UNIVERSIDADE D
COIMBRA

João Carlos Abrantes Jorge

3D Graph-Based SLAM Benchmark in Structured Environments and Loop Closure Detection

Dissertação no âmbito do Mestrado em Engenharia Eletrotécnica e de Computadores no ramo de Controlo, Sistemas Ciberfísicos e Inteligência Artificial
Orientada pelo Professor Doutor Cristiano Premebida
Apresentada no Departamento de Engenharia Eletrotécnica e de Computadores da Faculdade de Ciências e Tecnologia da Universidade de Coimbra.

Setembro de 2023

UNIVERSIDADE D
COIMBRA

FCTUC FACULDADE DE CIÊNCIAS
E TECNOLOGIA
UNIVERSIDADE DE COIMBRA

3D Graph-Based SLAM Benchmark in Structured Environments and Loop Closure Detection

João Carlos Abrantes Jorge

Coimbra, September 2023

3D Graph-Based SLAM Benchmark in Structured Environments and Loop Closure Detection

Supervisor:

Professor Doutor Cristiano Premebida

Jury:

Prof. Dr Cristiano Premebida

Prof. Dr. Jorge Manuel Moreira de Campos Pereira Batista

Prof. Dr. João Pedro de Almeida Barreto

Dissertation submitted in partial fulfillment for the degree of Master of Science in Electrical and Computer Engineering.

Coimbra, September 2023

Acknowledgements

Em primeiro lugar queria agradecer aos meus orientadores que foram excepcionais e mostraram-se disponíveis durante todo o desenvolvimento do trabalho, o professor Cristiano Premebida e o mestre Iago Gomes sempre estiveram lá para mim e dedicaram o seu tempo para este trabalho pudesse ser concluído, sem ajuda deles a realização deste trabalho não seria possível, e por isso estou eternamente agradecido. Destaco também a importância da dinâmica de trabalho no laboratório de mecatrónica, onde doutorandos, nomeadamente, Tiago Barros e Luís Garrote receberam-me de forma muito simpática e sempre estiveram abertos para me dar sugestões e soluções para os problemas que me foram surgindo, mostrando-se cooperativos e sempre disponíveis.

Em segundo lugar, gostaria de expressar um agradecimento especial aos meus amigos da faculdade. A união entre nós fez toda a diferença, e não tenho dúvidas de que ter um grupo de estudo consistente, onde nos motivávamos mutuamente, desempenhou um papel fundamental em todo o meu percurso académico. O curso não foi uma jornada solitária, e a ajuda deles foi essencial para a minha formação. Hoje, posso afirmar que não carrego apenas boas lembranças dessa época, mas também amizades que levo para toda a vida.

Queria agradecer aos meus pais que diariamente se esforçam para me sustentar e me proporcionaram uma vida recheada de oportunidades e sucesso, o apoio deles foi fundamental nesta fase da minha vida.

Por fim, queria também agradecer aos meus amigos mais próximos, que já me conhecem muito antes de integrar a faculdade, eles sabem que posso contar com eles para o que der e vier e este sentimento é mútuo.

Resumo

Parece evidente que os veículos autónomos irão trazer inúmeras vantagens, seja a nível de segurança, eficiência, como também permitir que pessoas com mobilidade reduzida possam circular, mesmo que não consigam conduzir. Há alguns anos que esta área tem vindo gradualmente a aumentar a atenção de pessoas por todo o mundo e não há dúvidas que este é um dos caminhos a seguir. Para que veículos autónomos naveguem em ambientes desconhecidos, a existência de algoritmos de SLAM robustos, que consigam operar em todo o tipo de condições adversas é essencial. Neste sentido, este trabalho tem em vista focar-se em estudar e compreender de forma clara sistemas de SLAM baseados em LIDAR, identificando quais as suas limitações e as suas vantagens. São estudados quatro algoritmos de SLAM (Cartographer, HDL-Graph SLAM, LIO-SAM e SC LeGO LOAM) e testados em dados adquiridas em ambientes reais. Numa primeira avaliação, consideraram-se sequências com loops diretos. De seguida, tendo em conta os resultados alcançados na primeira avaliação, foram selecionados dois algoritmos com melhores resultados de trajetória e então foram testados em ambientes que contivessem passagens pelos mesmo lugares, vistos de uma perspetiva complementar para avaliar a robustez do algoritmo de loop closure. Os resultados demonstraram que em termos de local SLAM o melhor algoritmo foi o LIO-SAM (embora o Cartographer tenha tido bons resultados de trajetória precisava de muitos ajustes nos parâmetros), apesar de ter apresentado uma pequena limitação quando existem altos na estrada. Ambos os algoritmos SC LIO-SAM e SC LeGO LOAM identificaram com sucesso os locais revisitados, devido a possuir um algoritmo de *place recognition* robusto e que apresentou bons resultados na secção de *loop closure*. O algoritmo com desempenho inferior foi claramente o HDL-Graph SLAM uma vez que não continha algoritmo de global map optimization que compensasse os erros obtidos no processo de local SLAM. Demonstrou-se com evidências experimentais a influência do loop closure na correção da trajetória e do mapa obtido.

Palavras-Chave: Localização e Mapeamento Simultâneos; Detecção de Loops; Mapeamento 3D; Optimização de Posição por Grafos; Veículos Autónomos

Abstract

It seems evident that autonomous vehicles will bring numerous advantages, including improved safety, efficiency, and increased mobility for individuals with reduced mobility who may not be able to drive. Over the past few years, this field has gradually gained the attention of people worldwide, and it seems clear that the overall adoption of automated vehicles (including robots) will be one of the paths to follow. To enable autonomous vehicles to navigate in unknown environments, the existence of robust SLAM (Simultaneous Localization and Mapping) algorithms capable of operating under various adverse conditions is essential. In this context, this work aims to focus on studying and gaining a clear understanding of LIDAR-based SLAM systems and identifying their limitations and advantages.

Four SLAM algorithms (Cartographer, HDL-Graph SLAM, LIO-SAM, and SC LeGO LOAM) are discussed and evaluated on data acquired in real-world scenarios. The first evaluation has considered sequences with direct loops. Then, two algorithms with the best trajectory results were selected and evaluated in environments that included revisits to the same locations from a different perspective to evaluate the robustness of the loop closure algorithm. The results demonstrated that, in terms of local SLAM, the best-performing algorithm was LIO-SAM (although Cartographer achieved good trajectory results, it required significant parameter tuning), although it showed a slight limitation when encountering road bumps. Both SC LIO-SAM and SC LeGO LOAM successfully identified revisited locations due to their robust place recognition algorithm, which yielded good results in the loop closure section.

The worst-performing algorithm was clearly HDL-Graph SLAM, which lacked a global map optimization algorithm to compensate for errors in the local SLAM process. The influence of loop closure on trajectory and map correction can be seen on the reported results.

Keywords: Simultaneous Localization and Mapping; Loop Closure Detection; 3D Mapping; Pose-graph Optimization; Autonomous Vehicles

"What we know is a drop, what we don't know is an ocean."

— Isaac Newton

Contents

List of Acronyms

AV	Autonomous Vehicle
GPS	Global Positioning System
RTK	Real Time Kinematic
IMU	Inertial Measurement Unit
SLAM	Simultaneous Localization and Mapping
LIDAR	Light Detection and Ranging
SC	Scan Context
LeGO	Lightweight and Ground Optimized
LOAM	Lidar Odometry and Mapping
LIO SAM	Lidar-inertial Odometry via Smoothing and Mapping
GTSAM	Georgia Tech Smoothing and Mapping
L-M	Levenberg-Marquardt
NDT	Normal Distribution Transform
GICP	Generalized Iterative Closest Point
ICP	Iterative Closest Point
UC	University of Coimbra

List of Figures

List of Tables

1 Introduction

1.1 Autonomous Vehicles

Over the last decades, intelligent and autonomous vehicles (AV), including autonomous robots, have become more integrated into our daily routine. By definition, an autonomous car [8] is a vehicle that can operate without human intervention, being able to make decisions by itself and navigate through an environment. Mobility and artificial intelligence (AI) improve the functioning capabilities of self-driving cars and robots, leading to the exploration of various research domains, including path planning, self-localization, mapping, exploration, coverage, team member coordination, collaboration, perception and SLAM.

In the past, mobile robots [48] were limited to performing repetitive or predetermined tasks, but as technology has advanced, this view has changed. Nowadays, due to AI's progress, the growth in computational resources, and the evolution of sensors, robots and AV are becoming more "intelligent", having the ability to sense the environment and make decisions with a high degree of autonomy. In this thesis, mobile robots and autonomous vehicles will often be referred to interchangeably in the sense that they can be understood as sophisticated machines that can operate in real-world environments and can be smart enough to avoid obstacles in their path and have the ability to travel freely without being constrained.

Autonomous vehicles pretend to revolutionize the future, offering improved transportation safety and efficiency [58]. They can be helpful for people with reduced mobility, such as the elderly and those with disabilities, reduce traffic congestion, and optimize energy efficiency. Recently, autonomous vehicles have been the subject of many research areas, and their evolution has been triggered by international projects such as the DARPA challenge [55], European Truck Platooning [1] and No Hands Across America [8], which aim at promoting the development of AVs [58].

The main objective of these projects was to create a fully autonomous and robust car capable of driving itself without human assistance, and that was reliable and safe enough to be deployed into real-world environments without compromising human lives in the process. All of them showed considerable contribution in this area, although several limitations had been identified, such as the robot's perception of the surrounding environment, failure to detect road marks and/or traffic

light signals, and the difficulty of operating autonomously. Many of these problems were due to the software/hardware available at the time they were carried out and external factors such as unfortunate weather conditions (fog, rain). Even so, with the limitations and problems that were encountered when carrying out the project, the progress was remarkable, having identified many solutions and/or alternatives that allowed researchers to overcome some of the limitations that were found.

Before going into more detail about AVs, it is necessary to understand the architecture of these autonomous systems. According to Nakhaeinia et al. [41], to perform autonomous navigation, an AV generally goes through 3 stages: (i) perception, (ii) planning and interpretation and (iii) movement. In the first phase of the process, the exteroceptive sensors (LiDAR, sonar or camera) provide all the information it needs to understand what is around it. Next, after all the data is collected, the vehicle has to be able to process and interpret all the information and be able to plan a route/path based on what it has perceived through the sensors, taking into account all the obstacles in its surroundings and its target point. Finally, after deciding which path to follow, control commands are sent to the motors, allowing the robot to move without colliding with obstacles.

1.2 Context and Motivation

For a robotic vehicle to be able to navigate without an *a-priori* map of the place where it is inserted, two important factors are necessary: (i) location in operational space; and, (ii) having a sensory-based representation of the space where it is. Thus, in this way, the robot knows its location and can interpret the data acquired through exteroceptive and proprioceptive sensors. The field of robotics that is dedicated to autonomous navigation has already been a predominant topic since the beginning of the 20th century, at the time when techniques for estimating the robot's position and building maps began to emerge. However, the problem was in performing both tasks simultaneously. To perform autonomous navigation required a representation of the space over which the robot was moving, which is a difficult problem to address given the computational power of the time and the complexity of the problem at hand. To understand the importance of Simultaneous Localization and Mapping (SLAM), it is enough to know that before its existence, it was only possible to perform autonomous driving deliberately if the robot had a map *a-priori* and knew its initial position in the map.

Frese [18] formulates that SLAM is an estimation theoretical problem, which encompasses the estimation of the robot's current position and the construction of a map that characterizes the structure of the environment where the vehicle operates. To perform both tasks, the vehicle must be equipped with onboard sensors that allow it to perceive the outside world. Without a map, it is not only difficult to locate the vehicle, but also to interpret the information acquired by the sensors.

The absence of a map makes navigation impracticable, as it is not possible to deliberately plan a path/trajectory to an objective point. On the other hand, by building a model of the surrounding environment, the vehicle becomes able to locate itself through position estimation algorithms, such as scan matching/feature matching and/or odometry . By achieving good localization, a consistent and robust map can be achieved, where the relative measurements of the obstacles to the vehicle are consistent. Briefly, it is not possible to map without localizing, nor the other way around, both have to be done simultaneously, hence the SLAM concept.

Although SLAM has a wide range of advantages and has overcome some problems of autonomous navigation, it is necessary to understand that it still has some limitations and problems, among them, the susceptibility to measurement errors of the sensors, tire slippage (drift errors), and the weather conditions that affect the performance of the sensors. All this leads to the accumulation of errors, and SLAM seeks to minimize the estimation error. Another problem is the limitation of some sensors, which can not overcome occlusions created by walls and other objects, have range limits, and are sensitive to light/fog and/or rain. Therefore, for a robust system, it is necessary to merge different sensors to introduce greater robustness and redundancy in the vehicle.

Loop closure plays a crucial role in SLAM by offering several key advantages. Firstly, enables a robot to recognize a previously visited location, which allows for correcting and refining its global map. Secondly, loop closure is able to handle loop errors. When revisiting a place, discrepancies, or errors in the map may arise due to sensor noise, perception limitations, or environmental changes, but loop closure provides a mechanism to detect and rectify these errors, ensuring the map remains consistent and coherent. Another advantage to leverage from loop closure is that the robot is able to optimize its own trajectory estimation. By detecting loops, the robot can optimize its path and reduce accumulated errors that occurred over time in localization.

In summary, SLAM aims to construct a map while simultaneously estimating the position. However, SLAM faces many challenges, such as sensors measurements in adverse conditions, leading to error accumulation. To address these issues, loop closure represents a vital module to enable recognition of revisited places, enhancing map corrections and reducing errors. Loop closure serves as a mechanism to ensure consistency and coherence of the map. Furthermore, loop closure contributes to the optimization of the trajectory estimation.

1.3 Main Objectives

This thesis addresses some relevant issues in the application of autonomous driving vehicles, more particularly in the evaluation of 3D SLAM techniques and the influence of loop closure in SLAM.

The initial phase of this thesis involves exploring some 3D SLAM techniques, using readily available open-source datasets to evaluate their performance and understand their practical implications,

such as their strengths and weaknesses, and fine-tune their parameters for optimal performance. At a later stage, acquisition of a dataset of our own, there was a need to create our dataset and validate SLAM algorithms, even though open source datasets provided a valuable starting point, the process of establishing this dataset presented a significant learning experience, requiring careful consideration of sensor errors, sensors setup, and calibration, obtaining accurate ground truth data, etc. The subsequent stage of the project involves studying and exploring the impact of loop closure in SLAM. This task can be accomplished by evaluating different loop closure detection algorithms, ranging from geometrical detectors (*e.g.*, radius search) to feature-based methods employing hand-crafted features (*e.g.*, scan context [30]), or learned features (*e.g.*, ORCHnet [6] and OverlapNet [11]).

In summary, we evaluated and validated the SLAM using open-source datasets such as KITTI [19], allowing for a quicker assessment. Subsequently, we utilized our own dataset for further evaluation and validation. At a later stage, a system was developed in a car equipped with a 64-channel LiDAR, RTK GPS, and IMU to acquire a dataset and create a map that provides a detailed description of the UC-POLO2 campus of the University of Coimbra, at Coimbra, Portugal. Furthermore, in conjunction with our work on SLAM, we will also assess loop closure algorithms to evaluate the performance of each one individually and when fused with another. Ultimately, conclusions and future directions will be drawn to identify potential improvements and highlight the strengths and weaknesses of the work.

1.4 Document Overview

This master thesis will go as follows: Chapter 2 presents a list of related works that have already been implemented in this field, such as SLAM and loop detection algorithms. At the end, a topic summarizes and discusses the state of the art developed on the subject of matter. Chapter 3 introduces the requirements to develop and test the proposed approach, along with a detailed description of it, as well as an explanation of its functioning. After that, Chapter 4 presents an in-depth experimental evaluation of the proposed approach. This chapter aims to validate and assess the effectiveness of the chosen method by subjecting it to rigorous testing and comparison with existing techniques. Finally, in Chapter 5, an overview of the developed work and future work directions are outlined.

2 Background and Related Work

2.1 Simultaneous localization and mapping

2.1.1 Introduction to SLAM

SLAM seeks to build a map of the environment while simultaneously localizing the robot within it. According to Debeunne [14], SLAM can be divided into four distinct groups: Image-based, LiDAR-based, Radar-based, and a combination of Visual-Lidar, fusing visual and LiDAR data. Considering that a LiDAR will be used in the proposed framework, we will discard the image-based systems and focus on LiDAR-based approaches. One of the preliminary reasons for choosing LiDAR over Radar is its ability to create 3D maps with high accuracy. Thrun [54] states that SLAM is the problem of estimating landmarks in the environment (mapping) and at the same time determining the agent's position (localization). Based on Aulinas et al. perspective [5] there are three probabilistic methods to address the SLAM estimation problem: EKF-based [27], Particle filter-based [39, 53], and Graph-based [34]. This thesis considered SLAM algorithms that use graph-based probabilistic filters and work under the ROS middleware [46]. The choice of selecting graph-based SLAM is because this type of framework the computational cost increases linearly with the map size, whereas EKF and Particle filters the computations increase exponentially. Therefore, graph-based approaches are a good choice for large outdoor environments [13].

2.1.2 Stages of SLAM

Debeunne [14] states that SLAM can be divided into three stages: in the first stage, the robot's position is estimated based on odometry, and a local map is generated based on the reading of the sensors. In the second step, an alignment is performed between the local and global maps, to calculate the robot's position in the map coordinate system. In the third step, the robot's position undergoes optimization, and the global map is updated, incorporating the local map. In case the SLAM contains, after an excursion of arbitrary length, a previously visited area, there is still a fourth stage that seeks to optimize the map through loop closure detection. An example of a generic LiDAR-based SLAM architecture is present in figure 2.1.

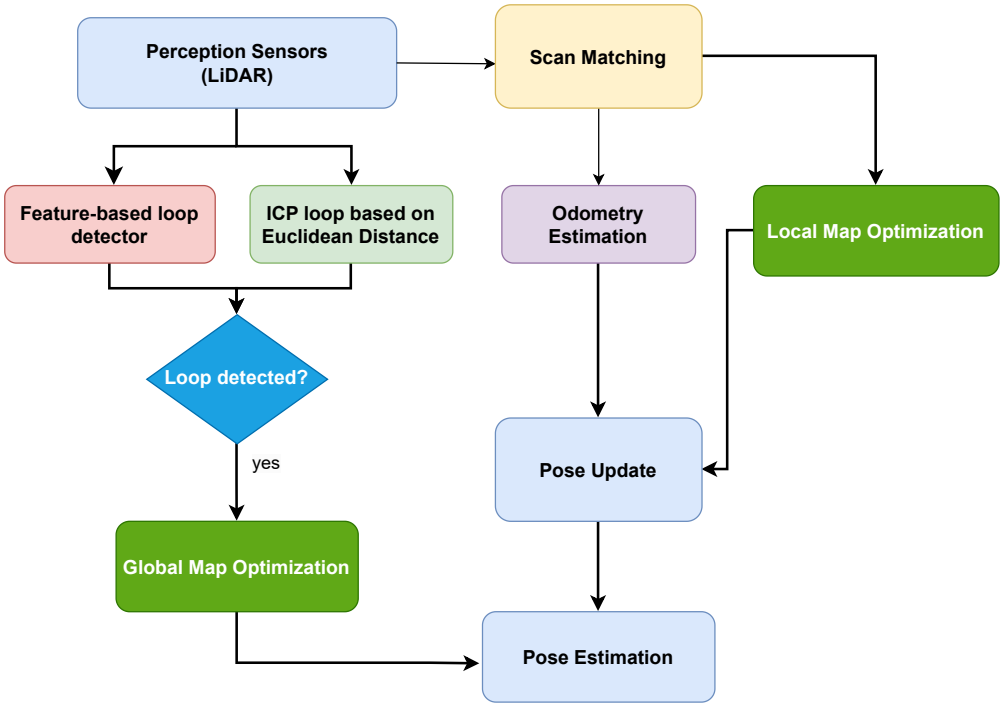


Figure 2.1: LiDAR-based SLAM diagram overview.

Graph-based probabilistic filter in SLAM

The graph-based SLAM approach, proposed by F. LU and E. Millios [34] in 1997, represents pose estimation as a pose-graph optimization problem. The poses obtained from the scans are considered a pose in the graph. The relative transformation between two scans is considered a constraint and represents an edge in the graph.

The network is composed of nodes and links, with each node denoting a pose within the map frame, while the links represent constraints. In this network, two distinct types of links are established. The first type of link emerges from the connection between two adjacent poses along the path, often referred to as the odometry link. The second link is established based on the level of overlap between two frames. When two frames exhibit a substantial overlap, a new constraint is introduced between them, effectively incorporating this overlapping information into the network. Network representation is present in figure 2.2 taken from [14].

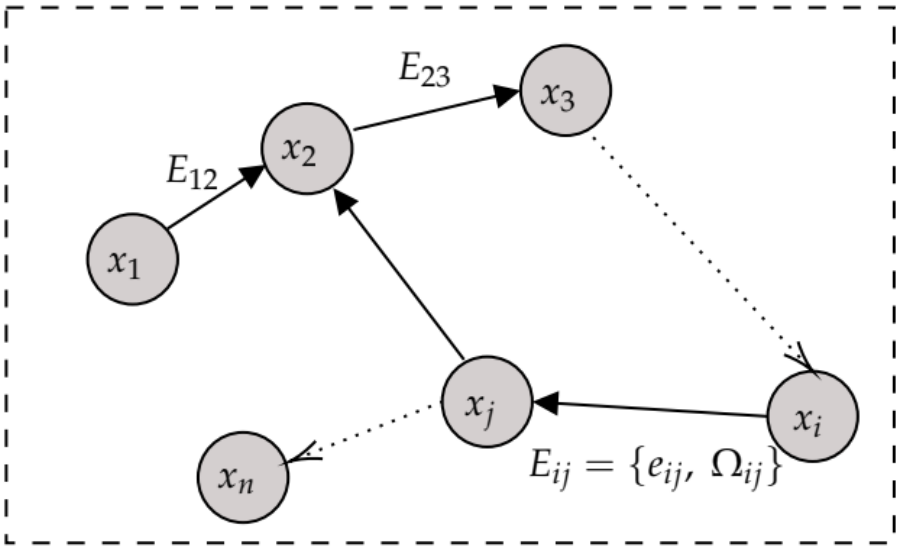


Figure 2.2: Representation of a pose-graph SLAM filter [14]. A node represents each pose x_i. Poses are connected by edges $E_{ij} = \{e_{ij}\}$. Edges contains spatial constraints that represent consecutive poses i and j withdrawn from sensor measurements (odometry links). Note that edges e_{ij} model the proprioceptive information (odometry constraints) and Ω_{ij} represent virtual measurements (overlap between non-consecutive frames).

The graph construction is performed during odometry estimation (front-end), whilst the graph optimization occurs in the mapping module (back-end). The mapping module is responsible for local and global optimization. Graph-based technique is present in most of the state-of-art algorithms and achieved interesting results when compared with other methods.

2.1.3 Pointcloud Registration

This section discusses 3D registration methods using LiDAR. Pointcloud registration is a crucial process in 3D mapping that consists of aligning pointclouds to create a unified representation of the environment. There are two types of pointcloud data registration: coarse-registration or fine-registration [12]. In fine-registration, the goal is to achieve maximum overlap between two pointclouds, using typically iterative methods (ICP) [7], normal distribution (NDT) [36] or random sample consensus method (RANSAC) [17]. They are accurate but not efficient computationally. ICP is a commonly 3D LiDAR registration method that performs matching between two pointclouds by minimizing the distance between corresponding points and find the optimal transformation. NDT, on the other hand, represents a pointcloud as a Gaussian distribution and aligns pointclouds by matching these distributions. The limitations of these two accurate techniques is the fact that they compute point-to-point scan matching and is not ideal for real-time as it takes large time-consuming. In order to reduce the number of points present in the registration, variants of ICP and NDT arose.

Matching algorithms like point-to-plane ICP, Fast-ICP revealed to have slighter better registration time, however the accuracy depended on a lot in the parameter configuration.

In coarse registration methods, rigid body transformations are obtained using feature-based techniques. Examples of feature-based coarse registration methods are the Point-based [62, 50, 61, 16, 25], Line-based [22, 59, 33], Surface-based [38, 20, 3, 4, 9] methods, and a combination of them [28, 47]. Feature-based LIDAR registration methods predominantly rely on feature points, but they encounter challenges related to sensor noise and limited geometric information. To address these limitations, line features were introduced because they offer stronger geometric constraints than points, resulting in improved registration accuracy. Surface features, on the other hand, contain even more information and are less susceptible to noise, making them crucial in structured environments, especially for ground structure representation. To achieve highly accurate registration, it is essential to extract surface features before the registration process, typically accomplished through point cloud segmentation.

In large-scale structured environments, researchers have explored combining point, line, and surface features. For instance, methods like SC LeGO LOAM, LIO-SAM, and LOAM identify keypoints based on geometric shapes, such as ground points (representing surfaces) and non-ground points (segmented points like corners and edges). These features are subsequently employed for registration purposes.

2.2 LiDAR-based SLAM frameworks

The present section addresses SLAM frameworks applied in the ROS middleware. There are dozens of SLAM implementations, however we have chosen four 3D LiDAR graph-based SLAM techniques after doing a review in state-of-art. The approaches discussed in the sequel have shown decent results in outdoor scenarios and contain all the criteria that we intended (sensors and graph-based SLAM).

2.2.1 LEGO-LOAM

LeGO-LOAM [51], presented in 2018 by T. Shan and B. Englot, is an optimized version of LOAM [68] (Lidar Odometry and Mapping) designed for 6D state estimation using a 3D LiDAR. This system targets small-scale embedded systems and addresses challenges related to real-time SLAM's smoothness and reliability on computationally constrained platforms. LeGO-LOAM's SLAM methodology comprises two key components: High-Frequency Odometry and Lower-Frequency Mapping. The high-frequency odometry component involves point cloud segmentation to remove outliers and extraction of corner and plane data points. These features are used to calculate the transformation between consecutive LiDAR scans, primarily focusing on feature matching against the previous

frame. The lower-frequency mapping component optimizes feature matching between the current frame and the surrounding point cloud, contributing to the creation of a detailed map. Recently, a novel algorithm called SC-Lego LOAM [65] was introduced, drawing inspiration from LeGO-LOAM and LOAM. It incorporates a loop closure detector called Scan Context (SC) to improve SLAM's performance in complex environments.

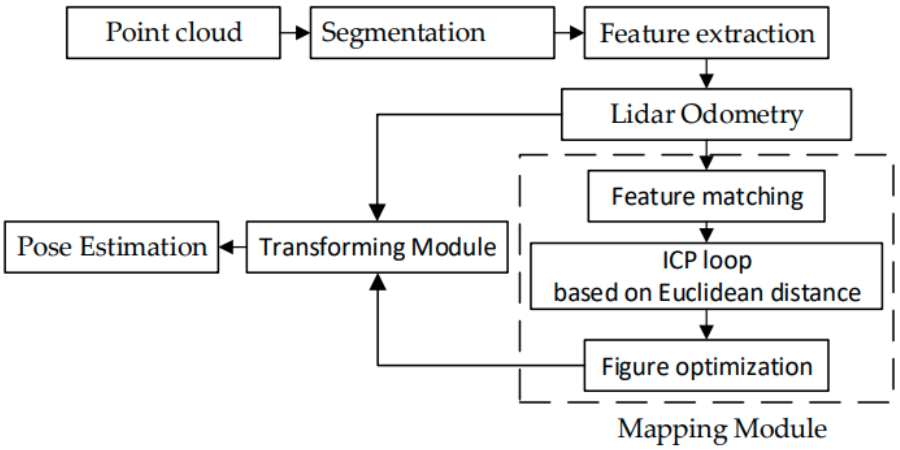


Figure 2.3: System structure diagram of the LeGO LOAM algorithm [51].

LeGO LOAM is composed of five main modules: pointcloud segmentation, feature extraction, LiDAR odometry, LiDAR mapping and transform fusion. The system architecture is shown in figure 2.3:

A. Pointcloud Segmentation

The *segmentation* module takes as an input a 3D pointcloud and projects it onto a range image [63]. A range image is a representation form of LiDAR pointcloud, where a 3D pointcloud is projected into a 2D image. The main reason for this is that a 2D image is capable of capturing the local geometry of the pointcloud. This intrinsic property allows obtaining neighboring points without having to construct a KD tree. There are immense algorithms for 2D projection, however, the most common are bird's eye view, cylindrical, and spherical projection. Range image belongs to spherical projection and is largely used for LiDAR pointcloud processing. Range image applications include object detection, object segmentation, motion prediction, semantic segmentation, LiDAR odometry.

Considering $P_t = \{p1, p2, \ldots, pn\}$ as the pointcloud acquired by LiDAR sensor at the time t, where p_i is a point in P_t and to each point is associated r_i, which represents the Euclidean distance from the correspondent point to LiDAR frame. The resolution of the range image depends on the LiDAR's properties. In Ouster LiDAR, the default horizontal resolution is 1024, but in Velodyne is 1800. The vertical resolution depends on the number of channels of the LiDAR used, for instance, there are LiDARs with 16, 32, 64, and 128 channels.

Then is applied an image-based segmentation method to the range image, so as to group points into clusters. The clusters are assigned a label: ground points or segmentation points. Points belonging to the same cluster receive the same label, and points marked as ground points do not enter the segmentation process. Utilizing pointcloud segmentation enhances processing efficiency and refines feature extraction precision. However, given the real-time operational nature of the target system, it is designed to ensure swift and dependable feature extraction from segmented pointcloud clusters. Consequently, features associated with clusters containing fewer than 30 points, which encompass items like small objects and tree leaves, are regarded as unreliable and subsequently excluded. At the same time, there is a selection of the clusters, the points inside the clusters are assigned three properties: its label, ground or segmented point, its column and row index in the range image and the distance value.

B. Feature extraction

The *feature extraction* module is responsible for extracting planar and edge features from ground points and segmented points. The procedure can be described in the following manner:

1. Consider A the aggregate of points belonging to the same row in the projected image (range image), roughness c of a p_i can be expressed by the formula below:

$$c = \frac{1}{|A| \cdot ||r_i||} \left\| \sum_{j \in A, j \neq i} (r_j - r_i) \right\| \tag{2.1}$$

   where, r_i and r_j are the range values from points p_i and p_j in relation to LiDAR, respectively.

2. Divide the range image horizontally, to obtain numerous equal sub-images, enabling the extraction of features uniformly.

3. The points are then sorted based on their roughness c. This process is carried through for each row of the sub image. The features are assigned as edge or planar features according to a certain roughness threshold c_{th} . Points with c larger than c_{th} are called edge features, and points with $c < c_{th}$ are segmented into planar features.

4. After step 3, $n\mathbb{F}_e$ edge feature points with the largest roughness c that do not belong to the ground are extracted from each row in the sub-image, and $n\mathbb{F}_p$ planar feature points with the lowest c are selected from either ground or segmented points, also for each row in the sub-image. In the end, $\mathbb{F}_e$ and $\mathbb{F}_p$ correspond to the groups of edge and planar features from all sub-images. Then non-ground edge feature points nF_e with the highest roughness c and ground plane feature points nF_p with the smallest roughness c are selected from each row in the sub-image and the result is an edge feature set F_e and a plane feature set F_p for all sub-images, where $F_e \subset \mathbb{F}_e$ and $F_p \subset \mathbb{F}_p$.

C. LiDAR Odometry

The LiDAR odometry modules estimates the motion between two consecutive scans. The transformation is found by performing point-to-plane and point-to-edge scan matching. By other means, we need to search the corresponding features for points in F_e^t and F_p^t from the feature sets $\mathbb{F}_e^{t-1}$ and $\mathbb{F}_p^{t-1}$ of the previous scan [68]. In order to improve the accuracy and efficiency of the feature matching of LOAM, a couple of alterations were made:

1. Label matching

   In order to reduce the amount of points searched for matching, it only looks for correspondences with the same label from $\mathbb{F}_e^{t-1}$ and $\mathbb{F}_p^{t-1}$, depending on which type of feature, in this case F_e^t and F_p^t, respectively. This process, narrows the potential candidates and improves the efficiency.

2. Two-step L-M optimization

   The two-step Levenberg-Marquardt (L-M) optimization method seeks to find the minimum distance transformation between two scans. The first step is to estimate $[t_z, \theta_{roll}, \theta_{pitch}]$ by matching the planar features, F_p^t with their correspondences in $\mathbb{F}_p^{t-1}$. Then $[t_x, t_y, \theta_{yaw}]$ are obtained using the edge features in F_e^t and their matching correspondences in $\mathbb{F}_e^{t-1}$, while taking into account the constraints obtained in the first step $[t_z, \theta_{roll}, \theta_{pitch}]$. Finally, the 6D transformation is achieved by fusing $[t_z, \theta_{roll}, \theta_{pitch}]$ and $[t_x, t_y, \theta_{yaw}]$.

D. LiDAR Mapping

The LiDAR mapping module, instead of matching the features extracted from the feature extraction module against the previous scan, it matches the features present in the feature set $\{F_e^t, F_p^t\}$ with the surrounding pointcloud map Q^{t-1} to enhance the precision of the attitude transformation, the process proceeds by utilizing the ultimately adjusted pose acquired through L-M optimization [40]. The adjusted pose is then utilized to integrate spatial limitations that link the newly added node in the point cloud map with the previously selected node. Simultaneously, new constraints are introduced through loop detection. Afterward, the altered pose map is sent to GTSAM for the purpose of graph optimization, culminating in the update of the final estimated pose.

E. Transform fusion

The transformation module combines the pose estimation outcomes derived from both the LiDAR odometry module and the LiDAR mapping module, producing the ultimate pose estimation output.

F. Loop detection

The loop detection algorithm, present in LeGO LOAM, uses a KD tree to find historical poses and nearby pointclouds based on Euclidean distance. Then it uses ICP to calculate the transformation between the two key frames and retrieves an ICP fit score between the two pointclouds. If the fit score is below the threshold, it uses the pose of the most similar historical frame to constraint and correct the current global pose. Afterward, it updates the map to obtain a more consistent global map. This loop detection algorithm contains many flaws, mainly the large amount of computation and low detection accuracy. Since it is desired to be able to perform in a real time scenario, the usage of a lower frequency loop detection is implemented. Furthermore, a considerable combined error can be observed in the mapping of extensive and distant scenarios.

2.2.2 LIO-SAM

Two years later in 2020, Tixiao Shan and colleagues created LIO-SAM [52]. This new proposed approach contained a lot of similarities in comparison with LeGO LOAM, being the main difference the IMU integration in odometry optimization, in contrast with LeGO LOAM and LOAM the fusion of LiDAR and IMU was tightly-coupled instead of loosely-coupled. Loosely-coupled systems [51, 68] use IMU as a prior for the scan registration, for instance ICP orientation, and to remove pointcloud distortions. However, the IMU is not integrated in the optimization step. An alternative for these frameworks it to implement loosely-coupled fusion with EKF [35]. Tightly-coupled approaches such as LIOM [66] offered an improvement by optimizing the LiDAR and IMU odometry estimation. However, the strategy lacked in real-time performance. LIO-SAM comprises four factors when performing optimization: (i) IMU preintegration factor, (ii) LiDAR odometry factors, (iii) GPS factors, (iv) loop closure factors. All of them contribute to add constraints in the pose-graph. The graph is then optimized using the incremental smoothing and mapping (ISAM2) [29]. The registration of the raw data measurements from the LiDAR is equal to LOAM and LeGO LOAM. The loop closure module of LIO is similar to a radius search loop detector, that computes history keyframes based on Euclidean distance. The proposed algorithm has similarities with LeGO LOAM thanks to the represention of optimized poses in a graph, and thus a new novel SLAM framework SC LIO-SAM has already been presented combining a more robust loop closure detector based on features, denominated as Scan context [30].

2.2.3 Cartographer

Cartographer [26] is a 2.5 D graph-based SLAM framework, decomposed into local SLAM and global SLAM. The local SLAM is produced through scan matching in conjunction with IMU data. As local maps may gradually accumulate errors over time, a submap reaches completion once it gathers a specific number of scans, triggering the creation of a new submap. The other subsystem is the global

SLAM, built through the Graph-SLAM algorithm with direct constraints (from odometry) and virtual constraints (observations). It fulfills the role of reorganizing the submaps among themselves to form a coherent global map. To perform the scan matching and graph optimization, a library developed by Google, Ceres [2], was developed to solve the non-linear least squares problem. Ceres has the flexibility to be adjusted to give more or less confidence to the odometry data. The more significant the weight of a data source, the greater the emphasis cartographer will place on this source for performing the scan matching. In other words, the role of local SLAM is to generate good submaps, while the function of global SLAM is to integrate them cohesively.

2.2.4 HDL Graph SLAM

HDL Graph SLAM, as described in Koide et al. work [31], is a 3D Lidar SLAM that employs the RANSAC algorithm to identify key features within the point cloud data. Similar to the LeGO LOAM method, these features serve as constraints to determine the robot's position. The pointcloud registration employed by HDL is NDT (Normal Distribution Transform). To perform local map optimization, HDL uses RANSAC combined with *g2o* pose-graph optimizer [32]. To detect loops, it uses translation and trajectory length between nodes, and then to validate loop candidates, performs scan matching through ICP, GICP or NDT. The final estimated pose, similar to LeGO LOAM, involves a step that integrates transformations. This integration is combined with the position estimated through odometry and mapping, employing the UKF (Unscented Kalman Filter) technique to refine the results.

2.3 Loop Closure Detection

Loop closure, also referred to as place recognition in some relevant literature, is one of the key problems in SLAM mapping, as the robot/vehicle transverses a large cycle, the need to correct the map under large errors in not an easy task. Burgard [23] et al. stated that loop closure is when the robot is able to recognize a place or location that it has visited before. By comparing the current sensory data with the data collected in the past, the robot identifies similarities or matching features. Once the robot identifies loop closure, it leverages this information to correct and update its global map. By aligning the new sensory data with the existing map, the robot can refine its understanding of the environment and, thus, improve the robot localization and navigation capabilities, correcting any inaccuracies/inconsistencies that may have emerged during the initial mapping phase.

According to Xue et al. [10] there are two ways to perform loop closure: using geometry-based method or feature-based method. In the following subsections, we will discuss only LiDAR-based loop closure techniques.

2.3.1 Geometry-based

The most basic and primary methods used in loop detection were the geometry-based ones [23], the idea behind it is simple, it detects a loop by identifying a revisited place through robot positions, if the robot passes a position with a high likelihood from the observation measurements, than it means that the current position is a loop candidate. The drawback of choosing this approach is the fact that it does not consider uncertainty in the position of the robot/vehicle and when the accumulated error is too large, makes it harsh to detect potential revisited places.

Recent SLAM frameworks, to achieve higher accuracy using geometry-based loop closure detectors, began to utilize pose-graph optimization along with ICP. Zhang and colleagues [68] considered a loop candidate, by listing keyframes that are within a radius threshold for the current frame, and then applied ICP for every loop candidate listed. Whenever a revisited place is identified, a new constraint is added into the factor graph, connecting the variables associated with the most recent keyframe and the potential candidate keyframe. Every element within the pose graph symbolizes a 3D pose and corresponds to a keyframe within the ICP layer. Each factor, on the other hand, signifies a restriction connecting two of these elements. This kind of approach was also applied LeGO LOAM [51] and LiTAMIN [37].

2.3.2 Feature-based

When performing loop closure using feature-based place recognition techniques, there is always a trade-off that has to be made between robustness and computational cost. If the system you are employing owns less computation cost, choosing a feature-based handcrafted loop detector might be the best choice. However, if it has tons of computation resources and what you desire is more robustness, selecting learning feature-based loop detectors will not disappoint. Recently, most of the place recognition techniques use feature-based loop detectors due to their robustness and computation efficiency.

Handcrafted feature-based techniques

This section gives an overview of some handcrafted place recognition approaches widely used in the state-of-art, among them Scan Context, M2DP and Histogram.

Giseop Kim and Ayoung Kim created Scan Context [30], which is a global feature descriptor that encodes the information of a 3D pointcloud into a $N_r \times N_s$ spatial descriptor. To clarify this matrix, this means that for each pointcloud a scan context descriptor contains N_r which corresponds to the radial coordinates (distance from the sensor), and N_s sectors for each ring, related with the orientation of the sensor, whose FOV is 360 degrees. By performing circular shifting in the descriptor, it is possible to recognize $\frac{360}{N_s}$ viewpoint alignments. The algorithm rotation invariant,

since it preserves structural information of the scene, and therefore can recognize loops with different viewpoints. For each scan, a descriptor is stored and is matched against previous descriptors through a similarity score. If the match is below a certain threshold, a loop candidate is identified.

M2DP [24] presented by Li He et al. stands for Multi-Modal Distribution Preserving Pointcloud Descriptor and what it does is projecting a 3D pointcloud into multiple 2D planes to generate a signature matrix descriptor. Signature points are distinctive key points that stand out from the others, by other means, are points that exhibit different patterns. The multi-modal is because M2DP combines orientation of the surface, local point density and spatial distribution. Once the signature points are identified, they are used as the baseline of the algorithm. just like in Scan Context, MD2DP match the current local descriptor against previous pointcloud descriptors within the buffer.

Histogram [49] approach projects a pointcloud into a single dimension, using histograms. An example of a histogram function is the distance from the points detected to the sensor frame or the height above the ground plane. As the vehicle moves, it keeps comparing histograms from different locations. A similarity is computed, and a threshold is applied. If two histograms are too similar, a loop closure is detected.

Learning feature-based techniques

There are several place recognition learning feature-based algorithms using 3D Lidar as input [6, 11, 15, 57, 67], but we will only focus on two: PointnetVLAD and ORCHnet. These methods use deep-learning techniques to perform feature-learning and aim to transform LiDAR data into feature representations. The most commonly-used Pointnet VLAD [57], processes 3D pointclouds using Vector of Locally Aggregated Descriptors (VLAD), which represents an image as a fixed-length vector that encodes spatial information about local features in the map. Very recently, Barros et al. presented a robust global feature descriptor approach (ORCHnet) [6] with the aim of fusing multiple aggregation algorithms (SPoC, GeM and MAC) to identify orchards.

2.4 State-of-the-art discussion

In order to summarize and discuss the state-of-art presented above, the Tables below are shown. Table 2.1 shows a review of graph-based SLAM algorithms and compares them according to type of pointcloud registration, pose-graph optimization algorithm, which kind of maps they produce and computation resources.

SLAM Frameworks	Map representation	Pose-graph Optimization	Pointcloud registration	Computation resources
LeGO-LOAM	3D	L-M	Planar and edges features	⋆⋆⋆
LOAM	3D	L-M	Planar and edges features	⋆⋆⋆
Cartographer	2.5 D	Ceres	ICP	⋆⋆⋆⋆
HDL-Graph SLAM	3D	g20	NDT	⋆⋆⋆⋆
LIO-SAM	3D	ISAM	Planar and edges features	⋆⋆⋆

Table 2.1: Comparative Table presented to compare different SLAM frameworks.

Note: ⋆ are defined in a scale from 1 to 5 and go from low computational cost (1 ⋆) to high (5 ⋆)

To delve into the topic of loop closure, we've organized the place recognition algorithms, as outlined in section 2.5, into two comprehensive tables 2.2 and 2.3. This classification has been based on specific criteria, including place recognition category, computational cost, and robustness.

The first table places them into place recognition categories:

Loop detection algorithms		
Geometric-based	ICP, GICP and NDT	
Feature-based	Handcrafted	Scan Context, Histogram, M2DP
	Learning	Overlapnet, ORCHnet, PointnetVLAD

Table 2.2: The table presented allocates the categories of each different loop closure detector.

To analyze computational cost and robustness per category, the table below is presented:

Loop closure categories	Computation Cost	Robustness
Hancrafted feature-based	⋆⋆⋆	⋆⋆⋆⋆
Learning feature-based	⋆⋆⋆⋆	⋆⋆⋆⋆⋆
Geometric-based	⋆⋆⋆⋆⋆	⋆⋆⋆

Table 2.3: Comparative table for computational cost and robustness for each place recognition category.

Note: ⋆ are defined in a scale from 1 to 5 and go from low computational cost (1 ⋆) to high (5 ⋆)

3 Methodolody

This section describes the principal modules of the framework used as the cornerstone of the SLAM system. To start, we will begin by examining the SC LeGO LOAM framework, which contains all the baseline architecture of LeGO LOAM. Then we will dig into the loop detection module of SC LeGO LOAM. At a later stage of this section, we will present all the implemented work, such as the dataset acquisition of UC - POLO2 to test the loop closure algorithm and SLAM performance, as well as the implementation of loop closure protocols that gave us significant insight into its performance.

3.1 System Architecture

Firstly, the algorithm chosen as the baseline of this work was SC LeGO LOAM [65]. The main reason behind this pick is the fact that it contains a novel loop closure detector using LiDAR, designed Scan Context [30], which when compared to various state-of-the-art place recognition approaches, appears to have decent and competitive results. Since the aim of this thesis was to perform a comprehensive study on graph-based 3D LiDAR-SLAM and evaluate his performance in terms of loop detection, we considered two datasets for this matter: KITTI dataset and UC - POLO2 campus dataset.

As briefly described in section 2.2.1, SC LeGO LOAM is a 3D SLAM framework that combines two key components: Scan Context and LeGO LOAM. In order to understand the architecture of the entire system, it is relevant to conduct a thorough analysis of the algorithm and then, go into more detail about what has been achieved.

The main outputs of the SLAM framework is to construct a 3D map using mainly LiDAR and IMU and the correspondent trajectory. The diagram below shows all the system overview of the present work (Figure 3.1).

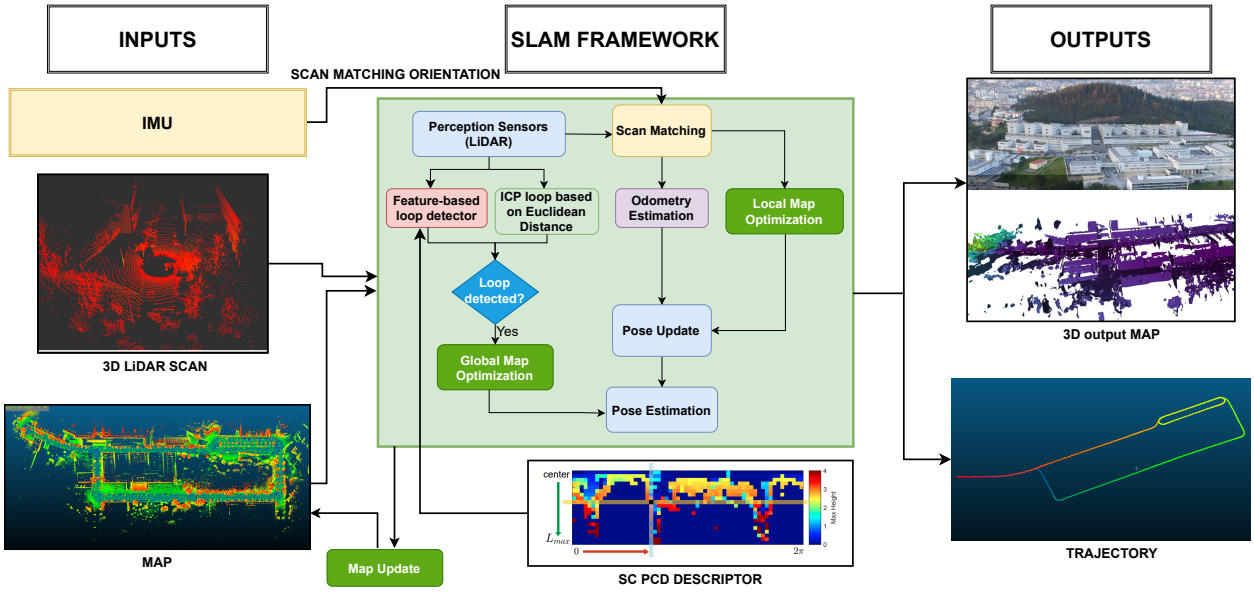


Figure 3.1: General system overview of the architecture.

3.1.1 Improved algorithm – SC LeGO LOAM

In order to enhance and address issues of efficiency and computational load, SC LeGO LOAM introduced the unique feature of incorporating a more robust loop closure detection algorithm in comparison to LeGO LOAM Euclidean distance ICP-based loop detector. At first glance, the initial advantages of Scan Context (SC) over Euclidean distance-based ICP (Iterative Closest Point) are twofold: a reduced amount of data to process while still effectively identifying previously revisited locations. To gain a clearer understanding, we will delve into the entire new loop detection algorithm, but first explain the SC LeGO LOAM algorithm architecture, shown in figure 3.2.

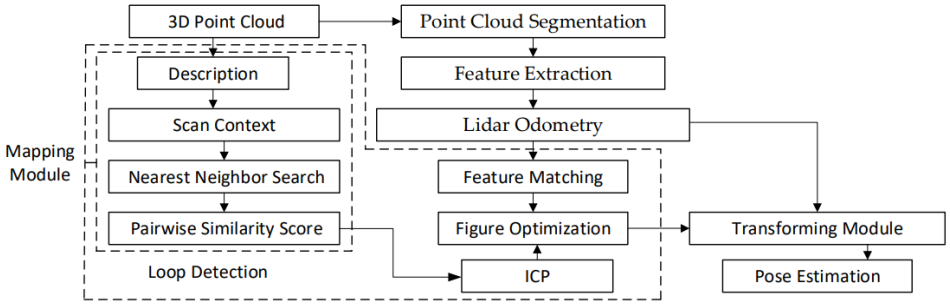


Figure 3.2: System structure diagram of the SC LeGO-LOAM algorithm [65].

The SC LeGO LOAM diagram flow is equivalent to the LeGO LOAM, with a little difference in terms of loop detection. The procedure is as follows:

1. Read LiDAR 3D pointclouds and project each pointcloud into a range image and perform segmentation. The segmentation module groups points in clusters, which can be marked as

ground points or non-ground points (segmented points). Three characteristics are assigned for each point: the label, row and column index in the range image and the distance value.

2. Subsequently, the feature extraction module calculates the roughness c and extracts edge feature points nF_e and plane feature points nF_p based on a ranking of surface roughness c.

3. The LiDAR odometry module estimates pose transformation between two consecutive frames through a two-step L-M optimization method that uses point-to-edge and point-to-plane scan matching.

4. In order to refine the pose estimation and update the global map, the LiDAR mapping module matches the features F_e^t and F_p^t with the surrounding map pointcloud and then uses the refined pose obtained through L-M optimization to add spatial constraints between the new node and the previous selected node of the pose graph constructed by GTSAM.

5. Alongside the LiDAR mapping module, to further eliminate the pointcloud drift, a scan context loop detector is applied to identify revisited places and also add new spatial constraints into the pose graph in case of loop closure, which will update and correct the map error that can have occurred in the mapping process. This loop detection runs at a slower rate, but enables to fix pointcloud drift.

6. To verify if a loop occurred, transform the current keyframe into the global coordinate system and perform registration against its history keyframes using ICP. If the ICP fit score is below a given threshold, the loop closure is considered successful, and the pose constraints are added between the current frame and the candidate frame. These constraints are then introduced into GTSAM for map optimization and refine pointcloud representation.

7. In the end, the transform fusion module combines the positions estimated from LiDAR odometry and LiDAR mapping and outputs a final refined estimated pose.

Scan Context

Scan Context [30] is an algorithm of loop detection present in SC LeGO-LOAM and is a global feature descriptor of a pointcloud. In other words, it is an outdoor place recognition that encodes a 3D pointcloud into a matrix $N_r \times N_s$. Then uses a ring key encoding function to provide a fast search and reduces the matrix to $N_r \times 1$, representing its final descriptor dimension. Then a KD tree is constructed through a stack of ring keys, and retrieves the nearest candidates. In the end, the retrieved candidates are compared against the query scan context, and those who satisfy the acceptance threshold and are close to the query are considered a loop. An overview of the scan context architecture is presented in figure 3.3.

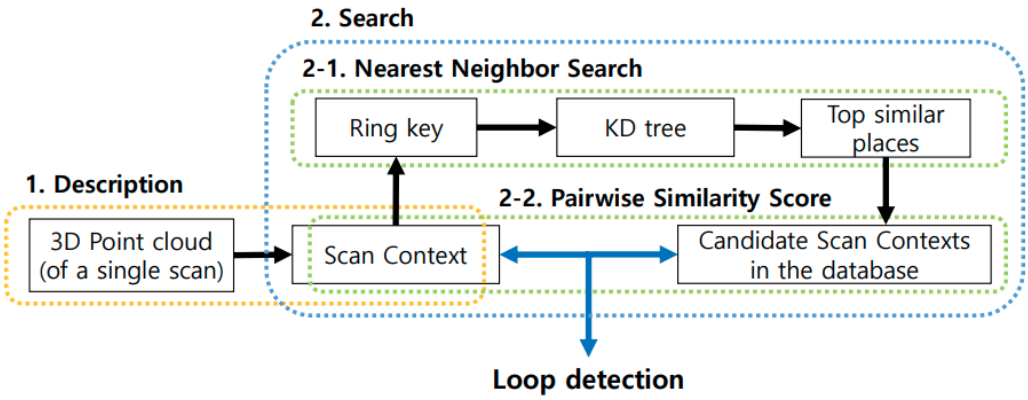


Figure 3.3: Scan Context algorithm overview [30].

To understand each process, it is crucial to know how the partition of the pointcloud into a matrix is performed. Firstly, the algorithm takes as input a 3D scan from the LiDAR, then the scan is divided into azimuthal and radial bins in the sensor coordinate system, ensuring equal spacing between them (figure 3.4a). Considering N_s and N_r number of sectors and rings, respectively, if we take L_{max} as the maximum range of the LiDAR sensor, the radial gap is $\frac{L_{\max}}{N_r}$ and the angle of each sector $\frac{2\pi}{N_s}$, assuming the parameters after some testing, were defined as $N_s = 60$ and $N_r = 20$, this means that from the center of the sensor to its maximum range, there will be 20 rings, and each ring is composed by 60 sectors.

Figure 3.4a shows that a scan context descriptor is a partition of the whole 3D pointcloud into bins. $\mathcal{P}_{ij}$ is the set of points belonging to a bin, where ith ring and jth sector overlapped. Thus, a partition can be expressed as:

$$\mathcal{P} = \bigcup_{i\in[N_r],j\in[N_s]} \mathcal{P}_{ij} \tag{3.1}$$

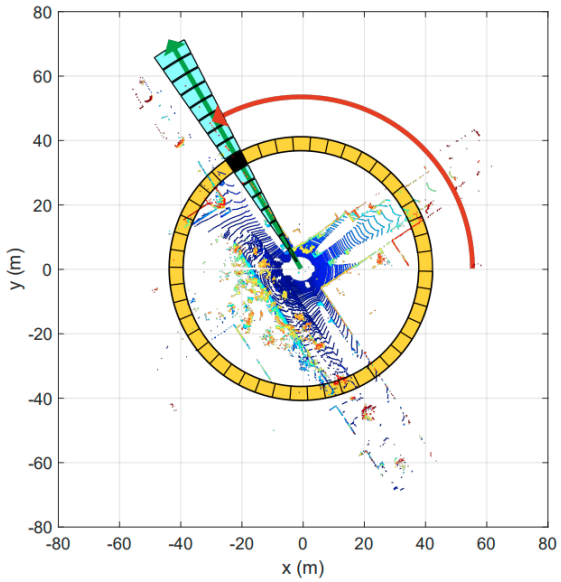
Since the pointcloud is divided into regular and constant intervals, a bin located farther away from the sensor will exhibit a significantly larger area compared to one that is closer. Nonetheless, both pieces of information are equally embedded within a single pixel of a scan context. Following the partitioning of the point cloud, each bin is allocated a singular numerical value by using the pointcloud in that bin. Different from the usual pointcloud descriptors, scan context bin encoding function ϕ does not use the histogram approach to gather the information, instead it uses the maximum height between the points belonging to the same bin. The purpose of employing the height is to effectively capture the vertical characteristics of nearby structures, accomplishing this without the need for complex computations to analyze the attributes of the point cloud. The bin encoding function is presented below:

$$\phi(\mathcal{P}_{ij}) = \max_{p \in \mathcal{P}_{ij}} z(p), \quad \text{where } \phi : \mathcal{P}_{ij} \to \mathbb{R} \tag{3.2}$$

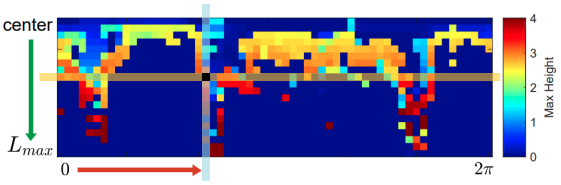
where $z(.)$ is a function that returns the height of a point $\mathbf{p}$ (z coordinate). Empty bins are assigned a zero value, and blue pixels in image 3.4b correspond to bins where the space is either free or not observed. Based on the previously mentioned procedures, a scan context denoted as I is ultimately illustrated as a matrix with dimensions $N_r \times N_s$:

$$I = (a_{ij}) \in \mathbb{R}^{N_r \times N_s}, a_{ij} = \phi(\mathcal{P}_{ij}) \tag{3.3}$$

The sector angle varies from 0 to 2π and corresponds to the azimuthal coordinate, whilst the radial coordinate is the distance from the center of the sensor to the maximum range. By storing information this way, the geometrical structure of the scene is preserved.



(a) Top 3D scan view divided in radial and azimuthal directions



(b) Scan context matrix

Figure 3.4: Creation of scan context [30]. Figure (a) shows the ground divided in azimuthal (0 and 2π) and radial directions (minimum sensor range to maximum). Yellow area corresponds to a ring, the cyan area is a sector, and black-filled dots are bins. Figure (b) represents a scan context matrix, as it can be seen the geometrical structure is preserved. The rows of the matrix correspond to rings and columns to sectors. The value extracted from the bins represent the maximum height of points belonging to that bin.

In figure 3.4a the yellow area corresponds to a ring, which are the points that are at the same radial distance, while the cyan area matches a sector, points that have equal orientation. The black filled dot represent a bin, this way a point can be expressed as polar coordinates $p(r, \theta)$, where radial equals r and azimuthal equals θ. It is observable that the center of the scan is a global keypoint, leading us to term a scan context as a self-centered location descriptor.

In the image on the right 3.4b the rows and columns are rings and sectors, respectively. As said before, it is visible how the internal structure of the pointcloud is preserved in contrast to the histogram. To verify if two different places match, scan context compares descriptors through a two-step search algorithm composed by similarity scoring and nearest neighbor search.

One of the steps, similarity score, calculates the distance between two scan context descriptors,

the distance measures how similar are both descriptors. Assuming I^q and I^c are two scan context descriptors, acquired from the query pointcloud and candidate pointcloud, respectively. In order to compare both, a pairwise distance is calculated in a colmnwise manner. To perform this operation, for the same column index of two different descriptors, c_q^j and c_c^j, query and candidate descriptor, respectively, for the same column index j. The cosine distance is employed to compute the distance, and this process is reiterated until the final point is reached. The distances are summed and subsequently divided by the total column count, the equation 3.4 shows how the process to calculate similarity between descriptors is performed:

$$d(I^q, I^c) = \frac{1}{N_s} \sum_{j=1}^{N_s} \left(1 - \frac{c_q^j \cdot c_c^j}{||c_q^j|| ||c_c^j||} \right) \tag{3.4}$$

The columns can be shifted due to the fact that the LiDAR is observing from a different viewpoint, for example, revisiting the same location but in the opposite direction, formally referred to as a reverse loop. The same doesn't apply to the rows, as the representation of the scan context relies on the sensor's location. Thus, the rows always maintain the same order and consistency, with only the columns varying if the sensor's coordinate system relative to the world coordinate system changes.

To address this issue, for a given descriptor, a circular shift is applied to all columns, and the similarity score is calculated for each shift. The minimum distance and the yaw of the shifted descriptor, I_n^c, are returned. This is achieved by comparing it with the original descriptor that hasn't undergone any shifting, I^c. Essentially, this process aligns two point clouds in terms of yaw. To calculate the yaw, one only needs to determine the number of columns by which it has been shifted compared to the original, and then multiply it by the resolution of $\frac{2\pi}{N_s}$. In summary, the minimum distance between two descriptors is returned, taking into account the shift across all columns, along with the number of columns that were shifted to achieve that distance, the expressions below show mathematically what was explained:

$$D\left(I^q, I^c\right) = \min_{n \in N_s} d\left(I^q, I_n^c\right) \tag{3.5}$$

$$n^* = \arg\min_{n \in N_s} d\left(I^q, I_n^c\right) \tag{3.6}$$

The other step of the searching algorithm consists of searching for nearest neighbors, it uses a KD tree constructed through ring keys. A ring key is a rotation-invariant descriptor obtained from the scan context descriptor, which consists of applying a mean to every bin value belonging to the same ring.

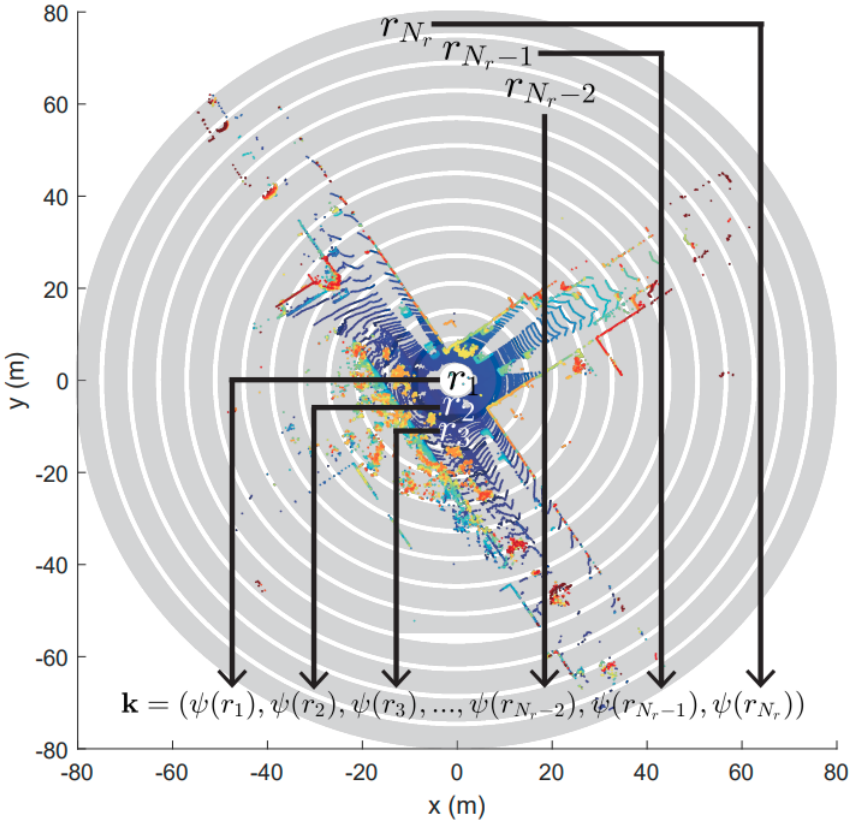


Figure 3.5: Ring key generated for fast search [30].

Considering an N_rxN_s matrix, where N_r and N_s are, respectively, number of rings and sectors, by applying a mean for each ring. The new descriptor will now be an N_rx1 matrix, where each row value contains the mean value of each ring section.

$$\psi(r_i) = \frac{1}{N_s} \sum_{j=1}^{N_s} \mathcal{P}_{ij} \tag{3.7}$$

where r_i defines the ith ring and $\mathcal{P}_{ij}$ the bin value for ith ring and jth column.

In picture 3.5, $\{r_1, r_2 \dots r_{N_r}\}$ are the rings of the scan context descriptor. While not as detailed as scan context, the ring key allows for quick searching to identify potential loop candidates. In SLAM context it is really useful as it stores a buffer of much lighter descriptors every time the algorithm concatenates pointclouds, by other means in context of SLAM, since the algorithms is continuously receiving 3D scans, whenever there is a new pointcloud, scan context stores in a **k** vector a descriptor using the ring key encoding function ψ present in expression 3.7.

$$k = (\psi(r_1), \dots, \psi(r_{N_r})), \quad \text{where } \psi : r_i \to \mathbb{R} \tag{3.8}$$

Therefore, the scan context buffer used in this work is as follows:

$$SC_{buffer} = (k1, k2, \dots, k_{N_p}), \quad \text{where } N_p \text{ is number of pointclouds} \tag{3.9}$$

Each **k** vector is used as a key to construct the KD tree. Simultaneously, the ring key of the query scan is used to look for similar keys and their corresponding indexes. The number of the top similar keys that are going to be retrieved is defined by the user, in case of SC LeGO LOAM by default it retrieves 10 candidates maximum. Right after the construction of the KD tree, the next steps run in parallel: search for nearest neighbors and compute the similarity score between the query scan context descriptor and the retrieved candidates (equation 3.4). The candidate with the lowest distance (higher similarity) is selected. If the candidate fits the acceptance threshold, the place is considered as a revisited place:

$$c^* = \arg\min_{c_k \in \mathcal{C}} D(I^q, I^{c_k}), D < \tau \tag{3.10}$$

where $\mathcal{C}$ is the set of candidates extracted from the KD tree, τ the given acceptance threshold and c^* the index of the place identified as a loop.

3.2 Software Requirements

In order to test the software, we used Ubuntu 20.04 with the ROS framework (ROS 1 Noetic).

3.2.1 SLAM setup using ROS

A brief introduction to ROS is present in appendix A.1. To be able to use ROS with SLAM, the robot/vehicle must be equipped with sensors and its drivers, as well as a workstation (computer) that can run ROS nodes. ROS must be configured on both robot/vehicle and the computer. To run SC LeGO LOAM on any SLAM application, it requires launching sensors and driver nodes in the platform where it is running, for instance, in a robot to be able to send velocity commands through a controller (via Bluetooth or Wi-Fi). In a car, if you are already driving it, this part can be skipped, unless it is an autonomous car, in that case, the procedure is similar. The following step is to launch the SLAM node from the computer, and to analyze whether it performs SLAM well or not, a visualization tool, such as Rviz, can be launched.

3.2.2 Inputs and Outputs

The SLAM node of SC LeGO LOAM subscribes to two main topics: IMU and LiDAR topic. The IMU topic uses ROS messages type `sensors_msgs/Imu`, which typically includes measurements of acceleration, angular velocity and orientation. The topic that reads this data, by default is set in `'/imu/data'`, any node that wants to receive data from IMU needs to search for this topic. In SC LeGO LOAM the IMU is used to de-skew the pointcloud and serve as an input for the ICP orientation (scan matching prior), this type of sensor fusion with the LiDAR is classified as loosely-coupled and was first introduced in LOAM [51]. The LiDAR topic messages are the type

`sensor_msgs/PointCloud2`, which represent a collection of 3D points in space and is our main source to estimate robot/vehicle motion through feature scan-matching. The published topic of the pointclouds is `'/velodyne_points'`. The algorithm does not have odometry as an input, because this way, we can study the weaknesses and strengths of LiDAR-based SLAM techniques more precisely. Besides, odometry obtained through GPS/wheel encoders or other odometry estimation algorithms can be used as ground truth to compare with the output of SLAM.

The outputs of SC LeGO LOAM are 6D pose estimated after and before the mapping process. The first is the refined position after passing through the transform fusion module of the type `nav_msgs/Odometry`, in the topic `'aft_mapped_to_init'`. The second one is obtained in the LiDAR odometry module, using only the two-step L-M optimization method, through feature scan matching. The message type is the same, but the topic name is `'integrated_to_init'`. Besides the position estimation output along the trajectory, the algorithm stores a final map in `.pcd` format. For localization-only algorithms, this stack of pointclouds and estimated poses during the trajectory represent good inputs, considering the map did not suffer any alterations over time, if not the case, a more complex approach must be taken. Figure 3.6 illustrates how to setup a computer with ROS and run SLAM.

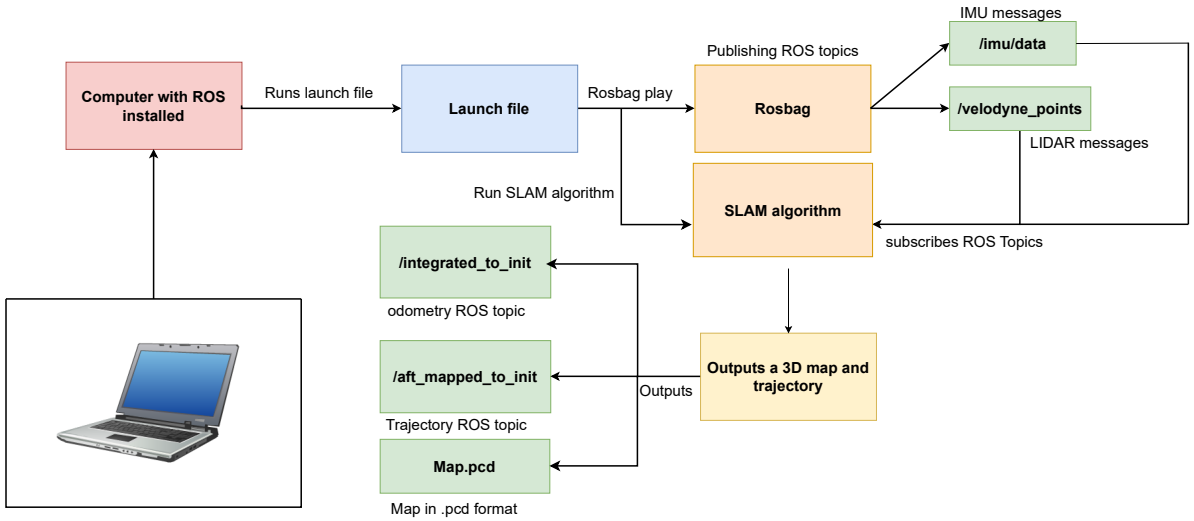


Figure 3.6: SLAM setup with ROS.

3.3 Sensors

As we already mentioned above, our SLAM algorithm uses LiDAR and IMU as inputs. However, to be able to compare and evaluate different algorithms and datasets, an additional sensor, namely GPS (Global Positioning System), was incorporated to obtain a ground truth trajectory. This trajectory served as a reference against which the performance of the SLAM (Simultaneous Localization and Mapping) algorithm was evaluated and compared. The utilization of GPS data

enabled a comprehensive assessment of the algorithm's accuracy and effectiveness in mapping and localization tasks.

Since we used two datasets for SLAM evaluation (KITTI and UC- POLO2 dataset) in the upcoming section, we exclusively elaborate on the sensor employed to collect the custom dataset. Subsequently, in the following section dedicated to the Datasets, we provide a concise overview of the sensors utilized in the open-source dataset of KITTI.

3.3.1 Xsens MTi-300 IMU

The xsens MTi-300-2A5G4 [64] is an Inertial Measurement Unit (IMU) renowned for its precision and versatility. This is a 9-axis degree-of-freedom IMU, that contains sensor fusion algorithms, several sensors (accelerometer, gyroscope and magnetometer). Each of these sensors provides specific information about motion and orientation, and when combined, they can be used to estimate position and orientation in three-dimensional space.

Figure 3.7: IMU xsens MTi-300-2A5G4

Here is how a 9-axis IMU works:

- Accelerometer

  An accelerometer measures linear acceleration along three axis (X, Y and Z). It detects changes in velocity and gravity, which allows the device to determine movement and orientation with respect to Earth gravitation field. By integrating acceleration over time, velocity is obtained, and by integrating velocity we estimate position. However, as any sensor, sensor drift error accumulates over time and the accuracy of the accelerometer tends to degrade after long periods.

- Gyroscope

  A gyroscope measures the angular velocity around the three-dimensional axis. By integrating gyroscope output over time, we estimate the orientation in terms of angle (roll, pitch, yaw). Just like the accelerometer, integration errors accumulate over time, generating drift.

- Magnetometer

  A magnetometer measures the direction of the magnetic field. This sensor is used to determine the orientation with respect to Earth's magnetic north. By combining a magnetometer with an accelerometer, orientation estimation can be improved, especially in case where linear acceleration does not change much. The magnetometer represents the main difference between a 6-axis IMU and 9-axis.

- Barometer

  A barometer sensor measures the atmospheric pressure. This pressure data can be used to estimate changes in vertical position. It does not provide orientation estimates like the accelerometer and gyroscope, but can enhance accuracy of position estimates when combined with other sensors, such as accelerometer and GPS.

The orientation estimation responsible for calculating the orientation angles (roll, pitch and yaw) is then a combination of the accelerometer gravity vector, gyroscope angular rates and magnetometer magnetic field data. It uses a Kalman filter to compute a statistical optimal 3D orientation estimate of high accuracy. The output angles represent the device orientation relative to the IMU reference frame, set during calibration. The position estimation is an integration of the accelerometer data, to get linear velocity and then a double integration to estimate position. The vertical position (z coordinate) is a combination of the barometer and accelerometer.

The IMU, presented in figure 3.7, has a rate of turn of around 450 degrees per second, and the interface to connect it to the computer is a converter from a RS232 Serial cable to USB.

Sensor Fusion performance		Gyroscope		Accelerometer	
Roll/Pitch	0.2º RMS	Rotation Turn	450 deg/s	Standard Full Range	20 g
Yaw/Heading	1º RMS	Bias Stability	10 deg/h	Bias Stab.	15 µg
SDI	Yes	Noise Density	$0.01°/s/\sqrt{Hz}$	Noise Dens.	$60\ \mu g/\sqrt{Hz}$

Table 3.1: Sensor technical specifications.

3.3.2 OS1-64 Ouster Lidar

The high-resolution *ouster LiDAR* [42] can achieve a range between 0.8 and 120 metres. The FOV of this device is 360º and outputs pointclouds at a rate frequency of 10 HZ to a maximum of 20 HZ.

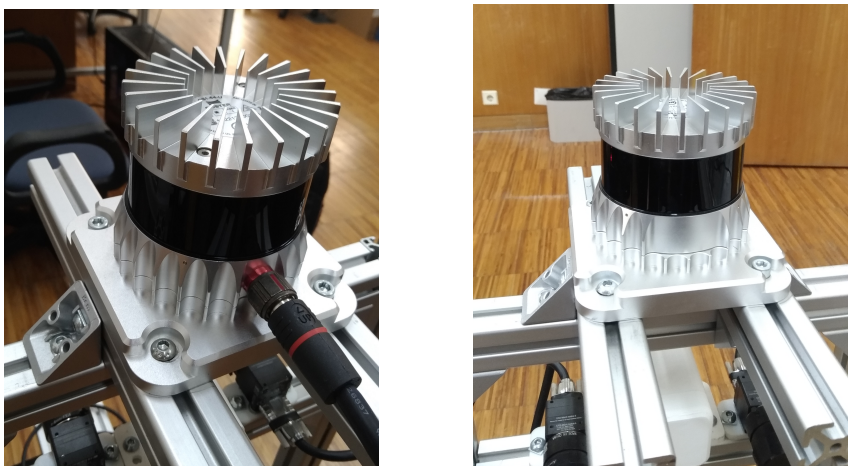


Figure 3.8: OS1-64 Ouster LiDAR.

The vertical FOV is from (-16.6º to 33.2º) and is composed by 64 vertical beams, whilst the number of horizontal beams by default is 1024, but can vary between (512,1024 and 2048). The vertical and horizontal resolution in degrees, using the configuration (10 hz rate, 1024 horizontal beams and 64 channels) can be calculated using the following formulas:

$$V_{res} = \frac{33.2^{\circ}}{V_{beams} - 1} \approx 0.53^{\circ} \tag{3.11}$$

$$H_{res} = \frac{360^{\circ}}{H_{beams}} \approx 0.35^{\circ} \tag{3.12}$$

The technical characteristics of this device are shown in the table below:

	OS1-64
Vertical Resolution	64 channels
Horizontal Resolution	512, 1024 or 2048 channels
Range	120 m
Vertical FOV	-16.6º to 33.2º
Vertical angular resolution	0.35º - 2.8º
Precision	1.5 - 5 cm
Rotation rate	10 or 20 HZ
Power draw	14 - 20 W

Table 3.2: Lidar technical specifications.

The Ouster LiDAR also contains an internal IMU integrated, which outputs at a maximum rate of 100 HZ. This IMU has the same coordinate frame system as the LiDAR and there is no need to provide to the SLAM algorithm the extrinsic transformation between them.

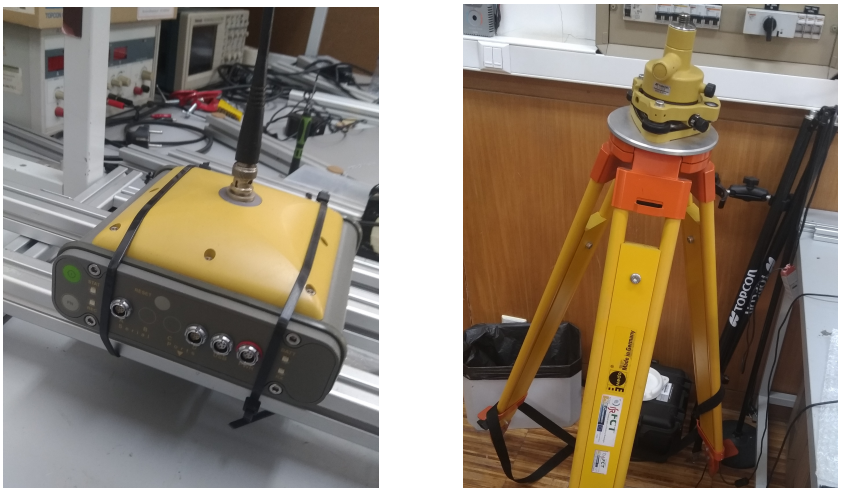
3.3.3 Topcon Hiper Pro GPS

Global Positioning System (GPS) is a satellite-based navigation system that allows users to determine their precise location and monitor their movements. Once GPS was used to obtain ground truth, high precision was necessary, and therefore, the use of GPS was only considered if the option of RTK was available. RTK is a technique used in GPS to achieve highly accurate positioning in real-time. The basic principle behind RTK is the way that distance is calculated. A traditional GPS uses pseudo-range measurements to calculate distance from the satellite, which provides accuracy of meter-level. Pseudo-range measurements involve measuring the time since the GPS signal was sent from the satellite until it reached the receiver. The GPS satellite transmits a signal that contains a specific code, and the receiver compares the code received with its own code. The time between the transmitted and received code (time of travel) $(t_r - t_s)$ multiplied by the speed of light c gives the distance between the satellite and the receiver. The biggest problem of this approach is that GPS signals can be affected by multiple sources. From Ionospheric and Tropospheric delays, reflection on surfaces (multipath interference), ephemeris errors, etc.

$$d = c \cdot (t_r - t_s) \tag{3.13}$$

Carrier-phase measurements were introduced as a significantly much better algorithm to reduce the distance error inherent in code-phase measurements, enabling a much higher level of precision in GPS positioning. The receiver tracks the exact number of carrier wave cycles between the GPS satellite and the receiver. Then the number of carrier wave cycles N multiplied by the signal wavelength λ combined with the phase ϕ. The carrier wavelength is around 20 centimeters, which has shown to be much shorter than the code length. The error using this approach is fractions of a wavelength and the way it is calculated is shown below:

$$d = N \cdot \lambda + \lambda_\phi \tag{3.14}$$



(a) GPS receiver (b) Base station tripod

Figure 3.9: GPS Topcon Hiper Pro [56]

In order to understand of how an RTK GPS works, we will present a brief sequence of the whole process:

1. Base Station Setup:

   The base station is set up at a known static location with a clear view of the sky to have precise coordinates. The base from now on will keep tracking signals from the satellites and calculates its position using pseudo-range and carrier-phase measurements (centimeter-level precision).

2. Rover Receiver Setup:

   The rover whose position changes over time as it is moving, calculates its initial position using GPS signal it receives from the satellites using code phase measurements (meter-level precision). Then it sends its initial position estimate to the base station.

3. Base Station Calculation:

   The base station receives the rover initial position estimate and compares it with its own measurements. The base calculates the differential corrections needed to refine the rover position estimate so that it align with its known (highly accurate coordinates of the base).

4. Correction Transmission:

   The calculated differential corrections are then sent back to the rover from the base station, usually via radio or NTRIP. The corrections are applied to the rover measurements and its position estimate is updated with a much higher precision. The rover

5. Continuous Updates:

   As the rover keeps moving, it continuously keeps receiving real-time corrections from the base.

The base station besides calculating the corrections for rover, also performs complex calculations to improve its accuracy. This process allows high precision positioning in real-time. The corrections are sent in `RTCM` format. The GPS outputs NMEA messages, in RTK the maximum rate is 20 HZ. To understand the `NMEA` message structure, let's examine the popular `$GPGGA` message:

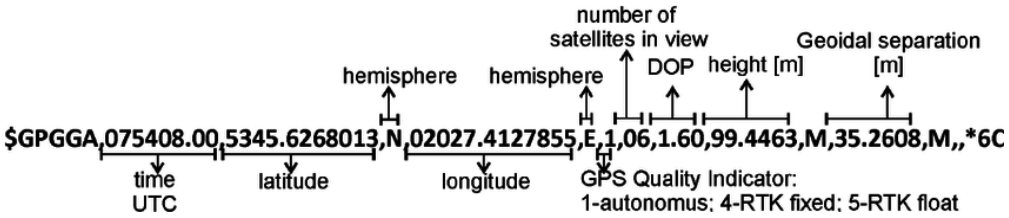


Figure 3.10: Example of a basic GPS NMEA message data structure (GPGGA)

`GPGGA` stands for Global Position System fix data and is a standard NMEA 0183 message (appendix B.1) used in GPS systems.

3.4 Datasets

Datasets in the context of SLAM are indispensable for many reasons. They serve as benchmarks for SLAM algorithms, allowing us to evaluate their performance and robustness in real-world scenarios. Acquiring datasets for SLAM testing, serves as a valuable tool that offers the best of both worlds. It is similar to running a simulation, yet with the advantage of utilizing real-world sensor measurements. These datasets enable rigorous testing and provide a platform for fine-tuning SLAM algorithms.

After careful consideration and evaluation of numerous autonomous driving datasets, we have deliberately chosen the KITTI [19] dataset as the most suitable option for our specific research requirements. KITTI stands out as an ideal choice due to its meticulous design, the algorithm was created with a target in autonomous driving applications. Moreover, KITTI's utilization of crucial sensors, such as the LiDAR and IMU aligns perfectly with our predetermined criteria.

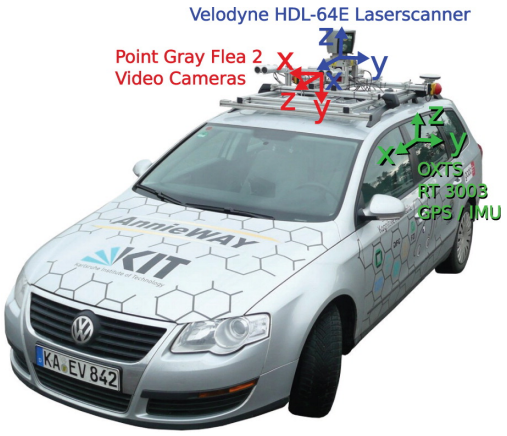
There were also other datasets that caught our attention, such as the FORD dataset [43]. However, the issue was related to the fact that it included a dataset that was excessively large and had too few revisited locations, which is essential for the loop closure module. The easiest way to run a SLAM algorithm using ROS is through rosbags. Therefore, we only considered datasets that were obtained or can be converted into a rosbag and are related to autonomous driving scenarios. The first dataset, KITTI is an open-source dataset which allowed to get quick results that contributed to validate the SLAM algorithm. The second dataset, UC – POLO2, is an in-house dataset. the main reasons behind the acquisition of this dataset are the fact that it contains reverse loops, even though KITTI sequence 02 also contains, it represented an advantage to test the loop closure algorithm Scan Context. The second motive is because of the sensors, which are highly advanced, slightly

more recent than those employed in KITTI dataset. Therefore, our objectives were to test the sensors to determine if there were any differences in the performance of the algorithm.

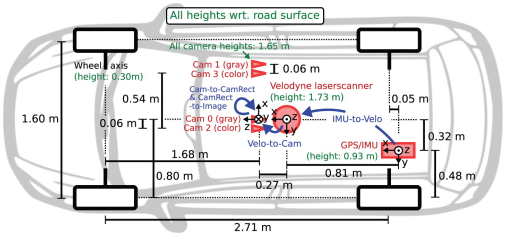
3.4.1 KITTI Dataset

KITTI [19] is an open-source dataset designed for autonomous driving applications. The dataset contained diverse sensory components, ranging from 2 RGB and 2 grayscale cameras (Sony ICX267 CCD 1.4 MP), 3D LiDAR (Velodyne HDL-64E), inertial measurement unit (OXTS RT3003 IMU) and RTK GPS. Synchronization and calibration measures were taken to avoid drift in orientation, which would lead to translation error. The dataset comprises several environment categories (Road, City, Residential, Campus) allowing diversity for SLAM validation.

The collected data was zipped into a single file containing the recording data and the sequence number. For testing purposes, we used a package called **"kitti2bag"** which converts the sensors readings into a rosbag file, containing all the relative transformations required. To obtain ground-truth data, there is a section in KITTI official site, that contains the start and end time for each KITTI sequence, then we used a package to obtain the ground truth poses synchronizing the raw data timestamps with the estimated ground-truth. The KITTI ground-truth was obtained using RTK-GPS/INS fusion with an accuracy below 10 cm. The figures below show how the setup configuration for this dataset:



(a) KITTI platform. The vehicle is equipped with four video cameras (two color and two grayscale cameras), a 3D laser scanner and a combined GPS/IMU inertial navigation system.



(b) This figure illustrates the sensor's setup (red) with respect to the vehicle body. Heights are marked in green and were measured with respect to the road surface plane. The transformations between sensor frames are marked in blue.

Figure 3.11: KITTI car setup.

3.4.2 UC - POLO 2 Dataset Acquisition

Dataset Description

Firstly, UC-POLO 2 Dataset was taken for exclusively Lidar-based frameworks, and does not provide cameras for visual SLAM. The experiment took place in the POLO 2 campus of the University of Coimbra, and represents a dataset for outdoor urban environment. This dataset was explicitly constructed to evaluate the performance of scan context loop detector, as it contains reverse loops that help to validate the algorithm robustness. Besides, once we afforded highly advanced and up-to-date sensors, and we wanted to obtain a dataset for our campus, we've decided to take a step forward and assume the construction of it.

This section will define and explain all the measures taken to produce the dataset, from hardware selection, setup preparation, sensor calibration, ground-truth data collection and data-post processing.

The major sensors used in this dataset included: Xsens MTi-300 IMU (figure 3.7), OS1-64 Ouster Lidar (figure 3.8) and RTK GPS (figure 3.9a). To start, we wanted to make this work within the ROS framework because of the SLAM frameworks we have chosen. To do that, some measures were taken. Installation of ROS drivers for each sensor, by installing the drivers we were able to record data through `rosbags` in ROS message formats. The IMU already had a driver for that matter, we just needed to check if it was aligned with the LiDAR and GPS reference. For the GPS, we installed a driver to parse the standard NMEA messages and publish in the ROS topic "`/fix`". And the last sensor, the ouster LiDAR also already owned ROS driver, we just had to configure it to use ROS timestamps, rate of data collection and horizontal resolution.

Figure 3.12: System overview. 64-channel Ouster LiDAR installed in the top of the car (sensor in the front row), Xsense MTI-300 IMU installed at the back of the car, and RTK GPS is in between LiDAR and IMU.

Sensors Setup and Data Collection

The sensors coordinate system employed was a right-handed coordinate system (x forward, y left and z up). The system was composed of a 64-channel LiDAR in the top of the car, slightly highly placed, to avoid detecting the car, which would interfere with the pointclouds. Alongside the LiDAR, there was a 9-axis IMU in the middle center located at the back of the car, and GPS complemented the list of sensors situated in the middle of LiDAR and IMU. The dataset contains two sequences for LiDAR-based SLAM applications.

Before initiating the system deployment, we wanted to make sure that every sensor that was going to be integrated was prepared for the ROS environment. The next step, included IMU calibration, to keep simplicity we wanted the sensors to have aligned system coordinates to avoid having to apply many transformations in the vehicle description. After calibrating the sensors, installing the drivers, and assembling the platform, the next step was to record the dataset and start collecting data. For that, we launched a ROS launch file, containing drivers of each sensor and a rosbag to record all the topics that were being published. The LiDAR was collecting data at 10 hz, whilst the IMU was acquiring data at 400 hz, so synchronization was not needed. Some SLAM algorithms like LIO-SAM and Cartographer require high-performing IMU and take advantage of

that to outperform other approaches, since they integrate IMU measurements in conjunction with LiDAR odometry to improve the pose estimation.

To be able to run the recorded dataset in ROS, a URDF model of the vehicle was highly demanded. URDF is characterized by the relative transformations of the sensors (extrinsic transformation between sensors) and ensures that they are accurately registered in a common reference frame. This step was important so that at the time of launching SLAM in ROS, the visualization of the frames would be correct. To achieve this, we considered the origin of the vehicle reference frame, as the center back wheels baseline. After some exhaustive measures, the system setup is presented below in figure 3.13.

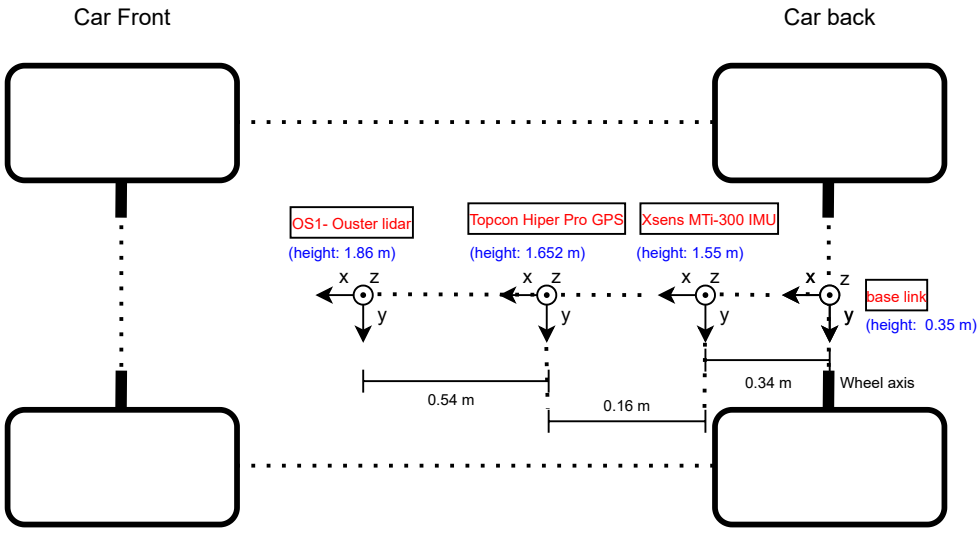


Figure 3.13: Sensors setup. This figure shows mounting positions with respect to the base link. Heights above the ground are marked in blue and measured with respect to the road surface. Transformation between sensors are explicited in meters.

Ground Truth Data Collection

To perform an assessment of the SLAM algorithms, acquiring ground truth data was essential to evaluate SLAM performance. We used an IMU as an orientation ground-truth and RTK GPS for positioning. The GPS was configured in RTK mode to provide high-precision data, we setup a base station with a PDOP (Position Dilution of Precision) under 1 meter, which is a very good value for GPS positioning, allowing accuracy of centimeters (section 3.2 explains how to setup an RTK GPS). The GPS was publishing ROS messages of the type `sensors_msgs/Navsat_fix` at a fixed rate of 5 hz. GPS coordinates provide information about the latitude, longitude, altitude, position covariance, and other GPS-related information in the WGS-84 reference system (most common reference system for GPS data). In cases where the RTK signal was lost (due to objects occlusion), the GPS would still send data in fix mode, obviously with a lower precision. For visualization of the sequences taken, we produced a KMZ file from the rosbag recorded. The procedure was the

following:

- Created a script to open the recorded rosbag and read the topic `'/fix'` for GPS messages.
- Parse the GPS ROS messages `'sensors_msgs/Navsat_fix'` and extract latitude, longitude and altitude from each message.
- Use a library to convert GPS data into KML file. Then compress the file into KMZ file.
- Import the KMZ file in Google Maps and display the sequences of GPS data from each dataset sequence.

The sequences of UC - POLO2 are shown in figure 3.14: Sequence 01 contains two long roads with reverse loops. Sequence 02 contains a reverse loop when returning and direct loops when crossing the roundabout, as it crosses it twice. To implement this GPS data as a ground truth option for SLAM comparison, two final procedures had to be implemented, system coordinates conversion and data post-processing.

Data Post-processing

After collecting the data, the first step was to perform post-data processing to remove noise, zero values and ground-truth outliers originating from the RTK signal loss. After analyzing the recorded bags, the IMU contained zero values and GPS data showed RTK signal loss in some stages, reducing the precision of our ground truth. Therefore, a post-process was required as it follows:

- Created a script to read the IMU data in the topic `'/imu/data'` and removed zero values from it.
- Read the topic of GPS, converted to UTM coordinates, according to the region. The first position was removed to start in the position (0,0).
- A filter was applied to the GPS to reduce the noise of the sensor, especially the z coordinate that got peaks in height. The filter used was a polynomial interpolation of the GPS data using the Savitzky Golay filter [44], with a 4-th order polynomial and 50 points in the moving window (*i.e.*, 10 seconds of data) for the xy-axis, and 250 points in the moving window for the z-axis (*i.e.*, 50 seconds of data).

(a) The trajectory of the vehicle for the Sequence 01 of UC - POLO2 dataset.

(b) The trajectory of the vehicle for the Sequence 02 of UC - POLO2 dataset.

Figure 3.14: Sequences of UC - POLO2 dataset. (a) and (b) are both trials around the POLO2 campus, the trajectory points are GPS traces of our recordings. The starting and ending points of each sequence individually is approximately the same.

3.5 Loop Closure Implementation

In this section, we present protocols to obtain ground truth data for loop detection, in order to perform the evaluation of the SC LeGO LOAM loop detection algorithm. The implementation of PointnetVLAD and ORCHnet inside the loop closure module is presented in Appendix C.1.

Loop Closure Protocols

To thoroughly assess the effectiveness of place recognition approaches, we employed two distinct evaluation protocols [60] to evaluate the recall and precision of each method we are comparing. Consider f_{kc} as the current keyframe, $L[f_{kc}]$ the output frame of the place recognition module, $H[f_{kc}]$ the list of history keyframes for the current frame and d_f a threshold applied in the place recognition module. The *protocol A* evaluates in case of a detected looop by the place recognition module, if there is a history keyframe for the current frame within a radius of 4 m, then the loop is considered a true positive. Otherwise, is a false positive. If the loop module outputs "-1" (means no loop candidate) and the condition for ground truth is verified (exists a history key frame within a radius of 4 m for the current frame) then is considered a false negative and if both output "-1" then is a true negative.

If the output of loop closure module, $L[f_{kc}]$, is under a distance feature threshold, d_f the prediction is positive. Otherwise, is negative. To calculate ground-truth data for loop closure, we use a KD tree to find 10 nearby history key frames within a radius for f_{kc}. Considering this an evaluation based on Euclidean distance, we discard the previous 300 scans (last 30 sec of data). Then we retrieve the minimal distance found for the current keyframe, $d[f_{kc}]$. If $d[f_{kc}]$ is less than 4 m then ground truth for f_{kc} is considered positive.

Algorithm 1 Protocol A for loop-closure evaluation

Variables

Let f_{kc} be the current keyframe, $H[f_{kc}]$ list of history frames for current key frame, $L[f_{kc}]$ the output of the loop module for the current frame, $S[f_{kc}]$ feature score for the current frame, d_f feature threshold and $d[f_{kc}]$ closest Euclidean distance between current keyframe and $H[f_{kc}]$.

end Variables

Find if f_{kc} has been transversed:

for `each` f_{kc} **do**

if $L[f_{kc}] ==$ "-1" **then**

No loop candidate found.

else if $(L[f_{kc}] \geq 0)$ and $(S[f_{kc}] \leq d_f)$ **then**

Potential Loop candidate. (Positive prediction)

for each $H[f_{kc}]$ **do**

if $d[f_{kc}] \leq 4$ **then**

Loop candidate confirmed. (True positive)

else if $d[f_{kc}] > 4$ **then**

Loop candidate predicted is a false positive.

end if

end for

else if $(L[f_{kc}] \geq 0)$ and $(S[f_{kc}] > d_f)$ **then**

Loop candidate rejected (Negative prediction).

end if

end for

The *Protocol B* uses ICP for ground-truth. The process is very similar to protocol A with the main difference being how we consider a positive and negative for ground-truth, instead of searching for nearby history keyframes it uses the ICP as a reliable source for ground-truth. The algorithm is as follows:

Algorithm 2 Protocol B for loop-closure evaluation

Variables

Consider I_{thresh} as the ICP fit score threshold, $I[f_{kc}]$ the ICP fit score for the current frame, $S[f_{kc}]$ feature score for the current frame, $L[f_{kc}]$ the output frame of the place recognition module, d_f feature threshold, f_{kc} be the current keyframe and N_{hf} number of near history key frames will be considered by ICP to fuse the query frame and the candidate frame.

end Variables

Find if f_{kc} has been transversed:

for `each` f_{kc} **do**

if $L[f_{kc}] ==$ "-1" **then**

No loop candidate found.

else if $(L[f_{kc}] \geq 0)$ and $(S[f_{kc}] \leq d_f)$ **then**

Potential Loop candidate. (Positive prediction)

Applies ICP for the N_{hf} (surround cloud)

if $I[f_{kc}] \leq I_{thresh}$ **then**

Loop candidate confirmed. (True positive)

else if $I[f_{kc}] > I_{thresh}$ **then**

Loop candidate predicted is a false positive.

end if

else if $(L[f_{kc}] \geq 0)$ and $(S[f_{kc}] > d_f)$ **then**

Loop candidate rejected (Negative prediction).

end if

end for

3.6 Evaluation Metrics

3.6.1 SLAM

This section explains the metrics used to evaluate SLAM and loop closure detection. ATE (Absolute Trajectory Error) and RPE (Relative Position Error) are by far, the most common metrics to evaluate accuracy of SLAM [45].

ATE

ATE quantifies the differences between the estimated trajectory and ground-truth poses. Before computing ATE, as both trajectories may not be aligned in the same reference frame, for SLAM drift evaluation we aligned both trajectories using Michael EVO package [21] `-align_origin` option. ATE is defined in several ways: root mean square error ($RMSE$), mean, and median. Assuming that

the sequence for the estimated poses of the trajectory is p_i and from the ground truth trajectory is g_i, where each pose represents a position in the world coordinate system, assuming that both trajectories are aligned and the rotation matrix R is the indentity matrix. The absolute trajectory error will be:

$$ATE_{rmse} = \sqrt{\left(\frac{1}{n}\sum_{n=1}^{n}||p_i - g_i||^2\right)} \tag{3.15}$$

$$ATE_{mean} = \frac{1}{n}\sum_{n=1}^{n}(p_i - g_i) \tag{3.16}$$

$$ATE_{median_{odd}} = \frac{(n+1)^{th}}{2}\ \text{term} \tag{3.17}$$

$$ATE_{median_{even}} = \frac{\left(\frac{n}{2}\right)^{th} + \left(\frac{n+1}{2}\right)^{th}}{2}\text{term}. \tag{3.18}$$

ATE represents the mean difference between the estimated trajectory and the ground truth trajectory for each frame.

RPE

The relative pose error measures the local accuracy of SLAM between over a fixed time interval δ. Consider $P \in SE_3$ as the estimated matrix and $G \in SE_3$ as the ground truth matrix, the relative pose error R_i at timestamp i is:

$$R_i = (G^{-1}G_{i+\delta}) \cdot (P^{-1}P_{i+\delta}) \tag{3.19}$$

For a sequence of n poses, we obtain $m = n - \delta$ relative pose errors, the δ parameter was set to 1 to calculate for all frames. The RPE is usually divided in two components: rotation and translation. Just like in ATE it calculates the RMS and mean error, for demonstration we will just show how to calculate RMS:

$$RPE^{i,\delta}_{trans} = \sqrt{\left(\frac{1}{m}\sum_{i=1}^{m}||trans(R_i)||^2\right)} \tag{3.20}$$

$$RPE^{i,\delta}_{rot} = \sqrt{\left(\frac{1}{m}\sum_{i=1}^{m}||\angle(rot(R_i))||^2\right)} \tag{3.21}$$

For SLAM evaluation, we consider delta equal to 1, to cover all possible pairs in translation and rotation.

3.6.2 Loop Closure

The loop closure assessment was essentially made through curves of precision-recall. Precision measures accuracy of loop detection, high precision indicates that the algorithm when detects a

loop, it is very likely to be an actual loop, and thus has a low rate of false positives. It is not about the number of actual loops that the algorithm misses, but how good the prediction is when it identifies loops.

Recall is the rate of true positive loops, it is a percentage of true loops correctly identified by the algorithm. A high recall indicates that the algorithm is very good at identifying true loops, and it doesn't miss any positive ground-truth loops. In general, these two concepts are a trade-off, and you can adjust the parameters of the algorithm or threshold to optimize one in detriment of the other. The formulas for precision and recall are shown below:

$$Pre = \frac{TP}{TP + FP} \tag{3.22}$$

$$Rec = \frac{TP}{TP + FN} \tag{3.23}$$

To create the curves for precision and recall, we used the feature score retrieved by the loop closure module, to understand for each threshold value how well the algorithm performs. The procedure is as it follows in appendix D.1 and was applied for both protocol algorithms 1 and 2.

4 Experimental Evaluation

The results reported in this thesis have been split into two main groups: evaluation of SLAM and loop closure performance. For SLAM, we considered two datasets (KITTI and UC - POLO2), when using the KITTI dataset, the ground-truth trajectory was obtained through RTK GPS and IMU fusion from the KITTI website. Whereas, ground-truth trajectory for UC - POLO2 dataset was acquired directly from RTK GPS, and we just added orientation from the IMU readings, but no fusion has been performed. The main framework to run each SLAM package was ROS.

4.1 SLAM

To evaluate SLAM performance we have chosen the metrics present in section 3.6.1 (APE and RPE), each trajectory was obtained from the transformation between the `map` and `base_link` referential frames. The sequence selection was based on evaluating direct loops (KITTI sequences 05 and 07) and reverse loops (UC-POLO2 sequence 02).

The SLAM was split into two datasets. In the KITTI dataset, four algorithms were tested under the same environments (KITTI sequences 05 and 07), while for UC - POLO2 dataset the results were generated chosing only the algorithms whose loop closure module was robust enough to perform fully reverse loops (SC LeGO LOAM and SC LIO-SAM).

4.1.1 KITTI Dataset

Default parameters were used, except for the cartographer. The four chosen algorithms to test SLAM were SC LeGO LOAM, LIO-SAM, Cartographer, and HDL-Graph SLAM. Figure 4.1 plots the results for KITTI sequence 5 for each of the four algorithms:

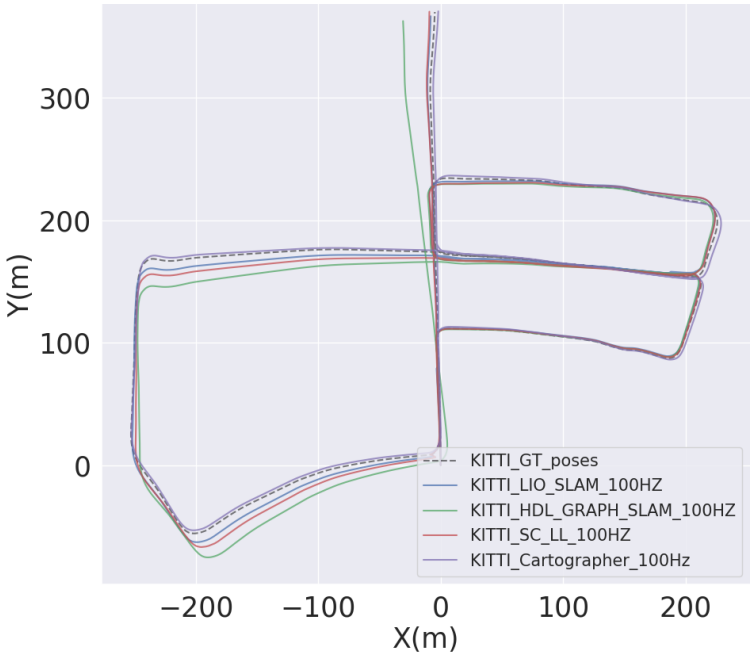


Figure 4.1: KITTI sequence 05 trajectory.

Tables 4.1 and 4.2 shows the APE and RPE metric results for this sequence, regarding each of the four algorithms:

ATE	Cartographer	HDL Graph SLAM	LIO-SAM	SC LeGO LOAM
RMSE (m)	**4.8536**	12.7791	6.4280	8.6908
Mean (m)	**4.4663**	10.6666	5.8731	7.5286

Table 4.1: APE results regarding KITTI sequence 05 trajectory.

RPE	Cartographer	HDL Graph SLAM	LIO-SAM	SC LeGO LOAM
RMSE (m)	0.0328	**0.0193**	0.1725	0.0919
Mean (m)	0.0241	**0.0143**	0.0289	0.2340
RMSE (deg)	0.0057	**0.0008**	0.0025	0.0076
Mean (deg)	0.0043	**0.0013**	0.0014	0.0060

Table 4.2: RPE results regarding KITTI sequence 05 trajectory.

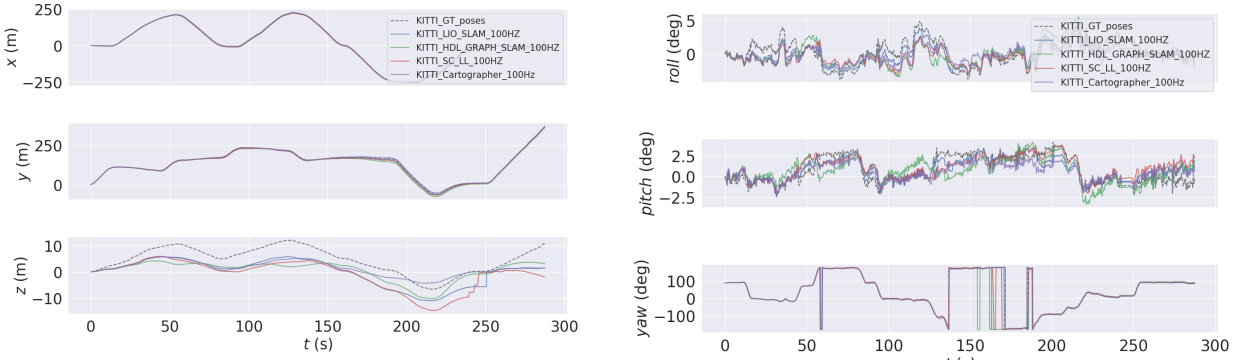
To have a good understanding of the results present in tables 4.1 and 4.2, it is important to acknowledge two main concepts. The APE metric is effective for evaluating the overall global accuracy of a SLAM system by measuring how closely the estimated trajectory aligns with the ground-truth trajectory. In contrast, RPE assesses local accuracy by examining the consistency of poses between consecutive pairs. If a SLAM algorithm performs well in RPE, it signifies strong local estimation.

However, it is important to recognize that even if an algorithm excels in local odometry estimation, as indicated by low RPE values, it can still underperform compared to others with less impressive local accuracy. This discrepancy can be attributed to error accumulation over time and the absence of robust global map optimization algorithms. This exact scenario occurs with HDL-Graph SLAM. While it achieves low RPE errors in translation and rotation, its APE error is the highest among the four metrics. The reason for these low RPE values lies in HDL's utilization of fine-registration techniques for point cloud registration, specifically the NDT method. As previously mentioned, these methods deliver exceptional accuracy in point cloud registration but come with a significant computational cost, particularly in point-to-point or point-to-plane matching. HDL-Graph SLAM employs the g20 algorithm with RANSAC for pose-graph optimization, which proves to be equally effective when compared to LIO-SAM and LeGO LOAM pose-graph optimizers (L-M). However, HDL does have a notable limitation, as it lacks a robust loop closure algorithm. Consequently, over time, global pose estimation tends to degrade, as the system fails to compensate for this absence of loop closure detection, as evidenced by HDL's inability to detect a loop closure at the end of the trajectory in figure 4.1.

LIO-SAM and LeGO LOAM, on the other hand, extract features from the 3D scans, and perform scan-matching through features, and reduce significantly the amount of points to compute transformation between scans, the RPE values achieved high accuracy, as the surface features and edges were well detected urban environments, like KITTI and UC – POLO2 datasets. This type of registration approach tends to perform really well in structured environment and saves computation time. With their ability to estimate local positions accurately, these algorithms possess robust global map optimization techniques, which is why they yielded decent results overall. As depicted in Figure 4.1, both LIO-SAM and SC LeGO LOAM demonstrate the capacity to correct their positions and update the global map trajectory when encountering a loop. This capability is notably absent in HDL-Graph SLAM.

The superior performance of LIO-SAM over SC LeGO LOAM in terms of APE can be attributed to a couple of key factors. First, the sequence in question lacks any reverse loops. In cases with reverse loops, LIO-SAM's loop closure algorithm might perform less effectively than SC LeGO LOAM's, as it may struggle to detect revisited locations from a different viewpoint. Additionally, LIO-SAM's superior RPE values result from its tightly-coupled approach, which integrates IMU data into local odometry estimation. This integration provides better accuracy than the loosely-coupled system employed by SC LeGO LOAM. Due to this lower accumulation of error over time, LIO-SAM outperforms SC LeGO LOAM, although the difference in performance is not substantial.

Analyzing the results for the best APE metric, it is visible that Cartographer produced the best results, however, with a high amount of parameter over-fitting. Even though Cartographer had decent results, the fact that it does not produce a 3D map and is very over-tuned, made it the least



(a) (b)

Figure 4.2: (a) Estimation of XYZ coordinates for KITTI sequence 05; (b) Roll, pitch and yaw estimation for KITTI sequence 05.

desirable option of all.

The next sequence was very a simple one, containing only one direct loop at the end, the KITTI sequence 07 trajectories is present in figure 4.3:

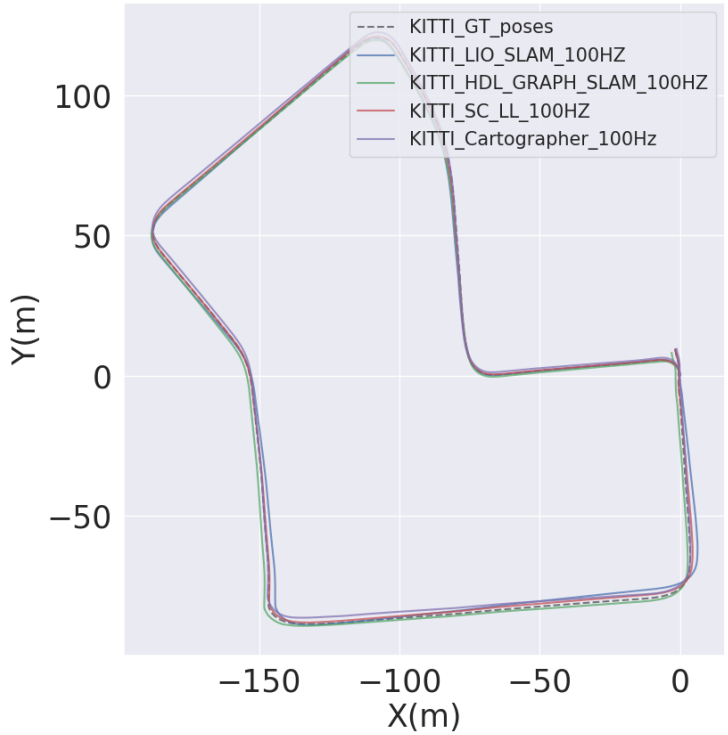


Figure 4.3: KITTI sequence 07 trajectory.

Tables 4.3 and 4.4 show the APE and RPE metric results for sequence 07, regarding each of the four algorithms:

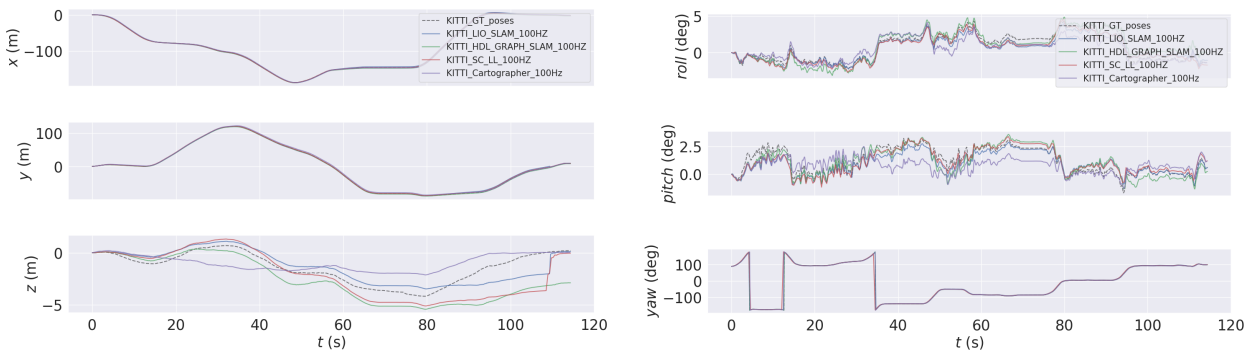
ATE	Cartographer	HDL Graph SLAM	LIO-SAM	SC LeGO LOAM
RMSE (m)	2.4372	2.1602	**1.5166**	1.8050
Mean (m)	2.1416	1.9053	**1.3299**	1.3911

Table 4.3: APE results regarding KITTI sequence 07 trajectory.

RPE	Cartographer	HDL Graph SLAM	LIO-SAM	SC LeGO LOAM
RMSE (m)	0.0264	**0.0195**	0.0755	0.1470
Mean (m)	0.0208	**0.0161**	0.0266	0.0619
RMSE (deg)	0.0021	**0.0012**	0.0020	0.0081
Mean (deg)	0.0017	**0.0009**	0.0015	0.0064

Table 4.4: RPE results regarding KITTI sequence 07 trajectory.

Once again, the results indicate that LIO-SAM and SC LeGO LOAM have a better overall performance, since they perform local and global map optimization. The RPE accuracy of HDL-Graph SLAM and Cartographer is slightly better in comparison to the others, because of using ICP-based registration algorithms, which as said above has high-precision but a lot of computation cost. However, this local accuracy, if not properly compensated by map optimization algorithms, tends to accumulate errors over time. As it can be seen, the ATE error that evaluates the alignment between the estimated and ground-truth trajectory is lower for LIO-SAM and SC LeGO LOAM, which are the algorithms with better global map optimization. Just like in sequence 05, LIO-SAM outperforms SC LeGO LOAM for local odometry estimation because of the way it calculates odometry (tightly-coupled system) and, therefore, is the algorithm with better ATE value, as it accumulates less error during local estimation. The cartographer this time presented worse results than all the results, even though correcting the trajectory at the end, due to not having been over-tuned (this time we did not perform any tuning in Cartographer parameters and the performance is visibly worse). Since the trajectory is short, HDL-Graph SLAM errors were similar to the other algorithms. However, as seen in longer sequences like sequence 05 of the KITTI dataset, this algorithm's global optimization is not the best because it lacks a robust approach to loop closure detection.



(a) (b)

Figure 4.4: (a) Estimation of XYZ coordinates for KITTI sequence 07; (b) Roll, pitch and yaw estimation for KITTI sequence 07.

4.1.2 Polo 2 Campus Dataset

For our second dataset, we have decided to choose algorithms that we found as reasonable to detect reverse loops. SC LeGO LOAM was clearly one we wanted to test, since it was our baseline algorithm and contained a robust loop detector. The second algorithm was LIO-SAM, but with the integration of the same detector of LeGO LOAM (scan context), we found interesting the performance of this algorithm in terms of trajectory precision. The trajectories for SC LIO-SAM and SC LeGO LOAM in the UC - POLO2 sequence 02 are plotted in figure 4.5.

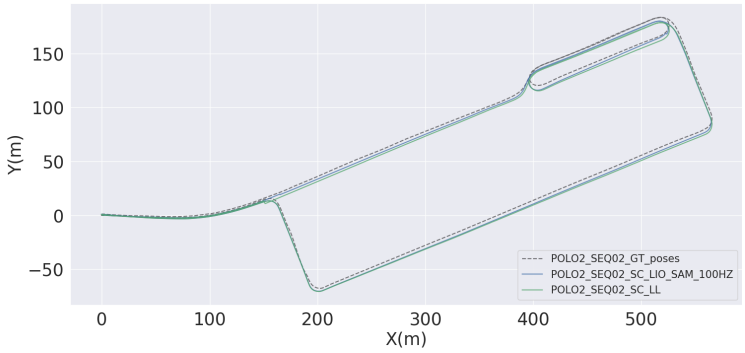


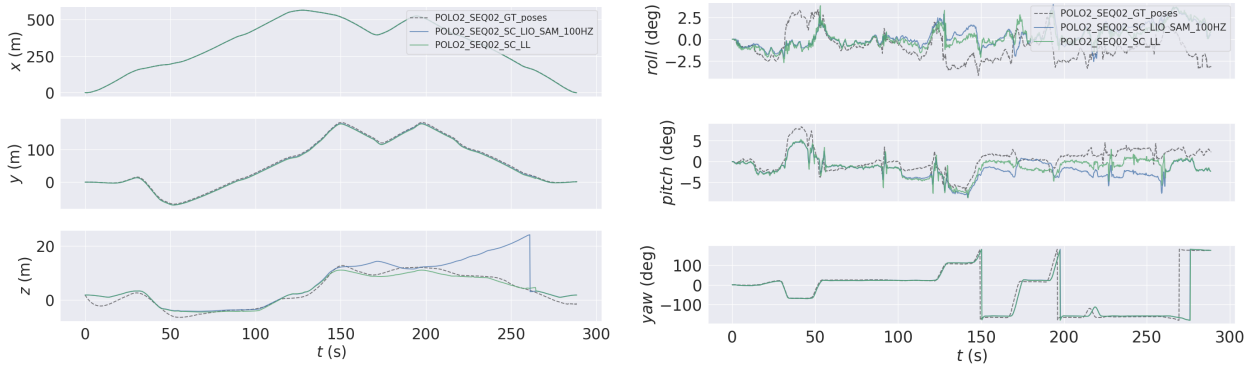
Figure 4.5: UC - POLO2 sequence 02 trajectory.

Table 4.5 shows the results for the APE and RPE metrics for UC - POLO2 sequence 02, regarding each of the two algorithms.

	ATE		RPE trans.		RPE rot.	
	RMS(m)	Mean(m)	RMS(m)	Mean(m)	RMS(deg)	Mean(deg)
SC LL	**4.2403**	**4.1295**	**0.2913**	**0.1962**	0.0235	0.0132
SC LIO-SAM	5.9378	4.7018	0.6230	0.2013	**0.0216**	**0.0107**

Table 4.5: APE and RPE results regarding UC - POLO2 sequence 02 trajectory.

The results on the UC - POLO2 sequence 02 have demonstrated that both algorithms have successfully achieved the intended result for this sequence, as they both identified and corrected revisited places, there is one loop in the roundabout and other at the end, when returning to the initial position. There is an interesting fact that can be withdrawn from the plotted results. The SC LIO-SAM executed almost the entire trajectory perfectly, with minor ATE and RPE errors. However, as soon as it encountered a bump in the road near the end of the sequence, the Z-coordinate spiked. One possible interpretation for this event is that the LIO-SAM integrates IMU readings into odometry estimation, unlike the SC LeGO LOAM, which uses IMU as input for scan matching orientation (both RPE rotations were nearly the same). If this hadn't happened, both the final RPE and ATE of the SC LIO-SAM would have been better, which was not the case. We can see in Figure 4.6 (a) the peak in the Z-coordinate of the SC LIO-SAM, which eventually manages to detect a loop and correct its position.



(a) (b)

Figure 4.6: (a) Estimation of XYZ coordinates for UC - POLO2 sequence 02; (b) Roll, pitch and yaw estimation for UC - POLO2 sequence 02.

4.2 Loop Closure

So far, we have been evaluating the trajectory and explaining the algorithm's performance without actually seeing the output map. Therefore, having a visualization of the map constructed instead of only the trajectory gives a much more illustrative demonstration of where loop closure has occurred. Having said that, we divided loop closure into two sections: The first is dedicated to understanding some hyper-parameter influence in the map correction, providing insights into the map constructed based on different parameter configurations, for example, with and without loop closure, bag rate, mapping interval. The second one is exclusively aimed at evaluating the loop closure algorithms in terms of performance.

4.2.1 Hyper parameters - map correction

We have chosen UC- POLO2 sequences 01 and 02, using a single algorithm (SC LeGO LOAM), to test and comprehend the influence of each parameter in the overall map creation and pose estimation.

Hyper parameters

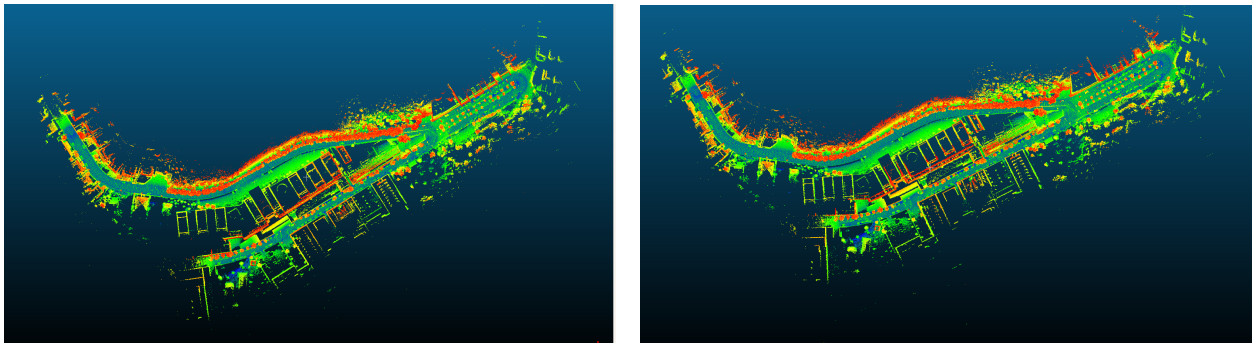
Before we go into the results, let's first explain each parameter individually. The hyper parameters were: `bagrate frequency`, `mapping interval`. The bagrate frequency tells us how fast we want the algorithm to read the data, if this parameter is set at `1x` it means that is equal to real-time. The mapping interval, is for local map optimization and tells us how fast SC LeGO LOAM saves keyframes and corrects the local odometry. A higher value of mapping interval will have more time to update the pose estimated by the odometry module, but skips more frames. On the other hand, a lower value for this parameter will not produce accurate results, since it will update almost at the same time as the odometry module (it is important that the mapping module is at a lower rate than the odometry module).

For every sequence, we changed the loop closure flag. It indicates if the algorithm performs global map optimization or not. For each result obtained, the ICP fit score used was the same. ICP fitscore is the protocol used by SC LeGO LOAM to analyze if the predictive candidate from SC is a true candidate or not (uses a 0.5 SC threshold + ICP), an alternative approach would be to use a much lower threshold value for SC (about 0.1) and remove ICP from the equation. After some testing, we selected a value that gives overall good results (ICP fitscore = 0.8).

First parameter configuration

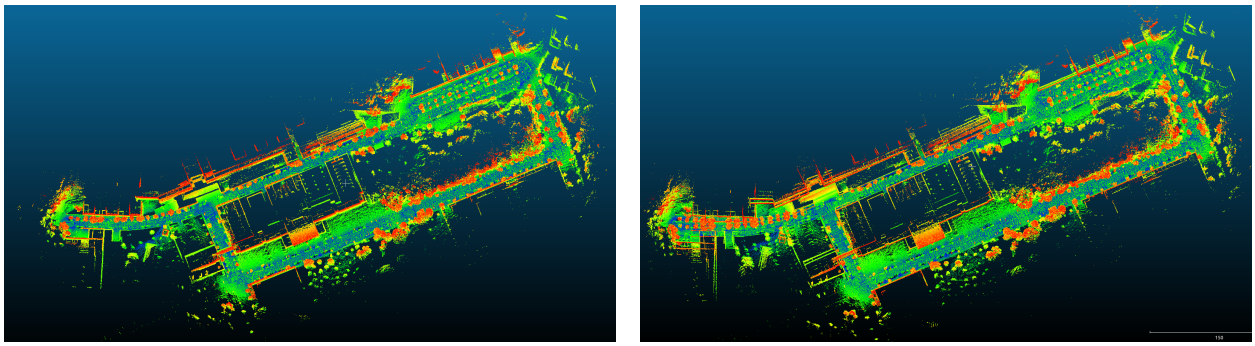
`Defaults parameters: Mapping 0.3, bagrate 1x`

This configuration allowed us to understand the influence of the loop closure integration in the global map optimization.



(a) (b)

Figure 4.7: UC - POLO2 sequence 01 map with (a) and without loop closure (b). Default parameters: `bagrate 1x, mapping 0.3`.



(a) (b)

Figure 4.8: UC - POLO2 sequence 02 map with (a) and without loop closure (b). Default parameters: `bagrate 1x, mapping 0.3`.

From UC - POLO2 sequence 01 (presented in figure 4.7 (a) and (b)), we can see that the algorithm, even without loop closure, achieves great results in the map, as the algorithm did not accumulate much error and was able to compensate using only local map optimization, the fact that the features extracted are extremely perceptible, makes the algorithm perform really well even in the absence of a loop closure algorithm. For sequence 02, in figure 4.8 (b) the map despite being decent almost the whole trajectory, it is seen right at the end, the absence of an existing loop closure algorithm that would correct the final position, figure 4.8 (a) shows a correction of it.

`Mapping changed from 0.3 to 0.2, bagrate 1x`: We decided to decrease the mapping value to see if sequence 01 in figure 4.7 (b) would still produce a coherent map:

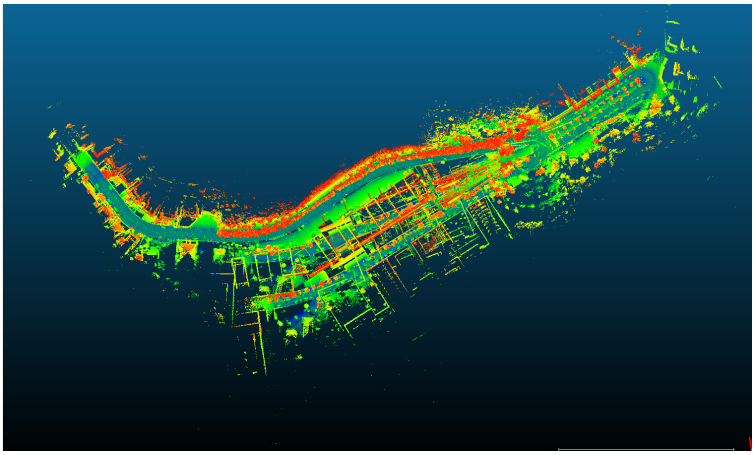


Figure 4.9: UC - POLO2 sequence 01 map without loop closure. Parameters: `bagrate 1x, mapping 0.2`.

It is noticeable how worse the map was by changing this parameter. The algorithm did not have time to optimize accurately the estimated positions, resulting in more accumulated drift in the trajectory.

`Mapping changed from 0.3 to 0.5, bagrate 1x`: The next step, was to increase the mapping value for sequence 02 and see if it would produce a decent map without loop closure, as it would have more time to compute optimizations in the local map.

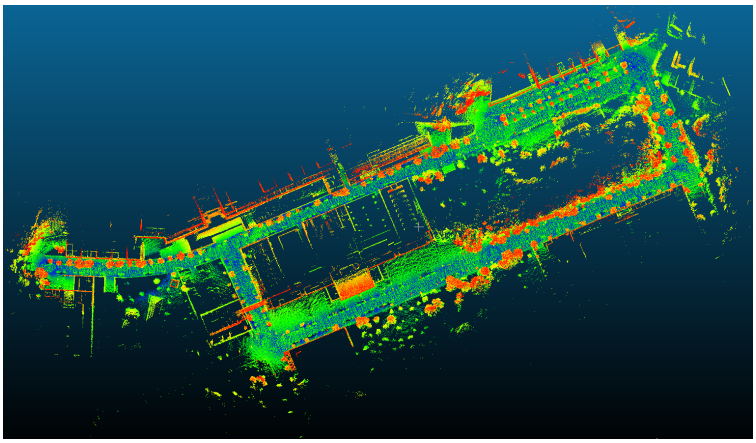


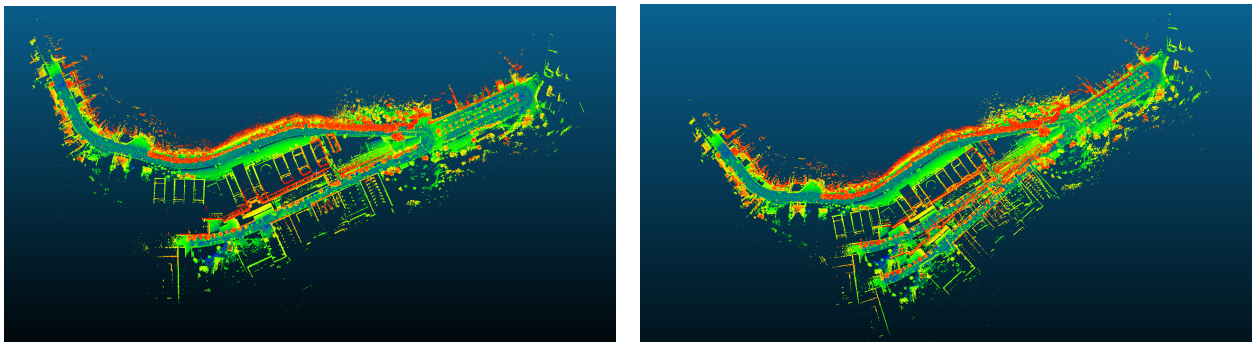
Figure 4.10: UC - POLO2 sequence 02 map without loop closure. Parameters: `bagrate 1x, mapping 0.5`.

The algorithm as we predicted achieved a really great map result without global map optimization.

Our last parameter change was increasing the bagrate for both sequences. Considering the best parameters of mapping for sequence 01 and 02 without loop closure, were respectively 0.3 and 0.5, we doubled the bagrate for each.

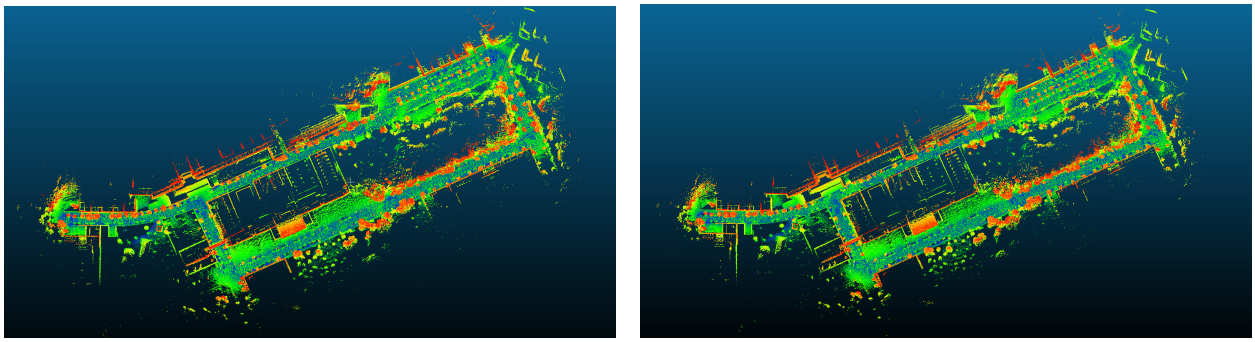
Sequence 01 parameters: `Mapping 0.3, bagrate 2x`

Sequence 02 parameters: `Mapping 0.5, bagrate 2x`



(a) (b)

Figure 4.11: UC - POLO2 sequence 01 maps with (a) and without loop closure (b). Parameters: `bagrate 2x, mapping 0.3`.



(a) (b)

Figure 4.12: UC - POLO2 sequence 02 maps with (a) and without loop closure (b). Parameters: `bagrate 2x, mapping 0.5`.

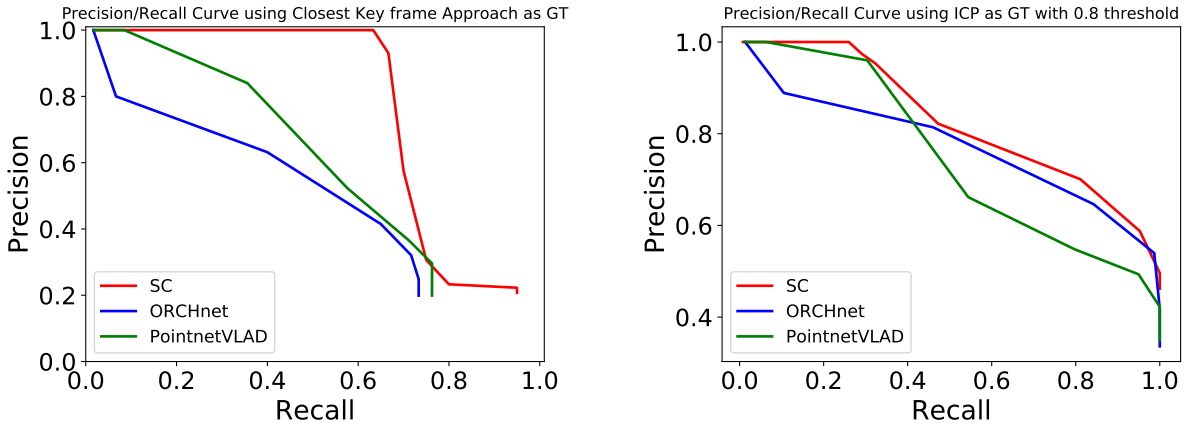
Taking a closer look at these last results, we observed that with a higher bagrate, sequence 01 suffered a lot, as the positions without an external global map optimizer degraded significantly and the result is the map in figure 4.11 (b). On the contrary, the map produced with loop closure is consistent and optimized. In sequence 02, both figures (with and without loop closure) were able to produce a decent final map. The increase of the bagrate presented difficulties, because this parameter will affect the processing time, the mapping is still the same (keyframes saved are the same), but if the machine does not have sufficient computation at this rate, it won't finish the calculations for local map optimization and map will degrade.

4.2.2 Scan Context evaluation

Our last results section was intended to test the loop closure algorithm exclusively, without taking into consideration the trajectory and map. For that purpose, we selected two sequences (KITTI sequence 05 and UC - POLO2 sequence 02). First, we were able to compare SC with two more place recognition techniques (ORCHnet and PointnetVLAD) through a file-based integration. The

second, since we did not own files for it, just SC was evaluated.

The curves of precision and recall for KITTI sequence 05 using protocol A and B are plotted in figure 4.13



(a) (b)

Figure 4.13: (a) Precision-recall curve for KITTI sequence 05 using protocol A (closest keyframe); (b) Precision-recall curve for KITTI sequence 05 using protocol B ($\text{ICP}_{thresh} = 0.8$)

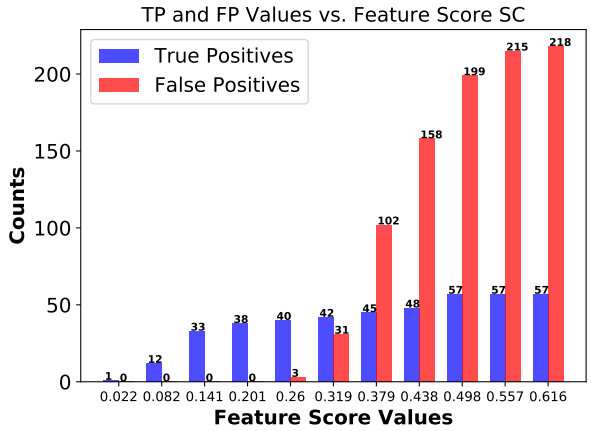
Before starting to discuss the results, it is important to mention the feature score ranging values for each place recognition algorithm.

$$SC_{min} = 0.022 \qquad ORCHnet_{min} = 0.043 \qquad Point.VLAD_{min} = 0.092$$

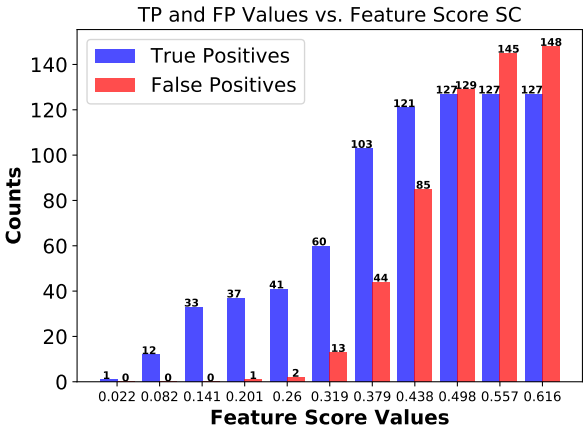
$$SC_{max} = 0.616 \qquad ORCHnet_{max} = 1.135 \qquad Point.VLAD_{max} = 0.881$$

Analyzing the results from figure 4.13a, we can clearly understand that SC outperforms both ORCHnet and PointnetVLAD, as it maintains a good prediction until a certain feature score threshold. Figure 4.14a shows that SC under a thresh = 0.2 has only true positives according to protocol A. Above this score, it starts failing to detect correctly a true positive. PointnetVLAD works well but with a lower threshold value of 0.15, but from there his precision starts falling down linearly, which is not ideal for an algorithm, as it does not distinguish well what is a true positive and what isn't. ORCHnet is similar to PointnetVLAD in terms of linearity in the curve, but performs worse than the others since from the beginning it starts failing its predictions, acquiring false positives with low feature score values, showing that even with low values the algorithm prediction does not match with the actual value. The recall values of the three algorithms are different, while SC presents a steep curve, around 0.7 recall, PointnetVLAD starts decreasing linearly around 0.2 and 0.5 and ORCHnet around 0.1, showing that their accurate true positive rate is very low compared to SC.

Using protocol B (figure 4.13b), SC and PointnetVLAD perform comparably until a certain threshold with high predictive accuracy, while ORCHnet again starts decreasing from the beginning, however, it performs better than PointnetVLAD around 0.4 recall. In figure 4.14b, it can be seen that SC predicts well again until threshold = 0.2.



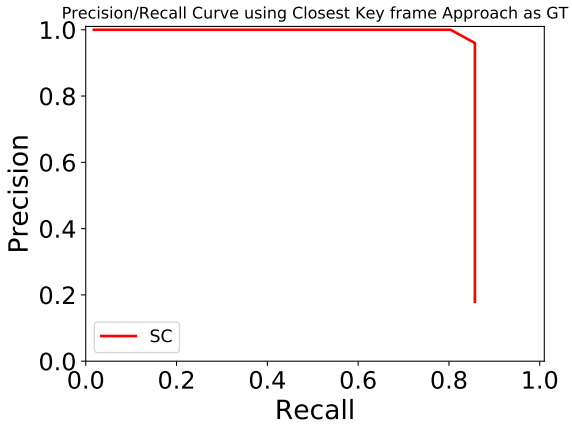
(a) SC: Number of TP and FP using protocol A in KITTI sequence 05. According to this protocol, there are 60 positive values for ground-truth data.



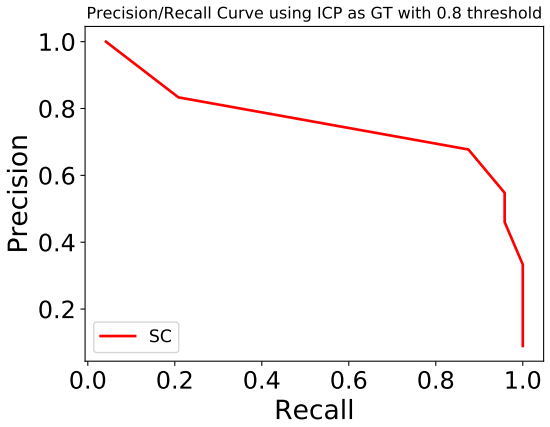
(b) SC: Number of TP and FP using protocol B in KITTI sequence 05. According to this protocol, there are 127 positive values for ground-truth data.

Figure 4.14

The results for SC in UC - POLO2 sequence 02 are plotted in figures 4.15a and 4.15b. Once again, SC demonstrated a solid performance in protocol A, being able to detect flawlessly until a threshold equal to 0.33 (figure 4.16a). Whereas in protocol B it detects with an efficiency of around 80% with a threshold of 0.2 (figure 4.16b).

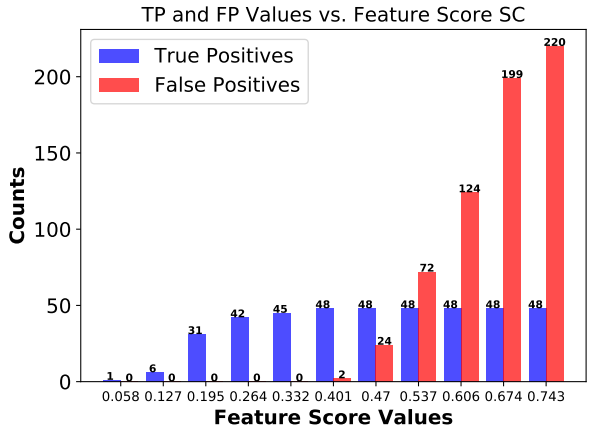


(a)

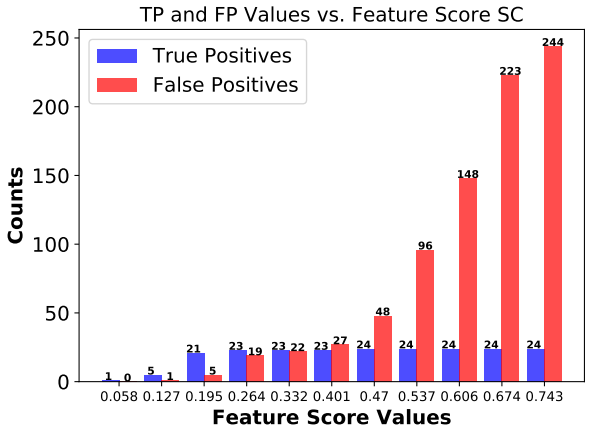


(b)

Figure 4.15: (a) Precision-recall curve for UC - POLO2 sequence 02 using protocol A (closest keyframe); (b) Precision-recall curve for UC - POLO2 sequence 02 using protocol B ($ICP_{thresh} = 0.8$)



(a) SC: Number of TP and FP using protocol A in UC - POLO2 sequence 02. According to this protocol, there are 56 positive values for ground-truth data.



(b) SC: Number of TP and FP using protocol B in UC - POLO2 sequence 02. According to this protocol, there are 24 positive values for ground-truth data.

Figure 4.16

5 Conclusion

SLAM takes an important role in autonomous driving applications, as it provides the solution for an autonomous vehicle to navigate through an unknown environment without an *a-priori* map. The aim of this work was to perform a comprehensive study on 3D outdoor SLAM. With the help of two datasets, KITTI and UC – POLO2, we were able to validate and understand the strengths and weaknesses of each algorithm based on its trajectory and map correction, providing us with a significant learning experience throughout the process. Alongside the SLAM, we investigated in detail the performance and influence of algorithms for global map correction (loop closure).

Analyzing the results experienced when performing a benchmark in SLAM, a few conclusions have been withdrawn. From the four algorithms tested, the ones that presented better results in terms of trajectory with default parameters were clearly LIO-SAM and SC LeGO LOAM. Cartographer despite the good results, had immense overturning and was not able to construct a 3D map. It is also important to note that, matching-based algorithms, like the ones used in Cartographer and HDL-Graph SLAM achieved slightly higher accurate results than the feature-based ones (LIO-SAM and SC LeGO LOAM), but at a considerable computation expense. However, this feature-based methods were able to compensate through global map optimization, reducing significantly the error.

As previously mentioned, LIO-SAM demonstrates superior performance in scenarios featuring smooth, direct loops, primarily due to its more accurate local odometry estimation. However, our evaluation, notably in the UC-POLO2 sequence 02, revealed limitations in the algorithm's performance. This limitation arises from the fusion of IMU data with LiDAR odometry to enhance pose estimations. LIO-SAM struggled when encountering road irregularities, such as bumps, which adversely affected the pose estimation, particularly in the Z coordinate. In contrast, SC LeGO LOAM, which integrates IMU data as a prior orientation for scan matching, proved to be more robust in these challenging conditions. This approach maintains a clear separation between IMU-based orientation adjustments and the position estimation (X,Y,Z), minimizing the impact of road disturbances.

Additionally, for our assessment of the POLO2 dataset, we opted to utilize a variant of SC LIO-SAM. This choice was driven by the algorithm's robustness in loop detection, particularly its ability to handle rotations and effectively recognize revisited places from different viewpoints. This capability was notably absent in the previous ICP-based Euclidean distance place recognition

module. In summary, although LIO-SAM excels in certain scenarios, its susceptibility to road irregularities became evident during our evaluations. SC LeGO LOAM, with its segregated IMU integration strategy, offered more reliable performance under such conditions. Furthermore, the choice to employ a variant of SC LIO-SAM for the POLO2 dataset underscores the importance of a robust loop detection mechanism capable of handling different perspectives on revisited locations.

In loop detection, SC clearly outperformed both ORCHnet and PointnetVLAD in both protocols, achieving positive results in its predictions. The presence of a loop closure mechanism in SLAM has been demonstrated to be indispensable in mitigating the detrimental errors accumulated by local SLAM. While local map optimization and odometry may refine the trajectory on a small scale scenario, they do not address the issue of error propagation. Our results suggest that even local optimization methods with high competence may not fully maintain long-term accuracy. The incorporation of a global map correction mechanism serves as a protective measure against cumulative errors that could endanger the reliability of autonomous navigation.

5.1 Future Work

After reviewing and comparing SLAM algorithms, we noticed flaws in terms of odometry estimation for both SC LeGO LOAM and LIO-SAM. LIO-SAM was able to perform better in scenarios with no bumps on the road, while SC LeGO LOAM due to its characteristic of not using IMU to estimate XYZ position, was able to outperform LIO-SAM exclusively in that type scenarios. We purpose a better way of fusing both algorithms (tighly-coupled and loosely-coupled) so that they complement each other and produce a final pose estimation robust enough to handle both scnenarios. In terms of dataset, we noticed that despite RTK GPS in open clear-sky view spaces that are not covered by buildings or other structures is able to achieve a precision of centimenters, in areas with objects occlusions might not be the best reliable approach to gather ground-truth data. Performing a fusion of the GPS data with IMU or odometry might bring a effective way, specially on these type of environments.

6 Bibliography

[1] Loes Aarts and Gerben Feddes. European truck platooning challenge. In *Proceedings of the HVTT14: International Symposium on Heavy Vehicle Transport Technology, Rotorua, New Zealand*, pages 15–18, 2016. 2

[2] Sameer Agarwal, Keir Mierle, and The Ceres Solver Team. Ceres Solver, 3 2022. 14

[3] Devrim Akca. Matching of 3d surfaces and their intensities. *ISPRS Journal of Photogrammetry and Remote Sensing*, 62(2):112–121, 2007. 9

[4] Devrim Akca. Co-registration of surfaces by 3d least squares matching. *Photogrammetric Engineering & Remote Sensing*, 76(3):307–318, 2010. 9

[5] Josep Aulinas, Yvan Petillot, Joaquim Salvi, and Xavier Lladó. The slam problem: a survey. *Artificial Intelligence Research and Development*, pages 363–371, 2008. 6

[6] T. Barros, L. Garrote, P. Conde, M. J. Coombes, C. Liu, C. Premebida, and U. J. Nunes. Orchnet: A robust global feature aggregation approach for 3d lidar-based place recognition in orchards, 2023. 5, 16

[7] P.J. Besl and Neil D. McKay. A method for registration of 3-d shapes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14(2):239–256, 1992. 8

[8] Keshav Bimbraw. Autonomous cars: Past, present and future a review of the developments in the last century, the present scenario and the expected future of autonomous vehicle technology. In *2015 12th international conference on informatics in control, automation and robotics (ICINCO)*, volume 1, pages 191–198. IEEE, 2015. 2

[9] C Brenner, C Dold, and N Ripperda. Coarse orientation of terrestrial laser scans in urban environments. *ISPRS journal of photogrammetry and remote sensing*, 63(1):4–18, 2008. 9

[10] Shoubin Chen, Baoding Zhou, Changhui Jiang, Weixing Xue, and Qingquan Li. A lidar/visual slam backend with loop closure detection and graph optimization. *Remote Sensing*, 13(14):2720, 2021. 14

[11] Xieyuanli Chen, Thomas Läbe, Andres Milioto, Timo Röhling, Olga Vysotska, Alexandre Haag, Jens Behley, and Cyrill Stachniss. Overlapnet: Loop closing for lidar-based slam. *arXiv preprint arXiv:2105.11344*, 2021. 5, 16

[12] Liang Cheng, Song Chen, Xiaoqiang Liu, Hao Xu, Yang Wu, Manchun Li, and Yanming Chen. Registration of laser scanning point clouds: A review. *Sensors*, 18(5):1641, 2018. 8

[13] Anweshan Das, Jos Elfring, and Gijs Dubbelman. Real-time vehicle positioning and mapping using graph optimization. *Sensors*, 21(8):2815, 2021. 6

[14] César Debeunne and Damien Vivet. A review of visual-lidar fusion based simultaneous localization and mapping. *Sensors*, 20(7):2068, 2020. xii, 6, 7, 8

[15] Renaud Dubé, Andrei Cramariuc, Daniel Dugas, Juan Nieto, Roland Siegwart, and Cesar Cadena. Segmap: 3d segment mapping using data-driven descriptors. *arXiv preprint arXiv:1804.09557*, 2018. 16

[16] Yang Dam Eo, Mu Wook Pyeon, Sun Woong Kim, Jang Ryul Kim, and Dong Yeob Han. Coregistration of terrestrial lidar points by adaptive scale-invariant feature transformation with constrained geometry. *Automation in Construction*, 25:49–58, 2012. 9

[17] Martin A Fischler and Robert C Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6):381–395, 1981. 8

[18] Udo Frese. A discussion of simultaneous localization and mapping. *Autonomous Robots*, 20:25–42, 2006. 3

[19] Andreas Geiger, Philip Lenz, Christoph Stiller, and Raquel Urtasun. Vision meets robotics: The kitti dataset. *The International Journal of Robotics Research*, 32(11):1231–1237, 2013. 5, 33, 34

[20] Armin Gruen and Devrim Akca. Least squares 3d surface and curve matching. *ISPRS Journal of Photogrammetry and Remote Sensing*, 59(3):151–174, 2005. 9

[21] Michael Grupp. evo: Python package for the evaluation of odometry and slam. `https://github.com/MichaelGrupp/evo`, 2017. 42

[22] Ayman Habib, Mwafag Ghanma, Michel Morgan, and Rami Al-Ruzouq. Photogrammetric and lidar data registration using linear features. *Photogrammetric Engineering & Remote Sensing*, 71(6):699–707, 2005. 9

[23] Dirk Hahnel, Wolfram Burgard, Dieter Fox, and Sebastian Thrun. An efficient fastslam algorithm for generating maps of large-scale cyclic environments from raw laser range measurements. In *Proceedings 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003)(Cat. No. 03CH37453)*, volume 1, pages 206–211. IEEE, 2003. 14, 15

[24] Li He, Xiaolong Wang, and Hong Zhang. M2dp: A novel 3d point cloud descriptor and its application in loop closure detection. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 231–237, 2016. 16

[25] Yuqing He and Yuangang Mei. An efficient registration algorithm based on spin image for lidar 3d point cloud models. *Neurocomputing*, 151:354–363, 2015. 9

[26] Wolfgang Hess, Damon Kohler, Holger Rapp, and Daniel Andor. Real-time loop closure in 2d lidar slam. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1271–1278, 2016. 13

[27] Shoudong Huang and Gamini Dissanayake. Convergence and consistency analysis for extended kalman filter based slam. *IEEE Transactions on Robotics*, 23(5):1036–1049, 2007. 6

[28] Jen Jer Jaw and Tzu Yi Chuang. Feature-based registration of terrestrial and aerial lidar point clouds towards complete 3d scene. In *29th Asian Conference on Remote Sensing 2008, ACRS 2008*, pages 1295–1300, 2008. 9

[29] Michael Kaess, Hordur Johannsson, Richard Roberts, Viorela Ila, John J Leonard, and Frank Dellaert. isam2: Incremental smoothing and mapping using the bayes tree. *The International Journal of Robotics Research*, 31(2):216–235, 2012. 13

[30] 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, 2018. xii, 5, 13, 15, 18, 20, 21, 23, 25

[31] Kenji Koide, Jun Miura, and Emanuele Menegatti. A portable three-dimensional lidar-based system for long-term and wide-area people behavior measurement. *International Journal of Advanced Robotic Systems*, 16(2):1729881419841532, 2019. 14

[32] Rainer Kümmerle, Giorgio Grisetti, Hauke Strasdat, Kurt Konolige, and Wolfram Burgard. g 2 o: A general framework for graph optimization. In *2011 IEEE International Conference on Robotics and Automation*, pages 3607–3613. IEEE, 2011. 14

[33] Jaebin Lee, Kiyun Yu, Yongil Kim, and Ayman F Habib. Adjustment of discrepancies between lidar data strips using linear features. *IEEE Geoscience and remote sensing letters*, 4(3):475–479, 2007. 9

[34] Feng Lu and Evangelos Milios. Globally consistent range scan alignment for environment mapping. *Autonomous robots*, 4:333–349, 1997. 6, 7

[35] Simon Lynen, Markus W Achtelik, Stephan Weiss, Margarita Chli, and Roland Siegwart. A robust and modular multi-sensor fusion approach applied to mav navigation. In *2013 IEEE/RSJ international conference on intelligent robots and systems*, pages 3923–3929. IEEE, 2013. 13

[36] Martin Magnusson. *The three-dimensional normal-distributions transform: an efficient representation for registration, surface analysis, and loop detection.* PhD thesis, Örebro universitet, 2009. 8

[37] Ellon Mendes, Pierrick Koch, and Simon Lacroix. Icp-based pose-graph slam. In *2016 IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR)*, pages 195–200, 2016. 15

[38] Oriol Monserrat and Michele Crosetto. Deformation measurement using terrestrial laser scanning data and least squares 3d surface matching. *ISPRS Journal of Photogrammetry and Remote Sensing*, 63(1):142–154, 2008. 9

[39] Michael Montemerlo, Sebastian Thrun, Daphne Koller, Ben Wegbreit, et al. Fastslam 2.0: An improved particle filtering algorithm for simultaneous localization and mapping that provably converges. In *IJCAI*, volume 3, pages 1151–1156, 2003. 6

[40] Jorge J Moré. The levenberg-marquardt algorithm: implementation and theory. In *Numerical analysis: proceedings of the biennial Conference held at Dundee, June 28–July 1, 1977*, pages 105–116. Springer, 2006. 12

[41] Danial Nakhaeinia, S Hong Tang, SB Mohd Noor, and O Motlagh. A review of control architectures for autonomous navigation of mobile robots. *International Journal of the Physical Sciences*, 6(2):169–174, 2011. 3

[42] Ouster. Os1: High resolution imaging lidar. Online, December 2019. Accessed in September 14-th 2023. 29

[43] Gaurav Pandey, James R McBride, and Ryan M Eustice. Ford campus vision and lidar data set. *The International Journal of Robotics Research*, 30(13):1543–1552, 2011. 33

[44] William H Press and Saul A Teukolsky. Savitzky-golay smoothing filters. *Computers in Physics*, 4(6):669–672, 1990. 38

[45] David Prokhorov, Dmitry Zhukov, Olga Barinova, Konushin Anton, and Anna Vorontsova. Measuring robustness of visual slam. In *2019 16th International Conference on Machine Vision Applications (MVA)*, pages 1–6. IEEE, 2019. 42

[46] Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, Andrew Y Ng, et al. Ros: an open-source robot operating system. In *ICRA workshop on open source software*, volume 3, page 5. Kobe, Japan, 2009. 6

[47] Tahir Rabbani, Sander Dijkman, Frank van den Heuvel, and George Vosselman. An integrated approach for modelling and global registration of point clouds. *ISPRS journal of Photogrammetry and Remote Sensing*, 61(6):355–370, 2007. 9

[48] Ravi Raj and Andrzej Kos. A comprehensive study of mobile robot: History, developments, applications, and future research perspectives. *Applied Sciences*, 12(14):6951, 2022. 2

[49] Timo Röhling, Jennifer Mack, and Dirk Schulz. A fast histogram-based similarity measure for detecting loop closures in 3-d lidar data. In *2015 IEEE/RSJ international conference on intelligent robots and systems (IROS)*, pages 736–741. IEEE, 2015. 16

[50] Radu Bogdan Rusu, Nico Blodow, and Michael Beetz. Fast point feature histograms (fpfh) for 3d registration. In *2009 IEEE international conference on robotics and automation*, pages 3212–3217. IEEE, 2009. 9

[51] 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, 2018. xii, 9, 10, 13, 15, 26

[52] 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. 13

[53] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. A real-time algorithm for mobile robot mapping with applications to multi-robot and 3d mapping. In *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No. 00CH37065)*, volume 1, pages 321–328. IEEE, 2000. 6

[54] Sebastian Thrun et al. Robotic mapping: A survey. 2002. 6

[55] Sebastian Thrun, Mike Montemerlo, Hendrik Dahlkamp, David Stavens, Andrei Aron, James Diebel, Philip Fong, John Gale, Morgan Halpenny, Gabriel Hoffmann, et al. Stanley: The robot that won the darpa grand challenge. *Journal of field Robotics*, 23(9):661–692, 2006. 2

[56] TOPCON. Hiper pro. Online, September 2023. Accessed in September 14-th 2023. xii, 32

[57] Mikaela Angelina Uy and Gim Hee Lee. Pointnetvlad: Deep point cloud based retrieval for large-scale place recognition. *CoRR*, abs/1804.03492, 2018. 16

[58] Jessica Van Brummelen, Marie O'Brien, Dominique Gruyer, and Homayoun Najjaran. Autonomous vehicle perception: The technology of today and tomorrow. *Transportation research part C: emerging technologies*, 89:384–406, 2018. 2

[59] Wolfgang Von Hansen, Hermann Gross, and Ulrich Thoennessen. Line-based registration of terrestrial and airborne lidar data. *Int. Arch. Photogramm. Remote Sens. Spat. Inf. Sci*, 37:161–166, 2008. 9

[60] Ying Wang, Zezhou Sun, Cheng-Zhong Xu, Sanjay Sarma, Jian Yang, and Hui Kong. Lidar iris for loop-closure detection, 2020. 40

[61] T Weber, R Hänsch, and O Hellwich. Automatic registration of unordered point clouds acquired by kinect sensors using an overlap heuristic. *ISPRS Journal of Photogrammetry and Remote Sensing*, 102:96–109, 2015. 9

[62] Martin Weinmann et al. *Reconstruction and analysis of 3D scenes*, volume 1. Springer, 2016. 9

[63] Tao Wu, Hao Fu, Bokai Liu, Hanzhang Xue, Ruike Ren, and Zhiming Tu. Detailed analysis on generating the range image for lidar point cloud processing. *Electronics*, 10(11):1224, 2021. 10

[64] XSENS. Mti user manual: Mti 10-series and mti 100-series 5th generation. Online, February 2020. Accessed in September 14-th 2023. 28

[65] Guanghui Xue, Jinbo Wei, Ruixue Li, and Jian Cheng. Lego-loam-sc: An improved simultaneous localization and mapping method fusing lego-loam and scan context for underground coalmine. *Sensors*, 22(2):520, 2022. xii, 10, 18, 19

[66] Haoyang Ye, Yuying Chen, and Ming Liu. Tightly coupled 3d lidar inertial odometry and mapping. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 3144–3150. IEEE, 2019. 13

[67] Anestis Zaganidis, Alexandros Zerntev, Tom Duckett, and Grzegorz Cielniak. Semantically assisted loop closure in slam using ndt histograms. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4562–4568, 2019. 16

[68] Ji Zhang and Sanjiv Singh. Loam: Lidar odometry and mapping in real-time. In *Robotics: Science and Systems*, volume 2, pages 1–9. Berkeley, CA, 2014. 9, 12, 13, 15

Appendix A

ROS

A.1 ROS Introduction

Robot Operating System (ROS) is an open source framework that provides tools, libraries that enable to efficiently create, simulate and deploy diverse robotic applications. The core objective behind creating ROS (Robot Operating System) was to establish a versatile and adaptable platform dedicated to the creation and management of robotic systems. With its emphasis on modularity and scalability, ROS accelerates the progession of robotics, by promoting standardized interfaces and smooth interaction among robotic components and software modules.

One application that ROS can support is SLAM, as already has been discussed, SLAM is essential for autonomous navigation, exploration etc. The main goal here is to produce a 3D map that is consistent with the surrounding environment, and allows a vehicle to navigate inside it. ROS works as a middleware running on top of an operating system, and allows different nodes/programs to communicate with each other through messages, services and actions. The interface with sensors, actuators, drivers, controllers and algorithms is already present in ROS.

ROS Core

ROS `Core` is a group of nodes and programs that are prerequisites of a ROS-based system. For the nodes to communicate, ROS Core must be running. When initializing roscore, three processes are created: ROS `Master`, ROS `Parameter Server` and `rosout`. ROS `Master` provides the registration of names for ROS graph resources, tracking publishers and subscribers to topics, as well as services, enabling ROS nodes to locate and communicate with one another. The communication is peer-to-peer. Since the nodes to communicate require a master, the ROS system is centralized.

Alongside the initialization of ROS Master, the `Parameter Server` is a centralized storage system used to manage configuration parameters and settings for ROS nodes. Finally, `rosout` is a standard ROS topic responsible for publishing logging messages.

Graph Resources

ROS graph resources includes: Nodes, Parameters, Topics and Services. ROS `Topics` serve as designated conduits through which nodes communicate by exchanging messages. Every ROS topic is characterized by a specific data format determined by the type of ROS `Message` employed for its publication. Consequently, nodes possess the capability to exclusively accept messages that conform to the corresponding data type. If nodes require the ability to participate in remote procedure calls, which involves receiving responses to requests, it is advisable to utilize ROS `Services` as an alternative approach.

ROS Communication Protocols

There are two principal ways to communicate between nodes. Nodes can communicate through ROS `Topics` and ROS `Services`.

The communication protocol employed in ROS `Topics` is based on publisher-subscriber model. When a node publishes data to a specific topic, it sends out messages without being aware of which nodes, if any, are interested in receiving them. Subscribing nodes express their interest in a particular topic. This asynchronous communication approach allows nodes to operate independently and facilitates efficient data exchange.

On the other hand, the communication protocol applied in ROS `Services` is request-response. Nodes interact by sending service requests and receiving corresponding responses. When a node sends a service request, it specifies the desired action, and another node that provides the service processes the request and sends back a response. This direct and synchronous communication approach enables nodes to work together on tasks involving interactions that demand specific actions and corresponding results.

Even though ROS `Topics` and ROS `Services` represent the primary communication mechanisms in ROS, it also supports other higher-level communication mechanism, that combines aspects of both topics and services, ROS `Actions`. Actions are ideal for scenarios where tasks need to be executed over time and involve multiple steps or feedback, for example, in navigation scenarios.

ROS Coordinate frames

ROS has a coordinate frame system that requires configuration according to our robot/vehicle. Frames are organized in a tree, where nodes signify components or sensors within our setup, and the connections denote the relative positioning and orientation between these nodes. This capability enables ROS to consider relative sensor positions while making use of their data.

Appendix B

GPS

B.1 NMEA-0183 protocol message

Below there is a field by field explanation of this data structure.:

```
$GPGGA,hhmmss.ss,llll.ll,a,yyyyy.yy,a,x,xx,x.x,x.x,M,x.x,M,x.x,xxxx

 hhmmss.ss = UTC time
 llll.ll = latitude of position
 a = Northern (N) or Southern(S) Hempishphere
 yyyyy.yy = longitude of position
 a = Eastern (E) or Western (W) Hemishphere
 x = GPS Quality indicator (0=no fix, 1=GPS fix, 2=Dif. GPS fix, 4= RTK fixed,
 5= RTK float)
 xx = number of satellites in use
 x.x = horizontal dilution of precision
 x.x = Antenna altitude above mean-sea-level
 M = units of antenna altitude, meters
 x.x = Geoidal separation
 M = units of geoidal separation, meters
 x.x = Age of Differential GPS data (seconds)
 xxxx = Differential reference station ID
```

Appendix C

Integration in files

To evaluate Scan Context against other algorithms, we used a file-based evaluation. Since the aim was to compare the performance of Scan Context, we used a predetermined set of loop candidates, that was available for the KITTI dataset. The files from where we took the readings contained all the LiDAR frames, 24 loop candidates per frame, and their respective score.

C.1 File-based integration

The algorithms that we chose to compare were PointnetVLAD and ORCHnet and were trained for all the KITTI sequences. The implementation of this procedure was quite simple: we fed to the algorithm inputs of potential loop candidates in the loop detector module, by just knowing the current global frame index and retrieved 5 of the best scores per frame. The most difficult problem we encountered using this approach was the fact that the SLAM algorithm did not concatenate all the pointclouds, which is normal due to the amount of real time processing. The maximum number of pointclouds combined was exactly half of the LiDAR publishing rate (*i.e.*, 5 hz). This represented a problem, because we had to search for a global index in the files and the algorithm was storing a different index to detect loops. The solution for this adversity was to create two buffers with the exact same size, where one of them stores the global index and the other a local index (corresponds to the frame the algorithm uses as input to the loop module). The global index was taken directly from the callback function that keeps track of the pointclouds received in a specific topic, we just added a counter for that matter. The local index was taken directly from the function that saves keyframes and stores descriptors for Scan Context. The idea is the following:

1. Scan context whenever a pointcloud is saved by the SLAM, creates a descriptor for each of them and stores them in a vector. Think this as a stack of descriptors, where the number of descriptors is equal to the number of pointclouds the algorithm concatenates. The problem is that the number of pointclouds saved is different from the number of pointclouds received.

Local buffer stores the indexes of the pointclouds integrated by the algorithm, which is half of the pointclouds received. This is the index that the loop closure module will treat as the current frame.

2. Then inside the callback function, we kept track of the number of pointclouds received, which is equal to the number of 3D scans published by the LiDAR. Here we have the global index.

3. Then, inside the function responsible for saving keyframes, we created two buffers: local buffer and global buffer. The local buffer stores the index of the pointclouds integrated by the algorithm and the global buffer saves the global indexes provenient from the pointcloud callback function.

4. To search for candidate frames in the files of PointnetVLAD and Orchnet, we use the indexes from the global buffer. Whilst Scan context uses the frames present in the local buffer.

5. The buffers are the key for this to work, since the buffers are created at the same time and belong to the same thread. To remap a global index into a local index, we just need to know the exact index in the buffer. Imagine a global buffer G and a local buffer L with the same dimensions, storing the local and global indexes continuously.

   To know the where a global index fits in the local index, let $G[i]$ be the global frame retrieved from the file at index i, and $L[i]$ the frame that represents an actual input for the loop closure module at index i. Then to know the correspondence between the frames from the file and the frames of the loop detector, we do $G[i] = L[i]$.

6. When searching for potential candidates for the current frame in the file, two things can happen: retrieve "-1" or the first 5 global indexes (which are the indexes with the best score).

7. If loop candidate retrieved from the file is "-1" means that there is no loop. Whereas if it retrieves the best 5 indexes, then we perform a search in the global buffer to verify if it contains any of the indexes retrieved. If yes, we convert do $G[i] = L[i]$, where i is the index found in the global buffer. The outputs of these operations is an input to the loop closure module.

Appendix D

Precision-Recall curves

D.1 Implementation of the precision-recall curves

```
Example of how to create a precision and recall curve

N = np.sum(dist[:, 2] <= 4)
dist[:, 1] = np.where((dist[:, 0] >= 0) & (dist[:,2] <=4), 1, 0)
resolution = (score_max-score_min) * 1.0 /10
for i in np.arange(score_min, score_max, resolution):
    for j in range(0, dist.shape[0]):
        if dist[j][3] <= i:
            p = p+1

            if dist[j][1] == 1:
                tp = tp+1
            else:
                fp = fp+1
    re = tp * 1.0 / N
    pr = tp * 1.0 / p
    rec.append(re)
    pre.append(pr)

    plt.plot(rec, pre))
    plt.show()
return rec,pre
```

In the code above, we consider `dist[j][3]` as the feature score for each frame and `dist[j][1]` the condition to be a true positive value, and N the number of ground truth positives.