PEOPLE'S DEMOCRATIC REPUBLIC OF ALGERIA

Ministry of Higher Education and Scientific Research

University of Amar Telidji - Laghouat

Faculty of Technology

Department of Electronics

MASTER THESIS

DOMAINE: Science & Technology

OPTION: Automation & Industrial Computing

Istighfar Chettih

Theme

Loop Closures in LiDAR-based SLAM systems

Jury members:

Mme. Djerfaf Fatima	Pr	President
Mr. Belkacem Rahmani	MCB	Examiner
Mme. Fatima Chouireb	Pr	Supervisor
Mr. Aissa Bencherif	MAB	Cosupervisor
Mr. Saadi Achour	PhD	Invited

2021 / 2022

Abstract

SLAM technology stands for Simultaneous Localization and Mapping. It is a process that uses sensors mounted on a robot to construct a map of an unknown environment while simultaneously keeping track of its own location within the map. Performing both localization and mapping simultaneously is a complex problem because the two processes are completely dependent of each other. Sensors provide information, but they always have some inaccuracy and noise which makes the SLAM process more complex. Therefore, a SLAM system suffers from long-term distance measurement drift, and the only way to overcome this issue consists of using loop closure detection algorithms to correct the drift error and build a globally consistent map of the environment. To do so, a loop closing technique based on Scan-Context descriptor is studied and compared to a conventional approach based on Scan-Matching, and it shows a sufficiently improved performance. Also, we investigate some recent SLAM implementations (both in ROS and in Matlab 2021b) commonly evaluated in indoor/outdoor environments.

Keywords: Simultaneous Localization and Mapping, LiDAR, Loop closure detection, Scan-Context, Scan-Matching, PGO.

Résumé

Le problème SLAM est un sujet très intéressant en robotique mobile, qui signifie la localisation et la cartographie simultanées (En anglais Simultaneous Localization and Mapping). Il s'agit d'un processus qui utilise des capteurs 'proprioceptifs et extéroceptifs' montés sur un robot pour construire une carte d'un environnement inconnu tout en estimant simultanément la trajectoire parcourue sur cette carte. En effet, effectuer simultanément la localisation et la cartographie est un problème complexe car les deux processus sont complètement dépendants l'un de l'autre. Les capteurs fournissent des informations, mais ils présentent toujours une certaine imprécision et du bruit, ce qui rend la tâche plus complexe. Par conséquent, la reconnaissance de lieu est un problème très important dans les systèmes SLAM, en particulier, cette reconnaissance fournit des candidats pour la fermeture de boucle, c.à.d., savoir si un robot est revenu à un endroit précédemment visité, ce qui est essentiel pour corriger l'erreur de dérive et construire une carte de l'environnement globalement cohérente. Pour ce faire, une technique de fermeture de boucle basée sur le descripteur Scan-Context est étudiée et comparée à une approche conventionnelle basée sur Scan-Matching, où elle montre des performances améliorées. Nous examinons également d'autres implémentations très récentes de SLAM (à la fois dans ROS et dans Matlab 2021b) couramment évaluées dans des environnements intérieurs/extérieurs.

Mots Clés : Localisation et cartographie simultanée, LiDAR, La détection de fermetures de boucles, Scan-Context, Scan-Matching, PGO.

ملخص:

ترمز تقنية SLAM إلى التموضع وبناء الخرائط في آن واحد في فضاءات مختلفة ليست معلومة مسبقاً، بإستخدام الليدار (LiDAR) و الأودوميتري (odometry) كوسائل لإدراك الفضاء المحيط، وهي عملية تمكننا من إنشاء خريطة لمحيط ما، وتحديد الموقع بدقة داخل هذه الخريطة. هذه ليست مهمة سهلة، وهي مستخدمة حاليا في أحدث الخوارزميات ثلاثية الأبعاد مثل: Scan-Context و LIO-SAM. تعاني مشكلة SLAM من الانجراف في قياس المسافات على المدى الطويل والطريقة الوحيدة لتصحيح ذلك هي من خلال تحسين المسار استنادًا إلى اكتشاف إغلاق الحلقة والتي نعتبرها محور هذه المذكرة. تعالج هذه المذكرة دقة خوارزميات SLAM من خلال المقارنة بينها، حيث قمنا بتسجيل مجموعة بيانات ثنائية البعد باستخدام نظام تشغيل الروبوتات ROS والذي يمكننا من محاكات أوساط متفاوتة التعقيد وأخذ القياسات باستخدام TurtleBot3. من خلال هذه البيانات قارنا بين خوارزميتي GMapping و GraphSLAM. بالنسبة للخوارزميات ثلاثية البعد (LIO-SAM, Scan-Context, Scan-Matching)، فقد قمنا باختبارها ومقارنتها من خلال تجربتها على أربع حزم من البيانات مخصصة لإخبار هذا النوع من الخوارزميات. أعطت الخوارزميات نتائج مذهلة خاصة LIO-SAM و Scan-Context. أثناء عملية تسجيل البيانات ثنائية البعد واجهتنا مشكلة صعوبة التحكم في الروبوت باستخدام teleoperation، تنعكس هذه المشكلة على كبر حجم حزم البيانات المسجلة (أحياناً يصل إلى 60 جيجابايت). تمكنا من حل هذه المشكلة من خلال استعمال الملاحة المستندة إلى الخريطة الناتجة عن خوارزمية GMapping.

كلمات مفتاحية: التموضع وبناء الخرائط في آن واحد، اكتشاف إغلاق الحلقة، ليدار، Scan-Context، Scan-Matching، PGO.

تشكر وعرفان

بادئ ذي بدء، الشكر لله أولاً وآخراً على كل نعمة جادها على أمته إستغفار شتيح، فلولاه ما كان لي الشرف اليوم لأكتب هذه الكلمات، فالحمد لله حمداً واسعاً يليق بجلال قدره وعظيم سلطانه.

إن ما أنا عليه الآن، ما هو إلا ثمرةٌ من ثمرات والدي العزيزين الكريمين "البروفسور بن يوسف شتيح والأستاذة خلف الضاوية" حفظهما الله ورعاهما وثبت خطاهما وغمرني بعطفهما ما حييت.

وبعدُ، فإني أتقدم بالشكر الجزيل والامتنان العريض إلى أولئك الذين وجدت فيهم سنداً ومدداً ونُصحاً خالصاً كي أمضي نحو أهدافي في عزيمةٍ وإصرارٍ، واثقةَ الخطى شامخةَ الرأس.

إن ما خطه القدر لي على أيديهم كان نعمة مضاعفة وعلى رأسهم مشرفتي الأستاذة الدكتورة شويرب فاطمة حفظها الله ورعاها والدكتور سعدي عشور، اللذان طوقاني بكرمهما وحفاني بالنصح والتوجيهات للمضي نحو الأفضل. فاللهَ أسألُ أن يسدد خطاهما ويحفظ أهلهما.

كما لا يفوتني في هذا المقام أن أتقدم بشكري الخالص لأعضاء لجنة المناقشة، الأستاذة الدكتورة جرفاف فاطمة والدكتور رحماني بلقاسم الذين لهم الفضل علي.

الشكر الجزيل للأستاذة الدكتورة صليحة شتيح على ما قدمته لي من دعم ونصح، فكانت سنداً لي في مشواري الدراسي. كما أشكر الأستاذ الدكتورمحمد شتيح الذي لم يدخر جهداً في مساعدتي وقدم لي يد العون في أحلك الظروف، فكان لي طوق نجاة، كما لا يفوتني أن أشكر صديقتي ندى و حياة على ما قدماه لي من عون.

الشكر موصول إلى جامعة عمار ثليجي وطاقمها التعليمي والإداري كافة وإلى طاقم كلية التكنولوجيا خاصةً "العميد الأستاذ الدكتور علي شقنان والأستاذ الدكتور لهذب محمد" على كل المجهودات المبذولة.

أشكر من لم يخط قلمي بشكره ليس سهواً ولا نسياناً إنما كانوا للخير عنواناً فمثلهم كقول الله تعالى في سورة غافر ﴿ وَلَقَدْ أَرْسَلْنَا رُسُلًا مِّن قَبْلِكَ مِنْهُم مَّن قَصَصْنَا عَلَيْكَ وَمِنْهُم مَّن لَّمْ نَقْصُصْ عَلَيْكَ ﴾ سورة غافر، الآية 78 - صدق الله العظيم-.

Contents

List of Figures

General Introduction

Robotics is considered as a combination of several areas of knowledge. Besides, it is crucial in particular fields such as: space exploration or rescuing tasks in disasters; which are hard for humans. In fact, a fundamental problem in robotics is the awareness of the environment. Thus, for robots to perform complex tasks autonomously, it is often required that a robot needs to navigate through arbitrary, previously unknown environments in a stable way. For navigation, a mobile robot needs to know information about the environment; for instance, if there are obstacles in the way. The robot gathers data about the surrounding using sensors, based on this data it can create and find out its own location.

The past decades have seen rapid and exciting process to solve the Simultaneous Localization And Mapping (SLAM) problem. Indeed, many robotics applications have gained ground, such as surgery augmented reality and self-driving cars. Thus, through the SLAM process, not only the trajectory is estimated, but also, a map of the surrounding scene is reconstructed in real-time. Mapping is a trivial process when the robot is aware of its location; however, performing both localization and mapping simultaneously introduces new challenges, because the two processes are completely dependent of each other.

Nowadays, a large variety of possible sensors like LiDAR, IMUs, and cameras have been proposed in the SLAM context. Sensors provide information, but they always have some inaccuracy and noise which makes the SLAM process more complex.

In the literature, two kinds of approaches can be distinguished, namely incremental and global ones. Incremental approaches use a limited history of local sensor data and usually employ some recursive state filtering thereon. The main disadvantage of incremental methods remains the accumulation of error over time. This disadvantage

can only be overcome by using global information which can be given by e.g., a GPS receiver or a global map. While GPS is inaccurate and not available in all environments, generating a detailed global map in advance is very costly.

The reasons mentioned above, motivated the development of SLAM methods and support place-recognition to close a loop and eliminate accumulated errors. Place recognition is a very important issue in SLAM systems, in particular, this recognition provides candidates for loop closure, i.e., knowing if a robot has returned to previously visited places, which is essential to correct the drift error and build a globally consistent map of the environment.

Since SLAM benefits from dense data, the choice of the used sensors is practically limited to cameras and laser scanners. In this work, we concentrate on the second, where typically both the precision and the field of view are larger. Filtering techniques were proposed to refine the produced offline data. One of the next steps could be to detect and handle loop-closure appropriately as e.g., in Graph-SLAM.

The objective of this Master thesis is to study and compare loop closure algorithms proposed in very recent works such as the Scan-Context algorithm and to integrate them into a LiDAR based GraphSLAM system. This system will also be implemented on the TurtleBot robot simulated by Gazebo in ROS (Robot Operating System).

Approach and outline

This thesis is spread over 3 chapters which allow describing all the key points of our work.

- **Chapter 1:** introduces in detail different types of robots and presents the concept of mobile robot navigation. Also, this chapter describes the basic concepts and challenges of LiDAR based SLAM systems.

- **Chapter 2:** In this chapter we summarize the loop closing problem and review some state-of-the-art loop closure detection techniques. In this chapter we focused on the Scan-Context algorithm through simulation.

- **Chapter 3:** In this chapter we implement and evaluate the investigated algorithms on different types of data using both Matlab and ROS.

Finally, we end our work with a conclusion and give an outlook to future perspectives.

Chapter 1

Robotics and SLAM

1.1 Introduction

Robots are systems used to perform different tasks, from cleaning a house to exploring a new planet. Robotics is the science and engineering discipline required to make a robot technically work. Mechanical, electrical, and control engineering of the physical structure, electronic hardware, and control software are all parts of robotics [1]. Autonomous mobile robots, which have always been and still are a hot issue in scientific research, this type of robots is frequently built for smart industrial contexts in which they interact with humans. Mobile robots must overcome three key difficulties to operate autonomously in an unknown environment "localizatsion, mapping, and path planning".

In order to enable the reader to delve into more important details to understand how mobile robots navigate. The chapter is split into sections as follows:

In section 1.2, we introduce in detail different types of robots, and in section 1.3 presents the concept of mobile robot navigation. Sequentially, we present the main idea of LiDAR-based SLAM including LiDAR sensor and its applications in section 1.4. Mapping, Localization, and SLAM are basic concepts, all are introduced in section 1.5, with the mathematical part of SLAM and the most familiar Algorithms. Section 1.6 is reserved to introduce LiDAR sensor and Scan-Matching algorithm. The Loop closure concept which is one of the main components of SLAM is presented in Section 1.7. Finally, a conclusion of this chapter is attached in section 1.8.

1.2 Robotics

Robotics is a research field that has gained much popularity in recent years, due to the growing interest of big companies. This increasing popularity has also promoted the creation of new corporations dedicated to particular markets in robotics. As a consequence of these facts, robotics has evolved from a purely academic field to be present in people's daily lives. Robotic applications cover an extensive range of possibilities in different areas, such as; for instance, surgery, space exploration, surveillance, security, personal assistance, an inspection of structures, or rehabilitation.

The term 'robot' has many connotations for various people. Science fiction books and movies have influenced many people's motions of what a robot should be and what it can do. Regrettably, robotics practice lags far behind this popular perception. They will be a significant technology in this century; in fact, vacuum cleaning robots have been on the market for over a decade, and self-driving cars are on the way such as Tesla cars. These are the forerunners of a wave of intelligent machines that will begin to appear in our homes and workplaces in the near to medium term.

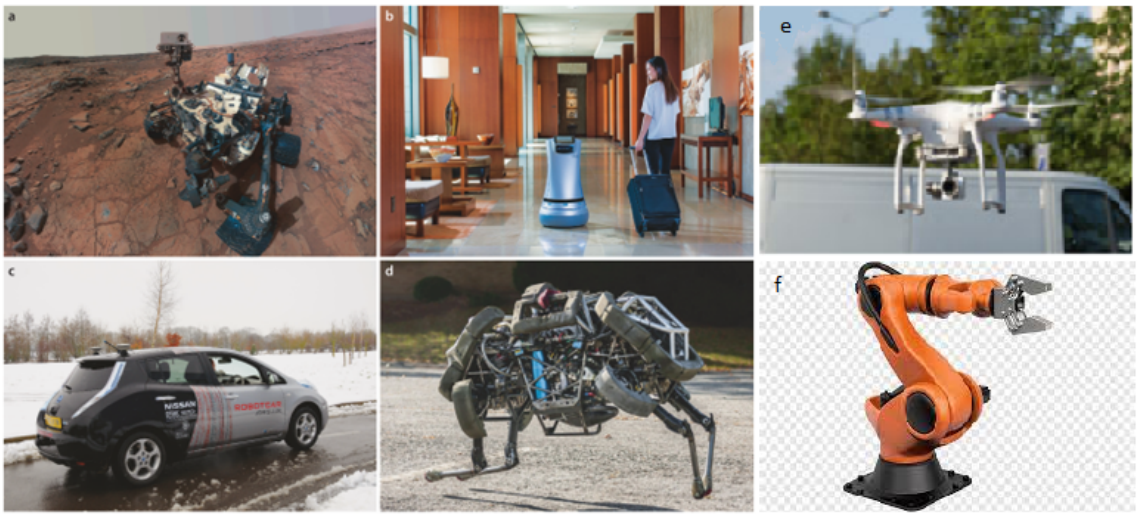


Figure 1.1: Mobile robots. a) NASA's Mars Science Laboratory Curiosity Rover Mission; b) Savioke Relay delivery robot; c) self-driving car; d) Cheetah legged robot; e) Unmanned Aerial Vehicle (UAV); f) Industrial robot Robotic arm.

Despite the fact that there are numerous definitions for the term, only a handful are accurate. A definition that will serve us well, is: 'a goal-oriented machine that

can sense, plan and act' [1]. By definition, an autonomous robot is an agent that can perform behaviors or tasks with a high degree of autonomy and/or without human intervention. This kind of robot is particularly interesting in fields such as space exploration or rescuing tasks in disasters, which are hard to reach for humans. Fig. 1.1 illustrates some of the different robots developed recently.

The separation between autonomous and non-autonomous robots is possible; nevertheless, these robots usually have quite different duties to complete. In order to complete a task, autonomous robots do not require any additional input from the user. It is critical for autonomous robots to be able to learn about their environment and react appropriately to changes in that specific environment. Autonomous robots frequently have a higher level of complexity than non-autonomous robots, making them more relevant for scientific research.

In order to perform more complex tasks, robots usually need to be moved, which leads us to a special kind of robot that is mobile. Therefore, a mobile robot is an automatic machine that has the ability to move around in its environment and it is not fixed to a physical location. The type of movement of a mobile robot is determined according to its locomotion mechanisms. Although a lot of actuators have been proposed by roboticists over the years, the election of the locomotion mechanisms to be incorporated into a mobile vehicle is normally governed by the domain of the application. As a result, mobile robots can be divided into four types based on their intended use:

- **Terrestrial robots:** walk on the ground and are designed to take advantage of solid contact with a surface. Initially, the most common terrestrial vehicles were wheeled (fig. 1.2), Humanoid robots, which are primarily centered on bipedal locomotion, have recently piqued people's interest;

- **Aquatic robots:** operate in water, either at the surface or underwater. This kind of robot is mainly equipped with water jets or propellers as locomotion mechanisms. The importance of theire robots arises from the fact that water, particularly in situations like oceans, is a medium that people have difficulty accessing (fig. 1.3).

Figure 1.2: Mobile ground robots: a) The Roomba robotic vacuum cleaner; b) Tartan racing team's autonomous car that won the Darpa Urban Grand Challenge, 2007 (Carnegie-Mellon University) [1].

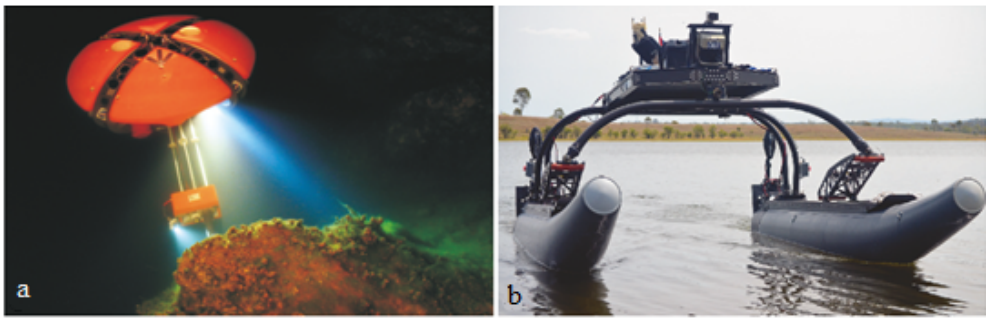


Figure 1.3: Mobile water robots: a) DEPTHX: Deep Phreatic Thermal Explorer; b) Autonomous Surface Vehicle.

- **Aerial robots:** which can fly. They share many issues and locomotion mechanisms with aquatic robots. This kind of robot has become an important research field in the last few years due to the intense development of Micro Aerial Vehicles (MAV) as illustrated in fig. 1.4.

- **Space robots:** are designed to operate in the microgravity of outer space. The locomotion strategies used for these robots vary, given that some of them are mainly devised for flying and others for ground exploration (fig. 1.5).

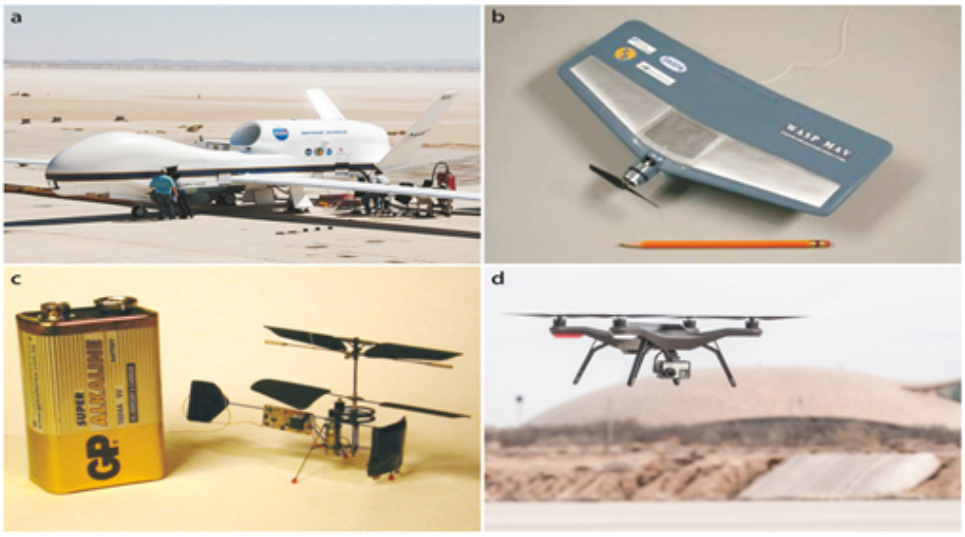


Figure 1.4: Flying robots. a) Global Hawk unmanned aerial vehicle (UAV) (photo courtesy of NASA), b) a micro air vehicle (MAV); c) a 1-gram co-axial helicopter with 70 mm rotor diameter; d) a quadrotor which has four rotors and a block of sensing and control electronics in the middle.

1.3 Mobile Robot Navigation

Several problems arise when converting a mobile robot into an autonomous platform. In addition to the motion mechanisms required for the task at hand, the robot needs to perceive information from its environment and process it in an intelligent way to plan routes, reach places and avoid dangerous situations. In this context, these tasks lead us to another important concept in mobile robotics navigation.

Mobile robot navigation can be roughly described as the process of determining a suitable and safe path between a starting and a goal point for a robot traveling between them. Likewise, navigation can be seen as a combination of, among others, three fundamental tasks [2]:

- **Localization:** which denotes the ability of the robot to know its position and orientation with regard to its environment;
- **Path planning:** which deals with finding the most optimal path between two points of the environment. Note that this task usually includes the ability to avoid obstacles;

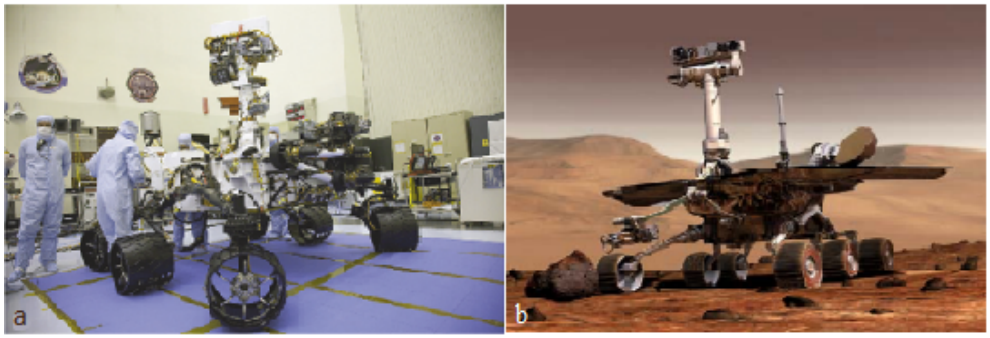


Figure 1.5: The Mars exploration rovers, spirit and opportunity, with a manipulator arm in front.

- **Mapping:** is in charge of creating maps, and abstract representations of the robot's environment, using the sensors the robot is equipped with.

Given the nature of each of these tasks, we can argue that maps are essential components in most robotic navigation systems; since the other tasks depend significantly on them. For instance, path planning, to be correctly carried out, require a map of the environment and the location of the robot within the map. It is also true that some authors have proposed reactive navigation techniques encompassed in a category called mapless navigation systems, where there is no global representation of the environment and the world is perceived as the system navigates through it.

Nonetheless, most parts of the solutions proposed during the last years belong to the group of map-based navigation systems, where a map of the environment is needed to navigate; fig. 1.6 shows the Time-lapse photograph of a Roomba robot cleaning a room as an example of daily navigation of a robot.

1.4 Perception and sensors used

A robot senses its surroundings and plans an action based on that knowledge. Driving a mobile robot somewhere may be the action. Thus, sensing is become critical to robots. In fact, Perception generally consists of collecting pieces of information from different kinds of sensors, to collect its data and acquire accurate information about the environment where the robot is evolving. It is a prerequisite and essential for localization and mapping issues.

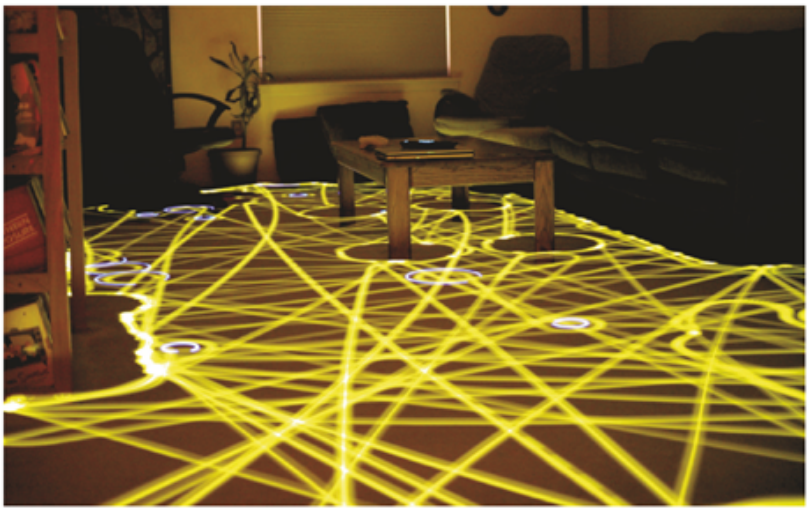


Figure 1.6: Time-lapse photograph of a Roomba robot cleaning a room.

The choice of a perception system is often dependent on the evoluting environment of the mobile robot as well as the cost of integrating sensors into the robot. The desired precision and the frequency of acquisition are all factors that increase the cost of a sensor. The classification of sensors is generally made of two families: proprioceptive sensors and exteroceptive sensors; while the most popular one is the laser rangefinder (see fig. 1.7). which tells the distance to the nearest object in the nearest direction or sector; such as sonars, radar, and LiDAR-based systems.

1.4.1 Proprioceptive sensors

They measure the state of the robot itself, such as; the angle of the joints on a robot arm, the number of wheel revolutions on a mobile robot, or the current drawn by an electric motor. which provides specific information to the internal behavior of the robot, i.e., determining its state at a given moment.

There are various types of proprioceptive sensors including:

- **Displacement sensors:** which include odometers, accelerometers (refer to fig. 1.7), Doppler radars, optical meters, etc. This category makes it possible to measure elementary displacements and variations in velocity or acceleration on rectilinear or curvilinear trajectories.

- **Attitude sensors:** measure two types of data; heading angles and roll and

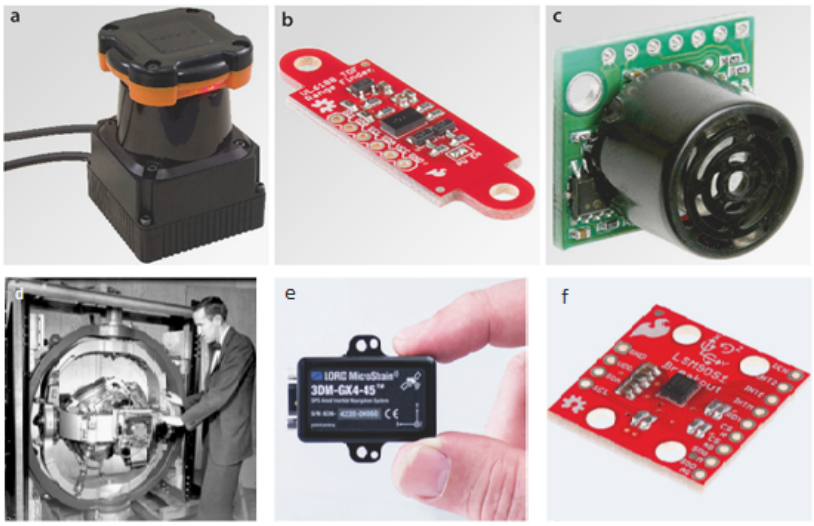


Figure 1.7: Robot rangefinders. a) A scanning laser rangefinder with a maximum range of 30 m, an angular range of 270 deg in 0.25 deg intervals at 40 scans per second; b) a low-cost time-off light rangefinder with maximum range of 20 cm at 10 measurements per second; c) a low-cost ultrasonic rangefinder with maximum range of 6.5 m at 20 measurements per second; d) Space Inertial Reference Equipment from 1953; e) A modern inertial navigation system the LORD MicroStrain 3DM-GX4-25 has triaxial gyroscopes, accelerometers and magnetometer; f) IMU.

pitch angles. They are mainly made up of gyroscopes, gyrometers, composite inertial sensors, inclinometers, magnetometers, etc. These sensors are mostly of the inertial type.

1.4.2 Exteroceptive sensors

They provide information about the world outside the robot; they measure the state of the world concerning the robot. In a way that the sensor might be a simple bump sensor on a robot vacuum cleaner to detect collision. Also, it could be a GPS receiver that measures distances to an orbiting satellite constellation or a compass that measures the direction of the Earth's magnetic field vector relative to the robot. It might also be an active sensor that emits acoustic, optical, or radio pulses in order to measure the distance to points in the world based on the time taken for a reflection to return to the sensor, e.g., LiDAR, camera, etc (see fig. 1.8) Both LiDAR and RADAR systems are

compatible with autonomous vehicles.

Figure 1.8: Right: commercial Laser scanner; Left: RGBD camera.

1.5 Mapping, Localization, and SLAM

Maps could be provided to a robot a priori, but sometimes this is not possible, and then the robot is required to build its own representation of the unknown environment. This process, as mentioned previously, is called mapping and it is one of the main topics in our work. As far as robotic mapping is concerned, two main paradigms are generally accepted: metric and topological mapping.

While metric maps describe the position of the robot in the world, along with the detected objects according to a global coordinate system. Topological maps represent the environment in an abstract manner using a graph. Despite mapping and localization can be performed as independent tasks, they are closely related. As a result of the mapping process, a representative map of the environment is generated while the localization process estimates the pose of the robot within this map, according to the sensor data perceived from the sensors.

As a consequence, both processes can be used for navigation tasks, and are of special interest for autonomous vehicles such as self-driving cars, which need to be able to operate without any human intervention. The estimated pose of the structures and the obstacles of the environment need to be known to build a map. Whereas, during localization, the pose against a reference map is estimated. In this case, the map of the working scenario must be available before starting the navigation, which limits the autonomy of the vehicle. To solve this problem, several approaches have

been proposed where both tasks take place at the same time, creating an incremental map of an unknown environment while localizing the robot within this map. These techniques are generically known as Simultaneous Localization and Mapping (SLAM).

1.5.1 Map categories

Robot mapping can be defined as the process of generating a representation of the environment useful for the task at hand. Despite this process seeming to be an easy task for humans, it is much more difficult for mobile robots. Therefore, mapping is currently a very active research field. An appropriate map for an autonomous robot should be constructed using the same sensors that the robot employs to observe the world. These sensors are usually corrupted by noise and interferences, which makes it more difficult for the data to associate between measurements and maps. Due to this reason, mapping approaches are heavily influenced by loop closure detection.

There are several ways to represent a map. The accuracy of the map depends on the information requirements. Three main paradigms for mobile robot mapping are usually accepted [2]:

- **Metric maps:** this kind of map represents the world as accurately as possible. They maintain a high amount of information about environment details, such as distances, measures, sizes, and so on, and they are referenced according to a global coordinate system. The main drawbacks of this approach are the processing time and the storage needs, which makes its use in some real-time applications more difficult.

- **Topological maps:** this approach generates an abstract representation of the world surrounding, usually as a graph with poses and links between them. poses represent environment locations with similar features and links are relationships or possible actions to take between the different locations. These maps are simpler and more compact than metric maps and require much less space to be stored.

- **Hybrid maps:** this last paradigm tries to maximize the advantages and minimize the problems of each kind of map alone and combine them in a different mapping technique.

Table 1.1: Advantages and disadvantages of metric and topological maps.

Feature	Metric maps	Topological maps
Accuracy	Yes	No
Storage needs	High	Low
Path planning	Complex	Simple
Optimal routes	Yes	No
CPU Needs	High	Low

Also; "Occupancy grid maps" and "feature-based maps" are two extensively used ways to represent metric maps of the environment in SLAM research.

Occupancy grid maps: are discrete cell maps that may contain 2D and 2.5D even 3D information. By separating the map into grids, each cell of the grid can be either occupied or free. Occupancy grid maps can show various forms of the environment. However, the divided grids demand a large amount of memory, and the update process is computationally intensive.

Landmark-based maps: identify and maintain the location of certain features in the environment. They may be corners, line segments, or points, and are called landmarks. Because of their compactness, feature-based maps are useful in SLAM research. Line-segment-based maps, which can represent structured environments adequately, are often employed in indoor environment applications due to their small memory requirement and low computational cost.

1.5.2 Mathematical solutions to the SLAM problem

This section provides an introduction to the main solutions of SLAM and the extensive research on it that has been undertaken over the past decade. In this dissertation, we are going to describe the probabilistic form of the SLAM problem, essential solution methods, and significant implementations.

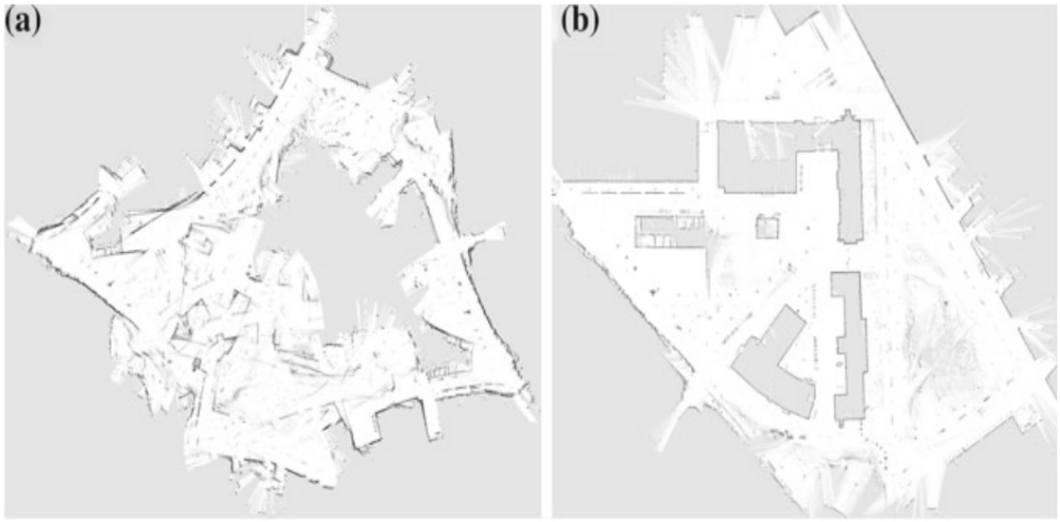


Figure 1.9: Example of metric map generated using a laser range finder. (a) Resulting map using raw sensor data. (b) The same map improved using, among other information, the detected loop closures for correcting the trajectory [2].

Preliminaries

Since SLAM is a mathematical problem, variables are utilized to explain the various aspects of its algorithm. Each term describes a set of data that can be used to manipulate one or more other sets of data using different algorithms. These terms are described below.

Consider a mobile robot moving through an environment taking relative observations of several unknown landmarks using a sensor located on the robot as shown in fig. 1.10. At a time, instant k, the following quantities are defined [3]:

- x_k : the state vector describing the location and orientation of the vehicle;
- u_k : the control vector, applied at time $k-1$ to drive the vehicle to a state x_k at time k;
- m_i : a vector describing the location of the ith landmark whose true location is assumed time-invariant;
- z_{ik} : an observation taken from the vehicle of the location of the ith landmark at time k. When there are multiple landmark observations at any one time or

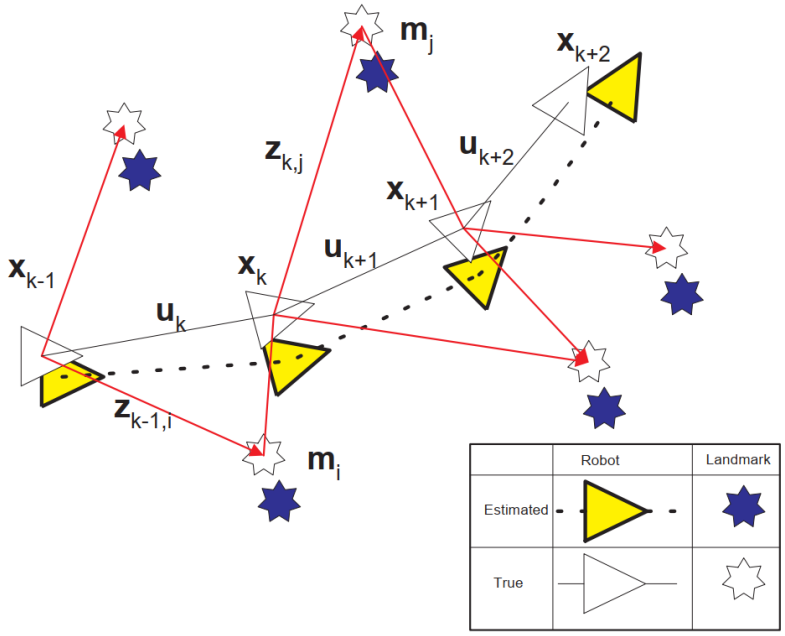


Figure 1.10: The essential SLAM problem. A simultaneous estimate of both robot and landmark locations is required. The true locations are never known or measured directly. Observations are made between true robot and landmark locations [3].

when the specific landmark is not relevant to the discussion, the observation will be written simply as z_k.

In addition, the following sets are also defined:

- $X_{0:k} = \{x_0, x_1, \cdots, x_k\} = \{X_{0:k-1}, x_k\}$: the history of vehicle locations;
- $U_{0:k} = \{u_1, u_2, \cdots, u_k\} = \{U_{0:k-1}, u_k\}$: the history of control inputs;
- $m = \{m_1, m_2, \cdots, m_n\}$: the set of all landmarks;
- $Z_{0:k} = \{z_1, z_2, \cdots, z_k\} = \{Z_{0:k-1}, z_k\}$: the set of all landmark observations.

Probabilistic SLAM

In probabilistic form, the Simultaneous Localization and Map Building (SLAM) problem requires that the probability distribution to be computed for all times k.

$$P\left(x_k, m \mid Z_{0:k}, U_{0:k}, x_0\right) \tag{1.1}$$

This probability distribution describes the joint posterior density of the landmark locations and vehicle state (at time k) given the recorded observations and control inputs up to and including time k together with the initial state of the vehicle. In general, a recursive solution to the SLAM problem is desirable. Starting with an estimate for the distribution $P\left(\mathrm{x}_{k-1},\ \mathrm{m} \mid \mathrm{Z}_{0:k-1}, \mathrm{U}_{0:k-1}\right)$ at time $k-1$, the joint posterior, following a control u_k and observation z_k, is computed using Bayes Theorem. This computation requires that a state transition model and an observation model are defined to describe the effect of the control input and observation respectively. The observation model describes the probability of making an observation z_k, when the vehicle location and landmark locations are known, and is generally described in the form:

$$P\left(z_k \mid x_k, m\right) \tag{1.2}$$

It is reasonable to assume that once the vehicle location and map are defined, observations are conditionally independent given the map and the current vehicle state.

The motion model for the vehicle can be described in terms of a probability distribution on state transitions in the form:

$$P\left(\mathrm{x}_k \mid \mathrm{x}_{k-1}, \mathrm{u}_k\right) \tag{1.3}$$

That is the state transition is assumed to be a Markov process in which the next state x_k depends only on the immediately preceding state x_{k-1} and the applied control u_k, also is independent of both the observations and the map.

The SLAM algorithm is now implemented in a standard two-steps recursive (sequential) prediction (time-update), correction (measurement-update) form:

- **Time-update:**

$$P\left(\mathrm{x}_k,\ \mathrm{m} \mid \mathrm{Z}_{0:k-1}, \mathrm{U}_{0:k}, \mathrm{x}_0\right) = \int P\left(\mathrm{x}_k \mid \mathrm{x}_{k-1}, \mathrm{u}_k\right) \times P\left(\mathrm{x}_{k-1},\ \mathrm{m} \mid \mathrm{Z}_{0:k-1}, \mathrm{U}_{0:k-1}, \mathrm{x}_0\right) \mathrm{dx}_{k-1}. \tag{1.4}$$

- **Measurement Update:**

$$P\left(\mathrm{x}_k,\ \mathrm{m} \mid \mathrm{Z}_{0:k}, \mathrm{U}_{0:k}, \mathrm{x}_0\right) = \frac{P\left(\mathrm{z}_k \mid \mathrm{x}_k,\ \mathrm{m}\right) P\left(\mathrm{x}_k,\ \mathrm{m} \mid \mathrm{z}_{0:k-1}, \mathrm{U}_{0:k}, \mathrm{x}_0\right)}{P\left(\mathrm{z}_k \mid \mathrm{z}_{0:k-1}, \mathrm{U}_{0:k}\right)}. \tag{1.5}$$

Equations (1.4) and (1.5) provide a recursive procedure for calculating the joint posterior $P(\mathrm{x}_k,\ \mathrm{m} \mid \mathrm{Z}_{0:k}, \mathrm{U}_{0:k}, \mathrm{x}_0)$ for the robot state x_k and map m at a time k based on all observations $Z_{0:k}$ and all control inputs $U_{0:k}$ up to and including time k. The recursion is a function of a vehicle model $P(\mathrm{x}_k \mid \mathrm{x}_{k-1}, \mathrm{u}_k)$ and an observation model $P(\mathrm{x}_k \mid \mathrm{Z}_{0:k}, \mathrm{U}_{0:k},\ \mathrm{m})$.

It is worth noting that the map building problem, may be formed as computing the conditional density $P(\ \mathrm{m} \mid \mathrm{X}_{0:k}, \mathrm{Z}_{0:k}, \mathrm{U}_{0:k})$. This assumes that the location of the x_k vehicle is known (or at least deterministic) at all times, the original location must be known. A map m is then constructed by fusing observations from different locations. Conversely, the localization problem may be formulated by computing the probability distribution $P(\mathrm{X}_k \mid \mathrm{Z}_{0:k}, \mathrm{U}_{0:k},\ \mathrm{m})$. This assumes that the landmark locations are known with certainty and the objective is to compute an estimate of vehicle location with respect to these landmarks.

Main Solutions

Solutions to the probabilistic SLAM problem involve finding an appropriate representation for the observation model equation (1.2) and motion model equation (1.3), which allows efficient and consistent computation of the prior and posterior distributions in equations (1.4) and (1.5). By far, the most common representation is in the form of a state-space model with additive Gaussian noise, leading to the use of the Extended Kalman filter (EKF) to solve the SLAM problem.

One important alternative representation is to describe the vehicle motion model in equation (1.3), as a set of samples of a more general non-Gaussian probability distribution. This leads to the use of the Rao-Blackwellised particle filter, or Fast-SLAM algorithm [8, 12], to solve the SLAM problem. While EKF-SLAM and FastSLAM are the two most important solution methods, newer alternatives, which offer much potential, have been proposed in the state of the art.

EKF-SLAM

There are three main paradigms, Kalman filters including EKF, particle filters, and graph-based SLAM. The first two are also referred as filtering techniques, where the position and map estimates are augmented and refined by incorporating new measure-

ments when they become available. Due to their incremental nature, these approaches are generally acknowledged as online SLAM techniques. Conversely, graph-SLAM estimates the entire trajectory and the map from the full set of measurements, and it is called the full SLAM problem. This section summaries major used algorithms with short introductions as follows:

- **Prediction step:** consists of predicting the state of the system at time k, from the robot's motion model using the corrected estimate of time $(k-1)$;
- **Step of correction:** during which the system state is corrected at time k, using the information from the exteroceptive sensors received at time k.

Nevertheless, the position of the robot and the landmarks, are represented by a state vector X in this method. Because Gaussian noise affects the measurements, the state vector is a vector of normal random variables. The EKF-SLAM method is based on describing vehicle motion in the form [3]:

$$P\left(x_k \mid x_{k-1}, u_k\right) \Leftrightarrow x_k = f\left(x_{k-1}, u_k\right) + w_k, \tag{1.6}$$

where $f(.)$ models vehicle kinematics and w_k are additive, zero mean uncorrelated Gaussian motion disturbances with covariance Q_k. The observation model is described in the form:

$$P\left(z_k \mid x_k, m\right) \Leftrightarrow z(k) = h\left(x_k, m\right) + v_k, \tag{1.7}$$

where $h(.)$ describes the geometry of the observation and v_k are additive, zero mean uncorrelated Gaussian observation errors with covariance R_k.

Particle filter SLAM

Particle filter-based SLAM is an alternative solution to EKF-SLAM. The uncertainty of the robot pose is modeled by a number of different weighted particles (hypothesis) and each particle maintains its own map. Usually, the map is processed by using EKF and the filter becomes Rao–Blackwellized particle filter [13]. Each particle is then weighted again according to its likelihood with measurements. The particles that capture negligible weights are replaced by new particles in the proximity of the particles with higher weights.

Thus, the particle filter-based SLAM techniques overcome the linearization error of EKF and do not limit to Gaussian assumptions. Since they conduct the data association for each particle, partial erroneous data association will not lead to disastrous consequences. However, a large number of particles is required to be maintained in order to acquire a certain level of accuracy in the estimation process. Therefore, a particle filter is a recursive filter that allows to estimate the state of a posteriori using a set of particles. Unlike parametric filters like the Kalman filter, a particle filter represents a distribution by a set of samples or particles created from this distribution. Each particle in the trajectory is associated with a set of landmarks. A particle filter is thus able to process strongly nonlinear systems with non-Gaussian noise. The complexity of particle filtering calculations increases exponentially with the number of landmarks in the environment, which is a major problem in the context of a real-time application.

After all, in the particle filter-based solution to the SLAM problem, the loop closing [14] is done by matching the newly found observation data to old scans based on the pose distribution of the particle. If the particle crosses a certain point where it believes to have been before, on a positive scan match, its belief will increase drastically and resampling can be carried out.

The Monte Carlo Localization approach (MCL), takes advantage of using enough particles, while the particle filter can approximate arbitrarily complex and multi-model probability distributions. As a result, it is a fast, accurate, and memory-intensive global localization method, in which the environment is known.

Graph-SLAM

Graph-based SLAM is a method to describe the SLAM problem as a graph. The main purpose of the algorithm is to correct noise, that is present in sensor data. These errors need to be corrected and can be described by a linear system of equations which has the size of the number of poses that the robot has visited. In addition, the number of poses visited can be large and the computational complexity of the algorithms used to solve large matrices increases quadratically. By restricting the algorithm on the amount of loop closing the complexity has been reduced to only grow linear.

However, a key disadvantage of filter techniques such as those discussed above, is that data is discarded after processing. This removes the possibility of revisiting

relevant data while building the map, in case corrections are in order. Thus, Graph-SLAM introduced by [9] seeks to avoid this issue by exploiting that the posterior of the full SLAM problem naturally forms a sparse graph. The obtained graph leads to a sum of nonlinear constraints, which when optimized yields a maximum likelihood map with a set of the corresponding robot poses.

An alternative approach to the SLAM problem is to separate it into two components: a front end and a back end, connected by a pose graph as shown in fig. 1.11. The robot's path is considered to be a sequence of distinct poses and the task is to estimate those poses. Constraints between the unknown poses are based on measurements from a variety of sensors including odometry, laser scanners, and cameras. The problem is formulated as a directed graph; a node corresponds to a robot pose or a landmark position. An edge between two nodes represents a spatial constraint between the nodes derived from some sensor data. As the robot progresses it compounds an increasing number of uncertain relative poses so that the cumulative error in the pose of the nodes will increase. Thus, by the time the robot reaches node number 4 the error is significant (see fig. 1.11). the estimated relative pose is based on the chain of relative poses from odometry.

The back-end algorithm will then pull all the nodes closer to their correct pose. The front end adds new nodes as the robot travels as well as edges that define constraints between poses. The back end adjusts the poses of the nodes so that the constraints are satisfied as well as possible.

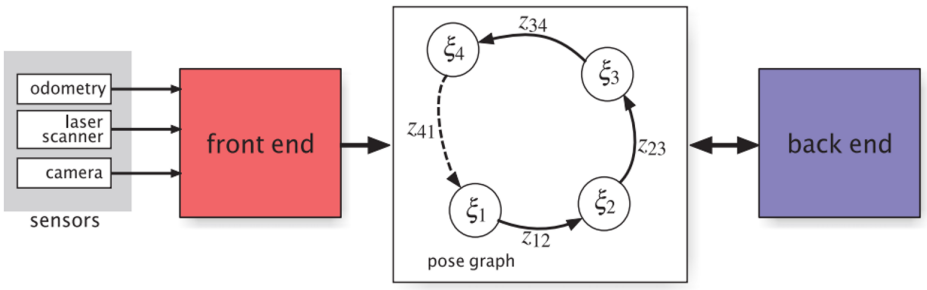


Figure 1.11: Pose-graph SLAM system. The front end creates nodes as the robot travels, and creates edges based on sensor data. The back end adjusts the node positions to minimize total error [1].

Nonetheless, efficient sparse least quadratic optimization techniques such as the famous Graph-based SLAM is becoming the state-of-art solution for the SLAM problem. The constraints between pose-to-pose and pose-to-feature are usually obtained by using odometry and Scan-Matching. The overall goal of these solutions is to find the configuration of state variables that minimize the sum of these constraints by taking the uncertainty of each constraint into consideration. The optimization processes are usually sensitive to the outliers of data association. Therefore, they rely heavily on the accuracy of the associated correspondences. And also, these methods are usually sensitive to the initial estimation of the state variables.

1.6 LiDAR-based SLAM

SLAM problems gained big attention from the scientific community in the last decades, being tackled in many areas. In fact, independently of the specific task, the great importance of precise navigation for autonomous robots will affect their success and outcome. Indeed, SLAM has been formulated and solved as a theoretical problem in many different forms. It has been implemented in several domains from indoor to outdoor [8].

The problem addressed in this thesis is formulated as an optimization problem on the data provided by a Light Detection And Ranging (LiDAR). It is then necessary to introduce the type of data we will use as well as the optimization method we intend to and reach our ultimate goal which is “The Place Recognition”.

First, we will introduce the description and applications of a LiDAR, the primary sensor considered in this work. Then, the representations of the data provided by the LiDAR sensor will be presented later in the second chapter.

1.6.1 LiDAR sensor

In robotics, navigation applications can be handled by different types of sensors. They all come with advantages and disadvantages. For instance, cameras are widely used because of the significant amount of information (for instance, color) that can be generated through images, moreover, it is a rather cheap technology.

However, a LiDAR is more expensive than the camera and it can only process spatial information. In addition, LiDAR sensors come with a range up to 100m with 2cm accuracy for the Velodyne sensors, whereas it is only limited to 5m for a Kinect. They may also provide a 360° field of view, which is convenient to locate a system in a structured environment. Thus, LiDAR sensors are not sensitive to ambient lighting. Moreover, a LiDAR is a sensor allowing to compute distances light as can be seen in fig. 1.12. A laser beam is emitted and its reflection on the target comes back to the transmitter. The reflected beam is analyzed and, thanks to light properties, it gives the target location.

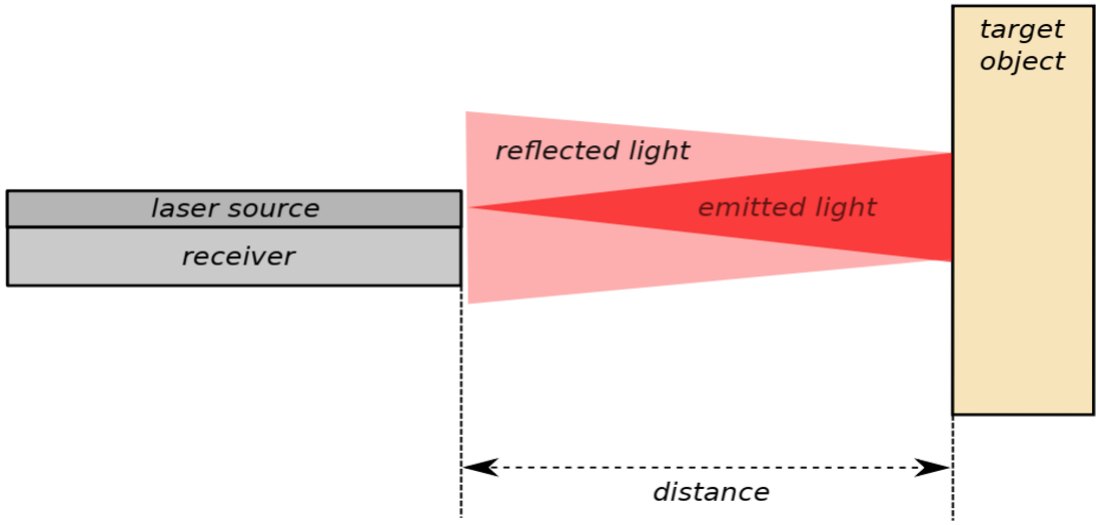


Figure 1.12: Principle of a LiDAR sensor [4].

For georeferencing purposes, a LiDAR can be combined with other types of sensors such as a Global Positioning System (GPS) or Inertial Measurement Unit (IMU). It can also be coupled with a camera to provide a synchronous acquisition of both sensors. Three types of LiDARs can be found, regarding the number of dimensions they exploit:

- **In 1D:** only one fixed beam is necessary, it measures the distance to the object in front of the LiDAR;
- **In 2D:** only one beam is required, it performs a rotation that delivers information on two axes;
- **In 3D**: the principle of rotation is the same as in 2D, but the sensor has multiple

beams. Each beam has its own angle. By rotating, the sensor will output a set of 3D points as depicted in fig. 1.13.

The first LiDAR sensors were used in aerospace applications in the early 1960s. The first application was an airborne topographic mapping of forests, oceans, and other terrains. The democratization of LiDAR sensor use in meteorology, atmospheric research, and shoreline monitoring was made possible by the development of effective GPS and satellite communications (fig. 1.14.c). A noticeable application is the model of "the Cathedral of Notre Dame" scanned in 2019, which is helping for its reconstruction (fig. 1.14.b).

In 2020, the vestige of a "Mayan city" was found below a dense jungle in Guatemala (fig. 1.15.a), thanks to LiDAR technology. Nowadays, LiDAR sensors are much more affordable, which makes the possibility to be used even more in research and industrial applications.

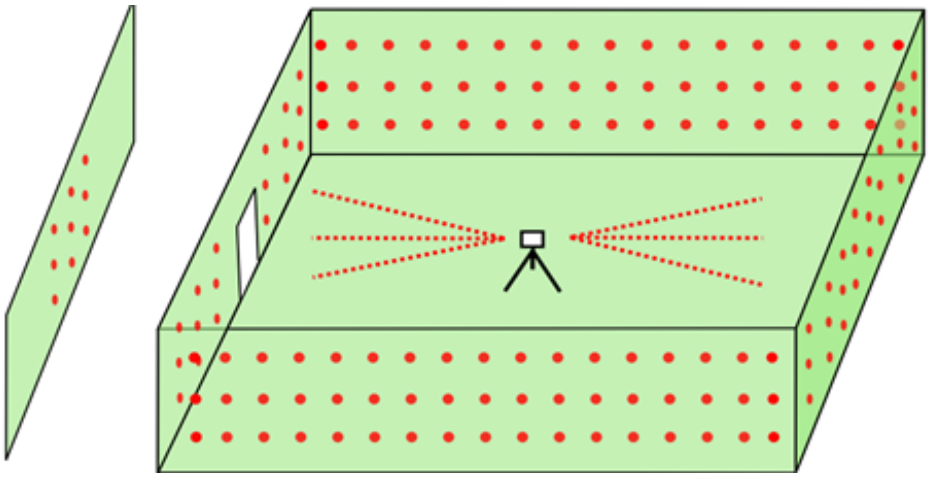


Figure 1.13: Depiction of a 3D LiDAR capturing a point cloud in a room. In this case, the LiDAR has three beams and rotates on itself. A 3D point cloud is generated (red points). A door is open and points are captured behind it [4].

The generated data is generally a set of 3D points, also called a 3D point cloud. It may also come with intensity information for each point is measured thanks to the reflectivity of the object hit by the laser. Thus, its strength varies with the composition of the surface of the object reflecting the signal. LiDAR sensors come with a few drawbacks:

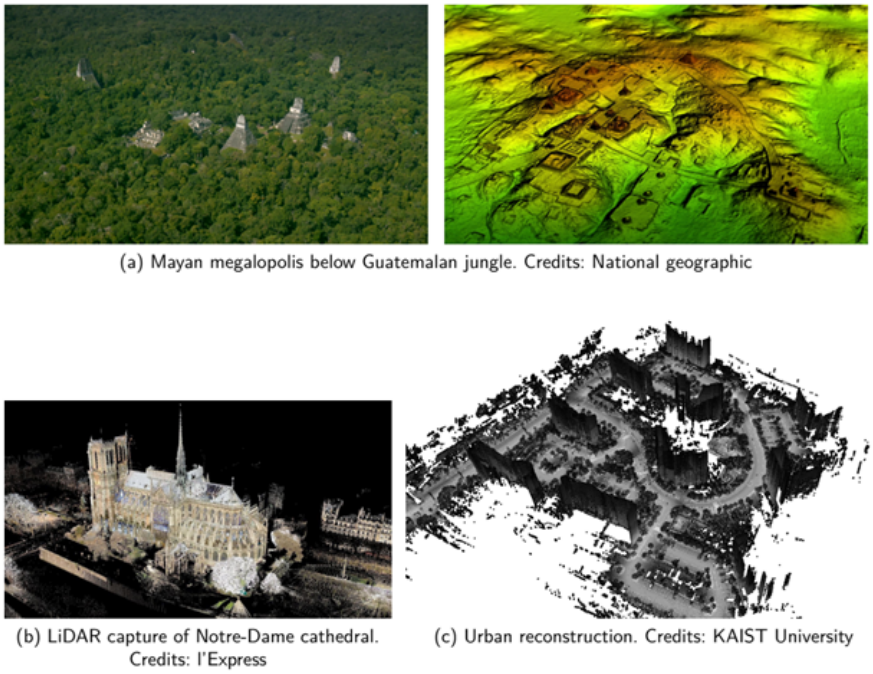


Figure 1.14: Applications of LiDAR in 3D modeling and self-driving vehicles [4].

- High operating costs in some applications: Although LiDAR is cheap when used in huge applications, it can be expensive when applied in smaller areas when collecting data;
- Ineffective during heavy rain or low hanging clouds: LiDAR pulses may be affected by heavy rains or low hanging clouds because of the effects of refraction. However, the data collected can still be used for analysis;
- The design of LiDAR sensors makes the density of points highly heterogeneous. The density of points is much higher close to the sensor, while it is scattered far from it, as in fig. 1.16;
- A precise 3D model implies a large number of input data which increases the processing time;
- Due to the functioning of a LiDAR sensor, since the laser is reflected by the first opaque surface it runs into, some part of the environment can be occluded. And, if it runs into a reflective surface a ghost effect may appear.

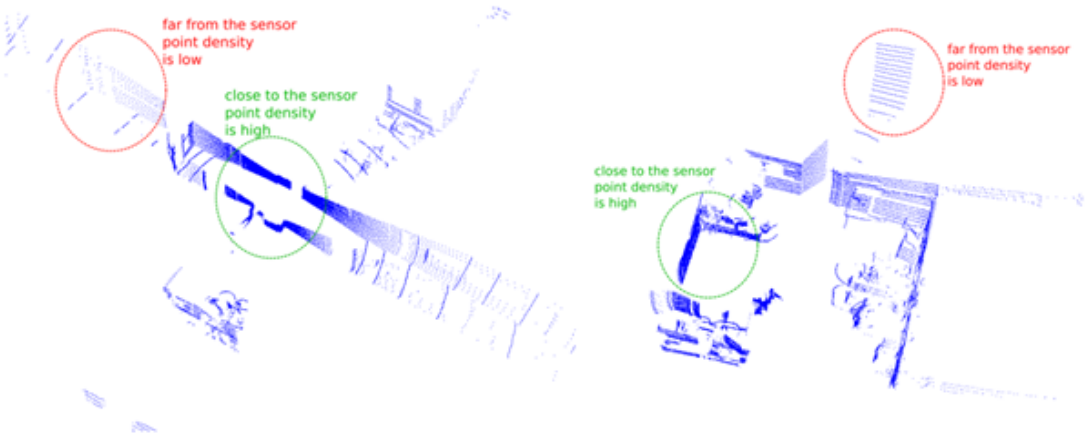


Figure 1.15: Depiction of the point density issue [4].

1.6.2 Scan-Matching

In recent years, surveys of techniques help to solve challenges in SLAM. Indeed, it has been implemented using several approaches, one of them is Scan-Matching algorithm. given a scan and a map, or a scan and a scan, or a map and a map, find the rigid-body transformation (translation+rotation) that aligns them best.

Scan-Matching is a concept frequently used in SLAM algorithms, and some solely rely on it. The concept is combining range measurements from one scan to the next. The result can be used to estimate the transformation between them; if the matching done with an existing map, new scans are implicitly matched with all preceding scans. There are numerous ways to perform Scan-Matching with Iterative Closest Point (ICP), and Normal Distribution Transform (NDT) seemingly being the most polar ones. Couple more details will be presented in the next chapter. ICP and NDT are investigated as follows [5]:

- ICP: is originally an approach for registering 3D point clouds, which works on Scan-Matching based localization started in 1992. Variations of the method are many, but the basic idea is to iteratively refine the relative pose of two overlapping scans by minimizing the sum of squared distances between the corresponding points in the two scans. By thresholding distances between matched scans and their change of orientation. The method seeks to eliminate false matches. Using Nearest Neighbor methods, ICP's search for point correspondences is, however,

expensive;

- NDT: models the distribution of all reconstructed 2D points of one scan by a collection of local normal distribution. In order to estimate a geometric transform relating to a new scan, a measure is defined by mapping all points according to the transform.

Scan-Matching objectives, even when not meaningful probabilities, can be used in GraphSLAM/pose-graph SLAM. Thus, Scan-Matching improved proposal distribution (e.g., GMapping detailed in chapter 3). Here we have Scan-Matching Overview:

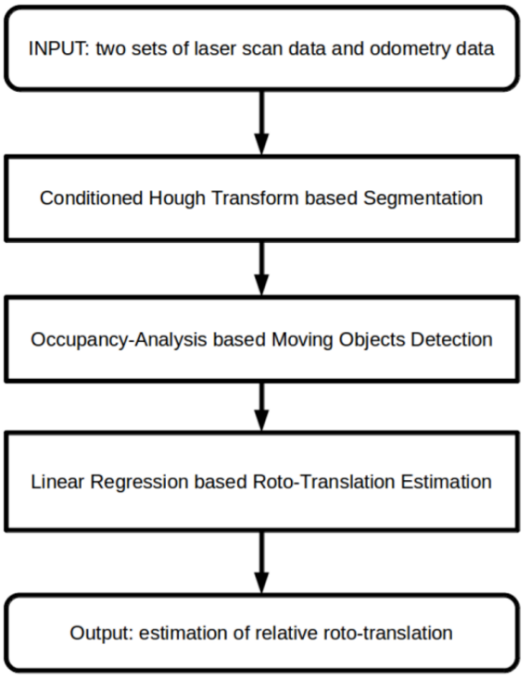


Figure 1.16: Overview of the phases in Scan-Matching algorithm [5].

1.7 Loop closing

An important issue in SLAM is loop closure detection. Indeed, loop closing is one of the main components of SLAM, in particular, the recognition provides candidates for loop-closure, this recognition of features or areas that the robot has observed before, and using this information to improve the state estimate which is essential for correcting drift error and building a globally consistent map. Loop closing in graph-based SLAM

is performed when the robot's pose estimate is within a given range of an earlier pose. The new pose estimate will create an edge between the earlier pose and itself.

Thus, loop closing happens between two nodes that are already in the graph, which means their positions and also their difference in position is stored. The loop closing calculates a new relative position and the difference with the previous position is called the error, which also has an x, a y, and a θ component. The error can be calculated by the error function. An error is used to improve the pose-graph estimate. For more details see chapter 2.

1.7.1 Drift Correction

Correct the drift in trajectory by accurately detecting the loops, which are places the robot has previously visited. Add the loop closure edges to the pose graph, which helps to correct the drift in trajectory during pose graph optimization. Thus, Loop closure detection determines whether the robot has previously visited the current location. The search is performed by matching the current scan against the previous scans around the current robot location, within the radius. A scan is accepted as a match if the match score is greater than the specified loop-closure Threshold.

1.8 Conclusion

Among other definitions, robotics could be stated as the branch of engineering that involves the design, manufacture, control, and robots programming. Robotics takes concepts from different subjects such as mechanical engineering, electrical engineering, electronic engineering, mathematics, physics, and computer science. Therefore, it can be seen as a combination of several areas of knowledge. Also, this chapter included description of the SLAM which is a process in which a mobile robot can build a map of its environment, and at the same time use this map to deduce its location. In SLAM both the trajectory of the platform and the location of all landmarks are estimated without the need for any a priori knowledge of the location. Furthermore, SLAM is an active research topic and fresh algorithms are continually being created to improve accuracy rates, therefore, new researchers must be able to understand this issue. From

what was presented above, the current chapter can be summarized as follows:

- SLAM is the process by which a mobile robot can build a map of an environment and at the same time use this map to compute its own location. Indeed, the past decade has seen rapid and exciting progress in solving this problem together with many compelling implementations of SLAM methods.

- This chapter describes possible sensors used, including Exteroceptive sensors (for many reasons we focused on LiDARs) and proprioceptive ones; and we have seen the main solutions needed to run SLAM-algorithms. The most popular approaches are usually developed based on probabilistic methods, such as Extended Kalman Filter (EKF), particle filter, and Graph-SLAM.

- In recent years, a robust technology named 'Scan-Matching' plays a very important role in solving the SLAM problem. Yet, Scan-Matching algorithms give erroneous position estimates, resulting in both pose drift and map inconsistencies.

- Several map representations are recognized as suited for SLAM purposes, and literature broadly divides them into metric and topological map representations.

- One another note, in hindsight LiDAR-based SLAM problem has been formulated, and implemented in a number of different domains; including indoor robots and outdoors.

- Metric maps capture the geometric properties of the environment and among are occupancy grid maps and landmark-based maps. Topological maps represent the environment as a list of significant places that are connected by arches graph, this simplifies the problem of mapping large extensions.

- EKs have numerous applications in technology. Therefore, KF is an algorithm using a series of measurements that contain statistical noise and other inaccuracies over time, and produces estimates of unknown variables by estimating and assuming gaussian-linear joint probability distribution over them for each time frame.

- EKF approach models the motion and observations as state-space models with additive gaussian noise, and uses high dimensional EKF to provide a posterior over a map and the robot pose.

- Finally, to conclude, in this chapter we have seen loop closing aspect in SLAM context, which allows us to maintain the drift correction.

Chapter 2

Loop Closure Detection

2.1 Introduction

Nowadays, Loop closure detection is becoming an essential and challenging problem in SLAM. Indeed, it is often tackled with Light Detection And Ranging (LiDAR) sensors due to its viewpoint and illumination invariant properties [15]; In a SLAM system, loop-closure is essential for correcting drift error and building a globally consistent map. This chapter describes the principle of LiDAR based loop closure detection and details an interesting loop closure method based on Scan-Context which is a novel spatial descriptor with a matching algorithm, specifically targeting outdoor place recognition using a single 3D scan [7].

This chapter is organized as follows. Section 2.2 briefly summarizes the loop closing problem some state-of-the-art loop closure detection techniques; while section 2.3 containes few ways to represent data, including point cloud and kD-tree. Last but not the least, section 2.4 introduces algorithms that where investigated such as the Scan-Matching based method and for various reasons we more focused on the Scan-Context algorithm. Then, we have conclude the chapter in section 2.5.

2.2 Loop closing: problem statement

Mapping and Localization are two essential processes in autonomous mobile robotics, since they are the basis of other higher-level and more complex tasks, such as obstacle

avoidance and path planning. Remember that mapping is the process through which a robot builds its own representation of the environment when its map is not available. In this context, there are mainly two types of maps "metric and topological" (more details in chapter 1).

While metric maps represent the world as accurately as possible, with regard to a global coordinate system; topological maps represent the environment in an abstract manner by means of a graph, which implies several benefits in front of the classic metric approaches.

Regardless of the type of the map to be built, any sensor used to perceive the environment (either a camera or LiDAR, radar) [15] causes an unavoidable noise, that should be taken into account in order to not compromise the accuracy of the resulting map and the localization process. For this reason, additional input is required to correct the inefficiencies produced in the map by this noise.

In SLAM problem, the estimated states and trajectories often come with inevitable drift. Additionally, in SLAM systems, place recognition is a critical challenge, since it gives candidates for the loop closure, i.e., knowing if a robot has returned to a previously visited location, which is required to correct the drift error and construct a globally consistent map of the environment (see fig. 2.1). Due to this reason, mapping algorithms usually rely on loop closure detection techniques, which entail the correct identification of previously seen places to reduce the uncertainty of the resulting maps.

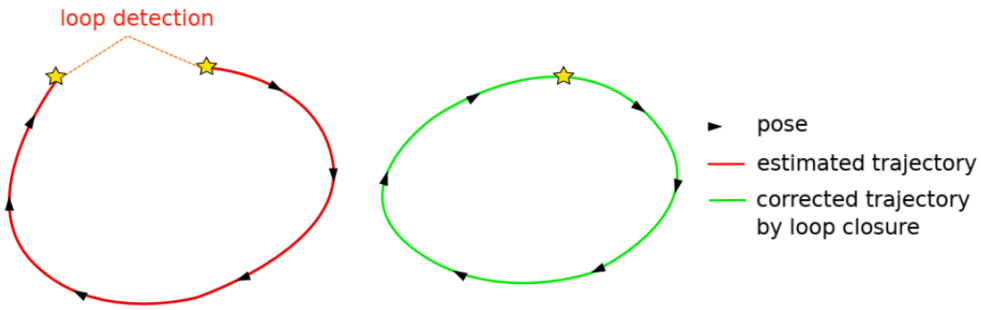


Figure 2.1: Loop closure detection example. The first estimated trajectory accumulates error and drifts, but a loop is detected. It allows to correct the estimation and generate more consistent trajectory [4].

However, loop closure detection is not an easy task and is even a more complex

problem than localization; it represents a gap in the literature due to the following reasons [2]:

- **Scalability:** The complexity of the problem increases as the map enlarges since more previous observations need to be compared with the current one to determine the existence of a loop;
- **Perceptual aliasing:** Different places of the environment are perceived as the same, resulting in false loop closure detections;
- **Sensor noise:** As we stated above, measurements include noise, which makes more difficult the data association;
- **Changes in the environment:** Dynamic obstacles; moving objects produce different perceptions of the same place.

In the literature, there are several loop closure strategies. According to the perception system, existing works on loop closure detection can be categorized as 'vision-based methods' and 'LiDAR-based methods'. Because of the wide use of camera sensor and motivated by its low cost, the richness of the sensor data provided, visual recognition has become extremely popular, despite the fact that it is fundamentally difficult due to lighting variations and short-term (e.g., moving objects, dynamic obstacles) or long-term (e.g., seasons) changes.

In general, given that this problem involves collections of images, any technical solution presented should be able to efficiently manage a larg volume of visual data, which typically expands in tandem with the map. This implies carrying out the loop closure detection using images as the main source of information. For better understanding, we can see fig.2.3 which shows an application example of detecting instances of semantic objects of a certain class using OpenCV library (such as humans, buildings, or cars) in digital images and videos.

However, when compared to visual loop closure detection techniques, LiDAR-based loop closure detection has the advantages of being more resistant to changes in illumination and having higher location accuracy [16].

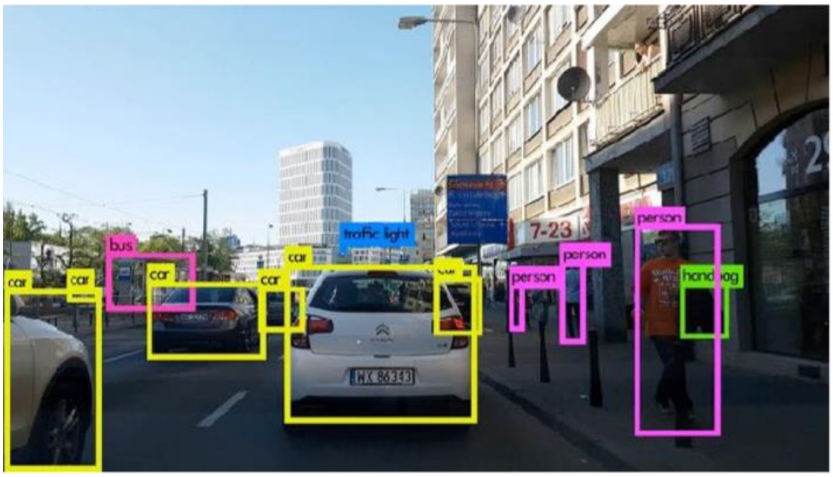


Figure 2.2: Object detection and recognition.

In recent years, few methods have been mainly proposed to apply LiDAR as the only sensor to the localization system such as LiDAR-SLAM. LiDARs, in contrast to visual sensors, have increasingly gained attention due to their high perceptual invariance. Despite their noise vulnerability, traditional local keypoint descriptors, which were originally designed for the 3D model in computer vision, were used for the place recognition early [16, 2]. These studies try to generate descriptors from structural data (e.g., point clouds) on local and global manners.

Existing LiDAR-based place recognition algorithms have been looking to solve two fundamental concerns. First, the descriptor must achieve rotational invariance regardless of viewpoint changes. The Second one, noise handling is another topic for these spatial descriptors.

On the other hand, LiDAR presents strong robustness to perceptual changes described above. In fact; LiDAR-based methods are further categorized into local and global descriptors. Namely, Muhammad and Lacroix proposed Z-projection [17], which is a histogram of normal vectors, and a double threshold scheme with two distance functions. He et al. proposed M2DP [18], which projects a whole 3D point cloud of a scan to multiple 2D planes and extracts a 192-dimensional compact global representation. M2DP showed higher performance than the existing point cloud descriptors, and robustness against noise and resolution changes.

Global descriptors have typically used histograms. SegMatch [19] introduced a

segment-based matching algorithm. This is a high-level perception but requires a training step, and points are needed to be represented in a global reference frame.

Local descriptors like PFH [20], SHOT [21], shape context [22], and spin image [23] search for a key point, then divide nearby points into bins and encode a pattern of bins into a histogram. In [24], Steder et al. presented a bag-of-words method for recognizing places utilizing point characteristics and the gestalt descriptor [25]. These keypoint descriptors, on the other hand, revealed weaknesses because they were developed for a 3D model part matching instead of place recognition. In opposition to the 3D model, the density of a point cloud in a 3D scan (e.g., from VLP-16 [7]) varies with distance from a sensor. Furthermore, due to unstructured objects (e.g., trees) in the real world, point normals are noisier than the model. In this chapter, we are interested in the Scan-Context descriptor for loop closure detection which will be described in the section II.4.

2.3 Data representation

The data collected by a LiDAR sensor would be in the form of a point cloud, so accordingly. We need to go over how such data might be represented and arranged in more details.

2.3.1 Point cloud

Point cloud is a representation of the data provided by a LiDAR sensor. This representation refers to a collection of points in space. It can be used to represent either an object or an environment. Each point in a 3D point cloud is represented by its coordinates X, Y, and Z. At each point in a point cloud, extra data such as light intensity, color, or normal can be found [4]. In contrast to a mesh, where connection relationships are provided by the edges, points recorded in the point cloud have no spatial relationship with each other.

However, the contributions presented in this manuscript sometimes need to use a "nearest neighbor search" to match points from one point cloud to another. So accordingly, the nearest neighbor search problem consists in finding the point that is closest to a given point in a set of data. To tackle this problem, and to handle point

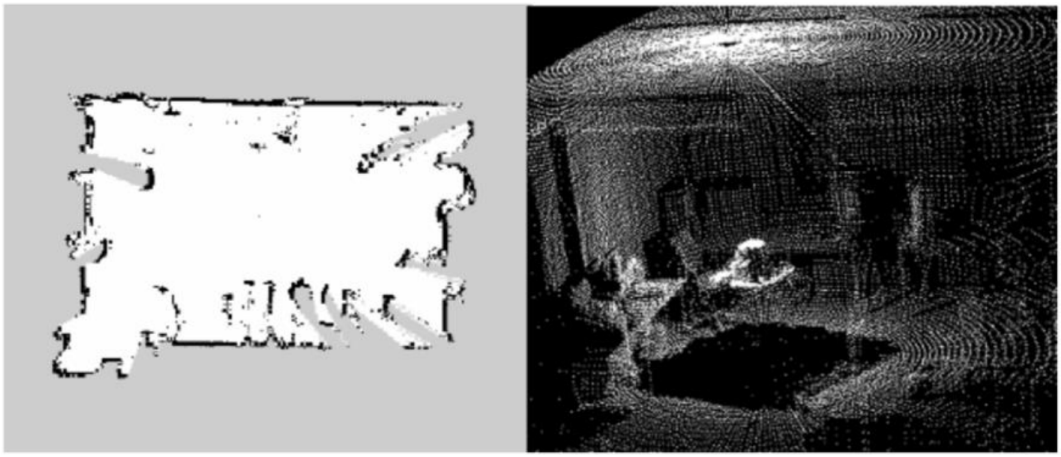


Figure 2.3: On the left is visualization of the occupancy grid 2D. On the right is visualized point cloud 3D of a room [6].

clouds more wisely, data structures were created such as "kD-trees" to reduce search complexity (see fig. 2.4 and fig. 2.5).

2.3.2 kD-tree structure

A kD-tree is a space-partitioning data structure; used to accelerate the nearest neighbor search. The building phase of a kD-tree is of complexity $O(mlogm)$ (the second cloud) with m the number of points in point cloud ${}^2P$, and the search complexity is $O(nlogm)$, with n the number of points in ${}^1P$ (the first point cloud) [4].

In addition, kD-tree is a special case of a Binary Space Partitioning (BSP) tree. A BSP tree splits k-dimensional space volume into two sub-volumes by a hyperplane of the space and then iterates on the two resulting volumes. Furthermore, the two sub-volumes are the children of the node that represents the whole volume. The hyperplanes are chosen such as their normal is one of the axes of the coordinate system.

In the literature, there are different ways to build and handle a kD-tree. In the classical method, with a 3D example, the normal of the plane splitting the root is aligned with the x-axis. Its children have their plane normal aligned with the y-axis, and its grand-children axes are aligned with the z-axis. The following splitting plane normal is aligned with the x-axis and so on. In order to have a balanced tree, the chosen point to perform the split is located at the median coordinate of the considered

direction. An example of a kD-tree is given in fig. 2.4.

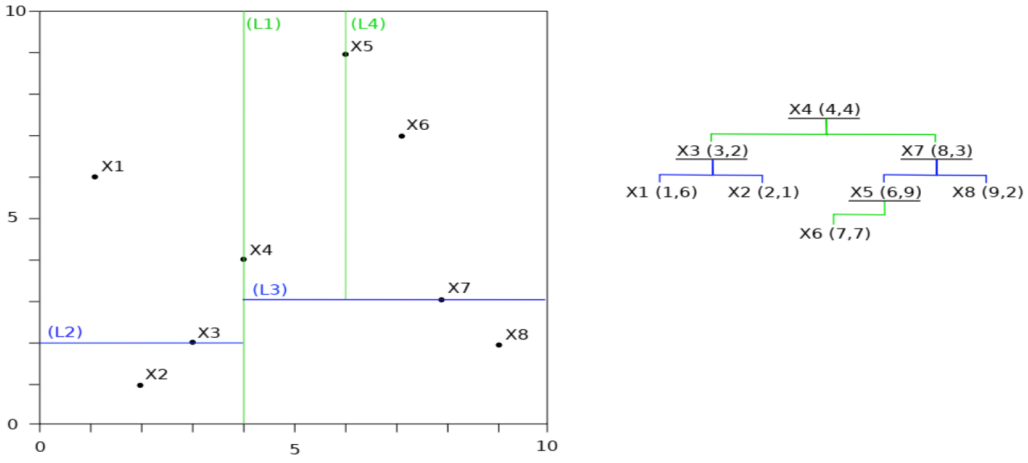


Figure 2.4: Left: a 2D point set with the kD-tree splits. Right: The corresponding tree. (Note: a 2D example is given to ease reading) [4].

2.4 Loop closure detection algorithms

2.4.1 Scan-Matching based algorithm

The loop closure algorithm determines whether the robot's current location has already been visited or not. The search is carried out by doing a Scan-Matching between the current scan with previous scans within a small radius around the robot's current location. Searching within a small radius is sufficient due to the low drift of LIDAR odometry, as searching against all previous scans takes time. The loop closure algorithm includes the following steps:

- Create submaps from consecutive scans.
- Match the current scan with the submaps within a certain radius around the robot's current position.
- Accept matches if the match score is above a given threshold. All scans representing the accepted submap are considered as loop closure candidates.

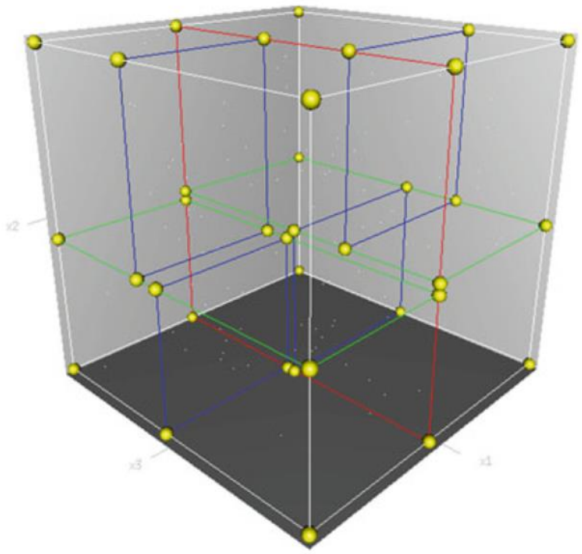


Figure 2.5: Example of a 3-dimensional tree. The first split (red) cuts the root cell (white) into two subcells, each of which is then split (green) into two subcells. Finally, each of those four is split (blue) into two subcells [2].

- Estimate the relative pose between the loop candidates and the current scan. A relative pose is accepted as a loop closure constraint only if the RMSE is below a threshold.

2.4.2 Scan-Context based algorithm

Nowadays, advanced research in SLAM provides dense 3D maps of the environment, and the localization is proposed by diverse sensors. Indeed, describing a place using structural information is relatively less reported. Toward the global localization based on the structural information, here we are going to present Scan-Context [7], it's a non-histogram based on global descriptors from 3D (LiDAR) scans. In contrast to previously reported methods, this proposed approach directly records a 3D structure of a visible space from a sensor, and does not rely on a histogram or on prior training. Moreover, this approach uses a similarity score to calculate the distance between two Scan-Contexts beside a two-phase search algorithm to efficiently detect a loop. Scan-Context and its search algorithm make loop detection invariant to LiDAR viewpoint changes so that loops can be detected in places such as reverse revisit and corner.

The overall pipeline of place recognition using Scan-Context is depicted in fig. 2.6. First, a point cloud in a single 3D scan is encoded into Scan-Context. Next, a Nearest Neighbor Search (NNS) is conducted to retrieve the nearest candidates. Finally, the retrieved candidates are compared to the query scan context. The candidate that satisfies the acceptance threshold and is closest to the query is considered the loop. This will be fully described in the following sub-sections.

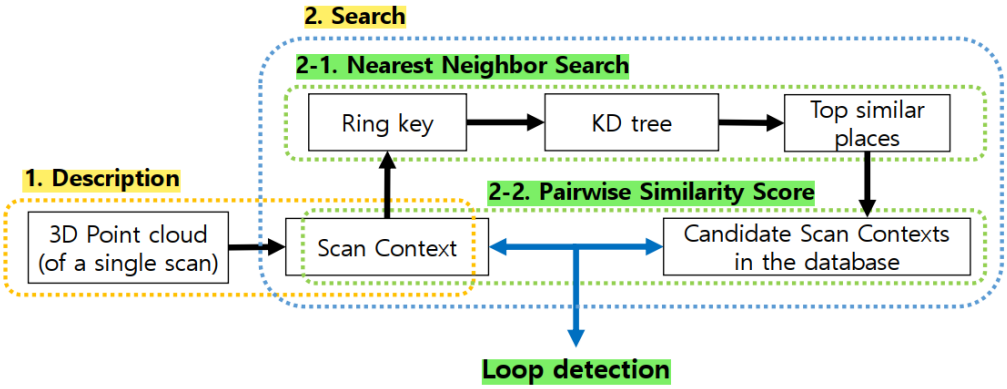


Figure 2.6: Scan-Context algorithm overview [7].

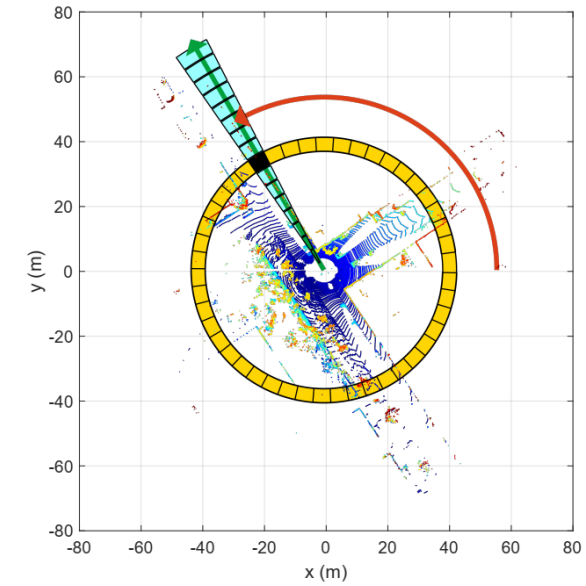
Scan-Context Descriptor creation

This technique encodes a whole point cloud in a 3D scan into a matrix (see fig. 2.7).

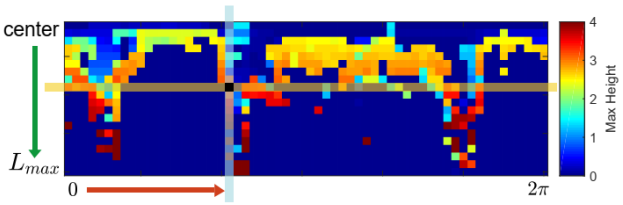
This process of creating a Scan-Context is divided into two parts; Using the top view of a point cloud from a 3D scan (check fig. 2.7.(a)). We partition ground areas into bins, which are split according to both azimuthal (from 0 to 2π within a LiDAR frame) and radial (from center to maximum sensing range) directions; we state that we refer to:

- the yellow area as a ring,
- The cyan area as a sector,
- The black-filled area as a bin.

Scan context is a matrix as in (fig. 2.7.(b)) that explicitly preserves the absolute geometrical structure of a point cloud. The ring and sector are described in (fig. 2.7.(b)) are represented by the same-colored column and row, respectively, in (fig. 2.7.(b)).



(a) Bin division along azimuthal and radial directions



(b) Scan context

Figure 2.7: Scan-Context creation [7].

The representative value extracted from the points located in each bin is used as the corresponding pixel value of (fig.6 2.7.(b)). The algorithm uses the maximum height of points in a bin. As a result [7], the density and normals of a point cloud are invariant with this bin encoding. Also, as we can see from (fig. 2.7.(b)), this algorithm preserves the internal structure of a point cloud.

As we said before, the descriptor is computed by first binning points from a 3D point cloud scan into concentric radial and azimuthal bins, and then selecting the points with the highest Z-height in each bin. In addition, the algorithm encodes the geometrical shape of the point cloud around a local keypoint into an image.

The center of a scan acts as a global keypoint (fig. 2.7) and thus we refer to a scan context as an egocentric place descriptor; while:

- N_s and N_r are the number of sectors and rings, respectively, in the next chapter we used the values $N_s = 60$ and $N_r = 20$;

- L_{max}, is the maximum sensing range of a LiDAR sensor;
- $\frac{L_{max}}{N_r}$ is the radial gap between rings;
- $\frac{2\pi}{N_s}$ is the central angle of a sector,
- $\mathcal{P}$ is the point cloud.

Let $\mathcal{P}_{ij}$ is the set of points belonging to the bin where the ith ring and jth sector overlapped. Thus, the partition is mathematically:

$$\mathcal{P} = \bigcup_{i \in [N_r], j \in [N_s]} \mathcal{P}_{ij} \tag{2.1}$$

Where the symbol $[N_s]$ is equal to $\{1, 2, \ldots, N_{s-1}, N_s\}$.

The point cloud is divided at regular intervals; so, a bin far from a sensor has a physically wider area than a near bin.

After the point cloud partitioning, a single real value is assigned to each bin by using the point cloud in that bin:

$$\phi : \mathcal{P}_{ij} \to \mathbb{R} \tag{2.2}$$

Thus, the bin encoding function is:

$$\phi\left(\mathcal{P}_{ij}\right) = \max_{\mathbf{p} \in \mathcal{P}_{ij}} z(\mathbf{p}) \tag{2.3}$$

Mention that $z()$ is the function that returns a z-coordinate value of a point p.

Note: We assign a zero for empty bins. For example, as seen in (fig. 2.7), a blue pixel in the scan context means that the space corresponding to its bin is either free or not observed due to occlusions. A scan context I is finally represented as an $N_r \times N_s$ matrix as

$$I = (a_{ij}) \in \mathbb{R}^{N_r \times N_s}, a_{ij} = \phi\left(\mathcal{P}_{ij}\right) \tag{2.4}$$

Similarity Score between Scan Contexts

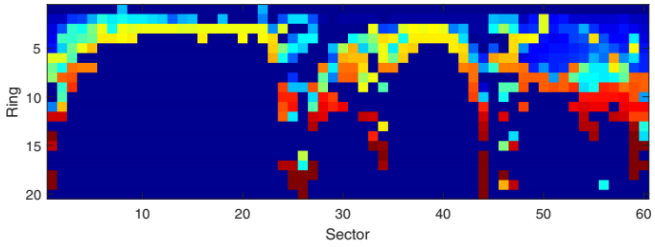
We need a distance measure for the similarity of two places while a scan context pair is given. Scan contexts I^q and I^c are acquired from a query and candidate point clouds,

respectively. They are compared in a column-wise manner. The distance is the sum of distances between columns at the same index. A cosine distance is used to compute a distance between two column vectors at the same index, c_j^q and c_j^c. Therefore, the distance function is:

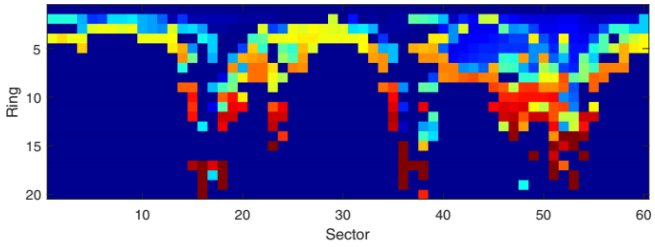
$$d\left(I^q, I^c\right) = \frac{1}{N_s} \sum_{j=1}^{N_s} \left(1 - \frac{c_j^q \cdot c_j^c}{\left\|c_j^q\right\| \left\|c_j^c\right\|}\right) \tag{2.5}$$

The column-wise comparison is particularly effective for dynamic objects by considering the consensus throughout sectors. However, the column of the candidate scan context may be shifted even in the same place, since a viewpoint of a LiDAR changes for different places (e.g., revisit in an opposite direction or corner).

Indeed; fig. 2.1 illustrates such cases. Since a scan context is a representation dependent on the sensor location, the row order is always consistent. However, the column order could be different if the LiDAR sensor coordinate with respect to the global coordinate changed. In (fig. 2.6) we note that the change of the sensor viewpoint at the revisit causes column shifts of the scan context as in (a). The two matrices contain similar shapes and show the same row order.



(a) The query scan context (3280$^{\text{th}}$ scan, KITTI00)



(b) The detected scan context (2345$^{\text{th}}$ scan, KITTI00)

Figure 2.8: Example of scan contexts from the same place with time interval [7].

Since a scan context is the representation dependent on the sensor location, the

row order is always consistent.

To overcome this problem, the authors [7] calculate distances using all possible column-shifted scan contexts and find the shortest distance. I_n^c is a scan context whose n columns are shifted from the original one, I^c. Then we decide on the number of column shifts () and the distance (equation) for the optimum alignment at that time:

$$D\left(I^q, I^c\right) = \min_{n\in[N_s]} d\left(I^q, I_n^c\right) \tag{2.6}$$

$$n^* = \underset{n\in[N_s]}{\operatorname{argmin}} d\left(I^q, I_n^c\right) \tag{2.7}$$

Two-phase Search Algorithm

Since the distance calculation in (2.7) is heavier than other global descriptors, the authors [26] provide a two-phase hierarchical search algorithm (see 2.6) via introducing ring key.

- Pairwise similarity scoring,
- Nearest neighbor search,
- Sparse optimization.

This search algorithm fuses both pairwise scoring and nearest search hierarchically to achieve a reasonable searching time.

Since the distance calculation in equation 2.7 is heavier than other global descriptors, we provide a two-phase hierarchical search algorithm by introducing a ring key.

Ring key is a rotation invariant descriptor, which is extracted from a scan context. Each row of a scan context, r, is encoded into a single real value via ring encoding function ψ . The first element of the vector k is from the nearest circle from a sensor, and the following elements are from the next rings in order as illustrated in fig. 2.9. Furthermore, the ring key becomes an N_r dimensional vector as as 2.8:

$$k = \left(\psi\left(r_1\right), .., \psi\left(r_{N_r}\right)\right), \ where\ \psi : r_i \to \mathbb{R}. \tag{2.8}$$

The used ring encoding function ψ is the occupancy ratio of a ring L_0 norm:

$$\psi\left(r_i\right) = \frac{\|r_i\|_0}{N_s}. \tag{2.9}$$

The ring key achieves rotation invariance. The ring key is used to enable fast search for finding possible candidates for loop. The vector k is used as a key to construct a kD-tree. At the same time, the ring key of the query is used to find similar keys and their corresponding scan indexes.

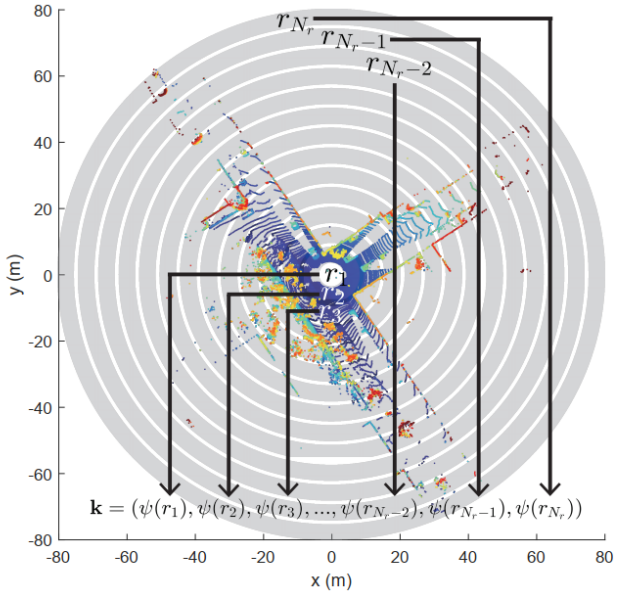


Figure 2.9: The ring key generation for the fast search [7].

The number of top similar keys retrieved is determined by a user. These constant number of candidates scan contexts are compared against the query scan context by using distance equation 2.7. The closest candidate to the query satisfying an acceptance threshold is selected as the revisited place:

$$c^* = arg\min D\left(I^q, I^{C_k}\right), D < \tau, \tag{2.10}$$

where:

- C is a set of indexes of candidates extracted from kD-tree;
- is a given acceptance threshold;
- c^* is the index of the place determined to be a loop.

2.5 Conclusion

While a brief introduction and SLAM basis was given in the previous chapter, this current chapter provides the background and motivation for loop closure detection techniques, on which this work is based. We also swiftly introduce the place recognition issues, since it's still an open problem in many mobile robotics applications. Thus, this recognition is critical for the SLAM because it gives candidates for loop-closure, which is necessary for correcting drift error and creating a globally consistent map. So according to this, we explore some algorithms within SLAM problem at a deeper level.

Scan context algorithm could be a better choice for place recognition scenarios. Indeed, it is presented as a spatial descriptor, while it is summarizing a place as a matrix that explicitly describes the 2.5D structural information of an egocentric environment. Also, scan context algorithm would be beneficial for a SLAM solution because it achieves a reasonable searching time. The performance of this algorithm will be evaluated in the next chapter.

Chapter 3

Experimental part and ROS implementation

3.1 Introduction

The field of robotics has improved dramatically over time, and several pieces of research have been conducted to improve the efficiency of robots. Indeed, one of the most important features of mobile robots is navigation. Therefore, a map of the environment is required for a robot to navigate properly in that specific area. In addition, SLAM is a robust mapping technique that may be applied to a variety of tasks. In this chapter, we evaluate some SLAM algorithms supplied by the Robot Operating System (ROS), which is an open-source framework, with other 3D simulators (RViz and Gazebo). We will try to analyze the 2D and 3D maps accuracy obtained by some SLAM algorithms to help the robot model to achieve its numerous destinations in an unfamiliar indoor/outdoor environment. We had to mention that the distinctive element of this thesis is the analysis of the 2D map quality scanned by the Waffle robot and furthermore to navigate autonomously with vehicles in very challenging data sets that can be freely found here[1] and here[2].

In indoor spaces, the Waffle robot must navigate to various destinations. through the Gazebo world scenarios, which contain many obstacles. The ultimate goal of our

[1] https://www.mrt.kit.edu/z/publ/download/velodyneslam/index.html

[2] https://drive.google.com/drive/folders/1gJHwfdHCRdjP7vuT556pv8atqrCJPbUq

project is to implement several autonomous navigation algorithms for Obstacle Avoidance that allow a robot to move and operate in an unknown environment through GMapping SLAM and navigation stack that were executed on a computer equipped with an Intel i5-7400U CPU using ROS Noetic in Ubuntu 20.04 focal also with pc laptop AMD Ryzen 5 4500U with 8,00 GB of RAM.

Sections, 3.2, 3.3, and 3.4 of this chapter introduces the robot used for the simulation and its operating system (ROS), also the environments where we did our simulation. GMapping, LIO-SAM, and LiDAR based-GraphSLAM all are SLAM algorithms that is presented in section 3.5. Then, we presented and discussed our results in section 3.6. Section 3.7 is the conclusion of this chapter.

3.2 The Used robots

3.2.1 TurtleBot3

Nowadays, TurtleBot is becoming a robot that operates on ROS. It is the most popular platform among developers and students. Turtle is derived from the Turtle robot, which was created in 1967. It has since become the standard platform for ROS[3]. There are three different categories of TurtleBot model that come in different variations:

a. TurtleBot 1 or turtlesim is a ROS-based research robot. It was created in 2010.

b. TurtleBot 2 was created in 2012, it is based on the research robot iClebo Kobuki.

c. TurtleBot 3 was released in 2017 with new capabilities to compensate for the shortcomings of its predecessors as well as consumer needs. The TurtleBot 3 is powered by the ROBOTIS DYNAMIXEL smart actuator.

In addition, turtleBot 3 is a ROS-based mobile robot that is simple, affordable, and programmable for use in education, research, hobby, and product development. TurtleBot3's objective is to reduce the platform's size and lower its price without sacrificing its functionality or quality. The TurtleBot3 can be customized in a variety of ways. Also, it has a tiny and cost-effective that is appropriate for 360-degree distance sensors,

[3] https://emanual.robotis.com/docs/en/platform/turtlebot3/overview/

and 3D printing technologies. The three variants of the TurtleBot3 are illustrated in fig 3.1.

SLAM, Navigation, and Manipulation are among TurtleBot3's primary technologies, making it suited for home service robots. Also, it can construct a map using SLAM techniques, and drive around a room.

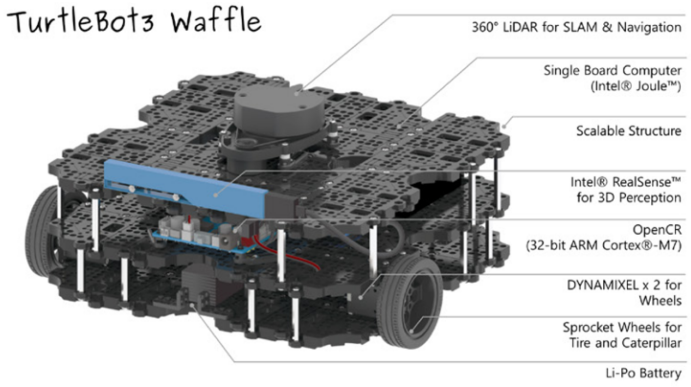


Figure 3.1: Shape and Dimension of TurtleBot3 Waffle.

3.2.2 ROS Ecosystem

ROS is an open-source framework, meta-operating system for robots. It provides the services we would expect from an operating system, including hardware abstraction, low-level device control, implementation of commonly-used functionality, message-passing between processes, and package management.

It also provides tools and libraries for obtaining, building, writing, and running code across multiple computers [27]. Additionally, ROS provides standard features or operating system services such as device drivers, implementation of commonly used features including sensing, recognizing, mapping, motion planning, message-passing between processes, package management, visualizers, and libraries for development as well as debugging tools.

In another word, it is a collection of free and open-source software packages used by a large and growing developer community to build, simulate, and test robotic systems.

Robot operating system

A considerable part of our experimental results consisted of getting acquainted with ROS. Hence, a short introduction to the software framework is given in this current section. As is most of the open-source software utilized and existing software on board. All the codes used during this thesis are written in programming language C++ and python. ROS Noetic Ninjemys, we used in our work, is the thirteenth ROS distribution release. It was released on May 23rd, 2020. It is primarily targeted at the Ubuntu 20.04 (Focal) release.

- **Node:** A node refers to the smallest unit of the processor running in ROS. Think of it as one executable program. ROS recommends creating one single node for each purpose, and it is recommended to develop for easy reusability. A specialized node is used for each function such as sensor drive, sensor data conversion, obstacle recognition, motor drive, encoder input, and navigation.

- **ROS master:** For node-to-node connections and message exchange, the master serves as a name server. When you run the master with the command roscore, you can register the names of each node and obtain information when you need it. Without the master, it is difficult to connect nodes and communicate messages such as topics and services. If multiple computers are within the same network, it can be run from another computer in the network [28]. However, except for a special case that supports multiple roscore, only one roscore should be running in the network.

- **tf-package:** Used to share the robot positions, coordinate frames, and transformations.

- **Rosbag file:** Some data present during sailing in different indoor environments (see section III.5) was recorded in ROS; they have been measured and collected to be treated latter in Matlab 2021b as rosbags files. The rosbag can be played, and simulate the TurtleBot3 publishing the saved messages.

The information contained in the ROS messages can be saved through a file format called a bag, and the file extension is ". bag.". Hence; Bag may be used in ROS

to record messages and play them back when needed to simulate the environment in which they were recorded. Sensor values are recorded in the message using the bag while executing a robot experiment using a sensor, for example. By playing the stored bag file, this recorded message may be loaded several times. When constructing an algorithm with frequent program revisions, rosbag's record and play functionalities [28].

3.3 ROS Tools and Simulators

ROS includes a number of tools that might be beneficial including those developed by ROS users. The tools that we will describe, represent the ones that have been most used in our work:

- **RViz:** is a 3D visualizer for displaying sensor data and state information from ROS. This tool is used both for visualizing the map and trajectory resulting from the SLAM algorithms including navigation stack, as well as inspection of the LiDAR data.

- **Rqt:** that is a software framework of ROS that implements the various GUI tools in the form of plugins,

- **Rqt graph:** a tool that visualizes the correlation between nodes and messages as a graph,

- **Gazebo Simulator:** Gazebo is a 3D simulator that supports realistic simulation with its physics engine and offers robots, sensors, and ecosystem models for 3D simulation essential for robot development. The Gazebo has become one of the most popular open-source robotics simulators in recent years, and it is frequently used in the robotics market because of its high performance and reliability. In ROS, there are several techniques to simulate a robot. The most advanced employs Gazebo that predates ROS but now works seamlessly with it.

3.4 Presentation of the used Gazebo environments

In our simulations, we used various environments during the ROS implementation to examine the compliance and the performance of the studied algorithms. They are as follows:

A. The TurtleBot3 world environment:

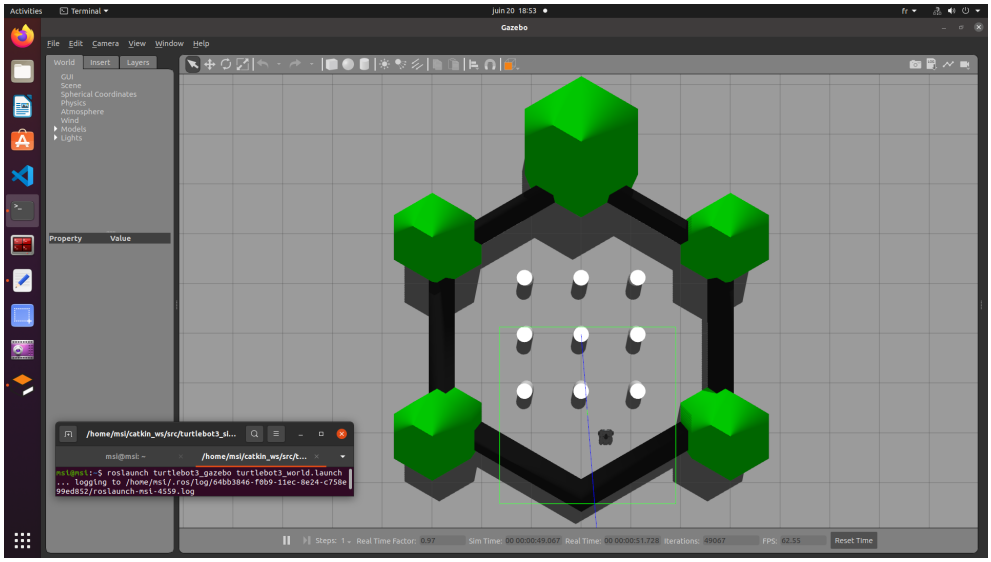


Figure 3.2: Launching tb3 in world envirnment using Gazebo.

B. One room with one obstacle:

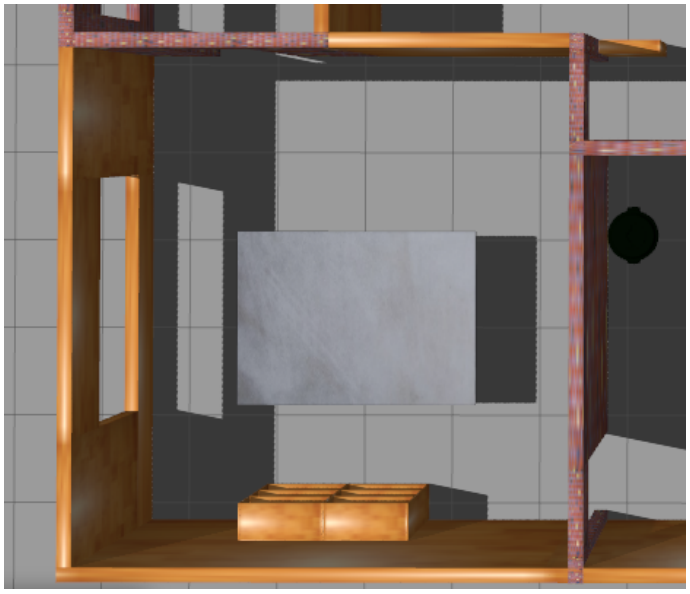


Figure 3.3: One room in Gazebo house, where we recorded rosbag files.

Gazebo House environment:

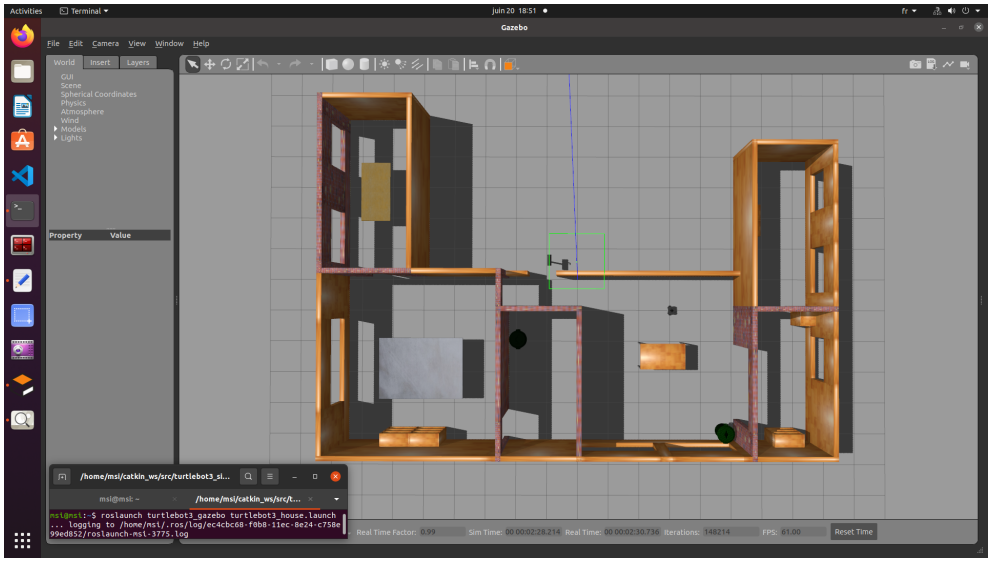


Figure 3.4: Launching tb3 in House envirnment using Gazebo.

Quadrotor in outdoor environment:

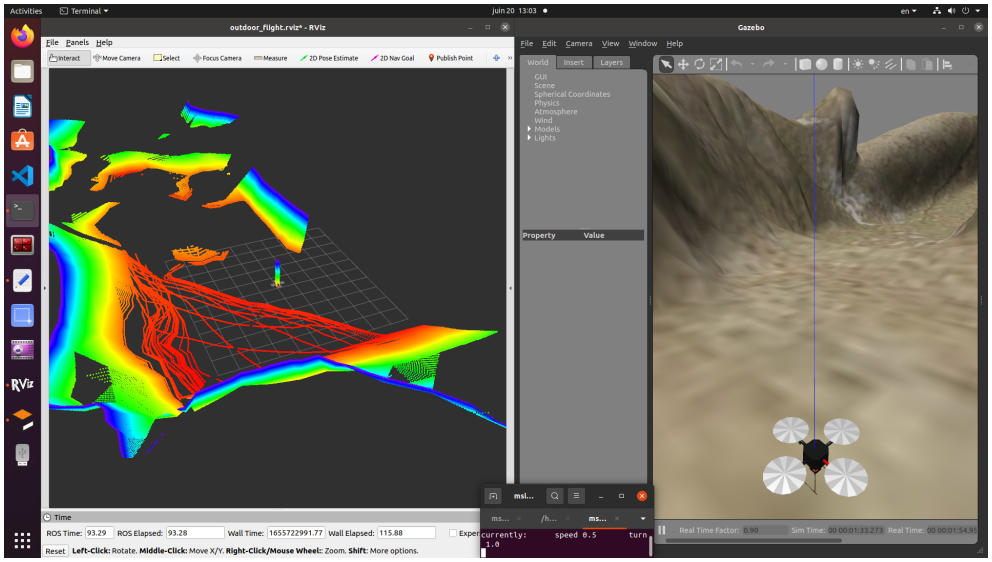


Figure 3.5: Launching hector-quadrotor in outdoor envirenment.

3.5 SLAM algorithms presentation

In this chapter, we used different algorithms for implementing SLAM such as Graph-SLAM algorithm coded in Matlab, GMapping which is provided by ROS and LIO-

SAM [29]. The remainder of this section, gives a brief introduction to these algorithms.

3.5.1 GMapping package

The GMapping package provides laser-based SLAM. It is a SLAM technique that depends on particle filters to create a 2D map. The filter used is called Rao Blackwellized Particle Filter (RBPF). GMapping may produce a 2D occupancy grid map like a building floorplan, from laser and posture data gathered by a mobile robot. To use this package, we need a mobile robot that provides odometry data and is equipped with a horizontally-mounted laser range-finder [27]. Note that the Adaptive Monte Carlo Localization (AMCL) node was employed to estimate the position.

3.5.2 LiDAR-based GraphSLAM

We use this approach to build a map that consists of:

a. **Alignment of LIDAR scans** that aligns successive scans using a point cloud registration technique (NDT for example) which estimates the relative pose between successive scans and build a Pose Graph. By successively composing the obtained relative transformations, each point cloud is transformed back to the reference frame of the first point cloud.

b. **Combine aligned scans** that generate the map by combining all the transformed point clouds.

c. **Correct the drift** in estimated robot trajectory by identifying previously visited places (loop closures). We use the Scan-Context based loop closing method or the Scan-Matching based method for comparison. The GraphSLAM algorithm adds the loop closure edges to the pose graph.

d. **Pose graph optimization** using the loop closure edges to correct the drift in trajectory.

3.5.3 LIO-SAM (Tightly-coupled LiDAR Inertial Odometry via Smoothing and Mapping)

LIO-SAM is a framework for tightly-coupled LiDAR Inertial Odometry via Smoothing And Mapping that performs real-time state estimation and mapping in complex environments[4]. By formulating LiDAR-Inertial odometry atop a factor graph, LIO-SAM is especially suitable for multi-sensor fusion. Additional sensor measurements can easily be incorporated into the framework as new factors. Sensors that provide absolute measurements, such as a GPS, compass, or altimeter, can be used to eliminate the drift of LiDAR inertial odometry that accumulates over long durations, or in feature-poor environments. Place recognition can also be easily incorporated into the system. The Scan-Matching at a local scale rather than a global scale facilitates the real-time performance of the LIO-SAM framework. This method is thoroughly evaluated on datasets gathered on platforms across a variety of environments. Figure III.6 ou 7 shows a diagram of Lio-SAM System architecture [29].

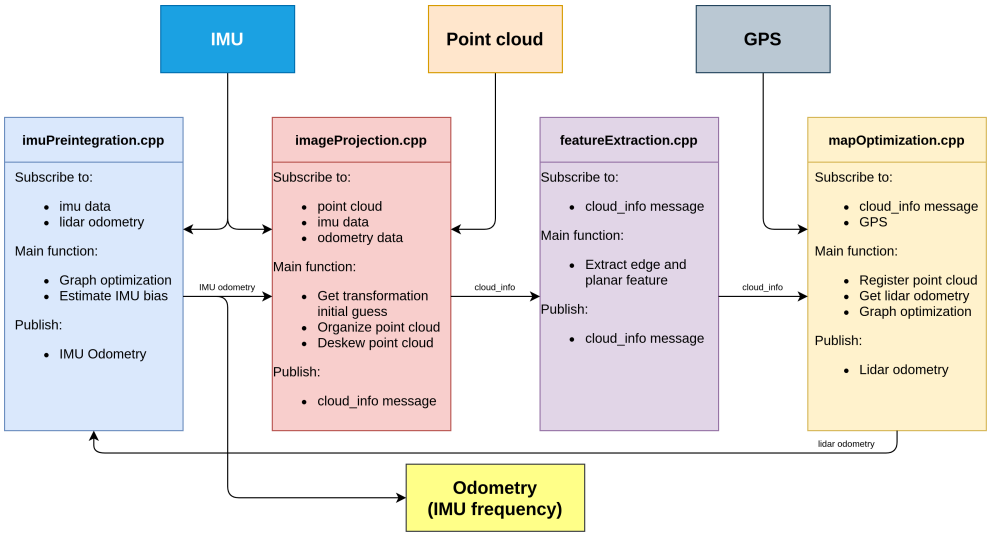


Figure 3.6: Lio-SAM System architecture overview.

[4] https://drive.google.com/drive/folders/1gJHwfdHCRdjP7vuT556pv8atqrCJPbUq

3.6 Results and Discussion

In order to compare the SLAM methods, two sets of experiments were conducted. The first set concerns 2D SLAM algorithms and the second one covered 3D SLAM algorithms with or without loop closure detection methods.

3.6.1 2D SLAM Results

Methodology

In our study, we aim to do a comparison between the 2D SLAM obtained by the GMapping algorithm in ROS and the 2D GraphSLAM in Matlab using 'Offline LiDAR DATA' recorded in ROS as rosbag files.

First, we create a simulation environment on the Gazebo simulator and we add the robot model of Turtlebot3 Waffle to that environment, then we will launch the GMapping package, which is responsible for performing the SLAM.

With this package we can create a two-dimensional occupancy grid map from laser and post data collected by a mobile robot. We would save the constructed map into a file to use it later for navigation. As mentioned before, GMapping uses sensor fusion technique based on laser scanners and odometer data to create a map of the environment.

If the robot does not move, the laser cannot reach all the locations of the environment because every laser scanner has a limited range, it can be 4 meters, 8 meters, or 20 meters. So, we need to move the robot to build the map for the entire environment. For this purpose, we use the teleoperation application to move the robot around the environment and therefore generate the map. Fig. 3.7 shows the flowchart of TurtleBot3 GMapping-SLAM algorithm.

For the experiments that will be conducted by the GraphSLAM algorithm implemented in Matlab, we will record the LiDAR data (and possibly other sensors' data) in a rosbag file while the robot moves around the considered environment in Gazebo. Then in Matlab, we will read the rosbag file, extract the LiDAR data and process it using GraphSLAM to generate the map and the the robot trajectory.

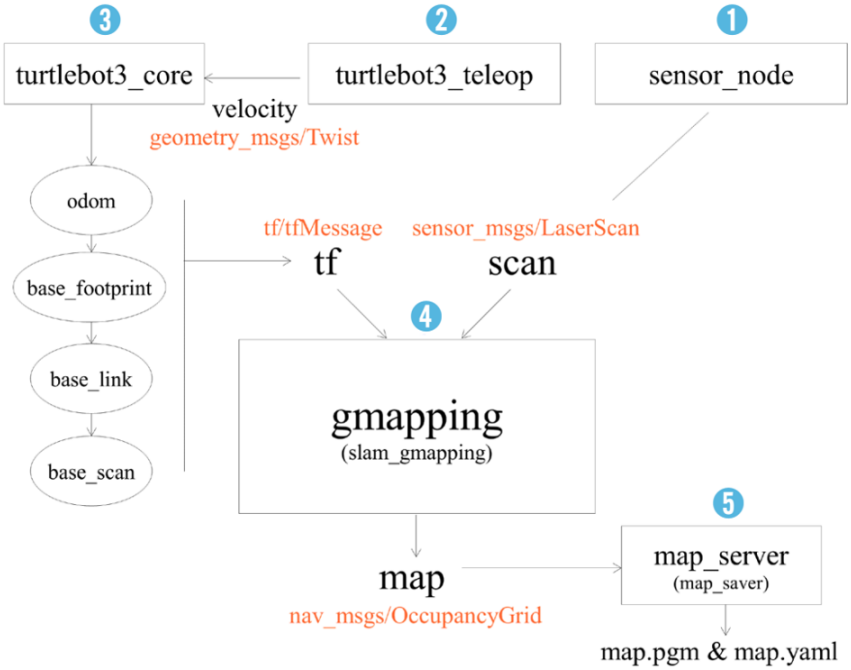


Figure 3.7: Flowchart of TurtleBot3 GMapping-SLAM algorithm.

Recording rosbag data using map-based navigation

As we mentioned before, to create a 2D map using GMapping algorithm, we had to move the robot around the place we want to map, using teleoperation. During the process, we noticed the difficulty of controlling the robot through the teleoperation. So in this section, we will try to overcome this issue by applying another method to control the robot which is easier and effective especially since our goal in this part is to record the environment offline-data through the robot's sensors in the format of .bag files. We also want to test the effectiveness and efficiency of the loop closure by rotating the robot around the same place a few times, which is difficult with the TurtleBot3-Waffle teleoperation.

The objective is to use a map of interest and request the robot to go to certain locations on that map during the offline-data record process. In the previous subsection, we have already learned the basic concepts of building a map with TurtleBot3 using the GMapping algorithm. However, in this part, we will learn how to use such a map to program navigation missions for the robot (see fig. 3.8).

For example, if we want to move the robot in the house environment, we will run

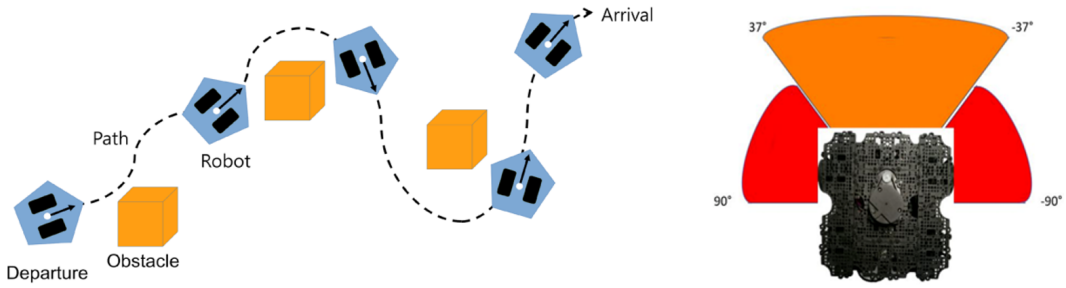


Figure 3.8: Avoiding obstacles during the navigation stack.

the command of the house Gazebo environment with TurtleBot3-waffle. Then, we need to load the navigation stack of the robot with the map of the environment (Generated by GMapping) as input. We use the command is shown in fig 3.9. Note that in the launch command, we provide as an argument, the path to the file that corresponds to the map of the environment.

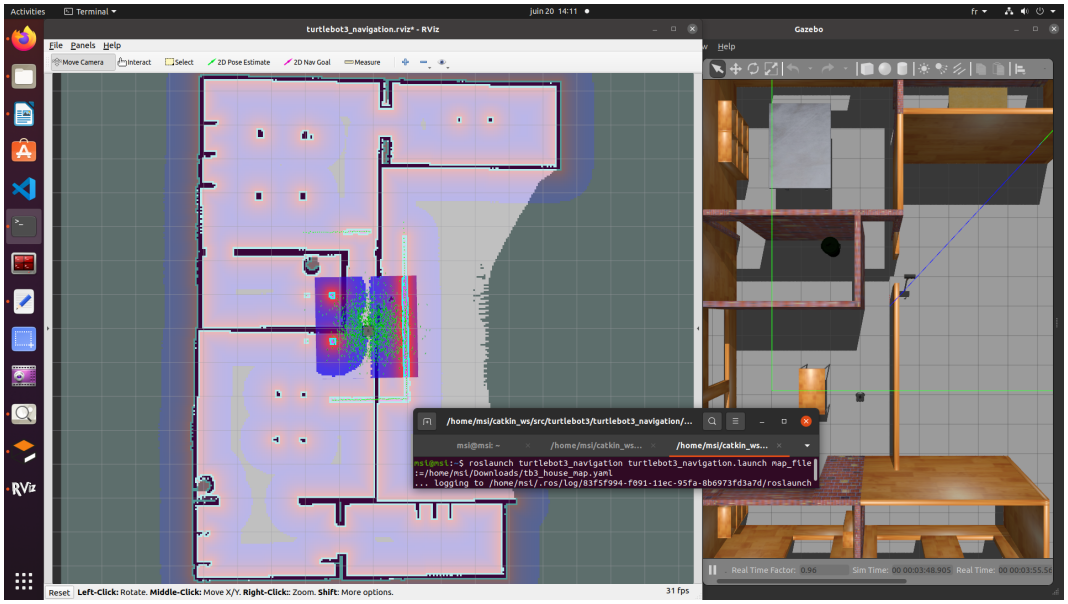


Figure 3.9: Navigation stack process in house environment.

When we start the navigation stack, RViz pops up and we can observe that the robot is initially placed at the center of the map in the coordinate $(0, 0)$. However, this initial location on the map does not correspond to the real location of the robot in the environment. This is normal because initially the navigation stack of the robot does not know where it is exactly located. Thus, we need to explicitly tell the robot its correct initial location manually and then it will know how to navigate afterwards.

Since the robot is badly placed, we notice that the laser scanner feedback shows some obstacles that do not correspond to reality. To fix this issue, we need to place the robot at its correct initial location. For this purpose, we need to use the button '2D pose estimate' and then press on the map at the correct pose (location and orientation) of the robot as shown in fig 3.9. Then, we can observe that the laser scanner becomes more consistent with the location of obstacles of the environment (see fig. 3.10).

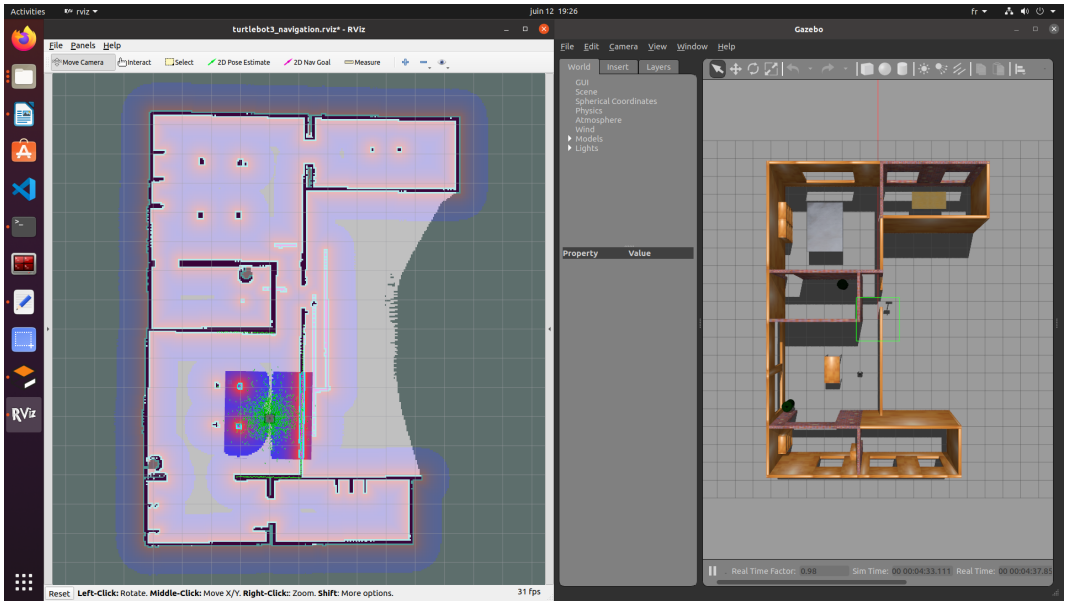


Figure 3.10: Moving the tb3 to the real position using 2D pose estimate button.

Now, we can send navigation waypoints or goal locations to the navigation stack of the robot so that it autonomously moves toward them. Of course, there are a lot of things happening behind the scenes when we send a location to the navigation stack where the global path planner/ local path planner and local cost-map/global cost-map are activated (see fig. 3.11).

To do so, we use the global path planner which is responsible for finding a global obstacle-free path from the initial location to the goal location using the environment map obtained from GMapping. In addition, the local path planner is responsible of the execution of the static path determined by the global path planner while avoiding dynamic obstacles that might come into the path using the robot's sensors. Fig. 3.12 and fig. 3.13 show the process of navigation when we give a pose goal to tb3 (turtleBot3) in the 2D map. As we can see the robot takes the shortest possible path.

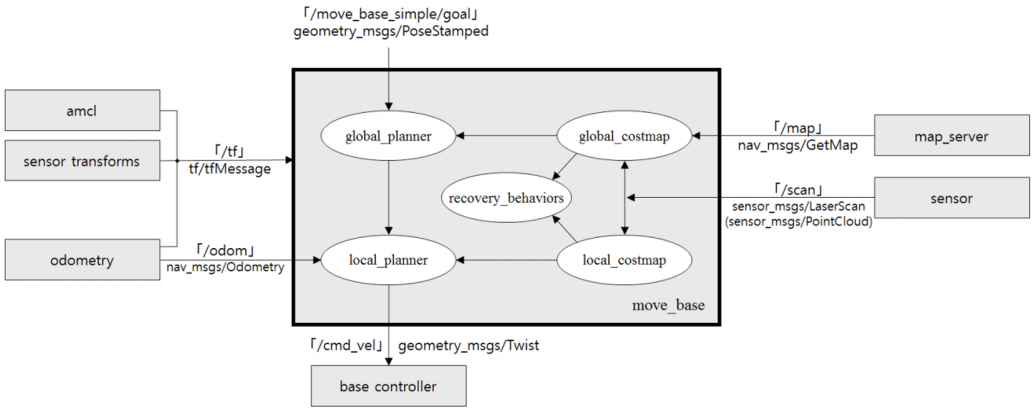


Figure 3.11: Relationship between essential nodes and topics on the navigation packages configuration.

Finally, we dive into the following record process by collecting our offline raw data. We take each time one of the two environments (a room in Gazebo house shown in fig. 3.4, and Gazebo world shown in fig. 3.3). Note that we didn't use the tb3-teleoperetion to move the robot; but instead, we used the map-based navigation method detailed below which makes navigation easier and faster using Global path planner and local path planner. Our goal is to move the robot around the target environment several times to make it do loop closures while recording scan measurements. It should be noted that the collected information was not limited to the scanner only, but also the camera, IMU measurements beside other important information from other sensors. We will present in the next subsection the way that we visualize some results recorded by the camera and the laser scanner. The acquired measurements are saved in rosbag format (see fig.3.14) and then processed using the Matlab GraphSLAM approach to build 2D maps and trajectories.

After recording and collecting our offline data, we can visualize it using 'rqt-bag' command. The fig. 3.15 and fig. 3.16 show the visualization process of the recorded rosbag files in the two selected environments. In addition to LiDAR data, we can also save other types of data such as images acquired by a camera sensor, 'IMU' messages as a row, 'geometry-msgs/twist' as a row and the 'OccupancyGrid' which we can also read in Matlab.

The scheme of all the connected nodes during the recording process is presented

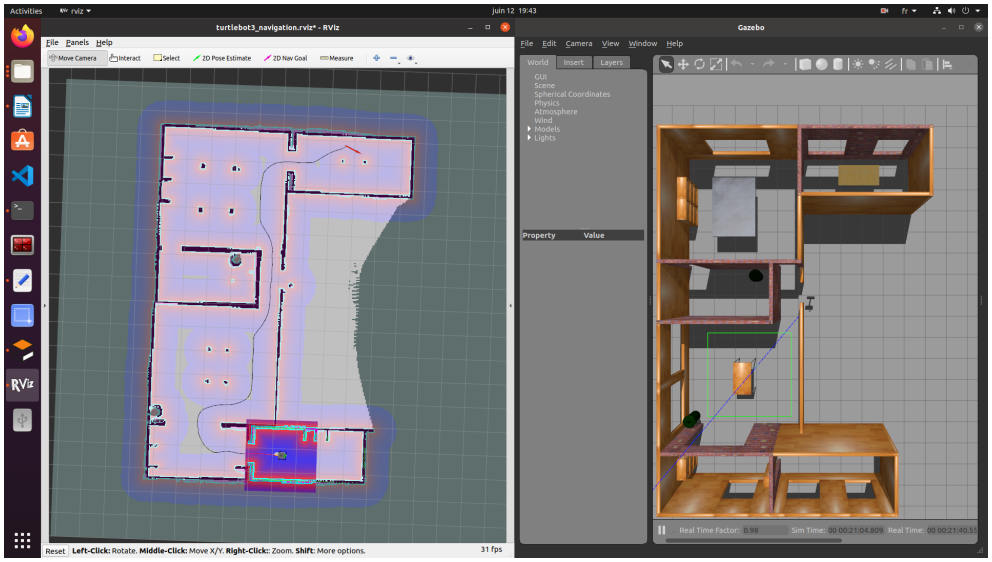


Figure 3.12: Navigation with tb3 in house environment.

in fig 3.17 using 'rqt-graph' command. From the first look at the diagram, we can see different parameters including all the active and connected nodes. Note that the navigation package is also connected because the record process is done using map-based navigation. However, the teleoperation node doesn't appear in the rqt-graph diagram, unlike the case of the mapping process using GMapping package.

Results

A. GMapping Results

In this section, we will present the maps obtained using GMapping for the house and world Gazebo environments as shown in fig. 3.18 (the details on how to generate maps in ROS using GMapping package are given in Appendix A).

We observe that the gray area represents an unknown or non-explored space, the white area represents the free space and the black area represents obstacles; for example, the walls shown in the previous figure.

Fig. 3.19 shows the trajectory of the TurtleBot3 when it was exploring the Gazebo world environment using teleoperation. This trajectory is obtained from the odometry input of the GMapping package.

From the output results below, we conclude that the GMapping algorithm is a

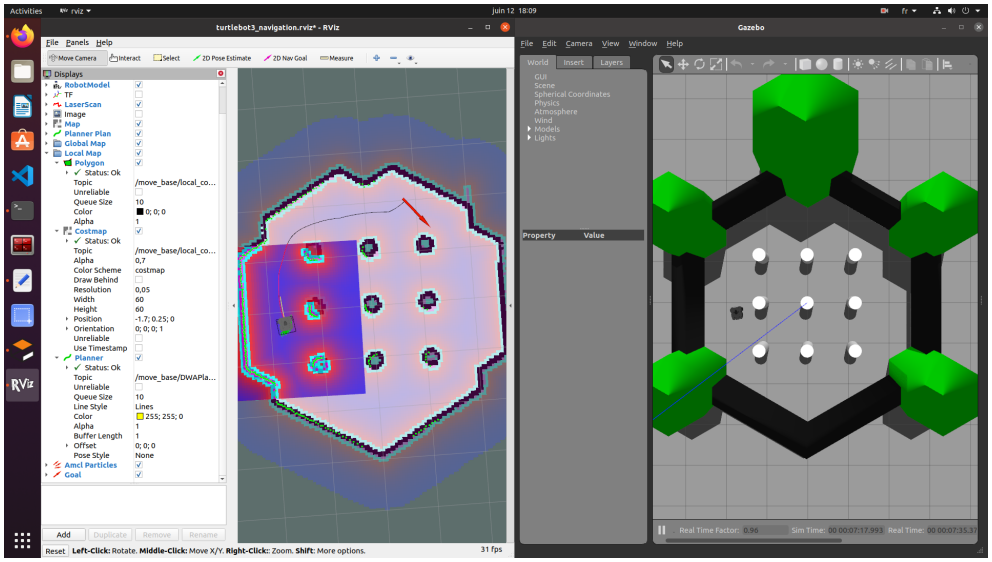


Figure 3.13: Navigation with tb3 in world environment.

great choice for real-time environment mapping. But being dependent on odometry input obtained from the TurtleBot, the GMapping method is not recommended for long trajectories.

B. 2D GraphSLAM results

Now, we aim to build 2D maps from 2D LiDAR scans using GraphSLAM algorithm implemented in Matlab. First, we extract the LiDAR data from the rosbag file, and then we use it as input to our GraphSLAM algorithm. In our tests, we will consider two cases. In the first case, the algorithm constructs a Pose Graph, detects the loop closures and therefore estimates the trajectory of the robot and builds the map of the considered environment without optimizing the Pose Graph. In the second case, the algorithm optimizes the Pose Graph after the detection of loop closures to see its effect on the obtained results.

- Test 1: in this test, we consider a short trajectory that consists of one loop in the Gazebo world and a room in the house environment. Fig. 3.20 shows the trajectory (in blue) of the robot before and after the Pose Graph Optimization (PGO) and the loop closure edges (in red) which represent previously visited places recognized through the Scan-Matching based algorithm. We notice that the estimated robot trajectory before optimization drifts over time. This drift is due to noisy scans from the sensor or may be due to inaccurate initial transformation especially when the rotation is significant.

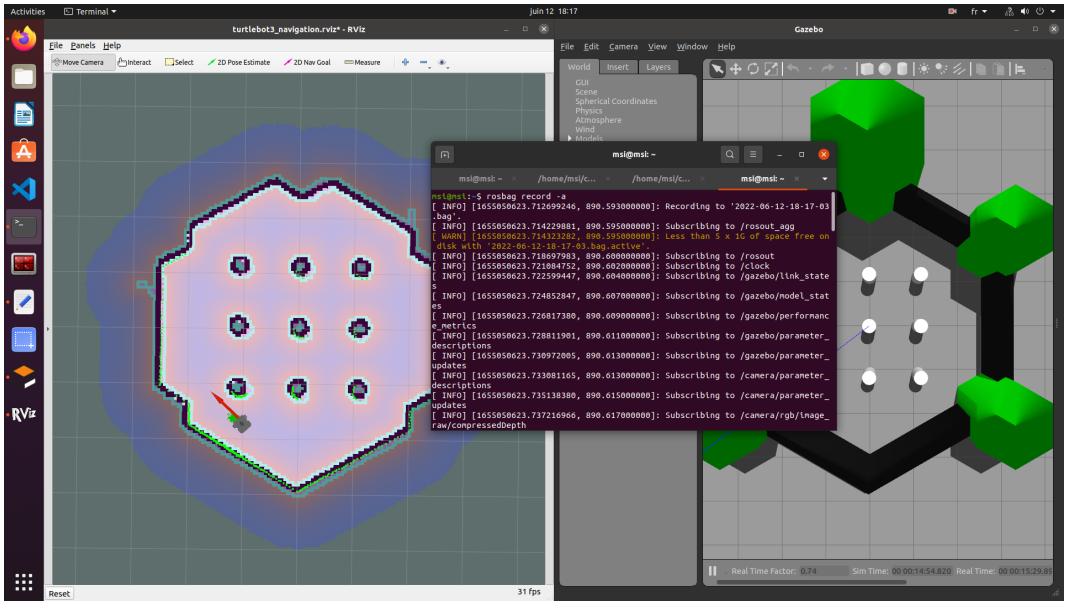


Figure 3.14: Recording rosbag Files in Gazebo world environment.

This can be seen clearly in fig. 3.21 where the map in the left is not well accurate. The GraphSLAM algorithm uses loop closure information to optimize poses and update the 2D map as shown in the right of fig. 3.21 where the map is more accurate than the one on the left. Fig. 3.21 shows the estimated and optimized trajectories in the two environments.

Fig. 3.23 shows some images extracted from the video sequence filmed by the camera mounted on the Turtlebot robot during its travel in the room environment. These images are read from the rosbag file; they show that the robot starts from one position as illustrated in image one, moves in the room and returns to the same position in the last image.

- Test 2: in this test, we consider a long trajectory that consists of several loops in the Gazebo world and a room in the house environments. The objective is to prove the importance of the loop closure detection in a GraphSLAM especially for long trajectories (with several loop closures) where the drift increases over time. We can see in fig. 3.24 that the maps in the left are less accurate because of the high drift whereas the maps in the right are fine and more precise thanks to the optimization using the loop closure information.

Fig. 3.25 shows that the accuracy of the map and the trajectory increases with the

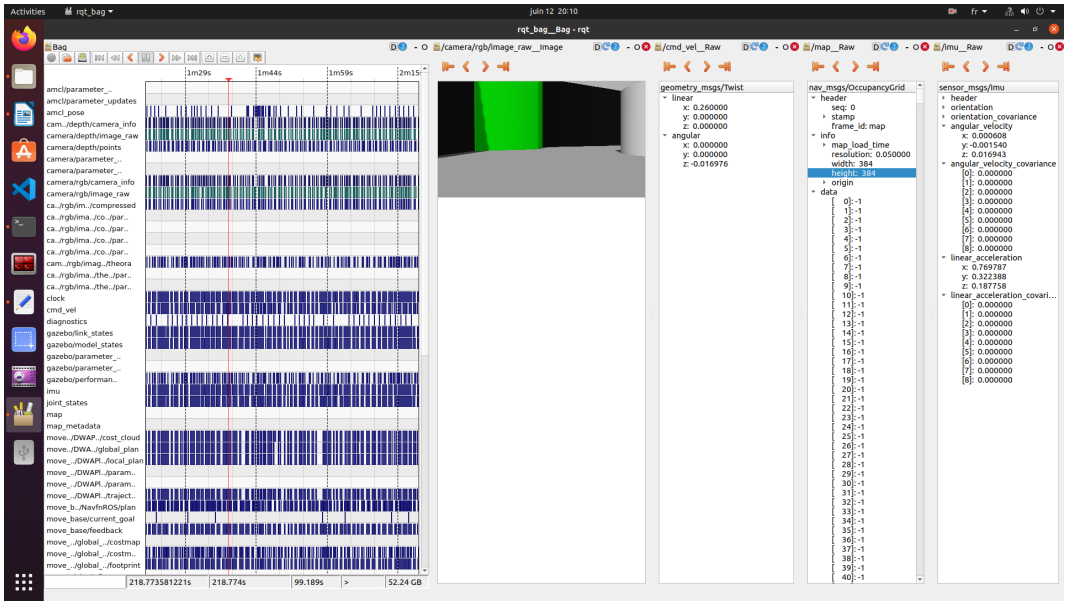


Figure 3.15: Reading the rosbag file of the tb3 in world environemet.

number of loop closures even if the trajectory is long.

C. Comparison between "GMapping" and GraphSLAM

This study investigates two SLAM algorithms, one of them is provided on open-source ROS framework: GMapping package and the GraphSLAM implemented in Matlab using the same indoor environment and the same scan measurements to do reasonable comparison.

Here we aim to compare the results of the two SLAM approaches in terms of 'map accuracy'.

To do so, we can say that the obtained results are almost similar in case of short trajectories. But in large environments and long trajectories the GMapping is not efficient because it depends on the odometry of the TurtleBot and it is highly sensitive to errors therein. In addition GMapping does not include a loop closure detection method unlike the GraphSLAM algorithm whose results demonstrate a strong correlation between the accuracy of the resulting 2D maps and two basic parameters: the loop closures and the PGO effect. Also, GMapping has a limitation of navigation in real-life environment via teleoperation using keyboard, which is impractical, unlike the other method that uses the navigation stack to move autonomously.

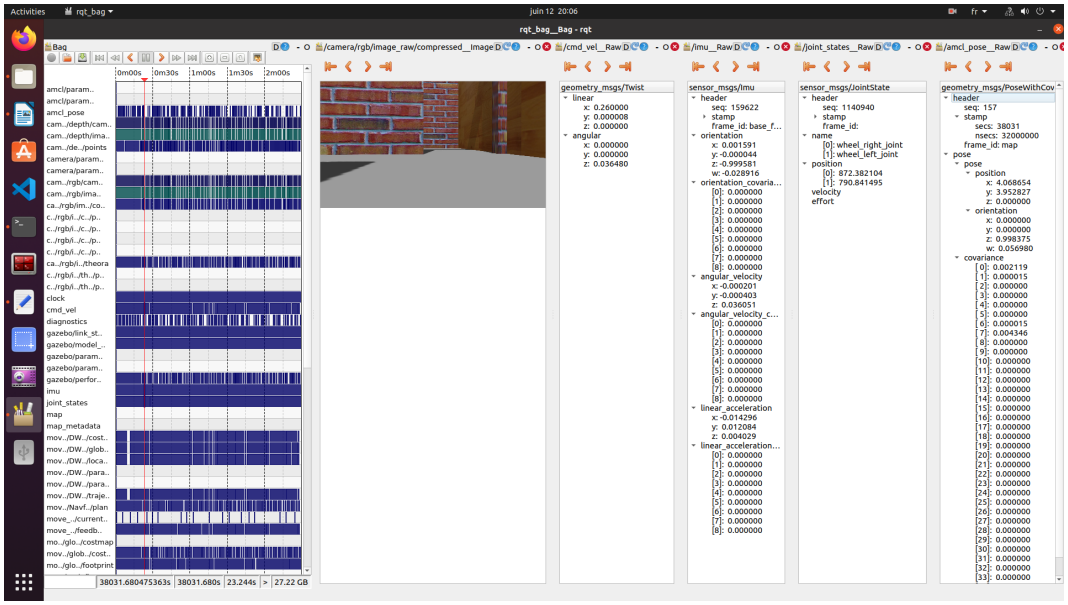


Figure 3.16: Reading the rosbag file of the tb3 in house environemet.

3.6.2 3D SLAM Results

3D GraphSLAM with Scan-Context loop closure detection

In this section, we will present the results obtained by a 3D GraphSLAM using a loop closure technique based on Scan-Context Descriptor and compare it with the same GraphSLAM using the conventional Scan-Matching based loop closure method. To do so, we test the SLAM system on a dataset which consists of two challenging 3D outdoor scenarios of 3D LiDAR data from a Velodyne sensor mounted on a vehicle. The 3D GraphSLAM will process this data to progressively build accurate 3D maps, and estimate the trajectory of the vehicle. In addition, Inertial Navigation Sensor (INS) is used to help build and update the 3D maps.

A. Tests on scenario 1

- **Step 1:** 3D trajectory before Pose Graph Optimization

Fig. 3.26 shows that the vehicle path consisting of two loops and many turns and returns to the starting point of coordinates $(0, 0, 0)$. But this trajectory drifts over time and the accumulated drift results in the second loop terminating a few meters away from the starting point.

- **Step2:** Scan-Context based loop closure detection before PGO

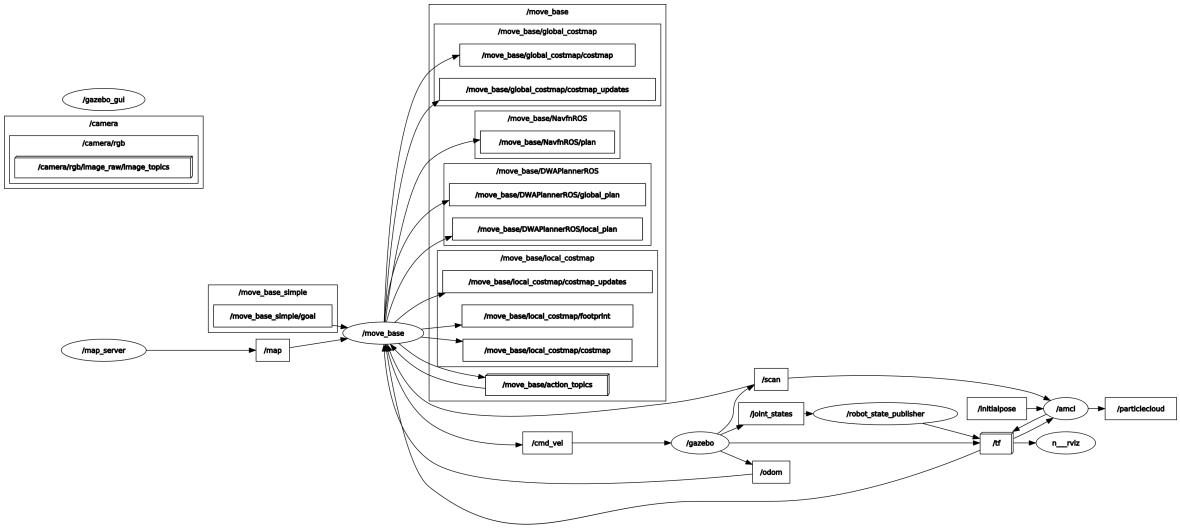


Figure 3.17: rqt-graph scheme of the offline data recording process.

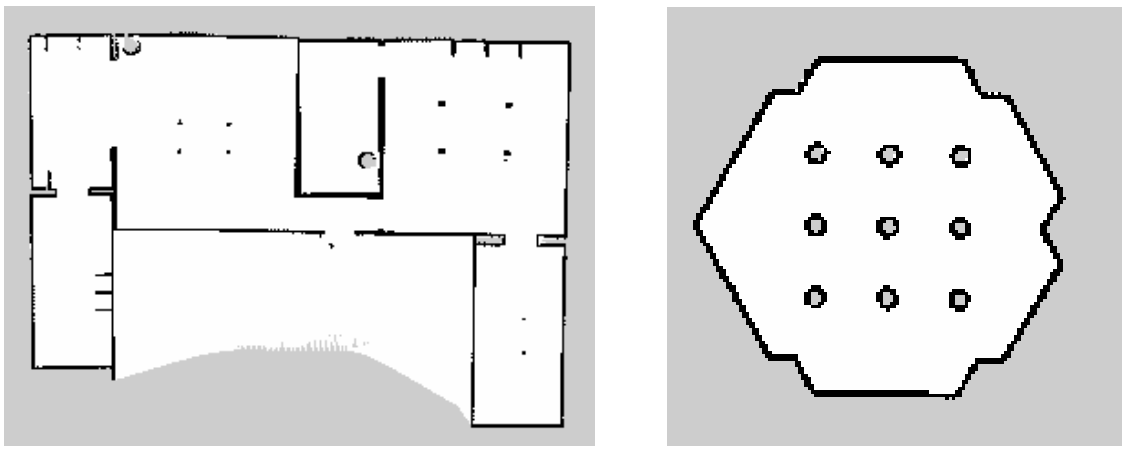


Figure 3.18: Maps obtained by GMapping for both environements.

A key aspect contributing to the effectiveness of graph SLAM in correcting drift is the accurate detection of loops. We used here the Scan-Context descriptor to detect loop closures. Fig. 3.27 illustrates a zoom of the estimated trajectory showing loop closures edges.

- **Step 3:** Pose Graph Optimization

Here, the Pose Graph is optimized using loop closure information. The result is presented in fig. 3.28 where we can see at the end that the trajectory returns exactly to the initial position $(0, 0, 0)$. Fig. 3.29 show the map obtained after optimization.

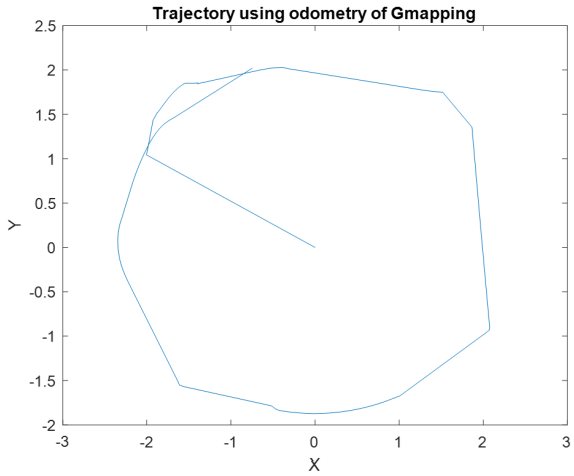


Figure 3.19: rqt-graph scheme of the offline data recording process.

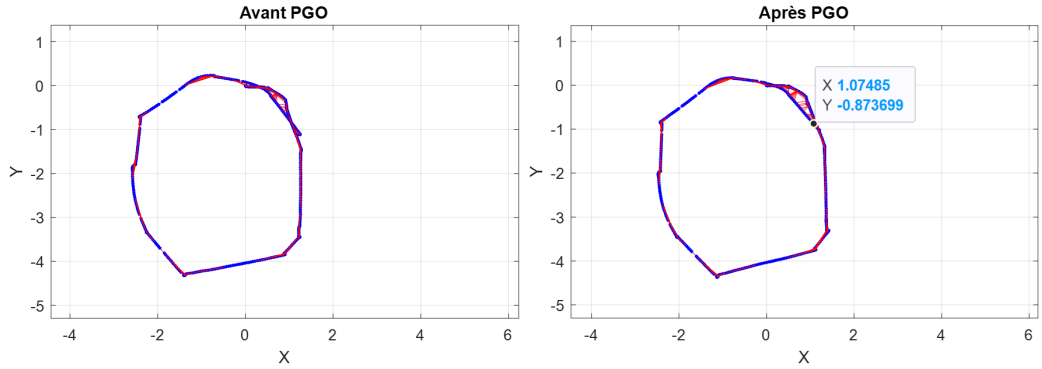


Figure 3.20: Trajectory of the robot in the world environment.

To prove the efficiency of the Scan-Context method we compared it with the Scan-Matching based loop closure method. The figure 3.30 below shows 'loop closure connections' before the optimization of the PoseGraph and fig 3.31 presents the optimized trajectory. We notice that the robot do not reach the initial position at the end of its travel as is done in the Scan-Context method (see 3.28) which mean that there is still a drift in the trajectory proving the superiority of Scan-Context method compared to the conventional Scan-Matching based method. The occupancy map is presented in fig. 3.32.

B. Tests on Scenario 2

In this sub-section, we have conducted the same experiments as in the previous sub-section but on the scenario 2 data. Fig. 3.33 the estimated and the optimized trajectories using the Scan-Context loop closure method. We can see that the optimized

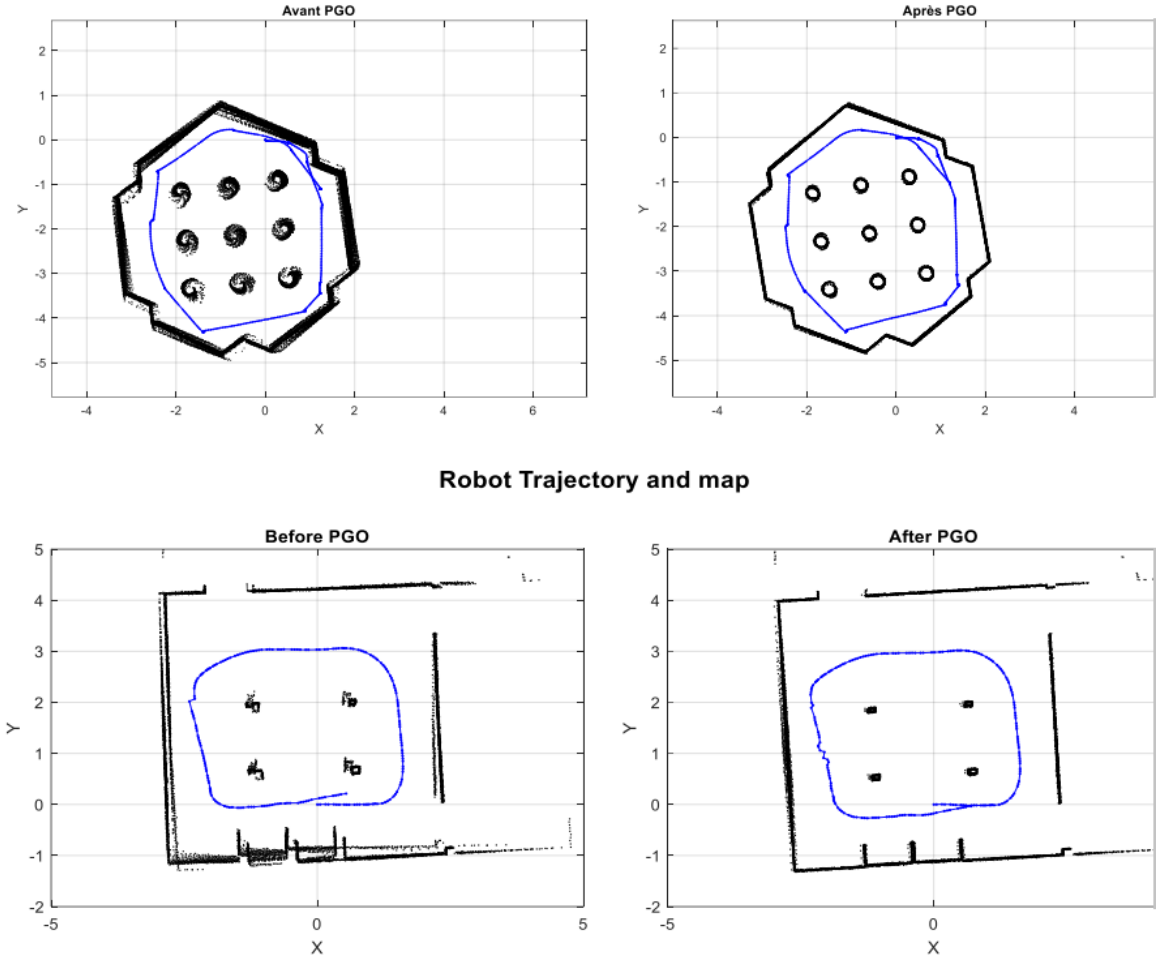


Figure 3.21: Estimated trajectory and map of the world and room environments; Left: before PGO; Right: after PGO.

trajectory reaches the initial position proving the efficiency of the Scan-Context descriptor for loop closure detection. Fig. 3.34 shows the obtained 3D map. When testing the Scan-Matching based loop closure technique, we notice that no loop closures are detected and hence there is no drift correction.

LIO-SAM

A. Methodology

LIO-SAM [29] is forced to run in real-time. We describe a series of experiments to qualitatively and quantitatively analyze the Lio-SAM's framework. The sensor suite used in the experimental data includes a Velodyne VLP16 LiDAR, a MicroStrain 3DM-

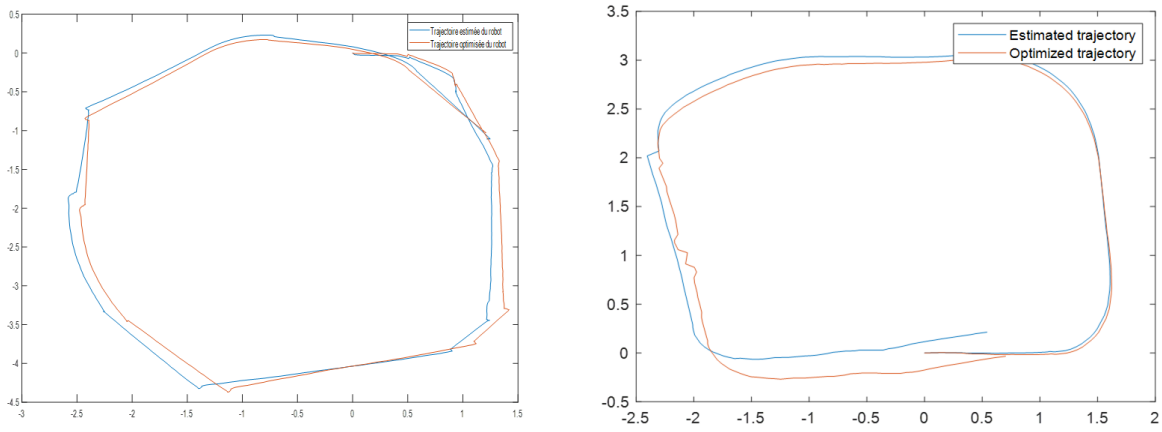


Figure 3.22: Robot estimated and optimized trajectories in the world environment in the left and the room environment is in the right.

GX5-25 IMU, and a Reach M GPS. We use here the information collected was from two different datasets across various scales. These datasets are referred to as Walking and Drive, respectively. The mounted sensor is shown in fig. 3.35. The datasets were collected using a custom-built handheld device on the MIT campus. The metadata of these datasets are shown in fig. A.3 and fig. A.3 in the appendix A.

Walking dataset contains 6502 Scans with 0.3m of elevation change and 801m as trajectory length, while the maximum rotation speed is 133.7 (°/s).

B. Loop closure factor

Thanks to the utilization of a factor graph, loop closures can also be seamlessly incorporated into the proposed system, as opposed to LOAM [19]. For the purposes of illustration, LIO-SAM implements a naive but effective Euclidean distance-based loop closure detection approach. We also note that the framework is compatible with other methods for loop closure detection such as Scan-Context [7].

In practice, adding loop closure factors is especially useful for correcting the drift in a robot's altitude, when GPS is the only absolute sensor available. This is because the elevation measurement from GPS is very inaccurate giving rise to altitude errors approaching 100m in the absence of loop closures.

C. Results and discussion

LIO-SAM is implemented in C++ and executed on a personal computer equipped with an Intel i5-7400U CPU using ROS Noetic in Ubuntu 20.04 focal. We note that

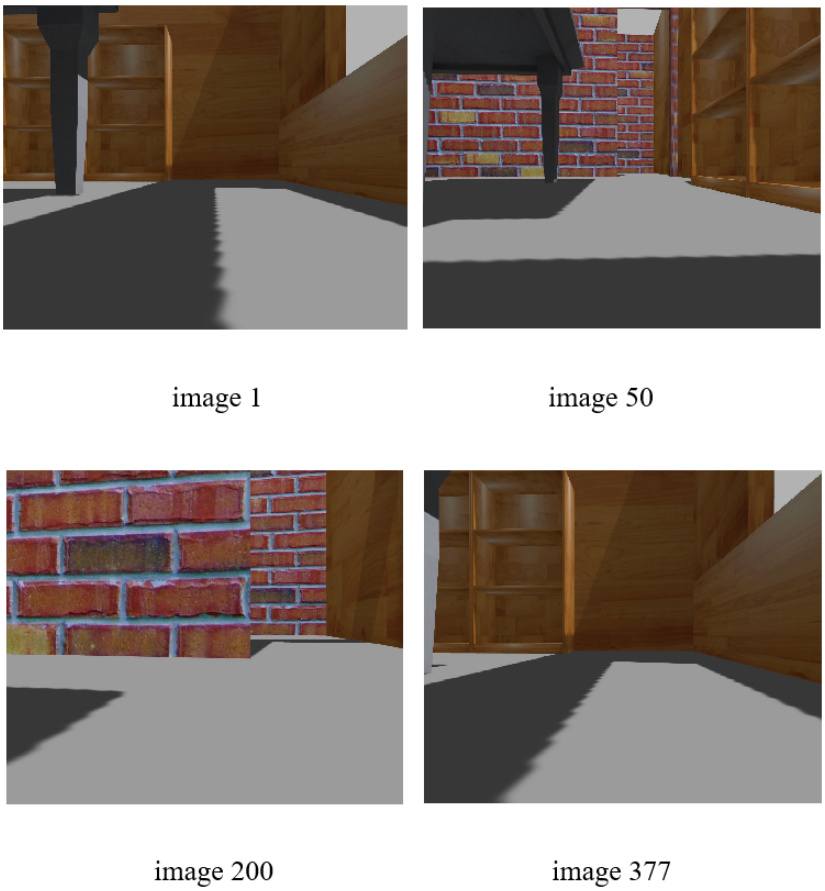


Figure 3.23: Robot estimated and optimized trajectories in the world environment in the left and the room environment is in the right.

only the CPU is used for computation, without parallel computing enabled. The implementation of LIO-SAM is freely available on Github. Supplementary details of the experiments performed, including all the outputs (trajectory, transformations, GlobalMap, CornerMap, and SurfMap which are all presented as .pcd files), can be found at the appendix below. In order to make the reader comfortable, we present the results in the form of elements as follows:

- **Case 01:** Walking dataset [30]

The test we present here is designed to evaluate the performance of the method when the system undergoes aggressive translations and rotations. The maximum translational and rotational speed encountered in this dataset is 1.8 m/s and 213.9 (°/s) respectively. During the data gathering, the user holds the sensor suite shown in fig. 3.35

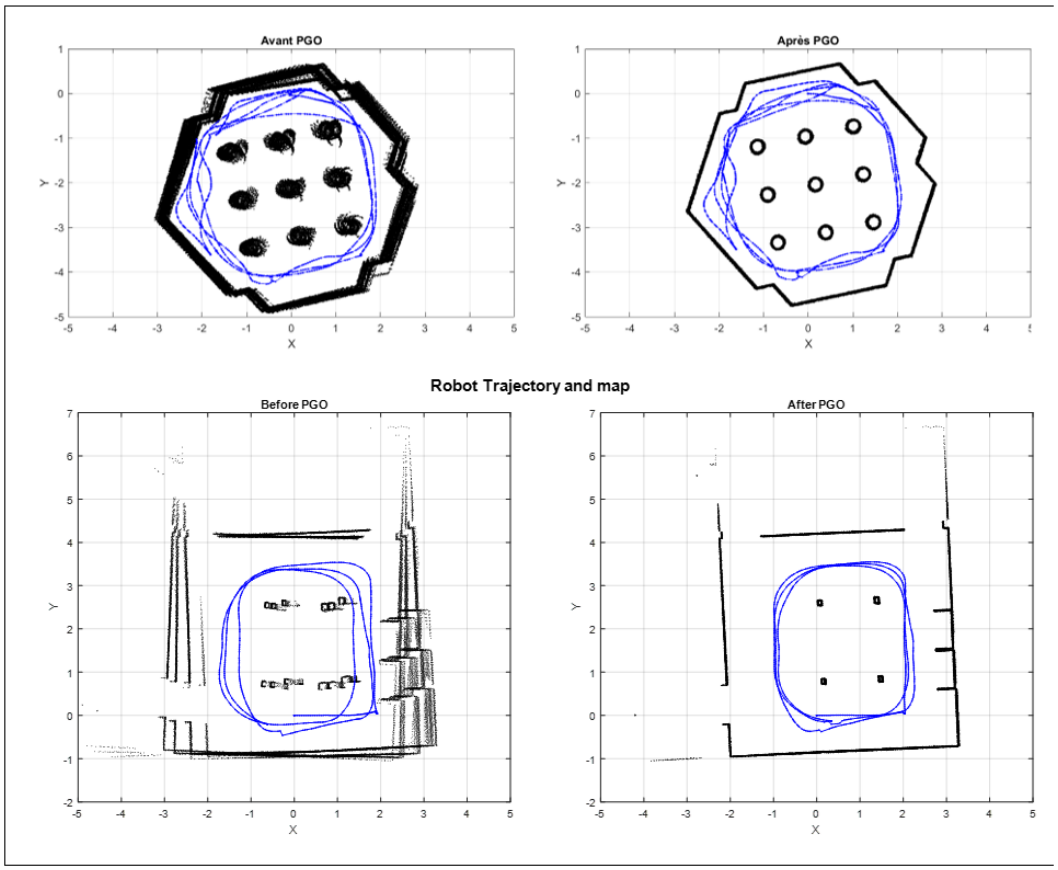


Figure 3.24: Trajectories and maps of the two environments (up: world, down: one room) before PGO and after PGO.

and walks quickly across the MIT campus. However, LIO-SAM outperforms produces a map that is consistent with the available Google Earth imagery (see fig. 3.36).

The maximum data playback speed is recorded when LIO-SAM achieves similar performance without failure compared with the results when the data playback speed is 1× real-time. As is shown, LIO-SAM is able to process data faster than real-time up to 13×.

We note that the runtime of LIO-SAM is more significantly influenced by the density of the feature map, and less affected by the number of nodes and factors in the factor graph (see fig 3.37). After the execution of the LIO-SAM algorithm, we can move the resulting map around to see all the hidden parts of the map. Fig; 3.37 is an example of a selected position.

The outputs of LIO-SAM are 5 point-cloud files: GlobalMap (fig. 3.37), trajectory (fig. 3.38), transformations (fig. A.6), SurfMap (fig. A.7) and CornerMap (fig. A.8).

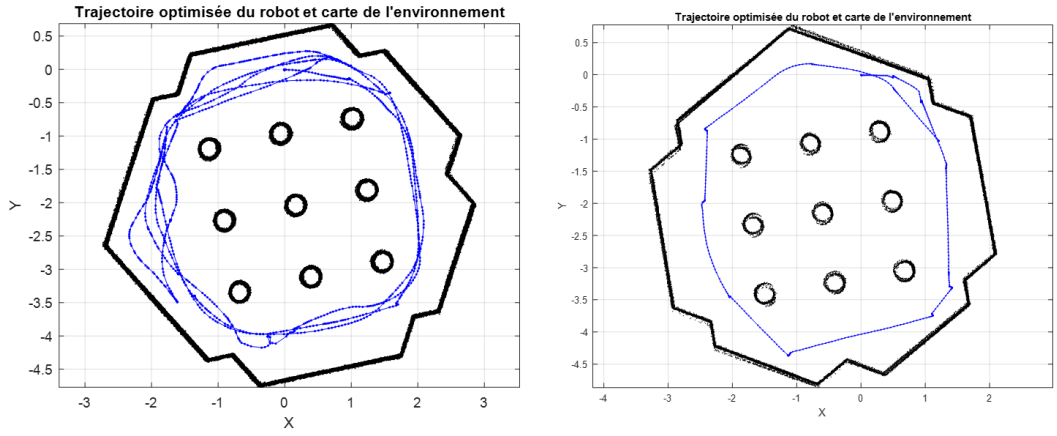


Figure 3.25: The effect of the number of closures on the map and the trajectory.

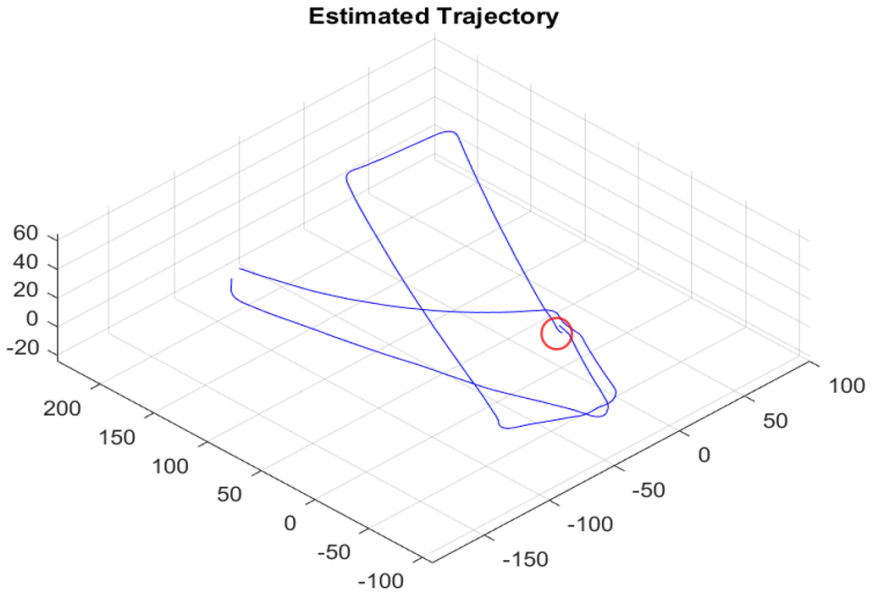


Figure 3.26: The estimated trajectory of scenario 1 before PGO.

The color bar shown in fig. 3.38 means the time scale, and the global map gives a clear view of the environment.

Fig. 3.40 shows clearly the yellow, blue, and red edges, which presents the loop closure detection. The color of the map the hue of the map indicate the difference in elevation of the building we want to create a 3D map.

- **Case 02:** Drive dataset [30]

Of course, changing the dataset will not change the outputs of the algorithm, fig. 3.41 and fig. 3.42 present the global map and the trajectory successively. While

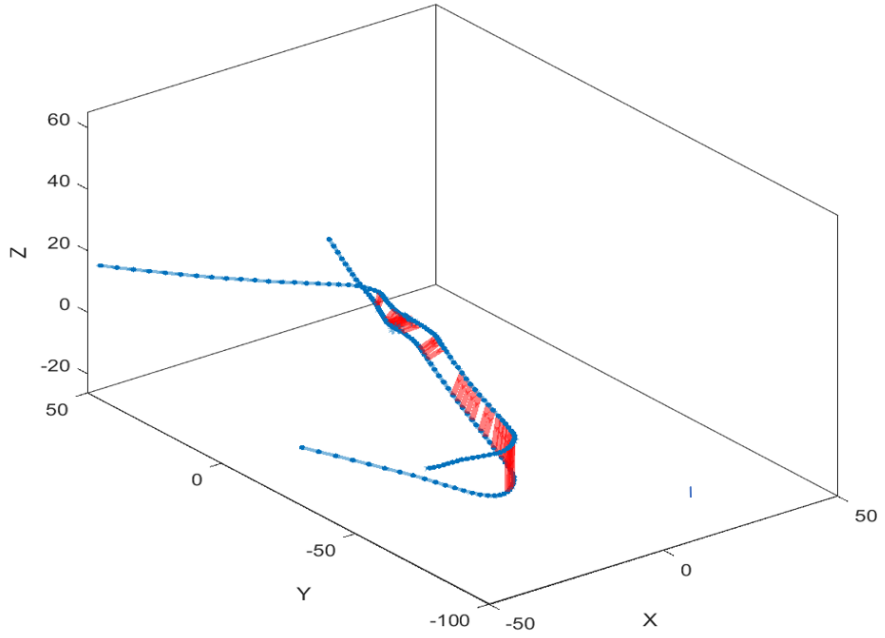


Figure 3.27: Loop closure edges obtained using Scan-Context descriptor (before PGO).

the other results are attached in the appendix A. Notice that the resulting map is different, while the type of this map is topologic unlike the map resulting from the walking dataset, which is a metric map.

The color bar shown in fig. 3.41 means the time scale, and the global map gives a clear view of the environment (see fig. 3.42).

It is worth noting that the five point-cloud files that are the results of the LIO-SAM algorithm were read using the point-cloud player integrated in Matlab.

Another important result is the diagram of all the active nodes during the LIO-SAM mapping process. The command rqt-graph offers detailes diagram that is shown in fig. 3.43.

Here we can observe some important notes from the results above:

- During indoor mobile mapping, localization plays an important role. It is difficult for IMU, odometer, LiDAR singly to meet the high efficiency, real-time, accurate, and robust performance. So, the fusion of the data measured by these different sensors is a current research hotspot. However, most of the researche still focus on the loosely coupled fusion based on filtering methods, and the data cannot be fully utilized. In this thesis, based on LIO-SAM framework, also,

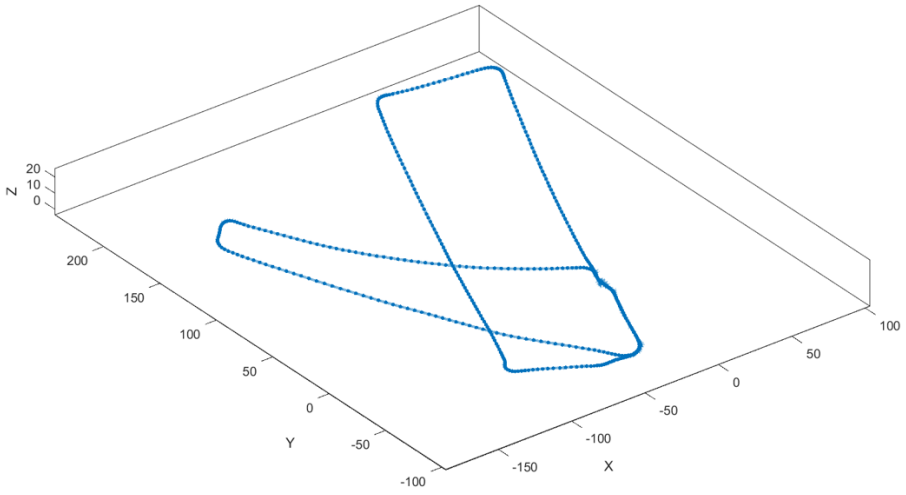


Figure 3.28: The optimized trajectory using Scan-Context descriptor (after PGO).

the Scan-Context is introduced to extract keyframes and detect loop closure in a robust manner which will provide a closed-loop, together with the IMU pre-integration factor and LiDAR odometry factor to construct the factor graph (see fig. 3.43). Then incremental smoothing optimization algorithm is used to get a high precision trajectory, realize the tightly-coupled LiDAR and IMU positioning. Thus, the results show that the number of keyframes is reduced, the elevation error is effectively decreased, and the real-time performance is improved without decreasing the accuracy of LIO-SAM.

- The aim of this thesis is to evaluate the accuracy of a SLAM algorithm (LIO-SAM), in rural environments, as well as evaluate the effect of different algorithmic configurations on the accuracy.

- Observe that the accuracy is evaluated by comparing the trajectory created by the SLAM algorithm to a reference trajectory measured using a commonly employed LiDAR mapping method that is based on the global navigation satellite system (GNSS) and inertial measurements (IMU). Note that by using the reference trajectory, the relative and absolute accuracy is evaluated for each SLAM configuration and dataset (walking+drive, see fig. A.3 and fig. A.4). Additionally, rigid alignment errors to world coordinates and local consistency errors are evaluated.

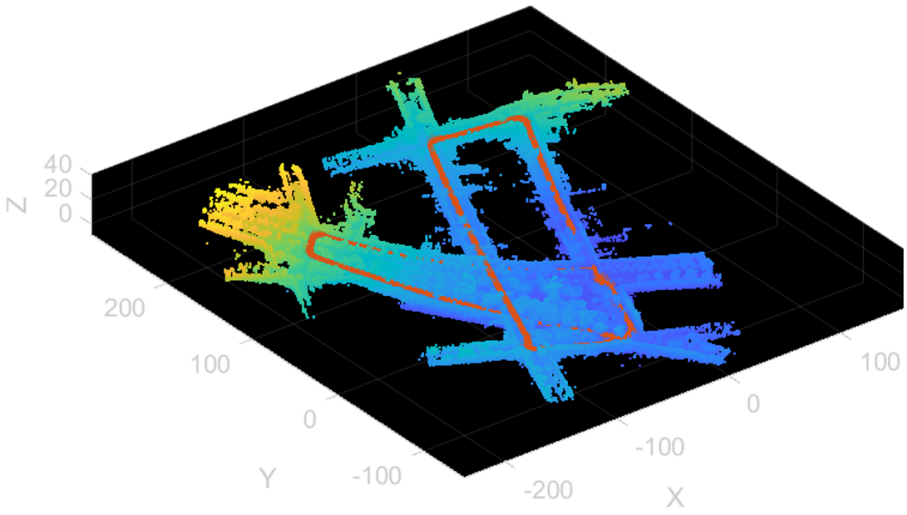


Figure 3.29: The 3D map using Scan-Context descriptor (after PGO).

- We analyze the data and the results show that the obtained accuracy of SLAM depends on the SLAM configuration and on the environment. Contrary to the initial expectation, rural environments did not always have worse accuracies than urban environments. However, based on the results, the accuracy of LIO-SAM deteriorates in environments in which only a few features are visible in the direction of travel (such as open fields and highways).

3.7 Conclusion

Real mobile robots require logistics, such as laboratory space, battery replacement, and operational quirk. Even the greatest robots go fail in the real world due to a variety of factors including operator errors, environmental circumstances, and manufacturing or design defects. By using virtual robots that move in a simulated world, these issues may be resolved. Thus, Software robots are extraordinarily useful, in the simulation we can model as much or as little reality as we desire. Sensors and actuators can be modeled as ideal devices, or they can incorporate various levels of distortion, errors, and unpredicted faults.

SLAM algorithms have received much attention due to their capability of constructing globally consistent maps. SLAM implementations are commonly evaluated in urban environments, which is why they can be expected to work accurately in ur-

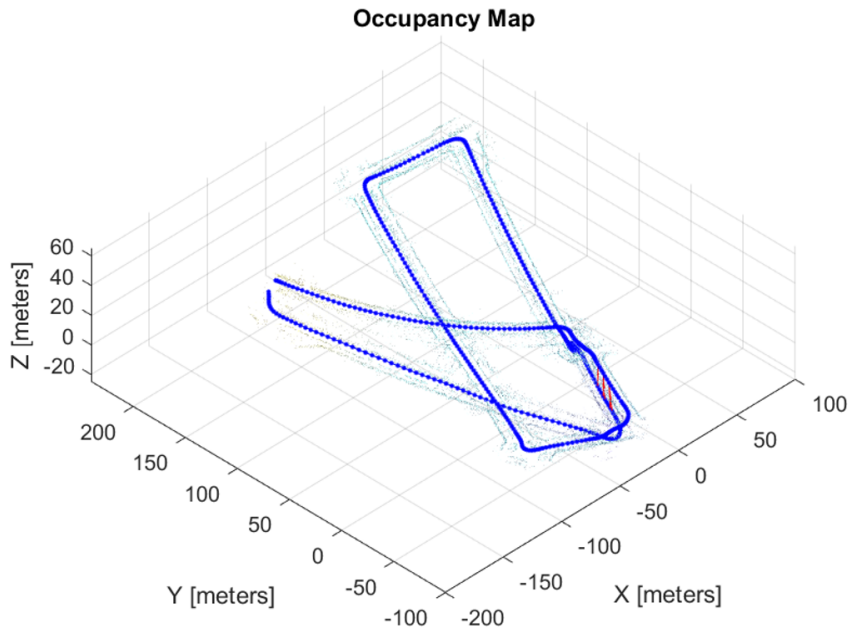


Figure 3.30: Place recognition using Scan-Matching based loop closure detection.

ban environments. However, SLAM would have meaningful applications also in rural environments using LIO-SAM or Scan-Context algorithms.

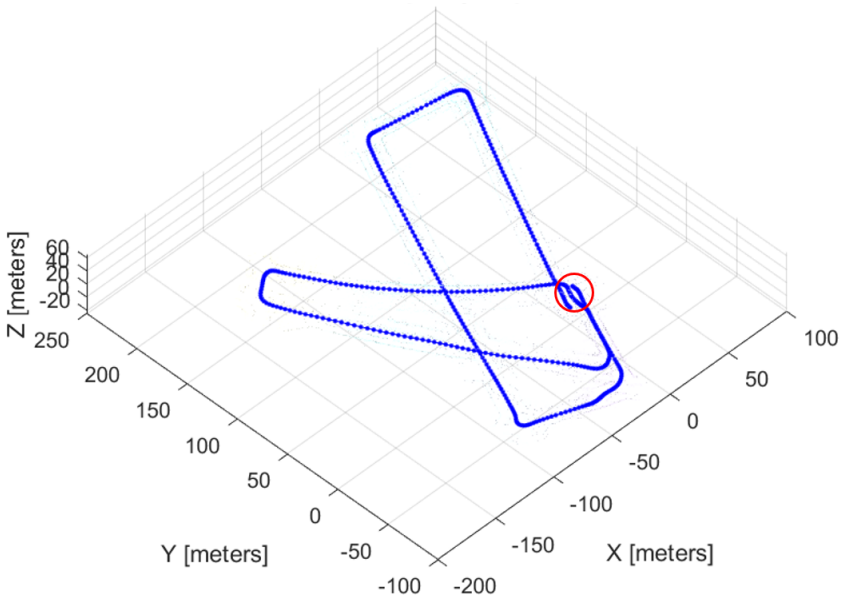


Figure 3.31: The optimized trajectory using Scan-Matching based loop closure method (after PGO).

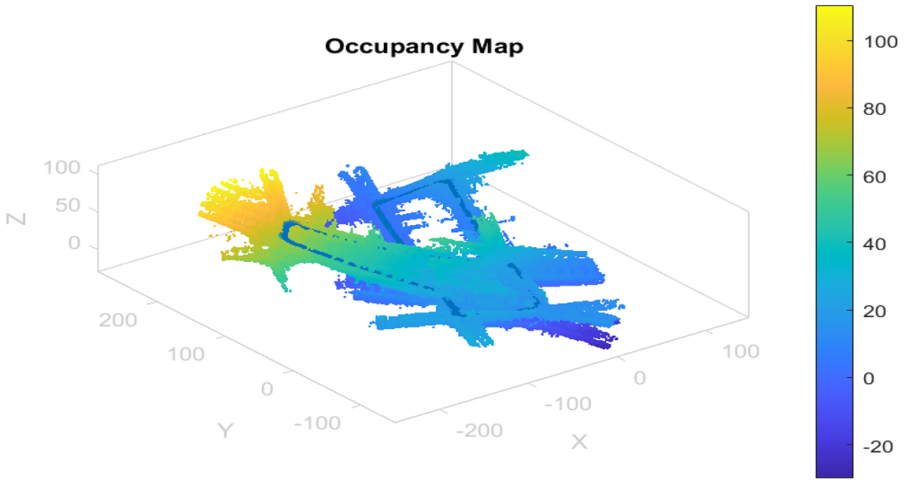


Figure 3.32: 3D optimized map of scenario1 using Scan-Matching based loop closure method.

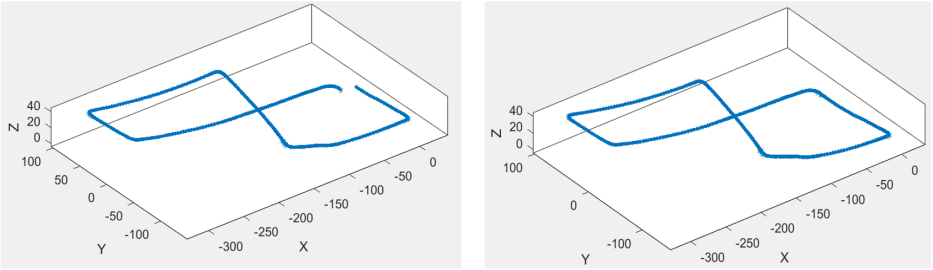


Figure 3.33: The estimated and optimized trajectories of scenario 2 using Scan-Context method before PGO (left) an after PGO (right).

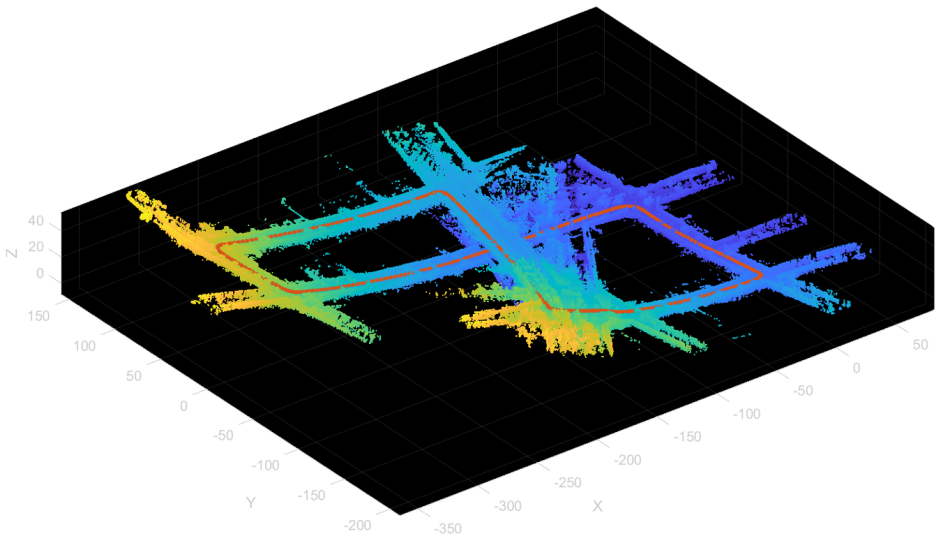


Figure 3.34: The optimized 3D map of Scenario 2 using Scan-Context descriptor (after PGO).

Figure 3.35: Velodyne Handheld device sensor.

Figure 3.36: Google Earth picture of MIT campus.

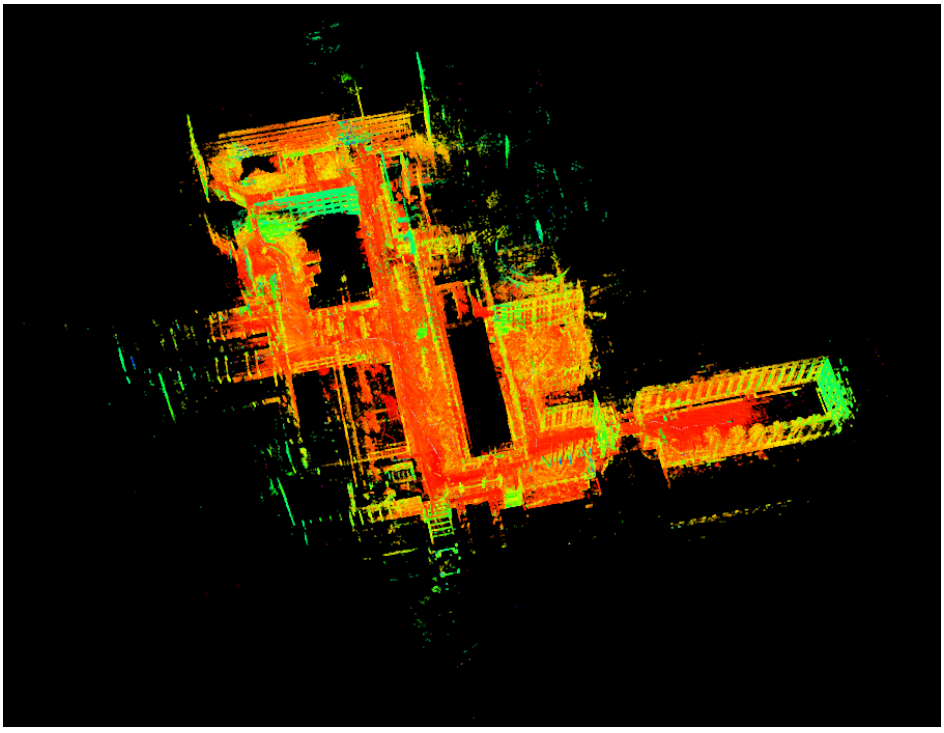


Figure 3.37: Resulting 3D map of LIO-SAM.

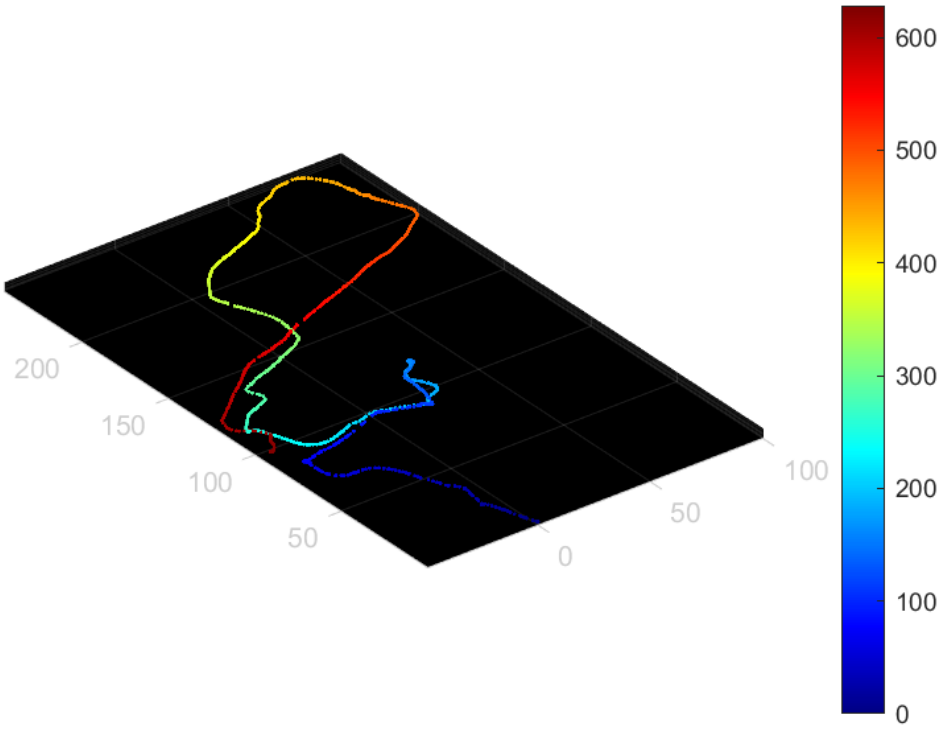


Figure 3.38: The full trajectory of walking dataset resulting by LIO-SAM algorithm.

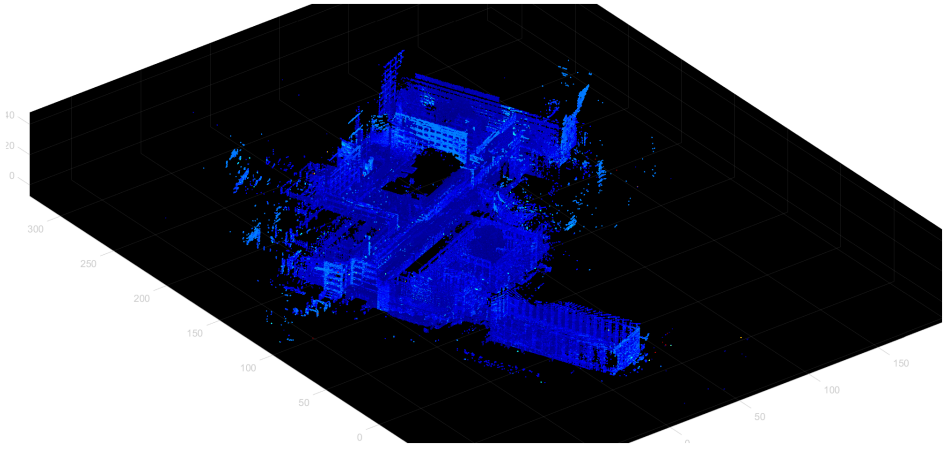


Figure 3.39: TGlobalMap output of LIO-SAM using walking dataset.

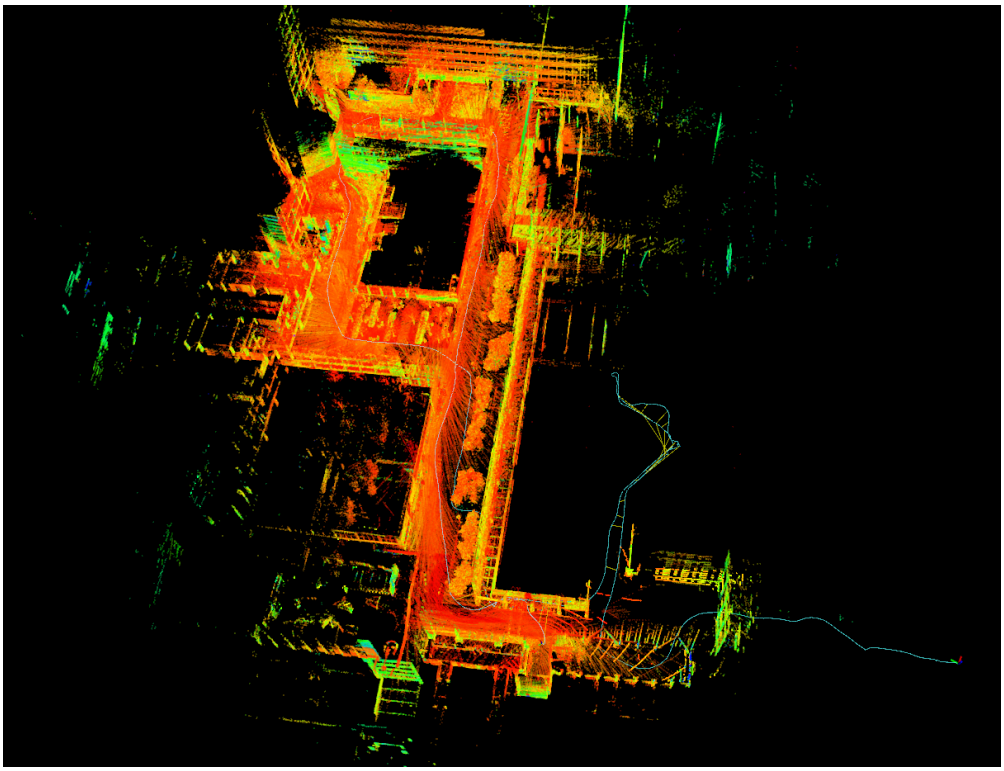


Figure 3.40: The resulting 3D map of LIO-SAM shows the loop closure detection.

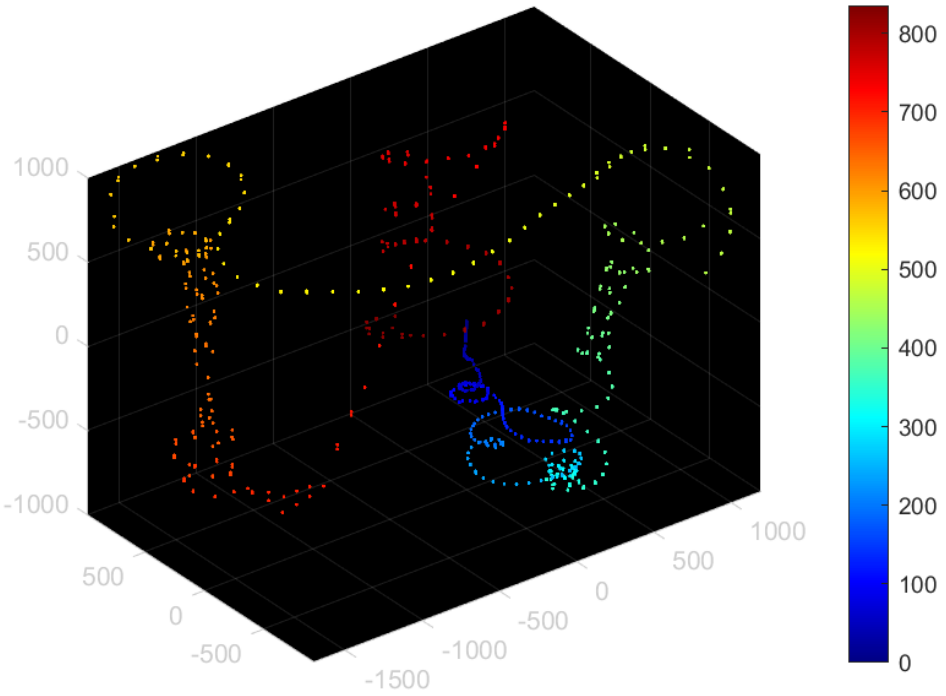


Figure 3.41: The full trajectory of drive dataset resulting by LIO-SAM algorithm.

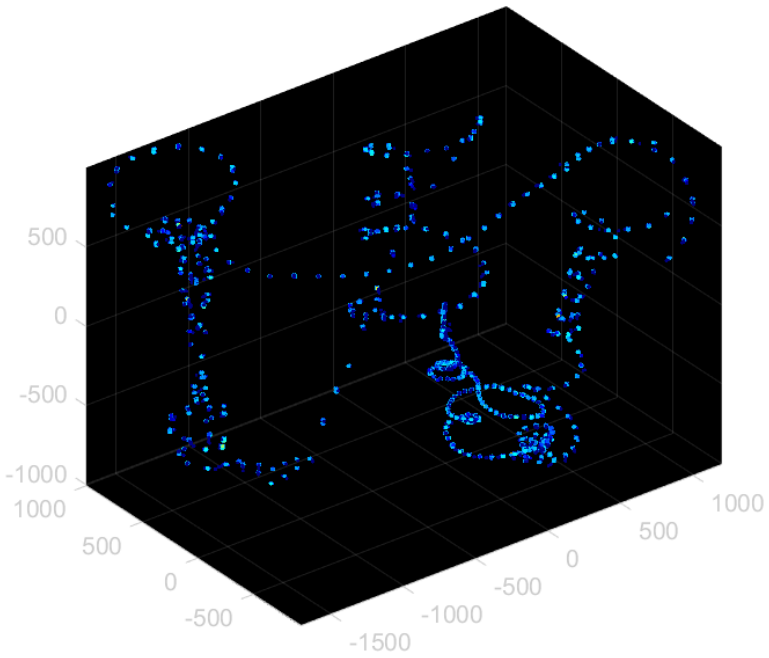


Figure 3.42: GlobalMap output of LIO-SAM using drive dataset.

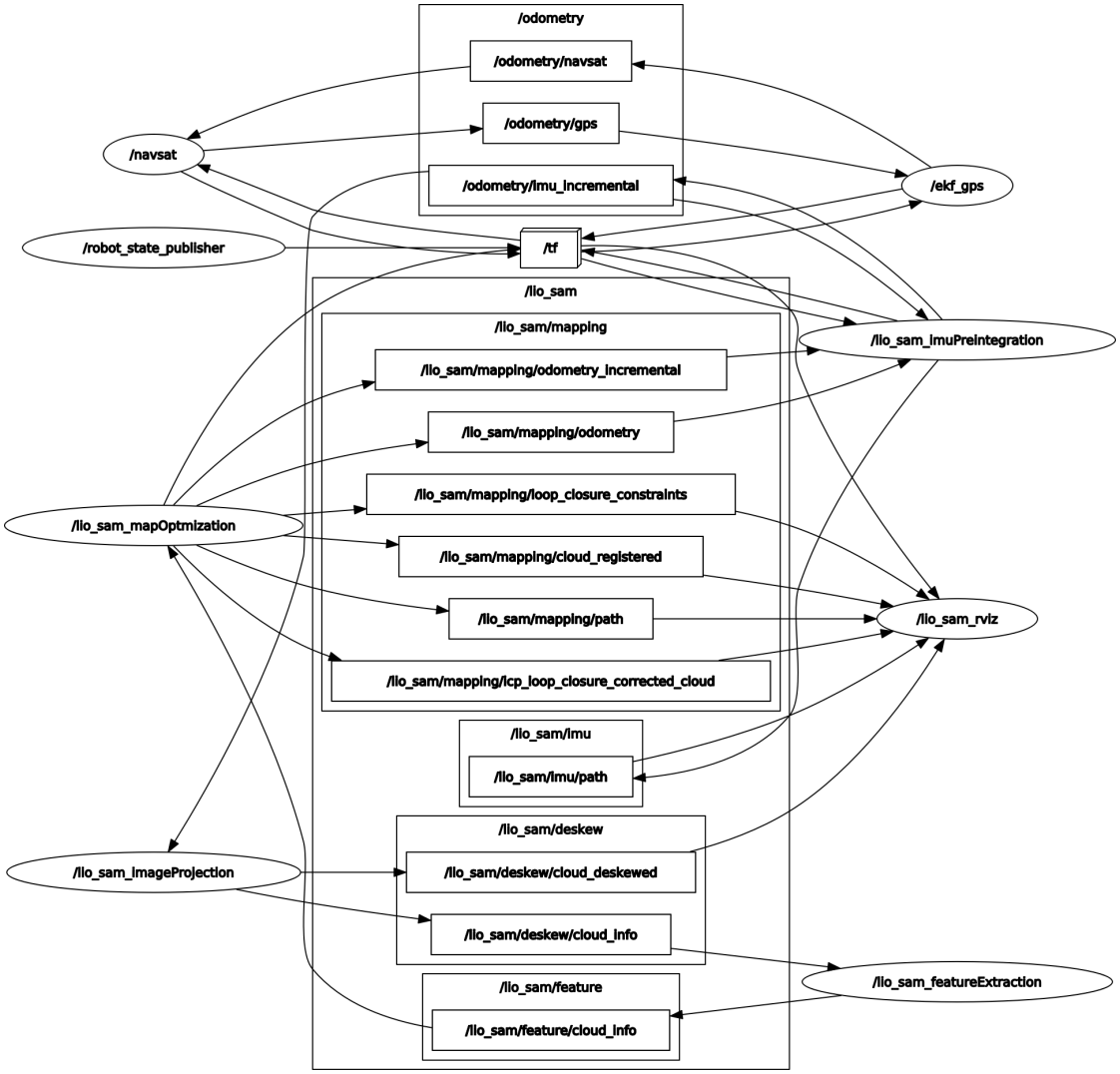


Figure 3.43: rqt-graph output diagram of LIO-SAM algorithm.

Chapter 4

General Conclusion

SLAM algorithms have received much attention due to their capability of constructing globally consistent 2D and 3D accurate maps in different frame works where the robot can also localize itself. Indeed, in robotics, there is a widely used set of algorithms that make a mobile robot perceive its surrounding and obtains accurate information for the navigation in real-time. However, this high accuracy comes at the cost of heavier algorithms.

In order to solve the SLAM problem in previously unknown environments, we use LiDAR as a mean of perception, a Pose Graph based SLAM algorithm in which we integrate in turn two loop closure detection methods: one based on the Scan-Context descriptor and the other is based on Scan-Matching to compare their effects on the obtained results.

Robot software platforms are extraordinarily useful in our simulation, we run TurleBot3-Waffle in different environments, which make it possible to model as much as little reality as we desire; via two powerful 3D simulators implemented in ROS (Gazebo and RViz).

In this manuscript, we aimed to prove the accuracy and robustness of some recent SLAM approaches and place recognition algorithms, including 3D GraphSLAM using Scan-Context or Scan-Matching for loop closure detection and LIO-SAM. We investigate some recent SLAM implementations (both in ROS and in Matlab 2021b) commonly evaluated in indoor/outdoor environments. In addition, we assessed 2D SLAM through 'GMapping' using teleoperation node and rosbag files as our own offline-data

(recorded using based-map navigation in ROS).

Scan-Matching is considered as a classical tool in the loop closure detection process; it is applied side by side with different methods, including NDT and ICP algorithm; and associated with multiple optimization approaches (PGO), and filtering techniques (EKF and PF). However, the Scan-Context technique is recent and based on a descriptor calculated for each scan data. It gives better results for resolving the drift in the robot trajectory.

Based on SLAM simulations, outcomes of small-scale 2D/3D environments and 3D large-scale environment through different datasets; we conclude that the results performance differs according to:

1. The plurality of the sources of measurements;
2. The pre-processing of the data.
3. The efficiency of the loop closure detection method.
4. The nature of the optimization.

The efficiency of the SLAM implementation is also measured according to its purpose, whether it is intended for real-time navigation or environment reconstruction for archival storage.

The Scan-Context method is evaluated on two complex outdoor scenes captured with the experimental vehicle Annie Way. The data is available as an online-dataset that depicts two trajectories. Mention that there is no jump in the position estimate and that loops close nicely which shows the quality and robustness of this method.

In order to quantitatively evaluate the accuracy of 3D SLAM obtained from the LIO-SAM algorithm, we used the characteristics of picture picked from Google Earth to generate ground-truth for the end accurate map.

Although widely accepted, in some cases the ROS suffers from some serious issues, because it has some limitations which is the reason why they can be expected to work accurately in ROS II as future work. In a future work, we aim to combine LIO-SAM algorithm side by side with Scan-Context global descriptor for place recognition.

Appendix A

we use a specific command to do this. Map server is a package that is responsible for publishing and manipulating maps. This package has a node called the map saver, which saves the map constructed by the same process into a file to be able to use it later for navigation.

The registration command generates two files, one image file with .pgm extension and another file with .yaml extension. The .pgm file contains the map image itself. The .yaml file contains the metadata about the map, including the location of the image file of the map, its resolution, its origin, the occupied cells threshold and the free threshold. The details for the different parameters shown in fig.A.1, the image parameter represents the path to the image file containing the occupancy data. It can be absolute or relative to the location of the .pgm file.

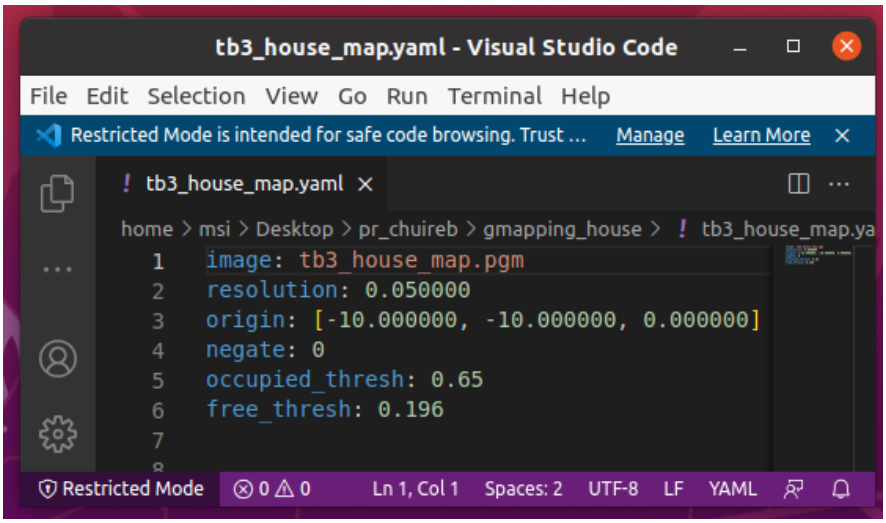


Figure A.1: .yaml file output of gmapping algorithm of house environment.

While the image file has a pgm format which contains numeric values between zero

and 255, the resolution, it is the resolution of the map and is expressed in meters per pixel, which means, in our results, 5 centimeters for each cell, meaning for each pixel. The origin, it is the two-dimensional pose of the lower left pixel in the image as x, y, yaw with yaw as a contour of required rotation. Note that:

While the image file has a pgm format which contains numeric values between zero and 255, the resolution, it is the resolution of the map and is expressed in meters per pixel, which means, in our results, 5 centimeters for each cell, meaning for each pixel. The origin, it is the two-dimensional pose of the lower left pixel in the image as x, y, yaw with yaw as a contour of required rotation. Note that:

- **Occupied threshold:** means any pixel value greater than 0.65 of max color value, it is considered as occupied. This is a way to convert the grayscale image into a binary image.

- **Negate:** if equal to 1, any pixel that is black become white and every white pixel becomes black.

Now for the second file, that is the .pgm image file, it is just a grayscale image of the map which you can open using any image viewer program.

ROS offers a scheme that describes all the nodes and the packages that are connected and actives by rqt-graph command. In the case of gmapping node the rqt-graph is shown in fig. A.2.

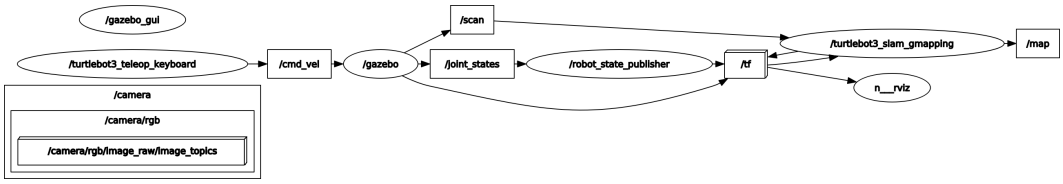


Figure A.2: rqt-graph of GMapping algorithm.

As a result of fig. A.2, we can see that the laser scan is generated by taking the point cloud from the 3D sensor and grabbing points from eye-level perspective of the robot. Thus, it does not use any camera data.

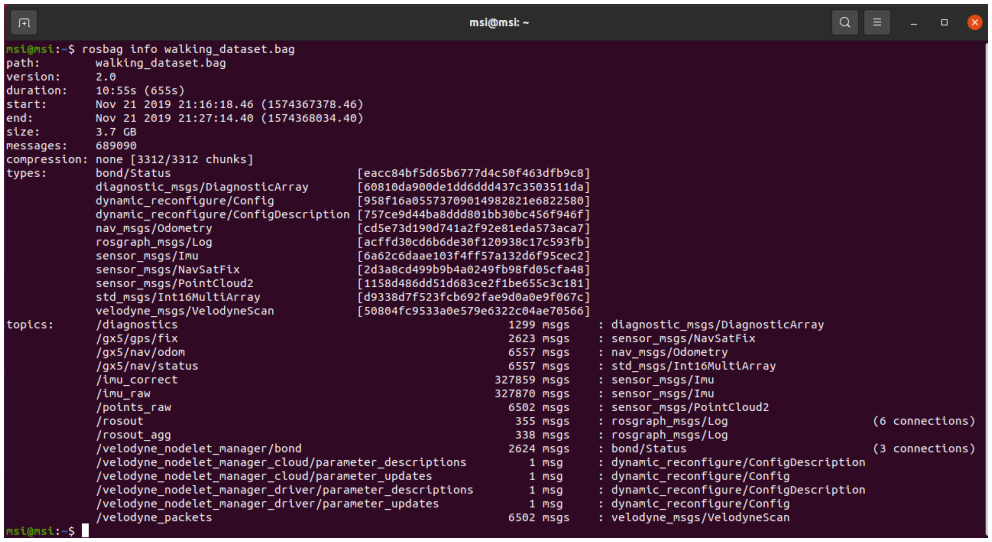


Figure A.3: ROSbag information of Walking dataset.

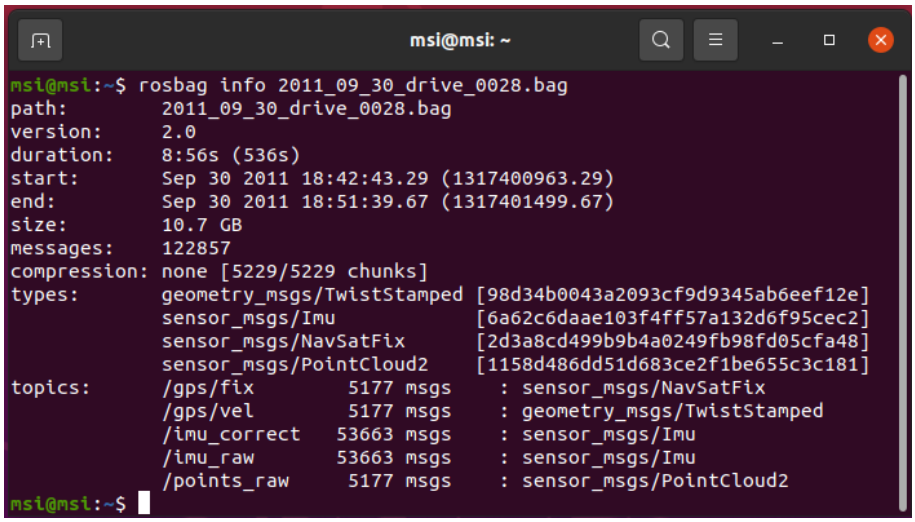


Figure A.4: ROSbag information of Drive dataset.

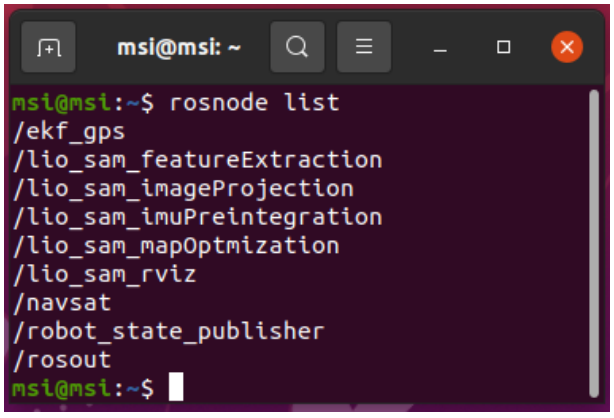


Figure A.5: ROS-node list of Lio-sam algorithm.

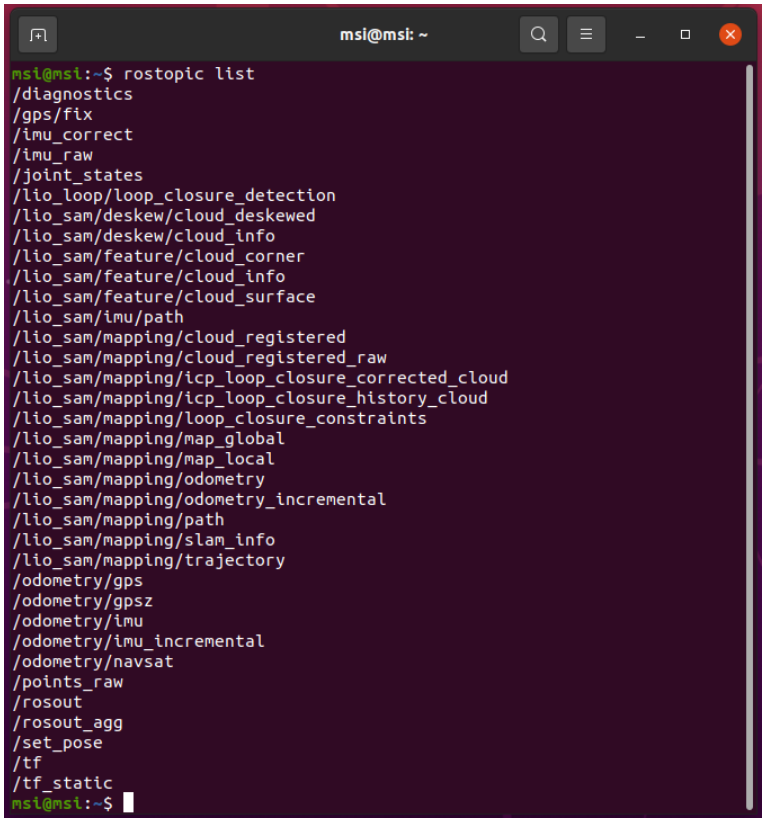


Figure A.6: ROStopic list of Lio-sam algorithm.

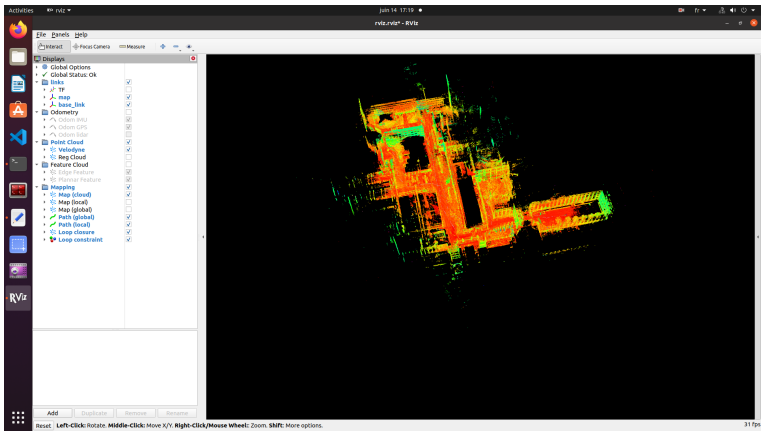


Figure A.7: The Global Map visualized by RViz in ROS of walking dataset.

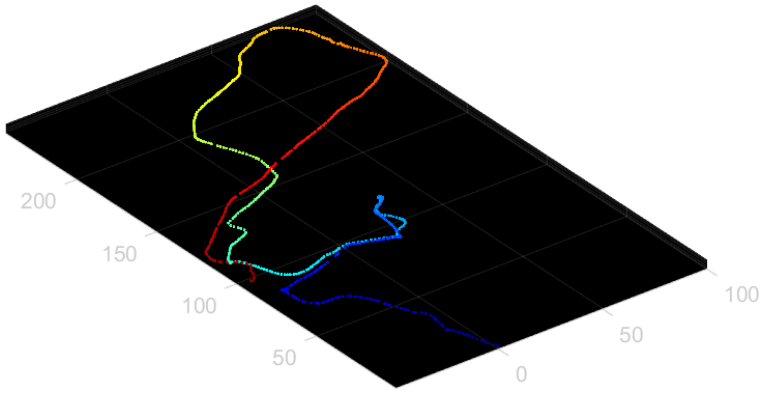


Figure A.8: Transformations of LIO-SAM using wolking dataset.

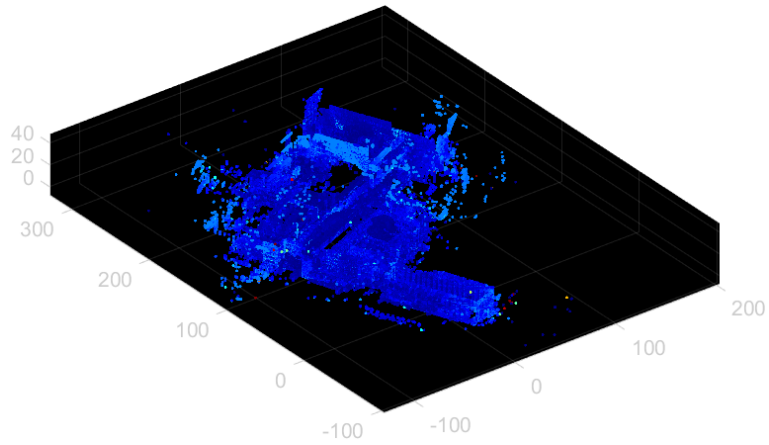


Figure A.9: SurfMap of LIO-SAM using wolking dataset.

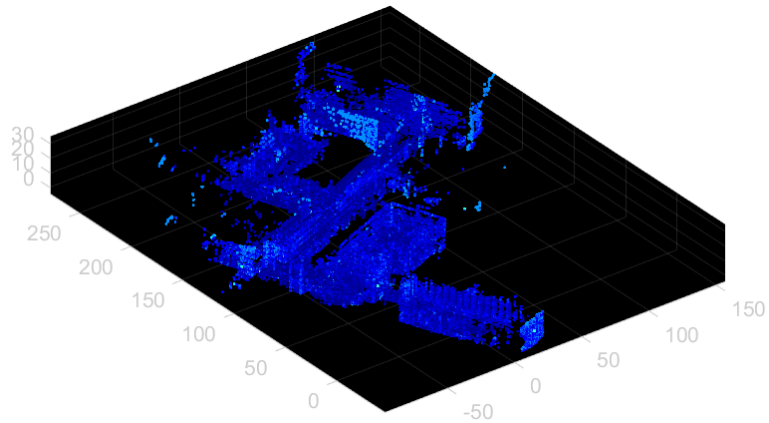


Figure A.10: CornerMap of LIO-SAM using wolking dataset.

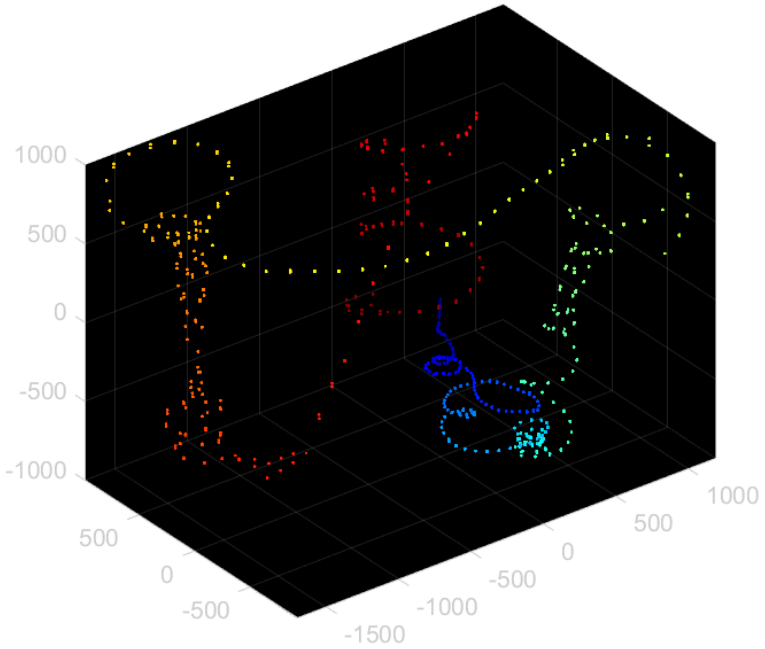


Figure A.11: Transformations of LIO-SAM using drive dataset.

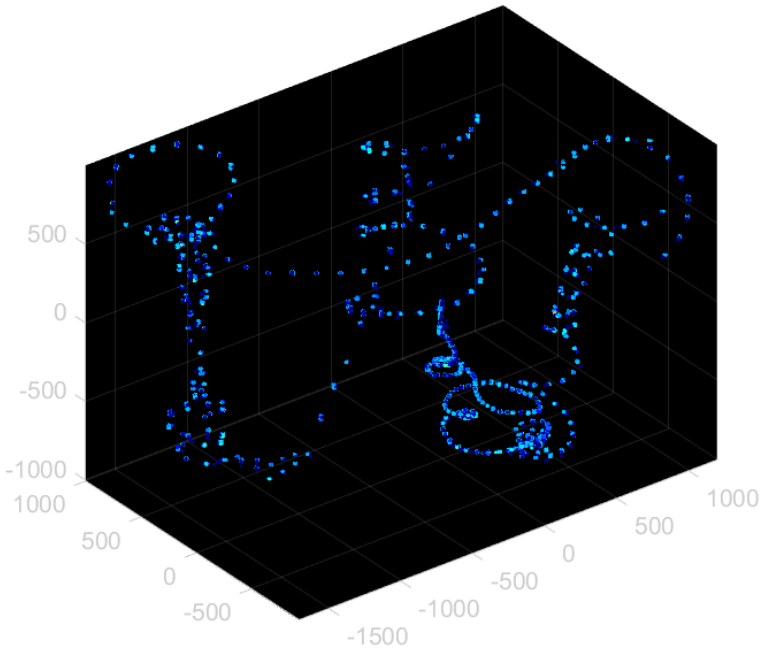


Figure A.12: SurfMap of LIO-SAM using drive dataset.

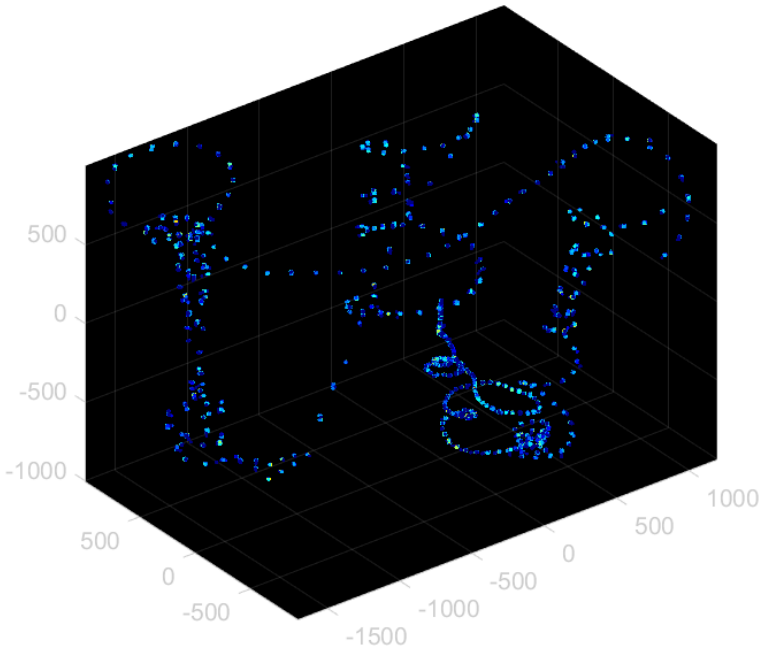


Figure A.13: CornerMap of LIO-SAM using drive dataset.

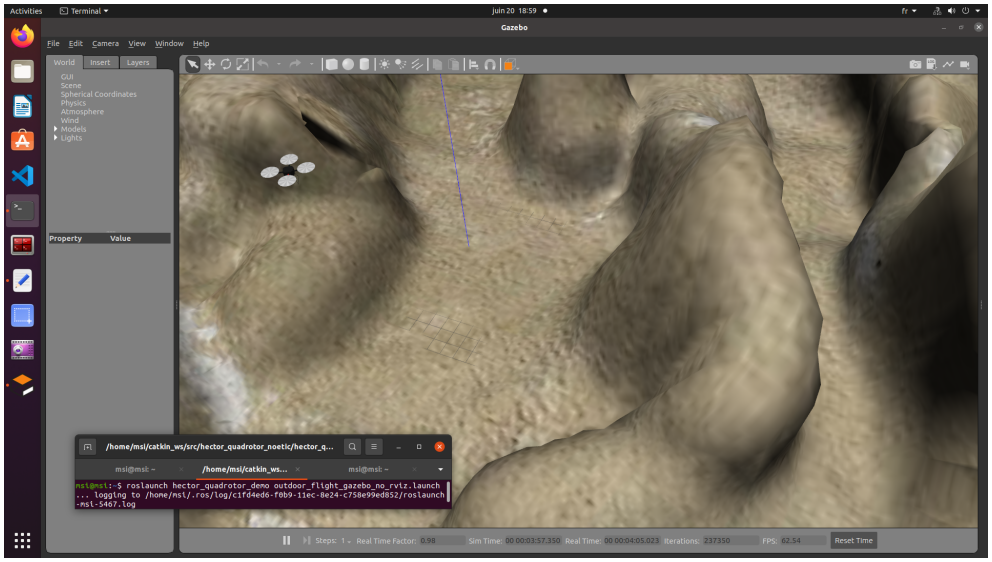


Figure A.14: Quadrotor in outdoor environment.

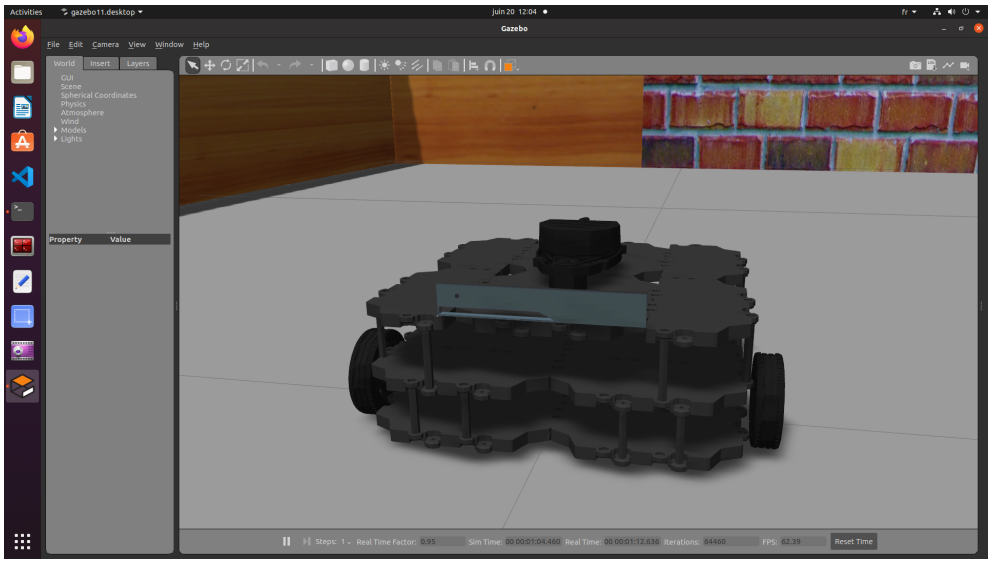


Figure A.15: Lauching the TurtleBot3-Waffle in Gazebo House.

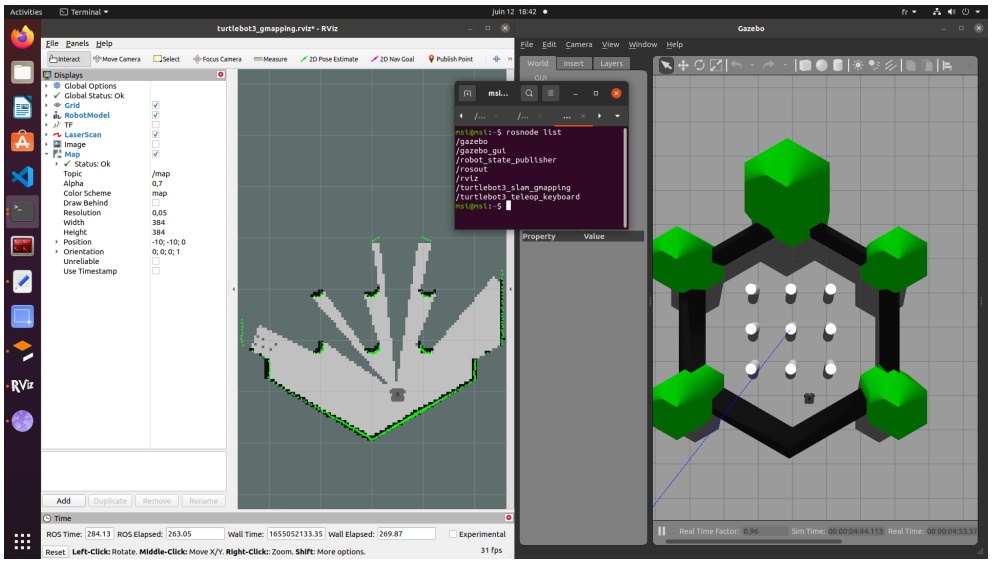


Figure A.16: RViz Launching the GMapping lamp, shows the ROS node list of GMapping.

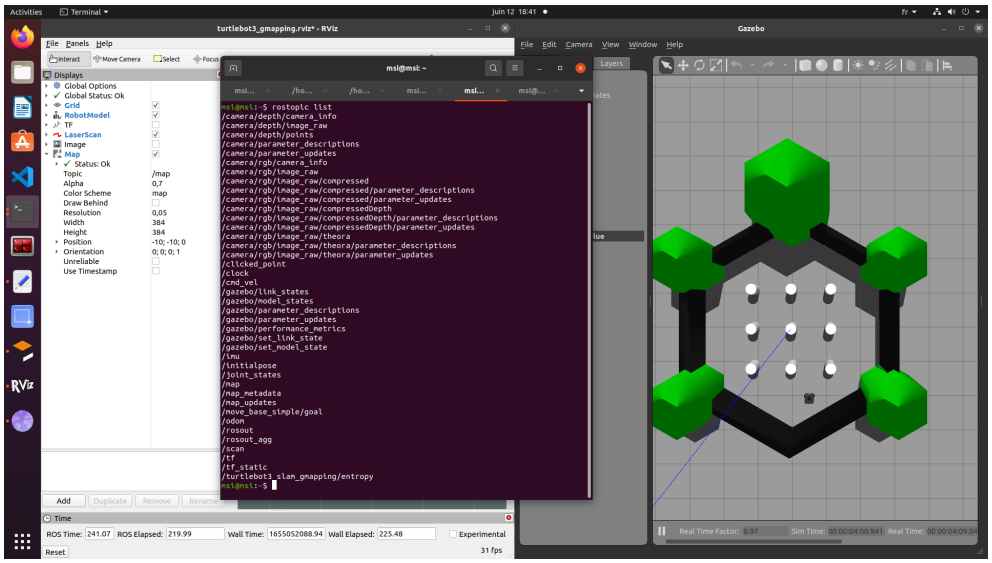


Figure A.17: ROS topic list of GMapping.

Bibliography

[1] Peter I Corke and Oussama Khatib. Robotics, vision and control: fundamental algorithms in MATLAB, volume 73. Springer, 2011.

[2] Emilio Garcia-Fidalgo and Alberto Ortiz. Methods for Appearance-based Loop Closure Detection. Springer, 2018.

[3] Hugh Durrant-Whyte and Tim Bailey. Simultaneous localization and mapping: part i. IEEE robotics & automation magazine, 13(2):99–110, 2006.

[4] Ketty Favre. LiDAR-based point clouds registration for localization in indoor environments. PhD thesis, Université Rennes 1, 2021.

[5] Jingchun Yin, Luca Carlone, Stefano Rosa, Muhammad Latif Anjum, and Basilio Bona. Scan matching for graph slam in indoor dynamic scenarios. In The Twenty-Seventh International Flairs Conference, 2014.

[6] Ryuki Suzuki, Ryosuke Kataoka, Yonghoon Ji, Kazunori Umeda, Hiromitsu Fujii, and Hitoshi Kono. Slam using icp and graph optimization considering physical properties of environment. In 2020 21st International Conference on Research and Education in Mechatronics (REM), pages 1–5. IEEE, 2020.

[7] Giseop Kim and Ayoung Kim. Scan context: Egocentric spatial descriptor for place recognition within 3d point cloud map. In 2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pages 4802–4809. IEEE, 2018.

[8] David Scaradozzi, Silvia Zingaretti, and AJSC Ferrari. Simultaneous localization and mapping (slam) robotics techniques: a possible application in surgery. Shanghai Chest, 2(1), 2018.

[9] Sebastian Thrun and Michael Montemerlo. The graph slam algorithm with applications to large-scale mapping of urban structures. The International Journal of Robotics Research, 25(5-6):403–429, 2006.

[10] Fanta Camara, Chris Waltham, Grey Churchill, and Charles Fox. Openpodcar: an open source vehicle for self-driving car research. arXiv preprint arXiv:2205.04454, 2022.

[11] Sheng Wang, Guohua Gou, Haigang Sui, Yufeng Zhou, Hao Zhang, and Jiajie Li. Cdsfusion: Dense semantic slam for indoor environment using cpu computing. Remote Sensing, 14(4):979, 2022.

[12] P Cheeseman, R Smith, and M Self. A stochastic map for uncertain spatial relationships. In 4th international symposium on robotic research, pages 467–474. MIT Press Cambridge, 1987.

[13] Yukinori Kobayashi Jixin LV. Scan Matching and SLAM for Mobile Robot in Indoor Environment. PhD thesis, Division of Human Mechanical Systems and Design Graduate School of Engineering, 2016.

[14] HH Folmer and RN Appel. Analysis, optimization, and design of a slam solution for an implementation on reconfigurable hardware (fpga) using cλash. Master's thesis, University of Twente, 2016.

[15] Ayoade Femi Olalekan, Jane Alam Sagor, Md Hasibul Hasan, and Adekunle Samuel Oluwatobi. Comparison of two slam algorithms provided by ros (robot operating system). In 2021 2nd International Conference for Emerging Technology (INCET), pages 1–5. IEEE, 2021.

[16] Han Wang, Chen Wang, and Lihua Xie. Intensity scan context: Coding intensity and geometry relations for loop closure detection. In 2020 IEEE International Conference on Robotics and Automation (ICRA), pages 2095–2101. IEEE, 2020.

[17] Naveed Muhammad and Simon Lacroix. Loop closure detection using small-sized signatures from 3d lidar data. In 2011 IEEE International Symposium on Safety, Security, and Rescue Robotics, pages 333–338. IEEE, 2011.

[18] Lukáš Jelínek. Graph-based slam on normal distributions transform occupancy map. 2016.

[19] Tixiao Shan and Brendan Englot. Lego-loam: Lightweight and ground-optimized lidar odometry and mapping on variable terrain. In 2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pages 4758–4765. IEEE, 2018.

[20] Renaud Dubé, Daniel Dugas, Elena Stumm, Juan Nieto, Roland Siegwart, and Cesar Cadena. Segmatch: Segment based place recognition in 3d point clouds. In 2017 IEEE International Conference on Robotics and Automation (ICRA), pages 5266–5272. IEEE, 2017.

[21] Radu Bogdan Rusu, Nico Blodow, Zoltan Csaba Marton, and Michael Beetz. Aligning point cloud views using persistent feature histograms. In 2008 IEEE/RSJ international conference on intelligent robots and systems, pages 3384–3391. IEEE, 2008.

[22] Samuele Salti, Federico Tombari, and Luigi Di Stefano. Shot: Unique signatures of histograms for surface and texture description. Computer Vision and Image Understanding, 125:251–264, 2014.

[23] Serge Belongie, Jitendra Malik, and Jan Puzicha. Shape matching and object recognition using shape contexts. IEEE transactions on pattern analysis and machine intelligence, 24(4):509–522, 2002.

[24] Andrew E Johnson and Martial Hebert. Using spin images for efficient object recognition in cluttered 3d scenes. IEEE Transactions on pattern analysis and machine intelligence, 21(5):433–449, 1999.

[25] Bastian Steder, Michael Ruhnke, Slawomir Grzonka, and Wolfram Burgard. Place recognition in 3d scans using a combination of bag of words and point feature

based relative pose estimation. In 2011 IEEE/RSJ International Conference on Intelligent Robots and Systems, pages 1249–1255. IEEE, 2011.

[26] Hao Zhang, Fei Han, and Hua Wang. Robust multimodal sequence-based loop closure detection via structured sparsity. In Robotics: Science and systems, 2016.

[27] Stanford Artificial Intelligence Laboratory et al. Robotic operating system.

[28] Hancheol ChoJJJJJJJ Yoonseok Pyo. ROS Robot Programming (English). ROBOTIS, 12 2017.

[29] Tixiao Shan, Brendan Englot, Drew Meyers, Wei Wang, Carlo Ratti, and Daniela Rus. Lio-sam: Tightly-coupled lidar inertial odometry via smoothing and mapping. In 2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pages 5135–5142. IEEE, 2020.

[30] Tixiao Shan, Brendan Englot, Drew Meyers, Wei Wang, Carlo Ratti, and Rus Daniela. Lio-sam: Tightly-coupled lidar inertial odometry via smoothing and mapping. In IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pages 5135–5142. IEEE, 2020.