POLITECNICO DI MILANO
DIPARTIMENTO DI ELETTRONICA, INFORMAZIONE E BIOINGEGNERIA
DOCTORAL PROGRAMME IN INFORMATION TECHNOLOGY

ART-SLAM: A FRAMEWORK FOR REAL-TIME MULTI-SENSOR 3D SLAM

Doctoral Dissertation of:
Matteo Frosi

Supervisor:
Prof. Matteo Matteucci

Tutor:
Prof. Francesco Amigoni

The Chair of the Doctoral Program:
Prof. Luigi Piroddi

2022 – 35th cycle

This work has been possible thanks to my advisor, Matteo, that supported me throughout these three years. Despite him always reprimanding me for being too formal, I will always consider him my mentor, a role which he still continues to play today. Furthermore, he is always ready to talk and discuss things with me, even during the middle of the night. For these and other reasons (including bizarre adventures in America), I am deeply grateful.

I also want to thank all people that supported me during this intense period of time. I am beholden to my mother Sandrina, who helped me during my whole life, and it is she who raised and allowed me to be the person I am now. I thank my aunts, Anna and Angela, that week after week made me laugh and spoiled me in every possible way. Lastly, and this will surprise the reader, I thank my plants, which were always there to make me smile and relax with their wonderful perfume and lush green color.

Abstract

AUTONOMOUS robot navigation represents a core aspect in the field of robotics, due to the many applications (e.g., agriculture and farming and automated driving). Simultaneous Localization and Mapping (SLAM) methods address the problem of constructing a model of the environment surrounding the robot, i.e., the map, while simultaneously estimating its pose within it. In literature, a plethora of SLAM systems have been proposed in the last decade. The created methods, however, greatly differ one from the other, in terms of architecture, data, and adopted frameworks. For this reason, extending or just using a SLAM method that suits specific criteria has become increasingly difficult over the past few years. For this reason, in the thesis, we present a common framework to perform SLAM, by developing multiple novel methods, distinguishable one from the other by the main sensor used and the specific real-world problem tackled. These systems can be summed up as follows: SLAM using a LiDAR as the main sensor, SLAM using both cameras and LiDAR, SLAM exploiting radar information and SLAM aided with information coming from third-party mapping services. All presented methods contribute in multiple ways. First, they improve existing algorithms, to mainly increase accuracy and performance. Then, some of the systems deal with situations that are not fully explored in literature, such as for SLAM where the main sensor is a radar, localization in GNSS-denied environments, and SLAM aided with prior maps. Lastly, the development of a unique framework allows users to adopt a SLAM system suited to their needs. The proposed methods are tested on state-of-the-art datasets, including KITTI and MulRan.

Contents

CHAPTER 1

Introduction

Autonomous robot navigation has been an increasingly studied topic in the last decades. A lot of tasks, which are repetitive or hazardous for humans, may be carried out by means of unmanned agents. Examples include mobile robots for cleaning, agriculture, and farming, automated driving vehicles, robots for delivery and transportation, search and rescue operations, and the list is expanding, especially towards industry-oriented operations.

Robots that perform all these tasks need to be built in a way that allows them to operate, safely and reliably, in complex or unknown environments, based only on the perceptions of their onboard sensors. To achieve this, a robot must be able to create an accurate map of the surrounding environment, while simultaneously estimating its own trajectory with high accuracy. Simultaneous Localization and Mapping (SLAM) methods address the problem of constructing a model of the environment surrounding the robot, i.e., the map, while simultaneously estimating its location within it.

Due to the growing number of applications, in the last decade, the SLAM problem involved an increasing number of researchers and experts, both from the world of academia and industry. The considerable variety of situations (and available hardware) where SLAM can be used, lead to the development of many systems, using different sensors and methodologies.

The first SLAM method to be considered exploited the probabilistic nature of the problem, which can be summarized as follows. The robot is assumed to move in an unknown environment, along a trajectory described by the sequence of random variables $x_{1:T} = \{x_1, \ldots, x_T\}$, where the single variable x_i represent the pose of the robot w.r.t. an initial pose x_0, arbitrarily set. While moving, in correspondence of each pose x_i, the robot acquires a sequence of values $u_{1:T} = \{u_1, \ldots, u_T\}$, which are data from motion sensors, and sensor measurements of the environment $z_{1:T} = \{z_1, \ldots, z_T\}$.

Solving the SLAM problem consists of estimating the posterior probability of the trajectory $x_{1:T}$ of the robot and the map m, given all the measurements plus the initial pose x_0. Probabilistic approaches presented many disadvantages, including poor scalability and lack of consistency.

Over the course of the years, the concept of SLAM moved away from the probabilistic approach, still improved in many recent works, and reached a more mature and modular representation. Instead of treating the SLAM problem as a whole, the state-of-the-art consists in breaking the process into multiple independent modules, while adopting exteroceptive sensors to both estimate the motion of the robot and construct a map of the environment.

The standard architecture of a SLAM system can be summed up as a combination of two main parts, namely the front-end, and the back-end. The front-end processes data coming from one or multiple sensors, and extracts spatial relations between each observation of the robot. Moreover, sensor measurements taken at different moments of time are confronted and used to estimate the motion of the robot, while creating a coarse and rough map of the surrounding environment. The front-end represents the core of the localization aspect of SLAM, as it keeps track of the location of the robot while performing data association using sensor-dependent techniques.

While the front-end provides an initial estimate of the path traveled by the robot, the back-end optimizes it to generate a more accurate trajectory. Optimization takes into account all possible sensor data and constraints, not necessarily associated with the front-end (e.g., a direct measurement coming from a proprioceptive sensor). Moreover, optimization can be either global, if the whole trajectory is adjusted and corrected, or local if only a limited set of estimated poses are considered. Lastly, in the back-end, the map of the environment is fully built and refined, to be later used for other tasks.

The most widely developed type of SLAM system is visual SLAM, where data is gathered by one or multiple cameras. In the last decades, visual SLAM has been actively discussed, because of the numerous advantages camera sensors have. First, a camera is one of the few sensors that were available when localization and mapping, especially in three dimensions,

started interesting the scientific community, many decades ago. Moreover, over the course of the years, cameras have become cheaper, while providing images with increasingly detailed resolution, optimal to perform visual SLAM. Lastly, their setup and configuration on robots are simpler w.r.t. other sensors, making them a viable solution for many applications.

The main goal of visual SLAM systems is to sequentially estimate the motion of the robot (i.e., the movement of the sensors) depending on the perceived movements of pixels in a sequence of two or multiple images. This is achieved in different ways. A possible approach is to detect and track salient points, named keypoints, in the image. All keypoints are then matched and aligned to estimate the relative motion of sensors. This method takes the name of feature-based visual SLAM. Another solution, also known as direct visual SLAM, involves using the entire image. Moreover, a robot can easily localize itself inside a map by checking for previous similar images, an operation also known in the literature as place recognition.

Even if visual SLAM provides accurate results, methods are prone to errors because of the sensitivity of cameras to light changes or a low-textured environment. Furthermore, image analysis, feature extraction, direct association, and similarity search still require a considerable amount of computational resources, especially when dealing with high-resolution images, which are commonly gathered from state-of-the-art sensors.

To overcome some of the problems associated with cameras, such as illumination dependencies, many SLAM systems adopted, instead, the use of laser rangefinders, e.g., LiDARs. As the majority of laser SLAM methods process LiDAR data, this sensor is considered the standard, and it is used in the rest of this thesis to describe and deal with laser SLAM.

By transmitting and receiving the laser beam, a LiDAR uses the Time of Flight (ToF) method to obtain the distance of the scanning point from the center of the sensor, which is almost unaffected by the illumination conditions, different from cameras, and has a long detection range and high precision (even millimeters), in indoor and outdoor scenes.

Laser scanning methods appear to be a cornerstone of both 2D and 3D mapping research. LiDAR sensors deliver point clouds that can be easily interpreted to perform SLAM, by performing a simple scan-to-scan alignment and estimating the transformation that best overlaps two input scans. Mainly thanks to the detailed nature of point clouds derived from laser sensors, which are characterized by a considerable field of view (possibly covering the whole surroundings), LiDAR SLAM methods achieve superior accuracy, and they are generally preferred when precision is a must (e.g., industrial applications). For the same reason, however, laser SLAM systems

are generally computationally demanding, especially when dealing with 3-dimensional data, making the algorithms not suitable for real-time tasks.

As just described, SLAM can be performed both thanks to visual sensors and lasers. Cameras have the main advantage of being plentifully studied in the literature. Even though visual SLAM systems provide accurate results, they have some issues, e.g., the drift of the scale when using only one camera, the poor depth estimation and small range of view, and the sparsity of the reconstructed maps. On the other hand, laser sensors have very good accuracy in ranging and, as a consequence, in mapping, despite being computationally intensive both in terms of memory and time needed.

For these reasons, it became evident that the fusion of both modalities could have been of great help in modern SLAM applications. The systems born from this approach take the name of hybrid SLAM and use data of different types to directly estimate the motion of the robot. Hybrid SLAM, however, is still a maturing field, as the majority of methods existing in the literature just use one type of datum to compensate for another (e.g., visual front-end and laser back-end), and no true sensor fusion is performed.

Moreover, hybrid systems share some of the common issues of visual and laser SLAM, including the high elaboration time needed to process images and LiDAR scans, but also being unusable under certain weather conditions, e.g., snow or fog. Lastly, to overcome these problems, another type of SLAM has been recently proposed, even though only a few algorithms are available at this moment. Radar sensors are unaffected by illumination and weather and they can be efficiently used to perform localization and mapping, as the gathered data can be either represented as images or point clouds living in two dimensions (hence, having low computational requirements).

1.1 Thesis contribution

From the introduction, one can understand the variety of simultaneous localization and mapping systems available in the literature. From cameras to radars, each created system deals with a specific situation, consisting of many elements. Sensors used, configuration, type of considered scenario, possible environmental constraints, and user needs are just a few of the multitude of aspects to take into account when developing a SLAM system.

Moreover, the created methods greatly differ one from the other, in terms of architecture, needs, data format, and frameworks adopted. For this reason, studying, extending, or just using a SLAM method that suits specific criteria has become increasingly difficult over the past few years.

To solve this issue and advance the state of the literature, in our Ph.D.

program, we developed a common framework for SLAM, which can be adapted and exploited to deal with a specific situation or user needs. The framework is formed by multiple novel SLAM systems, implemented by us, able to achieve accurate and real-time localization under different environmental and hardware constraints (e.g., different sensors or unavailability of GNSS), while also constructing an accurate map of the environment.

The base of the framework is also the first system developed, ART-SLAM [1], an accurate real-time LiDAR SLAM system. Here, an input point cloud (obtained from a LiDAR sensor) is processed and used to estimate the motion of the robot, by matching it against previous scans. The estimated pose, along with additional information (e.g., floor coefficients) and an efficient loop detection method, are later used to optimize the trajectory and build a map of the environment. The work has been published in the *IEEE Robotics and Automation Letters (RA-L)* and it was presented at the *International Conference on Robotics and Automation (ICRA) 2022*.

Aside from SLAM, also localization plays an important role in real-world applications, as maps may be already available through some means (e.g., coming from SLAM itself). To tackle the issue of localization in GNSS-denied scenarios, we developed a localization module [2] for ART-SLAM, based on an Unscented Kalman Filter. High-frequency IMU and odometric data are fed to the filter to predict the position of the robot, which is later corrected by matching a low-frequency input LiDAR scan with the available 3D map of the environment. The work has been recently published in the *Frontiers in Robotics and AI Journal 2023*, specifically in the Localization and Scene Understanding in Urban Environments topic.

From there, we extended an existing odometry estimation method into a SLAM system, using the infrastructure of ART-SLAM as the base. Instead of matching consecutive point clouds obtained from a LiDAR, input scans are converted into 2D range/depth images and possibly coupled with RGB images taken from cameras. From a pair of RGB and range/depth images, multiple cues are extracted, such as intensity, depth, and normals. These cues, coming from different sensors (hence the name MCS-SLAM [3], Multi-cues Multi-sensors Fusion SLAM), are used to perform photometric error minimization and estimate the motion of the robot. This front-end is then embedded into the developed framework for SLAM, to optimize the estimated trajectory and build a 3D map. The work has been published and presented at the *IEEE Intelligent Vehicles Symposium (IV) 2022*.

Returning to the original ART-SLAM, we then addressed the specific situation where prior information coming from third-party mapping services is already available, which is not much developed in the literature, despite

its importance. We integrated 2D maps obtained from OpenStreetMap into ART-SLAM, creating OSM-SLAM [4]. Before optimization of the estimated trajectory takes place, the new system tries to find the best alignment between the current input point cloud, opportunely processed, and the 2D map of buildings. This alignment enforces further constraints in the optimization step, increasing local accuracy w.r.t. ART-SLAM. The work has also been published in the *Frontiers in Robotics and AI Journal 2023*, again in the Localization and Scene Understanding in Urban Environments topic.

To fully exploit the availability of multiple sensors, we then integrated ART-SLAM with two additional front-end branches and one more module for loop detection. A backbone Error State Kalman Filter allows to continuously estimate the trajectory of the robot, acting as a support of the original LiDAR odometry estimation method. Moreover, a stereo visual tracker has been added to the system front-end, which can be used in the case of an only-cameras setup or to integrate poses estimated with different methods (e.g., laser rather than visual). The system, which is a visual inertial laser 3D SLAM method, named D3VIL-SLAM, achieves superior accuracy than the baseline, and it allows to also perform visual SLAM. The work has been presented at the *IEEE Intelligent Vehicles Symposium (IV) 2023*.

Lastly, we studied SLAM in case radar is used as the main sensor, which is not a typical choice for SLAM, as it provides only 2D range data. This choice is motivated by the fact that radar sensors are more robust to environmental changes w.r.t. cameras and LiDARs so they can almost always be used to perform SLAM. Moreover, radar SLAM is not as studied or developed as visual or laser SLAM, despite having noticeable advantages. As the last step of our Ph.D. program, we completed our framework by creating a radar-based accurate and real-time SLAM system, named RadART-SLAM, to ascertain the accuracy and precision attainable using only radars.

We can sum up the contributions of our Ph.D. program as follows.

- Development of a LiDAR SLAM system, ART-SLAM [1], able to perform accurate localization and mapping, also thanks to a novel loop detection method, and serving as the baseline for other systems.
- Extension of ART-SLAM with a localization module [2], to deal with scenarios in GNSS-denied environments, both indoor and outdoor.
- Creation of a system that exploits multiple sensors to extract image cues and achieve faster than real-time tracking, named MCS-SLAM [3].
- Extension of ART-SLAM, integrating it with 2D map priors from OpenStreetMap, and developing a new system able to also perform

localization only, named OSM-SLAM [4].

- Construction of a hybrid SLAM system, supported by a backbone Error State Kalman Filter and able to perform simultaneous visual and laser-based tracking and loop detection, named D3VIL-SLAM.
- Creation of a real-time radar-based SLAM system, RadART-SLAM, built upon the developed SLAM framework.

It should also be noticed that all methods, except MCS-SLAM [3] achieve greater localization accuracy than state-of-the-art systems, while also working in real-time (or even faster, in the case of MCS-SLAM).

1.2 Thesis outline

The thesis is organized into three parts. First, we provide the background materials needed to understand the contributions of the thesis.

- Chapter 2 gives general information about SLAM, including the description of typical architectures. Then, a review of the literature is presented, with a particular focus on state-of-the-art methods.

The second part contains one of the contributions of this thesis, consisting of the base framework of the whole work.

- Chapter 3 consists of two sections. The first illustrates ART-SLAM, an accurate and real-time LiDAR SLAM system, that can outperform state-of-the-art methods in laser SLAM. The second section describes an extension of ART-SLAM, consisting of a module for localization, implemented as an Unscented Kalman Filter (UKF). In this section we also give a detailed comparison of SLAM systems that are also able to perform localization, benchmarking them using real data.

The other contributions of the thesis are described in the remaining chapters, each detailing a new system that extends ART-SLAM.

- Chapter 4 presents a variant of ART-SLAM, named MCS-SLAM (Multi-Cues Multi-Sensors SLAM). MCS-SLAM performs sensor fusion by exploiting multi-cues extracted from sensor data, i.e., color/intensity, depth/range, and normal information. For each sensor, motion estimation is achieved through the minimization of the pixel-wise difference between two multi-cue images. All estimates are jointly optimized, to best satisfy the motion constraints derived from all sensors.

- Chapter 5 shows an improved version of ART-SLAM, which can exploit additional information coming from mapping services like OpenStreetMap, hence the name of OSM-SLAM (OpenStreetMap SLAM). The system integrates the 2D geometry of buildings in the trajectory estimation and optimization procedures, by matching a prior OpenStreetMap map with a single LiDAR scan. This way, the estimated trajectory of the robot can be corrected, further enhancing the localization and mapping accuracy of the SLAM system.
- Chapter 6 further extends the framework developed in our Ph.D. program by adding a vision branch, able to perform tracking based on stereo images, and a backbone Error State Kalman Filter, which collects odometry estimates coming from all other sensor branches to accurately predict the motion of the robot in real-time. All trajectory estimates obtained from the various front-ends are then merged around the poses acquired in almost the same instant and jointly optimized.
- Chapter 7 describes the last extension of the proposed framework, which uses radar images as input to estimate the motion of the robot.

Lastly, the thesis ends by revisiting briefly its contributions in Chapter 8, including challenges in SLAM, possible improvements, and future works.

CHAPTER 2

SLAM overview and state-of-the-art

In this chapter, the Simultaneous Localization and Mapping process, also known as SLAM, is presented, along with different classifications, state-of-the-art approaches, and techniques developed to solve it. The chapter begins with the introduction of concepts and definitions, in Section 2.1, which gives basic information to understand the SLAM problem. Section 2.2 follows with a description of a generic SLAM architecture. The next sections deal with three specializations of SLAM systems: laser SLAM, in Section 2.3, visual SLAM, in Section 2.4, and radar SLAM, in Section 2.5. Then, Section 2.6 briefly describes the concept of Hybrid SLAM, and the chapter ends in Section 2.7, with a high-level overview of SLAM algorithms used for dynamic environments, which are outside the scope of this thesis but they are included for completeness. Together with a description of the main features of these systems, some state-of-the-art works are also discussed.

2.1 Concepts and definitions

Simultaneous Localization and Mapping is the process by which a robot builds a map of the environment and, at the same time, uses this map to compute its location while navigating it. To solve the SLAM problem, many

techniques have been proposed in the last decades, all relying on the ability of robots to capture information from the environment as they go through it. To do so, they must be equipped with a sensorial system (e.g., cameras or radars), capable of extracting valuable data from the world, such as images of the scene or point clouds, to be used in finding its location.

Probability plays a large role in successful SLAM solutions, due to the inherent noise in sensor measurements. The robot has no prior knowledge of its surroundings so it must use its sensors in order to gather information. Therefore, SLAM must deal with the uncertainty of locations due to inaccurate sensor data, often making mistakes in localizing the robot.

SLAM can be modeled in a probabilistic way, as follows. The robot is assumed to move along a trajectory described by the sequence of random variables $x_{1:T} = \{x_1, \ldots, x_T\}$, where the single variable x_i represent the pose (position and orientation), either 2D or 3D, of the robot w.r.t. an initial pose x_0, arbitrarily set. In correspondence with each pose x_i, the robot acquires a sequence of odometry (i.e., the process of estimating the robot position and rotation) values $u_{1:T} = \{u_1, \ldots, u_T\}$, which are data from motion sensors (e.g. rotary encoders), and measurements of the environment $z_{1:T} = \{z_1, \ldots, z_T\}$, uniquely determined by the equipped sensors.

The map of the environment, denoted as m can be parameterized as a set of spatially located *landmarks*, which are salient space points characterized by similar appearance, by dense representations like occupancy grids, surface maps, or by raw sensor measurements. The choice depends on many factors, such as the used sensor, the characteristics of the environment, and the techniques adopted to perform both localization and mapping.

Solving the SLAM problem consists of estimating the posterior probability of the trajectory $x_{1:T}$ of the robot and the map m of the environment given all the measurements plus the initial pose x_0. A possible model to describe a SLAM system is the *Dynamic Bayesian Network*, which is a graphical model used to represent a stochastic process as a directed graph. The robot poses and measurements can be modeled separately, highlighting which elements of the SLAM process they depend on. The *motion model*, also known as the state transition model, describes the robot pose as a function of its previous pose and odometry measurements: $x_t = f(x_{t-1}, u_t)$. The *observation model*, instead, describes the robot sensor measurements as a function of the robot position and the map elements: $z_t = f(x_t, m)$.

A visual representation of the probabilistic interpretation and modeling of a SLAM process is given in Figure 2.1. Gray nodes indicate the observed variables, which are the odometry and available sensor measurements. White nodes represent, instead, hidden variables, $x_{1:T}$ and m, which model the

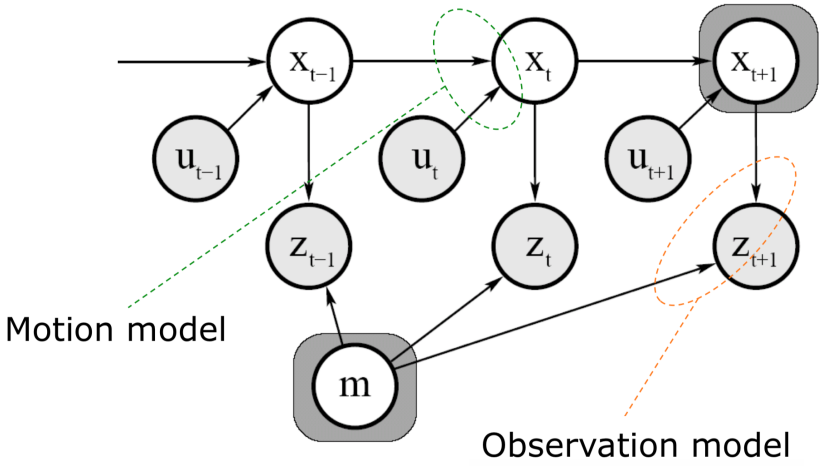


Figure 2.1: *Dynamic Bayesian Network of a SLAM process. The figure shows which parts of the graph are involved in the state motion (green) and observation (orange) models.*

trajectory of the robot and the map of the environment, respectively.

The connectivity of the graph follows a recurrent pattern, characterized by the motion and observation models. The transition model is represented by the two edges leading to x_t and consists in the probability that the robot, at time t, is in x_t, given that at time $t-1$ it was in x_{t-1} and it acquired an odometry measurement u_t. The observation model is represented by two edges pointing at z_t and it describes the probability of performing the observation z_t, given that the robot is at location x_t in the map m.

Expressing SLAM as a Dynamic Bayesian Network highlights its temporal structure, and therefore this formalism is well suited to describe filtering processes that can be used to tackle the SLAM problem. Since the '90s, different approaches have been proposed to solve the SLAM problem using its probabilistic interpretation, including *Kalman Filters* (KF), *Particle Filters* (PF) and *Expectation Maximization* (EM). A detailed explanation and comparison of the different probabilistic methods are discussed in [5], while the following paragraphs give a brief description of the various methods.

KF are Bayes filters that represent posteriors using Gaussians, i.e., unimodal multivariate distributions that can be expressed compactly by a small number of parameters. KF SLAM relies on the assumption that the state transition and the measurement functions are linear with added Gaussian noise, and the initial posteriors are also Gaussian. Many variations of KF can be found in literature, including the well known *Extended Kalman Filter* [6–9]

and its related *Information Filtering* (IF) or *Extended IF* [10].

PF [11, 12], also known as sequential Monte-Carlo (SMC) methods, are recursive Bayesian filters that are implemented in Monte Carlo simulations. The basic idea is to approximate the pose posterior distribution, i.e., the belief, with a set of sample states $x_t^{[i]}$, or particles, with index i ranging from 1 to M (considered as the size of the particle filter). At time $t > 0$, each particle $x_{t-1}^{[i]}$ from the previous belief follows the motion model of the robot, given actuation commands, reaching a new pose estimate $x_t^{[i]}$. From all the particles obtained, M new elements are generated with a probability proportional to the likelihood of the expected sensor value for that state value. Through these three steps, the belief of the robot is periodically updated, using the information of the particles. The procedure is repeated until the probability estimated from the extracted particles converges.

EM estimation is a statistical algorithm developed in the context of maximum likelihood (ML) estimation and it is able to build a map when the robot pose is known by means of expectation. EM iterates two steps: an expectation step, where the posterior over robot poses is calculated for a given map, and a maximization step, in which the most likely map is calculated given the expectations associated with the various poses. The final result of EM estimation is a series of accurate maps.

Probabilistic approaches have many disadvantages. Kalman filter-based SLAM results are often inconsistent and present a considerable computational complexity, which grows $e^{2.4}$ w.r.t. the number of features in the map, when the environment scale increases (e.g. urban scenery). Particle filter methods have many limitations, due to their constrained number of particles - again, for computational reasons - required to estimate the current pose of the robot. Lastly, expected maximization techniques are very limited and perform poorly in the localization aspect of SLAM.

Over the course of the years, the concept of SLAM moved away from the probabilistic methods, still improved in many recent works, and reached a more mature and modular representation. Instead of treating the SLAM problem as a whole, the state-of-the-art consists in breaking the process into multiple independent modules, each assigned to a different task. This way, the whole localization and mapping approach can be easily made more efficient, and each module can be highly specialized in dealing with a particular issue (e.g., motion estimation, rather than mapping).

2.2 SLAM system architecture

The standard architecture of a SLAM system can be described with different degrees of detail. From a high-level perspective, a SLAM process is composed of two main modules: the front-end and the back-end.

The *front-end* abstracts sensor data and extracts spatial relations between individual observations. In the front-end, sensor measurements from different time steps are processed and confronted to estimate the relative motion between the corresponding robot poses. This module is the core of the localization aspect of SLAM, as it keeps track of the robot trajectory while performing data association, exploiting different techniques (dependent on the type of data considered and sensors used). Moreover, in the front-end, also a rough map is created, either broken in pieces associated with each sensor measurement (local maps) or as a whole (global map), or both.

If the front-end builds a rough estimate of the robot trajectory, the *back-end* optimizes it to generate a more accurate set of robot poses. An important fact is that the trajectory can be optimized anytime and in different ways. If the whole trajectory is improved, the process takes the name of *global optimization*. If only a limited amount of poses is adjusted, the procedure is called *local optimization*. Global optimization produces more accurate results since it takes into account all the information associated with the robot poses, but it is slow. Instead, local optimization is faster than its global counterpart, but leads to minor improvements, due to its reduced scope.

In the back-end module, the map of the environment is fully built and eventually refined, along with the trajectory of the robot. Depending on the choice of the map representation, this creation and improvement task can degrade the performance of a system, especially when using certain sensors.

For example, if the sensor used is a laser rangefinder, which registers point clouds, and the map is represented as a set of points, the only task that the back-end module should complete is a simple and fast alignment of the sensor data. On the other hand, if the input sensor is a camera, which captures images of the scene, and the map is represented as a set of small filled volumes, the back-end module needs to perform conversion (from 2D images to 3D spaces), space-filling, and other demanding procedures.

A more detailed representation of a generic SLAM architecture is represented in Figure 2.2. The typical workflow includes the following steps.

Sensor data acquisition. Intuitively, it refers to the gathering of data by the sensor suite equipped on the robot. It also includes the acquisition and synchronization between encoders, IMU, and other sensors, if available.

Data pre-processing. Available in the front-end module, it refers to the

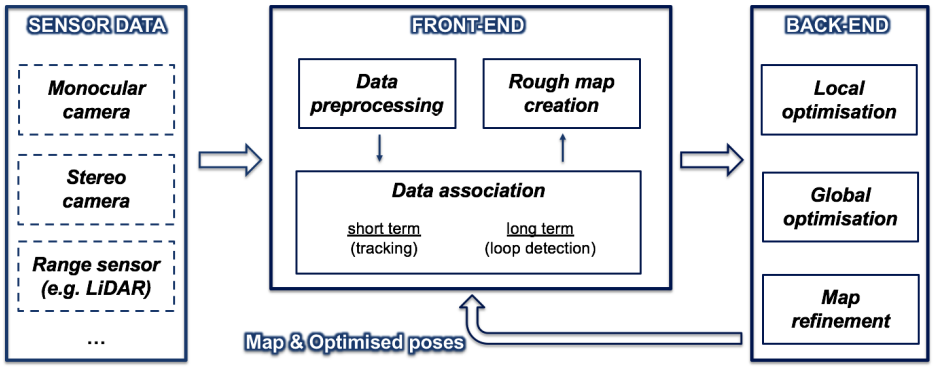


Figure 2.2: *Scheme of a generic SLAM architecture. Sensor data is processed by the front-end, which elaborates the measurements and tracks the poses of the robot, after data association. The trajectory is then passed to the back-end, which performs pose optimization and creates or updates the map of the environment around the robot.*

task of elaborating the acquired sensor data, making it suitable for the pose estimation procedure and all successive SLAM parts (e.g., loop detection).

Data association. Still remaining in the front-end part of a SLAM system, data association is the process of finding the associations between the data gathered at two different time steps (short term association if consecutive, otherwise long term), which then allow to estimate the motion of the robot.

Rough map creation. While performing data association and pose estimation, each sensor measurement, or the most important ones, is attached to a rough estimate of the map, to be processed later in the back-end.

Local and global optimization. These back-end activities optimize either parts or the whole trajectory of the robot, estimated during the data association phase, sometimes exploiting the corresponding rough maps.

Map refinement. It takes the local maps created in the front-end, which are then integrated and corrected in the global map of the environment.

The SLAM process begins with the acquisition of sensor data by the robot. Raw data is often not suitable to be used directly in the SLAM process and, dependently on the algorithms used to associate multiple measurements, it is processed and elaborated in different ways. Size reduction, feature extraction, data conversion, denoising, and filtering are just some of the possible ways to prepare the data for the other modules of a SLAM system.

After the raw measurements have been elaborated, the front-end performs the core task of a SLAM process: data association and pose estimation. The goal of this step is to first find correspondences between data, and then use them to estimate the motion of the robot between the two acquisitions. Data

association is classified into two types: short term and long term.

Short term data association is the process of finding correspondences between the data gathered at two consecutive time steps. This allows *pose tracking*, which is the task of finding the rigid transformation between the current pose of the robot, at time step t and the previous one, at time step $t-1$. Once the relative motion has been found, the current location can be easily computed from the previous position. Formally, given the robot pose x_{t-1} and the sensor measurements z_{t-1} and z_t, pose tracking, which follows short term data association, finds the transformation between the two data readings, with which it estimates the current location of the robot, x_t.

Using this procedure, the trajectory of the robot is incrementally updated using the information extracted from the environment. Moreover, each estimated pose is associated with a rough partial estimate of the map, obtained by combining the input data and the information obtained from tracking.

Long term data association is a more subtle task, but essential for back-end optimization. The goal of this procedure is to recognize whether the robot is visiting a previously discovered location in the environment, which translates into the estimated trajectory displaying a ring-like structure, i.e., a loop. This is known as *loop detection and closure*, and it allows for the enforcement of positioning constraints over the trajectory of the robot.

Figure 2.3 shows a 2D toy example to describe the concept of loop detection and closure: the robot moves along a trajectory made of consecutive poses $\{x_0, \ldots, x_{10}\}$. The last position x_{10} is close to previously visited locations, namely x_1 and x_2. This means that the robot has generated a loop in its trajectory while moving, represented by the dashed red arrows.

While the purpose of tracking is to find the current pose of the robot using consecutive processed sensor measurements, loop detection is simply a “yes and no" search in the set of known poses, also using the available data. In this sense, such a task is more time-consuming than simple tracking, since it involves the comparison of the current sensor measurement with all the previous ones, in pairs. Usually, these comparisons are done with the same or similar algorithms used to perform tracking. However, many works adopt a different approach to speed up the process and to find the relative motion between the poses corresponding to a loop, i.e., performing closure.

Lastly, one must not fall into the temptation of comparing only the poses to detect a loop, since they are already available (loop detection and closure are always done after pose tracking), to speed up the task. Estimated poses are an unreliable source of information, due to possible drifting errors and mistakes accumulated by tracking, while sensor measurements represent an accurate description of the environment, so the latter must be preferred.

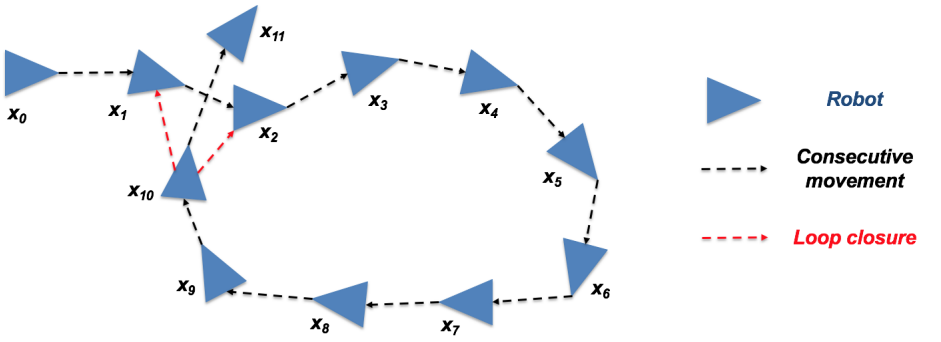


Figure 2.3: *Toy example of loop closure. The robot (blue triangle) moves along a certain trajectory, represented by dashed black arrows (where the head represents the direction of movement). When reaching position x_{10}, the robot is nearby the previously visited locations x_1 and x_2, detecting a loop in the trajectory (red arrows close the loop).*

The poses estimated with front-end tracking are influenced by the noise afflicting the input data (or even reading mistakes of the sensors, such as LiDAR measurements in foggy or snowy environments), diverging from the actual locations of the robot by a non-trivial amount. Using both the estimated map, or map parts, the detected loops, and optional additional constraints, the trajectory is optimized either locally, globally, or both.

Local optimization exploits the map representation to enforce consistency between multiple consecutive poses. Global optimization involves, instead, the whole estimated trajectory and it uses the constraints derived from loop closures and map elements (floors, walls, known locations) to adjust simultaneously all the poses, such that all the constraints are satisfied.

To sum up, in this section, a brief explanation of the common elements of all SLAM systems has been given. To better understand the various approaches adopted and evolved over the course of the years, a more detailed description of the already discussed steps is given in the following sections.

2.3 Laser SLAM

To overcome some of the problems associated with cameras, such as illumination dependencies, many SLAM systems use laser rangefinder sensors (e.g. LiDAR or Microsoft Kinect). As the majority of laser SLAM systems process LiDAR data, this sensor is considered the standard, and it is used in the rest of this thesis to describe and deal with laser SLAM methods.

By transmitting and receiving laser beams, a LiDAR uses the Time of

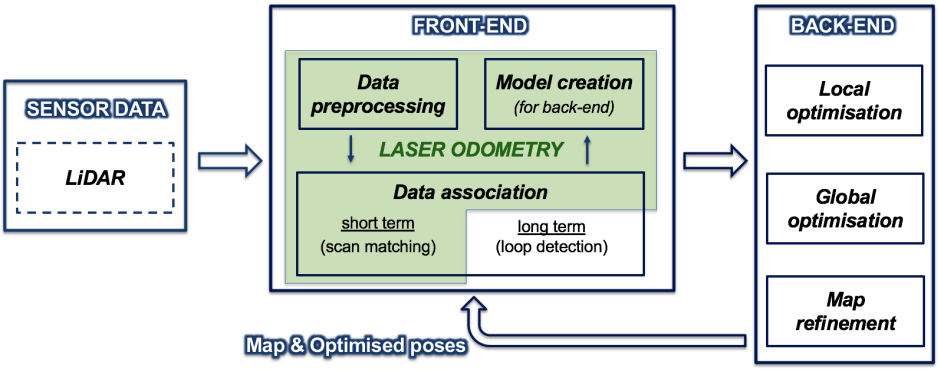


Figure 2.4: *Typical architecture of a Laser SLAM system. Modules covered by the green box represent laser tracking, the process of estimation of the robot movement by using consecutive input scans, which correspond to sparse or dense point clouds.*

Flight method to obtain the distance of the scanning point from the center of the sensor, which is almost unaffected by the illumination conditions. Moreover, LiDAR sensors have a long detection range and high precision.

LiDARs can deal with large-scale, light or dark environments, and are a reliable source of information for SLAM systems. The typical architecture of a LiDAR-based SLAM algorithm follows the same steps as the baseline architecture, presented in Section 2.2 and visible in Figure 2.2. The structure and pipeline of a LiDAR SLAM system can be seen in Figure 2.4.

Sensor data gathering. As described at the beginning of the section, in laser SLAM, the sensors used are laser rangefinders, which obtain a point cloud representation of the environment, which can be either sparse or dense.

Laser odometry (LO). This task is the front-end of a laser SLAM system, as it estimates the sensor movement (hence, the trajectory of the robot) between consecutive measurements. As each scan is already a rough map of the environment, different from visual SLAM methods, where the map must be extracted from scratch, laser SLAM requires only refining and stitching together the available local maps. The majority of laser SLAM systems take input scans obtained from sensors whose specifics (equivalent to camera intrinsic and distortion parameters) are given by the constructor.

Loop detection and closure. As described in Section 2.2, it is the task of finding if a robot is visiting a previously seen location, meaning that is passing through the same area. In laser SLAM, this is done by checking similarities between pairs of LiDAR scans. If a loop is detected, it will provide additional information to improve the accuracy of the optimization.

Optimization. Since it is considerably hard and computationally expen-

sive to enforce consistency between multiple point clouds, laser SLAM algorithms rely on additional data structures, usually known as keyframes, to model the whole trajectory and optimize them instead.

Laser SLAM systems have a more intertwined front-end and back-end since the latter highly relies on structures created and updated in the first phase. The state-of-the-art methods use graphs as data structures to model the SLAM process, enforcing consecutive motions and loop closures.

The front-end of laser SLAM consists of the following steps. First, a global coordinate system is defined for the robot. It is not needed to also associate it for the different scans, locally, since they are already directly related to the robot by known transformations, differently from visual SLAM methods, where features extracted from images must be correlated both with the camera sensor and the constructed map of the environment.

After this step, tracking and mapping are continuously performed to estimate the poses of the robot. In the tracking phase, either the estimated map or the previous point cloud are tracked in the current LiDAR scan, to find the current pose of the robot. This is done in different ways, either by aligning the whole clouds or by extracting 3D features and matching them.

After the current pose is estimated, the corresponding scan is integrated into the available map, eliminating redundant points and adjusting positions. Moreover, a secondary model of the SLAM process is built (usually a graph), to be used in the optimization phase, as mentioned before.

Loop detection and closure follow the same procedure as adopted for tracking: to search for loops in the estimated trajectory of the robot, the current scan (or its features) is compared either with the previous measurements or with the map of the environment (or, again, the extracted features).

The back-end of laser SLAM main purpose is to optimize the estimated poses. Differently from the visual counterpart, these systems do not usually require re-localization, since pose tracking is robust enough and does not suffer from the same problems as image-based tracking: scans represent the whole environment surrounding the robot, capturing rich and complete information about the surroundings, which is exploited to accurately track the movement of the robot. On the other hand, images show only a limited view of the scene, which may be lost in consecutive sudden movements. Optimization, as already stated, is done using side data structures, built in the front-end and are easier to manipulate, to model the SLAM process.

Laser SLAM systems can be classified into three categories, depending on how pose tracking is performed and how the environment is represented: *point cloud based methods*, *feature based methods*, and *grid based methods*.

2.3.1 Point cloud-based laser SLAM

In point cloud-based SLAM methods the map is represented as a set of points, forming a cloud, and the input scans directly contribute to the map construction. The core of point cloud-based methods is the *Iterative Closest Point (ICP)* [13–15] algorithm and its variants, proposed over the years.

Straightforward ICP works as follows. Given two scans $S = \{s_1, \ldots, s_n\}$ and $M = \{m_1, \ldots, m_n\}$ (source and model), the goal is to find the rigid transformation (rotation R and translation t) which best aligns the given clouds. One iteration of the algorithm consists of the following steps.

1. Compute the nearest point in set S for every point (or part of the points) in set M, e.g., using the Euclidean distance:
$$d_i = min\sqrt{m_i^2 - s_j^2},\ j = 1, \ldots, |S|. \tag{2.1}$$
2. If the computed distance is greater than a threshold, meaning $d_i > threshold$, the corresponding pair of points is removed.
3. Add weights w_{ij} to pairs of points i and j. In straightforward ICP, $w_{ij} = 1$ if the two points are close and form a correspondence, zero otherwise. In other variants of ICP, the weights can be set accordingly to the direction of the normal vectors associated with the points, as their dot product, $w_{ij} = n_i \cdot n_j$.
4. Compute the rotation matrix R and translation vector t using a Least Squares-based method for distance minimization.
5. Compute the transformation of set S using computed values: $R \cdot s_j + t$.
6. Compute the error between the model cloud and the transformed cloud and iterate until the required accuracy is achieved, i.e., convergence.

Firstly, the nearest point in the set S is calculated for every point in the set M. It is possible to use all of the points or they can be chosen randomly. The ICP algorithm assumes that the corresponding points are the nearest ones; hence, it is suited for the SLAM front-end due to its consecutive nature, i.e., because of the reduced amount of change in translation and orientation between consecutive poses. For these pairs of corresponding points, the rotation matrix and translation vector are computed, by minimization of the distance error between all point correspondences:

$$E(R, t) = \sum_{i}^{|M|} \sum_{j}^{|S|} w_{ij} ||m_i - (R \cdot s_j + t)||^2. \tag{2.2}$$

The previous distance error can be rewritten in the following way:

$$E(R,t) \propto \frac{1}{N}\sum_{i}^{N} ||m_i - (R \cdot s_j + t)||^2, \tag{2.3}$$

where $N = \sum_i^{|M|}\sum_j^{|S|} w_{ij}$ is the sum of all weights.

To find R and t, *Singular Value Decomposition (SVD)* is used. The centroids of both point sets are computed, to be used in SVD, as:

$$C_M = \frac{1}{N}\sum_{i}^{N} m_i \quad \text{and} \quad C_S = \frac{1}{N}\sum_{i}^{N} s_i. \tag{2.4}$$

Using these centroids, both point sets are re-aligned, as:

$$M' = m_i - C_{M_{1,\ldots,N}} \quad \text{and} \quad S' = s_i - C_{S_{1,\ldots,N}}. \tag{2.5}$$

From these values, the covariance matrix $H = M'S'^T$ is computed, and decomposed as $H = U\Lambda V^T$. The desired rotation matrix is given by $R = VU^T$, while the translation vector is $t = C_S - RC_M$. Lastly, the error between the transformed source set and the model point cloud is calculated, and the whole process is iterated until convergence (error below a threshold).

Different improvements of ICP have been proposed to satisfy better convergence and improve performance. ICP algorithm computation can be efficiently accelerated (up to 10 times [16]) by using KD trees, which are a special case of binary space partitioning trees, to perform fast neighbor correspondence between points from different input LiDAR scans.

Point clouds often contain noise points and outliers. To avoid wrong alignments of the scans, multiple criteria are added to eliminate, or at least mitigate this problem. The usual recommendation is to remove pairs of points that contain elements at the borders of the clouds. Another option is to remove points that have a greater distance than a given threshold.

A variant of ICP, named point-to-plane ICP [17], uses a different error function, with the goal of minimizing not only the distance of two points but also the approximate distance of one point to the plane to which the other point belongs. This method reduces the number of iterations, converging faster, but is not as robust as the standard version. Figure 2.5 shows the difference between ICP (on the left) and point-to-plane ICP (on the right).

Another version, the Generalized ICP algorithm (GICP) [18] combines normal ICP and point-to-plane ICP to achieve superior performance while dealing with multiple cases. A detailed summary of most of the possibilities

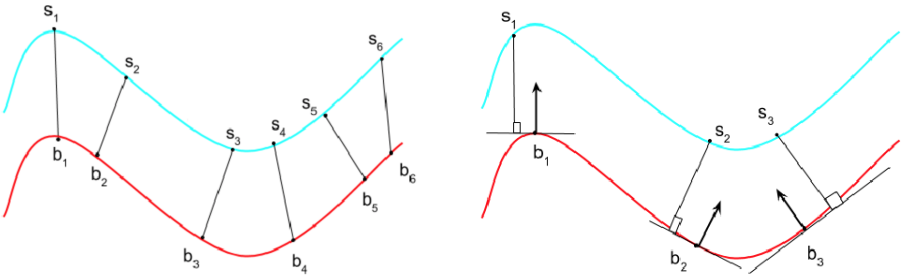


Figure 2.5: *Difference between the ICP (on the left) and point-to-plane ICP (on the right) algorithms. The blue and red lines represent the source and target point clouds.*

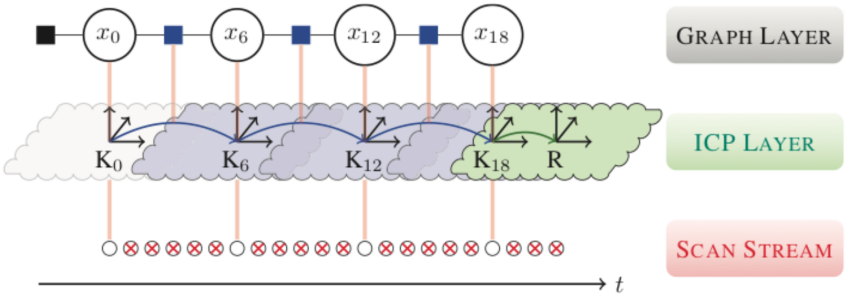


Figure 2.6: *Example state of the ICP-based SLAM system proposed in [23].*

in cloud matching methods is covered in [19]. More recent approaches modify ICP to incorporate semantic information [20] (e.g., from segmented LiDAR scans), use accelerated iteration methods [21] to improve performance, or even combine ICP with other point cloud matching algorithms [22].

Mendes et al. [23] proposed a purely *ICP-based SLAM* system, modeled as a graph. Figure 2.6 describes a hypothetical state of the system after processing some scans. There are two main layers plus the scan stream, which is simply the history of acquired sensor measurements.

The ICP layer is composed of keyframes K_i, and their associated scans, also called keyframe scans. Each keyframe K_i is associated with a node x_i in the graph layer, representing the pose of the robot when gathering the corresponding scan. Some of these keyframes are selected to compose local maps (blue clouds), which are used as reference inputs for the ICP process. The other keyframes (gray clouds) are stored and may be later used to correct local maps in case of loop detection. The robot pose X is always expressed w.r.t. the closest keyframe in the current local map.

Still referring to Figure 2.6, as the robot starts, a first keyframe K_0 is

associated with the first acquired LiDAR scan. The corresponding node x_0 is also created in the graph, and a constraint representing the initial pose of the robot w.r.t. the world origin is added (black square).

Every time a new input (green cloud) is available, ICP finds the transformation that aligns it with the last local map (green arrow), estimating the current pose of the robot. If the overlap between the current scan and the last local map is lower than a threshold, a new keyframe is created, otherwise, the scan is just discarded (crossed circles in the scan stream level).

When a new keyframe is created, it is initialized with the estimated location. A new pose variable is then added as a node to the graph, as well as a constraint (blue squares) containing the transformation between the new and former keyframes (blue arrows). Lastly, the local map is updated with the scan of the newest keyframe, while the oldest cloud is removed from it.

When the system detects a potential loop between two keyframes, a local map is built around the oldest between these candidates. Then an ICP call tries to align the scan of the other keyframes, i.e., the most recent one, with this local map. If ICP is successful, a new factor is added to the graph between the variables associated with these keyframes, closing a loop in the graph. Optimization is then triggered and the loop detection process ends with the system re-positioning all keyframes using the optimization results, and reconstructing all local maps with the corrected estimated poses.

6D SLAM [24] is another ICP-based SLAM system. To match two 3D scans with the ICP algorithm it is necessary to have a sufficiently accurate starting guess for the current scan pose, due to the fact that their points correspond to two different locations in the same environment.

To overcome this problem, in 6D SLAM, first, the transformation found in the previous scan matching is applied to the current pose estimation, to implement the assumption that the error model of the pose estimation is locally stable; then, a pose update is calculated by matching octree representations of the scan point (visible in Figure 2.7) sets, rather than the point sets themselves, to speed up the whole pose tracking process.

Loop closure in 6D SLAM is done by matching the current scan with earlier measurements. To avoid processing all the previous scans, first a hypothesis based on the maximum laser range. Then the octree-based method used for scan matching is applied to the candidate scans for loop detection. Lastly, if a loop is found, the corresponding relative transformation is distributed over all existing 3D scans, and the total displacement error is then minimized according to the simultaneous matching method [25, 26].

Behley and Stachniss [27] proposed a novel, dense approach to laser-based mapping that operates on 3-dimensional point clouds obtained from

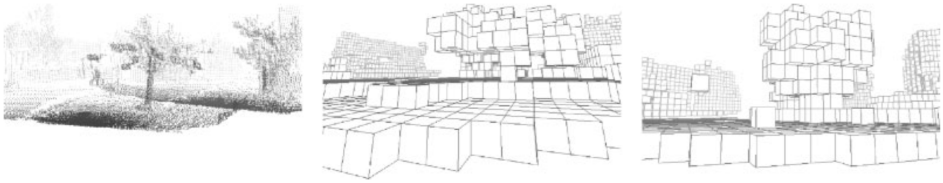


Figure 2.7: *Octree generation from point clouds, described in [24]. The left figure represents two scans; the middle panel shows the octree corresponding to the black/front point cloud; and the right image shows the octree based on the gray/back points.*

rotating laser sensors. They construct a surfel-based map and estimate the changes in the pose of the robot by exploiting the projective data association between the current scan and a rendered model view from that surfel map.

For detection and verification of a loop, the map representation is again leveraged to compose a virtual view of the map before a potential loop closure, which enables more robust detection, even with low overlap between the scan and the already mapped areas. The approach is efficient and enables real-time registration and loop closure, although it requires a GPU to keep such performance, making it unusable on low-end devices.

A more recent approach using full point clouds is *HDL* [28]. The system can be summed up as a pipeline of four steps. First, a laser input is pre-processed and filtered to reduce its size, a mandatory operation to improve the performance of scan matching, as aligning full clouds is demanding.

Then, the filtered scans are used to simultaneously perform tracking, through a scan-to-keyframe matching approach, and to possibly detect the sub-part of the cloud representing the ground plane in them. Poses estimated through tracking, and floor coefficients extracted from the point clouds, are lastly used to build a graph of the trajectory, i.e., a pose graph, which is later optimized to best satisfy all motion constraints. The system achieves superior performance, but, as for all full point cloud-based methods, it is slow, especially when dealing with considerably large point clouds.

Lastly, *CT-ICP* [29] developed a new real-time LiDAR-only odometry method called CT-ICP (for Continuous-Time ICP), completed into a full SLAM with a novel loop detection procedure. The core of this method is the introduction of the combined continuity in well known scan matching procedures, and discontinuity between scans. It allows both the elastic distortion of the scan during the registration for increased precision, and the increased robustness to high frequency motions from the discontinuity. Loop detection is based on the conversion of submaps into elevation images, which are then matched using a RANSAC-like approach.

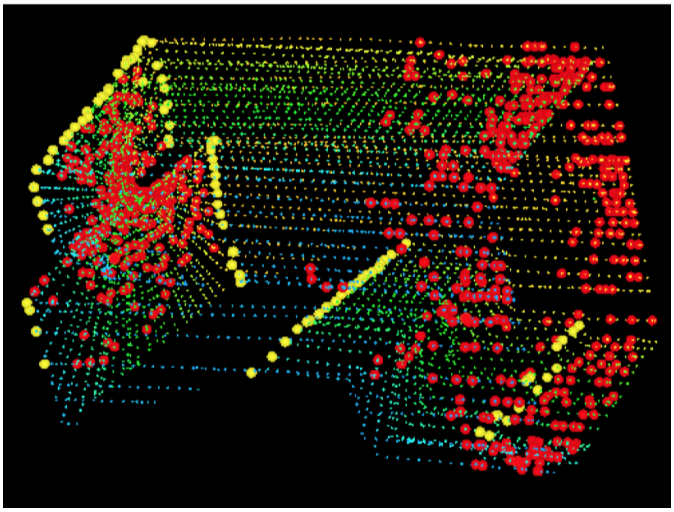


Figure 2.8: *Linear (yellow) and planar (red) points extracted from a 3D point cloud.*

2.3.2 Feature-based laser SLAM

In feature-based SLAM methods, the map of the environment can be either represented as a point cloud or as a set of space features, although the first is generally preferred. Particular features are extracted from the sensor scans in the form of corners, lines, and planes, to facilitate the subsequent scan matching process. Figure 2.8 shows linear and planar points, respectively colored in yellow and red, extracted from a 3D point cloud.

LOAM [30] represents one of the state-of-the-art in feature-based laser SLAM, and its architecture is represented in Figure 2.9. Let P_k be the set of points received in the current laser scan. P_k is processed in two algorithms.

The laser odometry, running at a frequency of around 10 Hz, takes the point cloud and computes the motion of the laser sensor between two consecutive scans. The estimated motion is used to correct distortion in P_k. The outputs are further processed by LiDAR mapping, which matches and registers the undistorted cloud onto a map, at a frequency of 1 Hz. Finally, the pose transforms outputted by the two algorithms are integrated to generate a result at around 10 Hz, regarding the pose w.r.t. the map.

Let i be a point in $P_k, i \in P_k$, and let S be the set of consecutive points of i, returned by the laser in the same scan. Since the sensor generates point returns in clockwise or counter-clockwise order, S contains half of its points on each side of i, at fixed angular intervals. A function is defined to evaluate

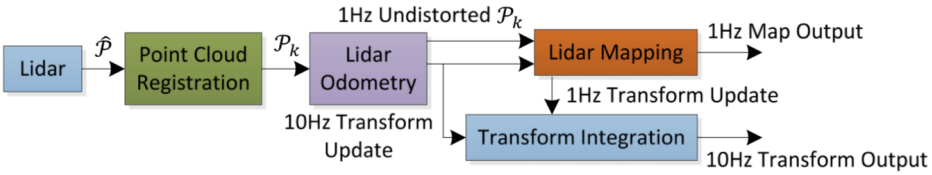


Figure 2.9: *System architecture of LOAM [30], including the various modules and data.*

the smoothness of the local surface formed by nearby points:

$$c = \frac{1}{|S| \cdot ||X^L_{(k,i)}||} || \sum_{j \in S, j \neq i} (X^L_{(k,i)} - X^L_{(k,j)}) ||, \tag{2.6}$$

where $X^L_{(k,i)}$ indicated the coordinates of point i in the sensor frame k.

The points in a scan are sorted based on c. Then, feature points are selected with the maximum c values, namely, edge points, and the minimum c values, namely planar points. To evenly distribute the feature points within the environment, scans are separated into four identical regions. Each region can provide a maximum of 2 edge points and 4 planar points. A point i can be selected as a feature only if its c value is larger or smaller than a threshold, and the number of already selected points does not exceed the maximum.

Features are then matched in the following way. Let E_k and E_{k+1} be the set of edge points extracted in scans k and $k+1$ respectively. Let i be a point in E_{k+1}. Let j be the closest neighbor of i in P_k, and let l be the closest neighbor of i in the two consecutive scans to the cloud to which j belongs. (j, l) forms the correspondence of i. Then, to verify that both j and l are edge points, the previous equation is used to enforce the smoothness of the related local surface, exploiting the associated c values.

The same thing is done for planes correspondences. Let H_k and H_{k+1} be the set of planar points extracted in scans k and $k+1$ respectively. Let i be a point in H_{k+1}. The planar patch is represented by three points. Similarly to the edge correspondence, the closest neighbour j of i in P_k is found. Then, points l and m, as the closest neighbors of i, are considered, one in the same scan of j, and the other in the two consecutive scans to the scan of j. This guarantees that the three points are non-collinear. To verify that j, l, and m are all planar points, the previous equation is checked once again.

The motion estimation is adapted to a robust fitting. For each feature point, a bi-square weight is assigned. Feature points that have larger distances to their correspondences, computed as point-to-line and point-to-plane distances, are assigned with smaller weights, and feature points with

distances larger than a threshold are considered outliers and assigned with zero weights (similarly to ICP). The pose transform between the two scans is computed and updated using non-linear optimization, formulated on the feature correspondences. The whole process is iterated until convergence.

Lastly, the mapping phase is done similarly to the laser odometry procedure described in the previous paragraphs. Here, the current scan is matched and inserted in the map of the environment, updating it. Differently from LOAM laser odometry, 10 times the number of features are used, in mapping, to achieve better correspondence and denser representations.

LOAM achieves good results in various scenes when integrated with an IMU to estimate initial guesses of the robot motion (speeding up the laser odometry algorithm). However, extracted features can be sparse in road-dominated autonomous driving scenes, and, without IMU, LOAM tends to drift as there is no loop closure, making it unsuitable for large trajectories.

Many feature-based works have been developed using LOAM as a baseline, due to its relatively high efficiency and performance. Deschaud et al. [31] proposed *IMLS-SLAM*, which relies only on LiDAR sensors. The proposed SLAM algorithm consists of three consecutive steps.

First, a local deskewed point cloud is computed from one scan of the LiDAR used. Second, specific samples from that point cloud are selected, to be considered as feature points. Lastly, the extracted samples are used to minimize the distance to a model cloud representing the environment. The main contribution of IMLS-SLAM is the way tracking of the robot motion is performed: sampled features are compared against the Implicit Moving Least Square (IMLS) representation, computed directly on the point cloud of the map formed by the last n localized LiDAR scans.

LeGO LOAM [32], whose architecture is represented in Figure 2.10, is a direct improvement of LOAM. The system receives input from a 3D LiDAR and outputs 6 degrees of freedom pose estimates. The method is characterized by the presence of five modules. The first, segmentation, takes the point cloud derived from the current scan and projects it onto a range image for segmentation. The segmented point cloud is then sent to the feature extraction module, which works in the same way as for LOAM.

Then, LiDAR odometry uses features extracted from the previous module to find the transformation relating consecutive scans, exploiting the labels obtained through segmentation. The features are further processed in LiDAR mapping, which registers them to a global point cloud map. Lastly, the transform integration module fuses the pose estimation results from LiDAR odometry and LiDAR mapping and outputs the final pose estimate.

MULLS [33] is a recent efficient, low-drift, and versatile 3D feature-based

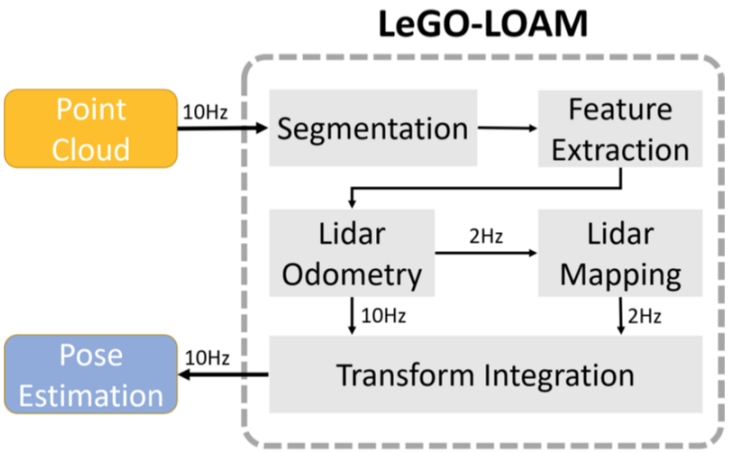


Figure 2.10: *System architecture of LeGO LOAM [32], including the various modules.*

LiDAR SLAM system. For the front-end, roughly classified feature points (ground, facade, pillar, beam, etc.) are extracted from each frame using dual threshold ground filtering and principal components analysis. Then, the registration between the current frame and the local submap is accomplished efficiently by the proposed multi-metric linear least square ICP algorithm.

Point-to-point (plane, line) error metrics within each point class are jointly optimized to estimate the ego-motion. Static features of the registered frame are appended to the local map to keep it updated. For the back-end, hierarchical pose graph optimization is conducted among regularly stored history submaps to reduce the drift resulting from dead reckoning.

Lastly, F-LOAM [34] adopts a non-iterative two-stage distortion compensation method to reduce the computational cost of scan matching. For each LiDAR scan input, the edge and planar features are extracted and matched to a local edge map and a local plane map separately, where the local smoothness is also considered for iterative pose optimization. Despite not performing loop detection, F-LOAM is able to achieve accurate results.

2.3.3 Grid map-based laser SLAM

Grid map-based SLAM algorithms focus on the discretised representation of the environment, which can be either 2D, 2.5D, or 3D.

2D grid maps are associated with planar SLAM, typically used for indoor localization and mapping. They represent the space as a set of relatively small cells, forming a regular grid. Each cell is associated with a random variable $m_{x,y}$ which holds the probability of this cell being occupied.

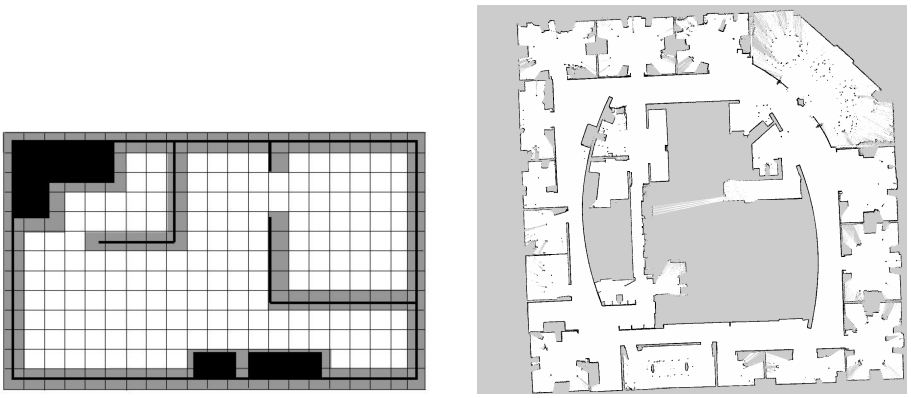


Figure 2.11: *On the left, 2D occupancy grid representation of a map; on the right, large scale 2D reconstructed map in an occupancy grid.*

Grid maps where an occupancy variable is associated with each cell are named occupancy grid maps. Sensor measurements determine the probability value of each cell, efficiently allowing the building of a 2D map during the SLAM process, as represented in Figure 2.11. Different SLAM systems can be considered state-of-the-art for 2D grid map-based laser SLAM.

Gmapping [35] uses a Rao-Blackwellized Particle filter SLAM approach. It is one of the most widely used 2D SLAM methods in robotics, especially for land-based mobile robots. In general, the particle filter family of the algorithm may require a large number of particles to obtain accurate results. Also, the depletion problem associated with this method decreases the accuracy of the algorithm. This arises with the elimination of a large number of particles, during the re-sampling step described in Section 2.1.

An adaptive re-sampling technique has been developed to minimize the depletion problem. Moreover, this approach takes into account not only the movement of the mobile robot but also the most recent sensor observation, with odometry information. This decreases the uncertainty of the robot pose in the prediction step of the particle filter.

Hector SLAM [36] does not require any auxiliary odometry sensor (e.g. wheel encoders) which directly measures the travel distance of a land-based robot, but only relies on the information from laser scan matching approaches. Therefore, Hector SLAM is more suitable for aerial vehicles. It takes advantage of the low distance measurement noise and high sampling rates of LiDAR sensors, using a fast scan matching method. Another advantage of Hector SLAM is its capability to generate multi-resolution grid maps to avoid singularities during point cloud registration.

Lastly, *Cartographer* [37] is another approach that provides real-time

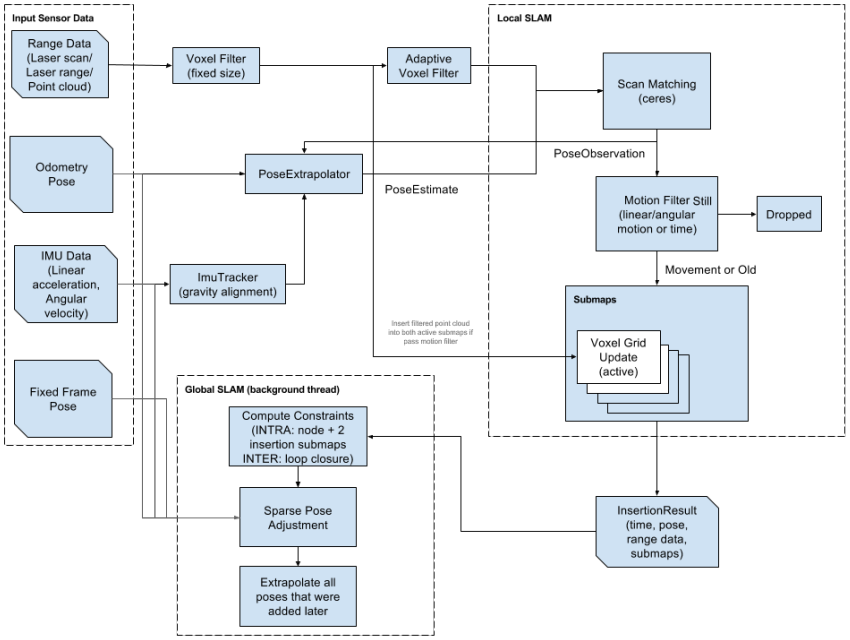


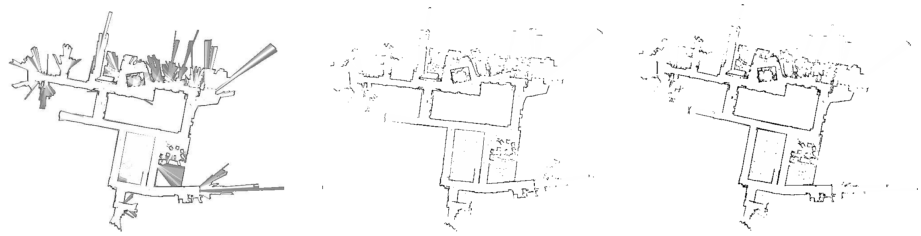
Figure 2.12: *Architecture of Google Cartographer [37], including 2D and 3D modules.*

SLAM in 2D and 3D across multiple platforms and sensor configurations, developed by Google. Cartographer does not require a particle filter algorithm for mapping and it overcomes the issue of error accumulation during long iterations by pose estimation against a recent submap.

Cartographer primarily consists of two subsystems, global SLAM, and local SLAM. Local SLAM is used to generate good submaps of a region and global SLAM is used to tie the submaps together as consistently as possible. Moreover, it generates a series of submaps, which are meant to be locally consistent, and it also creates the local trajectory. The local SLAM algorithm inserts the scan into the current submap via scan matching.

This process uses an initial guess from the pose extrapolation algorithm, which uses other sensors to make an initial prediction of where the scan should be inserted into the submap. Once registration is computed, a motion filter ensures that only scans resulting from significant motion (based on distance or angle, or time) are included in the submap.

A submap is considered complete when the local SLAM has received a given amount of range data. The local SLAM algorithm stores the submaps and their range data in a data structure known as a probability grid. The global SLAM algorithm takes the submaps and attempts to rearrange them so that they form a coherent global map and executes loop closure by scan



(a) (b) (c)

Figure 2.13: *Example of the different 2D Grid map-based SLAM systems on the 2011-01-28-06-37-23 MIT sequence. From left to right, maps are obtained with Cartographer [37], GMapping [35] and Hector SLAM [36], respectively.*

matching scans of sensor data against the submaps. Figure 2.12 shows the architecture of Cartographer, including also the 3D components.

Figure 2.13 represent the 2D occupancy map of the same environment reconstructed using Cartographer (Figure 2.13a), GMapping (Figure 2.13b) and Hector SLAM (Figure 2.13c), from left to right, respectively.

2.5D maps differ in many aspects from occupancy grids. Point cloud data is first rasterised in a 2D grid map. Let the spherical coordinates of the i^{th} LiDAR point p_i be $(\gamma_i, \phi_i, \theta_i)$, where γ is the depth of the point, ϕ is the horizontal angle and θ is the vertical angle. The 3D Cartesian coordinates of $p_i(x_i, y_i, z_i)$ are:

$$\begin{aligned} x_i &= \gamma_i \cdot \cos\theta_i \cdot \cos\phi_i \\ y_i &= \gamma_i \cdot \cos\theta_i \cdot \sin\phi_i \\ z_i &= \gamma_i \cdot \sin\phi_i. \end{aligned} \tag{2.7}$$

Each point is projected onto a 2D grid map, obtaining a new 2D point p'_i:

$$\begin{aligned} x'_i &= floor(\frac{x_i}{f_x} + c_x) \\ y'_i &= floor(\frac{y_i}{f_y} + c_y), \end{aligned} \tag{2.8}$$

where f_x and f_y are the grid map resolutions in the x and y axis respectively, and c_x and c_y are the centers of the grid map (half the number of row and columns of the grid). For each cell of the grid, the height z of all points in it is kept. This is done by considering the mean μ_i to represent the height expectation of each cell:

$$\mu_i = \frac{1}{n}\sum_{k=1}^{n} z_k \tag{2.9}$$

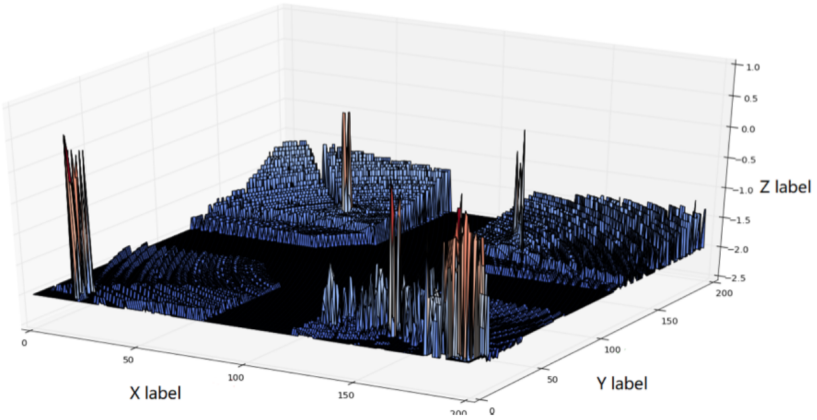


Figure 2.14: *Example of 2.5D map, which also shows the heights associated with each cell.*

where n is the number of cloud points projected into the grid cell i.

2.5D maps retain the advantages of easy storage and building as 2D grid maps and also represent the height information, which is crucial for registration in the outdoor environment. Furthermore, more precise models of height distribution can be adopted in these maps, as Gaussian mixture distributions. An example of a 2.5D map is represented in Figure 2.14.

DLO [38] adopts a 2.5D representation of the environment as it performs pose tracking by minimizing the height difference error between corresponding cells associated re-projections obtained from two consecutive scans. Instead of performing a point-wise or feature-wise matching, consistency is enforced between two height expectations $\mu_{i,t-1}$ and $\mu_{i,t}$.

Since DLO does not perform loop closure, a recent work [39] based on it, named *DL-SLAM*, has been proposed. The main contribution consists of the development of a loop closure detection method based on the matching of 2.5D segments obtained from the map. Segments are obtained by clustering grid cells with K-mean, assigning them different features: principal direction, height gradient, curvature, omni-variance, and eigen entropy.

The features of each segment in the current height map are matched with features of the segments in the map of the environment by calculating the score between the corresponding features vector and thresholding it.

Similarly to 2.5D maps, 3D grid maps, commonly referred to as voxel grid maps, are built by dividing the space into multiple cells of the same size and assigning to them cloud points according to their spatial location. Let $p_i(x_i, y_i, z_i)$ be a point belonging to a scanned point cloud. It is projected

onto a 3D grid map, obtaining a new 3D point p'_i such that:

$$\begin{aligned} x'_i &= floor(\frac{x_i}{f_x} + c_x) \\ y'_i &= floor(\frac{y_i}{f_y} + c_y) \\ z'_i &= floor(\frac{z_i}{f_z} + c_z), \end{aligned} \tag{2.10}$$

where f_x, f_y and f_z are the grid map resolutions in the x, y and z axis respectively; and c_x, c_y and c_z are the centers of the volumetric grid map.

Matching between two scans is then performed either by applying ICP on the cell points, meaning that only the mean locations are considered (highly reducing the computational cost of the algorithm), or exploiting the probabilistic information contained in each voxel, using the NDT [40] method. A detailed explanation of the NDT algorithm and its variants can be found at [41]. LiDAR odometry works based on NDT include [42] and [43].

2.4 Visual SLAM

When the main source of data is one or multiple cameras, the name used to describe the system is *visual SLAM*, or V-SLAM. Visual SLAM uses the images taken by the main sensor as observations of the environment [44].

In the last decade, SLAM using cameras has been actively discussed because the hardware configuration is simpler w.r.t. other sensors. Visual SLAM algorithms have been widely proposed in the field of computer vision, robotics [45, 46], and augmented reality [47]. Even though visual SLAM techniques have been proposed for different purposes, in different research communities, they share parts of implementation core ideas.

The typical architecture of a visual SLAM system does not differ much from the one presented in Section 2.2. As depicted in Figure 2.15, a generic workflow consists of multiple phases, divided into front-end and back-end.

Sensor data gathering. As already described at the beginning of the section, for visual SLAM systems, the gathered data comes mainly from a single camera (standard or RGB-D) or a pair of cameras (stereo setup).

Visual odometry (VO) [48]. This task is part of the front-end of a visual SLAM system, as it estimates the camera movement (hence, the trajectory of the robot) between adjacent steps, while generating a rough map of the environment (usually in the form of 3D points in the space), using features associated to the input images. In other words, visual odometry comprises short term data association and motion tracking. Visual SLAM systems

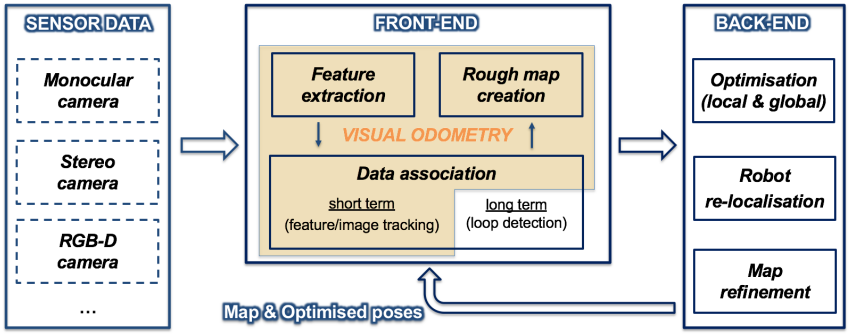


Figure 2.15: *Visual SLAM architecture. Modules covered by the orange box form visual odometry, the process of finding consecutive robot motions by using visual information.*

generally take as input images from calibrated cameras, so that intrinsic and distortion parameters are known. Therefore, a camera pose is equivalent to extrinsic camera parameters, w.r.t. a chosen coordinate system.

Loop closure detection. As described in Section 2.2, it is the task of determining whether the robot is visiting an already discovered location. In visual SLAM, this is done by checking similarities between pairs of images.

Optimization. The camera/robot poses and the loop closures are used to generate an optimized trajectory and map, eliminating the accumulated drift.

From a high-level point of view, visual SLAM is the combination of visual odometry, loop closure, and optimization [49,50]. While in pure visual odometry the geometric consistency of a map is considered only in a small portion and the camera motion is often computed without creating a map, in visual SLAM the global geometric consistency of a map is considered, as the robot trajectory is estimated and grows in size. This is done by performing both global and local optimization on the poses of the robot, tracked in the visual odometry part of the front-end, and the detected loop closures.

The front-end of visual SLAM is composed of three consecutive steps. First, it is necessary to define a certain coordinate system for camera pose and map estimation of an unknown environment. Therefore, in this initialization step, the global coordinates system is defined, and a part of the world is reconstructed as an initial map using pre-processed visual data. After the initialization, tracking and mapping are performed to continuously estimate camera poses, at each time step, exploiting the sensor measurements.

During the tracking phase, either the reconstructed map, the previous image, or both, are tracked in the current image to estimate the current camera pose. To do so, 2D-to-3D or 2D-to-2D correspondences are obtained, between the image and the map or between two consecutive images,

respectively. Then, the camera pose is computed from the correspondences by solving the Perspective-n-Point (PnP) problem [51]. Lastly, the estimated map is updated with the new areas of the environment seen by the robot.

The back-end of a visual SLAM system is tasked with the *re-localization* of the robot and the optimization of trajectory and map. Differently from laser SLAM systems, here re-localization is required, to cope with possible front-end failures, which may be caused by fast camera motion or images with low information (e.g., a white straight wall cannot be used for tracking). The front-end temporarily stops the motion estimation procedure and the current camera pose is computed w.r.t the map, to re-localize the robot.

The map and the estimated poses include accumulated errors, to be corrected enforcing consistency between the map and the images. Similarly to any SLAM system, the strongest type of consistency is given by loop closures. As described in Section 2.2, loop detection is performed by searching similarities between the current image and previously acquired images or between features extracted from the current image and the re-projection of the map features on it. In case a loop is found, the cumulative error that occurred during camera movement can be estimated and used in the optimization step. It should be noticed that loop detection and re-localization can be performed using, more or less, the same techniques.

Following different criteria, visual SLAM systems can be classified in multiple ways. The first classification is done depending on how visual odometry is performed: *feature based methods* [52–54] estimate the trajectory by matching points of interest in consecutive images, while *direct methods* [46, 55] compute the motion of the camera directly according to pixel information (brightness, color). A second classification is done considering which particular camera setup is used in the SLAM system, which can be *monocular* (only one camera), *stereo* (two nearby cameras), or *depth based* (e.g., structured light scanners, or Time of Flight 3D cameras).

2.4.1 Classification based on visual odometry

In the majority of visual SLAM systems, natural features present in the world have been used, such as corners, interest points, or edge segments. These features are extracted from the input images and from the estimated map of the environment. The features obtained from the current image are then compared and matched against the features of the map or the features from the previous image, to find the transformation corresponding to the current camera pose, hence upgrading the trajectory. These methods are known as feature-based, and they represent the map as a set of features.

In contrast to feature-based methods, direct methods directly use a whole input image to estimate the current camera pose, and for this reason, they are called featureless approaches. They optimize the geometry on the intensities by performing image alignment (differently from feature alignment). In addition to higher accuracy and robustness, in particular, in environments with few features, this provides information about the geometry of the environment, which can be very valuable to create semi-dense 3D maps.

Engel et al. [56] provided a good comparison between the two approaches, highlighting their advantages and disadvantages. Feature-based methods are more robust to geometric noise, but the features extraction and matching is time consuming. Moreover, the obtained representation of the environment is sparse and cannot be used for a detailed 3D reconstruction.

Direct methods perform better in low-texture regions but are generally more sensitive to dynamic objects. Moreover, none of the above methods can solve the problems caused by common dynamic objects in the scenes.

Feature-based visual odometry

Feature-based methods rely on a particular representation of the images, seen as a set of features. Before continuing with the discussion, some definitions are necessary to better understand this category of algorithms.

Feature points are structures formed by two elements, a keypoint, and its descriptor. A *keypoint* is the location of a salient point in an image, represented in the form of 2D coordinates, containing also useful information, such as scale and orientation. A *Descriptor* represents the statistics and values about the pixels surrounding the keypoint (e.g., intensity).

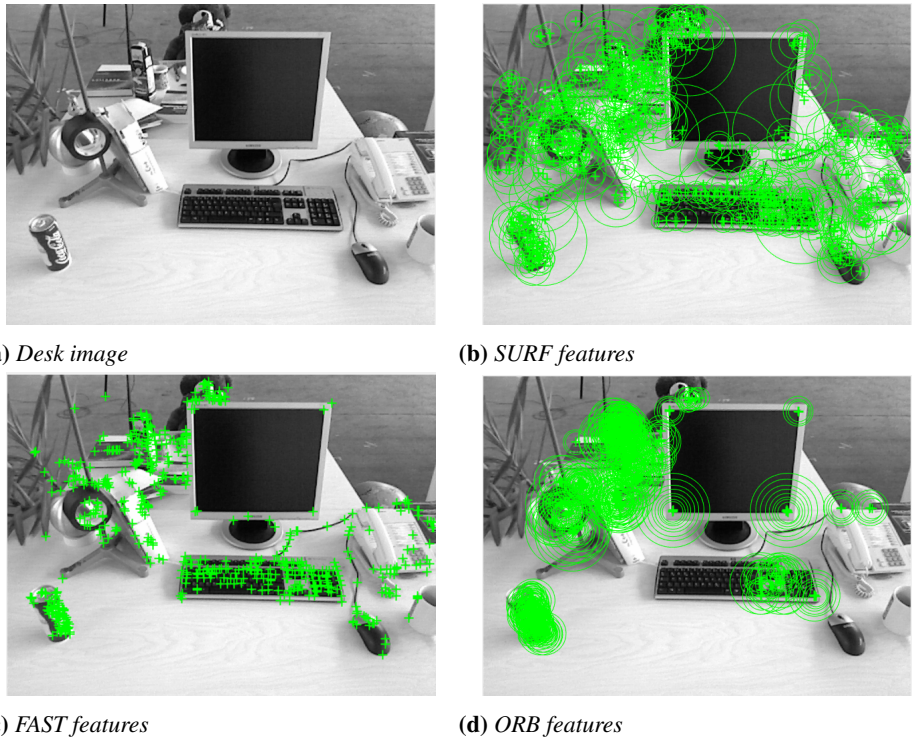
The steps of visual tracking for feature-based techniques are as follows.

1. Extraction of a set of sparse features from the input images and/or the estimated map, which can be points, edges, planes, or small areas.

2. Matching of the features of the current image with the features obtained from the previous image, or from the estimated map (data association).

3. Motion computation and tracking, by using the matched features to find the transformation that leads to the current camera pose.

Different state-of-the-art algorithms are used to perform feature extraction. The *SIFT* detector [57] is based on the Difference-of-Gaussians (DoG) operator, and feature points are found by searching local maxima using DoG at various scales of the images. SIFT is robustly invariant to rotations, scale, and affine variations but its main drawback is high computational cost.



(a) *Desk image* **(b)** *SURF features*

(c) *FAST features* **(d)** *ORB features*

Figure 2.16: *Results of feature detection methods on an image taken from the Desk sequence of the TUM Vision Dataset [65]. From top to bottom, left to right: desk image, SURF features, FAST features, and ORB features.*

The *SURF* detector [58] is based on the determinant of the Hessian Matrix computed at a point (u, v) in the image, at scale σ, which is used as a measure of the local change around the point. SURF features are invariant to rotation and scale but they have little affine invariance, w.r.t. SIFT detectors. The main advantage of SURF over SIFT is its low computational cost.

The *FAST* corner detector [59, 60] uses a circle of 16 pixels to classify whether a candidate point p is actually a corner. If a set of N contiguous pixels in the circle are all brighter than the intensity of candidate pixel p, plus a threshold value t, or all darker than the intensity of candidate pixel p, minus threshold value t, then p is classified as a corner point.

Other famous feature detectors are *ORB* [61], *BRIEF* [62], *BRISK* [63] and *KAZE* [64]. Figure 2.16 shows some of the features detection methods on the Desk sequence of the TUM Vision Dataset [65]. More detailed explanations on some of the mentioned detectors are available at [66–68].

In the features matching step, the descriptors associated with all the

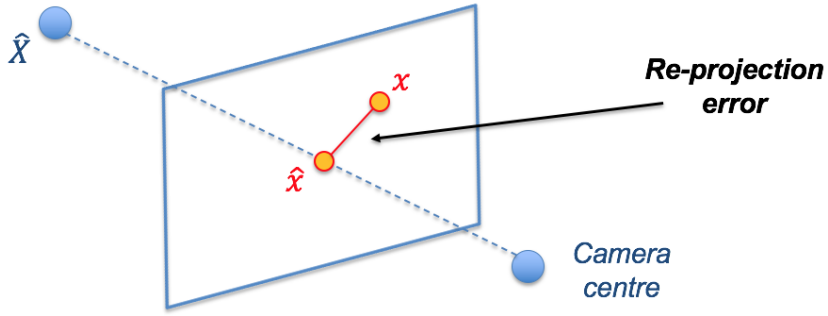


Figure 2.17: *Re-projection error of point $\hat{X}$ using the camera parameters. The location of the measured 2D point x lies at a certain distance from the re-projection of the corresponding 3D point $\hat{X}$, which is $\hat{x}$. This distance represents the re-projection error.*

extracted features from consecutive images (or from the current image and the estimated map) are confronted and matched. The simplest way to match features between two images is to compare all descriptors in the first image to all other feature descriptors in the second image using a similarity measure.

The pose corresponding to the current image is calculated through different methods. One of them is the RANSAC [69, 70], an iterative algorithm used to estimate the parameters of a model such that they best fit a set of data corrupted by many outliers. Another technique consists in minimizing the re-projection errors of the features, represented in Figure 2.17.

The re-projection error is a geometric error corresponding to the distance between a projected point and a measured one. Let P be the projection matrix of a camera and $\hat{x}$ the image projection of an estimated point $\hat{X}$. The re-projection error of $\hat{X}$ is given by $d(x, \hat{x})$, which is the distance between the image projection of the measured point x in the image, and $\hat{x}$.

Features are extracted from two images, I_i and I_j, and matched, obtaining pairs of imperfect 2D-to-2D correspondences. The problem of finding the transformation between the poses corresponding to I_i and I_j can be formulated as follows. The goal is to find a homography $\hat{H}$ that transforms the features in I_i to the features in I_j, corresponding to some 3D points.

This is done by minimizing the re-projection error function, defined as:

$$\sum_i d(x_i, \hat{x}_i)^2 + d(x'_i, \hat{x}'_i)^2, \tag{2.11}$$

where d is the re-projection error, x_i and x'_i are the image locations of point $\hat{X}$, and $\hat{x}_i$ and $\hat{x}'_i$ are the image points obtained by re-projecting $\hat{X}$ using the

estimated camera extrinsic parameters, i.e., its rotation and translation.

Feature-based methods have many disadvantages. The first is that storing the processed features can quickly become very costly. However, since this method eliminates all data that cannot be used (non-features), it is typically faster than direct methods. Then, it is difficult to extract features in poorly textured environments, leading to mandatory re-localization, which degrades visual SLAM performance. Lastly, feature-based methods usually originate sparse maps, making 3D reconstructions harder to obtain.

SLAM systems that exploit visual features can be further classified into filter-based and keyframe-based. *Filter-based* SLAM systems fuse measurements from all images sequentially by updating probability distributions over features and camera pose parameters. Localization and mapping are intertwined: the camera pose, with the entire state of all features in the map, are tightly joined and need to be updated at every processed image.

On the other hand, in *keyframe-based* SLAM systems, localization, and mapping are separated into two steps: the first takes place on consecutive images over a subset of the map, while the optimization takes place only on keyframes. As we have previously mentioned in laser SLAM 2.3, keyframes are data structures that contain useful information for SLAM, including the estimated pose of the robot, the corresponding input data, and so on.

Keyframes are generated during the tracking procedure, adopting at least one of the following criteria. If a large motion (either rotation or translation) is detected between the current input and the previous keyframe, a new keyframe is generated. Moreover, if enough time has passed since the creation of the last keyframe, a new one is built using the current data. Strasdat et al. [71], compared keyframe-based methods and filter-based methods, and found that the first has higher accuracy than the latter, under the same computational cost condition, but they are not scalable or extendable.

PTAM (Parallel Tracking and Mapping) [54] is a known SLAM keyframe-based method, as it marked the beginning of multi-threaded SLAM. In PTAM, the initial map is reconstructed using the five-point algorithm [72]. Tracking and mapping are separated as two independent tasks, executed in parallel. In the tracking thread, mapped points are re-projected onto an image to make 2D-to-3D correspondences using texture matching.

From there, the camera poses can be computed. In the mapping thread, scenery points are reconstructed by triangulation, at certain times, determining the 3D locations of feature points. Lastly, all keyframes, selected depending on the spatial distance between the correspondent cameras, and map points are adjusted using local and global bundle adjustment. The camera re-localization algorithm [73] adopted uses a randomized tree-based

feature classifier for searching the nearest keyframe of an input image.

Recently, an improved version of PTAM has been proposed, named *S-PTAM* [74], based on stereo vision. The system defines the global reference frame at the camera pose in the first frame of the sequence. An initial map is then estimated by matching and triangulating salient point features in the first stereo pair. For every consecutive image pair, the tracking thread estimates the corresponding camera poses by minimizing the re-projection errors between the projected map points and their image correspondences.

The system selects a subset of keyframes that is later used in a second thread, to estimate the map at a lower rate. The map points are triangulated from the stereo matches of each keyframe and added to the map. The mapping thread is constantly minimizing the local re-projection error by refining all the map points and the stereo poses using bundle adjustment.

The global consistency is enforced by modeling the SLAM process as a graph. Point correspondences are actively searched between keyframes in order to strengthen the constraints of such graph. S-PTAM runs a loop closure detection in a third thread, which searches for loop closure candidates using the visual appearance of features. The relative motions of loop closures are added to the graph, which is optimized to accommodate such constraints.

ORB-SLAM [52] is another feature-based system that uses and improves the PTAM framework in multiple ways. ORB-SLAM uses ORB [61] features rather than FAST [59, 60] corners, which have better angle invariance. ORB-SLAM adds a loop closing mechanism to eliminate error accumulation and it selects keyframes in a different, more refined, way.

Later, the same author proposed the *ORB-SLAM2* algorithm [53], which increases the support for stereo cameras and RGB-D cameras (similarly to S-PTAM [74], where a stereo configuration is supported), although the latter case is part of the sensor fusion SLAM category (and not purely visual).

Very recently, the system evolved in *ORB-SLAM3* [75] (Figure 2.18). The first main novelty of the method is a feature-based tightly-integrated visual-inertial SLAM system that fully relies on Maximum-a-Posteriori (MAP) estimation, even during the IMU initialization phase. The result is a system that operates in real-time, in small and large, indoor and outdoor scenarios, and is two to ten times more accurate than previous approaches.

The second novelty is a multiple map system that relies on a new place recognition method with improved recall. Thanks to it, ORB-SLAM3 is able to survive long periods of poor visual information: when it gets lost, it starts a new map that will be seamlessly merged with previous maps when revisiting mapped areas. Compared with visual odometry systems that only use information from the last few seconds, ORB-SLAM3 is the first system able

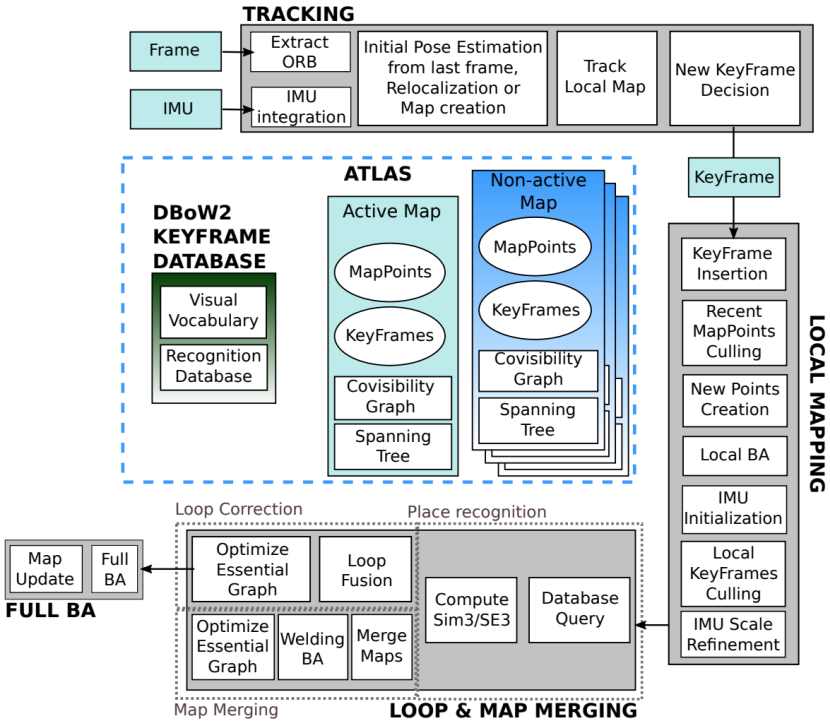


Figure 2.18: *System architecture of ORB-SLAM3 [75].*

to reuse, in all the algorithm stages, all previous information. This allows to include in bundle adjustment co-visible keyframes, that provide high parallax observations boosting accuracy, even if they are widely separated in time or if they come from a previous mapping session.

Linyan et al., recently proposed *Semantic Optical Flow SLAM (SOF-SLAM)* [76], a visual semantic system toward dynamic environments, which is built on the RGB-D mode of ORB-SLAM2. The system utilizes the complementary characteristic of motion prior information, from semantic segmentation of images, and motion detection information, from epipolar constraints, proposing a new algorithm named semantic optical flow.

In this way, SOF-SLAM can remove dynamic features effectively, leading to more accurate results when estimating the camera poses. Figure 2.19 represents the overall architecture of SOF-SLAM. One can immediately notice the similarities with ORB-SLAM3, by comparing Figure 2.19 and Figure 2.18. This must not be a surprise, since they are both built on ORB-SLAM2, which is considered state-of-the-art in feature-based visual SLAM systems and the majority of recent works rely on its architecture.

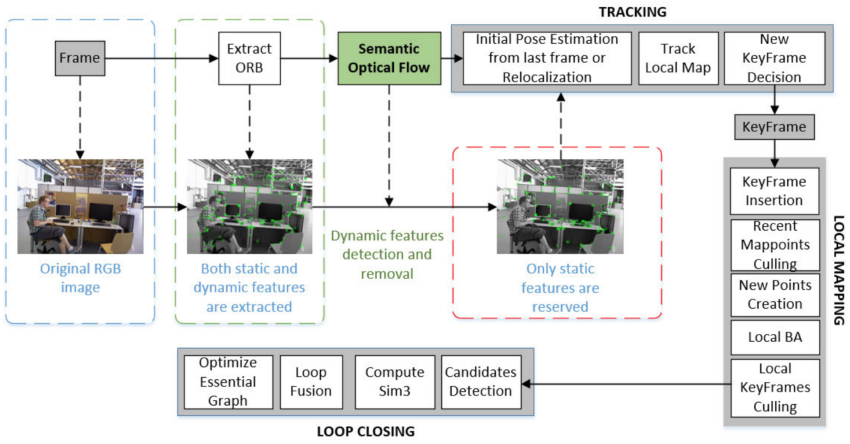


Figure 2.19: *System architecture of SOF-SLAM [76]. Similarities with the architecture of ORB-SLAM2 [53] are noticeable.*

Other methods, such as *SPM-SLAM* [77] and its successor *UcoSLAM* [78], solve some of the limitations of standard visual SLAM systems, such as unknown map scale, failure when the movement is only rotational and the need of rich textured environments, by using squared fiducial markers (squared patterns with a unique ID) instead of natural features.

Markers are freely placed in the environment and the two methods are able to create a map of them. Camera localization can be done in the correct scale by observing a single marker. The main drawback of these systems is that they require the physical placement of a lot of elements in the environment, in order to build the map. UcoSLAM, however, can perform well using markers only, feature points only, or even a combination of both.

Direct visual odometry

Direct methods, also known as featureless approaches, use all image pixels (dense approach [55]), pixels with sufficiently large intensity gradient (semi-dense approach [46]) or sparsely selected pixels (sparse approach [56, 79, 80]) to minimize the photometric error obtained by direct alignment of the selected pixels, estimating camera poses and pixel depths.

In other words, direct algorithms minimize the per-pixel intensity difference. Given two images, I_i and I_j, the corresponding motion $T_{i,j}$ is computed as:

$$T_{i,j} = \underset{T}{\operatorname{argmin}} \sum_i ||I_j(u'_i) - I_i(u_i)||^2 \tag{2.12}$$

$$u'_i = P(T \cdot (P^{-1}(u_i) \cdot d_i), \tag{2.13}$$

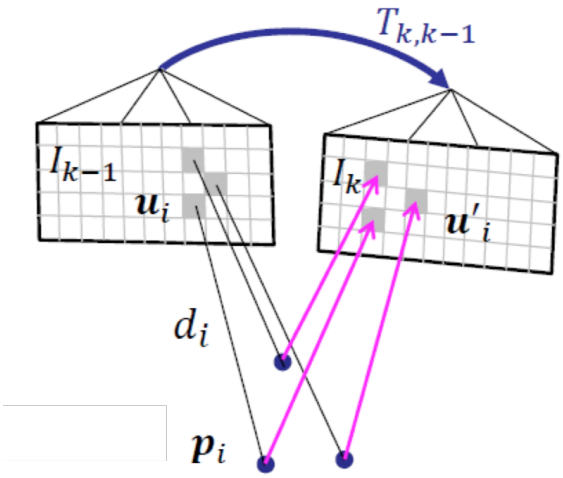


Figure 2.20: *Photometric error minimization scheme. Pixels/regions in the left image $k-1$ correspond to space points located at certain depths, which are re-projected in the right image k. The intensity difference between the original and re-projected elements is the photometric error of the 2D points.*

where P is the camera projection matrix, u_i is the pixel in image I_i and u'_i is the corresponding pixel in image I_j, obtained through the re-projection of the 3D point corresponding to u_i and located at space depth d_i.

The scheme for photometric error minimization is represented in Figure 2.20. It is important to notice that, instead of a pixel-by-pixel, another approach is to consider small image regions, as represented in the figure.

Different from feature-based methods, direct approaches are more robust in low-texture scenes and can deliver dense maps. However, they suffer from illumination changes, which reduce their performance.

In *DTAM* [55], tracking is done by reducing the photometric error between the current image and multiple synthetic images generated from the estimated map, whose initial creation is done similarly to PTAM [54], using a stereo measurement. Depth information is then estimated for every pixel by using multi-baseline stereo [81], and then, it is optimized by considering space continuity. DTAM is robust to feature deletion and image blur, but it cannot achieve real-time computation without a GPU.

Semi-direct Visual Odometry (SVO) [79] is a hybrid method combining the feature-based approach and the direct technique. As PTAM [54], tracking and mapping are separated into two threads. In the tracking thread, the motion between two consecutive images is obtained by minimizing the photometric errors. Mapping is then performed by minimizing the re-projection

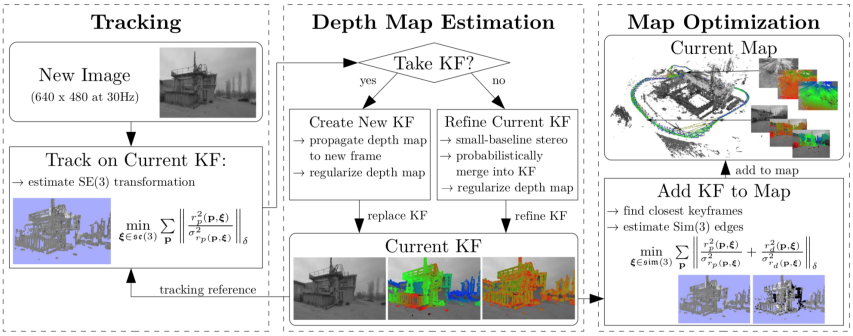


Figure 2.21: *High-level system architecture of LSD-SLAM [46].*

error between the obtained feature points and predicting their optimized positions in the images. SVO can be seen as a sparse version of DTAM.

In a more recent work, Engel et al. proposed the *Direct Sparse Odometry (DSO)* method [56]. Different from SVO, DSO is a completely direct approach. The front-end of DSO tracks the camera pose by matching the current frame and the keyframe, which is the same as for feature-based SLAM systems, by minimizing the photometric errors. In order to suppress the accumulated error, DSO removes error factors as much as possible, from geometric and photometric perspectives, by proposing a photometric camera calibration method. Later, a stereo version of DSO has been proposed [82].

LSD-SLAM [46] is part of the state-of-the-art in direct methods. The idea of LSD-SLAM follows the working principle of SVO. First, random values are set as initial depths for each pixel in the image. Then, camera motion is estimated by synthetic view generation from the reconstructed map.

Reconstructed areas of the maps are limited to high-intensity gradient areas in the images. Lastly, global optimization is employed to obtain a geometrically consistent map. The architecture of LSD-SLAM is represented in Figure 2.21. LSD-SLAM was born as a monocular SLAM system but has been later extended to stereo and omnidirectional cameras [83, 84].

A recent work based on DVO, named *VI-DVO* [85], integrates cameras with IMU measurements. Camera poses and sparse scene geometry are jointly estimated by minimizing photometric and IMU errors in a combined energy function. The visual part of the system performs bundle adjustment as optimization on a sparse set of points, but unlike feature-based systems, it directly minimizes the photometric error between consecutive images.

This makes it possible for the system to track not only corners but any pixels with large enough intensity gradients. IMU information is accumulated between several frames using measurement pre-integration and is

inserted into the optimization as an additional constraint between keyframes. VI-DVO performs partial marginalization of old variables (e.g., poses) so that updates can be computed in a reasonable time.

2.4.2 Classification based on camera type

Cameras are cheaper w.r.t. other sensors, have low weight, small size, and provide rich information about the scene. They can be divided into multiple categories [86], including *monocular* configuration, where only one camera is used to collect images, *stereo* configuration, where two cameras placed nearby simultaneously gather pairs of pictures, and *RGB-D* sensors, which collects images enriched by depth values for each pixel.

In monocular SLAM, a single camera, usually having 6 DoF (degrees of freedom), is the sole input sensor of the system. While range sensors, such as LiDAR, provide range and angular information, cameras are projective sensors, which measure the location of image features. For this reason, depth information cannot be obtained through points from a single image and requires multiple consecutive views. Examples of monocular SLAM systems are *Mono-SLAM* [87], *PTAM* [54], *ORB-SLAM* [52] and *DSM* [88].

A stereo camera consists of two synchronized monocular cameras, placed at a known distance from each other, the *baseline*. Since the physical distance of the baseline is available, the 3D position of each pixel can be found by performing a simple triangulation. A stereo camera setup is usually computationally expensive in the estimation of the depth for each pixel.

The depth range measured is related to the baseline length, and the longer a baseline is, the farther it can measure with a given accuracy. For this reason, stereo cameras mounted on robots have often a large size (the whole setup) but can be applied without the need for other sensors. The disadvantage of such a setup is, as already stated, the complexity of its configuration, the noticeable computational cost, and the low accuracy of the estimated depths.

There are two main advantages of using a stereo configuration for visual SLAM. First, stereo avoids scale ambiguity, inherent in monocular visual SLAM, especially during the beginning of the SLAM process, when little information of the environment is known. Second, there is no need for tricky initialization procedures of the map, since a stereo pair of images gives an estimate of the depth of points. Examples of stereo SLAM systems are *LSD-SLAM* [46, 83], *SVO* [80], *ORB-SLAM2* [53] and *ProSLAM* [89].

Lastly, an RGB-D camera, also known as *depth camera*, shares both the characteristics of a monocular camera and a laser scanner, providing an RGB image along with the depth of each pixel of the image, visible in

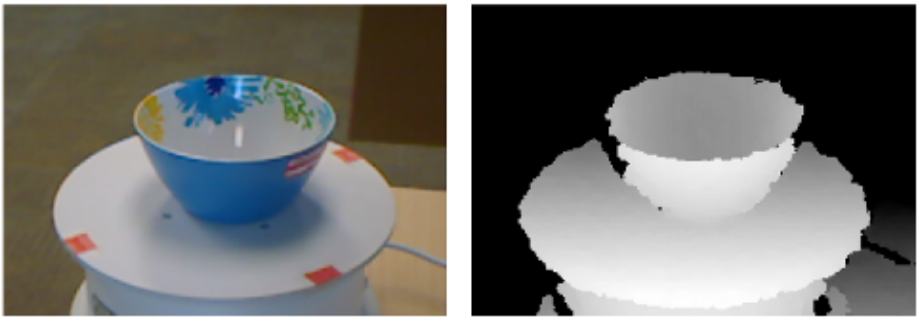


Figure 2.22: *Example of RGB-D image: the colored image on the left is associated with a depth image containing the depth values of each of its pixels, on the right.*

Figure 2.22. Relying on the infrared structure of light and Time-of-Flight principles, it is able to measure the distance between itself and an object by emitting light and receiving it back after having encountered a surface.

Differently from a stereo camera, the computation of the depth is not done *by code*, but by the sensor itself, greatly reducing the cost and showing an advantage over a stereo setup. However, depth cameras have many disadvantages. The gathered data is often noisy, cannot measure transparent objects, and is heavily influenced by sunlight. Moreover, the small field of view and measurement range limit depth cameras to indoor environments, making them not suited for outdoor applications (e.g. urban autonomous driving). Generally, RGB-D sensors are mainly used for indoor environments, as their technology limits their range from 1 to 4 meters.

By using RGB-D cameras, the 3D structure of the environment, along with its texture, can be obtained directly. In addition, different from monocular visual SLAM algorithms and similar to the stereo case, the scale of the coordinate system is known because the 3D structure can be acquired.

The basic framework of depth-based visual SLAM is as follows. Pose tracking is achieved by minimizing both geometric and photometric errors, with the aid of a scan matching algorithm (because the depth information is similar to the measurements of laser sensors). Then, the 3D structure of the environment is reconstructed by combining multiple depth maps.

Newcombe et al. proposed *KinectFution* [90], in 2011. In KinectFusion, a voxel grid is used for representing the 3D structure of the environment. This is reconstructed by combining obtained depth maps in the voxel space. Camera motion is estimated by matching the computed map and the input depth image. KinectFusion is implemented on GPU to achieve real-time.

Salas-Moreno et al. proposed an object-level RGB-D Visual SLAM

algorithm, named *SLAM++* [91]. In this method, several 3D objects are registered in the database in advance, and these objects are later recognized in an online process. In this way, the estimated map is continuously refined, and 3D points are replaced by 3D objects to reduce the amount of data. Lastly, Schops et al. [92] proposed an RGB-D-based variant of bundle adjustment for SLAM systems, included in their system, *BAD-SLAM*.

An extensive evaluation of RGB-D SLAM systems has been recently presented by Zhang et al. [93]. With their work, they mainly introduced the basic concept and structure of a typical RGB-D SLAM system and then discussed the differences, and open problems, between the various RGB-D SLAM methods existing in literature, distinguishing them by tracking, mapping, and loop detection. In the end, the authors conducted a large number of evaluation experiments on multiple RGB-D SLAM algorithms and analyzed their advantages and disadvantages, as well as performance differences in different application scenarios (e.g., indoor).

2.5 Radar SLAM

Radar, which is the acronym for radio detection and ranging, is a sensor that exploits radio waves to determine the velocity, range, and angle of surrounding objects w.r.t. itself. Radars are long-range active sensors, which are immune to poor weather conditions and can operate in low-texture environments, making them preferable over laser rangefinders or even cameras in these situations. For these reasons, various approaches have been proposed in the literature to estimate the ego-motion of ground and aerial vehicles based solely on radar measurements.

Radar sensors are available in two forms: pulse and continuous wave (CW). A pulse radar sensor emits short and powerful pulses and receives echoing signals during silent periods. CW radars, and in particular frequency-modulated continuous wave (FMCW) radars, transmit a steady stream of continuous wave signals, of a frequency that varies linearly with time. The main difference between the two is that a continuous wave radar can generate high-resolution images, while pulse sensors suffer from blind spots.

In particular, continuous wave radars have attracted a lot of attention in the fields of localization and object avoidance due to their characteristics, such as low power consumption, their usability in different weathers, and, more importantly, their minimum target range and maximum distance.

Radar sensors can be further categorized into automotive and scanning, depending on the working principle of their underlying hardware (Figure 2.23). With Doppler information, automotive radars offer radial velocity

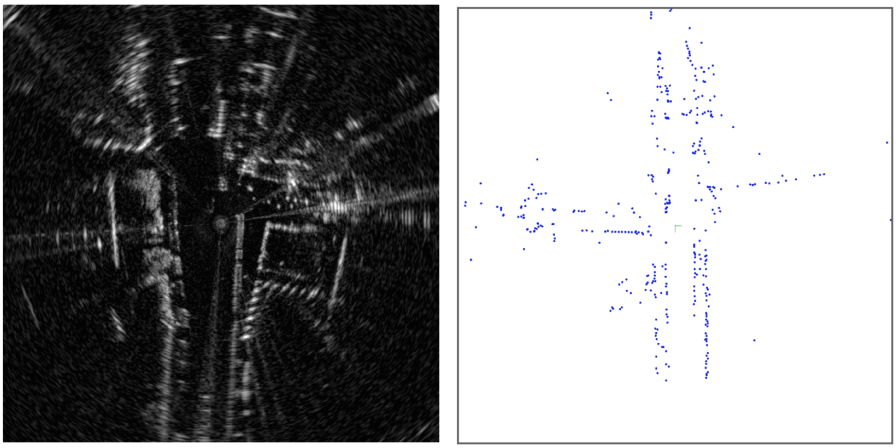


Figure 2.23: *On the left, an image obtained from a scanning sensor; on the right, a 2D point cloud derived from an automotive radar. The difference in density is noticeable.*

measurements. However, these data are with relatively low accuracy and are sparse, since only highly confident measurements are output. In contrast, scanning radars provide raw polar power-range images (i.e., images with each row representing the sensor reading at every azimuth and each column representing the raw power return at a particular range) with relatively high angular and range resolution. Nevertheless, radar images from scanning radars include noise and do not provide velocity information.

An early work on odometry and mapping (not to be mistaken with SLAM) using automotive radars is proposed by Kellner et al. [94]. They use radial velocity measurements to estimate the ego-velocity of the robot, while simultaneously removing outlier elements via RANSAC [69], according to the radial velocity of each 2D point estimated with the radar.

From there, new methods directly extracted point clouds from sensor data, to be later matched using associated descriptors (i.e., combining the ideas of visual and laser methods). Some examples are given by [95], which adopts Binary Annular Statistical Descriptor (BASD) visual descriptors, or the work in [96], which performs scan matching using ICP.

Rapp et al. [97, 98] later applied joint Doppler clustering-based NDT, which weights each radar point by the similarity between the measured and estimated velocities, removing outliers. Nevertheless, radar odometry methods based on cloud matching could fail when the set of points extracted from a radar input is too sparse, which is typical of automotive radars.

Recently, Holder et al. [99] proposed a system based on radar submaps, i.e., a combination of several radar scans. These are used to handle the

sparseness issue mentioned above, and tracking is performed via submap-to-submap matching, using ICP. However, this method does not consider the uncertainty of ego-motion, which influences the readings of the used sensor.

To avoid the sparsity issues of automotive radars seen before, researchers shifted their focus to the use of scanning sensors, which offer detailed power-range images. Earlier works leveraged constant false alarm rate (CFAR) [100] to extract features and associate them by minimizing a cost function defined by the distortion model of the scanning radar [101].

Later, vision-inspired works, such as [102, 103], exploited feature extractors and descriptors directly on radar images. Cen et al. [104, 105] proposed a new feature extraction approach (along with an update) and a descriptor for radar odometry, and used a shape similarity metric to remove outliers. More recently, the work in [106], also known as YETI radar odometry, included motion distortion and Doppler effect correction in their proposed pipeline.

Up to this point, one may have noticed that almost all works in literature are odometry and mapping methods, and not SLAM systems. The main reason is that loop detection using radars is inherently difficult (independently of the type of the sensor). Using a feature-based approach is out of the question, for the following reasons. First, radar images have less distinctive characteristics on pixels compared with camera images, which means similar feature descriptors can be repeated widely across radar images.

Second, the multi-path reflection problem in radar can introduce ambiguity for the feature descriptor. Lastly, a small rotation of the radar sensor may produce tremendous scene changes, significantly distorting the histogram distribution of the descriptors. On the other hand, directly performing point cloud alignment on clouds extracted from radar images does not yield accurate results, being the scans cluttered with noisy elements and outliers.

Holder et al. [99] presented a real-time pose graph-based SLAM system for automotive radars. In particular, loop closure is achieved by applying a technique for place recognition, originally thought for 2D LiDAR point clouds, to radar scans. In order to find the transformation between a candidate pair of submaps, a variant of RANSAC is applied, to find the largest set of feature correspondences. The transformations of the remaining pairs are then refined using their full point clouds for ICP matching.

2.6 Hybrid SLAM

Both visual and laser SLAM have advantages and disadvantages. Visual SLAM systems are easier to set up; plenty of algorithms have been developed to implement them in accurate and efficient ways; loop closure can be

performed with better performance; and input data can be efficiently stored. However, they require ad-hoc techniques to reduce the errors introduced by visual odometry, and the environment is usually represented as a sparse set of features, not suitable for 3D reconstruction. Moreover, input sensor measurements are highly affected by changes in illumination and view.

Laser SLAM solves many of these problems because LiDAR scans that are used for registration and map creation, output structures richer of information w.r.t. simple images and suitable for reconstruction, as they are dense representations of the environment. Laser SLAM technology needs to overcome the problems of large computation and motion distortion effects.

To get the best from both worlds, multi-sensor SLAM systems have been proposed in the last decade, becoming the trend in SLAM. Thanks to the advancement of technology, the decreasing prices and the standards in sensor configuration have been a huge impact factor on the development of SLAM systems, and multiple sensors are no longer seen as a limitation.

Due to the great variety of multi-sensor SLAM systems proposed in recent years, instead of describing the various steps and classifications of the architectures, as done for visual and laser SLAM, in this section only a few works are outlined in detail, to highlight the adopted methods and the way sensor data is used together to achieve accurate results.

2.6.1 ElasticFusion

The approach used by *ElasticFusion* [107, 108], whose architecture is represented in Figure 2.24, is grounded in estimating a dense 3D map of an environment, explored with a standard RGB-D camera, in real-time. As RGB-D cameras provide both visual information (RGB images) and depth information (depth image), systems built upon them are considered as multi-sensor systems. The key elements of ElasticFusion are as follows.

1. Estimation of a fused surfel-based model of the environment, divided into active and inactive parts, depending on their location.
2. Tracking and fusion of input data.
3. Detection of local loops, at every time step, by searching for common elements in the map.
4. For global loops, inclusion of predicted synthetic views of the scene to a randomized encoding database to find a matching predicted view.

Figure 2.25 provides an example visualization of the outlined main steps of ElasticFusion. Initially, all data is in the active model as the camera

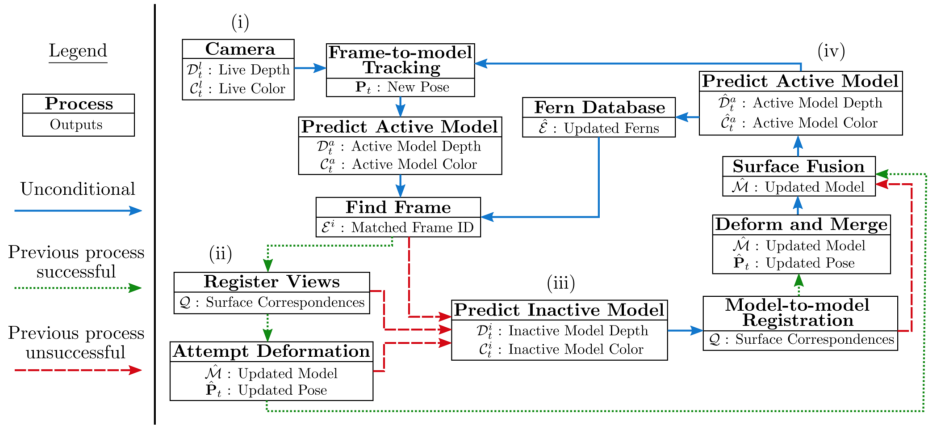


Figure 2.24: *Complete system architecture of ElasticFusion [107, 108].*

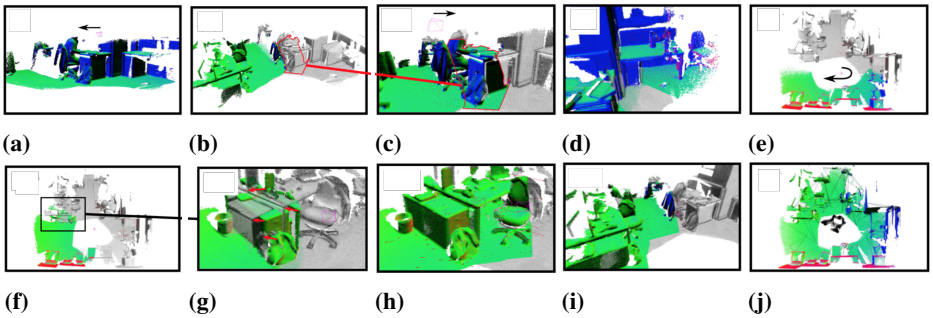


Figure 2.25: *Visualization of the different steps of ElasticFusion [107, 108].*

moves left (Figure 2.25a). As time goes on, the area of the map not seen recently is set to inactive, highlighted in red (Figure 2.25b).

Then, the camera revisits the inactive area of the map, closing a local loop and registering the surfaces together (Figure 2.25c). The previously highlighted inactive region then becomes active (Figure 2.25d). Camera exploration continues to the right and more loops are closed, continuing to new areas (Figure 2.25e), repeating the whole deactivation procedure.

Following this, the camera revisits an inactive area (Figure 2.25f) but has drifted too far for a local loop: the misalignment is apparent, with red arrows visualizing equivalent points from active to inactive (Figure 2.25g). For this reason, a global loop detection is triggered, which aligns the active and inactive models (Figure 2.25h). The exploration continues as more local loops are made and inactive areas are reactivated (Figure 2.25i). Lastly, the final full map, colored with surface normals, is computed (Figure 2.25j).

The scene is represented as a set M of surfels, data structures having the following attributes: a position $p \in \Re^3$, a normal $n \in \Re^3$, an RGB color $c \in \mathbb{N}^3$, a weight $w \in \Re$, a radius $r \in \Re$, the initialization timestamp t_0 and the last updated timestamp t. The radius of each surfel is intended to represent the local surface area around a given point while minimizing visible holes. A surfel in M is declared inactive when the time since it was last updated (i.e., had a raw depth used for fusion) is greater than δ_t.

The way sensor fusion is achieved consists in combining the pose tracking obtained from images and from depth information. The goal is to minimize the joint cost function:

$$E_{tracking} = E_{icp} + w_{rgb} E_{rgb}. \tag{2.14}$$

E_{icp} corresponds to the point-to-plane error between 3D re-projected points from the predicted active surfel model of the last input and the current depth map. E_{rgb} corresponds to the per-pixel photometric error between the current color image and the predicted active model color of the last input. The joint cost function is minimized using the Gauss-Newton non-linear least squares method with a three-level coarse-to-fine pyramid scheme.

2.6.2 Visual LiDAR SLAM

Visual LiDAR SLAM [109], abbreviated as VL-SLAM, is a laser SLAM algorithm that integrates visual information from images. It is composed of a LiDAR odometry estimator, a pose graph optimization back-end, and a set of loop detection and closure modules based on visual recognition. Its architecture is represented in the block scheme of Figure 2.26.

LOAM [30] is chosen as the laser odometry estimator, mainly due to its real-time capabilities. Even though it has a tendency to drift in challenging environments, it still exhibits a level of accuracy and efficiency superior to other algorithms of the same type. To mitigate the drift problem, an online pose optimization back-end has the purpose of building a graph of poses with the output from LOAM. Moreover, the back-end periodically queries a place recognition module to search for loops, which are added as edges to the pose graph. On every iteration, the trajectory of the robot is optimized by non-linear least squares, and the LOAM odometry estimate is corrected.

The pose optimization module of VL-SLAM keeps a collection of N keyframes K. Every keyframe K_i is associated with a timestamp t_i, a pose estimate P_i, a point cloud C_i, a node in the pose graph x_i and an image F_i. Both the pose and the cloud are outputs of LOAM and share the same timestamp. Images might arrive at different moments in time w.r.t. the

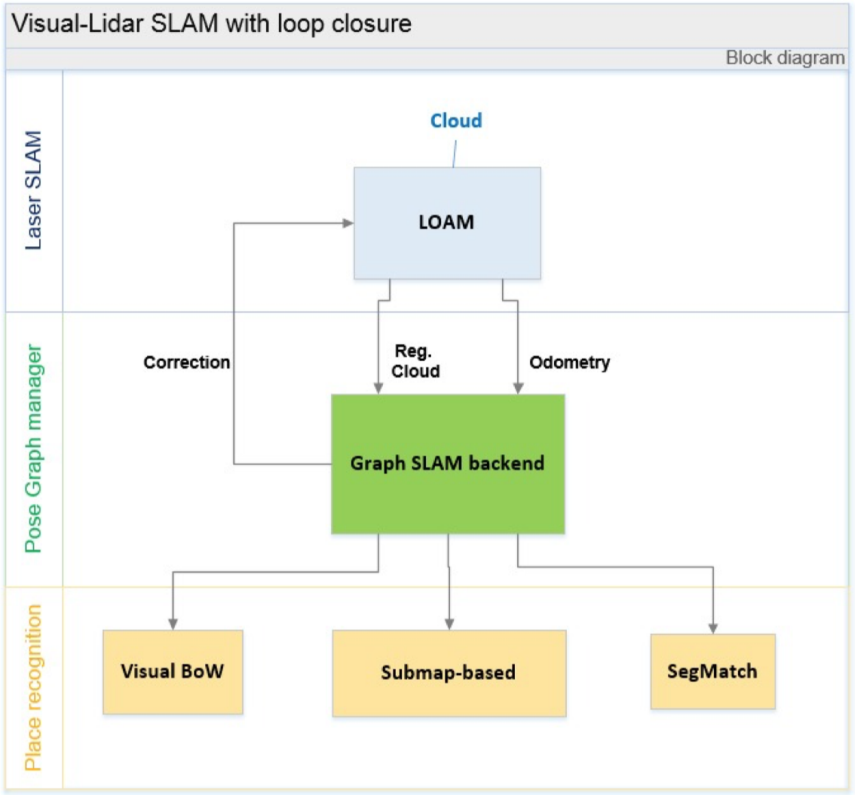


Figure 2.26: *High-level system architecture of Visual LiDAR SLAM [109].*

measured odometry, and for this reason, they are associated with the latest keyframe created, if it is not already associated with an image.

In order to detect loops in the trajectory, VL-SLAM integrates three different approaches for place recognition. One module, which matches point clouds, takes the latest keyframe captured K_i as a reference for place recognition. Then, previously captured keyframes K_0^{i-1} are considered part of the set of candidates S for matching if they satisfy multiple conditions.

The first condition is if keyframe K_j is within a fixed distance D_r from K_i. The second condition is if the distance along the robot path between both keyframes is higher than a threshold D_a. The last check is if they are at a certain distance D_l from the latest keyframe involved in a loop closure.

If a keyframe has an image frame F associated with it, FAST corners [59, 60] are extracted from the image, and ORB [61] descriptors are used to represent them. Then, a visual bag of words representation is constructed.

When searching for loops, the visual bag of words description allows the

system to identify which keyframes are most likely to be similar. Then, a geometric consistency check is performed, by running a nearest neighbor search between the feature points, and assessing if the proportion of matches within a certain distance is less than a percentage (Lowe's ratio test [57]).

Lastly, both the Euclidean and smoothness-based region growing segmenters, belonging to SegMatch [110, 111], are used by VL-SLAM for localization. SegMatch is constantly fed with point clouds from the newest keyframes created by VL-SLAM. It runs in a concurrent thread, and once a loop is detected, it adds a corresponding constraint to the pose graph.

2.6.3 DVL-SLAM

DVL-SLAM [112] is a visual SLAM system that combines the sparse depth measurements of LiDAR sensors with the intensity of camera images and utilizes the direct matching method, as described in Section 2.4.

Only sparse depth measurements are available from the LiDAR, as they are trimmed to be overlapped on the field of view of the camera. Although these sparse measurements can be associated with visual features to perform feature-based visual SLAM, their utility is diminished by the lack of meaningful information in blurred images, needed to compute the corresponding descriptors. For this reason, the direct approach is selected as an effective method for handling sparse depth measurements with the images.

In addition, direct SLAM is known to provide more accurate motion estimation results in low-resolution cameras. However, the narrow field of view of sparse 3D LiDAR remains an issue, as it only allows for an association of depth and image in the partial image region. To overcome this limitation, the depth of neighboring frames is integrated into keyframes, and motion estimation is performed using multiple keyframes.

DVL-SLAM consists of a front-end and a back-end similar to the general SLAM approaches. The input of the algorithm is sequential image data and sequential LiDAR scans. Given an image with an associated sparse depth, only the image is used for the tracking process. Then the module takes a point sampling strategy for fast and robust motion estimation. The front-end focuses on accurate motion estimation using windowed optimization and data association for loop-closing. The associated data received from the front-end is used for global pose graph optimization in the back-end.

The description of each part of the system, represented in Figure 2.27 is as follows. The visual odometry (VO) module performs the tracking process for fast motion estimation without visual features. The initial motion is estimated using only a small number of sampled points, named salient

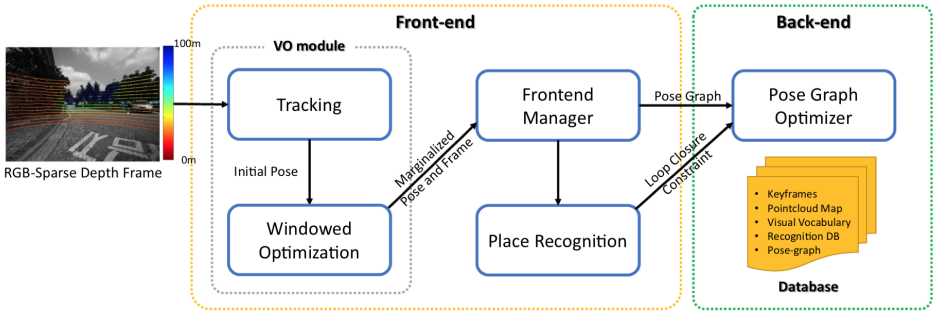


Figure 2.27: *High-level view system architecture of DVL-SLAM [112].*

points, which are directly tracked by the intensity in the image domain.

Tracking occurs at a fast rate, without requiring feature extraction and matching. In addition, using only LiDAR-associated points eliminates the triangulation phase, typical of visual SLAM, to estimate the depth. This simplicity benefits the various SLAM systems with cameras and LiDAR.

Once the tracking process has been successfully performed, the current frame is added to a sliding window that performs local optimization followed by optimization to improve local accuracy within the window. The window consists of N_w keyframes, and each keyframe has its image and point cloud. A point p_k in a keyframe is projected into all keyframes with covisibility.

Photometric residuals are then calculated between the patch at the existing point and the other patch at a projected point. Figure 2.28 shows this procedure. This refinement allows the motion from the tracking process to maintain the low drift. The oldest frames exceeding a user-defined number are marginalized and fed into the back-end pose graph.

The final component of the front-end is the place recognition module, i.e., the loop detector. This module confirms whether the position of a marginalized keyframe is revisited and then integrates depth information from adjacent frames to perform a two-view alignment. Lastly, the back-end module optimizes the global pose graph, including the current motion and the loop constraints provided by the front-end.

2.7 SLAM in dynamic environments

Environments in the real world include static elements, short-term changes, such as moving cars, and long-term changes, similar to those caused by the passing of seasons or the change from day to night. Dynamic environments, i.e., environments that are characterized by the presence of dynamic ele-

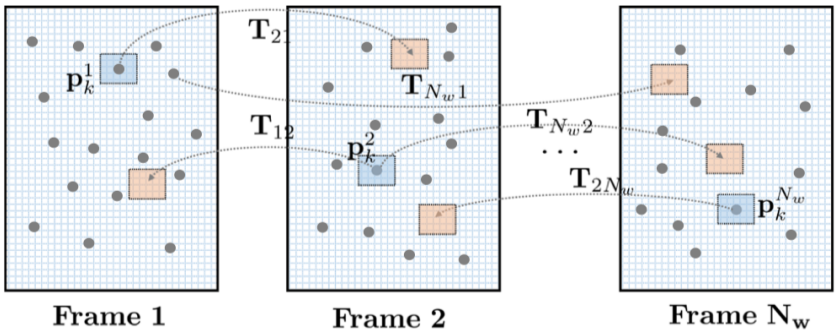


Figure 2.28: *Window-based local optimization used by DVL-SLAM [112]. Gray dots represent the point cloud and rectangles describe images. A point in a keyframe is projected into all keyframes with covisibility. Photometric residuals are then calculated between the patch at the existing point and the other patch at a projected point.*

ments, pose multiple challenges to the different modules of SLAM, including data association, pose estimation, loop detection, and map construction.

The standard approach adopted to solve SLAM in dynamic environments is to identify, track and elaborate (or even remove) the dynamic elements of the scene. Removing moving objects in the environment can benefit the pose estimation process, as it minimizes the error of both short and long-term data association. There are many works in the literature on dynamic object detection and tracking, based on prior information combined with visual or laser features, often including the usage of deep learning, and they can be classified into geometric-based and segmentation-based approaches.

2.7.1 Geometric-based approaches

Geometric-based methods for dynamic SLAM rely on the properties of data and sensors to directly identify moving elements in the scene. When considering laser scans, algorithms can be further divided into visibility-based and voxel-based. The visibility-based methods identify dynamic points in the scan by checking whether the laser elements (i.e., 3D points) are occluded on the corresponding optical path.

Xiao et al. [113] proposed a local cylindrical reference frame for interpolating occupancy between rays, to solve irregular point densities and occlusions. The occupancy of the reference point cloud is fused at the location of target points and then the consistency is evaluated directly on those. Lastly, a point-to-triangle (PTT) distance-based method is combined to improve the occupancy-based algorithm proposed in the work.

Qian et al. [114] presented RF-LIO, which builds on LIO-SAM [115]. The method adds adaptive multi-resolution range images and uses tightly-coupled LiDAR inertial odometry to first remove moving objects, and then match LiDAR scan to the currently active submap. Thus, it can obtain accurate poses even in highly dynamic environments. However, RF-LIO is sensitive to pose accuracy and image resolution. Considering the angle error, ranging error, and light spot effect of LiDAR frames, false deletion of scan points may also become a serious issue, influencing the overall accuracy.

Kim and Kim [116] proposed some tricks to solve the occlusion problem. They developed Removert, a multi-resolution range image-based false prediction reverting algorithm. First, definite static points are conservatively retained. Then, more uncertain static points are iteratively recovered by enlarging the scan-to-existing map association window size, which implicitly compensates for the LiDAR motion or registration errors. However, some large moving objects may block the LiDAR beams and the static points behind the dynamic points cannot be observed, which makes it difficult to remove these types of dynamic points even with this method.

Voxel-based methods, instead, maintain the map in a probabilistic way and they require preserving a large voxel map, which can consume a lot of memory and computing resources. For example, Dewan et al. [117] used object detection and a new voxel traversal method to speed up the process of building an occupancy map, enabling real-time dynamic object removal. However, the premise of using these methods is to have accurate localization, which cannot be achieved before removing dynamic objects.

Aside from LiDAR-based methods for geometric object detection, RGB-D data has been widely used in the field, thanks to the color and depth information embedded in each datum. Li et al. [118] proposed a real-time depth edge-based RGB-D SLAM system for dynamic environments based on frame-to-keyframe registration. To reduce the influence of dynamic objects, they adopted a static weighting method for edge points in keyframes. Static weight indicates the likelihood of one point being part of the static environment. This static weight is lastly added to the intensity-assisted iterative closest point method to perform the registration task.

Sun et al. [119] classify pixels using the segmentation of the quantized depth image and calculate the difference in intensity between consecutive RGB images. Tan et al. [120] proposed a novel online keyframe representation and updating method to adaptively model dynamic environments. The camera pose can reliably be estimated even in challenging situations using a prior-based adaptive RANSAC algorithm to efficiently remove outliers.

Although the geometric-based solution in dynamic environments can

restrict the effect of moving objects to some extent, there are some limitations, as these methods cannot detect the potential dynamic objects that temporarily remain static (e.g., chairs in a room or parked vehicles) and they lack the ability to detect true object-level semantic information.

2.7.2 Segmentation-based approaches

State-of-the-art methods for dynamic object detection rely on deep learning methods, which perform both object detection and semantic segmentation, information that is missing in geometric-based systems. More works have been developed in indoor environments than in outdoor scenarios, with RGB-D sensors used as the dominant sensor. Based on the semantic segmentation of RGB images and the combination of depth images, RGB-D sensors can achieve more accurate detection and tracking of dynamic objects in SLAM.

DS-SLAM [121], implemented on ORB-SLAM2 [53], combines a semantic segmentation network (SegNet [122]) with a moving consistency check to reduce the impact of dynamic objects and produce a dense semantic octree map [123]. DS-SLAM assumes that the feature points on the people, considered dynamic objects in the scene, are most likely to be outliers. If, however, a person is determined to be static, then matching points on the person can also be used to predict the pose of the camera.

DynaSLAM [124] combines Mask R-CNN and ORB-SLAM2 to achieve visual SLAM in dynamic environments; however, this method eliminates all moving objects (such as cars parked on the roadside), which may lead to errors in data association. The method can detect moving objects either by multi-view geometry, deep learning, or both, and inpaint the frame background that has been occluded by dynamic objects using a static map of the scene. It uses Mask R-CNN to segment out all the priori dynamic objects, such as people or vehicles. DynaSLAM II [125] tightly integrates the multi-object tracking capability. But this method only works for rigid objects (differently from people, as they may assume different poses).

Dynamic-SLAM [126] proposed a missed detection compensation algorithm and selective tracking method to improve pose estimation.

SuMa++ [127] uses semantic segmentation results as constraints to improve the ICP algorithm to achieve LiDAR-based SLAM in dynamic environments; the semantic information of the image is used to assist pose correction to achieve point cloud registration [128], and a feature map is constructed by extracting simple semantic features from point clouds.

Detect-SLAM [129], another system derived from ORB-SLAM2, integrates visual SLAM with a single-shot multi-box detector (SSD) [130].

The probability of a feature point belonging to a moving object is called moving probability. A visual keypoint can be divided into four states: high-confidence static, low-confidence static, low-confidence dynamic, and high-confidence dynamic. Considering the delay of detection and the spatiotemporal consistency of successive frames, only the RGB values of the images associated with keyframes are employed for detection. Once the result is obtained, a keyframe is inserted into the local map, updating the moving probability of its 3D points that matched with the keyframe.

SOF-SLAM [76], previously described in Section 2.4, is yet another visual semantic SLAM built on only the RGB-D mode of ORB-SLAM2. In the paper, a new dynamic features detection approach is proposed, which can fully take advantage of the dynamic characteristics of features. The pixel-wise semantic segmentation results generated by SegNet [122] serve as a mask in the developed semantic optical flow to get a reliable fundamental matrix, which is then used to filter out the truly dynamic features. Only the remaining static features are lastly reserved in the tracking and optimization modules, to achieve accurate camera pose estimation.

DM-SLAM [131] combines Mask R-CNN, optical flow, and epipolar constraint to judge outliers. The Ego-motion Estimation module estimates the initial pose of the camera. Fan et al. [132] proposed a novel semantic SLAM system with a more accurate point cloud map generation in dynamic environments, relying on BlizNet [133] to obtain the masks and bounding boxes of the dynamic objects in the image.

In general, the majority of segmentation-based methods wait for the semantic results of every frame, or keyframe, before estimating the camera pose. To solve this issue, Liu and Miura proposed RDS-SLAM [134], a real-time visual dynamic SLAM algorithm that is built on ORB-SLAM3 [75] and adds a semantic thread and a semantic-based optimization thread for robust tracking and mapping. These novel threads run in parallel with the others, and therefore the tracking thread does not need to wait for the semantic information anymore. Semantic information is updated and propagated using the moving probability, which is saved in the map and used to remove outliers from tracking using a data association algorithm.

CHAPTER 3

ART-SLAM: Accurate Real-Time 3D LiDAR SLAM and Localization

In Section 2.3 we described the main approaches presented in literature over the last decades. Overall, existing methods can be distinguished in feature-based and full point cloud-based algorithms, depending on the way motion tracking is performed. Feature-based SLAM systems are fast, but they often lack accuracy over long trajectories. On the other hand, full point cloud-based methods show the opposite properties, being extremely accurate, but not usable for online SLAM (i.e., they do not work in real-time).

Moreover, the majority of works only deal with simultaneous localization and mapping, not considering the case in which a map is already available, which is known as just localization. With the goal of improving the state-of-the-art, we developed an accurate, real-time laser system, ART-SLAM [1], and extended it with modules able to perform also only localization.

In Section 3.1 we describe ART-SLAM, showing also the improvements w.r.t. what exists in literature, to which it is compared. Follows Section 3.2, in which the localization module of ART-SLAM, named ART-SLAM LOC, is briefly explained, together with a thorough comparison with state-of-the-art methods that perform both SLAM and localization.

3.1 ART-SLAM

Trajectory estimation and map building represent core aspects of many applications in robotics, such as autonomous driving. A great amount of simultaneous localization and mapping (SLAM) systems with 6 degrees-of-freedom (6 DoF) have been proposed in the literature in the last decades, some of them described in Section 2.2, with the goal of estimating accurate trajectories in real-time. These methods can be grouped into vision-based and point cloud-based systems, depending on the main sensor used, with the latter preferable because of their overall greater accuracy and usability in different environments (e.g., changing illumination or weather).

All point cloud-based algorithms available in literature either achieve high accuracy at the cost of computational time or sacrifice the quality of the trajectory to obtain real-time performance. Furthermore, these systems are monolithic and difficult to modify and adapt, and they are usually bound to some existing framework (e.g., ROS [135]), often hindering portability on different operating systems and integrability with other software. Nevertheless, systems which fully exploit LiDAR scans are preferable, as the maps (or submaps) created can be used to efficiently obtain accurate 3D reconstructions of the environment, as described in [136], useful for many applications, such as autonomous driving, virtual and augmented reality.

For these reasons, the first step of our Ph.D. program consisted in the development of a new system, *ART-SLAM* [1], to perform point cloud-based graph SLAM, inspired by HDL [28], with multiple contributions. ART-SLAM achieves real-time performance, retaining high accuracy, even in scenarios without loops and considering dense point clouds as input. The proposed system is also able to efficiently detect and close loops in the trajectory, using a three-phased algorithm. ART-SLAM presents a high degree of modularity, due to its architecture, and can be easily integrated and improved. Moreover, ART-SLAM is a zero-copy software, as unnecessary data copies are avoided, e.g. the point clouds passed between modules, keyframes, or other types of sensor data. For these reasons, ART-SLAM has been used as the base system for all other methods developed in the thesis.

We first discuss related works in Section 3.1.1, to give a brief review of laser SLAM methods in literature. In Section 3.1.2 we give a high-level description of the architecture of ART-SLAM, followed by a detailed discussion about all of its modules. The pre-filterer is described in Section 3.1.3, while the motion estimation procedure is in Section 3.1.4, aided with the pre-tracker module of Section 3.1.5. Then, details about floor and loop detection are given in Section 3.1.6 and Section 3.1.7, respectively, with

information about pose graph building and optimization in Section 3.1.8. Lastly, in Section 3.1.9 and Section 3.1.10 we evaluate the system.

3.1.1 Related works

Point cloud-based algorithms can capture and represent the environment with a high level of detail, due to the density of the clouds, and they are not afflicted by the issues of vision-based methods, such as illumination and viewpoint changes. Moreover, tracking performed with point clouds is more accurate and stable than its visual counterpart, and it is generally preferred when LiDAR data is available. However, achieving real-time performance while keeping high accuracy, remains an open quest in point cloud-based systems, as previously discussed in Section 2.3.

The most critical aspect that hinders real-time point-cloud SLAM is the alignment of LiDAR scans. During the last decades, many algorithms have been created to find the relative motion between two point clouds, an operation also known as scan matching. The most widely used and known methods to perform scan matching are Iterative Closest Point (ICP) [13] and its many variants. The idea behind these algorithms is to align two point clouds iteratively until convergence or a stopping criterion is satisfied. Although ICP suffers from high computational cost, Generalized ICP [18] and more recent parallel versions (e.g., VGICP [137]) are faster, more accurate, and can be used as better alternatives than the original algorithm.

To overcome the computational shortcomings of full point cloud scan matching, feature-based approaches have been proposed. These methods work similarly to standard scan matching but require fewer resources. They achieve this by extracting 3D features from the clouds, such as edges or planes, and then matching them. A low-drift and real-time LiDAR odometry and mapping (LOAM) method is proposed in [30]. LOAM performs 3D point feature-based scan matching to find correspondences between point clouds. The performance of LOAM deteriorates when resources are limited and no loop detection is performed, leading to large estimation errors.

LeGO-LOAM [32] has successively been proposed, being a lightweight real-time pose estimation and mapping system, composed of five modules: segmentation, feature extraction, LiDAR odometry, LiDAR mapping, and transform integration. Speedup is achieved by filtering the input clouds through image-based segmentation performed on the 2D range projection of each scan. More recent variants of LOAM and LeGO-LOAM have been proposed, optimizing them, namely A-LOAM and LeGO-LOAM-BOR. Another improved system, w.r.t. LOAM, is LIO-SAM [115], which couples

mandatory IMU data and the registration method of LOAM, achieving better performance than the other systems, with the same precision.

Feature-based systems are, in general, less accurate than methods that perform scan matching on whole clouds. For this reason, loop closure and trajectory optimization are mandatory steps in their pipeline. These tasks can easily become computationally demanding as the size of a trajectory increases. To overcome this problem, graph SLAM systems have been proposed, such as [23] and [138], where the trajectory of the robot, estimated via scan matching, is modeled as a graph. There are multiple advantages of this approach, as described by Grisetti et al. in [139], such as the ability to introduce relationships between sensor data and/or observations from the environment, or the great availability of frameworks for efficient graph optimization, which results in the correction of the robot trajectory.

A recent point cloud-based system relying on a graph is HDL [28], which consists of four steps. First, laser scans are pre-processed and filtered to reduce their size. Then, the filtered clouds are used to simultaneously perform tracking and possibly detect the ground plane. Poses estimated through tracking and floor coefficients extracted from the point clouds are used to build a graph of the trajectory, i.e., a pose graph, which is later optimized. The system achieves superior accuracy, but it is slow, especially when dealing with large point clouds (e.g., more than 100K points).

3.1.2 System architecture and overview

An overview of the proposed framework is presented in Figure 3.1. The system is composed of multiple distinct modules that can be grouped into two main blocks. The first, mandatory (colored in gray), is the core of ART-SLAM [1], and it is formed by all the modules that perform SLAM on the input point clouds (orange, in the figure). The other blocks of the proposed framework are optional, as they can be used to integrate the main system with data coming from different sensors or with pre-processed input.

Given an incoming laser scan, the first step is to process it, in the *prefilterer*, to reduce its size and remove noisy points. The filtered cloud is then sent simultaneously to two modules. The most important one, the *tracker*, estimates the current displacement of the robot by performing an advanced version of scan-to-scan matching with previous filtered scans (keyframes). The other module, named *floor detector*, finds the robot pose w.r.t. the ground, adding height and rotational consistency to the estimated trajectory.

The current pose estimate is sent, along with its corresponding point cloud, to the *loop detector* module, which tries to find loops between new

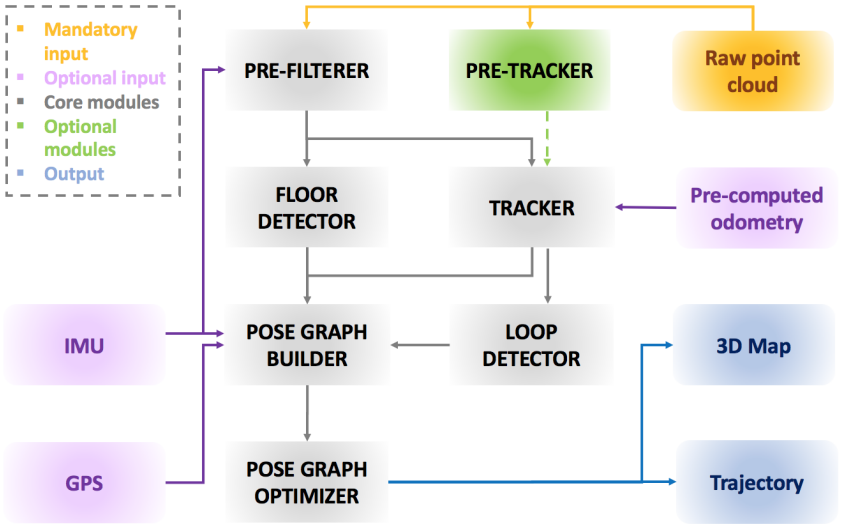


Figure 3.1: *Architecture of the proposed system, ART-SLAM. Core modules are the mandatory blocks of the pipeline, used to estimate an accurate trajectory and to build a 3D map of the environment, given a sequence of input point clouds.*

and previous point clouds, again performing scan-to-scan matching. In addition, poses, loops, and floor coefficients (estimated by the floor detector module) are used to build the pose graph, which represents the trajectory of the robot. Lastly, the pose graph is optimized to satisfy these constraints.

IMU and GPS data (pink in Figure 3.1) can be integrated into the *pose graph builder* module, to increase the accuracy of the estimated trajectory. IMU data can also be used in the pre-filterer to de-skew point clouds, which are usually skewed due to the motion of the robot. Moreover, *pre-computed odometry* (e.g., through a different sensor or system) can be fed to the tracker as an initial guess for the scan matching. The *pre-tracker* module (green in Figure 3.1) performs multi-level scan-to-scan matching, to obtain a rough estimate of the motion of the robot, before the tracking step: this estimate is sent to the tracker, to boost the performance of scan-to-scan matching.

Differently from the majority of systems available in the literature, our developed method for LiDAR SLAM is fully modular and its components work independently one from another. This is possible thanks to a *register and dispatch* technique used to create the system, leading to the creation of modules formed by the following elements. Observers allow a module to capture data as soon as it is available, independently of the type, which is then put into one or multiple dispatch queues, i.e., FIFO structures with the

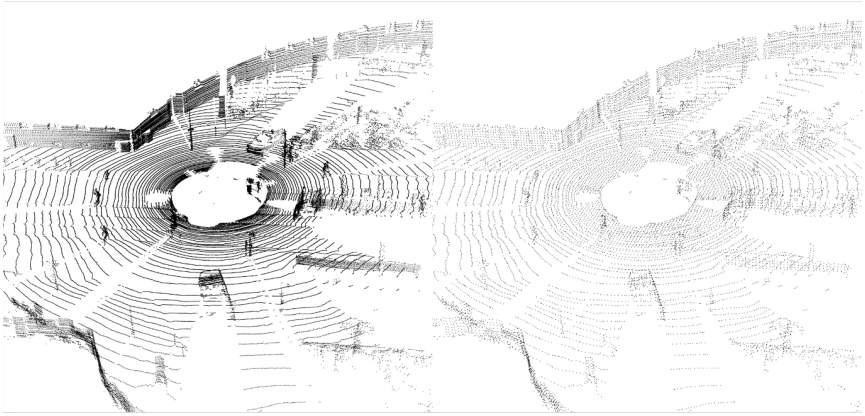


Figure 3.2: *Input LiDAR scan (left) and the corresponding processed output of the pre-filterer module (right). The difference in number and density of elements is noticeable.*

purpose of avoiding the loss of incoming data. The core of a module is its characteristic: it elaborates one datum per queue at a time, extracting it from the relative dispatch queue. As soon as the core finishes its task, it gives the byproducts of the module to the notifier, which broadcasts them to the modules in need (the same object, so that no copy is necessary).

There are three main advantages of using this architecture. First, input data, for each module, is safely stored for later usage, independently from its processing rate, meaning that it cannot be lost. Then, new modules can be easily integrated into the system, being only dependent on the type of data needed. Lastly, the same core task can be performed in parallel, on multiple threads, if it does not require temporal coherence (e.g., floor detection).

3.1.3 Pre-filterer

The pre-filterer module has the purpose of reducing the size of the input point cloud and removing noise and outliers, as can be seen in Figure 3.2. Data reduction, or downsampling, is essential because, as stated in Section 2.3 and Section 3.1.1, scan matching on full point clouds can become computationally demanding if the size of the cloud is large enough. Downsampling can reduce point clouds by a factor of five, or more, depending on the context, while retaining the spatial structure of the initial scan.

The clouds are then filtered to remove outliers and noise points. This operation is usually more costly w.r.t. the downsampling task. To improve performance w.r.t. HDL [28], we split the cloud into four pairs of octants and perform filtering on each separately, in parallel, obtaining a noticeable speedup (the amount depends on the hardware, but it is greater or equal to a

factor of 1). After that, all the smaller clouds are combined to form a larger, filtered point cloud, ready to be broadcasted to other modules.

3.1.4 Tracker

Pose tracking, which comprises short term data association followed by motion computation, establishes the roto-translation between consecutive poses. The tracker adopts a keyframe-based approach to estimate the trajectory of the robot, performing scan-to-scan matching using state-of-the-art algorithms (e.g., ICP [13], GICP [18], VGICP [137] and NDT [40]), selected depending on the user choice and the type of environment the robot is navigating (e.g., outdoor or indoor). For example, NDT has several advantages over ICP, such as surface representation capability, accuracy, and data storage. However, the performance of the NDT is directly related to the size of its cells, and there is no proven way of choosing an optimum cell size, making it less suitable in large outdoor scenes, such as urban scenarios.

Keyframes are data structures describing the motion of the robot in selected locations of its trajectory. They are represented by multiple variables, used to collect data associated with the various poses. In ART-SLAM, each keyframe contains a point cloud and the pose (odometry) estimated by the tracker, data which is also used for loop closure detection, pose graph construction, and map creation. Other useful information contained in a keyframe are the timestamp associated with the point cloud, the estimated accumulated distance from the beginning of the trajectory, and, if available, acceleration, orientation, and position coming from other sensors.

To reduce the computational resources needed to efficiently perform SLAM, not all the filtered point clouds in input to the tracker become keyframes. Except for the first keyframe, which corresponds to the first LiDAR scan received by the system, an input cloud must satisfy at least one of the following criteria to be considered a new keyframe.

- Be distant from the estimated location of the previous keyframe of a user-defined translation $\Delta trans$, in meters.
- Be rotated from the estimated location of the previous keyframe of a user-defined angle $\Delta orientation$, in radians.
- Have a time difference of a user-defined interval ΔT, in seconds.

The thresholds $\Delta trans$, $\Delta orientation$ and ΔT depend on the dataset considered (e.g., length, type, or complexity) and the type of trajectory to be estimated, and should be tuned accordingly to obtain a reasonable number

of keyframes, as too few would decrease the accuracy of the SLAM system, and too many would degrade its performance. In indoor scenes, for example, $\Delta trans$ could be set to 0.2 meters, while in large scale urban environments $\Delta trans > 5$ meters, where longer paths are traversed.

Given the cloud corresponding to the current keyframe K_n and the available new filtered point cloud in input c_t, scan-to-keyframe matching is performed between them, to find their relative motion. The algorithm requires an initial guess, to boost performance and accuracy, which can be chosen in two ways: either it is available through other means (e.g., odometry from another sensor), or a constant velocity motion model is assumed, and the previous transformation is used (the one computed between the point cloud of the current keyframe K_n and the previous filtered cloud c_{t-1}).

Usually, algorithms for point cloud-based tracking find the relative motion between consecutive LiDAR scans, c_{t-1} and c_t, and then compose this transformation with the previous ones, to estimate the current odometry. This method may seem more accurate, but it accumulates errors the more distant the clouds are from the current keyframe. In ART-SLAM, instead, the motion of the robot is always referred to the keyframe closest in time, and the previous motion is taken into consideration only to estimate the guess for scan matching if we assume a constant velocity motion model.

This approach, which is unique to ART-SLAM, also allows the system to skip the whole scan matching procedure if pre-computed odometry is available. The latter, even if not completely accurate, allows one to immediately check if the current point cloud is a candidate for the selection of a new keyframe. If it is not, the tracker does not perform scan matching, and the relative transformation between the current keyframe and the pre-computed odometry is used as the guess in the next tracking iteration.

Once the tracker has detected a point cloud that satisfies any of the keyframe creation criteria described previously, a new keyframe is built and it is broadcasted to the loop detection and pose graph builder modules, moving from the front-end part of ART-SLAM to its back-end.

3.1.5 Pre-tracker

Aligning two full scale point clouds would result in the best transformation estimate, as all the 3D points are accounted for. However, this approach is often unsuitable for real-time applications, especially on low-end devices. Moreover, not all elements of a LiDAR scan carry useful information, like flat ground surfaces, which are indistinguishable one from the other. For this reason, scan matching should be aided with a rough initial guess. The

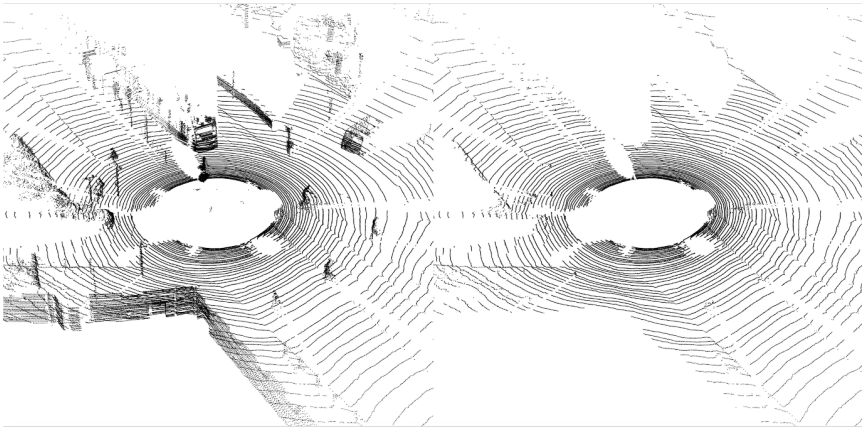


Figure 3.3: *Input LiDAR scan (left) and the corresponding floor, detected and extracted from the point cloud using the floor detection module (right).*

computation of the latter is the purpose of our pre-tracker module, which performs multi-scale alignment while working in parallel to the pre-filterer.

First, the same point cloud given in input to the pre-filterer is fed to this module, where it is heavily downsampled. The reduced cloud is then used to perform scan-to-scan matching with a previously downsampled cloud (i.e., the preceding one). This alignment is fast, due to the reduced size of the inputs, even if not as accurate as if it was done with non-downsampled clouds. This procedure can be repeated using the same point clouds but downsampled at a different scale, lower than the one used in the first phase. The relative motion resulting from these steps is broadcasted to the tracker, to be used as the initial guess in the current scan matching, allowing it to possibly skip the frame and reduce the computational resources needed.

3.1.6 Floor detection

To enforce height and orientation consistency in the trajectory, filtered point clouds are processed to find the ground plane in them, as visible in Figure 3.3. This can be modeled as a four-dimensional vector $GP(a, b, c, d)$ representing the plane equation $a * x + b * y + c * z + d = 0$.

Floor detection should handle multiple scenarios, such as planar or similar to planar motion (e.g., urban roads), rough terrains (e.g., rocky paths), and environments with ascents and descents. While HDL [28] deals only with planar motion, in ART-SLAM all scenarios are considered.

In the first case, i.e., planar or planar-like motion, the floor detector module takes a point cloud and manipulates it in the following way. As

the ground can be found within a small region of the input scan, the first step performed is clipping the cloud within an acceptable range of search. This step greatly reduces the size of the cloud, increasing performance when searching the floor. Then, the clipped output is filtered to eliminate points whose normal is highly non-vertical. This is done to avoid mistakes due to planar-like surfaces in the environment, such as walls or windows. Lastly, Random Sample Consensus (RANSAC) for plane detection is done on the filtered laser scan, to detect and estimate the ground plane coefficients.

When dealing with rough terrains, a floor cannot be found in the previous way, as no planar structures can be detected with RANSAC. The input scan is further clipped, this time horizontally: only the 3D points within a threshold distance from the center of the cloud are kept. This is done to trim the cloud to be as close as possible to the robot, removing outlier objects such as rocks, logs, or anything that is not planar-like. The few remaining points are then used to perform closed-form plane fitting with the least squares method. If the parameters $\{a, b, c, d\}$ are found, they are broadcasted, together with the timestamp associated with the corresponding point cloud, to the pose graph builder module, to enforce multiple constraints on the estimated trajectory.

The last scenario, i.e., inclined surfaces, is trickier to identify just by using a point cloud, as inclined planes are parallel to the robot body frame and cannot be directly distinguished from non-inclined planes. The process of discovering inclined planes takes place in the pose graph builder module. When a set of floor coefficients $\{a, b, c, d\}$ is associated with a keyframe, the builder checks if there is a noticeable change (user-defined) in a vertical orientation from the considered keyframe w.r.t. the previous one. In this case, it means that there has been a change in slope in the trajectory of the robot, and an inclined ground plane has been successfully detected.

3.1.7 Loop closure

While moving, the robot may return to a place that was previously visited, forming a loop in its trajectory. Finding loops adds motion constraints in the estimated robot poses, correcting drift and estimation errors. The hard part about loop identification and closure is not asserting the presence of a loop, which can be accomplished via simple scan matching, but detecting when loop closure is even a possibility. To do this, we need to decide when and where to look. In ART-SLAM, detection is performed in three consecutive steps, to efficiently search for loops within the collected keyframes.

First, each time a keyframe K_{query} is available, it is compared against all previous existing keyframes. Instead of performing scan-to-scan matching

between the possible pairs $\{K_{query}, K_{candidate}\}$, an odometry-based selection is performed. If K_{query} and $K_{candidate}$ are too close in terms of trajectory, meaning that they have a low accumulated distance, they cannot be considered candidates, as it is unlikely that two keyframes, corresponding to point clouds acquired shortly from each other, would result in a useful loop.

Moreover, the loop detector checks if the position, estimated through tracking, of $K_{candidate}$ is in the neighborhood of the pose corresponding to K_{query}, within a threshold range, which accounts for drift errors induced by the tracker module. If K_{query} and $K_{candidate}$ have sufficient time distance and satisfy these constraints, meaning that they are sufficiently close in space and far in time, they can be considered a loop candidate pair.

Once all candidate pairs have been found, they must be further thinned down to avoid unnecessary computation. The approach proposed in [140] converts point clouds into 2D polar grids and efficiently compares them using a KD tree to select the most similar one to a given point cloud query. The second phase for efficient loop detection in ART-SLAM slightly modifies this method, by comparing the 2D polar grid of the point cloud associated with the query keyframe with the 2D polar grids corresponding to the candidate keyframes. It should be noticed that the 2D polar grids are created once, whenever a new keyframe is generated, and not each time loop detection takes place. At the end of this procedure, only k candidate pairs for loop detection and closure remain, ready to be compared in the last step.

The few number of candidates allows for scan-to-scan matching on the elements of each pair (the point clouds corresponding to the new keyframe and candidate keyframe), to obtain a set of relative motions. All transformations are then compared to find the best one, i.e., the one with the highest accuracy, and corresponding to the smallest Euclidean distance between all the pairs K_{query} and $K_{candidate}$. If a best match is found, it means that a new loop has been efficiently detected, and it is added to the pose graph as a new constraint, to be closed during the optimization procedure.

Differently from HDL [28], where only the first and last steps are performed, in ART-SLAM, the addition of the Scan Context method allows for scalable and efficient loop detection. Indeed, as the length of the trajectory to be estimated increases, the number of pairs to be checked for loop detection and closure also grows in size, as more and more keyframes are added.

The first two steps are fast operations, with the former consisting mainly of matrix multiplication and the latter being proved to be scalable [140]. Moreover, the 2D polar grids are pre-computed when inserting the keyframes in the pose graph, further decreasing the time needed by the module. At the end of the second phase, there will always be at most k candidate pairs,

independently from the number of keyframes to check, making this three-phased approach suitable for efficient loop detection and closure.

3.1.8 Pose graph building and optimization

As mentioned in the description of the system, our framework is a form of graph SLAM [139]. In graph SLAM, the poses of the robot are modeled as nodes in a graph, named *pose graph*, and labeled with their pose in the environment. The nodes are connected with edges representing spatial constraints between poses, resulting from sensor measurements (e.g., IMU or GPS) or scene elements, as the floor coefficients, in our case.

Each node in the pose graph represents a robot pose, and at least one measurement (the point cloud is mandatory) that is acquired at that location; moreover, each node is associated with the corresponding keyframe. An edge between two nodes consists of a sample from a probability distribution over the relative transformation of the robot poses corresponding to the nodes. These transformations are either odometry measurements between consecutive poses or are determined by aligning the sensor measurements acquired between two keyframes, via tracking. Due to the noise that corrupts the sensors and the drift in the robot odometry, the associated edges represent soft constraints and are not fixed, to be later optimized.

It is also possible to insert absolute constraints that cannot be modified. Examples of hard constraints are floor coefficients, GPS, and IMU orientation data, although they can also be set as non-absolute elements, to account for the uncertainty of the sensors or the measurements (e.g., a GPS RTK sensor has centimeter precision accuracy, and the obtained positions can be used as hard constraints). Finally, edges can be added when performing loop detection, between non-consecutive nodes in the graph, forming a ring-like structure. The just described structure of the pose graph is represented in Figure 3.4, including all possible nodes and edges.

The pose graph is given to optimization algorithms to compute the optimal trajectory, which satisfies all sensors and motion constraints, giving high-accuracy estimates, while elaborating a large number of poses. In our implementation, we use the g2o optimization framework [141], as it proves to be fast and accurate over long trajectories. Moreover, g2o allows for the insertion of custom elements in the pose graph, and as such, it is an optimal solution for our modular system, allowing future upgrades.

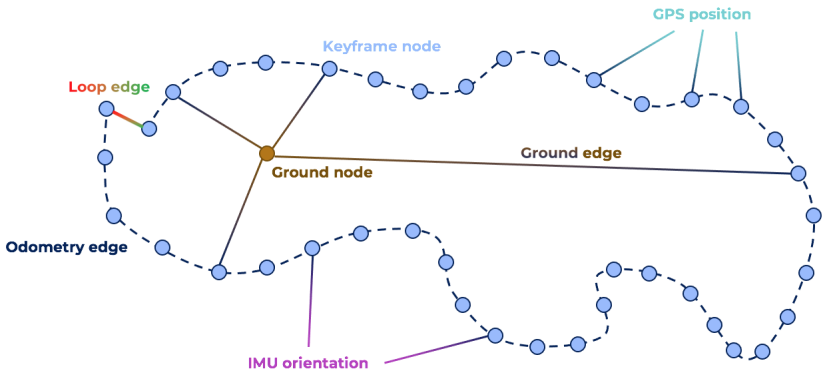


Figure 3.4: *Representation of a possible pose graph in ART-SLAM [1]: the estimated poses and ground coefficients represent nodes, while odometry, loops, IMU, and GPS measurements correspond to the edges of the graph.*

3.1.9 Experimental validation of the system

The proposed system is compared with other methods for point cloud-based SLAM: LOAM [30], LeGO-LOAM [32], A-LOAM, LeGO-LOAM-BOR, LIO-SAM [115] and HDL [28], with A-LOAM and LeGO-LOAM-BOR being two improvements of LOAM and LeGO-LOAM, respectively.

We also include, in the comparison, different variants of our system: ART-SLAM without Scan Context, ART-SLAM with Scan Context, ART-SLAM with IMU (for de-skewing and orientation correction in the pose graph), and ART-SLAM with GPS, which is used only in the back-end.

We evaluate these systems in three scenarios coming from the KITTI dataset [143, 144], corresponding to short, medium, and long sequences, respectively. Lastly, we perform a brief study on the Chilean underground mine dataset [145], to test ART-SLAM in rough environments, typical of long tunnels with very few distinctive features in the surrounding space.

LOAM, LeGO-LOAM, A-LOAM, LeGO-LOAM-BOR, and LIO-SAM do not require particular parameter tuning, although they need a custom implementation of the point cloud projection module, depending on the laser sensor used. Hence, in our tests, we changed such parameters accordingly. On the other hand, HDL and ART-SLAM share the same configuration parameters, e.g., keyframe selection thresholds and pre-filtering method. Table 3.1 shows the most important parameters used in the experiments, for both HDL and ART-SLAM, to allow reproducibility, as described in [142].

It should be noticed that in the KITTI dataset, both IMU and GPS data are acquired at very low frequency, the same as for LiDAR point clouds (about

Table 3.1: *The most important parameters of each module, used in the experimental validation, for reproducibility, as described in [142].*

Module	Parameter name	Value
pre-filterer	Downsample method	VOXELGRID
	Downsample resolution	0.25 [m]
	Outlier removal method	RADIUS
	Radius	0.4 [m]
tracker	ΔTrans keyframes	5.0 [m]
	ΔAngle keyframes	0.25 [rad]
	ΔTime keyframes	1.0 [sec]
loop detector	Loop closure search radius	35.0 [m]
	Loop closure min. distance	25.0 [m]
scan matching	Registration method	FAST_GICP
	Max. iterations	64
	Transformation epsilon	0.01

10 Hz). For this reason, we consider them unreliable, giving low weights in the pose graph, meaning they have only a minor contribution during the graph optimization phase. LIO-SAM, to correctly work, obligatorily requires IMU data at high frequency, and for this reason, to evaluate it, we used the unsynchronized version of the KITTI dataset, having IMU data taken at 100 Hz, giving the system a slight advantage.

For a fair comparison, the systems are tested on a 2021 XMG 64-bit laptop with Intel(R) Core(TM) i7-11800H CPU @ 2.30GHz x 8 cores, with 24576 KB of cache size. Although available, no GPU has been used.

3.1.10 Comparison and results

To evaluate the systems we compute the absolute trajectory error (ATE). This metric measures the difference between points of the true and the estimated trajectory. As a pre-processing step, we associate the estimated poses with ground truth poses using timestamps and point cloud indices.

We also include a visual evaluation of the estimated trajectories and show the reconstructed 3D map in the long sequences (KITTI 00 and Chilean). Finally, we also discuss the processing time of the modules of ART-SLAM, to prove its real-time performance even in complex scenarios.

Figure 3.5 shows the estimated trajectories of Sequence 07 of the KITTI odometry dataset [143]. All the methods considered for comparison, except for LOAM, accurately follow the ground truth, correctly finding the only loop and optimizing the poses. LOAM, instead, quickly drifts from the

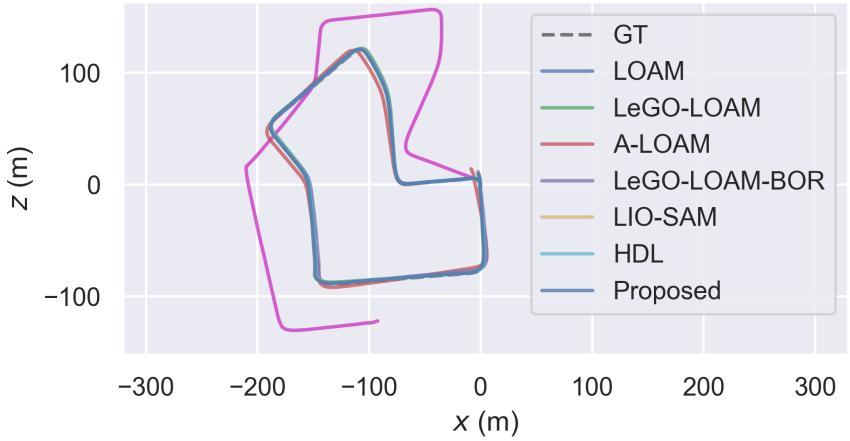


Figure 3.5: *Visual comparison between the trajectories estimated using LOAM [30], LeGO-LOAM [32], A-LOAM, LeGO-LOAM-BOR, LIO-SAM [115], HDL [28] and ART-SLAM [1], without Scan Context, on Sequence 07 of the KITTI odometry dataset [143].*

true trajectory. This behavior is caused by the fact that no loop detection is performed (being LOAM an odometry and mapping method only).

Table 3.2 further details the obtained results, as it represents the mean, root mean square error (RMSE), and standard deviation (STD) of the absolute trajectory error, in meters. The highest accuracy is achieved by ART-SLAM with IMU data. This was expected, as this method combines both the advantages of scan-to-keyframe matching, to track de-skewed clouds, and orientation integration in the pose graph, to correct the motion estimates.

It should be noticed that LIO-SAM comes in second place, proving that, with the same sensors, tracking relying on full point clouds is the best choice for accurate results. Furthermore, the slightly worse results of the other variants of ART-SLAM are also expected. The quality of ART-SLAM with Scan Context depends on the loops identified by the Scan Context method, which are not necessarily the best ones, hence reducing the overall accuracy. ART-SLAM with GPS shows a higher ATE because, as stated before, we consider GPS data as unreliable (high variance in the pose graph) and this negatively influences the optimization process, leading to a trajectory slightly farther from the ground truth and worse than the base method.

After having dealt with a large sequence with the presence of a loop at the end, we also evaluated the systems on a shorter sequence. As short datasets do not have a ground truth, we use, instead, raw GPS data, provided along with the point clouds. Figure 3.6 shows the estimated trajectories of city Sequence 05 of the KITTI raw dataset [144]. As before, all methods

Table 3.2: *Computed ATE on Sequence 07 of the KITTI odometry dataset [143].*

ATE [m]	MEAN	RMSE	STD
LOAM	>10	>10	>10
LeGO-LOAM	1.191	1.309	0.546
A-LOAM	2.467	2.741	1.195
LeGO-LOAM-BOR	1.604	1.807	0.832
LIO-SAM	0.509	0.675	0.351
HDL	0.954	1.253	0.767
ART-SLAM	0.698	0.777	0.341
ART-SLAM (SC)	0.730	0.813	0.358
ART-SLAM (IMU)	**0.343**	**0.366**	**0.127**
ART-SLAM (GPS)	0.782	0.869	0.382

Table 3.3: *Computed ATE on city Sequence 05 of the KITTI raw dataset [144].*

ATE [m]	MEAN	RMSE	STD
LOAM	>5	>5	>5
LeGO-LOAM	0.707	0.768	0.300
A-LOAM	0.938	1.044	0.459
LeGO-LOAM-BOR	1.094	1.169	0.409
LIO-SAM	**0.493**	**0.338**	**0.280**
HDL	0.893	0.912	0.476
ART-SLAM	0.742	0.812	0.331
ART-SLAM (SC)	0.742	0.812	0.331
ART-SLAM (IMU)	0.746	0.814	0.326
ART-SLAM (GPS)	0.343	0.588	0.477

accurately represent the ground truth, with small errors in the trajectory.

It should not come as a surprise that the results are more or less the same, as, for short trajectories, tracking is performed a limited amount of times, and there is not enough distance to accumulate errors.

Table 3.3 shows the ATE statistics, in meters. As before, all the systems but LOAM show good results, accurately following the GPS signal, here used as ground truth due to its relatively high accuracy. This is the reason ART-SLAM with GPS is the most accurate method, even though, for a fair comparison, LIO-SAM was highlighted as the most accurate system.

In Figure 3.7, we show the behavior of ART-SLAM on one of the most complex sequences of the KITTI odometry dataset, i.e., Sequence 00. In the visual comparison, we did not include A-LOAM and LIO-SAM, whose

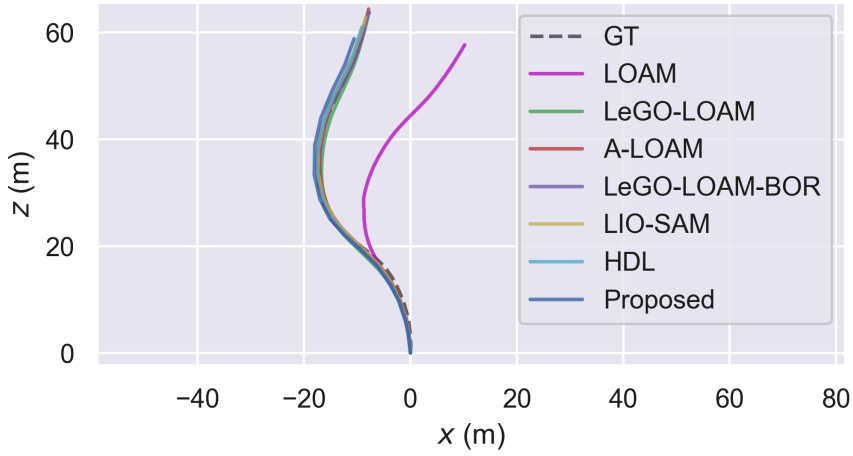


Figure 3.6: *Visual comparison between the trajectories estimated using LOAM [30], LeGO-LOAM [32], A-LOAM, LeGO-LOAM-BOR, LIO-SAM [115], HDL [28] and ART-SLAM [1], on city Sequence 05 of the KITTI raw dataset [144].*

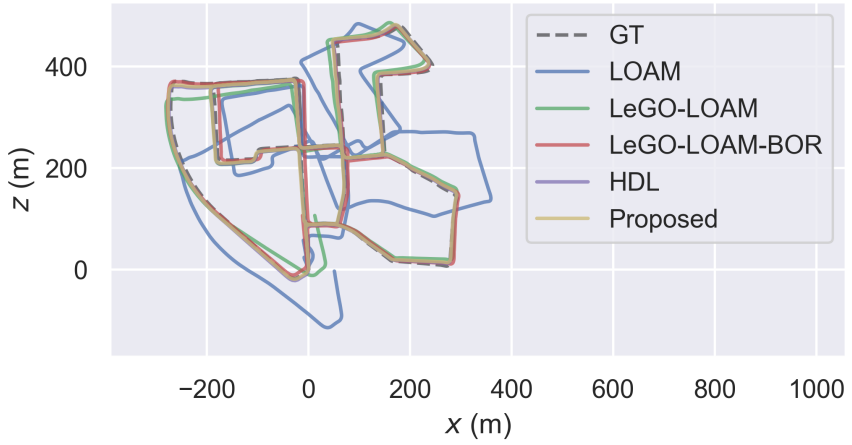


Figure 3.7: *Visual comparison between the trajectories estimated using LOAM [30], LeGO-LOAM [32], LeGO-LOAM-BOR, HDL [28] and ART-SLAM [1], without Scan Context, on sequence 00 of the KITTI odometry dataset [143].*

estimated trajectories are far off the ground truth, and would have cluttered the image. From Table 3.4, one can see the high degree of accuracy achieved by the proposed system, reaching low translation error. Once again, the best accuracy is achieved by ART-SLAM with IMU, followed by ART-SLAM without Scan Context. Figure 3.8 and Figure 3.9 show the 3D map reconstructed by ART-SLAM and a detailed area of it, respectively.

Table 3.5 shows the average processing times, per frame, of the various

Table 3.4: *Computed ATE on Sequence 00 of the KITTI odometry dataset [143].*

ATE [m]	MEAN	RMSE	STD
LOAM	>10	>10	>10
LeGO-LOAM	9.537	11.666	6.718
A-LOAM	>10	>10	>10
LeGO-LOAM-BOR	6.240	6.613	2.188
LIO-SAM	>10	>10	>10
HDL	1.378	1.424	0.779
ART-SLAM	0.981	1.092	0.478
ART-SLAM (SC)	1.232	1.409	0.684
ART-SLAM (IMU)	**0.907**	**1.014**	**0.454**
ART-SLAM (GPS)	1.092	1.156	0.380

Table 3.5: *Comparison of the processing time [ms], per frame between the variants of ART-SLAM [1] (base, with Scan Context, with IMU, and with GPS).*

Time [ms]	Seq.	Filterer	Tracker	Floor det.	Loop det.	Optimization
base	00	18.627	39.462	25.976	19.301	16.330
base + SC		18.627	39.462	25.976	9.380	17.791
base + IMU		21.667	39.462	25.976	13.747	14.486
base + GPS		18.627	39.462	25.976	14.681	12.733
base	07	18.627	39.462	25.976	10.226	1.320
base + SC		18.627	39.462	25.976	7.239	1.598
base + IMU		21.667	39.462	25.976	9.808	1.710
base + GPS		18.627	39.462	25.976	10.407	1.331
base	City	18.627	39.462	25.976	6.819	0.132
base + SC		18.627	39.462	25.976	6.819	0.132
base + IMU		21.667	39.462	25.976	6.720	0.091
base + GPS		18.627	39.462	25.976	9.085	0.165

ART-SLAM variants. Intuitively, independently from the sequence considered, pre-filtering, tracking, and floor detection take the same time (aside from the ART-SLAM with IMU case, where de-skewing is also performed, hence the higher pre-filtering time). It is important to notice, instead, the different times associated with loop detection and graph optimization, which clearly depend on the size and length of the trajectories to estimate.

Moreover, the table clearly shows the importance of Scan Context when performing loop closure: in Sequence 00, the loop detection time, using Scan Context, is half w.r.t. the case without it. Considering that data is acquired at 10 Hz and looking at the system architecture of Figure 3.1, one

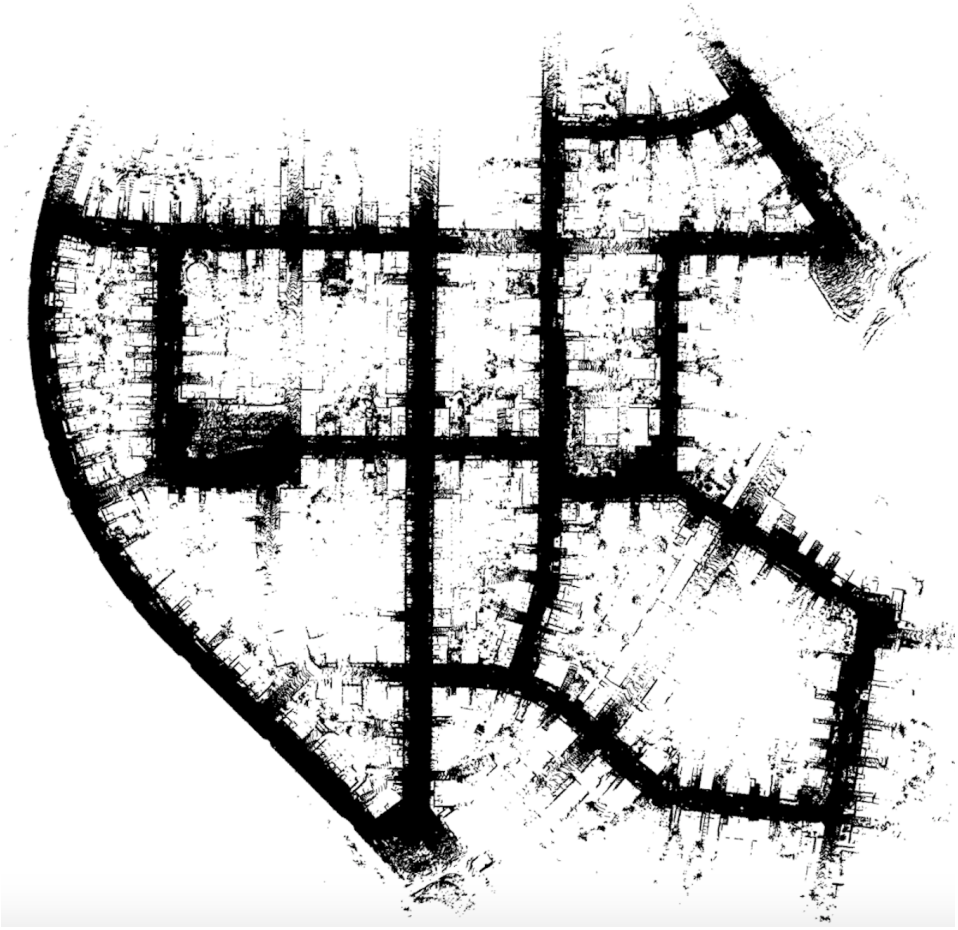


Figure 3.8: *Reconstructed 3D map of Sequence 00 of the KITTI odometry dataset [143], obtained with the developed system, ART-SLAM [1].*

can clearly see that ART-SLAM has real-time performance. All methods, except HDL, are feature-based, and written to run in real-time. Nevertheless, from Table 3.5, one can see that the proposed system, even if full point cloud-based, achieves the same processing performance.

Lastly, we show a visual evaluation of the accuracy achieved by ART-SLAM on the Chilean underground mine dataset [145]. This sequence is relatively long and contains multiple loop closures. The dataset is rather peculiar, as there are only 44 measurements, taken at 30 to 40 meters of distance each and with large rotations, of around 25M points per cloud (while a scan in KITTI has around 130K points). Due to this huge gap between the locations where data was gathered, any scan matching or method to perform

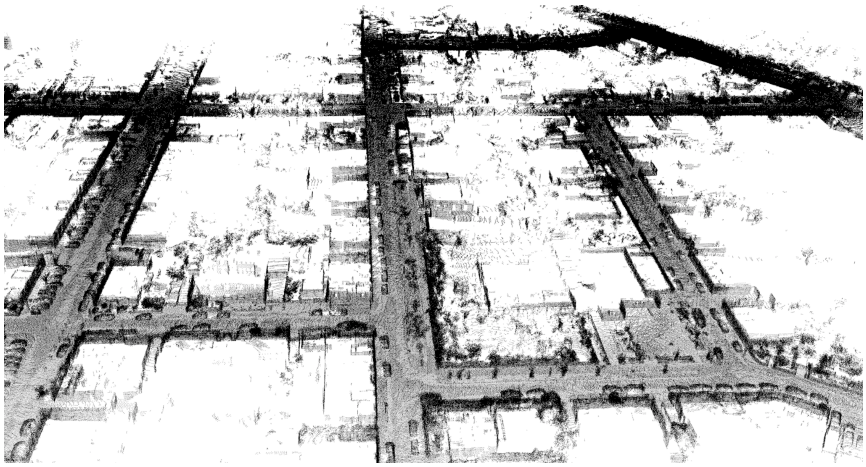


Figure 3.9: *Detailed area of the map corresponding to Sequence 00 of the KITTI odometry dataset [143], built with ART-SLAM [1].*

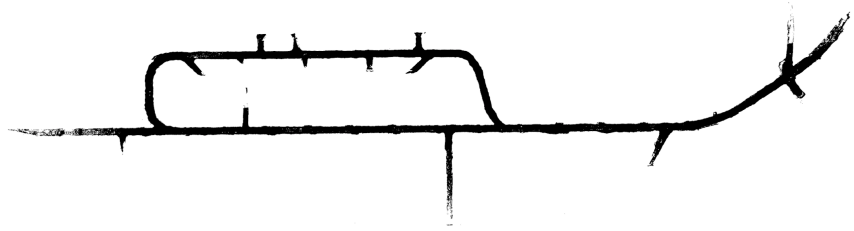


Figure 3.10: *Map corresponding to the Chilean underground mine dataset [145], built with the proposed algorithm, ART-SLAM [1]. The density of the map is related to the high number of 3D points acquired by each LiDAR scan (around 25M points each).*

rigid alignment would fail without proper initialization.

For this reason, we gave to the tracker module the ground truth, corrupted by noise, as an initial guess. The given noise is uniformly distributed in the x and y directions, within the range $\pm 1cm$. We achieved translation errors of about 1 to 5 millimeters, with a processing time much faster than each LiDAR scan period (10 seconds of pre-filtering and 10 seconds of tracking against 152.5 seconds needed to acquire each scan, due to their high number of points). Moreover, for each scan, we correctly identified the ground plane, which is distant around 1 meter from the LiDAR used. The map reconstructed with ART-SLAM can be seen in Figure 3.10.

3.2 ART-SLAM Localization

As we have already seen in the thesis, positioning represents one of the key problems in robotics. Real-world applications, ranging from small indoor spaces to large outdoor scenes, include unmanned vehicle trajectory planning, 3D reconstruction, or search and rescue during emergencies, and they all require an accurate 6 DoF positioning of the robot, which can be classified using the following, well known in the literature, convention.

When the robot moves through an unknown environment, it is referred to as *Simultaneous Localization and Mapping* (SLAM) (as the method developed in Section 3.1); here, the robot must build a map of the surrounding area while simultaneously localizing itself inside it. When the robot goes through a known environment, for example, having available a pre-computed map, only the term *localization* is used, as the robot must only determine its position in the existing map (e.g., obtained through SLAM).

Although outdoor localization is easily achieved using a Global Navigation Satellite System (GNSS) receiver, the range of situations where the sensor can be used is limited, as environments may contain physical obstacles that shade the GNSS signal (e.g., urban canyons). Also, the accuracy attained with the majority of GNSS systems is inadequate for applications where precise localization is crucial. Over the last decade, many algorithms have been proposed to get precise positioning in real-world indoor and GNSS-denied environments. The idea behind the majority of these works is to use multiple sensors to obtain good estimates of the poses of the robot. The combination of IMU and LiDAR sensors is usually adopted to achieve accurate results and to build detailed 3D maps.

Despite the number of systems available for 6 DoF LiDAR positioning being considerable, to the best of our knowledge, there is no work on the attainable positioning precision of these methods in real-world scenarios and with real data [146]. For this reason, as the second step of our Ph.D. program, we extended ART-SLAM [1] with new modules to perform localization in GNSS-denied environments, named ART-SLAM LOC. Then, we focused on a detailed evaluation of existing SLAM and localization algorithms.

The framework of the work is in relative localization accuracy, i.e., w.r.t. a local reference system, which is defined using a map. As explained later in Section 3.2.3 and Section 3.2.4, this map is assumed to be built using a SLAM algorithm, and then localization is performed w.r.t. such map.

The work presented in this part of the thesis consists of the following contributions; first, we provide a short literature review of existing 6 DoF LiDAR localization and SLAM systems, highlighting features, advantages,

and negative aspects. After a brief discussion about which algorithms are more suitable to perform both SLAM and localization, we benchmark those systems with two experimental campaigns, one indoor and one outdoor, evaluating the precision of the results obtained on manually collected data.

Although there are plenty of datasets in literature, the majority is gathered with high-end sensor suites, is pre-processed, and it is usually representing the same type of environments (e.g., cities). To truly stress the accuracy of SLAM and precision of localization, we chose to use data collected by us through a low-end sensor suite and without pre-processing it.

The two collected datasets represent environments that are not commonly seen in the literature. Moreover, we took multiple trajectories of more or less the same paths, to perform localization, which is not typical of standard datasets. The rest of the Section is outlined as follows.

We first discuss related works in Section 3.2.1, to give a brief but detailed insight into existing systems for SLAM and/or localization. Then, in Section 3.2.2 we show the sensors used in the experiments and describe the localization algorithms used, along with the adopted evaluation metrics. Follow Section 3.2.3 and Section 3.2.4, which are dedicated to the results.

3.2.1 Related works

Robots are commonly equipped with IMU sensors to measure linear acceleration and angular velocity. These data can be used to calculate the location of the robot relative to the starting point. However, the accumulated error will make the localization algorithm unreliable after a few meters. GNSS sensors are commonly used to correct the accumulated error of IMU sensors, in outdoor environments. Nevertheless, this is not possible in GNSS-denied scenarios. Thus, the scientific community explored alternative approaches exploiting LiDAR information, which allows to achieve accurate results, as we have already seen in Section 2.3 and Section 3.1.1.

A well-known method for 3D localization and mapping is *LiDAR Odometry and Mapping (LOAM)* [30], previously already discussed. We remind here that this work aims to divide complex tasks that are typically solved simultaneously using SLAM methods. These algorithms work by optimizing a large number of variables at the same time, resulting in low-drift but high-computational complexity algorithms. LOAM is designed to limit drift error while achieving real-time performance. Even if the methods show relatively low error estimations compared to the length of trajectories, the lack of loop detection and closure cause noticeable drifts over time.

LeGO-LOAM [32] is another LiDAR odometry and mapping method

that we have already previously encountered. The system is divided into five modules. The segmentation module produces a 2D representation of the LiDAR clouds. The obtained range images are then segmented via a clustering algorithm, and groups with few points are discarded. Features are successively used to match scans for finding the transformation between them in a module called LiDAR odometry. Finally, the transform integration module fuses the estimation results from LiDAR odometry and mapping.

LeGO-LOAM aims to improve the accuracy of the original LOAM framework. However, LeGO-LOAM is an odometry and mapping method. Thus, there is no easy way to adapt the implemented code just for the localization task. A new map is created at every run of the algorithm, and it is not possible to accurately localize the robot on an already created map.

PoseMAP is a localization method designed for 3D LiDARs. Based on the matching of extracted distinctive 3D features in point clouds, PoseMAP is thought for lifelong localization. It has been tested for 18 months through a mix of human-made structured and off-road unstructured environments without a single failure [147]. Even the changes that occurred in the environment during this long period did not affect the localization accuracy. Indeed, the system can update the map and extend it in case of newly seen environments. However, the authors did not make the code available.

BLAM! is an open-source software package for LiDAR-based real-time 3D localization and mapping. BLAM![1] was developed by Erik Nelson from the Berkeley AI Research Laboratory. However, the author provides neither a scientific paper nor a guide describing how the algorithm works. Thus, the only way to understand the system is to read the source code entirely.

GICP-SLAM [148] is a 3D LiDAR SLAM method thought for indoor and harsh environments, like mines. As the name suggests, it performs scan matching via the GICP algorithm [18]. GICP-SLAM is a graph-based SLAM [139] method; thus, it performs optimization of a graph, known as pose graph, where nodes are the robot poses, and edges are the transformations between the poses. Also, other constraints than transformations can be added to the pose graph. The general framework of GICP-SLAM is the following. The input LiDAR scan is sent to three modules.

One is in charge of plane detection to exploit this as an additional constraint. The second performs LiDAR odometry by matching two successive scans via GICP. The last one is in charge of handling loop constraints. After the robot motion between scans is estimated, another module refines it by scan matching the current scan with the map. All the constraints and the refined transformations are sent to the Pose Graph Optimise module that

[1]The source code can be found at this link: `https://github.com/erik-nelson/blam`

outputs the final pose result, used to build the map. Also in this case, the authors did not make the code available, making GICP-SLAM unusable.

HDL [28], consists of a 3D LiDAR-based SLAM algorithm for long-term operations. The system comprises two main modules, one dedicated to offline mapping, and the other for localization. The mapping is done with a graph-based SLAM algorithm that exploits a scan matching technique, similar to GICP-SLAM. Being a SLAM algorithm, a loop detection and closure procedure has been implemented. Moreover, in order to compensate for the accumulated rotational error, the authors introduced a ground plane constraint, to build consistent maps in long-term scanning processes. The localization algorithm is implemented as an Unscented Kalman Filter (UKF) [149], combining the information from the scan matching of the current scan to the previously constructed map and a prediction step, which uses the angular velocity and linear acceleration from an IMU sensor.

ART-SLAM [1], as described in Section 3.1, is also a 3D LiDAR-based SLAM algorithm, similar to HDL. The mapping follows the same graph SLAM approach, allowing for different context-based scan-matching techniques, while also performing fast and efficient loop detection, making the whole method scalable and suitable for real-time applications.

Being ART-SLAM easy to extend and to work with, we implemented a localization module, named ART-SLAM LOC, to be part of the evaluation presented in this section. The module works the same way as the HDL localization, with some minor performance improvements, such as ground removal from both map and input scans, for efficient pose correction, and up-sampled prediction, to account for biases in the robot motion.

3.2.2 Materials and methods

In Section 3.2.1, we have shown how 6 DoF positioning for indoor and GNSS-denied scenarios can be achieved with precision using LiDAR sensors, integrated with other data sources (e.g., IMU). With the purpose of understanding and evaluating existing algorithms for LiDAR-based 6 DoF localization, we decided to perform two experimental campaigns, collecting data from an indoor space and from a GNSS-denied outdoor environment, to evaluate the attainable precision of these methods in constrained scenarios.

Our intent is also to contribute to the literature, as little information is available regarding this context of navigation and positioning (GNSS-denied, 6 DoF LiDAR-based) in real-world environments. In the following, we first describe the sensors used in our experimental campaigns. Then, we compare the presented systems to choose which of them is suitable to perform both

Table 3.6: *Summary and comparison of the 3D localization and mapping approaches.*

System	Framework	Purpose	Localization	Code
LOAM [30]	ROS	Odometry\mapping	No	Yes
LeGO-LOAM [32]	ROS	SLAM	No	Yes
PoseMAP [147]	None	Localization	Yes	No
BLAM!	ROS	SLAM	No	Yes
GICP-SLAM [148]	None	SLAM	No	Yes
HDL [28]	ROS	SLAM	Yes	Yes
ART-SLAM [1]	None	SLAM	Yes	Yes

SLAM and localization and lastly, we detail the evaluation metrics adopted.

Materials

The sensor suite used in our experiments to perform all tasks consisted of an *Ouster OS-1 sensor* that provided both LiDAR and IMU data. The LiDAR component has a range resolution of $1.2\,\mathrm{cm}$, a vertical resolution of 64 beams, and a horizontal resolution of 1024. The vertical FOV is $33.2°$ and the horizontal FOV is $360°$. The angular sampling accuracy is $\pm 0.01°$, both vertical and horizontal, and the rotation rate is configurable at $10\,\mathrm{Hz}$ or $20\,\mathrm{Hz}$. We configured the sensor to retrieve data at $20\,\mathrm{Hz}$, thus having point clouds of about 65K points. The IMU component gathers higher frequency data ($100\,\mathrm{Hz}$), allowing for a fast localization. A member of our team mounted the sensor on a four-wheeled cart, which was manually pulled by him through selected locations of the experimental campaigns. It should be specified that the person always moved the cart while remaining crouched, remaining outside the field of view of the LiDAR sensor.

Methods

To find the most suitable approaches to test for benchmarking, we analyzed the characteristics of the systems discussed in Section 3.2.1. In Table 3.6, we report a summary of these 3D LiDAR odometry and mapping methods. We indicate the operative framework adopted in each work, the type and purpose of the algorithm, whether a localization algorithm is provided, and whether the code is available. SLAM methods differ from Odometry and Mapping systems because the firsts add loop detection and closure.

For the testing phase, we opted for the approach of Koide et al. [28] and ART-SLAM [1] (including the localization module, ART-SLAM LOC).

Both are SLAM algorithms and, as already said, they can rely on a drift correction procedure using loop detection and closure. This can guarantee long-term mapping operations for large-scale outdoor environments and accurate results in indoor and short scenes, as we have demonstrated in Section 3.1.10. Both are graph-based, meaning that trajectories and maps can be efficiently manipulated and stored. Both have the best performance relative to other state-of-the-art 6 DoF LiDAR SLAM approaches. They also both have mapping and localization modules, allowing us to benchmark the positioning using the maps generated by the algorithms themselves.

The SLAM components of both algorithms present great accuracy, even on long trajectories, w.r.t. other methods. In Section 3.1.10 we compared many of the considered systems, and found out that both HDL and ART-SLAM prove to be accurate in determining the trajectory of a robot, and consequently, in building a high-fidelity map of the environment, for successive localization. When considering datasets collected at high frequency, i.e., the ones used in this section, the accuracy and precision of the algorithms further increase, hence making them suitable systems for benchmarking.

Lastly, the localization method is implemented as an Unscented Kalman Filter [149], in both systems. The Extended Kalman filter solves the nonlinear estimation problem by linearising state and measurement equations. The linearisation yields approximation errors which the filter does not take into account in the prediction and update steps. In comparison, the Unscented Kalman filter picks so-called sigma point samples from the filtering distribution and propagates them through the (nonlinear) state and measurement models. The resulting weighted set of sigma points represents the updated filtering distribution, which, is then approximated as a Gaussian distribution. This results in state estimates which represent the state uncertainty better than the estimates obtained from the EKF with an increased cost.

Particle filters have some similarities with the UKF, in that they transform a set of points via known nonlinear equations and combine the results to estimate the mean and covariance of the state. However, in the particle filter, the points are chosen randomly, whereas in the UKF the points are chosen on the basis of a specific algorithm, i.e., the unscented transformation. Because of this, the number of points used in a particle filter generally needs to be much greater than the number of points in a UKF, usually less than ten.

While IMU data is used in the prediction step, the correction phase is performed by aligning a LiDAR scan with a global map, with scan-to-map registration. This way, errors in the correction step are independent of each other and not related temporally: each scan-to-map alignment is affected by its own error. The overall error thus depends on the distribution of the error

in the scan matching between the current scan and the map, meaning on the global accuracy of the map itself and not the path traversed so far.

To evaluate the selected algorithms, we estimated the localization accuracy on specific poses by manually re-positioning the cart in the same spots. Thus, we named this error “re-localization" error. In our indoor experiments, we recorded a first trajectory by stopping the cart in specifically marked poses. Then, we recorded a second trajectory, reasonably close to the first one, stopping more or less at the same specific positions.

A ground truth acquisition system, i.e., an OptiTrack motion capture system, was used in the indoor experiments to compensate for the difficulty of manually placing the sensor exactly in the same positions. Thus, in the indoor case, the re-localization error of the systems was measured on a set of poses by removing any bias due to incorrect re-positioning. In the outdoor scenario, i.e., an O&G refinery, the ground truth was not available, making it difficult to collect the positioning ground truths, due to the presence of many physical obstacles (e.g., pipes and trusses). Thus, in the outdoor case, we expect the re-localization error to include the error of manually re-positioning the sensor in the same poses, low but not negligible.

To measure the re-localization error, we adopted the metrics used in the most popular benchmarking works like [143, 150–152]. In the following, we provide the mathematical formulation of the problem. Figure 3.11 shows the plots of two indoor example trajectories from our indoor experiment, with the following conventions. In creating both trajectories, we recorded the estimated poses given by HDL and ART-SLAM, and the ground truth poses from the OptiTrack. The experiments were carried out by a member of our team, which manually pulled the cart throughout a laboratory, starting from the OptiTrack area, going outside it, then coming back again.

We decided to go beyond the tracked area to make a longer trajectory, and further stress the SLAM algorithms. We tried to make the paths as similar as possible. The idea was to estimate the accuracy of the localization algorithms when the system is placed in the same location, between different runs of the methods. Thus, the re-localization error tells us how an algorithm is consistent in giving the same pose over time, i.e., how the SLAM components of the evaluated systems are coherent over multiple runs.

For each area A_i, and for each couple of trajectories, we have manually picked a pose $P_{est,ij}$ estimated by the localization algorithm; where $i \in \{1, \ldots, n_{poses}\}$ is the area index and $j \in \{1, 2\}$ is the trajectory index. A pose P is a vector that contains position information along the three directions x, y, z, and orientation as Euler angles, i.e., $roll$, $pitch$, and yaw.

For each area, we associate the corresponding ground truth pose $P_{ref,ij}$

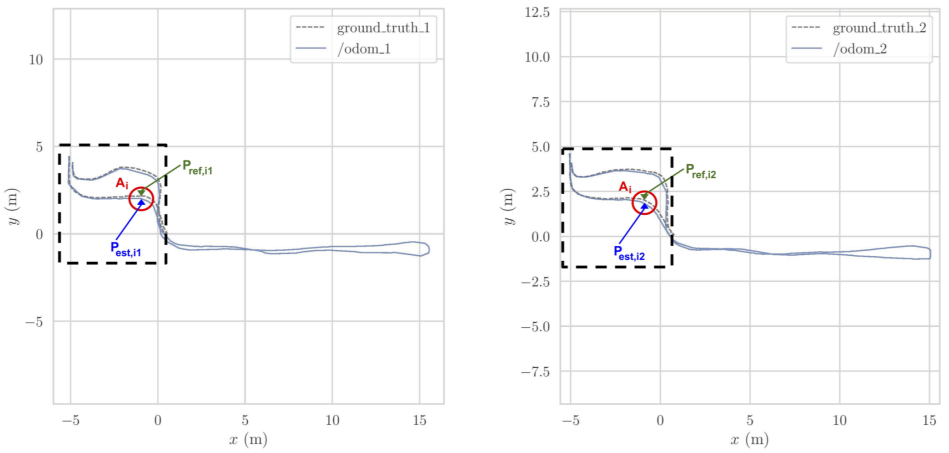


Figure 3.11: *Two indoor trajectories associated with the indoor re-localization error experiments; the OptiTrack area is black-dashed contoured. On the left is the first path, while on the right is the second trajectory.*

as the one closer in time according to the timestamps of the messages. Given that the OptiTrack produces data 50 to 100 times faster than the odometry algorithms, we are sure to find very close correspondences in time. We tried to manually park the system in the same areas in both trajectories; nevertheless, it is nearly impossible to manually re-locate in the same place. We estimate this difference in re-positioning the cart using the OptiTrack.

Mathematically speaking, we computed the transformation matrix $T_{ref,i}$ between ground truth poses for each area A_i as

$$T_{ref,i} = transm(P_{ref,i1}, P_{ref,i2}),$$

and the matrix $T_{est,i}$ between estimated poses for each area A_i as

$$T_{est,i} = transm(P_{est,i1}, P_{est,i2}),$$

where $transm()$ is a function that gives as output the transformation matrix necessary to express the second argument of the function in the reference system of the first argument (as both are 3D poses).

Then, we calculated the error matrix E_i as the inverse composition between the two transformations $T_{est,i}$ and $T_{ref,i}$

$$E_i = inv(T_{ref,i}) * T_{est,i},$$

where $inv()$ is the usual inversion matrix operator. Intuitively, the matrix E_i tells us the difference in translation and rotation between the two transformations $T_{est,i}$ and $T_{ref,i}$. Finally, we estimated the translation and rotation

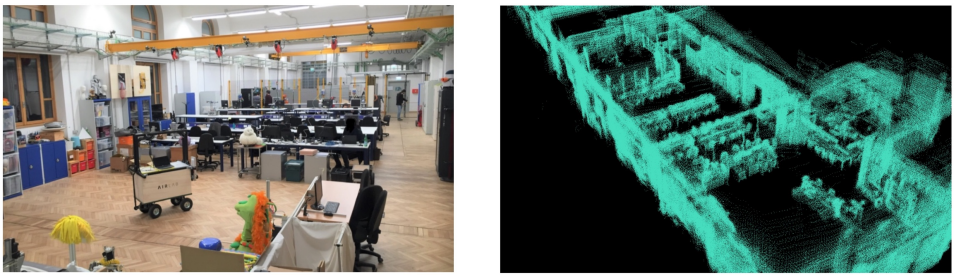


Figure 3.12: *Indoor area of the experimental campaigns. It consists of a long laboratory (left panel) with many obstacles, visible on the map, as a point cloud (right panel), obtained with HDL [28].*

errors, $trans\_e_i$ and rot_e_i, respectively, with the following metrics:

$$trans\_e_i = \|trans(E_i)\|,$$

where the function $trans()$ extracts the translation part of a given transformation matrix, while for the rotation part, we use

$$rot\_e_i = |angle(rotm2axang(E_i))|.$$

The function $rotm2axang()$ converts the rotation matrix extracted from a given transformation, to the corresponding representation of the axis angle, and the function $angle()$ extracts the angle from an axis-angle representation. The axis-angle representation is just one of many ways to represent rotations together with quaternions, Euler angles, rotation matrices, and many others.

The advantage of the axis-angle representation consists in condensing the rotation in one single number instead of having three components, as with Euler angles. A more detailed explanation of 3D rotation error metrics can be found in [153], including also the various representations.

In the outdoor scenario, since there was no ground truth, we only calculated the transformation matrix $T_{est,i}$ between the estimated poses $P_{est,i1}$ and $P_{est,i2}$, for each area A_i, considering it as error matrix. Obviously, this interpretation does not take into account possible re-positioning mistakes, which are then included in the whole re-localization error estimation.

3.2.3 Indoor experiment

In the following, we report the information relative to the first, indoor, experiment, corresponding to a large lab room. First, we give some generalities about the setup, data collection, and registered trajectories. Then, we show the comparison between the localization obtained through HDL and

Table 3.7: *Information about the two trajectories of the indoor experiment. The first one is used to build a 3D map of the environment using SLAM, while the other allows the evaluation of the selected localization algorithms.*

Feature	Trajectory 1	Trajectory 2
Duration	308 [s]	306 [s]
Estimated length	49.71 [m]	48.31 [m]
Estimated mean speed	0.28 [m/s]	0.24 [m/s]
Estimated max speed	0.52 [m/s]	0.50 [m/s]
# Laser scans	6167	6121
# IMU samples	30832	30606

ART-SLAM, discussing both translation and rotation errors of the second trajectory w.r.t. the map obtained from the first traversed path.

Setup and data collection

As an indoor scenario, we used the main large room of our laboratory (AIRLab, Politecnico di Milano, Italy), in Figure 3.12. The cart, described in Section 3.2.2 and visible in the left panel, was pulled along two trajectories, which characteristics are listed in Table 3.7. Ten areas $A_i, i \in \{1, \dots, 10\}$ have been selected in the room, to be used in the evaluation of the algorithms.

Indoor localization

Using the first trajectory $traj_1$ with both SLAM methods, ART-SLAM and HDL, we reconstruct an accurate representation of the room, in the form of a dense cloud. For example, the right panel of Figure 3.12 shows the map obtained with HDL, which contains about 1M points, while the number of acquired points sums up to 404M elements. The dimension and accuracy of the obtained map (and the same holds for ART-SLAM) are, along with the explanation done in Section 3.2.2, due to the algorithm being graph-based.

Not all the laser scans are saved and used to build the map, but just the most relevant (e.g., after a certain distance has been traveled). This allows to run long trajectories while maintaining high accuracy and being memory friendly, features that other algorithms in literature do not have.

As stated in Section 3.2.2, we use the first trajectory to build the 3D map. Then, both the first and the second trajectories are used for re-localization, obtaining multiple estimates of the robot pose (one for each laser scan). The translation re-localization errors for all the testing areas are represented in the left image of Figure 3.13, while the rotation re-localization errors

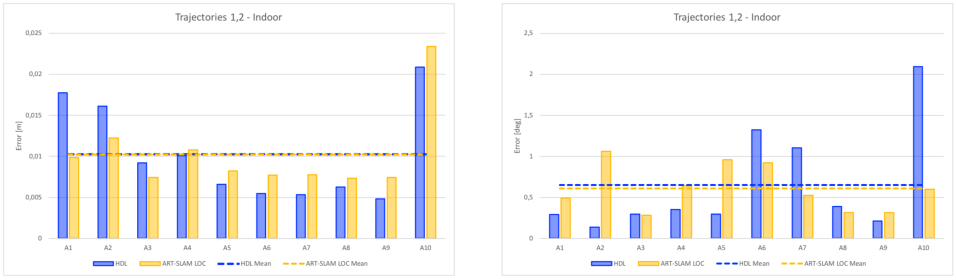


Figure 3.13: *Re-localization error of the indoor experiment, consisting of both translational and rotational components, using HDL [28] and ART-SLAM [1].*

can be seen in the right panel. Both HDL and ART-SLAM LOC achieve superior accuracy, resulting in errors in the magnitude of centimeters for the translation and tenth of degree angle for the rotation. While the accuracy is similar, ART-SLAM (with its localization module) has the advantage of being real-time (or even faster, see Section 3.1.10) and a zero-copy software.

Again, Figure 3.13 confirms that the length of a trajectory is not correlated to the attained precision. Both tested systems rely on UKF localization, with a correction step performed via scan-to-map point cloud alignment; this way, possible localization errors are bounded by the residual error of the previous scan and the error in one step prediction of the IMU. This was expected, as both errors correspond to one single step in the UKF algorithm and they get reset at each alignment, making both systems scalable and efficient.

Lastly, we give a brief comparison of the processing time of both algorithms when performing localization. ART-SLAM LOC processes inputs as fast as the data acquisition rate of the LiDAR (as the correction step is the bottleneck of the system, being dependent on scan matching between a point cloud and a whole 3D map), making it able to run in real-time. HDL, on the other hand, is slower, working at a lower processing rate.

3.2.4 Outdoor experiment

In the following, we describe the second experiment, which is conducted in an outdoor scene, which is an oil refinery. As before, first, we give some generalities about the setup, data collection, and registered trajectories. After that, we show the comparison between the localization obtained through HDL and ART-SLAM, on all five trajectories, discussing, once again, both translation and rotation errors w.r.t. the map obtained with the first trajectory.

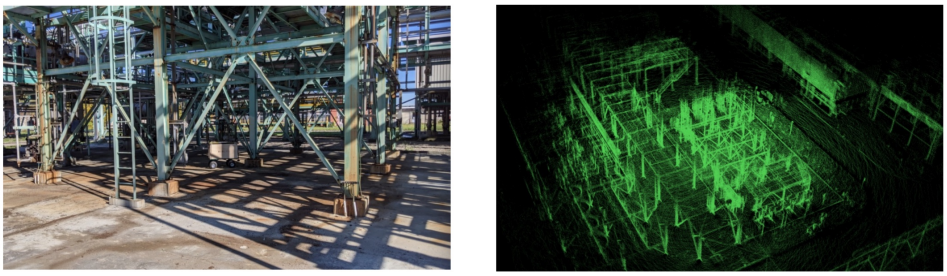


Figure 3.14: *Outdoor area of the experimental campaigns, representing an oil refinery, in Italy. The left panel shows the traversed environment, while the right image represents the 3D map, obtained with ART-SLAM [1].*

Table 3.8: *Information about the five trajectories of the outdoor experiment. The first one is used to build a 3D map of the environment using SLAM, while the other allows the evaluation of the selected localization algorithms.*

Feature	**Traj. 1**	**Traj. 2**	**Traj. 3**	**Traj. 4**	**Traj. 5**
Duration	489 [s]	454 [s]	403 [s]	552 [s]	519 [s]
Length	127.79 [m]	122.58 [m]	123.73 [m]	208.58 [m]	126.69 [m]
Mean speed	0.26 [m/s]	0.27 [m/s]	0.31 [m/s]	0.38 [m/s]	0.24 [m/s]
# Laser scans	9779	9084	8051	11043	10387
# IMU samples	48888	45416	40251	55208	51930

Setup and data collection

The chosen outdoor space for testing the mapping and re-localization algorithms is an oil refinery, in Italy, visible in the left panel of Figure 3.14. In this experiment, we pulled the cart along five trajectories (see Table 3.8).

Differently from the indoor scenario, where the two trajectories were almost identical, in the outdoor experiment, the fourth path is much longer than the one used to create the map, as it extends far from the designated area. Figure 3.15 left and right images correspond, respectively, to trajectories 1 and 4, making visible the difference between the two paths in terms of covered space. Overall, eleven areas $A_i, i \in \{1, \dots, 11\}$ have been selected in the outdoor area, to be used in the evaluation of the algorithms.

Outdoor localization

Using again the first of the five trajectories, $traj_1$, we are able to achieve a full representation of the environment, as a dense 3D map. Figure 3.14 shows the map obtained using ART-SLAM, which contains about 1M points,

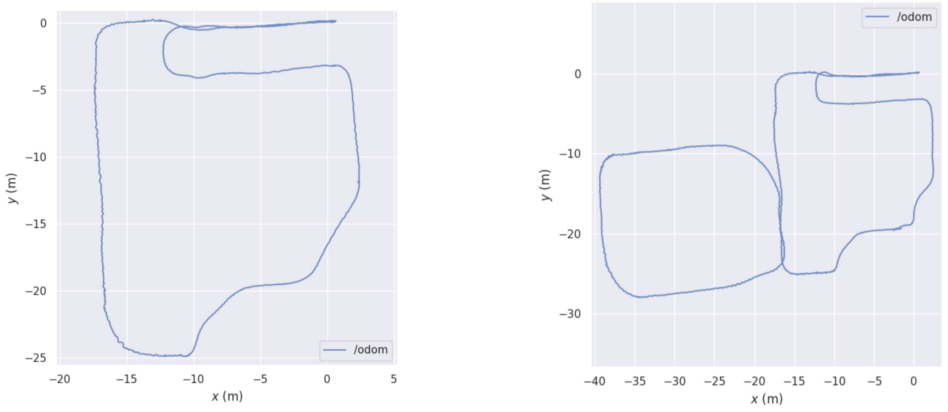


Figure 3.15: *Difference between the first and fourth trajectories of the outdoor experiment.*

Figure 3.16: *Translation re-localization errors of the outdoor experiment, using HDL [28] and ART-SLAM [1]. The accuracy of both systems is noticeable.*

the same as the one built during the indoor experiment. Stressing again the importance of using graph-based algorithms for SLAM, it is remarkable how comparing it with the indoor scenario, the outdoor experiment maintains the same degree of accuracy, even if the trajectory is longer. This is also a confirmation that the selected systems are scalable and efficient.

As stated in Section 3.2.2, we use the first trajectory to build the 3D map. Then, the five trajectories are used for re-localization, obtaining multiple

Figure 3.17: *Rotation re-localization errors of the outdoor experiment, using HDL [28] and ART-SLAM [1]. Again, both methods prove to achieve superior precision.*

estimates of the robot pose (one for each laser scan). Figure 3.16 and Figure 3.17 show, respectively, the translation and rotation re-localization errors for all the four pairs of trajectories (1-2, 1-3, 1-4 and 1-5), for all the testing areas (Figure 3.14). As for the indoor experiment, both HDL and ART-SLAM, with the localization module, are proven to be accurate.

Differently from the indoor scenario, errors are slightly larger, probably due to the noisy measurements, typical of outdoor environments, and the unavailability of ground truth compensating re-positioning errors. Moreover, the error distribution confirms that the precision of the systems does not depend on the size of the paths, confirming that they are both scalable.

CHAPTER 4

MCS-SLAM: Multi-cues and Multi-sensors Fusion SLAM

As we have seen in the state-of-the-art chapter of the thesis, and in particular in Section 2.6, to combine the advantages of both laser and visual SLAM systems, works in literature shifted their focus to hybrid approaches. Nevertheless, there is no true sensor fusion, as data coming from sensors of different natures are only used to support each other. To cope with this issue, which is essential to further improve existing SLAM methods, we developed a multi-cues and multi-sensors fusion SLAM algorithm, named MCS-SLAM [3], which we describe in the following section.

4.1 MCS-SLAM

Both in Chapter 2 and Chapter 3, we have seen the importance of accurate navigation and detailed map reconstruction, essential tasks in many real-world applications. While before we focused only on LiDAR-based methods, it is also opportune to remember that SLAM systems can also be classified as vision-based, if the considered main sensor is a camera, and hybrid if their input has multiple origins (usually in the form of images and clouds).

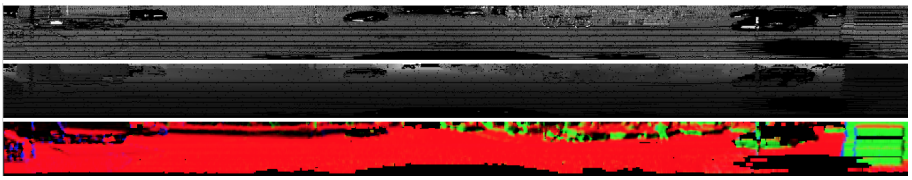


Figure 4.1: *From top to bottom: intensity, range, and normal images extracted from a point cloud of the KITTI odometry dataset [143]. The images cover the field of view of the LiDAR sensor used, i.e., 360 degrees horizontally and 26.8 degrees vertically.*

In particular, sensor fusion techniques using heterogeneous data could definitely allow for the improvement of the accuracy of existing SLAM systems. However, not only sensor fusion is hardly achieved in literature, but data association is most of the time explicit (as in ART-SLAM [1] or ART-SLAM LOC), as it heavily depends on the type of data used.

Della Corte and Bogoslavskyi et al. [154] recently proposed a general algorithm for multi-cue photometric registration of 3D point clouds, designed without considering a specific sensor, nor a particular cue, which represents information about sensor data. The method they proposed, to perform odometry estimation rather than SLAM, uses multiple cues (see Figure 4.1) from input data (either RGB-D images or LiDAR point clouds), namely color and range (or depth), which are then used to compute normals.

By doing this, the algorithm avoids an explicit point-to-point data association and is able to compute the transformation between viewpoints under realistic disturbances from an initial motion guess. However, as the length of the trajectory increases, the method drifts, being for odometry estimation only (no form of loop detection is done). Moreover, the approach can only handle one sensor at a time (e.g., RBG-D camera or LiDAR), and it is not able to exploit common sensor suites installed on autonomous vehicles.

To explore the possibilities of sensor fusion and to extend the already existing SLAM framework, consisting of both ART-SLAM and its localization module, we proceeded with our Ph.D. program by developing an extension of the work in [154] to multiple sensors, integrating it in the graph SLAM backbone of ART-SLAM. The goal was to perform fast and accurate multi-sensor and multi-cue SLAM, with the following contributions.

The proposed system, named MCS-SLAM (Multi-Cues Multi-Sensors Fusion SLAM) performs tracking by integrating data coming from multiple sensors while avoiding explicit data association. The method is also capable of efficiently detecting and closing loops, using a multiple-step algorithm, and it optimizes the estimated trajectory through the g2o optimization

framework, allowing for the inclusion of corrective data, e.g., from IMU.

We first discuss related works in Section 4.1.1, to give a brief review of hybrid SLAM methods in the literature. In Section 4.1.2 we give a high-level description of the architecture of MCS-SLAM, followed by a discussion about all of its modules. The pre-filterer is described in Section 4.1.3 along with the cloud projector module, in Section 4.1.4. The way tracking is performed is available in Section 4.1.5, followed by loop detection and pose graph handling in Section 4.1.6 and Section 4.1.7, respectively. Lastly, in Section 4.1.8 and Section 4.1.9 we describe the evaluation of the system.

4.1.1 Related works

To combine the advantages of visual and laser SLAM, the focus shifted to hybrid systems, which couple the data coming from different sensors, with the goal of achieving accurate results in real-time. The work in [155] presented an RGB-D camera with LiDAR EKF SLAM, with the purpose of tackling the issue of unsuccessful visual tracking. If visual tracking fails, the LiDAR pose is used to localize the point cloud data of the RGB-D camera.

In Limo [156], LiDAR measurements are used for depth extraction, as 3D points are projected on the corresponding RGB images, which are employed later in keyframe-based bundle adjustment. Zhu et al. [157] developed a 3D laser SLAM system associated with a visual method to perform loop detection through a keyframe-based technique, using visual bags-of-words. The work in [158] used both visual and LiDAR measurements by first running in parallel SLAM for each modality, and then coupling the data.

Hybrid systems give accurate results, but they do not truly target sensor fusion, as one type of data is often used to improve the quality of an existing system based on another type of data (e.g., point clouds to aid Visual SLAM methods or images to improve LiDAR SLAM efficiency). The closest work to sensor fusion is [159], where graph optimization is performed using a specific cost function, considering both laser and feature constraints.

4.1.2 Architecture and overview

A high-level overview of the system architecture of the proposed system can be seen in Figure 4.2. MCS-SLAM is made up of distinct modules, the gray boxes in the figure, which represent the core of the system. The current implementation of MCS-SLAM requires point clouds as mandatory input, as loop closure involves scan matching between a pair of clouds.

Given an incoming laser scan, the first step is to process it, in the *pre-filterer* and in the *cloud projector*. The first reduces the size of the cloud,

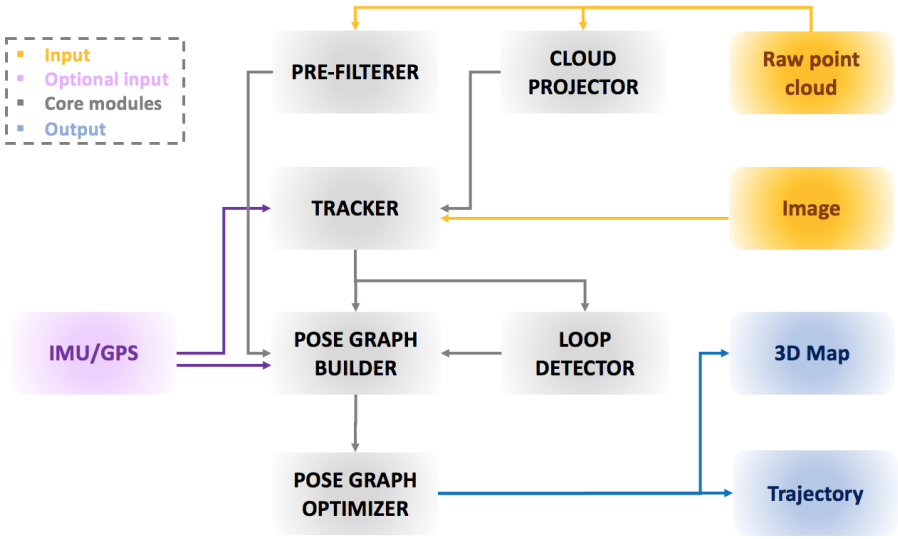


Figure 4.2: *High-level architecture of the proposed system, MCS-SLAM [3].*

downsampling it, and it removes noisy elements, as in ART-SLAM [1]. The filtered point cloud is used later for efficient loop detection, to correct the estimated trajectory. The cloud projector, instead, converts the 3D point cloud into two 2D images. One represents the ranges of all elements of the point cloud, while the other displays their intensities.

The two images are used by the core component of the system, the *tracker*, which estimates the current displacement of the robot by performing two consecutive steps. First, pairs of images derived from one or multiple sensors (e.g., two LiDARs, a LiDAR and a camera, an RGB-D camera and a LiDAR) are processed in parallel and independently one from the other, to obtain a rough estimate of the current motion. Then, these estimates are jointly optimized to get the transformation that best satisfies all constraints.

The current pose estimate is sent, along with its corresponding filtered LiDAR scan, to the *loop detector* module, which tries to efficiently find loops between new and previous clouds, via scan-to-scan matching. Poses and loops are used to build the pose graph, which is periodically optimized to increase the accuracy of the poses. *IMU and GPS* data (pink boxes in Figure 4.2) can also be integrated both in the tracker, to give it an initial guess for improving the estimated motion, and in the *pose graph builder*.

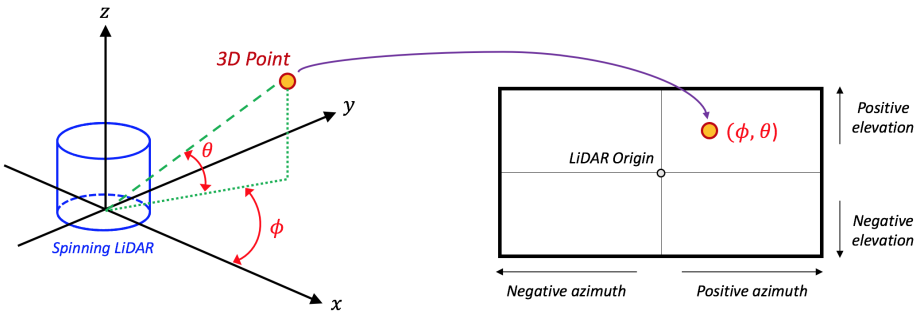


Figure 4.3: *Coordinate system of a spinning LiDAR on the left, with ϕ being the azimuth angle, and θ being the elevation angle. Given a 3D point, its 2D counterpart in the corresponding spherical image is represented on the right.*

4.1.3 Pre-filterer

The pre-filterer module is the same as in ART-SLAM. Its goal is to reduce the number of elements of a point cloud, while also removing noisy data and outliers. In MCS-SLAM, pre-filtering is not used for tracking, where having dense point clouds is, instead, a benefit, but for loop closure. Indeed, scan-to-scan matching is one of the steps followed to detect loops, and it is essential to have reduced clouds, to achieve detection in reasonable time.

4.1.4 Cloud projector

As the name implies, the cloud projector module has the purpose of converting a point cloud into 2D images, representing range and intensity. Projection is achieved through the spherical projection model, which best captures the characteristics of laser rangefinders. Spherical projection offers several advantages, such as being able to capture the full field of view and being able to maintain consistent point density across the whole projection.

The process of projecting point clouds from a spinning LiDAR sensor into a spherical image involves converting the Cartesian coordinates of the measurement points into spherical coordinates. Specifically, the process converts each point in the point cloud represented by its Cartesian coordinates $[x, y, z]$, into spherical coordinates represented by $[\phi, \theta, r]$. This conversion is illustrated in Figure 4.3. The ϕ coordinate corresponds to the azimuth angle of the point in the XY plane, θ is the elevation angle from the positive Z-axis, and r is the Euclidean distance from the origin. Spherical projection is a way to capture the geometry of the sensor in a single image.

Let K be a camera matrix, where f_x and f_y specify respectively the resolution of azimuth and elevation and c_x and c_y their offset in pixels, then

the spherical projection of a 3D point $P = [p_x, p_y, p_z]^T$ is given by:

$$proj(P) = K \begin{bmatrix} atan2(p_y, p_x) \\ atan2(p_z, \sqrt{p_y^2 + p_x^2}) \\ 1 \end{bmatrix}. \tag{4.1}$$

4.1.5 Tracker

Tracking, which follows short term data association, is the task of finding the relative motion between two consecutive poses of the robot, independently of the sensor(s) used. The tracker module operates in two consecutive phases: standalone motion estimation and collective motion optimization.

The *standalone motion estimation* phase is done on all data coming from multiple sensors, independently from each other. In particular, for each sensor, the pipeline described in [154] is executed, to obtain multiple, independent, rough estimates of the robot motion. As a consequence, considering S sensors, we are able to obtain S motion estimates by running the algorithm presented in [154], in parallel (at least one thread for sensor).

This approach seeks to register either two observations with respect to each other or an observation against a 3D model. Sensor observations are normalized into a 2D representation, e.g., an image from a regular camera, or a range image from a LiDAR (as explained previously in Section 4.1.4). This representation consists in a multi-cue image, where each pixel contains different types of information, i.e., light intensity, range (or depth) information, and surface normals, as represented in Figure 4.1.

Each measurement is aligned to a model $M = \{P_i, i = 0, \ldots, N\}$, for which 3D information is available in the form of a cloud enriched with the different cues. As in photometric error minimization approaches, the method tries to iteratively minimize the pixel-wise difference between the current multi-cue image I and the predicted image $\hat{I}(M, X)$, the latter being a multi-cue image obtained by projecting the model M onto a virtual camera located at the estimated pose P_X (occlusions in the model are handled with depth/range buffers so that only the closest point to the LiDAR sensor is kept). X is the transformation matrix that transforms the points of M from the global into the local camera coordinate system.

The goal is to find the best transformation X, such that:

$$\begin{aligned} X^* &= \underset{X}{argmin} \sum_{u,v,c} \left\| \hat{I}^c_{u,v}(M, X) - I^c_{u,v} \right\|^2_{\Omega^c} \\ &= \underset{X}{argmin} \sum_{u,v,c} e^c_{u,v}(M, X)^T \, \Omega^c \, e^c_{u,v}(M, X), \end{aligned} \tag{4.2}$$

where $e^c_{u,v}$ is the error at pixel (u, v) between the predicted value $\hat{I}^c_{u,v}$ and the measured value $I^c_{u,v}$ for a particular index c associated to a cue, and $\Omega = diag(\{\Omega^c\})$ is a diagonal information matrix used to weight the cues.

Instead of solving Equation 4.2, the approach in [154] re-formulates it as a linear system, where a perturbation Δx is calculated iteratively, as

$$X^* = X \oplus \Delta x, \tag{4.3}$$

where X is an initial guess of the transformation and Δx is a vector with six elements, corresponding to the difference in translation and orientation.

This formulation comes from the Taylor expansion of the error described above, combined with the just described Equation 4.3:

$$e^c_{u,v}(X \oplus \Delta x) \backsimeq e^c_{u,v}(X) + \left. \frac{\delta e^c_{u,v}(X \oplus \Delta x)}{\delta x} \right|_{\text{x=0}} \Delta x \tag{4.4}$$

$$= e^c_{u,v}(X) + J^c_{u,v}(X) \Delta x. \tag{4.5}$$

The minimization problem then becomes

$$\Delta x^* = \underset{\Delta x}{argmin} \sum_{u,v} w_{u,v} \sum_{c} \left\| e^c_{u,v}(X) + J^c_{u,v}(X) \Delta x \right\|^2_{\Omega^c}, \tag{4.6}$$

with $w_{u,v}$ being a regularization weight. This, in turn, is equivalent to solving the linear system $H \Delta x^* = b$, with the terms H and b, given by

$$H = \sum_{u,v} w_{u,v} \sum_{c} J^c_{u,v}(X)^T \Omega^c J^c_{u,v}(X) \tag{4.7}$$

$$b = \sum_{u,v} w_{u,v} \sum_{c} J^c_{u,v}(X)^T \Omega^c e^c_{u,v}(X). \tag{4.8}$$

This procedure is done incrementally and on multiple scales of the input 2D images, to find the best Δx^*, while avoiding falling into local minima.

As previously stated, in MCS-SLAM, this pipeline is executed once for each sensor, in parallel, such that data coming from sensor i generates a motion estimate $trans_i$. Once all estimates are computed, the tracker

proceeds to the next phase, namely the *collective motion optimization*, taking also into account the rotational offsets of the various sensors.

First, all motion estimates associated with the multiple sensors are weighted and summed, with each weight w_i being inversely proportional to the re-projection error previously computed as part of the procedure to find transformation $trans_i$. In this way, a single collective motion of the robot is obtained, combining the tracking information associated with each sensor.

This unique transformation is then fed again to each standalone motion estimation pipeline, as an initial guess, and optimized only for one iteration. These two steps form a single iteration of the collective motion optimization and are continuously repeated, until convergence, or up to N times.

The tracker of MCS-SLAM adopts a keyframe-based approach to estimate the trajectory of the robot, as in ART-SLAM. MCS-SLAM, differently from the majority of SLAM systems in literature, works using two types of keyframes: one for tracking and one for loop closure and pose graph construction and optimization, independently one from the other.

Keyframes associated with the tracker hold the current estimated position of the robot and the corresponding multi-cue images. For each sensor i, we keep track of the current keyframe $K_i^{tracker}$, as it has no other purpose than to avoid useless computations (as for the scan-to-keyframe matching in ART-SLAM). Keyframes related to the other modules (i.e., loop closure detection and pose graph handling) contain a point cloud and the pose (odometry) estimated by the tracker, along with the timestamp of the cloud, accumulated distance from the beginning and, if available, camera images. We refer to these keyframes with K_j^{graph}, since they are computed following the collective optimization phase and are mainly used in the pose graph.

The first tracker keyframe, for each sensor, and the first graph keyframe, correspond to the first point cloud received by the system. Consecutive keyframes (this applies independently from the type) must satisfy at least one of the criteria described for ART-SLAM, in Section 3.1.4.

Keyframes $K_i^{tracker}$ correspond to very low thresholds, as we want to compare images close in time, to avoid accumulating initialization errors. In our experiments, we use $\Delta trans$ and $\Delta orientation$ of, respectively, 50 centimeters and 1 degree. The poses obtained by the tracker after the collective optimization are then filtered through higher thresholds, to obtain multiple graph keyframes K_j^{graph}, which are stored in the pose graph. In our experiments, we set $\Delta trans$ to 5 meters and $\Delta orientation$ to 5 degrees.

4.1.6 Loop detection

As we have already seen, loops are additional useful constraints to insert into the pose graph, allowing the correction of drifts and estimation errors. In the current MCS-SLAM implementation, the detection of loops is done the same way as in ART-SLAM. It is strictly associated with the availability of point clouds, which are mandatory inputs in the current implementation of the system, and it involves only the K^{graph} keyframes. In the following, we drop the suffix $graph$, referring to keyframes only as K_j.

Whenever a keyframe K_{query} is created, it is compared with all the other keyframes, which are candidates for loop closure. To make loop detection scalable, odometry-based filtering is performed. A pair K_{query} and $K_{candidate}$ is kept only if the two keyframes correspond to estimated poses far in time but close in location. If K_{query} and $K_{candidate}$ satisfy these constraints, they probably correspond to a loop in the trajectory.

Once all the candidates have been discovered, they are compared using the approach described in [160], which converts the corresponding clouds into a bird-eye view grid and selects the pairs with the most similar cells. At the end of this step, only k candidate pairs for loop detection remain.

These few candidates are then compared using scan-to-scan matching, to find the alignment between the point clouds of each pair. Then, all the transformations obtained are confronted. If found, the transformation corresponding to the smallest distance and highest accuracy represents a loop, which is then added to the pose graph as a new constraint.

4.1.7 Pose graph building and optimization

As ART-SLAM, MCS-SLAM is a graph SLAM [139] system, where the poses of the robot are modeled as nodes in the pose graph, and edges represent spatial constraints resulting from tracking or measurements coming from different sensors, e.g., IMU or GPS. Moreover, each node j is associated with the corresponding keyframe K_j^{graph} and edges can be added also when performing loop detection and closure, between non-consecutive nodes in the graph. Periodically, the pose graph is optimized to best satisfy the constraints provided by the measurements associated with each edge.

4.1.8 Experimental validation of the system

MCS-SLAM [3] has been compared against four methods for point cloud-based SLAM (LeGO-LOAM-BOR, which is an improved variant of LeGO-LOAM [32], LIO-SAM [115], HDL [28] and the baseline, ART-SLAM [1]), and two vision-based systems (ORB-SLAM2 [53] and ORB-SLAM3 [75]).

Figure 4.4: *Range images obtained through the spherical projection of a point cloud split in two, from City Sequence 05 of the KITTI raw dataset [144]. The image on the left corresponds to the environment in front of the LiDAR acquiring the scan, while the image on the right represents the part of the scene behind the sensor.*

Figure 4.5: *From left to right, RGB image, sparse range image obtained through LiDAR-to-camera projection and enhanced range image, from city Sequence 05 of the KITTI raw dataset [144]. The enhancement is achieved through simple bilinear filtering.*

IMU data, which is mandatory in LIO-SAM, is also used in ART-SLAM, to deskew point clouds and to enforce rotational constraints in the pose graph. We evaluate MCS-SLAM in two scenarios coming from the KITTI dataset [143, 144], corresponding to a short path without loops and a medium sequence with one closure at the end of the trajectory.

To test the proposed system, we run it under five different conditions. In the first case, from a single point cloud covering the surrounding environment (360 degrees horizontal field of view), we generate a single multi-cue image, having no color information (meaning that we consider only range and normal cues). In the second and third scenarios, we split the point cloud in two and use only the points, respectively, in front and behind the LiDAR, covering a horizontal field of view of 180 degrees each, obtaining the range images represented in Figure 4.4. It should be noticed that the first three experimental campaigns involve only a single sensor, and they are used to evaluate the SLAM part of MCS-SLAM.

The fourth case considers the point clouds, from the second and third scenarios, as coming from two different sensors. This way, for each input, the standalone motion estimation is performed, followed by the collective motion optimization, described in Section 4.1.5 (differently from the first three cases, where only the standalone motion estimation is needed).

Lastly, in the fifth scenario, we project LiDAR data onto the corresponding RGB image, obtaining a large multi-cue image (color and range). As it can be seen in the central picture of Figure 4.5, the projected cloud is sparse w.r.t. the colored image. For this reason, we first apply windowed bilateral filtering on the sparse range image, then we fill each column with the highest range value in them, obtaining the right picture of Figure 4.5. The fourth

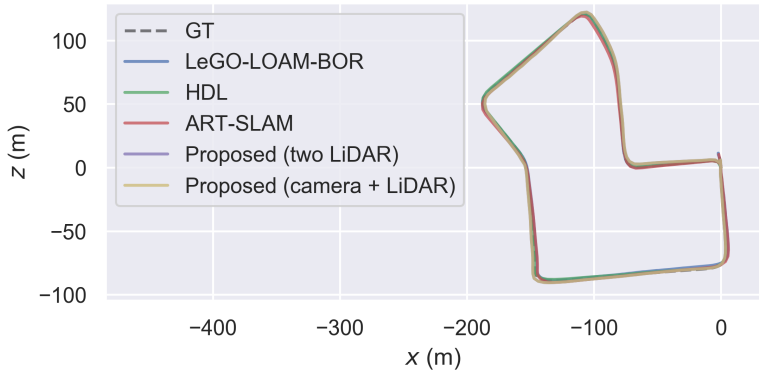


Figure 4.6: *Comparison between the trajectories estimated by LeGO-LOAM-BOR (derived from LeGO-LOAM [32]), HDL [28], ART-SLAM [1] and the proposed MCS-SLAM [3] (in the multiple sensors scenarios), on Sequence 07 of the KITTI odometry dataset [143]. The other methods considered are not included, to avoid overlapping.*

and fifth experimental campaigns involve two sensors (two pseudo-LiDAR and a LiDAR coupled with a camera, respectively), and they are used to evaluate the data and sensor fusion capabilities of MCS-SLAM.

As always, experiments are tested on a 2021 XMG 64-bit laptop with Intel(R) Core(TM) i7-11800H CPU @ 2.30GHz x 8 cores, each with 24576 of cache size, and no GPU has been used in the comparison.

4.1.9 Comparison and results

As done for ART-SLAM, to evaluate the systems we compute the absolute trajectory error (ATE), i.e., the difference between coordinates of the points belonging to the true and the estimated trajectories, and show the processing time, per frame, of the most important modules of MCS-SLAM.

Figure 4.6 shows some of the estimated trajectories on Sequence 07 of the KITTI odometry dataset [143]. All the evaluated methods present a high degree of accuracy, following the ground truth trajectory and easily detecting the loop at the end of the path. Table 4.1 further details the obtained results, as it represents the mean, root mean squared error (RMSE), and standard deviation (STD) of the absolute trajectory error, in meters.

The proposed MCS-SLAM presents an accuracy within an acceptable threshold of about 1.8 meters, which is similar to the LeGO-LOAM-BOR. This result was expected, as tracking performed through MCS-SLAM loses precision due to the projection of point clouds in 2D images, which are later back-projected in 3D clouds, as described in [154]. Moreover, all results associated with the five considered scenarios behave as expected, with the

Table 4.1: *Computed ATE on Sequence 07 of the KITTI odometry dataset [143].*

ATE [m]	MEAN	RMSE	STD
LeGO-LOAM-BOR	1.604	1.807	0.832
LIO-SAM (mandatory IMU)	0.509	0.675	0.351
HDL	0.954	1.253	0.767
ART-SLAM (with IMU)	**0.343**	**0.366**	**0.127**
ORB-SLAM2	1.189	1.326	0.585
ORB-SLAM3	1.310	1.527	0.785
MCS-SLAM (one LiDAR)	1.679	2.219	1.137
MCS-SLAM (one LiDAR, front)	1.894	2.392	1.215
MCS-SLAM (one LiDAR, back)	1.917	2.421	1.054
MCS-SLAM (two LiDAR)	1.714	1.892	0.798
MCS-SLAM (camera + LiDAR)	2.822	2.924	1.957

Table 4.2: *Computed ATE on city Sequence 05 of the KITTI raw dataset [144].*

ATE [m]	MEAN	RMSE	STD
LeGO-LOAM-BOR	1.094	1.169	0.409
LIO-SAM (mandatory IMU)	**0.493**	**0.338**	**0.280**
HDL	0.893	0.912	0.476
ART-SLAM (with IMU)	0.746	0.814	0.326
ORB-SLAM2	5.136	6.141	3.366
ORB-SLAM3	5.174	6.178	3.376
MCS-SLAM (one LiDAR)	0.679	0.807	0.437
MCS-SLAM (one LiDAR, front)	0.561	0.639	0.305
MCS-SLAM (one LiDAR, back)	0.708	0.806	0.383
MCS-SLAM (two LiDAR)	0.578	0.659	0.317
MCS-SLAM (camera + LiDAR)	1.341	1.551	0.892

one LiDAR case being the most accurate, followed by the fusion of front and back. The camera plus LiDAR setup is less accurate, as range image densification is achieved through simple bilinear filtering and completion.

To evaluate the accuracy when loop closures are not available, we also considered a short sequence. As the dataset corresponding to raw odometries does not have a ground truth, we use, instead, GPS data, provided along with the point clouds and RGB images. Figure 4.7 represents the estimated trajectories on city Sequence 05 of the KITTI raw dataset [144].

As for the medium-length sequence, all the methods considered for evaluation tightly follow the trajectory, even if no loop closure is available to correct drifts, even though for shorter sequences there is not enough room

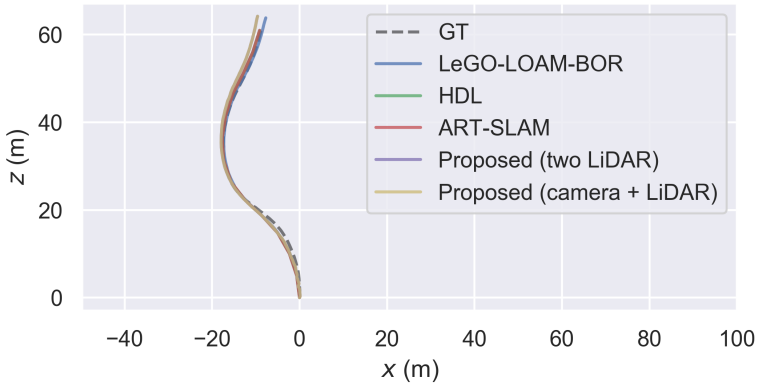


Figure 4.7: *Comparison between the trajectories estimated by LeGO-LOAM-BOR (derived from LeGO-LOAM [32]), HDL [28], ART-SLAM [1] and the proposed MCS-SLAM [3] (in the multiple sensors scenarios), on city Sequence 05 of the KITTI raw dataset [144]. The other methods considered are not included, to avoid overlapping.*

to accumulate such errors. Table 4.2 shows the ATE statistics, in meters. As before, all systems show good results, with the proposed method performing almost as well as the method with the best accuracy (with a difference of about ten centimeters). These results are also motivated by the fact that the trajectory is relatively simple, almost straight with very smooth turns.

Among all the systems, only HDL is not able to run real-time, being two to three times slower than the data acquisition rate (which is one frame every 100 milliseconds, i.e., 10 Hz). LeGO-LOAM-BOR, LIO-SAM, ORB-SLAM2, and ORB-SLAM3 are designed to perform real-time SLAM, with LIO-SAM being even faster. Moreover, as described in Section 3.1.10, also ART-SLAM is able to achieve real-time results, even on long sequences.

Table 4.3 shows the average processing time, per frame, of MCS-SLAM, in all five scenarios considered in the evaluation, for both sequences. MCS-SLAM, despite being less or as accurate as the other methods, is able to run more than five times faster than the data acquisition rate. It should be noticed that tracking is performed in parallel to the whole loop detection and graph construction and optimization procedure, proving, once again, that MCS-SLAM can be faster than real-time and used on low-end devices.

Intuitively, the split single-sensor scenarios (back and front) are associated with the lowest runtime, as we are dealing with small images (half the width of the image obtained in the one LiDAR case). The processing time associated with the camera plus LiDAR setup may surprise, but it is due to the densification of the projected point cloud onto the RGB image.

Lastly, we include a visual evaluation of the map obtained by MCS-

Table 4.3: *Comparison of the processing time [ms], per frame, of the various modules, between the variants of MCS-SLAM [3].*

Time [ms]	Seq.	Processing	Tracker	Loop det.	Optimization
base (one LiDAR)	07	10.172	22.648	10.226	1.320
base (one LiDAR, front)		8.907	13.893	10.226	1.320
base (one LiDAR, back)		8.907	13.753	10.226	1.320
base (two LiDAR)		9.371	16.393	10.226	1.320
base(camera + LiDAR)		149.823	19.712	10.226	1.320
base (one LiDAR)	City	10.172	22.648	6.819	0.132
base (one LiDAR, front)		8.907	13.597	6.819	0.132
base (one LiDAR, back)		8.907	13.642	6.819	0.132
base (two LiDAR)		9.371	17.241	6.819	0.132
base (camera + LiDAR)		155.712	21.712	6.819	0.132

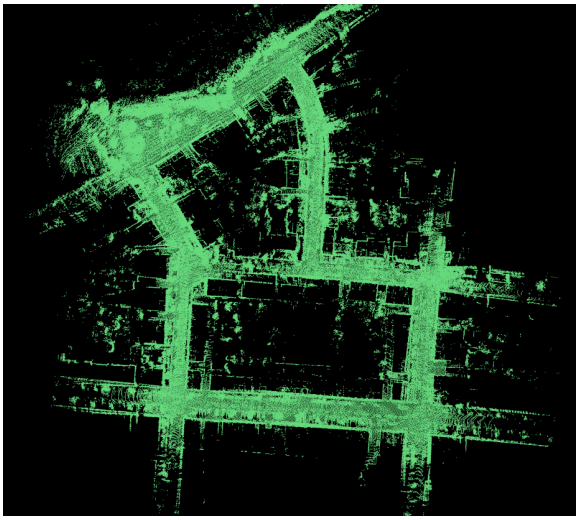


Figure 4.8: *Map obtained by MCS-SLAM [3] with two LiDAR, on Sequence 07 [143].*

SLAM (Figure 4.8), in the two LiDARs scenario, of Sequence 07 of the KITTI odometry dataset [143], proving the accuracy of the method.

CHAPTER 5

OSM-SLAM: SLAM with OpenStreetMap Priors

Simultaneous localization and mapping systems are increasingly becoming more accurate and less computationally demanding, independently of the type (e.g., laser SLAM, visual SLAM, and so on). Nevertheless, with the rise in Earth-level mapping systems and services, the availability of prior maps is resulting in a new standard, in many applications (e.g., mobile).

Not only these prior maps could be efficiently integrated into existing SLAM systems to improve their accuracy, but they could also be exploited to cope with common issues present in the majority of works, such as the need to re-localization in case of sensor failure or lack of input data. For these reasons, we developed a SLAM system aided with OpenStreetMap prior maps, named OSM-SLAM, described in the following section.

5.1 OSM-SLAM

SLAM methods address the problem of constructing a model of the environment surrounding the robot, i.e., the map, while simultaneously estimating its pose within it. As we have seen in the previous chapters of the thesis,

state-of-the-art systems are capable of building a map of the environment using only equipped LiDAR sensors and do not exploit existing prior maps, which can be integrated to achieve better localization. Moreover, temporary loss of input data, e.g., caused by a sudden sensor failure, proves to be a challenge for almost all methods, as it requires efficient re-localization to continue the whole trajectory estimation and mapping process.

To address these problems, we moved from the idea of multi-sensors SLAM, as described in Chapter 4, and returned to the base of our framework, being accurate, real-time, and laser-based. The fourth step of our Ph.D. program consisted of the development of an extension of ART-SLAM [1] (meaning that it is used as the baseline) in which we include information derived from OpenStreetMap maps, hence the name OSM-SLAM [4].

In particular, we improved the pose graph construction and optimization, achieved through the g2o framework [141], adding knowledge about the buildings surrounding the robot, at a given location. Our work stands from other approaches leveraging on external maps as we add buildings to the pose graph, which represents 3D information, as 2D nodes, actively participating in the optimization phase. This allows to simultaneously correct both the estimated trajectory, via LiDAR tracking, and the buildings themselves, in case their OSM placement is not coherent with at least one input scan.

It should also be noted that, in our work, the pose graph is built using nodes belonging to different dimensions (2D for buildings and 3D for robot poses), while in the majority of works, all data are brought in the same dimension, usually 2D. OSM-SLAM is able to cope with three different scenarios: buildings are fixed and only the estimated trajectory is corrected (*Prior SLAM*), buildings surrounding a robot pose are moved by the same rigid motion while correcting the estimated trajectory (*Rigid SLAM*), and every single building is roto-translated with respect to a single input scan while constraining the corresponding robot pose (*Non-rigid SLAM*).

We first discuss related works in Section 5.1.1, to give a brief insight into existing systems for LiDAR-based SLAM using OpenStreetMap data. Then, we explain OSM-SLAM in Section 5.1.2, going into more detail about its implementation. In particular, Section 5.1.3 and Section 5.1.4 are dedicated to the explanation of the processing steps needed to convert point clouds and OSM maps into a suitable form for successive alignment. Then, in Section 5.1.5 and Section 5.1.6 we describe three implemented approaches in OSM-SLAM, namely Prior SLAM, Rigid SLAM, and Non-rigid SLAM. Lastly, Section 5.1.7 and Section 5.1.8 are dedicated to the experimental validation of our system, which includes a discussion about the influence of the quality of the maps on the localization and mapping approach.

5.1.1 Related works

To further improve the accuracy in SLAM systems, over the past years, many works have been proposed, which exploit already available pre-computed 2D maps, coming from external mapping services, such as Google Maps, OpenStreetMap [161] (OSM), or maps from the local land registry.

In particular, information from OSM seems to be the best choice, in terms of availability, as it requires no permissions or tokens to retrieve 2D data. Many systems relying on OSM include maps into the observation model of a Monte Carlo localization, as in [162], in which buildings are extracted from the 2D map as a set of lines and are used to compute the expected range measurement at a given pose of the robot, or as in [163], where the trajectory of the robot is aligned w.r.t. the road network.

A more recent graph SLAM system, proposed by Vysotska et al. [164], directly related LiDAR measurements with the data associated with buildings coming from OSM. This alignment is included in the pose graph, in the form of a localization error w.r.t. the available OSM map. A disadvantage of this approach is that a precise alignment between buildings and LiDAR scans is required in order to get good accuracy on the estimated robot poses.

However, alignment becomes difficult when there is a lot of clutter in the environment, typical of urban areas, or in zones when there are few buildings surrounding the robot (in the OSM map, not necessarily in the physical world). Other problems arise when buildings are not correctly positioned in OSM, caused by human errors or wrong annotations when creating the 2D maps. Lastly, the OSM map could also not be representative of the real environment, as the topology may change over time.

Instead of directly using OSM maps, Naik et al. [165], proposed a graph-based semantic mapping approach for indoor robotic applications, which extends OSM with robotic-specific, semantic, topological, and geometrical information. They introduce models for basic structures, such as walls, doors, or corridors, which are semantically grouped into a graph. Its hierarchical structure is then exploited to allow accurate navigation, whose accuracy is compatible with grid-based motion planning algorithms.

OpenStreetMap provides many advantages, such as global consistency, a heavy-less map construction process, and a wide variety of publicly available road information. The work in [166] presented an autonomous navigation pipeline that exploits OSM information as environment representation for global planning. To overcome one major issue of OSM maps, i.e., low local accuracy, the authors proposed a LiDAR-based Naive-Valley-Path method, which exploits the idea of valley areas to infer the local path always further

from obstacles. This allows navigation through the center of trafficable areas, following the shape of the road, independently of OSM errors.

Cho et al. [167] developed a vehicle localization (not SLAM) method purely based on OSM maps. Their method generates OSM descriptors by calculating the distances to buildings from a location in OpenStreetMap at a regular angle, and LiDAR descriptors by calculating the shortest distances to building points from the current location at a regular angle. Comparing the OSM descriptors and LiDAR descriptors yields a highly accurate vehicle localization result. Compared to methods that use prior LiDAR maps, the algorithm presents two main advantages: vehicle localization is not limited to only places with previously acquired LiDAR maps, and the method is comparable, in terms of precision, to LiDAR map-based approaches.

Following the ideas of [164] and [167], in the following we introduce OSM-SLAM [4] and show how the issues currently present in state-of-the-art approaches leveraging OSM information have been faced, providing a robust yet flexible system for 3D LiDAR SLAM in urban cluttered environments. Besides handling clutter and possible errors in the OSM information, OSM-SLAM is also able to perform re-localization using OSM maps as prior 2D maps, to handle possible situations where sensor data is lost for brief time periods (e.g., sensor failure, disconnection, or other issues).

5.1.2 System overview

An overview of the proposed method is presented in Figure 5.1. As already stated, OSM-SLAM is built upon the first system developed in our Ph.D. program, ART-SLAM [1], with which it shares the majority of modules used to perform straightforward LiDAR-based graph SLAM.

An input laser scan is processed to remove noise, reduce its size, and possibly deskew it. The new cloud is then used to perform scan-to-scan matching, estimate the robot motion, and possibly detect the ground plane to be later integrated into the pose graph as height and rotational constraints. As for ART-SLAM, being keyframe-based, the proposed system extracts, from the tracker module, only a few odometry estimates (depending on the current rotation, translation, and time gap w.r.t. previous scans). These estimates, along with corresponding point clouds, form a keyframe and are used to find loops in the trajectory and to build the pose graph.

Before the optimization of the pose graph is performed, we introduce a new pipeline, to allow a direct association between point clouds and buildings derived from OSM data. The idea behind this pipeline is that by aligning a prior map with a LiDAR scan, one can obtain meaningful

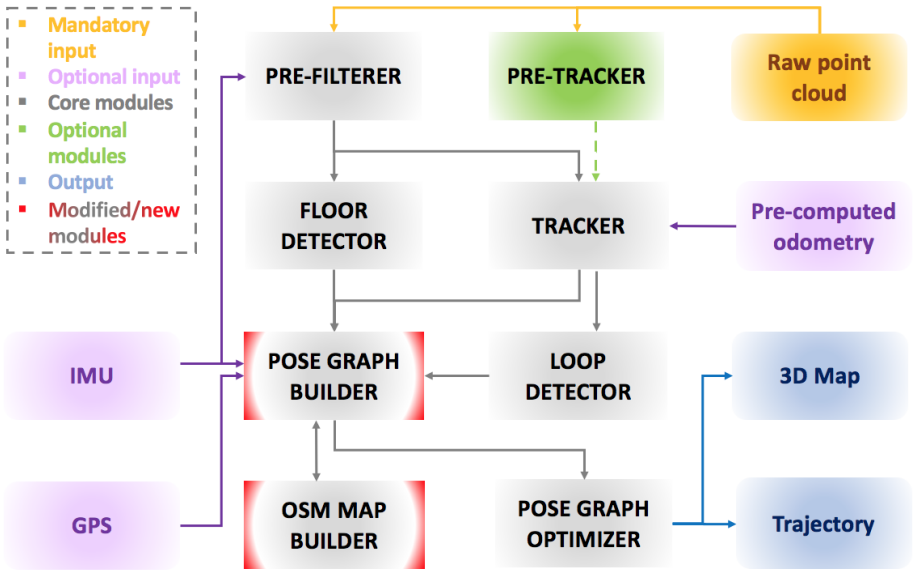


Figure 5.1: *Architecture of the proposed system. The majority of core modules come from ART-SLAM [1], while our improvements can be found in the pose graph builder, and in the new module OSM MAP BUILDER, which handles OpenStreetMap data.*

information about the pose of the robot (as the LiDAR scan is associated with it), w.r.t. the features present in the map (e.g., global coordinates of elements within it, like buildings), and vice-versa. A high-level scheme of the approach is represented in Figure 5.2. Once a keyframe, obtained from the front-end of the SLAM system, is inserted in the pose graph, it is ready to be given as input to the whole OSM data association procedure.

Odometry and 3D point cloud related to the keyframe considered are processed to extract a 2D map of the buildings, while odometry and optional GPS data (only one datum in the whole trajectory is required to make the method work, possibly at the beginning) are used to download and parse OSM data, to be converted in a 2D point cloud representing the buildings surrounding the robot. Once the buildings and the map from the LiDAR scan are retrieved, ICP [13] scan matching is performed, to obtain a first rough estimate of the rigid motion that should move the robot towards its true pose in the OSM map. This procedure is named *Rigid SLAM*, as alignment is done at a local level, involving all surrounding buildings.

Alignment can also (optionally) be repeated, using as an initial guess the just computed transformation, for each building, independently of one another. This allows to obtain more precise information about their displace-

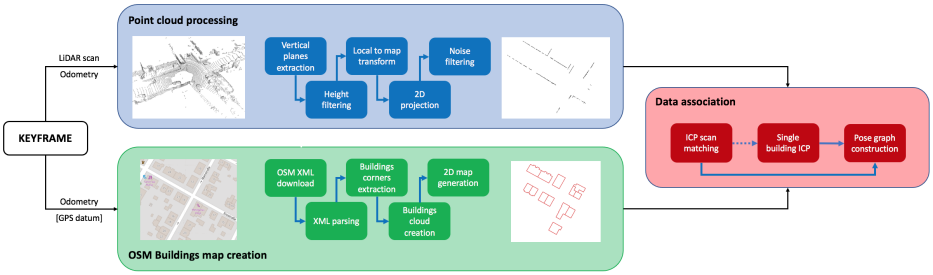


Figure 5.2: *High-level pipeline describing the contribution of the OSM-SLAM. The whole procedure happens during the pose graph construction before optimization takes place.*

ment, and possible errors in the OSM map. As, following this approach, buildings now exist as separate entities w.r.t. the pose of the robot, differently from the previous case where the computed transformation was the same for all buildings surrounding it, the name given is *Non-rigid SLAM*. Independently from the selected approach, i.e., Rigid or Non-rigid, all the constraints of the type {robot pose, building} are added to the pose graph.

It should also be noticed that buildings can be fixed during the graph optimization procedure, especially if it is known that the used 2D OSM map is precise and accurate. This solution, which takes the name of *Prior SLAM*, has the advantage of increasing the overall performance w.r.t. the non-fixed buildings counterparts, but it also maintains a relationship between buildings and pose of the robot, exploited in the optimization step. In other words, buildings are not modified but still constrain the estimated trajectory.

5.1.3 Point cloud processing branch

LiDAR scans cannot be directly aligned with a 2D prior map representing buildings. First, the point cloud is 3-dimensional, different from the prior map, leading to convergence errors, assuming that the scan matching procedure even starts. Then, LiDAR scans represent a richer environment than the one given by OSM maps, including many elements of disturbance, such as vehicles, road signs, fences, small walls, and even the ground itself. Lastly, using a whole point cloud would be wasteful, needing too many computational resources, due to the complexity of scan matching.

To overcome these problems, we propose a small sequence of preprocessing operations, to be applied on an input point cloud, with the goal of obtaining a 2D map of the buildings, extracted from it (blue box of Figure 5.2). As a first step, we remove all non-vertical and non-planar elements of the cloud, as they should not be part of the final map. This is achieved

through the iteration of a slightly modified version of RANSAC, i.e., Random Sample Consensus Model for Parallel Planes, to estimate all planes in the cloud which normal is perpendicular (meaning that the plane is parallel) to the normal to the ground, within a threshold angle. In our experiments, we consider a maximum value of 10 degrees, above which a plane is no longer considered a wall, to account also for errors in the LiDAR scans.

As a second step, we filter the obtained point cloud to only consider points above the LiDAR horizon. Indeed, it is very likely to find false walls between the ground and the considered height threshold, including small fences, cars, road signs, and so on. This way, we further de-clutter the point cloud, leaving in it almost only 3D points associated with walls and high vertical planes, mostly representing the contours of buildings in the cloud.

To perform accurate scan matching with a map derived from OSM, as a third step, we then transform the point cloud into the map coordinate frame, using the estimated corrected odometry (meaning that it follows a previous optimization, to take into account possible adjustments).

Once transformed, the point cloud is projected from 3D to 2D, onto the ground plane $z = 0$. At this point in the pipeline, the point cloud obtained is already usable to perform scan matching, having a similar representation to the OSM map. Nevertheless, we want to further clean up the 2D cloud, so, as a last step, we perform some noise filtering and outlier removal, using a radius search algorithm based on a KD tree representation of the cloud.

The blue box of Figure 5.2 follows the procedure described above, showing the initial point cloud to the left and the processed buildings map to the right. Thanks to this pre-processing, we are able to obtain great data reduction, while simultaneously refining more and more an initial scan to better fit our alignment purposes. However, the iterative RANSAC proves to be a bottleneck in the pipeline, as it involves a continuous search in the initial cloud. We believe that this step can be interchanged with equivalent methods, which are beyond the scope of this thesis, e.g., deep learning-based 3D point cloud segmentation or advanced forms of 3D clustering.

5.1.4 OSM buildings map creation branch

Data downloaded from third-party mapping services such as OpenStreetMap do not come already in the form of point clouds. Instead, they are given in a specific data format and structure, typical of many systems outside the field of robotics. For this reason, it is mandatory to collect OSM data surrounding the robot position and process it, to create a suitable point cloud representation that allows fast and efficient scan-to-scan 2D matching.

OSM implements a conceptual data model of the physical world based on components called elements. There are three types of elements: nodes, ways, and relations. All elements can have one or more associated tags. A node represents a point on the surface of the Earth and it comprises at least an ID number and a pair of high-precision coordinates corresponding to the latitude and longitude of the point. A way is an ordered list of nodes that define a poly-line. It can be used to represent linear features, such as roads or rivers, or the boundaries of areas and polygons (closed way).

A relation is a multi-purpose data structure that tells the relationship between two or more data elements, being an ordered list of nodes, ways, and other relations. Lastly, a tag describes the meaning of the particular element to which it is attached. It is made up of two fields: a key and a value, both represented as strings of characters. The key describes the meaning of the tag, such as “highway", and it is unique. The value is the description of the key, such as “residential", and it gives more detailed information.

It is possible to access and download map data from the OSM dataset in many ways. The most convenient, and suitable for the purpose of our work, is the Overpass API. It is a read-only API that allows accessing parts of the OSM map data selected in a custom way, given search location (latitude and longitude), radius, and optional filters (e.g., which elements to search for). It allows the client to send a query through an HTTP GET request to the API server, which will send back the dataset that resulted from the query.

There exist two languages in which to write a query: Overpass XML or Overpass QL. The Overpass QL syntax is more concise than Overpass XML and is similar to C-like programming languages. On the other hand, the Overpass XML syntax is safeguarded, because it uses more explicit named parameters than QL. We chose the QL language because it is easier to use. The response can also be in different formats, such as OSM XML, OSM JSON, custom templates, and pretty HTML output. In this case, we chose to get the data in the OSM XML format, as it can be parsed in an efficient way.

Figure 5.3 shows an XML response, following a query to the Overpass API. Aside from tags of the header, other elements inside the response are nodes and ways, as described above. In particular, inside the way, *nd* elements are specified, whose attribute *ref* coincides with the id of a node belonging to it. As the tags suggest, the object represented by the way in the response is a building, as we are interested in such structures to later align them with LiDAR scans associated with the estimated keyframes.

As a first step, to query the Overpass API and find the buildings surrounding the robot, we need a position to center the search of buildings. In fact, the query has the following structure: *(way["building"](around: radius,*

```
<?xml version="1.0" encoding="UTF-8"?>
<osm version="0.6" generator="Overpass API 0.7.56.9 76e5016d">
<note>The data included in this document is from www.openstreetmap.org.
    The data is made available under ODbL.</note>
<meta osm_base="2021-06-03T11:55:28Z"/>
  <node id="1205619743" lat="48.9850817" lon="8.3935543"/>
  <node id="1205619817" lat="48.9852221" lon="8.3935237"/>
  <node id="1205619851" lat="48.9851845" lon="8.3936350"/>
  <node id="1205619875" lat="48.9851194" lon="8.3934430"/>
  <way id="104492674">
    <nd ref="1205619743"/>
    <nd ref="1205619875"/>
    <nd ref="1205619817"/>
    <nd ref="1205619851"/>
    <nd ref="1205619743"/>
    <tag k="addr:city" v="Karlsruhe"/>
    <tag k="addr:housenumber" v="14"/>
    <tag k="addr:postcode" v="76199"/>
    <tag k="addr:street" v="Mainstrasse"/>
    <tag k="building" v="yes"/>
    <tag k="source" v="LA-KA"/>
  </way>
</osm>
```

Figure 5.3: *Example of an OSM XML response, following a query to the Overpass API. The structure described in this response is a building, as can be seen by its tags.*

latitude, longitude);); (._;>;); out;, which retrieves all buildings annotated in a certain area, specified by a radius centered in a given position.

As we have seen before, the geographic position in longitude and latitude coordinates is required. GNSS data is not always available, or even reliable, as it may happen in complex scenarios (due to errors caused by the Urban Canyon effect) or tunnels. For this reason, the pose, estimated through tracking, associated with the considered keyframe is used as the geographic location to query the Overpass API and download buildings.

The pose used is the same as mentioned in Section 5.1.3 when converting the LiDAR scan from local to map coordinate frame. Nevertheless, at least one GNSS datum is required to give a rough estimate of where, in the world, the robot is located. This value serves as the origin of the map, and it is used to convert both robot-estimated poses and buildings of OSM from Longitude-Latitude-Altitude to East-North-Up (ENU) coordinates.

From this query, we are able to obtain a list of buildings surrounding the estimated pose, of the keyframe, in the format described above. This response is then parsed, to form a point cloud for each building, containing the 2D points representing its corners, referenced w.r.t. the GNSS origin (meaning that the corners are expressed in ENU coordinates, easier to work with). Given the corners, for each building, we interpolate its edges, generating 100 to 1000 points for each one, obtaining a structured and refined

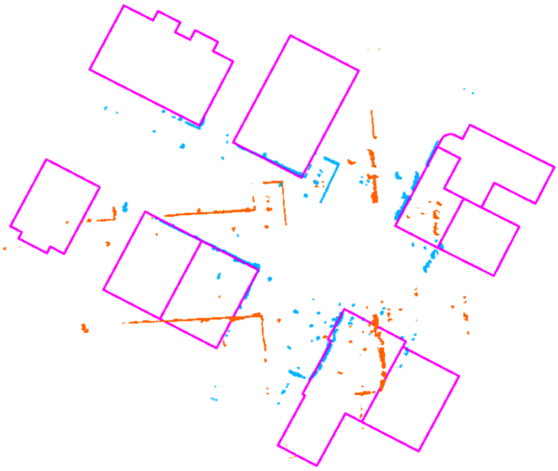


Figure 5.4: *Example of alignment between the 2D buildings map from OSM (pink) and the 2D map extracted from the LiDAR scan (orange). The aligned point cloud (blue) correctly overlaps the two maps, correcting the estimated trajectory of the robot.*

2D cloud. Lastly, all the point clouds associated with the various buildings are merged, to form an accurate 2D map of the buildings surrounding the robot. This map is now ready to be matched against the map computed in the point cloud processing branch, previously described in Section 5.1.3.

5.1.5 Data association - Rigid SLAM

Following the procedures described in Section 5.1.3 and Section 5.1.4, OSM-SLAM is able to obtain, starting from the pose associated with a keyframe and the corresponding point cloud, a rough 2D map that contains the visible edges of the buildings surrounding the robot and a refined 2D map formed by the contours of all buildings, derived from OSM data collected at the position of the robot in that time instant and location.

These two point clouds are matched using the ICP algorithm, fine-tuned to obtain very accurate alignments. Figure 5.4 shows the result of this alignment: the 2D map extracted from the LiDAR scan (orange) is matched against the OSM map (pink), obtaining an accurate transformed point cloud (blue, clearly overlapping the map derived from OSM, as it should). Now that the transformation is computed, we need to add this constraint to the pose graph. Figure 5.5 can be used to understand the insertion of new elements in the pose graph, including buildings, described as follows.

Given a building B_i, we always set its reference frame in the corner with the lowest x value, in ENU coordinates. $P_j^{B_i}$ is the 2D pose of the said

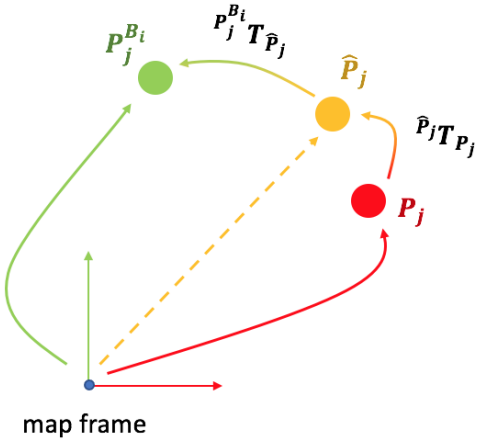


Figure 5.5: *First, we find the correct pose of the robot $\hat{P}_j$, w.r.t. OSM buildings map. Then, we compute the displacement between $\hat{P}_j$ and the pose of the building, $P_j^{B_i}$.*

corner when associated with keyframe j (the same building can be seen by multiple keyframes, but it has only one associated pose, i.e., the one that best satisfies all pose graph constraints). Building B_i is inserted in the pose graph as a node, and its pose corresponds to $P_j^{B_i}$. In particular, the translation part coincides with the associated ENU coordinates of the reference corner, while the rotation is the 2-by-2 identity matrix, meaning that the local frame of the building is oriented as the ENU frame.

Given the estimated pose of the robot P_j, again related to keyframe j, to which correspond a node in the pose graph, we would like to associate it with the building node B_i, through some sort of measurement. Let $\hat{P}_j$ be the correct pose of the robot w.r.t. the OSM map. Ideally, $P_j = \hat{P}_j$, which means that the current estimated pose of the robot is also the correct pose w.r.t. OSM. In practice, this does not happen, and the scan matching algorithm would return a rigid transformation between the buildings map and the processed LiDAR scan. We name this transformation ${}^{\hat{P}_j}T_{P_j}$, as it moves the estimated pose towards the correct one, such that:

$$\hat{P}_j = {}^{\hat{P}_j} T_{P_j} * P_j. \tag{5.1}$$

Now that we have an adjusted value of the pose of the robot w.r.t. the buildings map of OSM, we are able to find the rigid motion between the corrected keyframe and the pose of the associated building B_i, computed as:

$${}^{P_j^{B_i}}T_{\hat{P}_j} = P_j^{B_i} * \hat{P}_j^{-1}. \tag{5.2}$$

This transformation suggests how rotated and translated are, w.r.t. each other, the pose of the building (corner), and the estimated pose of the robot. The constraint is inserted into the pose graph as an edge between the two corresponding nodes and is later used in the optimization phase to adjust and correct the poses associated with the nodes. To better understand this concept, one can think, again, at the ideal case, where $P_j = \hat{P}_j$.

What happens in the pose graph is that nodes and constraints already satisfy an equality with no measurement error, as there is no displacement between the estimated and correct pose (same by assumption). However, this scenario does not happen, and the given measurement yields some information about the true displacement between the robot pose and the building pose, which is later satisfied and corrected by the optimizer. In other words, we try to align the LiDAR scan with the contour of the building.

The whole procedure takes the name of Rigid SLAM, as it always estimates a rigid motion between the considered keyframe and the buildings, considered as a whole, surrounding the associated robot pose. When inserted in the pose graph, all buildings are treated equally, as the same transformation ${}^{\hat{P}_j}T_{P_j}$ is used to find the measurements needed to characterize the constraint, as previously described. Nevertheless, consecutive keyframes may lead to different transformations for the same building, making the Rigid SLAM approach only “rigid" when considered w.r.t. a single keyframe.

Moreover, one can decide to fix the poses of the buildings, if it has the prior knowledge that the OSM map is mostly precise and correctly annotated. In this case, which takes the name of Prior SLAM, the poses associated with the corners of buildings are not modified during the optimization procedure. However, their influence on the corresponding keyframe is considered in the same way as described for Rigid SLAM, as all surrounding buildings are associated with it by transformation ${}^{P_j^{B_i}}T_{\hat{P}_j}$.

5.1.6 Data association - Non-rigid SLAM

As we have seen in Rigid SLAM, a single global alignment is performed to associate the pose of a keyframe and the corresponding map of the buildings surrounding it, downloaded from OSM. Although simple to implement and relatively accurate, this approach may lead to some problems.

When multiple keyframes see the same building in almost the same location, even if the building is wrongly positioned w.r.t. the real world, its pose would never be modified. In other words, the Rigid SLAM method is able to move and correct buildings but is not able to tell whether a building is in the wrong or correct location in the real world, leading also to mistakes

in the optimization procedure, e.g., incoherence between the relationships keyframe - buildings across multiple, consecutive, keyframes.

To address this issue, once the global alignment is computed, each building is taken individually, to repeat the same procedure described in Section 5.1.5, with few differences. Before, we tried to align the LiDAR scan, associated with keyframe j, against the point cloud that represents all the contours of the building surrounding the robot at that precise location.

This alignment resulted in transformation ${}^{P_j^{B_i}}T_{\hat{P}_j}$, which is the same for all buildings. In other words, let $i \in \{k, k+1, \ldots, k+s\}$ be all the buildings in the OSM map surrounding the robot whose estimated pose is described by keyframe j. In Rigid SLAM, we obtain the same transformation, i.e.,

$$ {}^{P_j^{B_k}}T_{\hat{P}_j} = {}^{P_j^{B_{k+1}}}T_{\hat{P}_j} = \ldots = {}^{P_j^{B_{k+s}}}T_{\hat{P}_j}, i \in \{k, k+1, \ldots, k+s\}. \quad (5.3) $$

In Non-rigid SLAM, after this procedure, each building is considered separately, being formed by a relatively low number of 2D points. Then, scan matching is performed between the LiDAR scan and the point cloud of the single building. This alignment is aided using transformation ${}^{P_j^{B_i}}T_{\hat{P}_j}$ as an initial guess, to facilitate it and boost performance. The resulting motion ${}^{P_j^{B_i}}\hat{T}_{\hat{P}_j}$ directly tells the correct pose of the building w.r.t. the keyframe, and it is independent of the transformations derived from the other buildings.

Let us consider again buildings $i \in \{k, k+1, \ldots, k+s\}$, located around the robot w.r.t. keyframe j. From Rigid SLAM, we are able to align the LiDAR scan of the keyframe with the clouds formed by all 2D buildings, resulting in a global transformation ${}^{P_j^{B_i}}T_{\hat{P}_j}$. It should be noticed that this relative motion is the same for all entities since they are considered jointly.

In Non-rigid SLAM, the buildings are then considered separately one from the other and are once again aligned with the point cloud of the same keyframe as before, using the global transformation as an initial guess. This way, for each building we can obtain a local transformation ${}^{P_j^{B_i}}\hat{T}_{\hat{P}_j}$, which is more precise and captures the true displacement between the 2D structure and the LiDAR scan, since it will be used later in the optimization step.

Differently from Rigid SLAM, each building is treated as a standalone element in the pose graph, associated with at least one keyframe by a unique transformation. With Non-rigid SLAM, we are able to detect errors present in the OSM map by using LiDAR scans, which directly model the environment surrounding the robot. From a local alignment, in fact, we are able to check for discrepancies between the OSM map and the map built using SLAM, possibly caused by annotation issues or due to a large time

gap between LiDAR acquisitions and the creation of the OSM outline.

5.1.7 Experimental validation of the system

We compare the proposed system with different methods for LiDAR SLAM: LOAM [30], LeGO-LOAM [32], LIO-SAM [115], HDL [28] and the baseline of our work, ART-SLAM [1] (without Scan Context). In particular, we evaluate all three variants of OSM-SLAM. First, we consider Prior SLAM, which follows the same procedure as in Rigid SLAM (described at the end of Section 5.1.5), in which the nodes in the pose graph associated with buildings are fixed and therefore cannot be modified by the optimization procedure. Then, we evaluate both Rigid and Non-rigid SLAM approaches, to see their differences and behaviors when dealing with long trajectories.

Lastly, we present a re-localization experiment on one of the sequences used for testing, where we stop the tracking for about 100 consecutive initial frames, and use OSM buildings to estimate the missing odometry and evaluate the re-localization possibilities using external mapping services.

We evaluate all systems on Sequence 07 and Sequence 00 of the KITTI odometry dataset [143], as they correspond to trajectories with medium and high complexities, respectively, and the associated OSM maps are sufficiently detailed (but not necessarily correct) for testing. Experiments are done on a 2021 XMG 64-bit laptop with Intel(R) Core(TM) i7-11800H CPU @ 2.30GHz x 8 cores, with 24576 KB of cache size.

5.1.8 Comparison and results

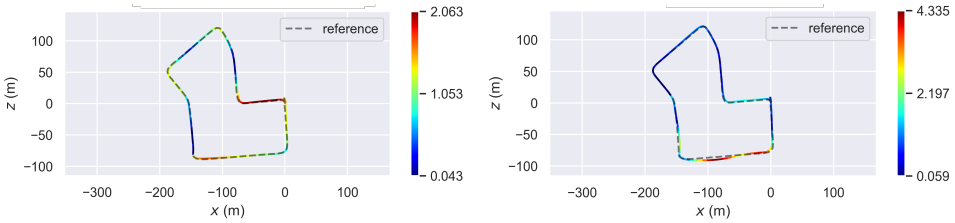
The goodness of the methods used for the comparison is computed by means of absolute trajectory error (ATE), as done in the previous chapters and for the other developed systems of our SLAM framework. We also include here a detailed visual evaluation of the estimated trajectory and placement of the buildings in the global map, overlapped with OSM tiles.

Table 5.1 details the obtained results (ATE statistics, in meters) on Sequence 07 of the KITTI odometry dataset [143]. Looking at the table, one may infer that the overall accuracy of the proposed system, in all three approaches, decreased w.r.t. the baseline, even if by an acceptable amount when compared with all other methods. However, using only these statistics does not explain in detail how the proposed methods truly behave.

Figure 5.6 shows the distribution of the ATE over the whole trajectory. In particular, Figure 5.6a refers to the results obtained with Prior SLAM, while Figure 5.6b represents the trajectory estimated with Non-rigid SLAM. It is clear how, especially for the latter case, the majority of the estimates prove

Table 5.1: *Computed ATE on Sequence 07 of the KITTI odometry dataset [143].*

ATE [m]	MEAN	RMSE	Standard Deviation
LOAM	> 10	> 10	>10
LeGO-LOAM	1.191	1.309	0.546
LIO-SAM	0.509	0.675	0.351
HDL	0.954	1.253	0.767
ART-SLAM (baseline)	0.698	0.777	0.341
Prior SLAM (ours)	1.084	1.198	0.510
Rigid SLAM (ours)	3.253	3.420	1.057
Non-rigid SLAM (ours)	0.912	1.157	0.485



(a) *Error distribution (Prior SLAM).* **(b)** *Error distribution (Non-rigid SLAM).*

Figure 5.6: *Local accuracy of two proposed approaches (Prior SLAM on the left, Non-rigid SLAM on the right) on Sequence 07 of the KITTI odometry dataset [143].*

to be definitely more accurate than the baseline, reaching error values below half a meter. Other sections, however, heavily influence the overall ATE mean value, possibly because of mistakes in the corresponding area of the OSM map or time discrepancies w.r.t. the acquisition date and when OSM annotations took place (OSM maps are usually more recent than KITTI).

These results also give us more insight into the proposed systems. In the case of Prior SLAM, where buildings are fixed, the constraints between buildings and keyframes are completely weighted on the keyframes themselves, leading to an overall distribution of the trajectory error. On the other hand, in Non-rigid SLAM, all buildings, independently one from the other, are free to move and adjust their location in the map, jointly with keyframes.

This means that the trajectory error will be majorly concentrated in areas with possible issues. Lastly, one can also see the optimization effects on the buildings in Figure 5.7, which shows two detailed areas of the 3D reconstruction obtained with Non-rigid SLAM, overlapped to OSM tiles. Elements are slightly moved (blue dots), w.r.t. their original position (red shape), to satisfy all constraints explained in Section 5.1.6.

As the last experiment on this sequence, we tried to evaluate the re-

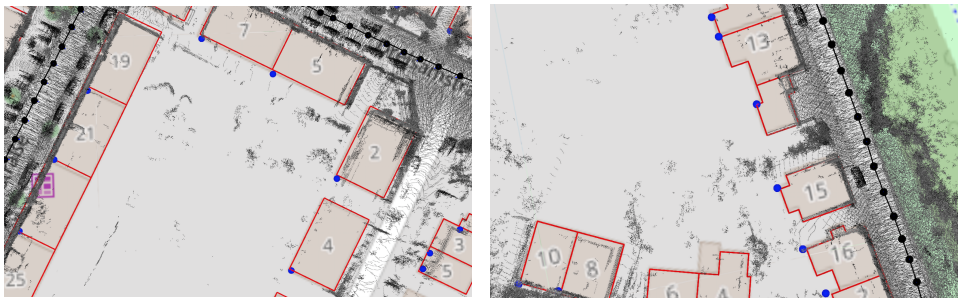


Figure 5.7: *Details of the reconstruction with Non-rigid SLAM on Sequence 07. Notice the displacement of the buildings (blue dots) w.r.t. their original location (red shape).*

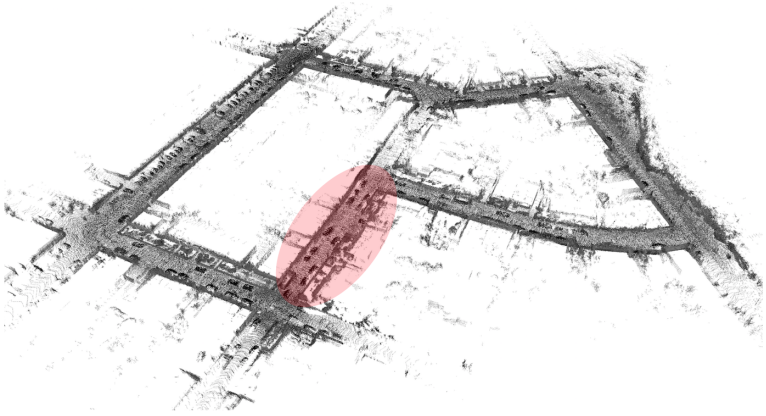


Figure 5.8: *3D map obtained through the re-localization experiment. The red zone consists of frames that are not used for tracking, and for which the odometry is computed through OSM map alignment.*

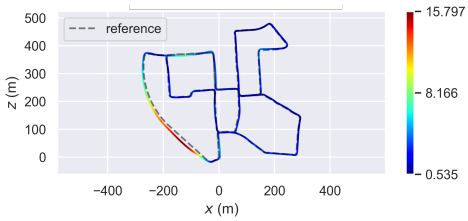
localization capabilities of OSM-SLAM. This was achieved by simulating a faulty tracking, for about 100 consecutive frames, right at the beginning of the trajectory, as depicted in Figure 5.8. Then, we used the available OSM maps to re-localize the robot within it (using the same scan matching procedure described for both Rigid SLAM and Non-rigid SLAM) estimating the relative motion of consecutive LiDAR scans.

While the baseline of the system would completely fail, OSM-SLAM works as intended, after re-localization, with a small loss in accuracy (a few centimeters worse than ART-SLAM). This proves that re-localization can be performed through means of 2D maps coming from mapping services.

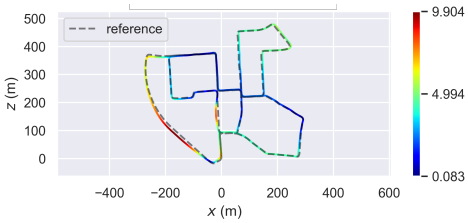
We then moved to Sequence 00 of the same dataset. Table 5.2 contains the ATE statistic. As in the previous case, the accuracy of OSM-SLAM

Table 5.2: *Computed ATE on Sequence 00 of the KITTI odometry dataset [143].*

ATE [m]	MEAN	RMSE	Standard Deviation
LOAM	> 10	> 10	>10
LeGO-LOAM	9.537	11.666	6.718
LIO-SAM	> 10	> 10	>10
HDL	1.378	1.424	0.779
ART-SLAM (baseline)	0.981	1.092	0.478
Prior SLAM (ours)	3.802	4.778	2.893
Rigid SLAM (ours)	4.064	4.494	1.918
Non-rigid SLAM (ours)	3.648	4.214	2.110



(a) *Error distribution (Prior SLAM).*



(b) *Error distribution (Non-rigid SLAM).*

Figure 5.9: *Localization accuracy of two proposed approaches (Prior SLAM on the left, Non-rigid SLAM on the right) on Sequence 07 of the KITTI odometry dataset [143].*

seems to be worse than the baseline. Remembering the same reasoning done for Sequence 07, one should also look at the trajectory error distribution, to effectively understand the impact of OSM maps. Figure 5.9 represents the distribution of the ATE over the whole trajectory, with Figure 5.9a referring to Prior SLAM, and Figure 5.9b showing the results of Non-rigid SLAM.

In the first case, most of the estimated trajectory has a low error, almost half of the baseline, whereas in a small area (curved road on the bottom right side) we see a noticeable drift that influences the mean error. On the other hand, in the Non-rigid SLAM scenario, we can clearly see that the error is spread over the whole trajectory, reaching a lowest value of less than 10 centimeters. This is a remarkable result, considering that we are dealing with a large and complex map, containing multiple sharp turns and loops.

CHAPTER 6

D3VIL-SLAM: 3D Visual Inertial LiDAR SLAM

With MCS-SLAM [3], described in Chapter 4, we demonstrated how true sensor fusion still remains a hard task in SLAM systems, mainly due to the difference between data coming from different sensors. Although speed is easily achievable, accuracy on par with state-of-the-art methods performing single-sensor localization and mapping is still an open issue.

Many systems in literature, in fact, adopt a multi-sensor-based direct data association, using heterogeneous sensor data such that one type is integrated to aid another (e.g., depth from lasers to support visual SLAM).

To enhance the framework proposed in this thesis also with a method able to perform accurate and real-time multi-sensor SLAM with direct data association, we developed a novel 3D visual inertial LiDAR SLAM system, named D3VIL-SLAM, described in the following section.

6.1 D3VIL-SLAM

As we have already demonstrated in Chapter 3 to Chapter 5, LiDAR-based SLAM systems are more robust w.r.t. other methods, even though additional information, like visual odometry, could improve the accuracy of the trajectories, via direct data association (differently from the system we developed

in Chapter 4, MCS-SLAM [3]). Moreover, using visual, LiDAR, and inertial measurements, would make a system more resilient in different scenarios, having the various motion estimates compensate for each other.

To address this situation, and improve the overall precision of our framework for SLAM developed in our Ph.D. program, we designed a 3D Visual Inertial LiDAR SLAM system, named D3VIL-SLAM, as a further extension of ART-SLAM [1]. In particular, D3VIL-SLAM is equipped with three main branches to perform motion tracking: a *laser front-end*, relying on scan-to-keyframe matching to estimate the pose of the robot, as in ART-SLAM; a *vision front-end*, which exploits consecutive pairs of stereo images to identify 3D points in the world and uses them to compute the robot motion by re-projection error minimization; and a *filter front-end*, where IMU data is given to an Error State Kalman Filter [168] to predict the robot location. The three branches can be either independent or collaborate with each other, e.g., having the filter front-end pass its predictions as initial tracking guesses for the laser front-end and vision front-end.

Loops in the trajectory are then detected using two, independent, algorithms. The first one finds pairs of point clouds that best overlap, while the second method tries to find similar locations by matching descriptors associated with 3D salient points, extracted from stereo images. Front-end and loop closure information are collected and used to build a g2o [139] pose graph, which is periodically optimized to satisfy all constraints.

We first discuss related works in Section 6.1.1, to give a brief insight into existing hybrid SLAM systems. Then, after a high-view description of D3VIL-SLAM architecture, in Section 6.1.2, we explain the three branches forming its front-end: the laser branch in Section 6.1.3; the vision path in Section 6.1.4; and the inertial-based front-end in Section 6.1.5. Follow Section 6.1.6 and Section 6.1.7, in which we describe how loop detection and pose graph building and optimization are performed, respectively. Lastly, Section 6.1.8, Section 6.1.9 and Section 6.1.10 are dedicated to the experimental validation of D3VIL-SLAM, including also an ablation study.

6.1.1 Related works

Despite being the most accurate type of SLAM, LiDAR-based localization and mapping needs to overcome the problems of large computation and possible motion distortion. To do so, different visual LiDAR SLAM systems have been proposed in the last decade. Visual LiDAR SLAM [169], abbreviated as VL-SLAM, is composed of a LiDAR odometry estimator, based on scan matching, a pose graph optimization back-end, and a set of

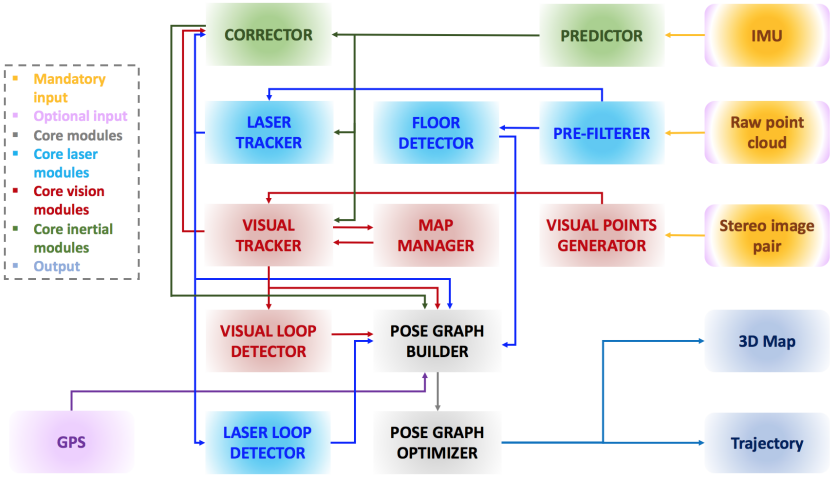


Figure 6.1: *Architecture of the proposed 3D Visual Inertial LiDAR system, D3VIL-SLAM.*

loop detection modules relying on visual recognition only.

DVL-SLAM [112] takes as input a sequence of images and LiDAR scans. Given an image with an associated sparse depth (range measurements from the LiDAR), only the image is used for the tracking process. The front-end focuses on accurate tracking using data association for loop-closing, and the associated data is then used for efficient pose graph optimization.

Recently, DV-LOAM [170] has been proposed, consisting of three modules. First, a two-staged direct visual odometry module estimates the pose of the camera. Then, every time a keyframe is generated, in the usual way already described, a LiDAR mapping module is utilized to refine the pose of the keyframe to obtain better robustness. Finally, a Parallel Global and Local Search Loop Closure Detection module that combines visual Bag of Words and LiDAR-Iris feature is applied for place recognition.

In contrast to all these approaches, our system is able to obtain multiple, independent, estimates of the position of the robot (one for each front-end), which are later combined in a pose graph and jointly optimized, avoiding the hard task of accurately and efficiently fusing heterogeneous data.

6.1.2 System overview

An overview of the proposed method is represented in Figure 6.1 (in the same way it was done for all other systems presented in this thesis) and Figure 6.2, which is more detailed. The system front-end, associated with the tracking procedure, consists of three branches for motion estimation.

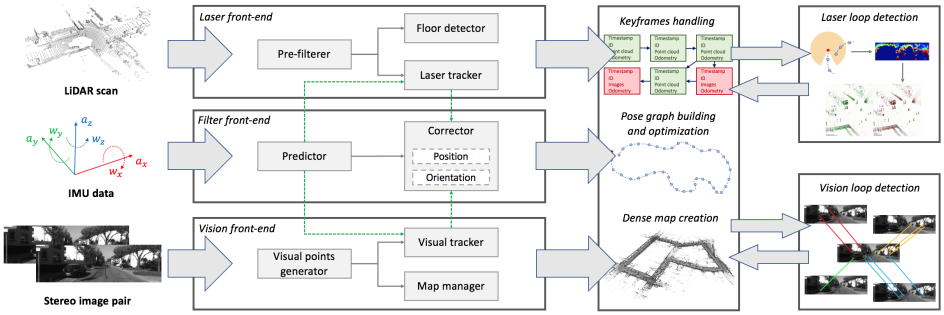


Figure 6.2: *In-depth architecture of D3VIL-SLAM, represented in a graphical way.*

The *laser front-end* comes from ART-SLAM. It takes as input a scan and processes it, to reduce its size. The filtered cloud is then given to two independent modules. The floor detector finds the robot pose w.r.t. the ground, enforcing height and rotational constraints. The tracker, instead, estimates the motion of the robot, by performing scan-to-keyframe matching.

The *vision front-end* uses a pair of stereo images as input, and as a first step, it generates 3D visual points from them, via triangulation. The visual points in the map are then matched and tracked from the previous pair of stereo images, to find enough associations to compute the current pose.

The *filter front-end* is the last branch of the proposed system tracking procedure. Through an Error State Kalman Filter, IMU data is processed and used to accurately predict the motion of the robot. This prediction can be used as tracking guess in all other front-ends, to increase their accuracy and boost their performance. Conversely, the poses estimated in the vision and laser front-ends can be used by the filter to perform a correction step.

The current pose estimate is sent, along with its corresponding data (images or point clouds), to the *loop detector* modules, which try to find loops using different techniques: efficient scan-to-scan matching, in case of point clouds (as in ART-SLAM), and similarity between descriptors, when considering stereo image pairs. Poses, loops, and floor coefficients are used to build a pose graph, which is optimized to satisfy all motion constraints.

6.1.3 Laser front-end

The laser front-end shares the same modules of ART-SLAM [1], since the proposed system builds on it. As a first step, input point clouds are fed to the *pre-filterer* module, which downsamples them and removes outliers.

The filtered scans are then simultaneously sent to both floor detector and tracker. The *floor detector* module tries to find the ground plane,i.e.,

the plane with equation $a * x + b * y + c * z + d = 0$, in the given scans, to enforce height and rotational consistency in the trajectory. To avoid searching a whole cloud, the floor detector clips it and removes all points having orthogonal normal w.r.t. the ground direction.

Working in parallel to the floor detector, the *tracker* uses the filtered clouds to perform short term data association. The tracker adopts a keyframe-based approach to estimate the trajectory, meaning that not all inputs are used to create keyframes. While the first received scan will always be part of a keyframe, a new one is generated only if the current cloud is far enough from the previous keyframe, both spatially and temporally.

6.1.4 Vision front-end

The goal of the vision path is to correctly estimate the motion of the robot, given consecutive pairs of left and right stereo images. This is achieved thanks to three modules: visual points generator, tracker, and map manager.

As the name implies, the *visual points generator* creates visual points (i.e., 3D points derived by image processing) following a series of consecutive steps. First, left and right keypoints, which are salient image features, are detected in the stereo image pair. For each keypoint, the corresponding descriptor is then extracted, and directly associated to it. In the current implementation of our system, we use a FAST keypoints detector and ORB descriptors. Together, keypoints and descriptors form a visual feature.

Instead of directly matching the descriptors of left and right visual points by brute force, the visual points generator performs a fast epipolar search, motivated by the fact that being the images belonging to a stereo pair, correspondences should lie more or less on the same image row and that a visual point in the right image should be on the same or lesser column w.r.t. its counterpart in the left image (being the images in stereo configuration).

Once all correspondences between left and right visual features have been found, they are triangulated (using the stereo camera baseline as a given value), to obtain the coordinates of the associated point in the real world. 3D position, left and right visual features are combined to form a visual point. Together, all detected visual points are grouped into a visual frame, which also contains information about the estimated pose of the robot.

Consecutive visual frames are used by the *tracker* module, to estimate the motion that the left camera went through to get from the previous to the current position. As a first step, visual points of the previous visual frame are projected on the current left image (using a motion guess), generating keypoints predictions. These elements are then compared with the keypoints

detected as described in the previous paragraphs.

If the predicted and true keypoints have similar descriptors, then the corresponding previous visual point is a candidate for tracking. In case of a successful match, the same epipolar search of the visual points generator is followed, to find a corresponding visual feature in the right image. If the search gives a positive result, the visual point of the previous visual frame is said to have been successfully tracked in a visual point of the current frame.

All tracked visual points are then used to estimate the motion of the robot. The problem can be solved via minimization of the re-projection error:

$$error = \sum_{i=0}^{\#VP} ||f(VP_{i,t-1}) - VF_{i,t}||^2, \tag{6.1}$$

where $f(VP_{i,t-1})$ is the re-projection of the previous i^{th} visual point in the current left image and $VF_{i,t}$ is the corresponding visual feature in it.

Instead of solving this problem, it is re-formulated as a linear system, where, rather than finding the transformation X^* that yields the best result, a perturbation Δx is calculated iteratively, starting from an initial guess X (e.g., obtained under the assumption of constant velocity).

Once the motion of the robot has been estimated, and, consequently, its position in the world, the last step of the vision front-end is to create a sparse map using the available 3D visual points. Not all elements are used to create a map, but only the ones tracked along a certain number of consecutive visual frames. These points are also known as visual landmarks, as they represent stable elements that have been tracked along multiple images.

The *map manager module* takes all estimated visual landmarks and generates multiple structures. First, it determines when to create a local map, which is a collection of visual frames, along with the corresponding landmarks. As in the laser front-end, also here keyframes are generated. In particular, a visual keyframe contains a local map and the corresponding position of the robot (i.e., the position associated with the last visual frame).

The keyframe generation procedure follows the same rules as stated in Section 6.1.3. Let VF_i be the last visual frame of the local map associated to keyframe K_n. If the current visual frame VF_{i+k} is displaced enough from VF_i, then a new local map is created, using all visual frames from VF_{i+1} to VF_{i+k}, and it is associated to a new keyframe K_{n+1}.

6.1.5 Filter front-end

A standard approach in SLAM is to support the front-end with a Kalman Filter, which employs IMU data to predict the position of the robot. This

prediction is then used as an initial guess in the tracking step, and the estimated odometry serves as correction data for the filter. In our system, we also introduce an optional Kalman Filter-based module, to continuously predict the motion of the robot. In particular, D3VIL-SLAM adopts an Error State Kalman Filter (ESKF), which has many advantages w.r.t. other methods [168]. The main benefit of this choice is that complex dynamic modeling of the robot and its interaction with the environment is avoided.

An ESKF consists of true, nominal, and error-state values, with the first being a suitable composition of the others. Input IMU data is integrated into a nominal state, which does not take into account imperfections, accumulating errors. These are, instead, collected in the error state, which can be defined by a time-variant linear system, with its dynamic control and measurement matrices derived from the nominal state.

In the filter front-end of our system, the nominal state is a vector of 16 elements, which represent the following estimated quantities: 3D position of the robot P_t, velocities V_t, orientation Q_t (expressed as a quaternion), acceleration bias AB_t and angular velocity bias WB_t, where t is the current time step. The error-state is a vector of 15 elements (because, for convenience, the rotation is expressed in Euler angles), representing the differences w.r.t. the nominal-state: δP_t, δV_t, $\delta\theta_t$, δAB_t and δWB_t.

The equations of the nominal state follow simple kinematics, given input acceleration a_m and angular velocities w_m:

$$\begin{aligned} P_{t+1} &= P_t + V_t\Delta t + 0.5(R_t(a_m - AB_t) + g)\Delta t^2, \\ V_{t+1} &= V_t + (R_t(a_m - AB_t) + g)\Delta t, \\ Q_{t+1} &= Q_t \otimes Q\{(w_m - WB_t)\Delta t\}, \\ AB_{t+1} &= AB_t, \\ WB_{t+1} &= WB_t, \end{aligned} \tag{6.2}$$

with g being the gravity vector, R_t the rotation matrix associated with Q_t, and Δt the time difference between the current and previous IMU measurements. The nominal, deterministic, part is integrated normally, while the integration of the error, stochastic, component is:

$$\begin{aligned} \delta P_{t+1} &= \delta P_t + \delta V_t\delta t, \\ \delta v_{t+1} &= \delta v_t - R_t([a_m - AB_t]_\times\delta\theta_t + \delta AB_t)\Delta t + v_i, \\ \delta\theta_{t+1} &= R^T\{(w_m - WB_t)\Delta t\}\delta\theta_t + \delta WB_t\Delta t + \theta_i, \\ \delta AB_{t+1} &= \delta AB_t + a_i, \\ \delta WB_{t+1} &= \delta WB_t + w_i, \end{aligned} \tag{6.3}$$

where v_i, θ_i, a_i, and w_i are random perturbations applied to velocity, orien-

tation, and bias estimates, respectively. In the case of the error state, we also need to compute the covariance matrix C.

Let F_x and F_i be, respectively, the Jacobians of the error-state dynamics function w.r.t. the error and perturbation vectors. Let also be Q_i be the composition of the covariance matrices of the perturbations. Then, the error-state prediction can be fully written in the following way:

$$\begin{aligned} \delta state_{t+1} &= F_x(state_t, \{a_m, w_m\})\delta state_t, \\ C_{t+1} &= F_x C_t F_x^T + F_i Q_i F_i^T. \end{aligned} \tag{6.4}$$

When data is received from the laser or vision front-ends, the ESKF performs a correction step. Let H be the Jacobian matrix, defined w.r.t. the error-state and evaluated at the best true-state estimate; the correction equations are:

$$\begin{aligned} K &= C_{t+1}H^T(HC_{t+1}H^T + V)^{-1}, \\ error_{t+1} &= K(y - h(nominal_{t+1})), \\ C_{t+1} &= (I - KH)C_{t+1}(I - KH)^T + KVK^T, \end{aligned} \tag{6.5}$$

where I is the identity matrix, y is the measurement, V is its covariance matrix, and h is the measurement function.

The corrected error state is then injected in the nominal state, by simple composition and summation (e.g., $P_{t+1} = P_{t+1} + \delta P_{t+1}$). Lastly, the error state is reset, by setting its mean to zero and updating the corresponding covariance matrix as $C_{t+1} = GC_{t+1}G^T$, with G being the Jacobian matrix of the reset function w.r.t. the error-state itself.

6.1.6 Loop detection

In our system, we decouple, as in the front-end, vision and laser approaches, adopting two different techniques, working in parallel, to detect possible loops in the trajectory and close them with matching techniques.

In D3VIL-SLAM, loop detection with point clouds is achieved through three consecutive phases, as in ART-SLAM. First, a new keyframe K_{query} is compared against all existing keyframes, using a spatial and temporal selection method. If K_{query} and a candidate $K_{candidate}$ have far enough timestamps and their associated positions are within a threshold range, both keyframes pass the first selection of candidate pairs for loop detection.

Scan-Context [140] is an algorithm that converts 3D scans into 2D polar grids and compares them using KD-tree search to get the most similar to a given point cloud query. Loop detection in the laser branch continues by using a slightly modified version of Scan-Context, by allowing the indices of the k-most similar point clouds w.r.t. K_{query} to be retrieved.

The corresponding point clouds are directly used to perform scan-to-scan matching, which tries to align them with the cloud associated with K_{query}. All the retrieved transformations are then filtered to obtain the one with the highest accuracy above a fixed threshold. If such transformation is found, then the corresponding pair, K_{query} and $K_{candidate}$, represents a loop.

To detect loop closures using input pairs of stereo images, instead, we exploit the fact that the front-end already extrapolates keypoints and descriptors, packing them together in visual features, visual points, and visual landmarks. As explained in Section 6.1.4, a visual keyframe contains a local map, which is a collection of visual frames and the corresponding landmarks. Intuitively, visual landmarks are a good representation of a small region in the world and they can be exploited to find possible loops.

The idea is that if a robot passes through a previously visited location, then the landmarks representing the current map will be similar to the visual landmarks of the local map associated with that location. To estimate how similar two local maps are, we use the Hamming binary search tree library (HBST [171]), which performs similarity search directly on descriptors. This allows the system to retrieve landmark-to-landmark matching pairs. All pairs of corresponding landmarks are then aligned using a linearized version of the ICP [13] algorithm. If the resulting transformation is acceptable and the number of ICP inliers is high enough, then the two local maps effectively form a loop, and their relative pose is given by the estimated transformation.

6.1.7 Pose graph building and optimization

As already mentioned in Section 6.1.2, D3VIL-SLAM is a form of graph SLAM [139], where the poses of the robot are modeled as nodes and spatial constraints between said poses are represented as edges. Each node in the graph, named pose graph, represents the pose of the robot and at least one sensor measurement (e.g., filtered clouds or local visual maps).

In D3VIL-SLAM, the pose graph construction does not follow fixed rules, and the graph can be built in different ways, depending on the scenario considered, the reliability of the sensors, and so on. Without loss of generality, we will describe the implementation used in our experimental campaigns. Two separate sub-graphs are incrementally created, one associating nodes to laser keyframes, and the other for nodes of visual keyframes. If two keyframes of different types correspond to more or less the same timestamp, it means that they represent the same pose in the world and they can be joined as a single keyframe, merging and constraining also the pose graphs.

Other than the constraints derived from the various front-ends, the pose

graph is enriched with other edges. As seen in Section 6.1.3, the floor detector finds the distance and orientation (roll and pitch) of the robot w.r.t. the ground plane, given an input LiDAR scan. This information is added to the pose graph, including also one fixed node representing the ground.

Visual keyframes hold the position in the world of the last visual frame added to the corresponding local map. The relative motion between two local maps, which is already added in the pose graph, does not directly involve visual point matching and tracking between the associated visual frames. To enforce a further constraint, we take all common visual landmarks available in the last visual frames of the two local maps. Then, if enough elements are available (at least 10), we estimate the essential matrix and retrieve the relative orientation between the considered visual frames, which is then added as an additional constraint in the visual pose graph.

Also, edges can be added when performing successful loop detection and closure (either using point clouds or image descriptors), between non-consecutive nodes in the graphs, forming a ring-like structure.

As a last step, the pose graph is given to an optimization algorithm, to estimate the trajectory that best satisfies all constraints induced by the existing edges. To build and optimize the graph, we use the g2o framework [141], as it proves to be fast, accurate over long trajectories, and easily customizable.

We briefly mention another possible design for the pose graph construction. One could decide to use the corrected poses from the ESKF as main nodes, which are later constrained by the odometry estimations coming from both vision and laser tracking branches, separately, obtaining a single graph with two or more edges connected to each node (from the inertial branch).

6.1.8 Experimental validation of the system

The developed system D3VIL-SLAM is evaluated against other methods for SLAM (not odometry, for a fair comparison), both laser-based (LeGO-LOAM [32], LIO-SAM [115], HDL [28] and ART-SLAM [1] with Scan-Context) and vision-based (ORB-SLAM2 [53] and ORB-SLAM3 [75]). To measure the accuracy of the methods, we use, as we have always done in the thesis, Sequence 00 and Sequence 07 of the KITTI odometry dataset [143] and short, loopless, City Sequence 05 of the KITTI raw dataset [144].

We evaluate the implementation of our system described in Section 6.1.7, which proved to be more accurate in the considered scenarios. Here, the ESKF is used to support only the laser front-end, and its estimates are added as additional constraints in the pose graph, coupling them with edges associating visual keyframes only when representing the same location in the

Table 6.1: *ATE of the compared systems on three sequences of the KITTI dataset [143,144].*

ATE [m]	Sequence 00		Sequence 07		City Sequence 05	
	MEAN	STD	MEAN	STD	MEAN	STD
LeGO-LOAM	9.537	6.718	1.191	0.546	0.707	0.300
LIO-SAM	>10	>10	0.509	0.351	**0.493**	**0.280**
HDL	1.378	0.779	0.954	0.767	0.893	0.476
ART-SLAM (SC)	1.232	0.684	0.730	0.358	0.742	0.331
ORB-SLAM2	1.639	0.818	1.189	0.585	5.136	3.366
ORB-SLAM3	1.464	0.817	1.310	0.785	5.174	3.376
D3VIL-SLAM (proposed)	**1.017**	**0.502**	**0.470**	**0.165**	0.745	0.334

world (i.e., close timestamps). The reasons for this choice are that LiDAR odometry is more accurate than visual odometry, and IMU data is gathered at a low frequency (10 Hz), making the ESKF pose estimates unreliable.

To conclude the results, we perform an ablation study of the system, where laser odometry and loop detection are not used. In particular, we evaluate, using the same sequences, the proposed system in case of only vision, vision enhanced with essential matrix information, and vision with essential matrix and ESKF predictions as support in the pose graph.

Tests are done on a 2021 XMG 64-bit laptop with Intel(R) Core(TM) i7-11800H CPU @ 2.30GHz x 8 cores, 24576 KB of cache size, and using Ubuntu 20.04 LTS as OS, with ROS Noetic (many hybrid systems could not be tested because relying on incompatible previous versions of ROS).

6.1.9 Comparison and results

The metric used to compare the various systems is, once again, the absolute trajectory error (ATE), which measures the difference between the estimated trajectory and the corresponding positions of the ground truth.

Table 6.1 shows the mean and standard deviation (STD) of the absolute trajectory error, in meters, on all tested sequences. In particular, the first three columns refer to Sequence 00, which is one of the most complex. D3VIL-SLAM achieves the highest accuracy, demonstrating how the integration of vision and filter front-ends contributes positively to the overall precision. Of particular importance is the addition of visual loop closures, which, combined with LiDAR loop closures, strengthen the estimated trajectory by adding more information to the pose graph. Figure 6.3 supports the results by showing the trajectory error distribution on Sequence 00.

The same considerations can be applied to Sequence 07 (Table 6.1),

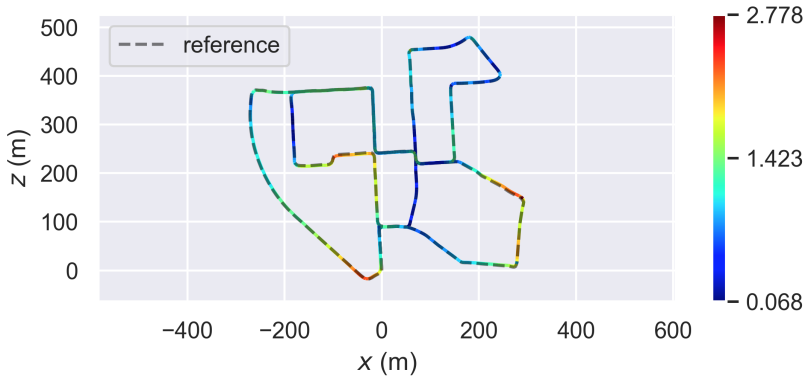


Figure 6.3: *Error distribution on Sequence 00 of the KITTI odometry dataset [143].*

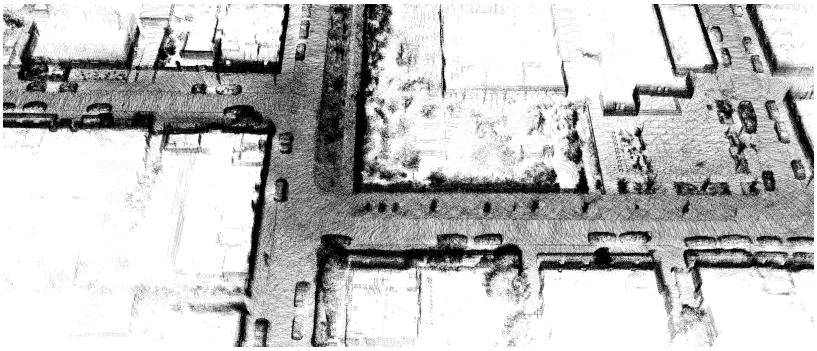


Figure 6.4: *Detailed area of the map corresponding to Sequence 00 of the KITTI odometry dataset [143], built by the proposed algorithm, D3VIL-SLAM.*

which has only one loop at the end of the trajectory. Lastly, the results associated with the short testing sequence are on par with the baseline, probably caused by the fact that, for short trajectories, there is not enough information to obtain accurate estimates, even when including visual information.

All laser modules of D3VIL-SLAM work in real-time, being modules of ART-SLAM. Moreover, vision tracking runs at 10 Hz, while loop detection runs at 25 Hz, on average. Considering that all modules work in parallel, the whole system is able to maintain real-time performance. Lastly, Figure 6.4 shows a detailed area, associated with multiple loop closures, of the 3D map of Sequence 00, reconstructed with D3VIL-SLAM.

6.1.10 Ablation study

As done for Section 6.1.9, we use the ATE metric to evaluate the accuracy of the trajectories estimated by our system, constrained to work with only

Table 6.2: *Ablation study ATE on three sequences of the KITTI dataset [143, 144].*

ATE [m]	Sequence 00		Sequence 07		City Sequence 05	
	MEAN	STD	MEAN	STD	MEAN	STD
Vision only	10.075	5.557	1.365	0.465	0.912	0.291
Vision + ess.	**6.66**	**4.602**	1.354	0.462	0.788	0.282
Vision + ess. + ESKF	7.762	4.897	**1.348**	**0.460**	**0.778**	**0.275**

vision, vision added with essential matrix information, and vision with essential matrix and ESKF predictions as support in the pose graph.

Table 6.2 shows the behavior of the various methods. Two main considerations can be done. First, the complete vision with essential matrix and ESKF support system almost always guarantees better results, with the exception of Sequence 00 (probably because IMU is gathered at the same frequency of camera images, making the whole ESKF not robust in the long run). Then, the addition of the essential matrix as an additional constraint in the pose graph always improves the accuracy of the estimated trajectory.

CHAPTER 7

RadART-SLAM: Using Radars for Accurate and Real-Time SLAM

As it should be clear at this point of the thesis, SLAM has been extensively studied in the literature, exploiting data coming from different sensors, e.g., camera and lasers. Although we proved, multiple times, that the latter is the best choice in terms of accuracy for outdoor large-scale SLAM, ensuring a high degree of precision can be very challenging when dealing with non-typical scenarios, like in adverse weather conditions (e.g., snow or rain). For example, LiDAR sensors do not work properly in areas with high refraction or with fog and particles, and cameras heavily depend on possible illumination changes that may occur, naturally or not, in the environment.

Radars, instead, are not afflicted by these kinds of problems and can be considered as reliable sources of data, serving as replacements of more conventional sensors, to perform SLAM. To enhance the framework proposed in this thesis and complete it with a new sensor, not so common in literature, we developed a radar-based accurate and real-time SLAM system, named RadART-SLAM, which we describe in the following section.

7.1 Radar SLAM

In recent years, the Frequency-Modulated Continuous Wave (FMCW) radar sensors, which can work in many kinds of weather (differently from cameras or LiDAR sensors), have been increasingly used in the field of autonomous driving. Nevertheless, a still open issue is the development of a system that performs robust, accurate, and real-time SLAM in large-scale environments, using only radars, to cope with all possible weather conditions.

Due to their longer wavelength, radar sensors are robust to small environmental artifacts, such as particles of dust, or even snow. Moreover, they can see through certain materials, extending their detection area, going further than the line of sight that LiDAR sensors have. Nevertheless, radars gather data with low spatial resolution and they suffer from higher noise, making localization and mapping inherently difficult to achieve.

To address this problem, the last step of our Ph.D. program was the creation of a radar-based SLAM system, known as Radar for Accurate and Real-Time SLAM, i.e., RadART-SLAM. In the developed method, we perform 2D motion tracking, as the majority of radar sensors provide only 2D information, different from all other systems developed in the thesis.

Similarly to the pipeline described in [106], we take radar data as input in the form of a polar image and convert it into its Cartesian representation, which is more suitable for the successive steps. Then, from each Cartesian image, we extract salient features, adopting well-known state-of-the-art methods for feature detection and extraction on radar sensors. These features are matched between pairs of consecutive images, similar to what we have seen for D3VIL-SLAM, in Section 6.1.4. Here, however, motion distortion and Doppler correction are also applied, to increase the accuracy of the system. From this procedure, we are able to obtain precise motion estimates.

Loop detection is the hardest task since visual features, which represent 2D points in the world, are very noisy and there are too few of them to adopt a simple scan matching approach, as done in ART-SLAM or any other LiDAR-based system. For this reason, we adopt a novel loop estimation procedure based on submap-to-submap matching and filtering.

We first discuss related works in Section 7.1.1, to give a brief insight into the few works, in literature, that use radars as main sensors. Then, after a high-view description of RadART-SLAM, in Section 7.1.2, we explain its various modules. Feature detection is described in Section 7.1.3, followed by tracking in Section 7.1.4, loop detection in Section 7.1.5 and pose graph building and optimization in 7.1.6. Lastly, Section 7.1.7, and Section 7.1.8 are dedicated to the experimental validation of RadART-SLAM.

7.1.1 Related works

Due to the advantages of radars over other sensors, over the course of the last decade, many works relying on radars have been proposed in the literature, with the majority being only odometry and mapping systems (different from SLAM, because of the presence of loop detection and optimization).

One widely adopted standard is to exploit multiple automotive radar sensors [172], which nowadays offer range and azimuth resolutions comparable with laser rangefinders, to estimate many relative target velocities (Doppler effect), which can be used to find the robot motion [94]. This approach, however, is not necessarily optimal for both odometry estimation and mapping, due to the inherent low accuracy of the velocity measurements.

The focus then shifted to the possibility to exploit the underlying signal data of radars, from which salient features can be extracted in a similar fashion to vision-based systems. Nevertheless, extracting keypoints from radar data and using them for direct data association proved to be challenging.

Jose and Adams [173] were the first to research the application of radars in outdoor SLAM. In particular, they proposed a feature detector that estimates the probability of target presence while augmenting their SLAM formulation to include radar cross-section as a discriminating feature. The radar cross section (RCS) of a target is the fictitious area intercepting that amount of power which, when scattered equally in all directions, produces an echo at the radar equal to that from the target. The works in [174] and [175] directly found the transformation between pairs of dense scans, using 3D cross-correlation and the Fourier-Mellin transform.

Cen and Newmann [104] then presented a method to extract stable keypoints in radar images, which are then used to perform scan matching, and accurately estimate the motion of the robot. The same authors later presented a update [105] to their radar odometry pipeline, which improved keypoint detection, and descriptors, and proposed a new graph matching strategy.

More recently, works using only radars either focused on improving aspects of odometry [102, 176, 177], developing more accurate SLAM systems [99, 103], or performing place recognition [178] (i.e., loop detection).

Particularly interesting are the odometry estimation methods described in [106] and [177]. In the first, also known in the literature as YETI, the authors quantified the importance of motion distortion and showed that Doppler effects should be removed during both localization and mapping. In PhaRaO, instead of using feature-based methods for motion estimation, a direct method has been proposed. In particular, the Fourier-Mellin transform is applied on Cartesian and log-polar radar images, to estimate rotation and

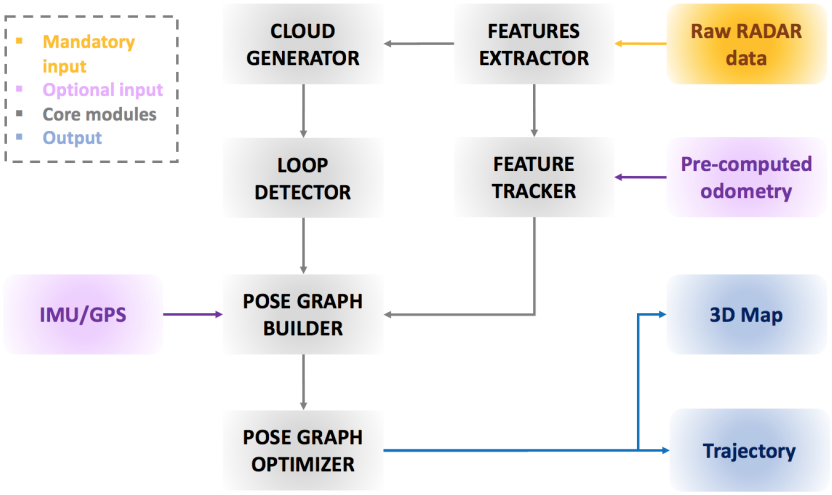


Figure 7.1: *Architecture of the proposed real-time Radar SLAM system, RadART-SLAM.*

translation, decoupled in a coarse-to-fine manner for real-time performance.

Alhashimi et al. [179] presented the current state-of-the-art in radar odometry. Their method builds on prior work by Adolfsson et al. [180] by using a feature extraction algorithm called Bounded False Alarm Rate (BFAR) to add a constant offset to the usual Constant False Alarm Rate (CFAR) threshold. The resulting radar point clouds are registered to a sliding window of keyframes using an ICP-like optimizer.

7.1.2 System overview

The architecture of RadART-SLAM, in Figure 7.1, is very similar to all pipelines presented in the thesis, as it is built on the same common framework. In particular, one may think of the architecture of ART-SLAM [1], from which it takes most of the modules, changing the way they work.

Instead of pre-filtering a point cloud, here a radar image must be processed, to detect from it salient points, i.e., keypoints, and extract the corresponding image descriptors. It should be noticed that conventional feature detectors cannot be used in our case, as radar images are much different from standard camera images, so the same algorithms cannot be applied. Indeed, many visual artifacts are present in radar images, such as multi-path reflection, receiver saturation, or speckle noise, and they influence the visual descriptors, as those generally require the computation of pixel statistics around an image feature. Moreover, these artifacts can quickly disappear between consecutive frames, making standard visual descriptors unsuitable

(the same feature may end up having relatively different descriptors).

Once radar features are extracted between a pair of consecutive scans, they are matched to estimate the current motion of the robot, adopting a particular strategy. Tracking, in RadART-SLAM, consists of three consecutive steps. First, brute force matching allows to find correspondences between two sets of features, using their descriptor distance as a similarity metric.

All pairs of correspondences, which represent elements in radar images but also 2D points surrounding the sensor, are then used to estimate the velocity of the robot. This is exploited to compensate for motion distortion and, successively, reduce the Doppler effect on each considered pair of correspondences. This procedure is iterated multiple times, until the best correspondences, also known as inliers, are found. Lastly, they are used to estimate the current motion of the robot through a modified RANSAC.

As in all other developed SLAM systems, the pose estimate is sent, along with the radar image and corresponding 2D cloud to the loop detector module. Loop detection works similarly to ART-SLAM, but instead of matching single scans, submap-to-submap alignment is performed, instead. This is motivated by the fact that while 3D LiDAR scans are more informative and contain a considerable amount of points that can be used for scan matching, the 2D cloud associated with radar features is sparse, reduced in size, and noisy. Poses and loops are lastly used to build the pose graph, which is periodically optimized to satisfy all constraints.

7.1.3 Feature detector

As mentioned in Section 7.1.1, feature detection on radar data has proven to be more challenging w.r.t. its laser and vision counterparts. There are multiple reasons for this issue, including higher noise floor and much lower spatial resolution, one order of magnitude less than for laser sensors.

Nevertheless, the work in [104] proposed a feature detector that estimates multiple statistics about the noise of a radar signal, and then it scales the power at each measured range by the probability that it represents a real detection. Successively, in [105] the same authors developed a second, improved, detector, which identifies continuous regions of the radar scan characterized by high return intensities and low gradients. Keypoints are then detected and extracted by locating the middle of each region. YETI [106] improved the performance issues of these detectors by applying several modifications, such as the use of a Gaussian filter instead of a binomial filter.

Radar data is usually given either as a list of signals, i.e., a sequence of power returns, or in the form of a polar image, where each row represents

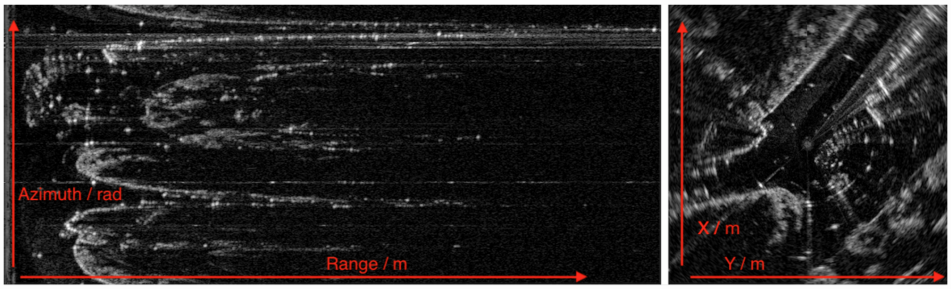


Figure 7.2: *Power returns in Polar form (left), and converted to Cartesian form (right).*

the sensor reading at the corresponding azimuth and each column contains the raw power return at a particular range. The majority of sensors output radar scans in polar form (left image of Figure 7.2). However, both feature extraction methods described above work directly on the raw data, extracting a list of salient points from the numerical values. For this reason, we convert polar images into lists of power returns, by reading them pixel by pixel.

Once the features have been detected and extracted, they need to be associated with a visual descriptor, to be later matched as in vision-based SLAM. This cannot be achieved by calculating descriptors on the polar representation, as it contains both azimuths and distances along those angles. Instead, the majority of descriptors must be computed on images representing only distances. For this reason, we convert the radar scan output, which is in polar form, into a Cartesian image (right picture of Figure 7.2), similar to a bird's eye view. We then calculate a descriptor for each extracted keypoint, directly on it. In particular, we chose ORB descriptors [61], as they are characterized by rotational invariance and resistance to noise.

7.1.4 Tracker

Even in RadART-SLAM, the tracker, which also performs short-term data association, adopts a keyframe-based approach to estimate the trajectory of the robot. As plentifully seen in literature, pure feature matching is not enough to correctly estimate the motion between consecutive radar scans, due to their noise. For this reason, following the approach described in [106], we perform tracking in multiple, consecutive steps, described as follows.

Feature matching

To even begin data association, the extracted features, along with the corresponding descriptors, need to be related in some way. In RadART-SLAM,

we perform brute-force matching of ORB descriptors, coupling together keypoints, from consecutive radar scans, by their overall similarity. We then apply a nearest-neighbor distance ratio test to remove false matches. First, given a feature in a radar scan, we find its two nearest neighbors in the following input. Let d_1 and d_2 be the distances to the nearest and second nearest neighbors. A feature is considered a good match if the ratio d_1/d_2 is smaller than a given threshold. The motivation is that we expect a good match to be much closer to the query feature than the second best match.

Compensating motion distortion

The output of feature matching is not perfect and often contains outliers. The second step of the tracking procedure is the usage of an additional rejection scheme that simultaneously keeps into account motion distortions caused by the rotation of the radar sensor, known as motion-compensated RANSAC.

Instead of direct estimation, in motion-compensated RANSAC the goal is to derive the velocity of the sensor, under the assumption of constant linear and angular velocities between a pair of consecutive radar scans (which is not unrealistic, given that the two inputs are usually taken at high frequency). In other words, we want to estimate the velocity vector $\bar{w} = [v\ \omega]$, where v and ω are the linear and angular velocities, respectively, in the sensor frame.

From the feature matching step, we obtained two sets of corresponding good features, $f_{m,t-1}$ and $f_{m,t}$, where $m \in \{1, \ldots, M\}$ is the index of the pair of matching features, and $t-1$ and t refer to the radar scans being consecutive, taken at timestamps $time_{m,t-1}$ and $time_{m,t}$, respectively. Let $T_s(t)$ be a 4 by 4 homogeneous transformation matrix representing the pose of the sensor frame F_s w.r.t. the inertial frame, at time t. The transformation between a pair of measurements is then defined as

$$T_m = T_s(time_{m,t})\ T_s^{-1}(time_{m,t-1}). \tag{7.1}$$

The local transformation T_m can be used to predict the Cartesian pose of one feature in the matched radar image, and consequently compute an estimation error: $e_m = f_{m,t} - T_m\ f_{m,t-1}$, leading to the error function:

$$J(\bar{w}) = \frac{1}{2} \sum_m e_m^T R_m^{-1} e_m, \tag{7.2}$$

where R corresponds to the covariance in the local Cartesian frame.

A transformation $T \in SE(3)$ is related to its associated Lie algebra $\epsilon^\wedge \in \mathfrak{se}(3)$ through the exponential map. Considering the constant velocity assumption between consecutive radar scans, the velocity vector we are

searching for can be reformulated as a transformation matrix, such that:

$$T = exp(\Delta t\ \bar{w}^{\wedge}). \tag{7.3}$$

To optimize the objective function exploiting Lie algebra, we first derive the relationship between T_m and $\bar{w}$. The velocity vector can, in fact, be rewritten as a sum of nominal velocity and a small perturbation:

$$T_m = exp(\Delta t_m(\widetilde{\bar{w}} + \delta\bar{w})^{\wedge}) = \widetilde{T}_m + \delta T_m, \tag{7.4}$$

where $\widetilde{T}_m$ is the nominal transformation and δT_m the small perturbation.

The quantity $T_m\ f_{m,t-1}$, which is non-linear due to the transformation, can be linearized around a nominal operating point, using Tailor expansions:

$$\begin{aligned} T_m\ f_{m,t-1} &= exp(\Delta t_m \delta\bar{w}^{\wedge})\ \widetilde{T}_m\ f_{m,t-1} \\ &\approx (1 + \Delta t_m \delta\bar{w}^{\wedge})\ \widetilde{T}_m\ f_{m,t-1}. \end{aligned} \tag{7.5}$$

Lastly, we swap the elements using the $()^{\odot}$ operator [181], such that:

$$\begin{aligned} T_m\ f_{m,t-1} = \widetilde{T}_m\ f_{m,t-1} + \Delta t_m(\widetilde{T}_m\ f_{m,t-1})^{\odot}\ \delta\bar{w} \\ \widetilde{g} + G_m\ \delta\bar{w} \end{aligned} \tag{7.6}$$

This way we are able to re-write the prediction error as dependent on the velocity vector perturbation. By inserting this new error into Equation 7.2, representing the objective function, and taking its derivative w.r.t. the perturbation and setting it to zero, we obtain the optimal update $\delta\bar{w}^{*}$. This optimal perturbation is then used, iteratively, in a Gauss-Newton optimization scheme and the estimation process repeats until convergence.

This method allows the efficient estimation of the linear and angular velocities between a pair of consecutive radar scans, while also accounting for motion distortion. In doing so, given the time difference between two inputs, we can also estimate the corresponding relative motion, which is needed to perform pose graph and optimization, other than for motion tracking.

Doppler effect correction

As for all sensors, the motion of a radar sensor results in an apparent relative velocity between the sensor itself and the environment surrounding the robot. The radial component of this relative velocity causes the frequency received by the sensor to be altered according to the Doppler effect.

Let $v = v_x cos(\theta) + v_y sin(\theta)$ be the relative velocity. Then, the Doppler frequency can be formulated as:

$$f_d = \frac{2v}{\lambda}, \tag{7.7}$$

where λ is the wavelength of the signal. From the equation we can see that an object moving towards the sensor (i.e., with positive velocity) will correspond to a positive Doppler frequency, resulting in a higher frequency received by the radar (i.e., the object seems closer than it actually is).

For FMCW radar sensors, the distance to a target is determined by measuring the change in frequency between the received signal and the carrier wave Δf:

$$r = \frac{c\,\Delta f}{2\,(df/dt)}, \tag{7.8}$$

where c is the speed of light, and df/dt is the slope of the modulation pattern used by the carrier wave. FMCW radar sensors require two measurements to disentangle the frequency shift resulting from range and relative velocity. For sensors that scan each azimuth only once, the measured frequency shift is the combination of both the range difference and Doppler frequency, which can be compensated in the following way. Let $\beta = f_t/(df/dt)$, where f_t is the transmission frequency of the sensor. To correct for the Doppler distortion, the range of each target needs to be corrected by the amount:

$$\Delta r_{corr} = \beta(v_x cos(\theta) + v_y sin(\theta)). \tag{7.9}$$

7.1.5 Loop detection

As described in Section 2.5, loop identification using only radar sensors is inherently difficult, due to the nature of the collected data. Vision-based methods cannot be adopted, as radar images corresponding to the same place may differ greatly if gathered with a certain time gap one from the other (even after a few minutes only), due to the many artifacts that afflict them, including multi-path reflections, sensor saturation and speckle noise.

Using point clouds is, instead, a more feasible solution, but radar data, especially from scanning radars, is usually given in the form of a 2D polar image. Thanks to the feature extraction method, previously described in 7.1.3, we are able to extrapolate 2D points from the input images. It should be noted that the extracted points are already in world coordinates, as the sensor resolution is almost always known. This way, for each scan we are able to extract a 2D noisy cloud, which can be used for loop detection.

Similarly to ART-SLAM, loop detection and closure in RadART-SLAM are performed in three consecutive steps, to efficiently search for loops within the collected radar keyframes. Each time a radar keyframe is available, it is compared against all previously created keyframes. A previous keyframe is considered a candidate for loop closure if the corresponding estimated pose is close enough to the pose associated with the current keyframe.

Moreover, if the two structures are too close in time or have a low difference in accumulated distance, they cannot be considered a loop candidate pair.

The second phase for loop detection exploits the method described in [140], which demonstrates to be resilient also when dealing with 2D point clouds (e.g., from radar images). Radar clouds are binned into 2D polar grids, which are compared using a KD tree, to select the most similar ones to a given point cloud query. Although the method was developed for dense point clouds derived from LiDARs, it can be also used in our case without loss in detection accuracy. From this procedure, we are able to obtain k candidate scans w.r.t. the cloud associated with the new keyframe.

Performing scan-to-scan matching on the elements of each pair (formed by new and candidate radar scans) is not enough to correctly identify loops. 2D clouds derived from radar data are too noisy and sparse to obtain a truthful result using scan matching, which is instead recommended when working on 3D LiDAR scans. The idea is then to create two submaps, one associated with each element of a pair for loop closure. In particular, the radar keyframe that is a candidate for loop detection is associated with a large submap, including at least 40 radar scans around it (20 temporally before it, and 20 after). For the other element, which is the newly constructed keyframe, is not possible to include in the submap all elements that temporally came after it, since the structure is the last formed in the trajectory. For this reason, the submap associated with the new keyframe consists of 1 to 10 keyframes that precede it in the estimated trajectory.

The two available submaps can now be used to perform submap-to-submap matching, to obtain a set of relative motions. All transformation matrices are then compared to find the best one, i.e., the one with the highest accuracy and corresponding to the smallest Euclidean distance between all the cloud pairs. If a best match is found, it means that a new loop has been efficiently detected and closed, and it is added to the pose graph as a new constraint, to be later used during the optimization procedure.

7.1.6 Pose graph building and optimization

As part of the framework developed in this thesis, RadART-SLAM is a form of graph SLAM [139]. Each node in the pose graph represents a radar keyframe, including the corresponding location in the world, the radar image, and the 2D point cloud obtained from the features extracted from the input, as described in Section 7.1.3. An edge between two nodes corresponds to the relative motion between the associated 2D poses. This transformation is either the result of tracking, or a constraint derived from loop detection.

Periodically, the pose graph is optimized, to align all nodes such that their corresponding poses best satisfy all measurements associated with the edges.

7.1.7 Experimental validation of the system

To our knowledge, no radar-based SLAM system has been made publicly available and not many radar odometry works are fully functional. For example, PhaRaO [177] is only partially implemented and currently discontinued. For this reason, we compare RadART-SLAM with the algorithm used as the baseline to perform tracking only, YETI [106], whose code is accessible.

We evaluate the two systems on the five most complex sequences, in terms of trajectory shape and presence of loops, of the MulRan dataset [182]. For both methods, we use the same parameters for motion tracking, as described by the authors of YETI. In particular, we use the algorithm described in [105] to extract 2-dimensional features from radar images, as it is efficient and reliable. Moreover, we set the inlier rejection threshold for motion distorted RANSAC to 0.35 and the parameter for Doppler compensation (β, as we have seen in the last part of Section 7.1.4) to 0.049.

As always, for a fair comparison, the systems are tested on a 2021 XMG 64-bit laptop with Intel(R) Core(TM) i7-11800H CPU @ 2.30GHz x 8 cores, with 24576 KB of cache size. Although available, no GPU has been used.

7.1.8 Comparison and results

As done for all systems developed over the course of the Ph.D. program, to evaluate RadART-SLAM and YETI we compute the absolute trajectory error (ATE), i.e., the difference between coordinates of the locations belonging to the true and the estimated trajectories, in all considered scenarios.

Table 7.1 represents the mean, root mean square error (RMSE), and standard deviation (STD) of the absolute trajectory error, in meters, on all tested sequences on DCC and KAIST, which are two areas of the MulRan dataset. RadART-SLAM proves to be the most precise method of the two, reaching an accuracy of 2 to 10 times better than the other method. This was expected, as YETI is a radar odometry method, and loop detection and optimization are not performed, differently from RadART-SLAM, which is a complete SLAM system. Moreover, one should also consider the error obtained when compared with the length of all five trajectories. In particular, DCC is 4.9 km long, on average, while KAIST is 6.1 km, which demonstrated that the resulting errors are acceptable and justified, also keeping into account that such results are achieved using only radar data, which is 2-dimensional, sparse, and inherently noisy.

Table 7.1: *ATE of the compared systems on five sequences of the MulRan dataset [182].*

ATE[m]		YETI	RadART-SLAM
DCC01	MEAN	27.928	**7.418**
	RMSE	30.696	**8.676**
	STD	12.741	**4.499**
DCC02	MEAN	23.856	**9.108**
	RMSE	25.386	**11.079**
	STD	8.681	**6.309**
DCC03	MEAN	30.753	**8.767**
	RMSE	35.534	**9.839**
	STD	17.803	**4.208**
KAIST02	MEAN	30.090	**7.459**
	RMSE	32.877	**8.035**
	STD	13.244	**2.968**
KAIST03	MEAN	113.132	**7.947**
	RMSE	120.214	**8.904**
	STD	49.749	**4.015**

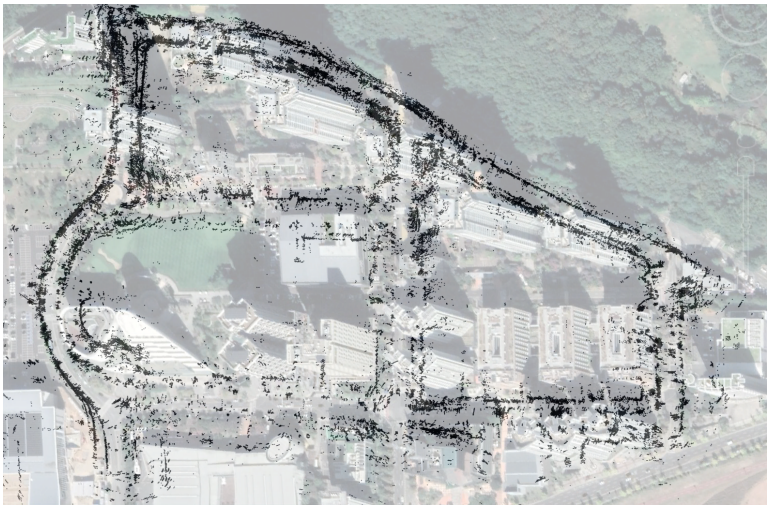


Figure 7.3: *Map of Sequence DCC03 of the MulRan dataset [182], built by the proposed algorithm, RadART-SLAM, overlaid on the corresponding area visible in Google Earth.*

We also include a visual evaluation of two of the reconstructions obtained using RadART-SLAM. Figure 7.3 shows the map representing Sequence DCC03 of the dataset, overlaid on the corresponding area visible in Google Earth. Similarly, Figure 7.4 shows the map, again obtained with RadART-

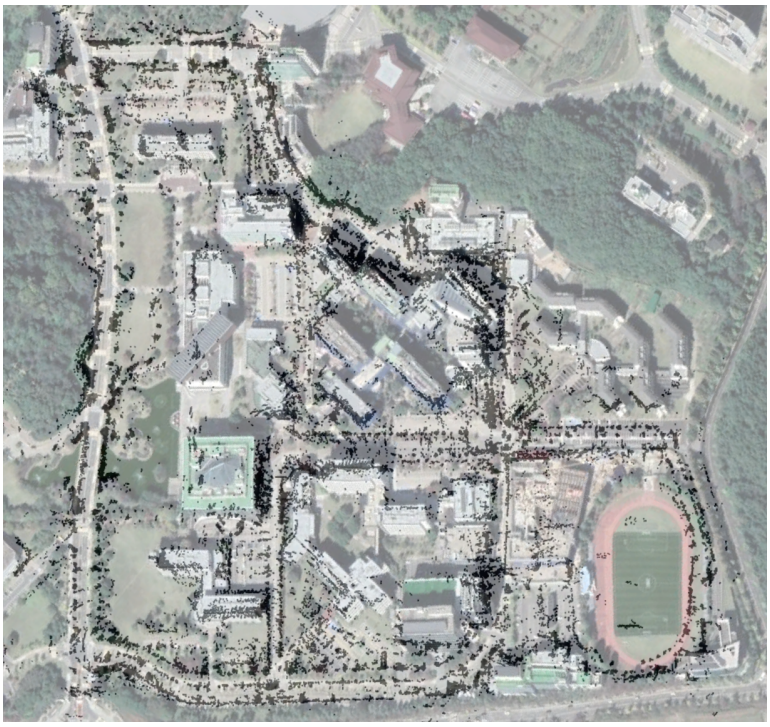


Figure 7.4: *Map of Sequence KAIST03 of the MulRan dataset [182], built by the proposed algorithm, RadART-SLAM, overlaid on the corresponding area visible in Google Earth.*

SLAM, of Sequence KAIST03, placed on top of the analogous world location, which can be seen using Google Earth. Both maps, even if sparse, show how accurate RadART-SLAM is, being correctly located.

To conclude, we give some information about the processing time of the most important modules of RadART-SLAM, in a similar fashion to how has been done for all other systems developed and presented in the thesis. The fastest module of the proposed SLAM method is the tracker, as it is mainly based on RANSAC on a limited amount of points. Its average processing time per frame is less than 8 ms, including also the operations needed to compensate for motion distortion and Doppler effect. Feature detection and extraction is, instead, the bottleneck of the system, requiring at least 40 ms per frame. Lastly, the loop detector proves to be a very fast operation, with an average of less than 10 ms per frame. As all modules work in parallel, it is intuitive how RadART-SLAM is able to work in real-time, or even faster.

CHAPTER 8

Conclusions and future works

The goal of our Ph.D. program was the development of a common framework for 2D and 3D SLAM, with a major focus on the latter, which can be adapted and used to deal with a specific situation or user needs. The majority of works existing in literature are monolithic and highly different one from the other, making it difficult to extend or even find an already available system that tackles a particular situation (e.g., SLAM in tunnels using only lasers).

With the intent of overcoming this issue, the developed framework consists of a series of novel SLAM systems, implemented by us, all able to achieve accurate and real-time localization under different environmental and hardware constraints. All developed methods outperform state-of-the-art methods in terms of localization accuracy, while also being real-time, or even faster, memory efficient, modular, and easily extendable.

As a first step of our Ph.D. program, we developed the base of our framework, ART-SLAM [1], which is a LiDAR graph-based 3D SLAM system. As for all laser methods, ART-SLAM achieves good localization accuracy (better than state-of-the-art systems) and is able to run in real-time. In ART-SLAM, an input point cloud is processed and used to track the robot poses, by matching it against previous scans. The estimated pose, along with additional information, and an efficient loop detection method, are used

to optimize the trajectory and to build a detailed 3D map.

From there, we also considered the possibility of performing not only SLAM but also localization using already available 3D maps [2]. As the second step of our Ph.D. program, we extended ART-SLAM with a module used for UKF-based localization in GNSS-denied environments, and we benchmarked the framework comparing it with other methods able to perform both SLAM and localization, using manually collected data.

We then developed a hybrid SLAM method, as the third step of our Ph.D. program, and derived MCS-SLAM [3]. Here, instead of matching consecutive input LiDAR scans, as in ART-SLAM, point clouds are converted into 2D range/depth images and possibly coupled with RGB images taken from cameras. From a pair of RGB and range/depth images, multiple cues are extracted, such as intensity, depth, and normals, which are later jointly exploited to efficiently estimate the motion of the robot.

Returning to the original ART-SLAM, in the fourth step of our Ph.D. program we then addressed the situation where prior information coming from mapping services is already available. To do so, we integrated 2D maps obtained from OpenStreetMaps into the existing framework, developing OSM-SLAM [4]. The system tries to find the best alignment between the current LiDAR scan, opportunely processed, and the 2D map of buildings downloaded from OSM. This alignment enforces further constraints in the optimization step, increasing local accuracy and correcting the OSM map.

To fully exploit the availability of multiple sensors by direct data association, as the fifth step of our Ph.D. program we developed a visual inertial laser 3D SLAM method, named D3VIL-SLAM. The system combines the laser front-end of ART-SLAM with a backbone Error State Kalman Filter and a stereo vision-based front-end, which can be used in the case of pure visual SLAM. Moreover, an enhanced double-loop detection method, which exploits both scans and images, further increases the accuracy of the system.

As a last step of our Ph.D. program, we considered the case, still not completely addressed in the literature, where SLAM is performed using only radar sensors, due to their multiple advantages. For this reason, we developed RadART-SLAM, a localization and mapping system able to achieve accurate and real-time results even on kilometers-long trajectories.

The development of this common framework for SLAM allows for many future improvements. Each of the presented systems can be extended, to tackle particular situations, not fully addressed in the literature. Moreover, the SLAM methods can be strengthened to further improve their localization accuracy, and all algorithms could be massively benchmarked, to show their true capabilities and performance in different scenarios.

One of the main challenges for SLAM algorithms is to deal with various perturbations that can affect the quality of the map and the localization, such as fast motion, non-uniform illumination, or dynamic objects. These perturbations can cause errors in feature extraction, data association, loop detection and closure, and many other modules. Such errors can lead to drift or failure of the developed SLAM systems. To improve the robustness and accuracy of SLAM algorithms, some possible solutions exist.

SLAM systems could be integrated with methods to detect and reject outliers or incorrect matches in the sensor data, such as using robust estimators, geometric verification, or learning-based methods. Moreover, the handling of dynamic objects in the environment may greatly benefit the accuracy of localization and the quality of the created maps. Existing approaches that can be exploited include motion segmentation, object detection and tracking, or even probabilistic modeling. Also, when working on vision-based SLAM, methods should be developed to cope with illumination changes or low-texture scenes, such as using adaptive feature detection and description, multi-modal sensors, or deep learning techniques.

Another existing challenge in SLAM is to include semantic information. This task is of extreme importance nowadays, due to the various applications in emerging or improving fields, such as robotic agriculture. Semantic SLAM is the branch of SLAM that aims to enrich the typical geometric map with semantic labels or attributes, such as object classes, categories, and names. Semantic information can help to improve map readability, data association, loop detection and closure, and scene understanding.

A relatively easy solution for integrating semantic information into SLAM is the development of methods to perform semantic segmentation or detection on the sensor data, such as using deep neural networks, graphical models, or multi-modal fusion. Another approach could be implementing object-level SLAM, which treats objects as the basic units of mapping and localization, rather than points or voxels. Object-level SLAM can reduce the map size and complexity, and enable object manipulation and interaction.

Autonomous data association is also a possibility, as it can associate semantic labels with geometric features without relying on manual annotation or prior knowledge, and it also can enable online learning and adaptation of semantic models. Lastly, SLAM can be integrated with semantic reasoning and inference, which can exploit the semantic relations and constraints among objects and scenes. This can enhance the robustness and accuracy of SLAM, and enable high-level tasks such as planning and navigation.

Deep learning is a powerful tool for feature extraction, representation learning, and end-to-end learning. Deep learning can be used to enhance

various components of SLAM, such as visual odometry, loop detection, mapping, or even scan matching. Deep learning can also be used to learn SLAM models from data without relying on hand-crafted features or assumptions.

In detail, deep-based methods can extract and match features from the sensor data, such as using convolutional neural networks, self-supervised learning, or attention mechanisms. Moreover, deep learning techniques can be used to estimate the relative or absolute pose of one or multiple sensors, and this can be achieved through regression networks, recurrent neural networks, or geometric consistency. Neural networks, such as using generative adversarial networks, auto-encoders, or graph neural networks, can be employed to build and refine the map of the environment. Lastly, reinforcement learning, imitation learning, or differentiable SLAM, can allow to perform end-to-end SLAM, which directly maps the sensor data to the map of the environment and the pose without intermediate steps.

SLAM algorithms often face difficulties when dealing with large-scale or long-term environments that require high memory and computational resources. Developing methods to reduce the complexity and redundancy of the map, handle dynamic changes in the environment, and enable lifelong learning and adaptation is a key challenge for SLAM.

A possible solution is to perform online global loop detection and closure, which can detect revisited places and correct the accumulated drift in the map without storing all the sensor data or performing batch optimization. Moreover, the mapping aspect of SLAM can be improved through submapping or hierarchical mapping, which divides the map into smaller submaps or levels of abstraction, to be later used for local or global optimization, accordingly to the specific system requirements.

Lastly, the developed SLAM systems can be integrated with modules that maintain and update the created map, detecting and handling changes in the environment over time. This is a concept known as lifelong SLAM, and it can also be coupled with self-improving SLAM, which can learn from previous experiences and improve the performance or robustness of SLAM.

Bibliography

[1] Matteo Frosi and Matteo Matteucci. Art-slam: accurate real-time 6dof lidar slam. *IEEE Robotics and Automation Letters*, 7(2):2692–2699, 2022.

[2] Matteo Frosi, Riccardo Bertoglio, and Matteo Matteucci. On the precision of 6 dof imu-lidar based localization in gnss-denied scenarios. *Frontiers in Robotics and AI*, 10:1064930–1064930, 2023.

[3] Matteo Frosi and Matteo Matteucci. Mcs-slam: Multi-cues multi-sensors fusion slam. In *2022 IEEE Intelligent Vehicles Symposium (IV)*, pages 1423–1429. IEEE, 2022.

[4] Matteo Frosi, Veronica Gobbi, and Matteo Matteucci. Osm-slam: Aiding slam with open-streetmaps priors. *Frontiers in Robotics and AI*, 10, 2023.

[5] Josep Aulinas, Yvan R Petillot, Joaquim Salvi, and Xavier Lladó. The slam problem: a survey. *CCIA*, 184(1):363–371, 2008.

[6] Andrew J Davison and David W. Murray. Simultaneous localization and map-building using active vision. *IEEE transactions on pattern analysis and machine intelligence*, 24(7):865–880, 2002.

[7] John Leonard and Paul Newman. Consistent, convergent, and constant-time slam. In *IJCAI*, pages 1143–1150, 2003.

[8] Patric Jensfelt, Danica Kragic, John Folkesson, and M Bjorkman. A framework for vision based bearing only 3d slam. In *Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006.*, pages 1944–1950. IEEE, 2006.

[9] Yanhao Zhang, Teng Zhang, and Shoudong Huang. Comparison of ekf based slam and optimization based slam algorithms. In *2018 13th IEEE Conference on Industrial Electronics and Applications (ICIEA)*, pages 1308–1313. IEEE, 2018.

[10] A Mallios, D Ribas, and P Ridao. Localization advances in the unstructured underwater environment. In *Proceedings of the 9th Hellenic Symposium of Oceanography and Fishery*, pages 111 – 116, 01 2009.

[11] Michael Montemerlo, Sebastian Thrun, Daphne Koller, Ben Wegbreit, et al. Fastslam: A factored solution to the simultaneous localization and mapping problem. *Aaai/iaai*, 593598, 2002.

[12] Sebastian Thrun. Particle filters in robotics. In *Proceedings of the Eighteenth conference on Uncertainty in artificial intelligence*, pages 511–518. Morgan Kaufmann Publishers Inc., 2002.

[13] Paul J Besl and Neil D McKay. Method for registration of 3-d shapes. In *Sensor fusion IV: control paradigms and data structures*, volume 1611, pages 586–606. International Society for Optics and Photonics, 1992.

[14] Feng Lu. *Shape registration using optimization for mobile robot navigation.* University of Toronto, 1996.

[15] Feng Lu and Evangelos Milios. Robot pose estimation in unknown environments by matching 2d range scans. *Journal of Intelligent and Robotic systems*, 18(3):249–275, 1997.

[16] Shihua Li, Jingxian Wang, Zuqin Liang, and Lian Su. Tree point clouds registration using an improved icp algorithm based on kd-tree. In *2016 IEEE International Geoscience and Remote Sensing Symposium (IGARSS)*, pages 4545–4548. IEEE, 2016.

[17] Yang Chen and Gérard Medioni. Object modelling by registration of multiple range images. *Image and vision computing*, 10(3):145–155, 1992.

[18] A Segal, D Haehnel, and S Thrun. Generalized-icp. *Robotics: Science and Systems V*, 25, 2009.

[19] Fang Wang and Zijian Zhao. A survey of iterative closest point algorithm. In *2017 Chinese Automation Congress (CAC)*, pages 4395–4399. IEEE, 2017.

[20] Steven A Parkison, Lu Gan, Maani Ghaffari Jadidi, and Ryan M Eustice. Semantic iterative closest point through expectation-maximization. In *BMVC*, page 280, 2018.

[21] Artem L Pavlov, Grigory WV Ovchinnikov, Dmitry Yu Derbyshev, Dzmitry Tsetserukou, and Ivan V Oseledets. Aa-icp: Iterative closest point with anderson acceleration. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1–6. IEEE, 2018.

[22] Xiuying Shi, Jianjun Peng, Jiping Li, Pitao Yan, and Hangyu Gong. The iterative closest point registration algorithm based on the normal distribution transformation. *Procedia Computer Science*, 147:181–190, 2019.

[23] 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. IEEE, 2016.

[24] Andreas Nüchter, Kai Lingemann, Joachim Hertzberg, and Hartmut Surmann. 6d slam-3d mapping outdoor environments. *Journal of Field Robotics*, 24(8-9):699–722, 2007.

[25] Hartmut Surmann, Andreas Nüchter, and Joachim Hertzberg. An autonomous mobile robot with a 3d laser range finder for 3d exploration and digitalization of indoor environments. *Robotics and Autonomous Systems*, 45(3-4):181–198, 2003.

[26] Hartmut Surmann, Andreas Nüchter, Kai Lingemann, and Joachim Hertzberg. 6d slam-preliminary report on closing the loop in six dimensions. *IFAC Proceedings Volumes*, 37(8):197–202, 2004.

[27] Jens Behley and Cyrill Stachniss. Efficient surfel-based slam using 3d laser range data in urban environments. In *Robotics: Science and Systems*, volume 2018, page 59, 2018.

[28] Kenji Koide, Jun Miura, and Emanuele Menegatti. A portable 3d lidar-based system for long-term and wide-area people behavior measurement. *IEEE Trans. Hum. Mach. Syst*, 2018.

[29] Pierre Dellenbach, Jean-Emmanuel Deschaud, Bastien Jacquet, and François Goulette. Ct-icp: Real-time elastic lidar odometry with loop closure. In *2022 International Conference on Robotics and Automation (ICRA)*, pages 5580–5586. IEEE, 2022.

[30] Ji Zhang and Sanjiv Singh. Loam: Lidar odometry and mapping in real-time. In *Robotics: Science and Systems*, volume 2, 2014.

[31] Jean-Emmanuel Deschaud. Imls-slam: scan-to-model matching based on 3d data. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2480–2485. IEEE, 2018.

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

[33] Yue Pan, Pengchuan Xiao, Yujie He, Zhenlei Shao, and Zesong Li. Mulls: Versatile lidar slam via multi-metric linear least square. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 11633–11640. IEEE, 2021.

[34] Han Wang, Chen Wang, Chun-Lin Chen, and Lihua Xie. F-loam: Fast lidar odometry and mapping. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4390–4396. IEEE, 2021.

[35] Giorgio Grisetti, Cyrill Stachniss, and Wolfram Burgard. Improved techniques for grid mapping with rao-blackwellized particle filters. *IEEE transactions on Robotics*, 23(1):34–46, 2007.

[36] Stefan Kohlbrecher, Oskar Von Stryk, Johannes Meyer, and Uwe Klingauf. A flexible and scalable slam system with full 3d motion estimation. In *2011 IEEE international symposium on safety, security, and rescue robotics*, pages 155–160. IEEE, 2011.

[37] 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. IEEE, 2016.

[38] Lu Sun, Junqiao Zhao, Xudong He, and Chen Ye. Dlo: Direct lidar odometry for 2.5 d outdoor environment. In *2018 IEEE Intelligent Vehicles Symposium (IV)*, pages 1–5. IEEE, 2018.

[39] Jun Li, Junqiao Zhao, Yuchen Kang, Xudong He, Chen Ye, and Lu Sun. Dl-slam: Direct 2.5 d lidar slam for autonomous driving. In *2019 IEEE Intelligent Vehicles Symposium (IV)*, pages 1205–1210. IEEE, 2019.

[40] Peter Biber and Wolfgang Straßer. The normal distributions transform: A new approach to laser scan matching. In *Proceedings 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003)(Cat. No. 03CH37453)*, volume 3, pages 2743–2748. IEEE, 2003.

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

[42] Todor Stoyanov, Martin Magnusson, Henrik Andreasson, and Achim J Lilienthal. Fast and accurate scan registration through minimization of the distance between compact 3d ndt representations. *The International Journal of Robotics Research*, 31(12):1377–1393, 2012.

[43] Naoki Akai, Luis Yoichi Morales, Eijiro Takeuchi, Yuki Yoshihara, and Yoshiki Ninomiya. Robust localization using 3d ndt scan matching with experimentally determined uncertainty and road marker matching. In *2017 IEEE Intelligent Vehicles Symposium (IV)*, pages 1356–1363. IEEE, 2017.

[44] Jorge Fuentes-Pacheco, José Ruiz-Ascencio, and Juan Manuel Rendón-Mancha. Visual simultaneous localization and mapping: a survey. *Artificial intelligence review*, 43(1):55–81, 2015.

[45] Jakob Engel, Jürgen Sturm, and Daniel Cremers. Camera-based navigation of a low-cost quadrocopter. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2815–2821. IEEE, 2012.

[46] Jakob Engel, Thomas Schöps, and Daniel Cremers. Lsd-slam: Large-scale direct monocular slam. In *European conference on computer vision*, pages 834–849. Springer, 2014.

[47] M Billinghurst, A Clark, and G Lee. A survey of augmented reality. found trends hum comput interact 8: 73–272, 2015.

[48] Khalid Yousif, Alireza Bab-Hadiashar, and Reza Hoseinnezhad. An overview to visual odometry and visual slam: Applications to mobile robotics. *Intelligent Industrial Systems*, 1(4):289–311, 2015.

[49] Davide Scaramuzza and Friedrich Fraundorfer. Visual odometry [tutorial]. *IEEE robotics & automation magazine*, 18(4):80–92, 2011.

[50] Friedrich Fraundorfer and Davide Scaramuzza. Visual odometry: Part ii: Matching, robustness, optimization, and applications. *IEEE Robotics & Automation Magazine*, 19(2):78–90, 2012.

[51] David Nistér and Henrik Stewénius. A minimal solution to the generalised 3-point pose problem. *Journal of Mathematical Imaging and Vision*, 27(1):67–79, 2007.

[52] Raul Mur-Artal, Jose Maria Martinez Montiel, and Juan D Tardos. Orb-slam: a versatile and accurate monocular slam system. *IEEE transactions on robotics*, 31(5):1147–1163, 2015.

[53] Raul Mur-Artal and Juan D Tardós. Orb-slam2: An open-source slam system for monocular, stereo, and rgb-d cameras. *IEEE transactions on robotics*, 33(5):1255–1262, 2017.

[54] Georg Klein and David Murray. Parallel tracking and mapping for small ar workspaces. In *2007 6th IEEE and ACM international symposium on mixed and augmented reality*, pages 225–234. IEEE, 2007.

[55] Richard A Newcombe, Steven J Lovegrove, and Andrew J Davison. Dtam: Dense tracking and mapping in real-time. In *2011 international conference on computer vision*, pages 2320–2327. IEEE, 2011.

[56] Jakob Engel, Vladlen Koltun, and Daniel Cremers. Direct sparse odometry. *IEEE transactions on pattern analysis and machine intelligence*, 40(3):611–625, 2017.

[57] David G Lowe. Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 60(2):91–110, 2004.

[58] Herbert Bay, Andreas Ess, Tinne Tuytelaars, and Luc Van Gool. Speeded-up robust features (surf). *Computer vision and image understanding*, 110(3):346–359, 2008.

[59] Edward Rosten and Tom Drummond. Machine learning for high-speed corner detection. In *European conference on computer vision*, pages 430–443. Springer, 2006.

[60] Deepak Geetha Viswanathan. Features from accelerated segment test (fast). In *Proceedings of the 10th workshop on Image Analysis for Multimedia Interactive Services, London, UK*, pages 6–8, 2009.

[61] Ethan Rublee, Vincent Rabaud, Kurt Konolige, and Gary Bradski. Orb: An efficient alternative to sift or surf. In *2011 International conference on computer vision*, pages 2564–2571. Ieee, 2011.

[62] Michael Calonder, Vincent Lepetit, Christoph Strecha, and Pascal Fua. Brief: Binary robust independent elementary features. In *European conference on computer vision*, pages 778–792. Springer, 2010.

[63] Stefan Leutenegger, Margarita Chli, and Roland Y Siegwart. Brisk: Binary robust invariant scalable keypoints. In *2011 International conference on computer vision*, pages 2548–2555. Ieee, 2011.

[64] Pablo Fernández Alcantarilla, Adrien Bartoli, and Andrew J Davison. Kaze features. In *European Conference on Computer Vision*, pages 214–227. Springer, 2012.

[65] J. Sturm, W. Burgard, and D. Cremers. Evaluating egomotion and structure-from-motion approaches using the TUM RGB-D benchmark. In *Proc. of the Workshop on Color-Depth Camera Fusion in Robotics at the IEEE/RJS International Conference on Intelligent Robot Systems (IROS)*, Oct. 2012.

[66] Shimiao Li. A review of feature detection and match algorithms for localization and mapping. In *IOP Conference Series: Materials Science and Engineering*, volume 231, page 012003, 2017.

[67] Rafał Scherer. Feature detection. In *Computer Vision Methods for Fast Image Classification and Retrieval*, pages 7–32. Springer, 2020.

[68] Shaharyar Ahmed Khan Tareen and Zahra Saleem. A comparative analysis of sift, surf, kaze, akaze, orb, and brisk. In *2018 International conference on computing, mathematics and engineering technologies (iCoMET)*, pages 1–10. IEEE, 2018.

[69] 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.

[70] Konstantinos G Derpanis. Overview of the ransac algorithm. *Image Rochester NY*, 4(1):2–3, 2010.

[71] Hauke Strasdat, José MM Montiel, and Andrew J Davison. Visual slam: why filter? *Image and Vision Computing*, 30(2):65–77, 2012.

[72] David Nistér. An efficient solution to the five-point relative pose problem. *IEEE transactions on pattern analysis and machine intelligence*, 26(6):756–770, 2004.

[73] Brian Williams, Georg Klein, and Ian Reid. Real-time slam relocalisation. In *2007 IEEE 11th international conference on computer vision*, pages 1–8. IEEE, 2007.

[74] Taihú Pire, Thomas Fischer, Gastón Castro, Pablo De Cristóforis, Javier Civera, and Julio Jacobo Berlles. S-ptam: Stereo parallel tracking and mapping. *Robotics and Autonomous Systems*, 93:27–42, 2017.

[75] Carlos Campos, Richard Elvira, Juan J. Gómez, José M. M. Montiel, and Juan D. Tardós. ORB-SLAM3: An accurate open-source library for visual, visual-inertial and multi-map SLAM. *IEEE Transactions on Robotics*, 37(6):1874–1890, 2021.

[76] Linyan Cui and Chaowei Ma. Sof-slam: A semantic visual slam for dynamic environments. *IEEE Access*, 7:166528–166539, 2019.

[77] Rafael Munoz-Salinas, Manuel J Marin-Jimenez, and Rafael Medina-Carnicer. Spm-slam: Simultaneous localization and mapping with squared planar markers. *Pattern Recognition*, 86:156–171, 2019.

[78] Rafael Munoz-Salinas and Rafael Medina-Carnicer. Ucoslam: Simultaneous localization and mapping by fusion of keypoints and squared planar markers. *Pattern Recognition*, 101:107193, 2020.

[79] Christian Forster, Matia Pizzoli, and Davide Scaramuzza. Svo: Fast semi-direct monocular visual odometry. In *2014 IEEE international conference on robotics and automation (ICRA)*, pages 15–22. IEEE, 2014.

[80] Christian Forster, Zichao Zhang, Michael Gassner, Manuel Werlberger, and Davide Scaramuzza. Svo: Semidirect visual odometry for monocular and multicamera systems. *IEEE Transactions on Robotics*, 33(2):249–265, 2016.

[81] Masatoshi Okutomi and Takeo Kanade. A multiple-baseline stereo. *IEEE Transactions on pattern analysis and machine intelligence*, 15(4):353–363, 1993.

[82] Rui Wang, Martin Schworer, and Daniel Cremers. Stereo dso: Large-scale direct sparse visual odometry with stereo cameras. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 3903–3911, 2017.

[83] David Caruso, Jakob Engel, and Daniel Cremers. Large-scale direct slam for omnidirectional cameras. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 141–148. IEEE, 2015.

[84] Jakob Engel, Jörg Stückler, and Daniel Cremers. Large-scale direct slam with stereo cameras. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1935–1942. IEEE, 2015.

[85] Lukas Von Stumberg, Vladyslav Usenko, and Daniel Cremers. Direct sparse visual-inertial odometry using dynamic marginalization. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2510–2517. IEEE, 2018.

[86] Xiang Gao, Tao Zhang, Yi Liu, and Qinrui Yan. 14 lectures on visual slam: from theory to practice. *Publishing House of Electronics Industry*, 2017.

[87] Andrew J Davison, Ian D Reid, Nicholas D Molton, and Olivier Stasse. Monoslam: Real-time single camera slam. *IEEE transactions on pattern analysis and machine intelligence*, 29(6):1052–1067, 2007.

[88] Jon Zubizarreta, Iker Aguinaga, and Jose Maria Martinez Montiel. Direct sparse mapping. *IEEE Transactions on Robotics*, 2020.

[89] Dominik Schlegel, Mirco Colosi, and Giorgio Grisetti. Proslam: graph slam from a programmer's perspective. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3833–3840. IEEE, 2018.

[90] Richard A Newcombe, Shahram Izadi, Otmar Hilliges, David Molyneaux, David Kim, Andrew J Davison, Pushmeet Kohi, Jamie Shotton, Steve Hodges, and Andrew Fitzgibbon. Kinectfusion: Real-time dense surface mapping and tracking. In *2011 10th IEEE International Symposium on Mixed and Augmented Reality*, pages 127–136. IEEE, 2011.

[91] Renato F Salas-Moreno, Richard A Newcombe, Hauke Strasdat, Paul HJ Kelly, and Andrew J Davison. Slam++: Simultaneous localisation and mapping at the level of objects. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1352–1359, 2013.

[92] Thomas Schops, Torsten Sattler, and Marc Pollefeys. Bad slam: Bundle adjusted direct rgb-d slam. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 134–144, 2019.

[93] Shishun Zhang, Longyu Zheng, and Wenbing Tao. Survey and evaluation of rgb-d slam. *IEEE Access*, 9:21367–21387, 2021.

[94] Dominik Kellner, Michael Barjenbruch, Jens Klappstein, Jürgen Dickmann, and Klaus Dietmayer. Instantaneous ego-motion estimation using doppler radar. In *16th International IEEE Conference on Intelligent Transportation Systems (ITSC 2013)*, pages 869–874. IEEE, 2013.

[95] Frank Schuster, Christoph Gustav Keller, Matthias Rapp, Martin Haueis, and Cristóbal Curio. Landmark based radar slam using graph optimization. In *2016 IEEE 19th International Conference on Intelligent Transportation Systems (ITSC)*, pages 2559–2564. IEEE, 2016.

[96] Erik Ward and John Folkesson. Vehicle localization with low cost radar sensors. In *2016 IEEE Intelligent Vehicles Symposium (IV)*, pages 864–870. IEEE, 2016.

[97] Matthias Rapp, Michael Barjenbruch, Klaus Dietmayer, Markus Hahn, and Jürgen Dickmann. A fast probabilistic ego-motion estimation framework for radar. In *2015 European Conference on Mobile Robots (ECMR)*, pages 1–6. IEEE, 2015.

[98] Matthias Rapp, Michael Barjenbruch, Markus Hahn, Jürgen Dickmann, and Klaus Dietmayer. Probabilistic ego-motion estimation using multiple automotive radar sensors. *Robotics and Autonomous Systems*, 89:136–146, 2017.

[99] Martin Holder, Sven Hellwig, and Hermann Winner. Real-time pose graph slam based on radar. In *2019 IEEE Intelligent Vehicles Symposium (IV)*, pages 1145–1151. IEEE, 2019.

[100] Hermann Rohling. Radar cfar thresholding in clutter and multiple target situations. *IEEE Transactions on Aerospace and Electronic Systems*, AES-19(4):608–621, 1983.

[101] Damien Vivet, Paul Checchin, and Roland Chapuis. Localization and mapping using only a rotating fmcw radar sensor. *Sensors*, 13(4):4527–4552, 2013.

[102] Roberto Aldera, Daniele De Martini, Matthew Gadd, and Paul Newman. What could go wrong? introspective radar odometry in challenging environments. In *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*, pages 2835–2842. IEEE, 2019.

[103] Ziyang Hong, Yvan Petillot, Andrew Wallace, and Sen Wang. Radarslam: A robust simultaneous localization and mapping system for all weather conditions. *The International Journal of Robotics Research*, page 02783649221080483, 2022.

[104] Sarah H Cen and Paul Newman. Precise ego-motion estimation with millimeter-wave radar under diverse and challenging conditions. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6045–6052. IEEE, 2018.

[105] Sarah H Cen and Paul Newman. Radar-only ego-motion estimation in difficult settings via graph matching. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 298–304. IEEE, 2019.

[106] Keenan Burnett, Angela P Schoellig, and Timothy D Barfoot. Do we need to compensate for motion distortion and doppler effects in spinning radar navigation? *IEEE Robotics and Automation Letters*, 6(2):771–778, 2021.

[107] Thomas Whelan, Renato F Salas-Moreno, Ben Glocker, Andrew J Davison, and Stefan Leutenegger. Elasticfusion. *International Journal of Robotics Research*, 35(14):1697–1716, 2016.

[108] Thomas Whelan, Renato F Salas-Moreno, Ben Glocker, Andrew J Davison, and Stefan Leutenegger. Elasticfusion: Real-time dense slam and light source estimation. *The International Journal of Robotics Research*, 35(14):1697–1716, 2016.

[109] Yoshua Nava Chocron. Visual-lidar slam with loop closure, 2019.

[110] Renaud Dubé, Daniel Dugas, Elena Stumm, Juan Nieto, Roland Siegwart, and Cesar Cadena. Segmatch: Segment based loop-closure for 3d point clouds. *arXiv preprint arXiv:1609.07720*, 2016.

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

[112] Young-Sik Shin, Yeong Sang Park, and Ayoung Kim. Dvl-slam: Sparse depth enhanced direct visual-lidar slam. *Autonomous Robots*, 44(2):115–130, 2020.

[113] Wen Xiao, Bruno Vallet, Mathieu Brédif, and Nicolas Paparoditis. Street environment change detection from mobile laser scanning point clouds. *ISPRS Journal of Photogrammetry and Remote Sensing*, 107:38–49, 2015.

[114] Chenglong Qian, Zhaohong Xiang, Zhuoran Wu, and Hongbin Sun. Rf-lio: Removal-first tightly-coupled lidar inertial odometry in high dynamic environments. *arXiv preprint arXiv:2206.09463*, 2022.

[115] 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.

[116] Giseop Kim and Ayoung Kim. Remove, then revert: Static point cloud map construction using multiresolution range images. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 10758–10765. IEEE, 2020.

[117] Ayush Dewan, Tim Caselitz, Gian Diego Tipaldi, and Wolfram Burgard. Motion-based detection and tracking in 3d lidar scans. In *2016 IEEE international conference on robotics and automation (ICRA)*, pages 4508–4513. IEEE, 2016.

[118] Shile Li and Dongheui Lee. Rgb-d slam in dynamic environments using static point weighting. *IEEE Robotics and Automation Letters*, 2(4):2263–2270, 2017.

[119] Yuxiang Sun, Ming Liu, and Max Q-H Meng. Improving rgb-d slam in dynamic environments: A motion removal approach. *Robotics and Autonomous Systems*, 89:110–122, 2017.

[120] Wei Tan, Haomin Liu, Zilong Dong, Guofeng Zhang, and Hujun Bao. Robust monocular slam in dynamic environments. In *2013 IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*, pages 209–218. IEEE, 2013.

[121] Chao Yu, Zuxin Liu, Xin-Jun Liu, Fugui Xie, Yi Yang, Qi Wei, and Qiao Fei. Ds-slam: A semantic visual slam towards dynamic environments. In *2018 IEEE/RSJ international conference on intelligent robots and systems (IROS)*, pages 1168–1174. IEEE, 2018.

[122] Vijay Badrinarayanan, Alex Kendall, and Roberto Cipolla. Segnet: A deep convolutional encoder-decoder architecture for image segmentation. *IEEE transactions on pattern analysis and machine intelligence*, 39(12):2481–2495, 2017.

[123] Armin Hornung, Kai M Wurm, Maren Bennewitz, Cyrill Stachniss, and Wolfram Burgard. Octomap: An efficient probabilistic 3d mapping framework based on octrees. *Autonomous robots*, 34:189–206, 2013.

[124] Berta Bescos, José M Fácil, Javier Civera, and José Neira. Dynaslam: Tracking, mapping, and inpainting in dynamic scenes. *IEEE Robotics and Automation Letters*, 3(4):4076–4083, 2018.

[125] Berta Bescos, Carlos Campos, Juan D Tardós, and José Neira. Dynaslam ii: Tightly-coupled multi-object tracking and slam. *IEEE robotics and automation letters*, 6(3):5191–5198, 2021.

[126] Linhui Xiao, Jinge Wang, Xiaosong Qiu, Zheng Rong, and Xudong Zou. Dynamic-slam: Semantic monocular visual localization and mapping based on deep learning in dynamic environment. *Robotics and Autonomous Systems*, 117:1–16, 2019.

[127] Xieyuanli Chen, Andres Milioto, Emanuele Palazzolo, Philippe Giguere, Jens Behley, and Cyrill Stachniss. Suma++: Efficient lidar-based semantic slam. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4530–4537. IEEE, 2019.

[128] Anestis Zaganidis, Li Sun, Tom Duckett, and Grzegorz Cielniak. Integrating deep semantic segmentation into 3-d point cloud registration. *IEEE Robotics and Automation Letters*, 3(4):2942–2949, 2018.

[129] Fangwei Zhong, Sheng Wang, Ziqi Zhang, and Yizhou Wang. Detect-slam: Making object detection and slam mutually beneficial. In *2018 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 1001–1010. IEEE, 2018.

[130] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. Ssd: Single shot multibox detector. In *Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11–14, 2016, Proceedings, Part I 14*, pages 21–37. Springer, 2016.

[131] Junhao Cheng, Zhi Wang, Hongyan Zhou, Li Li, and Jian Yao. Dm-slam: A feature-based slam system for rigid dynamic scenes. *ISPRS International Journal of Geo-Information*, 9(4):202, 2020.

[132] Yingchun Fan, Qichi Zhang, Shaofeng Liu, Yuliang Tang, Xin Jing, Jintao Yao, and Hong Han. Semantic slam with more accurate point cloud map in dynamic environments. *IEEE Access*, 8:112237–112252, 2020.

[133] Nikita Dvornik, Konstantin Shmelkov, Julien Mairal, and Cordelia Schmid. Blitznet: A real-time deep network for scene understanding. In *Proceedings of the IEEE international conference on computer vision*, pages 4154–4162, 2017.

[134] Yubao Liu and Jun Miura. Rds-slam: Real-time dynamic slam using semantic segmentation methods. *Ieee Access*, 9:23772–23785, 2021.

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

[136] Enrico Piazza, Andrea Romanoni, and Matteo Matteucci. Real-time cpu-based large-scale three-dimensional mesh reconstruction. *IEEE Robotics and Automation Letters*, 3(3):1584–1591, 2018.

[137] Kenji Koide, Masashi Yokozuka, Shuji Oishi, and Atsuhiko Banno. Voxelized gicp for fast and accurate 3d point cloud registration. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 11054–11059. IEEE, 2021.

[138] Marek Pierzchała, Philippe Giguère, and Rasmus Astrup. Mapping forests using an unmanned ground vehicle with 3d lidar and graph-slam. *Computers and Electronics in Agriculture*, 145:217–225, 2018.

[139] Giorgio Grisetti, Rainer Kümmerle, Cyrill Stachniss, and Wolfram Burgard. A tutorial on graph-based slam. *IEEE Intelligent Transportation Systems Magazine*, 2(4):31–43, 2010.

[140] Giseop Kim and Ayoung Kim. Scan context: Egocentric spatial descriptor for place recognition within 3D point cloud map. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, Madrid, Oct. 2018.

[141] Giorgio Grisetti, Rainer Kümmerle, Hauke Strasdat, and Kurt Konolige. g2o: A general framework for (hyper) graph optimization. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA), Shanghai, China*, pages 9–13, 2011.

[142] Francesco Amigoni, Monica Reggiani, and Viola Schiaffonati. An insightful comparison between experiments in mobile robotics and in science. *Autonomous Robots*, 27(4):313–325, 2009.

[143] Andreas Geiger, Philip Lenz, and Raquel Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012.

[144] Andreas Geiger, Philip Lenz, Christoph Stiller, and Raquel Urtasun. Vision meets robotics: The kitti dataset. *International Journal of Robotics Research (IJRR)*, 2013.

[145] Keith Leung, Daniel Lühr, Hamidreza Houshiar, Felipe Inostroza, Dorit Borrmann, Martin Adams, Andreas Nüchter, and Javier Ruiz del Solar. Chilean underground mine dataset. *The International Journal of Robotics Research*, 36(1):16–23, 2017.

[146] Jörg Röwekämper, Christoph Sprunk, Gian Diego Tipaldi, Cyrill Stachniss, Patrick Pfaff, and Wolfram Burgard. On the position accuracy of mobile robot localization based on particle filters combined with scan matching. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3158–3164. IEEE, 2012.

[147] Philipp Egger, Paulo VK Borges, Gavin Catt, Andreas Pfrunder, Roland Siegwart, and Renaud Dubé. Posemap: Lifelong, multi-environment 3d lidar localization. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3430–3437. IEEE, 2018.

[148] Zhuli Ren, Liguan Wang, and Lin Bi. Robust gicp-based 3d lidar slam for underground mining environment. *Sensors*, 19(13):2915, 2019.

[149] Eric A Wan and Rudolph Van Der Merwe. The unscented kalman filter for nonlinear estimation. In *Proceedings of the IEEE 2000 Adaptive Systems for Signal Processing, Communications, and Control Symposium (Cat. No. 00EX373)*, pages 153–158. Ieee, 2000.

[150] Jürgen Sturm, Nikolas Engelhard, Felix Endres, Wolfram Burgard, and Daniel Cremers. A benchmark for the evaluation of rgb-d slam systems. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 573–580. IEEE, 2012.

[151] Rainer Kümmerle, Bastian Steder, Christian Dornhege, Michael Ruhnke, Giorgio Grisetti, Cyrill Stachniss, and Alexander Kleiner. On measuring the accuracy of slam algorithms. *Autonomous Robots*, 27(4):387–407, 2009.

[152] Giulio Fontana, Matteo Matteucci, and Domenico G Sorrenti. Rawseeds: Building a benchmarking toolkit for autonomous robotics. In *Methods and Experimental Techniques in Computer Engineering*, pages 55–68. Springer, 2014.

[153] Du Q Huynh. Metrics for 3d rotations: Comparison and analysis. *Journal of Mathematical Imaging and Vision*, 35(2):155–164, 2009.

[154] Bartolomeo Della Corte, Igor Bogoslavskyi, Cyrill Stachniss, and Giorgio Grisetti. A general framework for flexible multi-cue photometric point cloud registration. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4969–4976. IEEE, 2018.

[155] Yinglei Xu, Yongsheng Ou, and Tiantian Xu. Slam of robot based on the fusion of vision and lidar. In *2018 IEEE International Conference on Cyborg and Bionic Systems (CBS)*, pages 121–126. IEEE, 2018.

[156] Johannes Graeter, Alexander Wilczynski, and Martin Lauer. Limo: Lidar-monocular visual odometry. In *2018 IEEE/RSJ international conference on intelligent robots and systems (IROS)*, pages 7872–7879. IEEE, 2018.

[157] Zulun Zhu, Shaowu Yang, Huadong Dai, and Fu Li. Loop detection and correction of 3d laser-based slam with visual information. In *Proceedings of the 31st International Conference on Computer Animation and Social Agents*, pages 53–58, 2018.

[158] Youngwoo Seo and Chih-Chung Chou. A tight coupling of vision-lidar measurements for an effective odometry. In *2019 IEEE Intelligent Vehicles Symposium (IV)*, pages 1118–1123. IEEE, 2019.

[159] Guolai Jiang, Lei Yin, Shaokun Jin, Chaoran Tian, Xinbo Ma, and Yongsheng Ou. A simultaneous localization and mapping (slam) framework for 2.5 d map building based on low-cost lidar and vision fusion. *Applied Sciences*, 9(10):2105, 2019.

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

[161] Mordechai Haklay and Patrick Weber. Openstreetmap: User-generated street maps. *IEEE Pervasive computing*, 7(4):12–18, 2008.

[162] Matthias Hentschel, Oliver Wulf, and Bernardo Wagner. A gps and laser-based localization for urban and non-urban outdoor environments. In *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 149–154. IEEE, 2008.

[163] Georgios Floros, Benito Van Der Zander, and Bastian Leibe. Openstreetslam: Global vehicle localization using openstreetmaps. In *2013 IEEE International Conference on Robotics and Automation*, pages 1054–1059. IEEE, 2013.

[164] Olga Vysotska and Cyrill Stachniss. Exploiting building information from publicly available maps in graph-based slam. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4511–4516. IEEE, 2016.

[165] Lakshadeep Naik, Sebastian Blumenthal, Nico Huebel, Herman Bruyninckx, and Erwin Prassler. Semantic mapping extension for openstreetmap applied to indoor robot navigation. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 3839–3845. IEEE, 2019.

[166] Miguel Ángel Muñoz-Bañón, Edison Velasco-Sánchez, Francisco A Candelas, and Fernando Torres. Openstreetmap-based autonomous navigation with lidar naive-valley-path obstacle avoidance. *IEEE Transactions on Intelligent Transportation Systems*, 2022.

[167] Younghun Cho, Giseop Kim, Sangmin Lee, and Jee-Hwan Ryu. Openstreetmap-based lidar global localization in urban environment without a prior lidar map. *IEEE Robotics and Automation Letters*, 7(2):4999–5006, 2022.

[168] Joan Sola. Quaternion kinematics for the error-state kalman filter. *arXiv preprint arXiv:1711.02508*, 2017.

[169] Yoshua Nava. *Visual-LiDAR SLAM with loop closure*. PhD thesis, KTH Royal Institute of Technology, 2018.

[170] Wei Wang, Jun Liu, Chenjie Wang, Bin Luo, and Cheng Zhang. Dv-loam: Direct visual lidar odometry and mapping. *Remote Sensing*, 13(16):3340, 2021.

[171] Dominik Schlegel and Giorgio Grisetti. Hbst: A hamming distance embedding binary search tree for feature-based visual place recognition. *IEEE Robotics and Automation Letters*, 3(4):3741–3748, 2018.

[172] Holger Caesar, Varun Bankiti, Alex H Lang, Sourabh Vora, Venice Erin Liong, Qiang Xu, Anush Krishnan, Yu Pan, Giancarlo Baldan, and Oscar Beijbom. nuscenes: A multimodal dataset for autonomous driving. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 11621–11631, 2020.

[173] Ebi Jose and Martin David Adams. Relative radar cross section based feature identification with millimeter wave radar for outdoor slam. In *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)(IEEE Cat. No. 04CH37566)*, volume 1, pages 425–430. IEEE, 2004.

[174] R Rouveure, MO Monod, and P Faure. High resolution mapping of the environment with a ground-based radar imager. In *2009 International Radar Conference" Surveillance for a Safer World"(RADAR 2009)*, pages 1–6. IEEE, 2009.

[175] Paul Checchin, Franck Gérossier, Christophe Blanc, Roland Chapuis, and Laurent Trassoudaine. Radar scan matching slam using the fourier-mellin transform. In *Field and Service Robotics*, pages 151–161. Springer, 2010.

[176] Pou-Chun Kung, Chieh-Chih Wang, and Wen-Chieh Lin. A normal distribution transform-based radar odometry designed for scanning and automotive radars. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 14417–14423. IEEE, 2021.

[177] Yeong Sang Park, Young-Sik Shin, and Ayoung Kim. Pharao: Direct radar odometry using phase correlation. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2617–2623. IEEE, 2020.

[178] Daniele De Martini, Matthew Gadd, and Paul Newman. kradar++: coarse-to-fine fmcw scanning radar localisation. *Sensors*, 20(21):6002, 2020.

[179] Anas Alhashimi, Daniel Adolfsson, Martin Magnusson, Henrik Andreasson, and Achim J Lilienthal. Bfar-bounded false alarm rate detector for improved radar odometry estimation. *arXiv preprint arXiv:2109.09669*, 2021.

[180] Daniel Adolfsson, Martin Magnusson, Anas Alhashimi, Achim J Lilienthal, and Henrik Andreasson. Cfear radarodometry-conservative filtering for efficient and accurate radar odometry. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5462–5469. IEEE, 2021.

[181] Timothy D Barfoot. *State estimation for robotics*. Cambridge University Press, 2017.

[182] Giseop Kim, Yeong Sang Park, Younghun Cho, Jinyong Jeong, and Ayoung Kim. Mulran: Multimodal range dataset for urban place recognition. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6246–6253. IEEE, 2020.