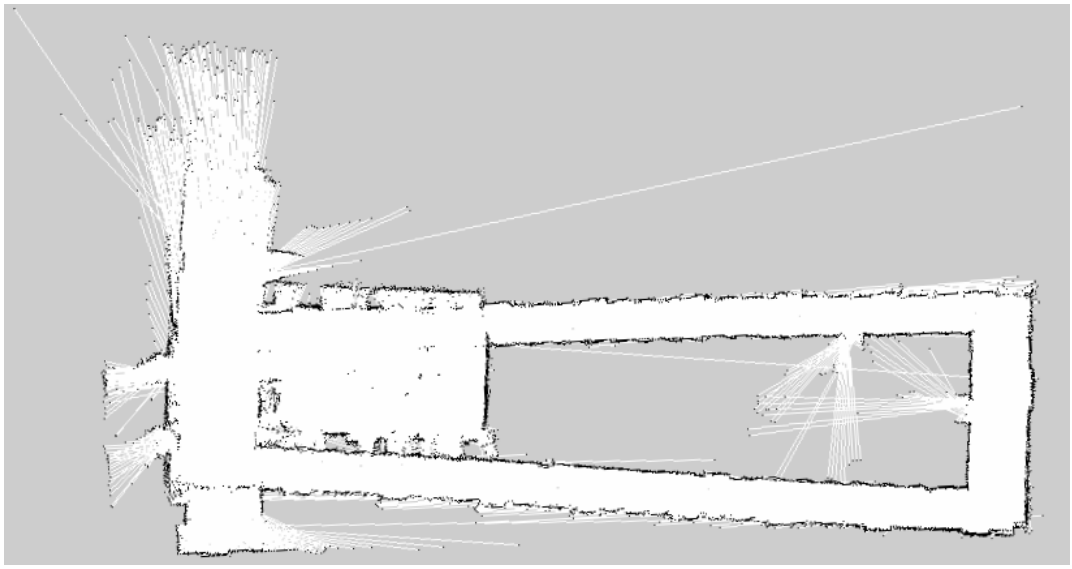


Figure 4.20: Sequence 04 occupancy grid map obtained with GMapping.

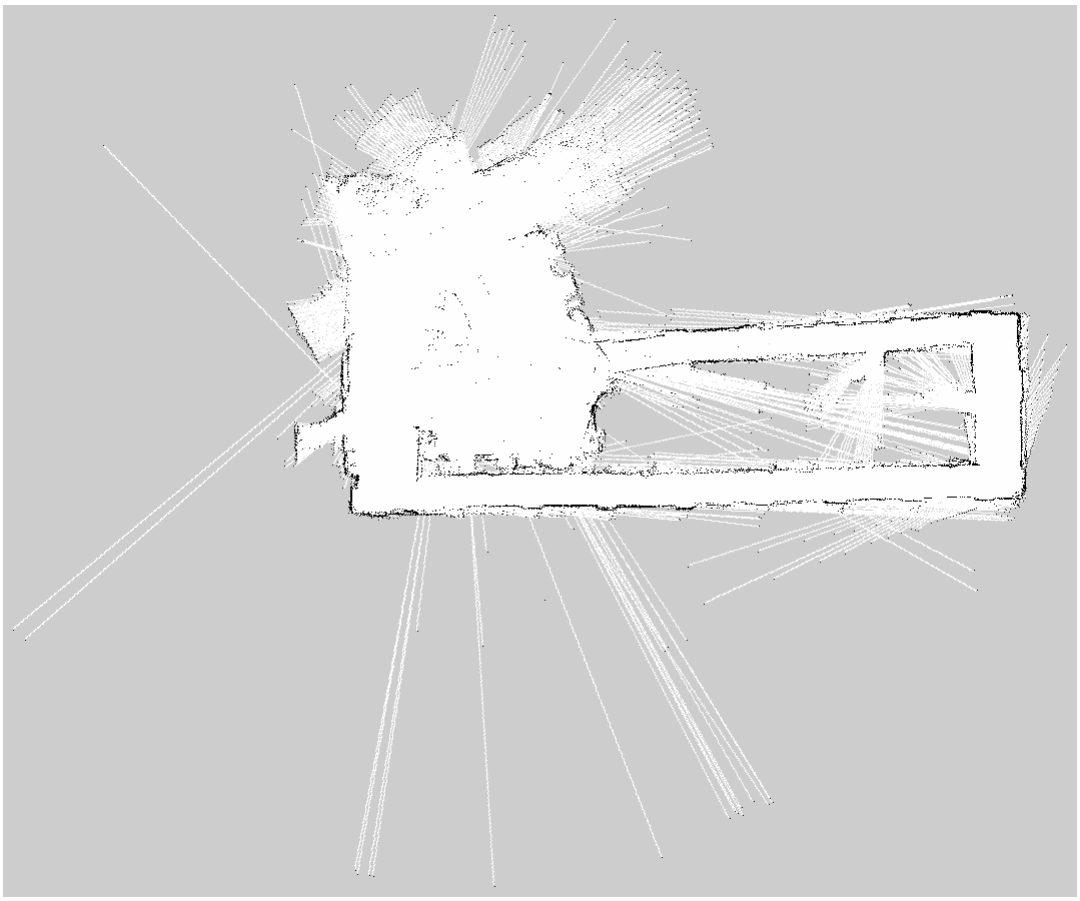


Figure 4.21: Sequence 04 occupancy grid map obtained with RTABMap.

Chapter 5

Conclusions and future work

In this final chapter the conclusions about the experiments are discussed and final considerations are made, based on the acquired knowledge and the possible future explorations of the SLAM topic.

5.1 Conclusions

During the dissertation, the theory behind the SLAM problem was explained and practical experiments were carried out with a mobile robot. Although a long-term SLAM in real-world conditions cannot yet be considered a solved problem, the experiments show that good solutions already exist for medium static environments. The two algorithms in the chapter 4 were able to create good 2D maps of the environment that can be used for path planning, in order to move the robot autonomously from a source point to a target point without human interaction. By analyzing the results of the experiments more in depth, it is possible to state that the best solutions are those obtained by using registration strategies that takes into account depth information. It is quite intuitive, in fact information from laser are crucial to create a good 2D map, especially in an environment with a few visual landmarks, such as long white corridors, where it is difficult to obtain a correct odometry from the visual information, while it is obtainable with depth information. As explained in 1.3.2, several state-of-the-art solutions merge different types of odometry. A consideration that arises from the experiment of this thesis is that good 2D occupancy grid maps

can be obtained by exploiting SLAM solutions using wheel and lidar odometry and enhancing the loop closure detection process with visual feature, which are clearly more discriminating and rich in semantic meaning than the information about the 3D structure of the environment.

5.2 Future work

As mentioned in 1.5.2, the currents SLAM solutions are based on traditional methods and algorithms that are handcrafted by programmers. In the front-end, traditional feature-based methods are used for the odometry computation, bag of words are used for the loop closure detection and classical graph optimization algorithms, such as Guass-Newton and Levenberg-Marquardt, are used in the back-end. All these methods are effective and also efficient for simple SLAM scenarios, but do not fit large-scale SLAM scenarios, both in time and space. It is the already discussed lifelong SLAM problem, for which these traditional techniques are probably not sufficient. To deal with the novelty, new environment, but also the same environment that changed, the robot must be equipped with some sort of learning capabilities, in order to update its knowledge of the world, while it changes. Modern methods are based on artificial intelligence techniques, such us artificial neural network and deep learning. These methods can be exploited in almost all the different parts that compose the SLAM problem and if they are used in they right way, they can improve the robustness of the overall process, but the robots still lack of a truly topological knowledge of the world. In addition, they are still slower and require more computing capabilities with respect to the traditional methods, so they are not used in the most common implementations of SLAM today.

To understand the future work of this field of research it is worthy to define what is the purpose of the SLAM process: it is necessary to have autonomous robots that moves in the physical world. But autonomy is not an all-or-nothing property, there exist several degrees of autonomy. For a robot that works inside a warehouse, classical SLAM approaches could be sufficient to create a 2D occupancy grid map, as shown in the experiments and thanks to this map it can move inside

the warehouse autonomously. But in order to have more autonomous robots, which move inside houses and can even go out and walk on a busy street, explore new large environment, it is mandatory to have a spatial notion of the environment as humans or animals have. This second consideration probably goes beyond the SLAM problem. In fact, humans do not need a precise map to travel the world or to know precisely how much they have rotated and translated when they take a step, they rather do a qualitative estimation about how they are distant from an obstacle or from a target and how to move in order to avoid or reach it. This capabilities is something that humans learn in their childhood, it's not coded inside their brain. They also learn to remember a place when they revisit it and they are able to learn how a place is made the first time they visit it.

Thinking about SLAM, there are several possible improvements in the process that can be leveraged in the near future, such as better memory management, better optimization algorithms, and so on. One improvement that I think is crucial, is the introduction of some sort of topological intelligence, even if it is hard-coded by a programmer, to avoid the detection of false positive loop closures. In case of a SLAM system that detects loops only through the match of visual features, the worst case that must be avoided is that the environment presents two places really similar each other, so the robot could confuse them. In fact, it could happen that the robot visits one of these places, it memorizes its visual features and later in the exploration it arrives to the second place. There, it matches the visual features and due to the similarity it re-localizes itself on the first location. If it also took into account odometry information, it would know that it is very unlikely that it has returned to the first place, because, despite the drift, the discrepancy between the actual pose and the pose in the previous location is too large, and the loop closure wouldn't be performed.

Regarding the use of artificial intelligence and in particular ANNs in the SLAM problem, they can be introduced in all the sub-part[7] of the process: they can be use in the odometry estimation[53, 45, 28, 14, 22], to extract better features and to give them a semantic meaning, in the graph construction and optimization[26, 3], using graph neural networks to represent the path of the robot in order to optimize

it, for enhancing the place recognition[8] and the loop closure detection[2, 16] too. In particular, I think that it would be very difficult to achieve a good lifelong SLAM solution without a robot that continuously learns, as humans do. In the recent years, the field of continual learning[29] is rising, that is a branch of deep learning that studies how to make robots learn continuously. Such a robot could learn how a place changes during the day, with different light condition, with object added, moved or removed. This is obviously useful in detecting loop closure, but also in creating of robots that have a higher topological notion of the world.

Following what I have read and studied, the knowledge acquired in writing this thesis, I think deep learning techniques and continual learning will likely be the turning point in achieving lifelong SLAM.

Appendix A

Technical details of the experiments

This appendix is added to provide a more in-depth explanation of the technicalities of the experiments made, in terms of robot configuration, ROS nodes used and parameter settings and to provide a better understanding of the work done.

A.1 Robot configuration

The robot configuration was performed following the Xaxxon [50] documentation in order to control the robot remotely, using a local PC. Once the configuration is complete, it is possible to navigate the robot file system and launch the autocrawler ROS applications using the Secure Shell (SSH) [49]. SSH is a protocol that enable a remote encrypted communication between two hosts in the same network through a command-line interface. Using a host name and a password, the local PC can connect to the robot and launch ROS packages and send commands to make it move remotely. During the ROS bags acquisition, the autocrawler was commanded remotely, making this task easier and more comfortable. Using the Firefox Browser and the Xaxxon browser application, or alternatively Rviz, it was possible to observe in real time the images seen by the front camera of the robot, so it could be moved remotely, without any danger.

A.2 ROS packages and nodes

This section explains all the packages and technical aspects of the nodes used in the experiments, distinguishing in which part of them they were used.

A.2.1 Sensors activation

To acquire the ROS bags, all the sensors of the robot that are useful for the SLAM task were activated, namely the Intel Realsense D435 depth camera and the Xaxxon OpenLidar. The packages and the nodes used are:

- the *realsense-ros* package to manage the depth camera;
- the *lidarbroadcast.py* node of the package *xaxxon_openlidar* to manage the rotating lidar.

To launch the two sensors in a single shot the file sensor.launch was written and it is the following XML file:

```
<launch>
    <!-- run the realsense_aligned.launch of the autocrawler
        package -->
    <include file="$(find autocrawler)/launch/
                realsense_aligned.launch" />

    <!-- run the xaxxon_openlidar.launch of the autocrawler
        package -->
    <include file="$(find autocrawler)/launch/
                xaxxon_openlidar.launch" />
</launch>
```

The realsense_aligned.launch file runs the launch file of the Intel Realsense D435 depth camera from the package realsense_ros and sets the parameter *align_depth* to `true`, to align the depth images and the RGB images captured at each time and it is the following XML file:

```
<launch>
    <node pkg="tf2_ros" type="static_transform_publisher"
            name="base_to_realsense_broadcaster"
            args="0.152 0.012 0.163 0 0 0 base_link camera_link" />
    <!-- run the rs_camera.launch file of the package
        realsense2_camera -->
    <include file="$(find realsense2_camera)/launch/
                rs_camera.launch">
        <!-- parameter to align the depth images with the
            rgb images -->
        <arg name="align_depth" value="true"/>
    </include>
</launch>
```

The xaxxon_openlidar.launch file runs the node of the autocrawler package *lidarbroadcast.py* that manages the rotating lidar. It sets the *masks* parameter to those angles for which the robot must ignore the laser acquisition, because the sensor is covered by the robot itself. It is the following XML file:

```
<launch>
  <node pkg="autocrawler" type="odom_tf.py" name="odom_tf"
          required="true" output="screen" >
              <param name="odom_lag" value="0.035"/>
        </node>
        <node pkg="tf" type="static_transform_publisher"
              name="base_to_xaxxon_lidar_broadcaster"
              args="0.103 0.0 0.120 0 0 0 /base_link
                    /laser_frame 100" />
        <node pkg="xaxxon_openlidar" type="lidarbroadcast.py"
              name="lidarbroadcast" required="true" output="screen">
                <param name="masks" value="54 80 162 174
                                    190 208 279 305" />
```

```
        </node>
</launch>
```

A.2.2 Robot movement

To move the robot remotely, the nodes used are:

- *teleop_twist_keyboards.py* of the package *teleop_twist_keyboard*, that allows to send commands from the keyboard to the robot, to make it move. This node was run on the local PC;
- *cmd_vel_listener.py* of the package *autocrawler_ros*, that receives the commands and it activates the wheels to move the robot accordingly. This node was run on the robot, to receive the commands from the PC via the SSH protocol.

A.2.3 Images acquisition

In the bags, the images were captured in a compressed format and they were downsampled from 30fps to 10fps. The node *throttle.cpp* of the package *topic_tools* was used for this task. The command template is the following:

```
rosrun topic_tools throttle messages <intopic> <msgs_per_sec>
[outtopic]
```

where *intopic* is the input topic, the *msgs_per_sec* is the maximum number of messages per second to let through and the *outtopic* is an optional parameter that is the name of the output topic, that is the *intopic* by default.

The command used in the experiment were:

- `rosrun topic_tools throttle messages /camera/color/image_raw/compressed 10.0 /camera/color/image_raw_throttle/compressed` to capture the RGB images;

- `rosrun topic_tools throttle messages /camera/aligned_depth_to_color/image_raw/compressedDepth 10.0 /camera/aligned_depth_to_color/image_raw_throttle/compressedDepth` to capture the depth images.

A.2.4 Registration and reproduction of ROS bags

To record the bags, the *rosbag* built-in package of ROS was used. Only the topics useful for SLAM are specified, to lighten the bags that are the compressed and throttled images, the info about the camera, the laser scan, the odometry and the transformation tree. The command used is:

```
rosbag record /camera/color/image_raw_throttle/compressed
/camera/aligned_depth_to_color/image_raw_throttle/compressedDepth
/camera/color/camera_info
/camera/aligned_depth_to_color/camera_info
/scan /odom tf tf_static
```

To test the algorithms with the bags captured, the SLAM tools were launched and the ROS bags were played back with the command `rosbag play --clock [bag name]`. With the `--clock` parameter, the clock time is also published.

A.2.5 Mapping

To create the maps of the environments, the ROS bags were played and the two ROS packages, GMapping an RTABMap_ROS were run. For what concern GMapping, *slam_ gmapping* is the node used to perform SLAM. RTABMap_ROS is much more complex, because several nodes are involved. The most important nodes used by RTABMap_ROS are:

- *rtabmap*, that is the main node and it is a wrapper for the desktop application. This is where the graph of the map is incrementally built and optimized when a loop closure is detected. The online output of the node is the local graph with the latest data added to the map [38];

- *rgbd_ odometry* and *icp_ odometry*, to perform the odometry task depending on the registration strategy used, visual or lidar odometry respectively;
- *map_ assembler*, to incrementally build the map returned by the *rtabmap* node;

obviously, there are also other nodes involved, to handle the information of the sensors and to perform other tasks, such as the visualization of the map.

A.2.6 Visualization

In the Rviz window it is possible to add topics to subscribe to, in order to receive the corresponding messages. Once all the topics of interest are selected, it is possible to save this configuration as a .rviz file, that can be loaded for the next times. RTABMap_ROS provides a Rviz configuration file, that shows the RGB-D images, the lidar measurements and the map, while for GMapping a new .rviz file was created, selecting the `/scan` topic for the lidar and the `/map` topic for the map.

A.2.7 Map saving

The *map_ saver* node of the *map_ server* package was used to save the map. It allows to store the map published by a ROS topic or service in a PGM file, that is a grayscale image. It saves also a YAML file which describes the map meta-data. With the command `rosrun map_server map_saver map:=topic_name` it is possible to save the map published with a given topic name. In the experiment, the topic names `/map` and `/rtabmap/octomap_grid` were used to save the 2D occupancy grid maps created by GMapping and RTABMap_ROS, respectively.

A.3 RTABMap_ROS launching parameters

To launch the ROS version of RTABMap, the ROS package launching command command was used:

```
roslaunch rtabmap_ros rtabmap.launch
rtabmapviz:=false
rviz:=true
```

```
use_sim_time:=true
queue_size:=100
frame_id:=base_link
visual_odometry:=true
subscribe_scan:=true
compressed:=true
odom_topic:=/odom
scan_topic:=/scan
rgb_topic:=/camera/color/image_raw_throttle
depth_topic:=/camera/aligned_depth_to_color/image_raw_throttle
camera_info_topic:=/camera/color/camera_info
depth_camera_info_topic:=/camera/aligned_depth_to_color/camera_info
args:="-d --Reg/Strategy 0 --RGBD/NeighborLinkRefining true"
```

The main command is `roslaunch rtabmap_ros rtabmap.launch` that is the standard ROS command to launch a package. It contains several arguments which can be summarized as following:

- `rtabmapviz:=false` and `rviz:=true` set the visualization tools, that is rviz instead of rtabmapviz which is the built-in RTABMap tool;
- `use_sim_time:=true` specifies that is a simulation, so a ROS bag is used and it sets RTABMap to use the bag clock;
- `queue_size:=100` sets to 100 the size of the queues of the callbacks to receive messages in RTABMap_ROS;
- `frame_id:=base_link` specifies the frame of reference, that is the baseline of the robot;
- `subscribe_scan:=true` specifies to subscribe also the laser scan;
- `compressed:=true` specifies that images are in a compressed format;
- `odom_topic:=/odom`,
  `scan_topic:=/scan`,

`rgb_topic:=/camera/color/image_raw_throttle`, `depth_topic:=/camera/aligned_depth_to_color/image_raw_throttle`, `camera_info_topic:=/camera/color/camera_info`, `depth_camera_info_topic:=/camera/aligned_depth_to_color/camera_info` define the name of the topics;

- `visual_odometry:=true` and `--Reg/Strategy 0` argument are the default in RTABMap but are highlighted because they set the algorithm to use a visual odometry strategy. To run RTABMap_ROS with the lidar odometry strategy, the two parameters are changed to `visual_odometry:=false` and `--Reg/Strategy 1`.

Bibliography

[1] Sherif A.S. Mohamed et al. "A Survey on Odometry for Autonomous Navigation Systems". In: *IEEE Access* PP (July 2019), pp. 1–1. DOI: 10.1109/ACCESS.2019.2929133.

[2] Saba Arshad and Gon-Woo Kim. "Role of Deep Learning in Loop Closure Detection for Visual and Lidar SLAM: A Survey". In: *Sensors* 21.4 (2021). ISSN: 1424-8220. DOI: 10.3390/s21041243. URL: https://www.mdpi.com/1424-8220/21/4/1243.

[3] Rana Azzam et al. "Pose-Graph Neural Network Classifier for Global Optimality Prediction in 2D SLAM". In: *IEEE Access* 9 (2021), pp. 80466–80477. DOI: 10.1109/ACCESS.2021.3084599.

[4] T. Bailey and H. Durrant-Whyte. "Simultaneous localization and mapping (SLAM): part II". In: *IEEE Robotics Automation Magazine* 13.3 (2006), pp. 108–117. DOI: 10.1109/MRA.2006.1678144.

[5] Michael Burri et al. "The EuRoC micro aerial vehicle datasets". In: *The International Journal of Robotics Research* 35 (Jan. 2016). DOI: 10.1177/0278364915620033.

[6] Cesar Cadena et al. "Simultaneous Localization And Mapping: Present, Future, and the Robust-Perception Age". In: *IEEE Transactions on Robotics* 32 (June 2016). DOI: 10.1109/TRO.2016.2624754.

[7] Changhao Chen et al. *A Survey on Deep Learning for Localization and Mapping: Towards the Age of Spatial Machine Intelligence*. June 2020.

[8] Liang Chen, Sheng Jin, and Zhoujun Xia. "Towards a Robust Visual Place Recognition in Large-Scale vSLAM Scenarios Based on a Deep Distance Learning". In: *Sensors* 21.1 (2021). ISSN: 1424-8220. DOI: 10.3390/s21010310. URL: https://www.mdpi.com/1424-8220/21/1/310.

[9] Technical University of Munich Computer Vision Group TUM Department of Informatics. *Useful tools for the RGB-D benchmark*. URL: https://vision.in.tum.de/data/datasets/rgbd-dataset/tools.

[10] Intel Corporation. *Depth Camera D435*. URL: https://www.intelrealsense.com/depth-camera-d435/.

[11] Intel Corporation. *Which Intel RealSense device is right for you?* URL: https://www.intelrealsense.com/which-device-is-right-for-you/.

[12] H. Durrant-Whyte and T. Bailey. "Simultaneous localization and mapping: part I". In: *IEEE Robotics Automation Magazine* 13.2 (2006), pp. 99–110. DOI: 10.1109/MRA.2006.1638022.

[13] A. Filatov et al. "2D SLAM quality evaluation methods". In: *2017 21st Conference of Open Innovations Association (FRUCT)*. 2017, pp. 120–126. DOI: 10.23919/FRUCT.2017.8250173.

[14] Philipp Fischer et al. *FlowNet: Learning Optical Flow with Convolutional Networks*. 2015. arXiv: 1504.06852 [cs.CV].

[15] F. Fraundorfer and D. Scaramuzza. "Visual Odometry : Part II: Matching, Robustness, Optimization, and Applications". In: *IEEE Robotics Automation Magazine* 19.2 (2012), pp. 78–90. DOI: 10.1109/MRA.2012.2182810.

[16] Dasong Gao, Chen Wang, and Sebastian Scherer. *AirLoop: Lifelong Loop Closure Detection*. 2021. arXiv: 2109.08975 [cs.RO].

[17] Xiang Gao et al. *14 Lectures on Visual SLAM: From Theory to Practice*. Publishing House of Electronics Industry, 2017.

[18] Andreas Geiger, Philip Lenz, and Raquel Urtasun. "Are we ready for Autonomous Driving? The KITTI Vision Benchmark Suite". In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2012.

[19] Giorgio Grisetti, Cyrill Stachniss, and Wolfram Burgard. “Improved Techniques for Grid Mapping With Rao-Blackwellized Particle Filters”. In: *IEEE Transactions on Robotics* 23.1 (2007), pp. 34–46. DOI: 10.1109/TRO.2006.889486.

[20] Giorgio Grisetti et al. “A tutorial on graph-based SLAM”. In: *IEEE Transactions on Intelligent Transportation Systems Magazine* 2 (Dec. 2010), pp. 31–43. DOI: 10.1109/MITS.2010.939925.

[21] J. M. M. Montiel H. Strasdat and A. J. Davison. “Visual SLAM: Why filter?” In: *Computer Vision and Image Understanding (CVIU)* (2012), pp. 65–72.

[22] Eddy Ilg et al. *FlowNet 2.0: Evolution of Optical Flow Estimation with Deep Networks*. 2016. arXiv: 1612.01925 [cs.CV].

[23] M. Labbé. *Real-Time Appearance-Based Mapping*. URL: https://introlab.github.io/rtabmap/.

[24] M. Labbé. *Simultaneous Localization and Mapping (SLAM) with RTAB-Map*. 2015. URL: https://introlab.3it.usherbrooke.ca/mediawiki-introlab/images/3/31/Labbe2015ULaval.pdf.

[25] Mathieu Labbé and François Michaud. “RTAB-Map as an open-source lidar and visual simultaneous localization and mapping library for large-scale and long-term online operation: LABBÉ and MICHAUD”. In: *Journal of Field Robotics* 36 (Oct. 2018). DOI: 10.1002/rob.21831.

[26] Xinyi Li and Haibin Ling. “PoGO-Net: Pose Graph Optimization With Graph Neural Networks”. In: *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*. Oct. 2021, pp. 5895–5905.

[27] *Lifelong SLAM Challenge*. URL: https://lifelong-robotic-vision.github.io/competition/SLAM.html.

[28] Qiang Liu et al. “Unsupervised Deep Learning-Based RGB-D Visual Odometry”. In: *Applied Sciences* 10.16 (2020). ISSN: 2076-3417. DOI: 10.3390/app10165426. URL: https://www.mdpi.com/2076-3417/10/16/5426.

[29] Vincenzo Lomonaco. *Why Continual Learning is the key towards Machine Intelligence*. URL: https://medium.com/continual-ai/why-continuous-learning-is-the-key-towards-machine-intelligence-1851cb57c308.

[30] F. Michaud M. Labbé. "Appearance-based loop closure detection in real-time for large-scale and long-term operation". In: *IEEE Transactions on Robotics* 29.3 (2013), pp. 734–745.

[31] Doaa Mahmoud et al. "3D Graph-Based Vision-SLAM Registration and Optimization". In: *International Journal of Circuits, Systems and Signal Processing* 8 (June 2014), pp. 123–.

[32] Adam Milstein. "Occupancy Grid Maps for Localization and Mapping". In: 2008.

[33] Tae Nam, Jae Shim, and Young Cho. "A 2.5D Map-Based Mobile Robot Localization via Cooperation of Aerial and Ground Robots". In: *Sensors* 17 (Nov. 2017), p. 2730. DOI: 10.3390/s17122730.

[34] Samer B. Nashed. "A Brief Survey of Loop Closure Detection: A Case for Rethinking Evaluation of Intelligent Systems". In: *NeurIPS 2020 Workshop: ML Retrospectives, Surveys Meta-Analyses (ML-RSA)* (2020).

[35] IEEE organization. *International Conference on Intelligent Robots and Systems (IROS)*. URL: https://www.ieee-ras.org/conferences-workshops/financially-co-sponsored/iros.

[36] F. Pedregosa et al. "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[37] Open Robotics. *Documentation*. URL: http://wiki.ros.org/.

[38] Open Robotics. *rtabmap_ros*. URL: http://wiki.ros.org/rtabmap_ros.

[39] D. Scaramuzza and F. Fraundorfer. "Visual Odometry [Tutorial]". In: *IEEE Robotics Automation Magazine* 18.4 (2011), pp. 80–92. DOI: 10.1109/MRA.2011.943233.

[40] Thomas Schöps, Torsten Sattler, and Marc Pollefeys. "BAD SLAM: Bundle Adjusted Direct RGB-D SLAM". In: *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2019, pp. 134–144. DOI: 10.1109/CVPR.2019.00022.

[41] X. Shi et al. "Are We Ready for Service Robots? The OpenLORIS-Scene Datasets for Lifelong SLAM". In: *2020 IEEE International Conference on Robotics and Automation (ICRA)*. 2020, pp. 3139–3145. DOI: 10.1109/ICRA40945.2020.9196638.

[42] Bruno Silva, Rodrigo Xavier, and Luiz Gonçalves. "Mapping and Navigation for Indoor Robots under ROS: An Experimental Analysis". In: (July 2019). DOI: 10.20944/preprints201907.0035.v1.

[43] Gonçalves Silvia Xavier. "Mapping and Navigation for Indoor Robots under ROS: An Experimental Analysis". In: (2019). DOI: doi:10.20944/preprints201907.0035.v1.

[44] Jürgen Sturm et al. "A benchmark for the evaluation of RGB-D SLAM systems". In: *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2012, pp. 573–580. DOI: 10.1109/IROS.2012.6385773.

[45] Sen Wang et al. "DeepVO: Towards end-to-end visual odometry with deep Recurrent Convolutional Neural Networks". In: *2017 IEEE International Conference on Robotics and Automation (ICRA)* (May 2017). DOI: 10.1109/icra.2017.7989236. URL: http://dx.doi.org/10.1109/ICRA.2017.7989236.

[46] Wikipedia. *Occupancy grid mapping*. URL: https://en.wikipedia.org/wiki/Occupancy_grid_mapping.

[47] Wikipedia. *Odometry*. URL: https://en.wikipedia.org/wiki/Odometry.

[48] Wikipedia. *Scalability*. URL: https://en.wikipedia.org/wiki/Scalability.

[49] Wikipedia. *Secure Shell*. URL: https://en.wikipedia.org/wiki/Secure_Shell.

[50] Xaxxon. *Oculus Prime - Setup ROS Networking Between Robot and Workstation*. URL: http://www.xaxxon.com/documentation/view/oculus-prime-ros-setup-networking-between-robot-and-workstation.

[51] Xaxxon. *Xaxxon AutoCRAWLER Autonomous Robot*. URL: http://www.xaxxon.com/xaxxon/autocrawler.

[52] Xaxxon. *Xaxxon OpenLIDAR Sensor*. URL: http://www.xaxxon.com/xaxxon/openlidar.

[53] Nan Yang et al. *D3VO: Deep Depth, Deep Pose and Deep Uncertainty for Monocular Visual Odometry*. 2020. arXiv: 2003.01060 [cs.CV].

[54] Zichao Zhang and D. Scaramuzza. "A Tutorial on Quantitative Trajectory Evaluation for Visual(-Inertial) Odometry". In: *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (2018), pp. 7244–7251.