POLITECNICO DI TORINO

Master's Degree in Computer Engineering

Master's Degree Thesis

Distributed Lidar-based Simultaneous Localization and Mapping

Supervisor

Prof. Marina INDRI

Advisors

Dott. Gianluca PRATO

Dott. Enrico FERRERA

Candidate

Francesco AGLIECO

December 2022

Abstract

Simultaneous Localization and Mapping (SLAM) algorithms provide a robust solution for mobile robots localization and map building of surrounding environment even when most used positioning systems (GPS) are not available for autonomous navigation, such as in indoor or subterranean locations. Being able to set a team of robots to resolve this common task and enable it to collaborate, it is possible to obtain better results in shorter time. In this thesis work, carried out in collaboration with LINKS Foundation, a fully distributed collaborative SLAM system, based on lidar sensor data processing, has been designed and partially developed exploiting the ROS2 open-source framework. A distributed approach, where every robot is implied on perception and optimization tasks, is selected according to robustness, scalability and security requirements, facing with algorithm complexity. The solution is modeled in five blocks following the typical structure characterising implemented systems in recent research. The point clouds coming from sensors are firstly processed by the Signal Processing module to provide odometry and a downsampled representation of data, useful to ease successive elaborations. The odometry gives ego-motion of a robot, here lidar-based algorithms are chosen in order to exploit the better results regarding this first localization information as compared to other sensor suites. In order to reduce estimation error caused by odometry drift, robots must recognize whether a position is already crossed, detecting the so-called Loop Closures. A robot within a team can perform the detection during its navigation (Intra-Robot Loop Closure detection) or during a rendezvous (Inter-Robot Loop Closure detection). In the latter case the robot shares its estimations and compares visited locations with those of the neighbor. Local and collaborative versions of modules are developed using Scan Contexts: point cloud descriptors, compact and sufficiently representative, used to save computational and communication resources. Errors on detection can occur due to perceptual aliasing: the set of loops is filtered by the Outlier Rejection module. Finally the resultant loops, together with odometry information, are processed by the Distributed Mapper which optimizes trajectories and updates the maps of the entire team, reaching consensus on final estimation.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

In recent years, an old research topic became relevant: algorithms working on autonomous mobile robots are no longer unfeasible due to significant improvements on hardware possibly mounted on board.
This interest is justified since robot-based algorithms can be easily converted to work in many widely different areas from industry (particularly focused on logistics, figure 1.1) to defense (exploration in hostile environments, figure 1.2) or can be seen directly by users like in home cleaning applications.
The necessity of research in this field is supported by the market: 100.000 Automated Guided Vehicles (AGV) and Autonomous Mobile Robots (AMR) are shipped only in 2021 [1].
In such context, Simultaneous Localization and Mapping (SLAM) is usually adopted to resolve a well known problem: runtime **localization** of the robot and **mapping** of the environments.
Regarding localization, a robot observes environments with respect to itself [2] evaluating the so-called pose. The algorithm is performed without considering any supplementary contributions usually used in this context, involving satellite technologies like GPS. The independence from this source of information is a feature of SLAM algorithms: the robot is able to move even in indoor or subterranean environments where the signal is badly reached.
Later on, while computing his localization, the robot can combine its position with environment objects captured by sensors, in a common global frame, providing a map.
These two elements (pose and map) are essential to enable robust navigation since they are involved in path planning and collision detection algorithms.
Leveraging a fleet of robot executing a SLAM task, the computational time required to calculate final estimation can be significantly reduced.
Such collaboration can be enabled by exploiting a single centralized node of computation, which merge information collected by the whole team, or by distributing

the whole workload on the single components of the fleet. Although collaboration has obvious advantages, it introduces many problems related to communication between robots which complicates the entire analysis increasing parameters to be chosen.

In this thesis work, a collaborative solution based mainly on lidar sensors elaboration is designed. Then, a initial development is provided implementing the base modules involved during the first cooperation phase.

Figure 1.1: Mobile robots in logistic application [3]

Figure 1.2: Mobile robots applied in the military [4]

The entire modeling and developing process is carried out in collaboration and supervision with **LINKS Foundation** centre of research.

Thesis is structured as follows:

- **Chapter 2**: *State of art analysis.* Single robot and collaborative SLAM problems are formulated and a brief overview of two complete solutions is provided.
- **Chapter 3**: *Distributed SLAM design.* Sensor suite to be used is selected and collaborative approach is chosen between centralized or decentralized. Starting from these assumptions a complete model is structured by connected modules. For each of these an algorithm overview is discussed, in order to be evident how libraries are developed to ease development process of modules.
- **Chapter 4**: *Development and Result.* Three modules are implemented and tested, referring to the first phase of collaboration during a rendezvous from two robots.

Chapter 2

State of the Art analysis

Collaborative SLAM (C-SLAM) has become in recent years not only a research topic but, thanks to the improvement on communication and computation resources, a reality in actual industries. Many solutions are developed having different sensor suites and algorithms elaborating them.
C-SLAM approaches derive directly from the stand-alone version, hence, many of the concepts explained in the first part of the chapter, related to the single robot SLAM, will be detailed while presenting the collaborative one.
In the second part, collaborative version are the main subject where architecture and different approaches are explained and compared.
Most famous collaborative solutions are presented at the end of chapter.

2.1 SLAM Single Robot

SLAM algorithm's objectives are the runtime joint estimation of the robot's state and a suitable representation of surrounding environment.
Robot state is generally represented by its pose (combination of position and orientation) with respect to a coordinate system. The environment (where robots are located) can be represented through maps made by landmarks generated using perceptive sensors like cameras or lidars.
SLAM solutions are essential in contexts of mapping of unknown environments and collision-avoidance where external source of informations usually involved in localization (e.g. GPS) can not be reached.

2.1.1 Problem definition

The problem can be summarized through map and state maximization [5].
By considering a variable X summarizing state and map landmarks and variable Z

as the data coming from sensors, the likelihood below is calculated at each step:

$$p(X|Z) \tag{2.1}$$

Since the first estimation as result of (2.1) is not enough to give acceptable results, robot states must be calculated through an optimization step.
It can be approached by two methodologies: estimation can be calculated every time step (**filtering**) or through the entire robot's route (**smoothing**). The latter in recent years has become popular in research compared with the former.
In the *smoothing formulation*, the problem becomes a *Maximum a Posteriori* (*MAP*) estimation, where a prior distribution (given by odometry) is introduced and it is used to solve the maximization problem.
Giving single robot α, the solution of MAP estimation problem is finding X_α^*

$$X_\alpha^* \doteq \arg\max_{X_\alpha} p(X_\alpha|Z_\alpha) \tag{2.2}$$

Applying Bayes theorem:

$$X_\alpha^* \doteq \arg\max_{X_\alpha} p(X_\alpha|Z_\alpha) = \arg\max_{X_\alpha} p(Z_\alpha|X_\alpha)p(X_\alpha) \tag{2.3}$$

where:

- $p(Z_\alpha|X_\alpha)$: likelihood of finding Z_α (*measurements*) giving a certain X_α
- $p(X_\alpha)$: prior distribution referred above, for example is referred to odometry information.

So optimization must find the set of variables X_α that better reproduces Z_α measurements, giving a prior distribution.

2.1.2 Architecture

It is possible to underline two major sub-problems: one related to the elaboration of sensor measurements and other about optimization. Although the division is often blurry (mainly in code implementation, a tight separation may produce redundancy), many solutions try to follow it calling **Front-End** the first task and **Back-End** the latter.
Concept is shown in figure 2.1.

- **Front End**: elaboration of measurements. Perception-related tasks such as feature extraction and data association: processing data in order to calculate ego motion and loop closure detection
- **Back End**: solve optimization problem using front-end estimation. Algorithms core are referred to graph and probability theory, robot trajectory is generated as a graph of poses.

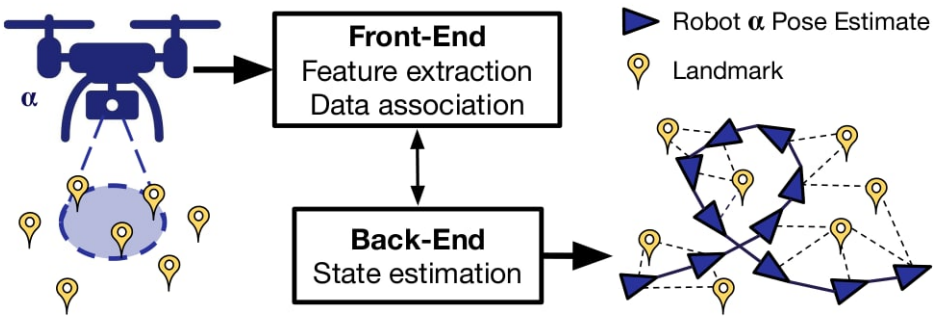

Figure 2.1: Local SLAM architecture [5]

2.2 Collaborative SLAM

A single mobile robot may take long time to produce good results during mapping and localization estimation. A fleet of robots can be configured to resolve a common localization and mapping problem, in order to perform tasks efficiently saving useful time to be employed in other operations.
Collaborative SLAM aim is to merge all data coming from robots in a single global consistent result, so a robot can rely on the experience of the entire team. This approach introduces challenges to be added to previous problems explained: communication between robots become central in all possible solutions.
The problem is challenging and many design parameters can be chosen introducing variety on solutions.
A first set of parameters is **team-dependent**:

- **Size**: as intuition may suggest, as the size of the robot team increases, the results should be obtained faster. However, the choice of this parameter is not so simple, team size determination depends also on other parameters such as robotics characteristics (homogenous or heterogenous composition), robot initial position knowledge or dynaminicity of ambient [6].
- **Composition**: all robots in a team are composed by the same (homogenous) or different (heterogeneous) hardware suite.
- **Team unit processing ability**: collaborative algorithms are computationally expensive, so decisions depend also on hardware of a single unit.

Other parameters are **communication-dependent**:

- **Range**: how many times two or more robots has the possibility to communicate, depending on the communication protocol (e.g *wifi*, *bluetooth*,..) and its mutual position.

- **Topology**: how robots are distributed in the working area.

- **Bandwidth**: crucial in determining how much data two robots can exchange on the channel.

2.2.1 Problem definition

The problem definition is a natural extension from the single-robot case.
In this context, considering a rendezvous from robot α and robot β, not only **local measurements** must be considered (named by Z_α and Z_β) but also **inter-robot measurements** $Z_{\alpha\beta}$.
Maximization final product is given by X^*_α and X^*_β
If the starting distribution is known, indicated by probability distribution $p(X_\alpha, X_\beta)$
It is possible to apply Bayes theorem likely to the local definition:

$$\begin{aligned}(X^*_\alpha, X^*_\beta) &\doteq \arg\max_{X_\alpha X_\beta} p(X_\alpha, X_\beta | Z_\alpha, Z_\beta, Z_{\alpha\beta}) \\ &= \arg\max_{X_\alpha X_\beta} p(Z_\alpha, Z_\beta, Z_{\alpha\beta} | X_\alpha, X_\beta) p(X_\alpha, X_\beta)\end{aligned} \tag{2.4}$$

If initial guess $p(X_\alpha, X_\beta)$ is not available, the problem has not unique solutions since infinite possibilities on initial guess can be considered.
In the latter case, the definition becomes more general and C-SLAM is formulated as a Maximum Likelihood Estimation (MLE) problem:

$$(X^*_\alpha, X^*_\beta) \doteq \arg\max_{X_\alpha X_\beta} p(Z_\alpha, Z_\beta, Z_{\alpha\beta} | X_\alpha, X_\beta) \tag{2.5}$$

2.2.2 Centralized, Decentralized and Distributed

In general, in a multi-robot algorithm a distinction between a **local** and a **global** perspective has to be considered:

- **Local perspective**: pose and landmarks of the map calculated in the robot local coordinate system (during the path are different between each other).

- **Global perspective**: final estimation to give to the robots navigation tool, referred to a global and common coordinate system.

Global view can be reached using different approaches: **centralized** and **decentralized**.

Centralized approach

In centralized solutions a single agent, high computationally equipped, called *estimator* computes the results and share it to robots in the team. In order to calculate consistent results, the estimator must have constant access to raw or elaborated measurements of the entire team.
The figure 2.2 shows a centralized SLAM architecture.

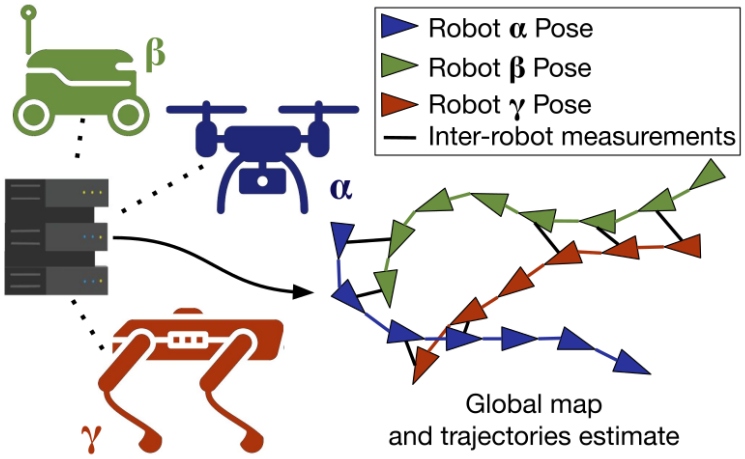

Figure 2.2: Centralized Collaborative SLAM approach [5]

Choosing this approach in design, the benefits are:

- Algorithms **complexity** contained
- **Single unit workload** contained

And drawbacks:

- **Security**: in other contexts different to LAN, if the estimator is attached from a malicious agent the robot does not have enough information to move alone
- **Bandwidth**: network type and topology is constrained since single robots information must be always available
- **Scalability**: centralized algorithms does not scale increasing number of robots in team
- **Single Point of failure**: if the estimator breaks, since it should be expansive, It is not easy to replace

Considering all drawbacks listed, recent research moves to decentralized solutions.

Decentralized approach

In a decentralized approach, each robot in a team has access only in its local view, during rendezvous neighbors share partial information in order to calculate common global view. Using this approach, final estimation can not be reached for all robots at once, but it is built **iteratively until consensus**.
This approach results to be more *scalable* on the number of robots, *secure* and works even with *low bandwidth* but implies higher **algorithm complexity**.
The figure 2.3 shows decentralized architecture.

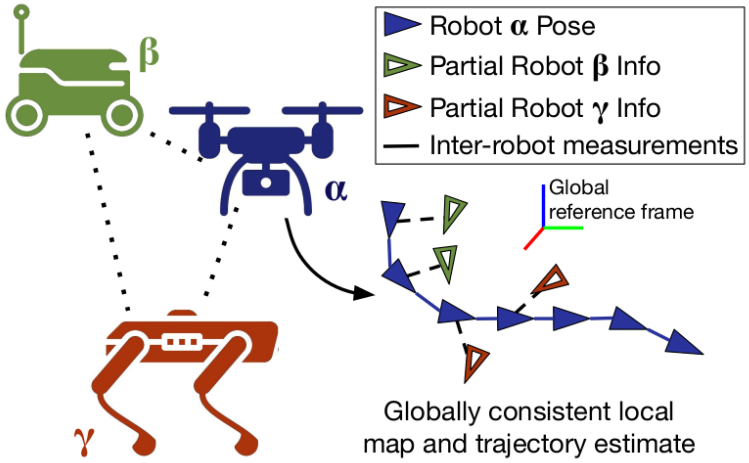

Figure 2.3: Decentralized Collaborative SLAM approach [5]

Besides decentralized and distributed are different concepts, they are often confused as synonymoun.
Distributed refers to computation resource distribution where can be located on different units. Using decentralized and distributed approach, a single unit must hold heavy computation.
Collaborative SLAM architecture follows the structure proposed in the single robot version with the division in **Front End** and **Back End**.

2.2.3 Front End

Front end role is to provide landmarks estimation, odometry measurements, and perform loop detection.
It must recognize translation and rotation during robot movements (odometry): processing inertial data, based on IMU sensors measuring wheel movements, match consecutive images coming from cameras or laser scans captured from lidars. Results

based on inertial sensors (e.g. IMU) are less accurate but computationally less expensive, other based on perceptive sensors gives better results but in larger processing time and does not work with all hardware compositions on board of robots.
Front end is composed by a perception-based algorithms involving **loop closure detection** based on *place recognition*. Loop closures are portions of path previously visited by robots (figure 2.4), the detection is critical for optimization in order to perform correction in the odometry already calculated. At the optimizer the loops are given in the form of relative pose transformations between two poses.
Algorithm in the local version is called **Intra-robot loop closure detection**.

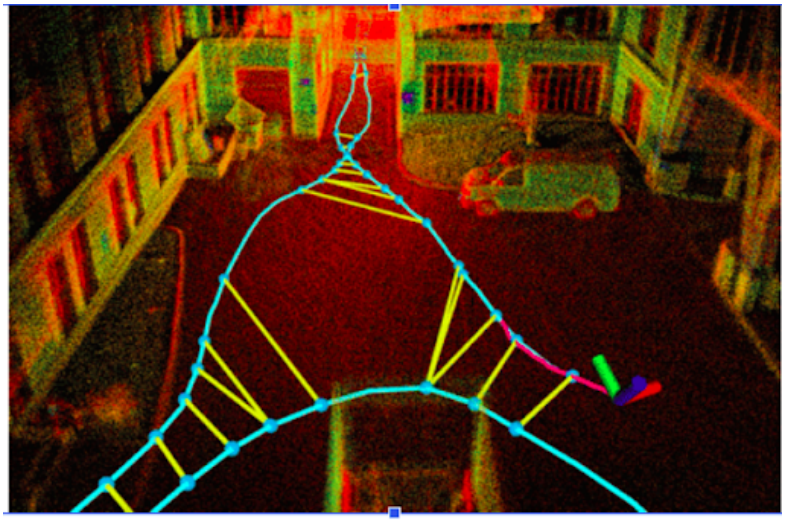

Figure 2.4: Loop Closure detection example [7]

Inter-robot loop closure detection is the collaborative version of the algorithm. Using a decentralized approach, two robots, during rendezvous, must mutually recognize if the neighbor already crossed its visited positions. Otherwise, in a centralized approach, detection is done by a single unit.
Detection can be **direct** or **indirect**:

- **Direct inter-loop closure detection**: two robots use direct sensing when they physically met in the same location. Loop closure detection is performed and relative transformation is calculated and sent back.
- **Indirect inter-loop closure detection**: loop detection is done looking for possible overlaps between the maps build by two distinct agents. This approach is particularly challenging since a lot of complex data (e.g. 3D point clouds or images) must be possibly exchanged by robots, so many improvements can be done in order to reduce these exchanges and optimize detection of loops.

Algorithms include a first part of place recognition: from raw complex data is calculated a descriptor, then it is used to identify a set where loop can be possibly found, denoted as candidate set. After place recognition, geometric estimation filters set of real loops form the candidate and calculate relative poses.

2.2.4 Back End

Back End takes as input Front End measurements and optimizes pose and map during robot movement (similar to local single robot case) taking into account inter-robot information calculated, ensuring consensus in the final estimation in every robot composing the team.
Two main approaches in literature are briefly explained.

Filtering-based Estimation

Current pose of robot is calculated without considering all the previous estimations, so the result depends only from data coming at time $t-1$ and the current observation at time t.
From all the optimizers, those based on **Extended Kalman Filter** (EKT) are the most used: complex optimization problem is reduced and solved using a local linear approximation. The linearization leads to errors when noise is too large.
Other solutions are based on **Rao-Blackwellized Particle Filters** exploiting Particle Filters concept: prediction of robot state is evaluated using a set of weighted samples called particle. Those weights and sets of samples are updated in function of successive predictions.

Smoothing-based Estimation

Optimization techniques where past measurements are considered, improving accuracy and efficiency.
Considering two robots α and β, a pose-graph formulation is useful to explain smoothing-based solutions and it is used as a starting point by different optimization solutions.
In this common formulation, map landmarks are indicated by nodes of a graph, each node is formed by a single odometry or loop closure measurements.
Assuming that noises in the measurements are uncorrelated (multiplication of all uncorrelated probabilities is allowed), it is possible to rewrite the MLE formulation in (2.5) as follows:

$$
\begin{aligned}
(X_\alpha^*, X_\beta^*) &\doteq \arg\max_{X_\alpha, X_\beta} p(Z_\alpha, Z_\beta, Z_{\alpha\beta} | X_\alpha, X_\beta) \\
&\doteq \arg\max_{X_\alpha, X_\beta} ((\prod_{i=1}^{l} p(z_\alpha^i | X_\alpha^i)(\prod_{j=1}^{m} p(z_\beta^j | X_\beta^j) \\
&\qquad (\prod_{k=1}^{n} p(z_{\alpha\beta}^k | X_\alpha^k, X_\beta^k))
\end{aligned}
\tag{2.6}
$$

where:

- i,j: measurements number of robot α and β.
- $p(z_\omega^z$: likelihood of the z^{th} measurements in the robot ω on a subset of variables X_ω.
- $p(z_{\alpha\beta}^k | X_\alpha^k, X_\beta^k))$: likelihood of the k^{th} iter-robot measurements on a subset X_α and X_β.

Assuming that noise in measurements are distributed like a zero-mean Gaussian noise with matrix Ω, each likelihood can be expressed:

$$
p(z_\alpha^i | X_\alpha^i) \propto exp(-\frac{1}{2} || h_\alpha^i(X_\alpha^i) - z_\alpha^i ||_{\Omega_\alpha^i}^2)
\tag{2.7}
$$

where h_α^i maps state with measurements.
Final aim is to maximize the likelihood, that is equivalent to minimize the negative log-likelihood. So maximization can be expressed like a nonlinear least squared problem:

$$
\begin{aligned}
(X_\alpha^*, X_\beta^*) \doteq \arg\min_{X_\alpha, X_\beta} -log((\prod_{i=1}^{l} p(z_\alpha^i | X_\alpha^i)(\prod_{j=1}^{m} p(z_\beta^j | X_\beta^j) \\
(\prod_{k=1}^{n} p(z_{\alpha\beta}^k | X_\alpha^k, X_\beta^k))
\end{aligned}
\tag{2.8}
$$

Replacing (2.7) and rewriting the expression:

$$
\begin{aligned}
(X_\alpha^*, X_\beta^*) \doteq \arg\min_{X_\alpha, X_\beta} ((\sum_{i=1}^{l} || h_\alpha^i(X_\alpha^i) - z_\alpha^i ||_{\Omega_\alpha^i}^2) + (\sum_{j=1}^{m} || h_\beta^j(X_\beta^j) - z_\beta^j ||_{\Omega_\beta^j}^2) + \\
(\sum_{k=1}^{n} || h_{\alpha\beta}^k(X_\alpha^k, X_\beta^k) - z_{\alpha\beta}^k ||_{\Omega_{\alpha\beta}^k}^2))
\end{aligned}
\tag{2.9}
$$

This formulation can be used both with centralized or decentralized approach.

2.3 Complete solutions

In this final section a brief overview of most common solutions is presented. In particular:

- **LAMP**: lidar-based centralized solution
- **DOOR-SLAM**: cameras-based decentralized solution

2.3.1 LAMP: Large-Scale Autonomous Mapping and Positioning

LAMP is a collaborative SLAM system equipped with a sensor suite mainly based on lidar and cameras, enable it to work in complex and hostile subterranean environments. Considering this requirement, inertial informations (e.g. IMU) can not be exploited as main source for odometry computation, the uneven terrain makes wheel movement elaboration noisy. [8].
In the sudden environment are distributed known objects called *artifacts*: an image elaboration algorithm is involved in detection, then relative artifact positions with respect to the robot are calculated in order to be used in the creation of pose-graph structure.
The solution exploits a centralized approach: a base station elaborates final estimation of poses and map collecting all the robots measurements. Furthermore an operator interface is implemented where it is possible to manually manage pose-graph adding a loop closure or filtering noisy estimations.

Architecture

All robots are connected to a central station. If communication goes down, robot performs a stand-alone version of SLAM, final optimization is done when communication is recovered.
Single robot front end and back end produce G_i as pose graph, X_i as poses and P_i as a set of point clouds attached to every pose. During the communication phase tuple $(G_i, X_i, P : i)$ is sent to the base station.
Figure 2.5 shows architecture's structure.

Front End

Front End algorithms calculate relative robots pose estimations and relative transformation matrix regarding loop closure detected.

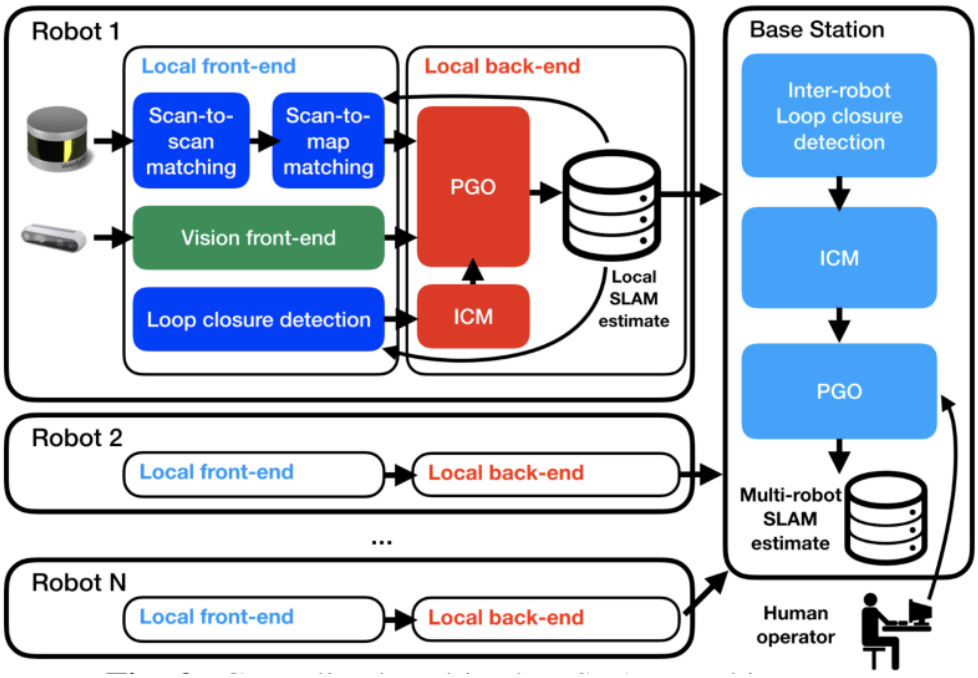

Figure 2.5: LAMP architecture [8]

- **Odometry**: **scan-to-scan** and **scan-to-map** matching.
  Scan-to-scan matching calculates ego motion of the robot, comparing the current point clouds coming from the 3D lidar sensor with the precedent. In order to evaluate relative position between the two poses of the robot, the point clouds are involved in a *Generalized Iterative Closed Point* (GICP) algorithm step: considering the two point clouds, it is iteratively searched the two closest point and then it is provided relative transformation matrix. In order to overcome bad efficiency in the results (due to environment characteristics), odometry is matched against a submap (subset of point cloud nearly inserted) performing a **scan-to-map** matching.

- **Artifact detection**: module detects artifacts and calculates relative position against the robot. Image detection is performed using *YOLO* deep learning algorithm on image, those underline an object with a 2D box. The relative position is calculated using depth information on RGB-D camera with respect to the center of the box.

- **Loop Closure detection**: in order to detect loops, current scan is compared to scans on P_i within a range. GICP algorithm is performed and fitness score is taken as reference: if this score is less than a threshold is considered as valid and passed to optimization.

Back End

Back end computes poses as result of optimization problem in graph-pose structure. Before optimization, the graph is filtered by outliers using Incremental Consistency Maximization algorithm.

- **ICM**: module checks quality of loop closures detected removing outliers. Here an optimization is done in the standard Pairwise Consistency Measurement (PCM) algorithm, in order to enable it to work online.
  Incremental Consistency Measurement (ICM) is divided in steps:

  – **Odometry check**: assuming a generic matrix transform **T** which represents relative pose measurements, when a loop closure is detected (T_{ij}^{lc}) a triangle can be indicated by the two poses (T_{ji}^{odom}) and the candidate loop. Considering the product of these matrices along the triangle, it might produce an identity matrix otherwise the loop must be discarded. Identity is assumed since the informations contained in both of matrices must be the same. Taking into account non-ideal conditions and noise, identity is a condition too restrictive, so a error matrix is calculated:

  $$T_{ij}^{err} \doteq T_{ij}^{lc} \cdot T_{ji}^{odom} \tag{2.10}$$

  where:

  * T_{ij}^{err}: matrix error.
  * T_{ij}^{lc}: candidate loop.
  * T_{ji}^{odom}: matrix obtained combining poses along the loop.

  The check is passed and loop candidate is considered as valid only if average error is less than a threshold.

  – **Paiwise consistency**: check is done between couples of loops following the same principles of the last explained.

  $$T_{ij,kl}^{err} \doteq T_{ij}^{lc} \cdot T_{jl}^{odom} \cdot T_{lk}^{lc} \cdot T_{ki}^{odom} \tag{2.11}$$

  First product is a generic loop candidate, the latter is a loop already counted on as valid. Each couple is added to an adjacency matrix creating a graph incrementally.

  Generic PCM is performed in this adjacency matrix which is a maximum clique search problem discussed in following chapters of this thesis.

- **PGO**: after initial pose graph structure is filtered by previous ICM step, pose graph optimization is performed using *GTSAM* algorithm both in the

local and collaborative versions. A new key-scan is added after odometry displacement and It is added in the pose graph as well as loop closure detected. In the base station, each graphs G_i are merged into a single one and, before optimization, inter-robot loops are detected using the same methodologies of the local version.

Human operator can interact in the node-graph creation, loop closure can be added manually using a ROS1 service.

2.3.2 DOOR-SLAM: Distributed, Online, and Outlier Resilient SLAM

DOOR-SLAM is a fully distributed multi-robots SLAM solution. Exploiting the system's functionalities, full connectivity is not required since collaborative algorithms are executed during a rendezvous. In the moment of no-connectivity from other units of the team, the single robot performs a local SLAM algorithm. The sensor suite is composed of stereo cameras builded in platform on board of drones. The system is developed using Buzz, a scripting programming language useful in multi-robots contexts. Exploiting its functionalities, communication details between robots (dependent on the particular type or model) can be neglected in order to focus the point only on SLAM algorithms. Buzz works on top of the ROS1 framework.

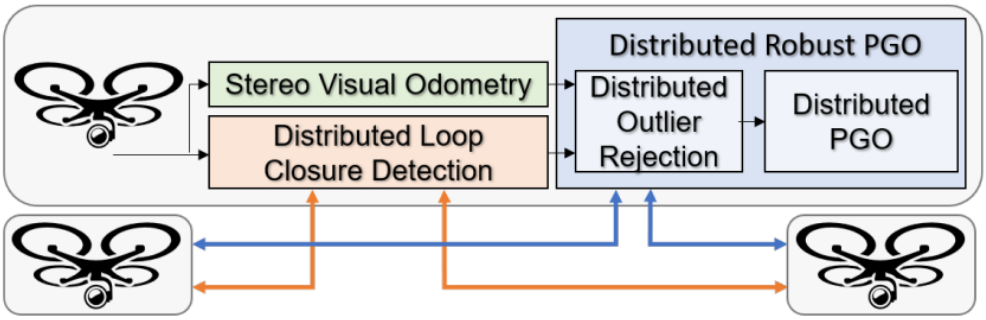

Figure 2.6: DOOR-SLAM architecture [9]

Architecture

System architecture is shown in figure 2.6.
In the solution only inter-robot loop closure detection is performed, so optimization comes only during rendezvous of two robots. This limitation can be overcomed if Stereo Visual Odometry is replaced with a complete single robot SLAM solution, so the robot can improve its estimates exploiting even intra-robot loop closures.

Images are collected by robots through stereo cameras, then odometry is calculated using RTAB-MAP (since it is a well-known module the authors do not provide any other supplement informations) and loop closures are detected.
In figure 2.6:

- **Distributed Loop Closure detection** (figure 2.7): it is formed by two submodules. The first is **place recognition** submodule: NetVLAD descriptors are calculated from images corresponding to each keyframe. During rendezvous, robot α sends those descriptors to the neighboard β. Robot β compares all the incoming descriptors with those collected during its movement. Through this comparison, robot β calculates candidate loop closures, where the Euclidean distance from the two indexes is below a given threshold. It is important to underline that candidate loop closures indexes are evaluated without exchanging raw images between robots and necessity to any pre-step training phase.
  The second is the **geometric verification** submodule: after candidate indexes computation, robot β sends to α the corresponding visual features associated to image indexed.
  Those features are useful to robot α to perform geometric verification performing *solvePnpRansac* function (provided by OpenCV): this function gives back a set of inlier features and relative pose transformation. Whether the number of inlier features is large enough, the candidate loop closure is considered as valid and the transformation matrix is passed to optimization.

- **Distributed outlier rejection module**: since the place recognition module can be biased by perceptual aliasing, some inter-robot loop closures are considered as valid even if they must be considered as outliers. The module exploits a distributed version of Pairwise Consistent Measurement Set Maximization (PCM) algorithm to detect those outliers. Since the library is used in the solution proposed in this work thesis, all the details are discussed in the following chapter.

- **Distributed PGO module**: the module creates a pose-graph structure from odometry and inlier inter-loop closures and performs a distributed optimization. Details are discussed in the following chapter.

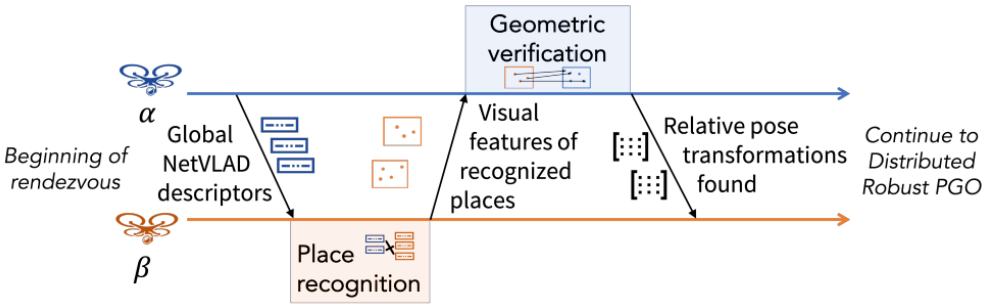

Figure 2.7: Inter-robot loop closure detection representation [9]

Chapter 3

Distributed SLAM design

In this chapter, the design of the distributed SLAM solution is presented. The design aspects are covered both with an extended discussion on deeper concepts already implemented by libraries. This is used in the development process as a starting point.
In the first part of the chapter, a ROS2 Overview is useful to highlight in which framework the solution must run.
In the second part the subject is the conceptual model: all the modules are analyzed pointing out relative inputs and outputs.

3.1 ROS2 overview

Robots, in order to accomplish tasks like navigation or mapping, need to be programmed. Not only its hardware components are important: without a software involved to the processing of information from sensors, a robot is useless [10].
Programming a robot from scratch by a coder could be a complex task, since many problems need to be solved: the robot operates into a dynamic and unpredictable world, many sensors and types of actuator are involved (each of these needs drivers to be integrated), etc.
In order to make the developing process easier, ROS2 (*Robotic Operating System*) framework is used: it is a middleware from operating system and user applications (figure 3.1), useful in the integration of software components. It provides drivers, libraries, development and monitoring tools and a common development methodology. It is not the only middleware used in the robotic field during years, but it grew up in popularity due to a wide developers community with open-source propension. Now it is the most used by research, organization, and by companies worldwide located.
ROS2 is the second version of the ROS1 framework of which shares many concepts.

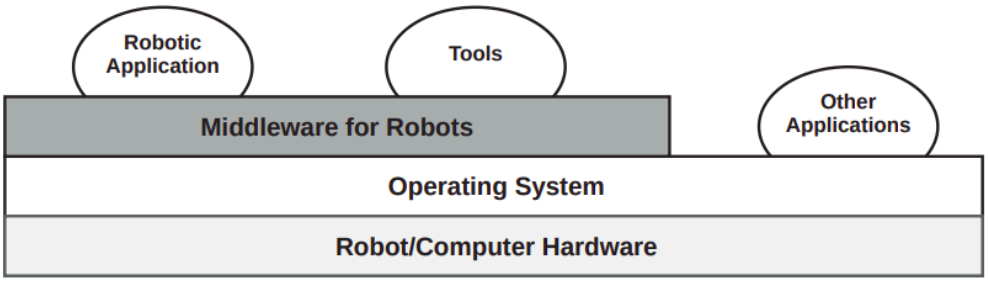

Figure 3.1: Software layers in a robot [10]

A complete new version of the middleware is due to the necessity of essential missing features (such as DDS standard for message definition and serialization), adding it directly into ROS1 could make applications already implemented as unstable.
ROS2 respects real-time, safety, certification and security requirements which makes it compatible with industrial applications [11].
Applications developed in ROS2 form a **computational graph**.

3.1.1 Computational Graph

A ROS2 computational graph is a network of elements which properly interacts with each other using different communication paradigms, in order to split complex tasks into simpler problems. It can be seen as what a robot software application sees during its execution.
A computational graph is composed by **nodes**, each one is the primary execution element in ROS2. A node follows object data structure and can be written in all of the two standard languages supported: C++ and Python.
During node bootstrap [12], it advertises its presence to neighbors in the same network (it is marked by ROS_DOMAIN_ID properly setted during installation). At the time when the other nodes respond to advertisement with its own information, a connection can be established: now the nodes can communicate. ROS2 nodes periodically advertise their presence and communicate when they go offline.

Communication by nodes can happen only if the two entities have compatible quality of service settings (**QoS**): QoS are policies useful to tune communication, which can be reliable as TCP or fast like UDP.
All nodes use ROS2 client library (**rclcpp** in C++, **rclpy** in Python) to communicate with other nodes through different communication methodologies:

- **Publisher/Subscriber**: asynchronous communication. Nodes create different messages due to data type exchanged (message type format can be custom

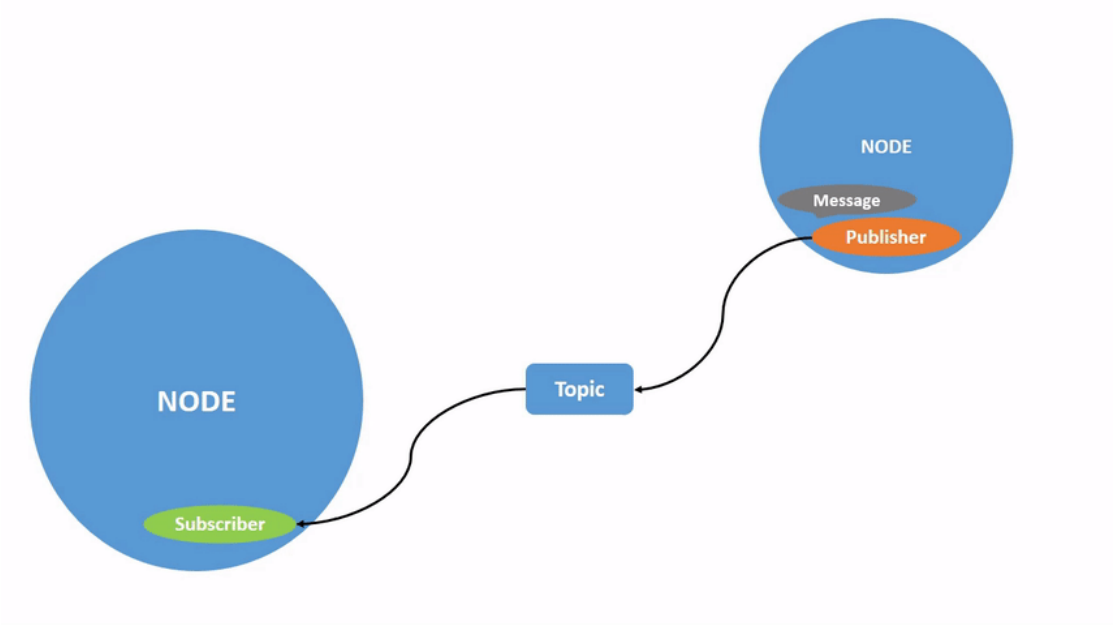

Figure 3.2: Publisher node publish a message to a topic, received by Subscriber node [12]

or standard, the latter is recommended since it can be easily used by the community). Publisher nodes send messages through a communication channel called **topic** (figure 3.2). Each topic is characterized by a name, necessary to those subscribers who want to receive messages from the publisher. Multiple nodes can be publisher or subscriber of a single topic.

- **Services**: synchronous communication. A node called client sends a request of information/data, a node called server responds with the message resultant.
- **Actions**: asynchronous communication. They are similar to service servers: a node client makes a request (similarly to services) to a server. This one can not be fulfilled immediately since it requests some computational time, so the server periodically sends back a feedback to the client with the process status until the result is ready.

3.1.2 Nav2 Project

The nav2 project takes the heredity of the ROS1 navigation stack [13]. Nav2 includes useful tools to make robots move from a point A to a point B: those tasks are accomplished by navigation, dynamic path planning, motor velocities computation, obstacles avoidance and so on.

Nav2 is organized by several particular nodes: **lifecycle nodes**. In those action servers are called to accomplish one of the tasks mentioned before. Lifecycle nodes are organized in a tree-structure called **behavior trees**. In figure 3.3 it is shown a representation of the package architecture.

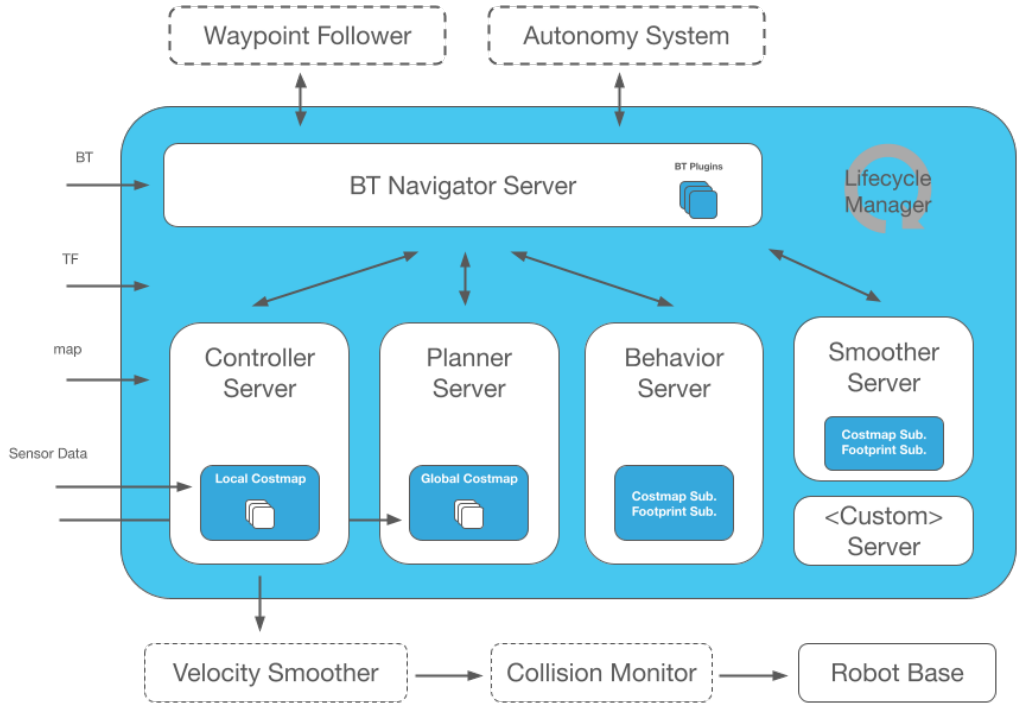

Figure 3.3: Nav2 package software architecture [13]

Lifecycle Node

Type of unique nodes in ROS2. Using those nodes it is possible to manage the life of a node like a state machine during phases like startup and shutdown. They allows to:

- Force deterministic behavior during those two phases.
- Help to design better software structures, useful during the debugging process.

At the startup, the node is in an **unconfigured state**, the object enters into constructors and no one of the parameters are read. The node switches to the configuration stage whether the $on\_configure()$ method is triggered.
If the $on\_active()$ method is called the node is in the **activation stage**: here the ROS2 network interface is activated and the task is ready to be accomplished.

When a node shutdown is triggered, the node switches in succession on **deactivation**, **cleaning up**, **shutting down** state.
In the community the usage of these nodes is recommended to overcome indeterministic behaviors even if their design is quite recent.

Behavior trees

Behavior trees are a tree-structure of tasks to be accomplished, typically used in complex robotics contexts.
It is a scalable and human-understandable framework useful in those contexts where a classical finite state machine (FSM) is chosen as an alternative: this one is prone to errors and it is hard to debug [14]. Since its modular structure of components, it is possible to encourage code recycling.

Navigation Servers

Navigation task is accomplished by four different servers activated by robot during a run calling the corresponding Behaviour Tree node: **planner**, **controller**, **smoother** and **recovery** servers (this is used only to recover the robots in trouble situations).

Planners Planner server computes a path considering some constraints: path can be created to go in a given position (plant to a goal), to complete coverage (occupy all the free space in the environment) or to compute path having sparse pre-calculated routes.

Controllers Controllers server try to follow a given calculated path or complete a local tasks. It has access to local representation of environment data to compute a suitable update iteration to give at the actuators of the robots.

Smoothers Path planning can be further improved by successive refinements based on optimization criteria: a path pre-calculated can cross near many obstacles, a smoother can reduce the probability of collision. Smoothers can be useful also in the presence of multiple planners, in order to combine all the output results.

3.2 Model

The solution modelled to resolve the Collaborative SLAM problem follows the typical structure that can be found in the recent research. A graphical representation is shown in figure 3.4.
The model is created following a concrete path of decisions. At first, centralized

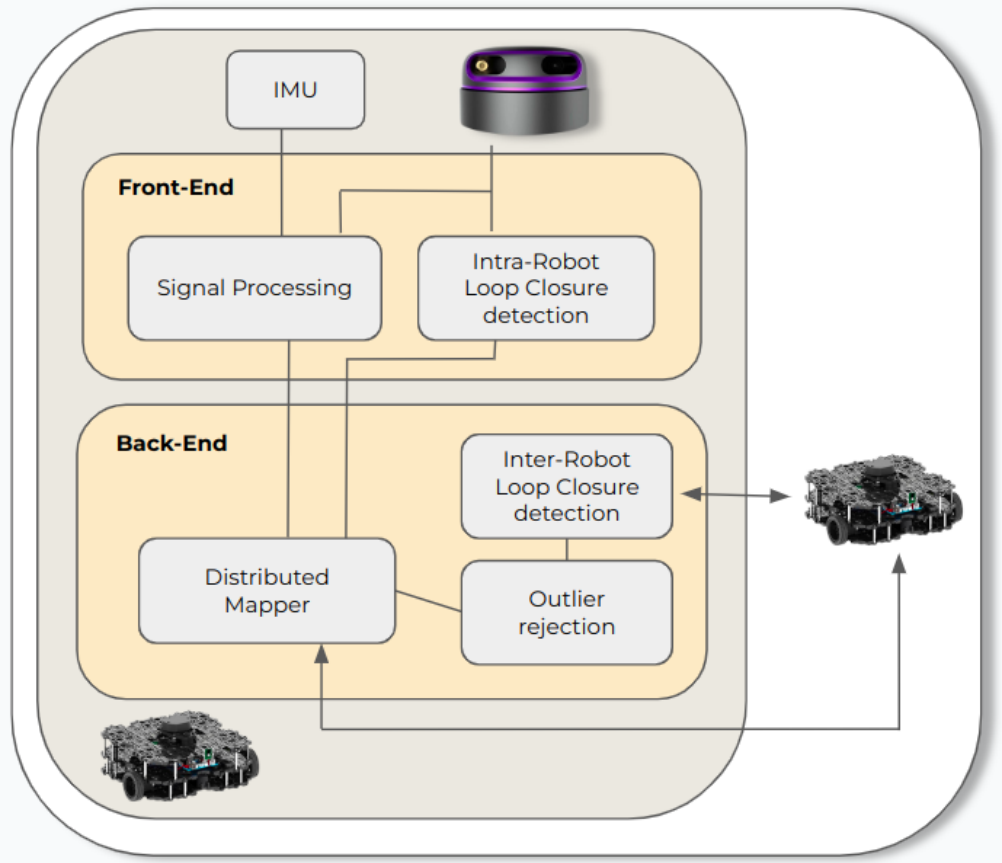

Figure 3.4: Lidar-based Distributed Simultaneous Localization and Mapping model

or distributed approach must be chosen according to what have been discussed in Chapter 2. Even if centralized one could be a good choice due to low computational resources tipically equipped in mobile robots, distributed approach is chosen based on:

- better scalability in the number of robots.
- bandwidth constraint.
- security constraint.
- lack in lidar-based distributed solution.

The sensor suite is composed by lidar sensors and (optionally) IMU measurements. Usually in Collaborative SLAM solutions already implemented different primary sensors are used: vision stereo cameras are chosen rather than lidar sensors. Algorithms based on image processing are commonly used since SLAM sub-processing

requests place recognition features, in this solution this task becomes challenging since lidar feature extraction is a relatively new research topic.

In this solution, lidar data manipulation is stressed in order to:

- contribute with new algorithms to the C-SLAM problem
- try to give a compatibility with local SLAM tooblox, based on 2D lidar scan
- try to deal with less computation: image processing requests high computational unit.

IMU data processing is optionally used in order to calculate point cloud adjustment and a downsampled representation used to safe both computational and bandwidth resources.

In model 3.4 two subgroup can be visualized:

- **Front End**: takes as input sensors raw data and performs early manipulations.
- **Back End**: considering data coming from front end, it performs optimization in two phases: build of graph-poses structure to be given to optimizer, optimization algorithm performed by it.

In the **front-end**, the modules:

- **Signal Processing**: raw data from lidar and imu is taken, odometry information about the robot (ego-motion of the robot) is calculated using scan matching algorithm from point clouds. Odometry evaluation is mainly implemented exploiting variants of Iterative Closest Point algorithm developed in the PCL library [15].
  A downsample representation of point cloud is calculated based on corner and surface features are developed useful for future updates of perception modules.
- **Intra-Robot Loop Closure detection**: Loop closures are portions of paths previously visited by robots, the module proposes a detection algorithm fully based on point clouds. Feature descriptors called Scan Context [16] are calculated, the detection is then performed in two successive phases using the current point cloud: scan contexts matching and scan matching.

In the **back-end**, the modules:

- **Inter-Robot Loop Closure detection**: during a rendezvous two robots must detect mutual loop closures. Sharing scan context descriptors, it is possible to develop a distributed version of the algorithm without stressing the bandwidth constraint.

- **Outlier Rejection**: filtering step in order to overcome aliasing.
- **Distributed Mapper**: given odometry and loop closures, it estimates poses corrected performing a distributed algorithm.

In the relative subsections all of these modules are analyzed, pointing out functionalities offered by library used in implementation. In the chapter 4, the developed modules and relative testing is presented.

Pose-Graph representation

Except from Distributed Mapper, all other modules aim is to create a pose-graph representation where the relative poses of the robots are optimized and extracted. This pose-graph, used as starting point in optimization, is in the form of **Factor Graph** [17]. This representation is popular since it is used by one of the most common optimizer in single-robot contexts called GTSAM [18].
Factor graph is a representation where unknown **variables** $[X_1, X_2, \ldots]$ are connected to **factors** (figure 3.5), that mantein probabilistic information.
In the SLAM context:

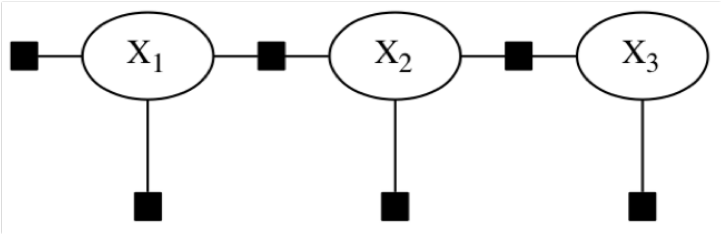

Figure 3.5: Factor Graph representation [17]

- **Variables** are the poses of robots with progressive time instant.
- **Factor** between those poses is expressed through the matrix transformation associated. Those matrices express consecutive pose expressions given by odometry or loop closure factors detected referred to poses calculated in different time instant. The noise in the estimation of those matrices is suitably modelled and inserted in the factor information.

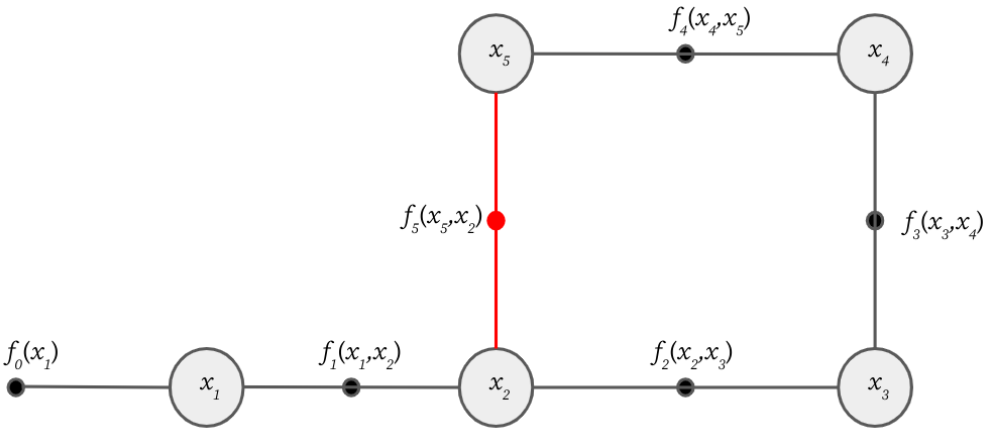

Figure 3.6: Factor Graph for SLAM

Those concepts are showed in figure 3.6 where:

- $f_0(x_1) =$ **prior** of x_1, it is the initial value of pose. It is common to consider null pose where a model of noise is applied.
- $[x_1, x_2, \ldots]$: poses estimates during robots movement.
- $f_k(x_k, x_{k+1}$: odometry estimates between poses at instant k and $k+1$.
- $f_5(x_3, x_5)$: loop closure detected between poses x_5 and x_3

3.2.1 Signal Processing module

Signal Processing module elaborates raw data coming from lidar and IMU sensors. It calculates odometry and a downsampled representation of point cloud based on surface and corner features used in future upgrades of successive modules (figure 3.7).

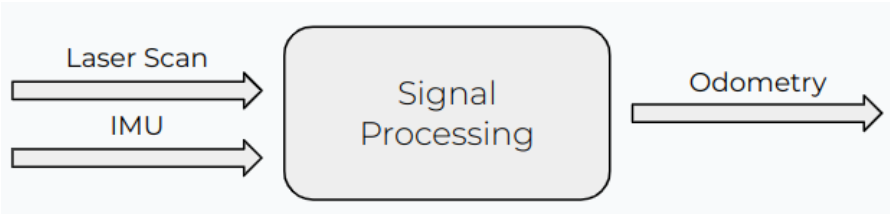

Figure 3.7: Signal Processing module representation

The odometry is calculated performing scan matching algorithm using consecutive point clouds. Those algorithm are referred to **registration** section of *Point Cloud Library*, where algorithm based on **Iterative Closest Point** are provided. In the implementation of the module, two version of scan matching (**Generative Closest Point** and one based on **Normal Distribution Transform**) is implemented and resultant information is compared using a **score** metrics as reference. In the following, an insight of the two algorithms is presented.

Iterative Closest Point and Generalized Iterative Closest Point (GICP)

Iterative Closest Point belongs to algorithms group based on Registration, properly developed in Point Cloud library [15]. Point Cloud library is a massive open source project with many useful tools for elaboration of 3D point clouds.
Registration is a technique that provides alignment between different point clouds (figure 3.8) in the form of a transform matrix in a global reference coordinates [19]. Elaborating this result, It is possible to calculate relative pose between keyframes where point clouds are captured by sensors.
Registration task is necessary not only referring to SLAM but also as pre-processing of data in computer vision before segmentation or object recognition.

Problem formulation

A point cloud is a suitable representation of groups of multi-dimensional points $\mathbf{p} \in \mathbb{R}^n$; those points take information about a sampled surface: cartesian information (n=3), about curvature or normal surface (n>3).
The registration problem finds correspondences between points in point cloud source P and point cloud target Q, the first formed by points $\mathbf{p} \in P$ and the latter by points $\mathbf{q} \in Q$, forming couples (p_i, q_i).
Later on finding all correspondences, a transformation $\mathbf{T}$ is estimated between all the pairs. Using this matrix $\mathbf{T}$ and applied to P, the point cloud resultant is similar to Q as much as possible. Those correspondences are found during the registration algorithm, usually they are not known.
Registration algorithms are divided in two classes:

- **Feature-based registration**: before performing alignment, geometric feature descriptors are computed in order to find better set of correspondences.
- **Iterative registration**: it does not perform any previous computations, candidates correspondence is taken from the closest mutual points in P and in Q, evaluated on cartesian space. Transformation T is computed by minimization

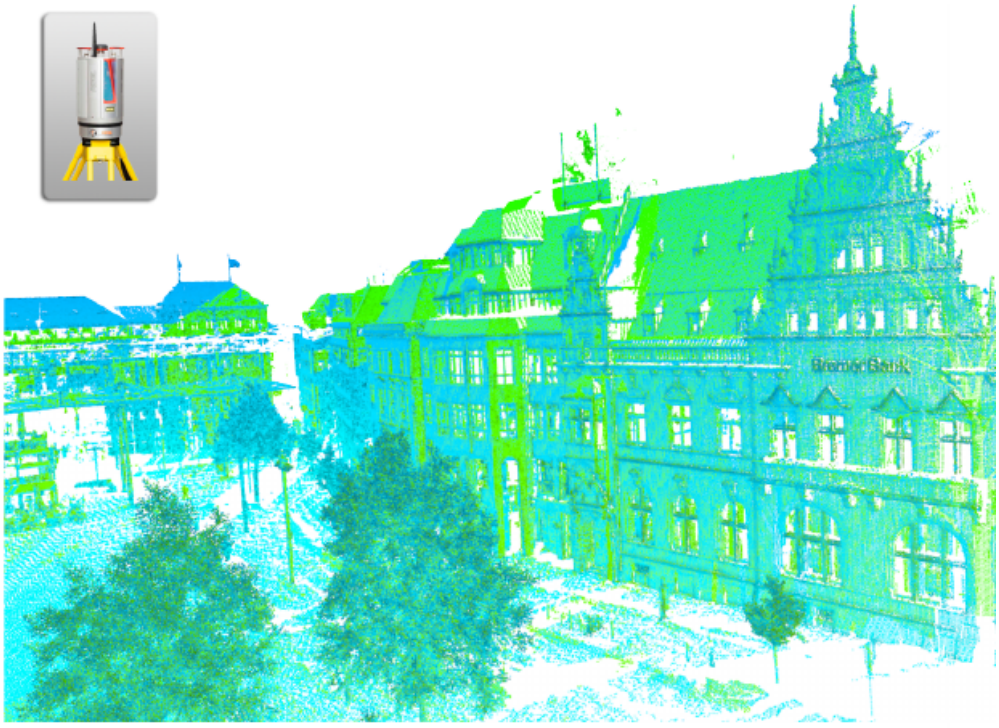

Figure 3.8: Alignment resultant by Registration between couple of point clouds (blue and green) [19]

of distances between those pairs of closest points exploiting a least squares approach. Final estimation is not reached at once, source point is expected to be aligned to target iteratively until convergence.

The latter approach is used in ICP and it is discussed below with relative variations.

Algorithm

The approach is iterative: correspondences between source and target points are found without calculates feature descriptors. Since minimization of distances between P and Q is a non-convex problem, the solution can collapse into a local minima. The algorithm is divided in many steps, in order to overcome to non-convexity and speed up the estimation.

```
INPUT:
- Point Cloud P (source) and Q (target).
- Initial transformation T_0
OUTPUT:
- Transformation T, which aligns A to B
```

```
6
7  while not_converged do
8      for i=1 to N do
9          m_i = FindClosestPointInS(T b_i);
10         if !! m_i - T b_i || <= d_max then
11             w_i = 1;
12         else
13             w_i = 0;
14         end
15     end
16
17     T = argmin{Optimization metric}
18 end
```

Algorithm 3.1: ICP or GICP algorithm

Selection Raw lidar data manipulation results to be quite computationally expensive due to density of information: point clouds are a dense representation of the environments. In order to properly estimate the relation between two point clouds, much of this information is redundant. So source and destination point clouds are sampled following a given distribution.

Matching In order to make connections between points p_i in source point cloud P and the closest points q_i in Q, a matching algorithm is performed. At the end of the step the couples (p_i, q_i) are formed.
Matching can be performed using an iterative approach: for each of points p_i search the closest in Q. Although this is the simplest solution, it is never used due to long computational time. The alternative algorithm is based on supplement data structures: **kd-tree** or **octrees**.
The algorithms complexity, later on an initialization phase $O(NlogN)$, becomes $0(logN)$, where N is the number of points in P.
PCL library usually uses KD-tree implementation based on FLANN library [20]. This data structure is discussed later on of this work thesis, during implementation of Loop Closure detection (chap 5).
Matching algorithm developed in PCL can be used by user as 3.2.

- *source*: point cloud in P
- *target*: point cloud in Q
- *max_dist*: can be imposed max distance, beyond this line all the other points are discarded
- *determineCorrespondences*: API that calculates matching.

```
#include <pcl/registration/correspondence_estimation.h>
// code initialization of source and target point clouds is not
   presented

CorrespondencePtr corr (new Corresponcences);
CorrespondenceEstimation<PointXYZI, PointXYZI> est;

est.setInputSource (source);
est.setInputSource (target);
est.determineCorrespondences(*corr, max_dst);
```

Algorithm 3.2: Matching algorithm

Rejecting and Filtering Correspondences The rejection and filtering steps are necessary in order to discard invalid correspondences to speed up the convergence to the global minimum, since those spurious values can cause errors in the final estimation.
There are many rejection methods: all of them can be combined to form a pipeline structure, where the input of one rejector is the output of the precedent (Algorithm 3.3 shows typical use of a rejectors pipeline, where max distance and RANSAC number of iterations are fixed values).
Here an overview of rejectors is presented, considering only those used in the implementation or considered in the experiments.

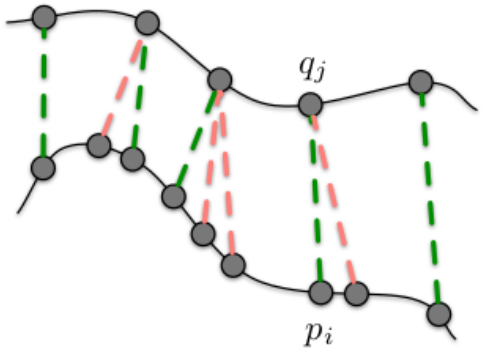

Figure 3.9: Correspondence rejection on multiple target points [19]

- **Correspondence rejection based on distance** (*CorrespondenceRejectionDistance*): Fixing a given distance threshold, point pairs are discarded whether the distance calculated is larger than this limit value.

- **Rejection based on median distance** (*CorrespondenceRejectorMedianDistance*): Rather than using a static threshold, it is possible to calculate all the distances and use the median of these values as a dynamic threshold. Median operator is more robust to outliers with respect to use the medium of distances.

- **Rejecting pairs with duplicate target matches** (*CorrespondenceRejectorOneToOne*): Correspondences are created from source points to target points, it might happen that a target point is joint with multiple source points. Listing all of these multiple correspondences $\{(p_i, q_j)\}$, only the one with the minimum distance $(p_{i_{min}}, q_j)$ is considered (figure 3.9).

- **RANSAC-based rejection** (*CorrespondenceRejectorSampleConsensus*): Random Sample Consensus algorithm is applied to subsets of correspondences: outliers are eliminated from the set calculating euclidean distances between points after that previous transformation matrix evaluated is applied to source's points.

- **Rejection based on normal compatibility** (*CorrespondenceRejectorSurfaceNormal*): 3D points can hold different information other than cartesian coordinates, one of these is the normal vector. This rejector method exploits the normal vector and reject the correspondence where it is not consistent.

- **Rejection based on surface boundaries** (*CorrespondenceRejectorBoundaryPoints*): considering a surface represented by point clouds, the boundary points introduces errors when these are considered as correspondence. The rejector filters those values (figure 3.10).

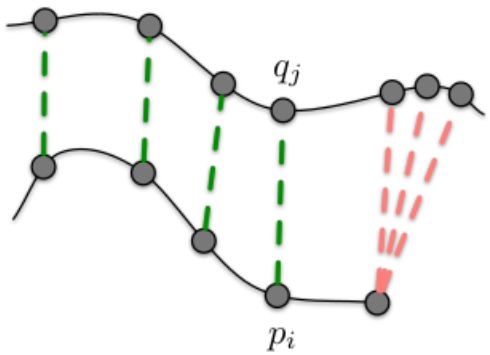

Figure 3.10: Correspondence rejection on boundary points

```
#include <pcl/registration/correspondence_rejection_distance.h>
// code initialization of source and target point clouds is not
   presented

CorrespondenceRejectorDistance rej;
rej.setInputSource<PointXYZI, PointXYZI> (source);
rej.setInputTarget<PointXYZI, PointXYZI> (target);
rej.setInputCorrespondences (n_corresponds);
rej.setMaximumDistance(max_dst);
ref.setRANSACIteration(n_iterations);
rej.getCorrespondences (corresponds_filtered);
```

Algorithm 3.3: Pipeline Rejection Algorithm

- *n_corresponds*: max number of couples accepted
- *max_dst*: max distance considered
- *n_iterations*: max number of iterations of RANSAC algorithm

Alignment and Error metrics The aim of alignment problem is to find transformation matrix T (pose and orientation) that minimizes alignment errors on correspondences found in the previous steps.
Considering point p_k in P, q_k in Q and matrix **T**, alignment error can be classified in two typologies: **point-to-point** and **point-to-plane** (figure 3.11).

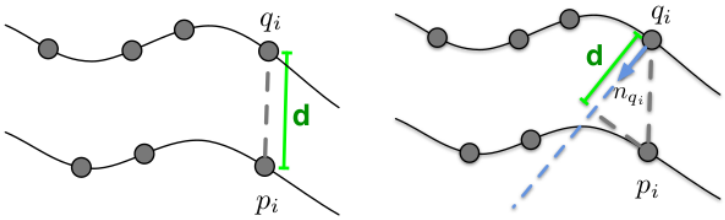

Figure 3.11: Point to Point and Point to Plane errors [19]

$$E_{point-to-point}(\mathbf{T}) = \sum_{k=1}^{N} w_k ||\mathbf{T}\mathbf{p}_k - \mathbf{q}_k|| \tag{3.1}$$

$$E_{point-to-plane}(\mathbf{T}) = \sum_{k=1}^{N} w_k ((\mathbf{T}\mathbf{p}_k - \mathbf{q}_k) \cdot \mathbf{n}_{\mathbf{q}_k})^2 \tag{3.2}$$

many optimization algorithm can be performed by choosing one of these errors (or both in the GICP case):

- **Point-to-point error metric**: based on (3.1), it is used in the first version of ICP. PCL uses a solution based on singular value decomposition (SVD).

- **Point-to-plane error metric**: normal surface information is used improving optimization performance [21]. With respect to the first, the optimization has not a closed form. Considering small angles between points, a linearization can be done ($sin\theta \sim \theta$, $cos\theta \sim 1$) enabling the use of linear solvers. Otherwise only non-linear solvers can be used.
  A similar algorithm exploits weight w_k in the algorithm. This variable can improve convergence, considering less determinant those correspondences that can make errors in the final result.

- **Linear least squares point-to-plane**: collect all correspondences in a matrix $\mathbf{A}$, poses and orientations are calculated solving least spare problem in the classic form $\mathbf{A}^t\mathbf{A}\mathbf{v} = \mathbf{A}^t\mathbf{b}$, where $\mathbf{v}$ = six-dimensional vector representing (x, y, z) coordinates and angles (*roll*, *pitch*, *yaw*).
  Similarly to point-to-plane error-base algorithms, also a least spares algorithm variation can exploit weight w_k to improve convergence in the final estimation.

- **Generalized error metrics and Generalized ICP** [21]: this approach resolves optimization problem by using a probabilistic model to minimization step of the algorithm 3.1, in order to create a general problem representation where both point-to-point and point-to-plane metrics can be derived.
  Starting from optimization step from 3.1, it is possible to consider two point cloud $A = \{a_i\}_{i=1,\dots,N}$ and $B = \{b_i\}_{i=1,\dots,N}$ with correspondences couples $\{(a_i, b_i)\}$. Those who does not pass $||m_i - T \cdot b_i|| > d_{max}$ check (where d_{max} is a fixed parameter) are discarded.
  It is built probabilistic model where set point points $\hat{A} = \{\hat{a}_i\}$ and $\hat{B} = \{\hat{b}_i\}$ are considered, where $a_i \sim N(\hat{a}_i, C_i^A)$ and $b_i \sim N(\hat{b}_i, C_i^B)$.
  $\{C_i^A\}$ and $\{C_i^B\}$ are covariance matrices referred to measured points.
  Considering perfect correspondences (couple of points are taken without errors in the sampling and filtering steps) and correct transformation $\mathbf{T}^*$:

$$\hat{b}_i = \mathbf{T}^*\hat{a}_i \tag{3.3}$$

  For any $\mathbf{T}$, it is possible to define the error as:

$$d_i^{(\mathbf{T})} = b_i - \mathbf{T}a_i \tag{3.4}$$

where $d_i^{(\mathbf{T}^*)}$ is contained. Using Maximum Likelihood Estimator (MLE) it is possible to compute iteratively $\mathbf{T}$ as:

$$\mathbf{T} = \arg\max_{\mathbf{T}} \prod_i p(d_i^{(\mathbf{T})}) = \arg\max_{\mathbf{T}} \sum_i log(p(d_i^{(\mathbf{T})})) \tag{3.5}$$

Since a_i and b_i are modelled as normal indipendent distributions, it is possible to rewrite the expression above as:

$$\mathbf{T} = \arg\min_{\mathbf{T}} \sum_i d_i^{(\mathbf{T})^T} (C_i^B + \mathbf{T} C_i^A \mathbf{T}^T)^{-1} d_i^{(\mathbf{T})} \tag{3.6}$$

This equation represents the point of Generalize-ICP algorithm: point-to-point or point-to-plane version can be reconstructed by changing covariance matrices C_i^A or C_i^B. The algorithm also allows to tune other version of the matrices to improve optimization performance.
As examples, here both of two representations (point-to-point and point-to-plane) are derived from (3.6).
The ICP point-to-point can be calculated considering:

$$\begin{aligned} C_i^B &= I \\ C_i^A &= 0 \end{aligned} \tag{3.7}$$

The optimization algorithm can be rewritten as below, representing the update step:

$$\mathbf{T} = \arg\min_{\mathbf{T}} \sum_i d_i^{(\mathbf{T})^T} d_i^{(\mathbf{T})} = \arg\min_{\mathbf{T}} \sum_i ||d_i^{(\mathbf{T})}||^2 \tag{3.8}$$

The ICP point-to-plane can be calculated considering:

$$\begin{aligned} C_i^B &= \mathbf{P_i}^{-1} \\ C_i^A &= 0 \end{aligned} \tag{3.9}$$

where $\mathbf{P_i}$ is a matrix corresponding to projection onto the span of the surface normals where b_i belongs.
Equation (3.6) can be rewritten, finding update step of the optimization algorithm:

$$\mathbf{T} = \arg\min_{\mathbf{T}} \{ \sum_i ||\mathbf{P_i} \cdot d_i||^2 \} \tag{3.10}$$

Since $\mathbf{P_i}$ is orthogonal,

$$||\mathbf{P_i} \cdot d_i||^2 = d_i^T \cdot \mathbf{P_i} \cdot d_i \tag{3.11}$$

So (3.10) can be rewritten as:

$$\mathbf{T} = \arg\min_{\mathbf{T}} \{ \sum_i d_i^T \cdot \mathbf{P_i} \cdot d_i \} \tag{3.12}$$

which is the update step that minimized the distance $\mathbf{T} \cdot a_i$, the distance from the plane defined by b_i.

Termination criteria Since the algorithm is iterative, a termination must be defined in the refinement process of the final solution. PCL library offers many termination criteria other than maximum number of iterations, these can be combined using a common interface.

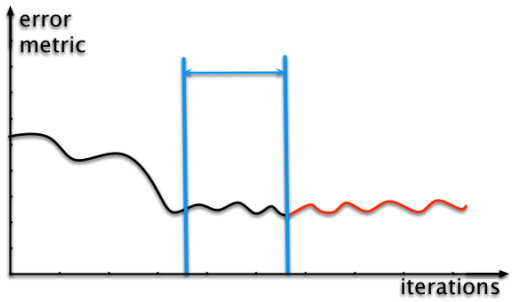

Figure 3.12: Error based on maximum number of similar iterations [19]

- **Maximum number of iterations**: if the number of iterations is over a given threshold, the optimizer estimation is considered as diverged. This is possible when the two point clouds are too different and registration process should require more iterations to converge.
- **Absolute transformation threshold**: stopping criteria when absolute current estimate is far away from initial transformation, particularly useful as an early check to detect divergence.
- **Relative transformation threshold**: calculated the transformation difference from current and last estimation, if the value is small enough the optimizer detects convergence criteria.
- **Maximum number of similar iterations**: similar to relative transformation threshold but number of iterations are executed anyway. This is useful to overcome situations where an algorithm converges to a local minima but it can still reach a global one (figure 3.12).
- **Relative mean square error**: similar to last two metrics but, instead of pose and rotation relative increment used to calculate the error, it is used mean square error metric (figure 3.13).
- **Absolute mean square error**: algorithm converges when the error between two clouds is below a threshold.

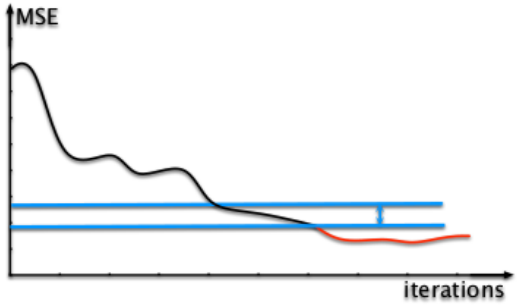

Figure 3.13: Error based on relative mean square [19]

Normal Distribution Transform (NDT)

A different approach for scan matching is based on **Normal Distribution Transform** [22]. This algorithm, developed by many libraries like PCL, has become famous in the research.
Here an overview is presented, focusing only on 2D Scan Matching.
Matching is performed by calculating Normal Distribution Transform of the source range scan, a differentiable probability density. This is used to find the transformation matrix (alignment between source and target point clouds) by optimization of the sum of a suitable score (figure 3.14).

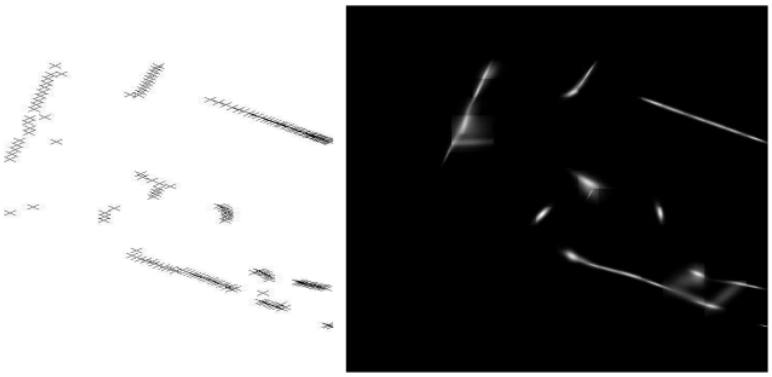

Figure 3.14: NDT calculated on single scan [22]

NDT of 2D Scan

2D Scan, in order to calculate normal distribution density, is suitably manipulated. In a pre-elaboration step, the area near by robot is subdivided into cells with fixed size containing at least three points. For each cell:

- All 2D points $\mathbf{x}_{i=1..n}$ in the cell are collected.
- The mean is calculated: $\mathbf{q} = \frac{1}{n}\sum_i \mathbf{x}_i$
- The covariance matrix is calculated: $\Sigma = \frac{1}{n}\sum_i (\mathbf{x_i} - \mathbf{q})(\mathbf{x_i} - \mathbf{q})^T$

Having this information about the single cell, a probability can be calculated by a normal distribution $N(q, \Sigma)$ whose probability density function is in equation (3.13): it represents the probability of measuring a single sample for each position within the cell. This probability density is used subsequently.

$$p(\mathbf{x}) \sim \exp\left(-\frac{(\mathbf{x_i} - \mathbf{q})^T \Sigma^{-1} (\mathbf{x_i} - \mathbf{q})}{2}\right) \tag{3.13}$$

Since 2D range scans are afflicted by discretization (only a noised shape of the environment is often considered [23]), in order to save potentially useful information, four overlapping grids are used where each one is shifted of a fixed length by the other. All points belong to different cells with four different densities, the one used later is calculated summing all the entries.
Noise introduced by lidar sensors can make the covariance matrix as singular, so it is not invertible. In order to prevent singularity, a check on eigenvalue is done, if it is not larger than a threshold it is set to this fixed value.
PLC library enables to set an important hyperparameter relevant to algorithm's general performance used in the testing phase: **resolution** parameter is referred to voxel resolution of the grid structure where point cloud is located in the intern development of the algorithm.

Algorithm

As the standard ICP, alignment produces transformation matrix T composed by translation (x, y) and orientation θ, between two points collected in different positions.
Having two scan (source and target):

1. It is build the NDT of the first scan.
2. The final estimation is initialized with zeros or odometry previously calculated.
3. Map each 2D point of the second scan into the coordinate frame of the first, using the initial value.
4. Calculate the normal distributions for all the points.
5. Calculate the **score** of the parameters, which is evaluated summing the latest value for all the points.

6. New parameter is estimated in order to optimize that score.

The algorithm is iterative, taking new points of the second scan, until convergence in the final estimation.
The score is calculated by:

$$score(\mathbf{p}) = \sum_i \exp\left(-\frac{((\mathbf{x}_i' - \mathbf{q_i})^{\mathbf{T}} \Sigma_{\mathbf{i}}^{\mathbf{-1}} (\mathbf{x}_i' - \mathbf{q_i}))}{2}\right) \tag{3.14}$$

where

- $\mathbf{p}$: vector of parameters (translations and rotations).
- $\mathbf{x_i}$: points of the second scan into the coordinate frame of the second scan.
- $\mathbf{x_i'}$: points x_i into the coordinate frame of the first scan.
- Σ_i, $\mathbf{q_i}$: covariance matrix and mean of the NDT of $\mathbf{x}_i'$.

The optimization (and the optimization step) is evaluated by **Newton's algorithm**, where the maximization is done by minimization of **-score**.
The algorithm, iteratively, solves the equation:

$$\mathbf{H}\Delta\mathbf{p} = -\mathbf{g} \tag{3.15}$$

Where:

- $\mathbf{g}$: transposed gradient of **-score(p)**.
- $\mathbf{H}$: Hessian of -**score(p)**

The solution is built by adding an increment $\Delta\mathbf{p}$ properly calculated.

3.2.2 Intra-Robot Loop Closure detection module

Robots in the team, during its movement, must detect an already crossed position. Those positions are defined as **loop closures**. The module aim is to detect those loops giving the point cloud raw (or an downsampled representation) and the odometry calculated by Signal Processing module (figure 3.15).

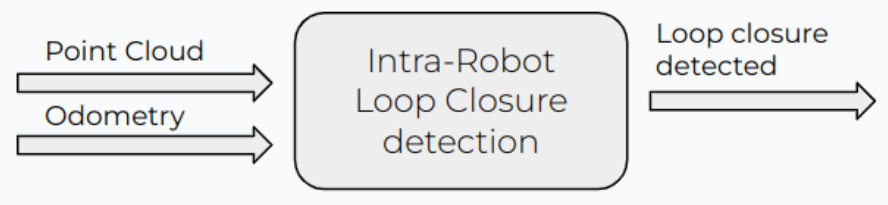

Figure 3.15: Intra-Robot Loop Closure detection module

Algorithm

The problem is an application of place recognition algorithms. Here a version based on **Scan Context** descriptors is presented. Those descriptor (in the form of a matrix) are created from raw data and stored in a vector.
The algorithm is called at a given frequency, passing through these steps:

1. last index in the vector of point cloud is taken.
2. **scan context matching** is performed: the scan context corresponding to last index is compared to the entries in the vector of descriptors.
   This matching can be done using brutal force approach (comparing iteratively all the entries) or exploiting particular data structure of support.
   In this context the last approach is used, it is created a **KD-tree** dependent on scan context collected until time k and a **KD-tree search** algorithm is performed. This algorithm selects a subset of possible scan contexts next to the query whose distances are calculated. The minimum of these values is then compared to a threshold, if it is below the couple of index (current and resultant of matching) is considered as a *candidate* loop closure.
3. **scan matching** algorithm: considering the resultant couple of index, an ICP iteration is set: the source is the current scan, the target is the result of the previous step. At first of setting parameters procedure, both point clouds are aligned considering the odometry calculated at those time instant.
   ICP score is then considered to be compared to a threshold, if it is below the *candidate* loop closure is considered as *valid* and the transformation matrix is stored to be passed at the optimizer.

Those steps are developed, the discussion is in Chapter 5.
In the following a description of scan context is presented. In particular, it is explained:

- how to create a scan context descriptor from raw data.
- how to build a comparison metrics to evaluate distances between two scan contexts.

Scan Context

Scan contexts [16] are lidar data descriptors, built on point cloud raw without using any histogram formulation or pre-step training. They are particularly useful in SLAM context since loop-closure detection is a place recognition task. Only point clouds are used so the descriptor has to face with two problems: it must be invariant to rotation with respect to viewpoint and it must be robust to noise as much as possible.
In order to calculate Scan Context from a single 3D scan, a egocentric structure of scan is used. This is divided into bins equally distributed following azimuthal and radial axes (in polar coordinates) as shown in 3.16. Following the axes direction, parameters N_s and N_r can be fixed as number of **sectors** and **rings**.

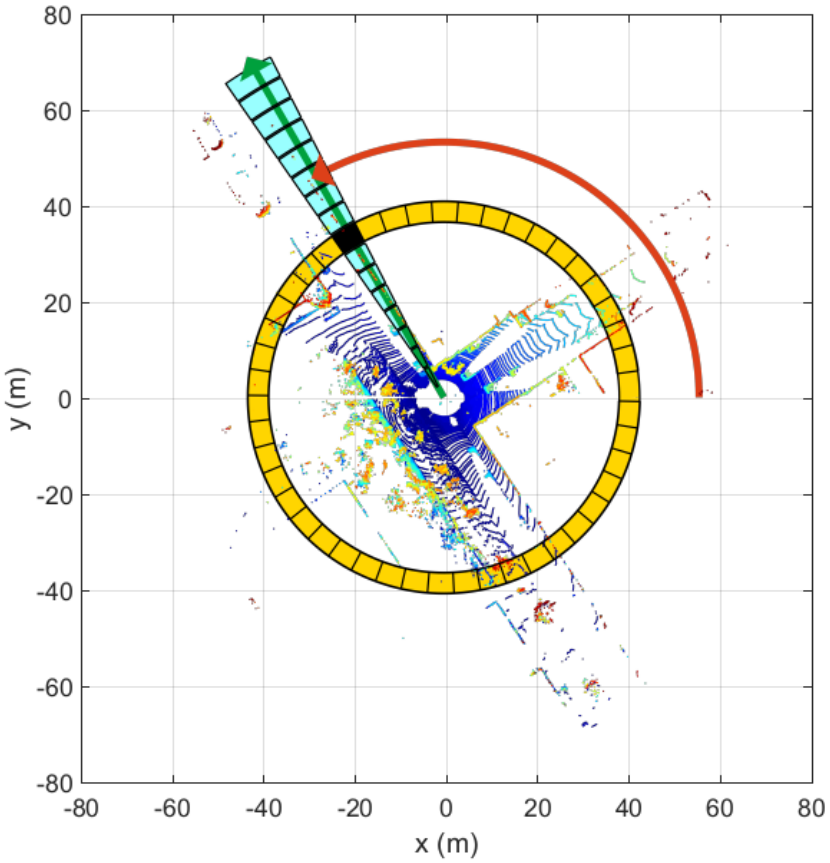

Figure 3.16: Polar coordinates bins division [16]

Whole 3D scan is partitioned into different point clouds overlapped, through

which any points belong to a bin. Calling P_{ij} the set of points into a bin (bins are indexed with respect to sector and ring numbers), the entire set of points can be reconstructed as:

$$P = \bigcup_{i \in N_r, j \in N_s} P_{ij} \tag{3.16}$$

Choosing this polar structure, locations far from sensor are less represented by bins, so sparsity of information about far points are not so much considered.
Later on this first part of partitioning, at any bin is associated a single real value:

$$\phi : P_{ij} \to \mathbb{R} \tag{3.17}$$

The choice of this metrics can be discussed, scan context uses bin encoding function based on **maximum height**:

$$\phi(P_{ij}) = \max_{\mathbf{p} \in P_{ij}} z(\mathbf{p}) \tag{3.18}$$

Where $z(\cdot)$ is the function that returns the height of points contained to bin. Empty bins return 0 as bin value.
Scan context can be summarized by a $N_r X N_s$ matrix:

$$I = (a_{ij}) \in \mathbb{R}^{\mathrm{N_r X N_s}}, \mathrm{a_{ij}} = \phi P_{ij} \tag{3.19}$$

A descriptor representation of point clouds in figure 3.16 is shown in figure 3.17.

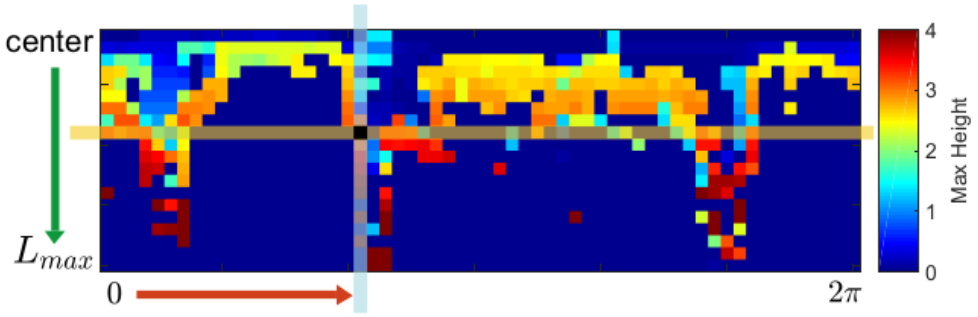

Figure 3.17: Scan Context representation [16]

A new data is created so must be defined a metric to compare two or many descriptors. Considering I^q and I^c two different scan contexts, they are compared following its columns:

$$d(I^q, I^c) = \frac{1}{N_s} \sum_{j=1}^{N_s} (1 - \frac{c_j^q \cdot c_j^c}{||c_j^q|| ||c_j^c||}) \tag{3.20}$$

Where c_j^q is the j-th column of q.
Since the same location can be represented by a different scan context which is column shifted with respect to the first (due to the lidar sensor coordinate system respect to global coordinate system, figure 3.18), final distance value is evaluated as the minimum value of the distance calculated from different column-shifted descriptor.

$$D(I^q, I^c) = \min_{n \in N_s} d(I^q, I_n^c) \tag{3.21}$$

where I_n^c is the n-th shifted view of scan context I^c.

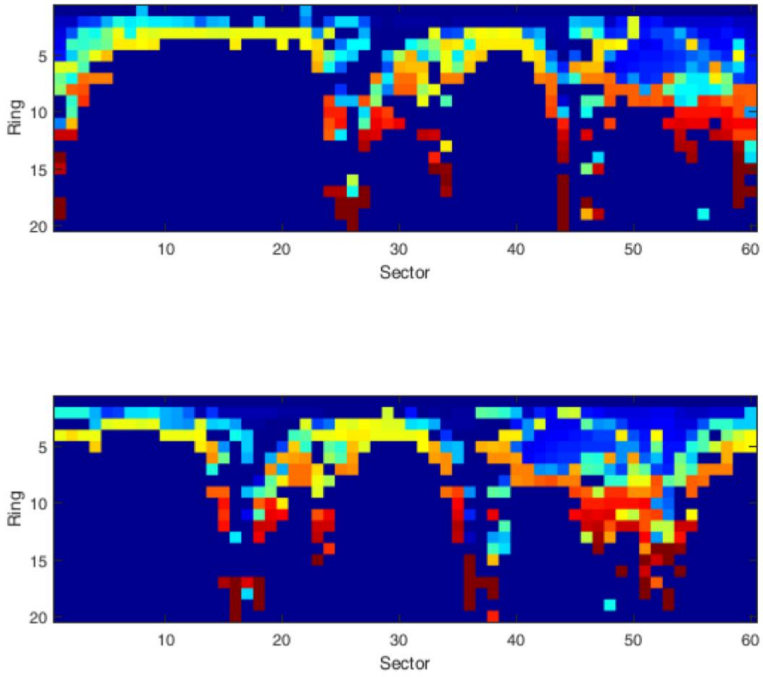

Figure 3.18: Scan Contexts of different view of same place revisited [16]

3.2.3 Inter-Robot Loop Closure detection module

In a collaborative context, each robot must consider the contribution of the team in order to have better final pose and map estimation saving up time. Inter-robot loop closure detection module is the first module called during an rendezvous of two robots.
The aim of the module is to detect portions of environments crossed also by the neighbors, called inter-robot loop closures, in order to calculate relative matrix

transformation to add in pose-graph structure. Similarly to the intra-robot case, the module implies place recognition subtasks: in collaborative scenarios this is a critical aspect since bad valuations on algorithms to be used can bring the solution to be unworkable.
The module is fully distributed, exploiting compact representation of scan context descriptors (figure 3.19).

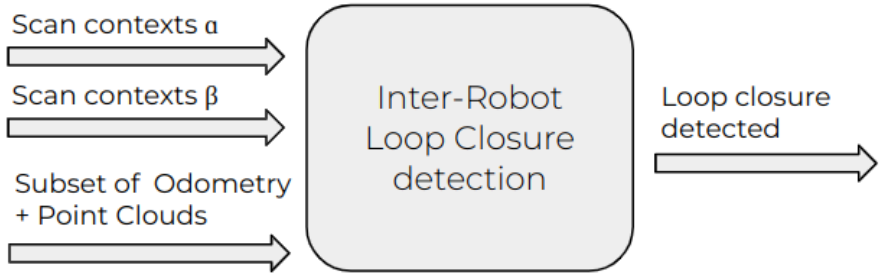

Figure 3.19: Inter-Robot loop closure detection module representation

Algorithm

The algorithm is inspired by the DOOR-SLAM inter-robot loop closure detection module (figure 3.20) where visual descriptors are used. In this solution, the idea is to replace those descriptors with others based on point clouds. During development it must be necessary to introduce differences mainly regarding the sharing process of information by the two robots.

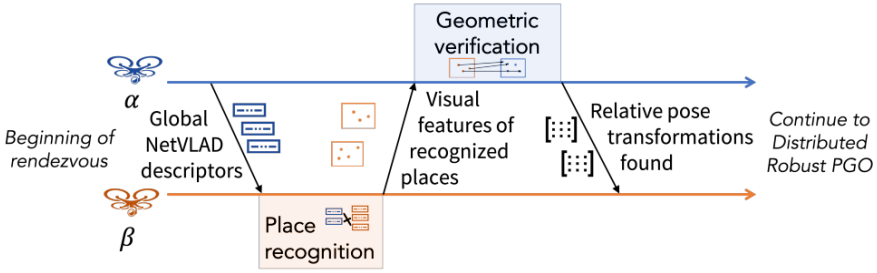

Figure 3.20: Inter-Robot loop closure detection used in DOOR-SLAM [9]

Considering robot α and β:

1. Robot α sends to robot β its vector of scan contexts.
2. Robot β performs an updated version of scan context matching algorithm: KD-tree is created, for each entry of scan contexts vector of α is searched set

of near scan contexts of β. The minimum of this set is inserted in a vector of couples of indexes representing all loop closure candidates.

3. Robot β sends this vector of couples of indexes to α. Later on reception, robot α sends back set of Point Clouds given by candidate indexes.

4. For each of the entries of set of point clouds, robot β performs an ICP algorithm to calculate ICP score. If that score is below a threshold, a loop is finally detected.

5. Robot β sends back set of matrices transformations to robot α so it is possible to update pose-graph structure used by optimizer.

Using scan context descriptors, it is not needed to share all point clouds collected until time of rendezvous but only a subset.

3.2.4 Outlier Rejection module

Inter-robot loop closure detection module, which detects if robots revisit locations crossed by others, is a perception-based task (such as the local version of loop closure detection). A location may be seen as similar to others even if it is not true causing **perceptual aliasing**.

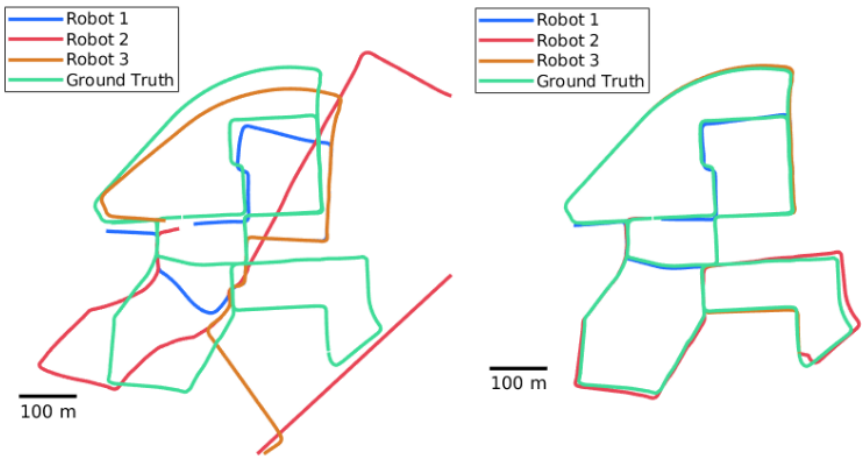

Figure 3.21: Results without (left) and with (right) outlier rejection module [9]

In order to improve quality of results calculated by optimization module, these spurious measurements must be filtered (as shown in figure 3.21 in DOOR-SLAM [9], where is applied to a complete solution in the state of the art, this step can not be neglected).

The outlier rejection modules takes all the computations done in previous modules through pose-graph structures as input and suitable filters this graph from outliers (figure 3.22).

In order to perform outlier rejection a distributed algorithm must be chosen.

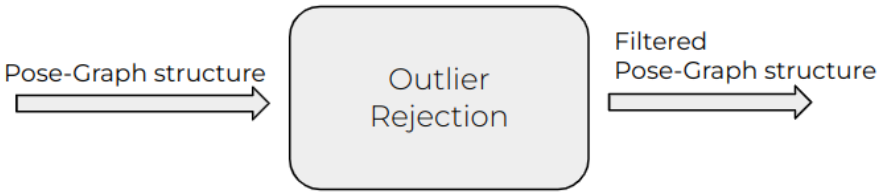

Figure 3.22: Outlier Rejection module representation

In this model it is used a variation to classic version of Pairwise Consistency Maximization (PCM) algorithm to be used in distributed contexts. Distributed PCM has an open-source implementation in [9].
In the following classic PCM algorithm is presented to have an insight of modules functionalities to be implemented in the future.

Pairwise Consistent Maximization (PCM)

The approach in PCM [24] does not classify directly inlier or outlier measurements but converts the problem into finding the largest consistent set of inter-robot measurements.
Giving a consistency metric $\mathbf{C}$ and a threshold γ, a set of measurements is **pairwise internally consistent**, indicated by $\tilde{\mathbf{Z}}$, if:

$$C(\mathbf{z}_i, \mathbf{z}_j) \leq \gamma, \quad \forall \mathbf{z}_i, \mathbf{z}_j \in \tilde{\mathbf{Z}} \tag{3.22}$$

Since all-encompassing condition, all the measurements in the pair must be consistent with each other.
Consistency metric C function can be expressed depending on inter-robot measurements.

$$C(\mathbf{z}^{ab}_{ik}, \mathbf{z}^{ab}_{jl}) = \left\| (\ominus \mathbf{z}^{ab}_{ik}) \oplus \hat{\mathbf{x}}^{a}_{ij} \oplus \mathbf{z}^{ab}_{jl} \oplus \hat{\mathbf{x}}^{b}_{lk} \right\|_{\Sigma} = \| \epsilon_{ikjl} \|_{\Sigma_{ikjl}} \tag{3.23}$$

the elements on (3.23) are shown in figure 3.23.

In (3.23):

- $\mathbf{z}^{ab}_{ik}$, $\mathbf{z}^{ab}_{kl}$: inter-robot measurements.
- $\hat{\mathbf{x}^{a}_{ij}}$, $\hat{\mathbf{x}^{b}_{lk}}$: current pose estimates by robots a and b.

The operator $\oplus$ and $\ominus$ are commonly used to formulate spatial relationships between poses in robotics contexts [25].

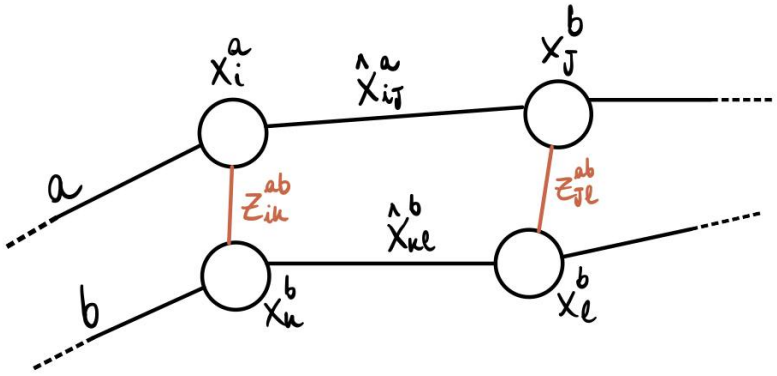

Figure 3.23: Two robots (a and b) pose-graph sub-sequences. Inter-robot measurements in red.

In particular, $\oplus$ operator is called **compounding** and gives the resultant relationship between two consecutive poses and it can be used to find the location of a mobile robot.
Giving measurements $\mathbf{x}_{ij}$ and $\mathbf{x}_{jk}$ in 2D context, $\mathbf{x} = [x, y, \phi]^T$.
The **compounding** is defined as:

$$x_{ik} = x_{ij} \oplus x_{jk} = \begin{bmatrix} x_{jk}\cos\phi_{ik} - y_{ik}\sin_{\phi_{ij}} + x_{ij} \\ x_{jk}\sin\phi_{ik} + y_{ik}\cos_{\phi_{ij}} + x_{ij} \\ \phi_{ij} + \phi_{jk} \end{bmatrix} \tag{3.24}$$

Otherwise, $\ominus$ operator indicates **inverse relationship**.
Giving same measurements $\mathbf{x}_{ij}$ and $\mathbf{x}_{jk}$, it is defined as:

$$x_{ji} = \oplus x_{ij} = \begin{bmatrix} -x_{ij}\cos_{\phi_{ij}} - y_{ij}\sin\phi_{ij} \\ x_{ij}\sin\phi_{ij} - y_{ij}\cos\phi_{ij} \\ -\phi_{ij} \end{bmatrix} \tag{3.25}$$

In (3.23), the $\|\cdot\|_\Sigma$ is the Mahalanobis distance [26]. Considering a probability distribution Q in $\mathbb{R}^n$ with mean $\mu = (\mu_1, \mu_2, \ldots, \mu_N)$ and covariance matrix Σ, the Mahalonobis distance of two points $\mathbf{x} = (x_1, x_2, \ldots, x_N)$ and $\mathbf{y} = (y_1, y_2, \ldots, y_N)$ is defined as:

$$\|\mathbf{x}, \mathbf{y}\|_\Sigma = \sqrt{(\mathbf{x} - \mathbf{y})^T \Sigma^{-1} (\mathbf{x} - \mathbf{y})} \tag{3.26}$$

Now that pairwise internal consistent check is defined, PCM algorithm must be discussed.

Algorithm The final goal of PCM is to find the largest consistent subset of inter-robot measurements $\mathbf{Z}^{ab}$, for simplicity called $\mathbf{Z}^*$.
In order to do this, a particular problem formulation is built. A binary switch variable s_u is considered, different for each of constraints in the set $\mathbf{Z}^{ab}$, where 1 is taken if measure is contained, 0 if it is not. For simplicity notation z_{ij}^{ab} is substituted by z_u.
The problem is to find the max clique as:

$$\begin{aligned} \mathbf{S}^* &= \arg\max_{\mathbf{S}\in 0,1^m} \|\mathbf{S}\|_0 \\ s.t.\ &\|\epsilon_{uv}\|_{\Sigma_{uv}}\, s_u s_v \leq \gamma \end{aligned} \tag{3.27}$$

where:

- **m**: number of measurements.
- ϵ_{uv}: error term in (3.23).
- Σ_{uv}: covariance matrix.

Later on finding $\mathbf{S}^*$, the indices associated with s_u corresponds to measurements in the maximum size set.
Resolving (3.27) can be a hard computational problem, since brutal force solutions are $O(n^2)$ in complexity. A manipulation can be applied in order to ease a solution. Considering that the first part of (3.23) can be calculated separately from this problem, a **consistency graph** can be built from the matrix **Q** where the elements correspond to consistent measurements z_u and z_u.
In this graph $G = \{V,\ E\}$, for each vertex V is associated a measurement and for each edges E a relation of consistency between measurements.
The problem, in this form, is converted to find the **maximum clique** of the graph, a well-known graph theory problem: the subject is to find its largest subset of nodes. Since the amount of research in this field, even if the problem is NP-hard, many well-optimized approaches can be used to find the solution.
Here a greedy one is used based on [27], where the search tree is pruned in order to find maximum clique quickly in large graphs. This operation is done during algorithm execution, in order to does not consider those vertices in the successive step of the algorithm since the inclusion on the set does not imply a maximum solution.
Considering *max* variable as size of current maximum clique found. Pruning is done multiple times:

- **Pruning 1**: vertices having number of neighbors lower than *max* are filtered.
- **Pruning 2**: during the procedure of creation of a set of neighboard for vertex v_i, only its neighbors which are not already considered belonging into maximum clique are considered.

- **Pruning 3**: vertices with degree less than current max value are pruned.
- **Pruning 4**: check even if all vertices in the neighbor set were added to get the solution, the size does not exceed max current value.

The algorithm proposed in [27] has an open source implementation and it is prone to concurrent implementations.

3.2.5 Distributed Mapper module

Distributed mapper module has as input intra- and inter-robot measurements given by other modules and gives back the robots trajectory estimation by means of pose graph optimization. Those corrected trajectory is necessary to deskew the point clouds which build the map in any robots in the team.
The inputs are in the form of a filtered pose-graph structure given by Outlier Rejection module (figure 3.24).

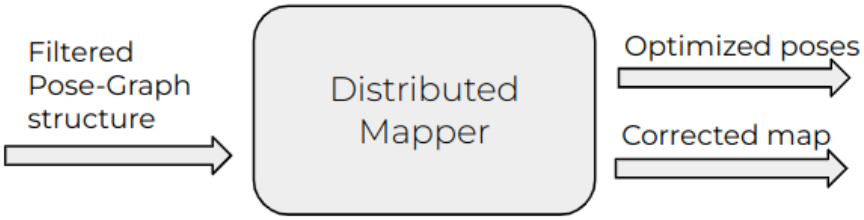

Figure 3.24: Distributed Mapper module representation

In this scenario, a distributed graph pose optimizer [28] is selected which is implemented following bandwidth and privacy constraints: during a rendezvous robots exchange the minimum information to find the final result. This feature makes the module compatible to particular working conditions like in military applications.
Even if computation is displaced into different computational units, convergence constraint in the final estimation is always perceived: the resultant trajectory and map is the same in all the components of the team (figure 3.25).

The mapper discussed is fully implemented in the open source library [29].

Problem definition

Giving a set of robots defined by $\Omega = \{\alpha, \beta, \gamma, \ldots\}$. A robot α at time i has pose $\mathbf{x}_{\alpha_i} \in SE(3)$ where *SE(3)* is the Special Euclidean group of 3D rigid transformations. A single robot pose $\mathbf{x}_{\alpha_i}$ can be indicated by $(\mathbf{R}_{\alpha_i}, \mathbf{t}_{\alpha_i})$, where $\mathbf{R}_{\alpha_i} \in SO(3)$ is the rotation matrix and $\mathbf{t}_{\alpha_i} \in \mathbb{R}$ is the position. The entire robots trajectory is in the

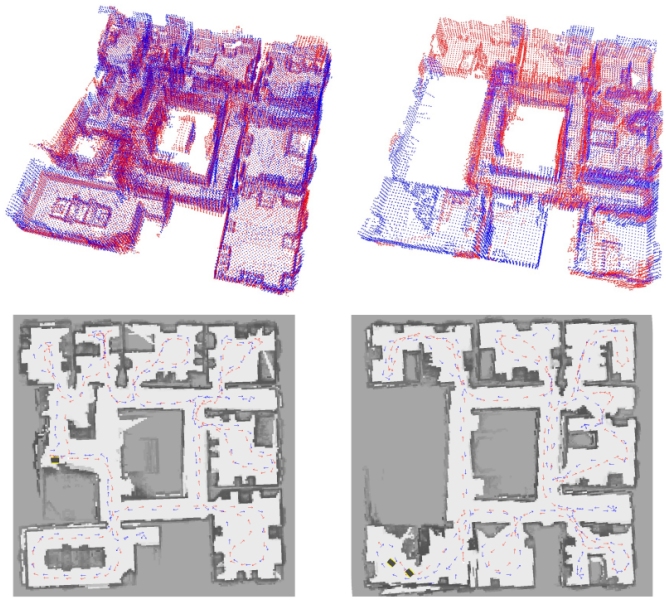

Figure 3.25: Two robots (red and blue point clouds) after pose graph optimization [28]

form of $\mathbf{x}_\alpha = [\mathbf{x}_{\alpha_1}, \mathbf{x}_{\alpha_2}, \ldots]$.
Inter and intra-robot measurements can be summarized by:

$$\begin{aligned} \bar{\mathbf{x}}_{\beta_j}^{\alpha_i} &\doteq (\bar{\mathbf{R}}_{\beta_j}^{\alpha_i}, \bar{\mathbf{t}}_{\beta_j}^{\alpha_i}) \\ \bar{\mathbf{R}}_{\beta_j}^{\alpha_i} &= (\mathbf{R}_{\alpha_i})^T \mathbf{R}_{\beta_j} \mathbf{R}_\epsilon \\ \bar{\mathbf{t}}_{\beta_j}^{\alpha_i} &= (\mathbf{R}_{\alpha_i})^T (\mathbf{t}_{\beta_j} - \mathbf{t}_{\alpha_i}) + \mathbf{t}_\epsilon \end{aligned} \tag{3.28}$$

where:

- $\bar{\mathbf{x}}_{\beta_j}^{\alpha_i}$: generic relative pose measurements, by changing α, β, i and j it is possible to derive intra- and inter-robot formulation.
- $\bar{\mathbf{R}}_{\beta_j}^{\alpha_i}$: generic relative rotation measurements, it describes rotation of $\mathbf{R}_{\beta_j}$ at time i, with respect to coordinate frame of robot α. A matrix $\mathbf{R}_\epsilon$ is included to take into account measurement noise.
- $\bar{\mathbf{t}}_{\beta_j}^{\alpha_i}$: generic relative position measurements, it describes the pose of robot $\mathbf{t}_{\beta_j}$ at time i, with respect to coordinate frame of robot α. Term of noise is added as $\mathbf{t}_\epsilon$.

The set of the all robots intra-robot measurements is indicated by $E_I = \bigcup_{\alpha \in \Omega} E_I^\alpha$, where E_I^α is the measurements related to robot α. Similarly, $E_S = \bigcup_{\alpha \in \Omega} E_S^\alpha$ is the

set of all robots inter-robot measurements.
The problem to be resolved is in the form of a Maximum Measurements Likelihood, as discussed in the Front End sub-chapter of Chapter 2. Here a simplification of notation is done.
The result is the set of poses $\mathbf{x} = [\mathbf{x}_\alpha, \mathbf{x}_\beta, \mathbf{x}_\gamma, \ldots]$. The estimate for $\mathbf{x}$, considering independent measurements, is defined as:

$$\mathbf{x}^* = \arg\max_{\mathbf{x}} = \prod_{(\alpha_i,\beta_j)\in E} p(\bar{\mathbf{z}}^{\alpha}_{\beta_j}|\mathbf{x}) \tag{3.29}$$

The likelihood (3.29) can be manipulated imposing a suitable noise distributions to the measurements [30], in the form of $\mathbf{R}_\epsilon$ and $\mathbf{t}_\epsilon$.
The translation noise is distributed as a zero-mean Gaussian $\mathbf{t}_\epsilon \sim gaussian(\mathbf{0}_3, \omega_t^2\mathbf{1}_3)$.
The rotational noise is distributed as a Von Mises $\mathbf{R}_\epsilon \sim vonMises(\mathbf{1}_3, \omega_R^2)$.
Von Mises distribution is a continuous probability distribution on the circle, it is used as a approximation of the wrapped normal distribution (normal distribution around the unit circle) since the better tractability and mathematical simplicity [31].
Following these noise assumption, expression (3.29) can be computed as:

$$\min_{\substack{\mathbf{t}_{\alpha_i}\in\mathbb{R}^3, \mathbf{R}_{\alpha_i}\in \mathrm{SO}(3) \\ \forall\alpha\in\Omega, \forall i}} \sum_{(\alpha_i,\beta_i)\in E} \omega_t^2 \left\|\mathbf{t}_{\beta_j} - \mathbf{t}_{\alpha_i} - \mathbf{R}_{\alpha_i}\bar{\mathbf{t}}^{\alpha_i}_{\beta_j}\right\|^2 + \frac{\omega_R^2}{2}\left\|\mathbf{R}_{\beta_j} - \mathbf{R}_{\alpha_i}\bar{\mathbf{R}}^{\alpha_i}_{\beta_j}\right\|_F^2 \tag{3.30}$$

where $\left\|\mathbf{R}_{\beta_j} - \mathbf{R}_{\alpha_i}\bar{\mathbf{R}}^{\alpha_i}_{\beta_j}\right\|_F$ is the *Chordal distance* ($\|\cdot\|_F$ is the Frobenious matrix norm, the sum of the squares of the all entries) used as distance metrics between different rotational matrices.
The mapper must resolve (3.30) optimization. In order to explain distributed solution a first centralized approach is analyzed, then the solution calculated is evolved in distributed manner using concepts derived before.

Centralized Approach: Two-Stage Pose Graph optimization

The centralized solution works with the assumption that all measurements E of the robots in the team are collected in a single unit operating all the computations.
The optimization problem (3.30) has quadratic form but $\mathbf{R}_{\alpha_i} \in SO(3)$ constraints is non-convex. In fact, special orthogonal group SO(3) implies that all the matrices belonging into it must be orthogonal with determinat equals to 1. $SO(3) \doteq \{\mathbf{R} \in IR^{3\times3} : \mathbf{R}^T\mathbf{R} = \mathbf{1}_3, det(\mathbf{R}) = 1\}$ [32].
In order to solve optimization and consider non-convexity, the problem (3.30) is solved in two stages: first a rotation matrix estimation $\mathbf{R}_{\alpha_i}$ is evaluated (calculated projecting a previous rotation estimated with a relaxed problem definition), then full poses are estimated by a Gauss-Newton iteration (algorithm used non-linear least square problems [33]) using previous result.

Rotation estimation The first rotation estimation is taken from the solution of sub-problem of the left side of (3.30).

$$\min_{\substack{\mathbf{R}_{\alpha_i}\in SO(3)\\ \forall\alpha\in\Omega,\forall i}} \sum_{(\alpha_i,\beta_j)\in E} \omega_R^2 \left\|\mathbf{R}_{\beta_j} - \mathbf{R}_{\alpha_i}\bar{\mathbf{R}}_{\beta_j}^{\alpha_i}\right\|_F^2 \tag{3.31}$$

which intends to estimate the result using only relative rotation measurements. Since the problem 3.31 is non-convex, due to $\mathbf{R}_{\alpha_i} \in SO(3)$ constraints, first a solution based on a relaxed version is calculated: the constraints on $\mathbf{R}$ is extended to all matrix. Later, considering the set of solution, only those belonging into SO(3) group must be taken: a projection step is performed.
The relaxed optimization problem is in the form:

$$\min_{\substack{\mathbf{R}_{\alpha_i}\\ \forall\alpha\in\Omega,\forall i}} \sum_{(\alpha_i,\beta_j)\in E} \omega_R^2 \left\|\mathbf{R}_{\beta_j} - \mathbf{R}_{\alpha_i}\bar{\mathbf{R}}_{\beta_j}^{\alpha_i}\right\|_F^2 \tag{3.32}$$

which is quadratic in the unknown $\mathbf{R}_{\alpha_i}$. It is possible to a non-linear least squared form:

$$\min_{\mathbf{r}} \|\mathbf{A}_r\mathbf{r} - \mathbf{b}_r\|^2 \tag{3.33}$$

where the unknown matrices (for all robots in Ω) are all stacked in a single $\mathbf{r}$ vector and known matrix $\mathbf{A}_r$ and $\mathbf{b}$ are built. Solution of 3.33 can be found solving the equation:

$$(\mathbf{A}_r^T\mathbf{A}_r)\mathbf{r} = \mathbf{A}_r^T\mathbf{b}_r \tag{3.34}$$

Solutions $\hat{\mathbf{r}}$ in vector form are then converted into matrices $\hat{\mathbf{R}}_{\alpha_i}$, $\forall\alpha\in\Omega,\forall i$.
As mentioned before $\hat{\mathbf{R}}_{\alpha_i}$ solutions set includes matrices which are not into *SO(3)* group. This set is projected to SO(3) performing an Singular Valure Decomposition (SVD) for each rotations. The matrices found are indicated by $\tilde{\mathbf{R}}_{\alpha_i}, \forall\alpha\in\Omega,\forall i$.

Full pose estimation Full problem 3.30 is considered to recover full pose. In order to bring it into the form of a least squares problem, a manipulation is done: it is considered that each rotation $\mathbf{R}_{\alpha_i}$ depends on result previously calculated ($\tilde{\mathbf{R}}_{\alpha_i}$) and on a unknown perturbation modelled as the unknown θ_{α_i}:

$$\mathbf{R}_{\alpha_i} = \tilde{\mathbf{R}}_{\alpha_i} Exp(\theta_{\alpha_i}) \tag{3.35}$$

where $Exp(\cdot)$ is the exponential map for SO(3).
The exponential map applied to a element in SO(3) is equal to its matrix exponential [34]:

$$Exp(\mathbf{X}) = \sum_{k=0}^{+\infty} \frac{\mathbf{X}^k}{k!} = \mathbf{1} + \mathbf{X} + \frac{\mathbf{X}^2}{2} + \frac{\mathbf{X}^3}{6} + \dots \tag{3.36}$$

where $\mathbf{X} \in SO(3)$, $\mathbf{1}$ is the identity matrix.
Introducing this factor 3.30 can be rewritten:

$$\min_{\substack{\mathbf{t}_{\alpha_i}\in\mathbb{R}^3,\theta_{\alpha_i}\in\mathbb{R}^3 \\ \forall\alpha\in\Omega,\forall i}} \sum_{(\alpha_i,\beta_i)\in E} \Big\{\omega_t^2 \left\|\mathbf{t}_{\beta_j} - \mathbf{t}_{\alpha_i} - \tilde{\mathbf{R}}_{\alpha_i} Exp(\theta_{\alpha_i})\bar{\mathbf{t}}_{\beta_j}^{\alpha_i}\right\|^2 + \frac{\omega_R^2}{2}\left\|\tilde{\mathbf{R}}_{\beta_j} Exp(\theta_{\beta_j}) - \tilde{\mathbf{R}}_{\alpha_i} Exp(\theta_{\alpha_i})\bar{\mathbf{R}}_{\beta_j}^{\alpha_i}\right\|_F^2 \Big\} \tag{3.37}$$

Using θ_{α_i} parameter it is possible to drop constraints *SO(3)* on matrix and moved the non-convexity condition on it.
Now an approximation can be done exploiting exponential map of the parameter: the operator is non-linear in general (which brings the constraint to be non-convex), here a first order approximation is used:

$$Exp(\theta_{\alpha_i}) \simeq \mathbf{1}_3 + \mathbf{S}(\theta_{\alpha_i}) \tag{3.38}$$

where $\mathbf{S}(\theta_{\alpha_i})$ is a symmetric matrix depending on the parameter.
Applying this approximation into 3.37:

$$\min_{\substack{\mathbf{t}_{\alpha_i}\in\mathbb{R}^3,\theta_{\alpha_i}\in\mathbb{R}^3 \\ \forall\alpha\in\Omega,\forall i}} \Big\{\omega_t^2 \left\|\mathbf{t}_{\beta_j} - \mathbf{t}_{\alpha_i} - \tilde{\mathbf{R}}_{\alpha_i}\bar{\mathbf{t}}_{\beta_j}^{\alpha_i} - \tilde{\mathbf{R}}_{\alpha_i}\mathbf{S}(\theta_{\alpha_i})\bar{\mathbf{t}}_{\beta_j}^{\alpha_i}\right\|^2 + \frac{\omega_R^2}{2}\left\|\tilde{\mathbf{R}}_{\beta_j} + \tilde{\mathbf{R}}_{\beta_j}\mathbf{S}(\theta_{\alpha_i}) - \tilde{\mathbf{R}}_{\alpha_i}\bar{\mathbf{R}}_{\beta_j}^{\alpha_i} - \tilde{\mathbf{R}}_{\alpha_i}\mathbf{S}(\theta_{\alpha_i})\bar{\mathbf{R}}_{\beta_j}^{\alpha_i}\right\|_F^2 \Big\} \tag{3.39}$$

the expression is finally in quadratic form. So it can be converted into a least-squared problem (similarly to previous step), with solutions taken from the last equation.

$$\begin{aligned} \min_{\mathbf{p}} \|\mathbf{A}_p\mathbf{p} - \mathbf{b}_p\| \\ (\mathbf{A}_p^T\mathbf{A}_p)\mathbf{p} = \mathbf{A}_p^T\mathbf{b}_p \end{aligned} \tag{3.40}$$

where $\mathbf{p}$ is a single vector with the combined unknown $\mathbf{t}_\alpha, \theta_\alpha$. The solution $\mathbf{p}$ is then split into $\mathbf{t}_{\alpha_i}$ and θ_{α_i}. The last parameter can be used to correct rotation matrix $\mathbf{R}_{\alpha_i}$.

Distributed approach

Considering the last results based on centralized approach, it can be shown that the two least-squares problems (3.33) and (3.40), which gives the solutions of the two steps, can be suitable re-written splitting the computation among the robots in the team.
For example, considering vector $\mathbf{r}$ in (3.33) or $\mathbf{p}$ in (3.40), these can be split into

sub-vectors $\mathbf{r} = [\mathbf{r}_\alpha, \mathbf{r}_\alpha, \ldots]$ and $\mathbf{p} = [\mathbf{p}_\alpha, \mathbf{p}_\alpha, \ldots]$ where the single robot contribution is isolated.
Considering known symmetric matrix $\mathbf{H}$ and vector $\mathbf{g}$, merging all unknown variable on $\mathbf{y}$, these two least-squares problem can be written in general form:

$$\mathbf{H}\ \mathbf{y} = \mathbf{g} \Leftrightarrow \begin{bmatrix} \mathbf{H}_{\alpha\alpha} & \mathbf{H}_{\alpha\beta} & \ldots \\ \mathbf{H}_{\beta\alpha} & \mathbf{H}_{\beta\beta} & \ldots \\ \vdots & \vdots & \ddots \end{bmatrix} \begin{bmatrix} \mathbf{y}_\alpha \\ \mathbf{y}_\beta \\ \vdots \end{bmatrix} = \begin{bmatrix} \mathbf{g}_\alpha \\ \mathbf{g}_\beta \\ \vdots \end{bmatrix} \tag{3.41}$$

In particular, the entries of matrix $\mathbf{H}$ are referred to values on single robot along its diagonal or evaluated through communication between its neighbors in the other entries.
In order to isolate single robot set of computation to find $\mathbf{y}_\alpha$, equation 3.41 is manipulated:

$$\begin{aligned} \sum_{\delta \in \Omega} \mathbf{H}_{\alpha\delta}\mathbf{y}_\delta = \mathbf{g}_\alpha \quad & \forall \alpha \in \Omega \\ \mathbf{H}_{\alpha\alpha}\mathbf{y}_\alpha = -\sum_{\delta \in \Omega \setminus \alpha} \mathbf{H}_{\alpha\delta}\mathbf{y}_\delta + \mathbf{g}_\alpha \quad & \forall \alpha \in \Omega \end{aligned} \tag{3.42}$$

In this form, each variable associated to the robot is shown. Starting from (3.42), two different methods can be used to resolve the equation: **Successive Over-Relaxation** (SOR) and **Jacobi Over-Relaxation** (JOR).
Since the open source library implementation [29] resolves (3.42) with the last of the two methods cited, in the following only this method is explained.

Distributed Jacobi Over-Relaxation (JOR) Starting from an arbitrary initial estimate $\mathbf{y}^{(0)} = [\mathbf{y}_\alpha^{(0)}, \mathbf{y}_\beta^{(0)}, \ldots]$, linear system in 3.42 can be resolved iteratively using at each step k:

$$\mathbf{y}_\alpha^{(k+1)} = (1-\gamma)\mathbf{y}_\alpha^{(k)} + \gamma\mathbf{H}_{\alpha\alpha}^{-1}\Big(-\sum_{\delta \in \Omega \setminus \alpha} \mathbf{H}_{\alpha\delta}\mathbf{y}_\delta^{(k)} + \mathbf{g}_\alpha\Big), \quad \forall \alpha \in \Omega \tag{3.43}$$

where γ is called **relaxation factor**.
At each iteration, robot α computes its own estimation $\mathbf{y}_\alpha^{(k+1)}$ assuming neighbors contribution $\mathbf{y}_\delta^{(k)}$ as constant.
After some iteration, this local estimate converge to *system* solution given by global expression 3.41 and calculated if centralized approach would be used.
Convergence of estimation using JOR algorithm is properly demonstrated, experimental results proposed by authors shown a convergence using $\gamma \leq 1$.

Chapter 4

Development and Results

In the following chapter is presented implementation and relative testing of a subset of modules deeply analyzed in the previous thesis work.
Development results covered the first part of collaborative algorithms, related to creation of pose-graph ready to be filtered by Outlier Rejection module.
Looking at figure 3.4 modules, in future work the libraries related to Distributed PCM and Distributed Mapper must be integrated in the solution to give the final results as optimized poses and a map.
At first a general description about dataset used in the testing process is given. The development subchapters are explained following a common structure: at first the initialization process is analyzed (subscription to topics, publishing purpose of messages, and so on), then a brief high-level code is presented.
As mentioned in Chapter 3, ROS2 Foxy framework is exploited to built the solution. All the nodes are written in C++. Launch files useful to calls all those nodes in one instance are developed in Python.
The solution can be seen in repository referred to link.

4.1 Dataset

The modules developed are tested exploiting different datasets provided by authors with open source license.
Datasets are in the form of ROS1 **bag** file [35]. Bag file is a convenient file format to store one or more topics messages collected during robot movements.
Since the framework used to develop this solution is ROS2, a conversion of bag files is performed: message in ROS1 format are re-published in ROS2 topics, then they are collected in a ROS2 bag file.
In ROS2 Foxy, a bag file is manipulated using the following commands:

- **ros2 bag record** *<topic_name>*: content of topics referred to the name is

collected in one binary file (it is possible to collect more than one topic at once).

- **ros2 bag play** *<bag_name>*: messages are published in the topic referred to the same name originally taken during recording phase.

4.1.1 LidarSlam dataset

The most used testing dataset is referred to [36]. The author of odometry source code releases a dataset where a mobile robot, equipped with lidar sensors, circumnavigate a position.
Since locations are crossed two times it is possible to test loop closure detection both in the local and collaborative implementations. In the inter-robot loop closure detection testing, the dataset is split in two parts and it is simulated that those are collected from different robots.
The dataset is collected using a Velodyne VLP-32 sensor.
Path crossed by robot is represented in figure 4.1.

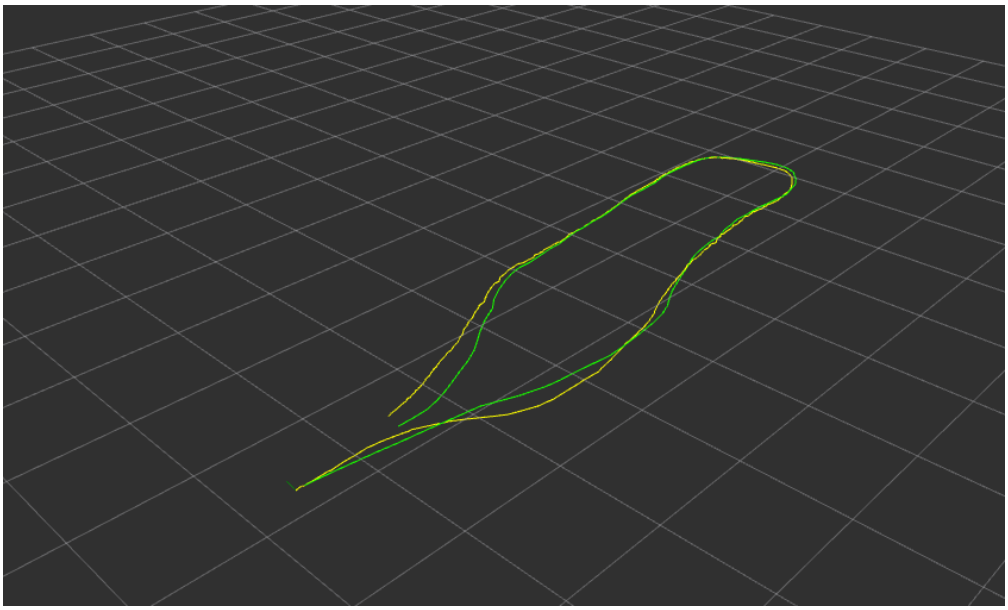

Figure 4.1: Lidarslam dataset path representation [36]

4.2 Signal Processing

Signal processing calculates odometry referred to raw lidar data coming at consecutive time instants from sensor. Furthermore it evaluate corner and surface features

taken from a raw point clouds useful in future implementation. Whether IMU data are supplied they are collected and used as a tentative adjustment of point clouds before odometry computation.
The odometry implementation is inspired by code referred to [36] (a complete open-source lidar-based local SLAM solution), where it is used an optimized multi-core version of NDT-based scan matching for odometry, an already implemented library in [37].
In this development the results from author are reconstructed and compared to another scan matching algorithm based on GICP, which is given by the PCL library [15].
The code structure is:

```
ld_slam/
├── include/
│   └── ld_slam/
│       ├── odometry.h
│       └── lidar_undistortion.h
├── src/
│   ├── odometry_node.cpp
│   ├── odometry.cpp
│   ├── imageProjection.cpp
│   └── featureExtraction.cpp
└── launch/
    └── odometry_launch.py
```

In the */include/ld_slam/* subdirectory:

- **odometry.h**: odometry object and relative methods definition.
- **lidar_undistortions.h**: it contains reference to *projectLaser* API, lidar point cloud is possibly adjusted involving IMU data.

In the *src/* subdirectory:

- **odometry_node.cpp**: ROS2 note creation.
- **odometry.cpp**: odometry implementation.
- **imageProjection.cpp**: from a point cloud, using IMU data, a deskwed representation is provided.
- **featureExtraction.cpp**: surface and corner features extraction from deskew point cloud.

In the *launch/* subdirectory is provided a launch file, with the proper *ros2 launch ld_slam odometry_launch.py* command, the odometry node and relative parameters is going in execution.

Initialization process

Only content referred to **odometry.cpp** is descripted.
In the initialization, the scan matcher is selected between **GICP** or **NDT** and relative parameter are chosen like below. The object *registration_* is a generic matcher.

```
1  if (registration_method_ == "NDT") {
2      NormalDistributionsTransform<PointXYZI, PointXYZI>::Ptr
3        ndt(new NormalDistributionsTransform<PointXYZI, PointXYZI>()
       );
4      ndt->setResolution(ndt_resolution);
5      ndt->setTransformationEpsilon(trans_eps);
6      ndt->setNeighborhoodSearchMethod(pclomp::DIRECT7);
7
8      registration_ = ndt;
9    } else {
10     GeneralizedIterativeClosestPoint<PointXYZI, PointXYZI>::Ptr
       gicp(new GeneralizedIterativeClosestPoint<PointXYZI, PointXYZI
       >());
11     gicp->setMaxCorrespondenceDistance(gicp_corr_dist_threshold);
12     gicp->setTransformationEpsilon(trans_eps);
13     registration_ = gicp;
14   }
```

Algorithm 4.1: Matcher selection algorithm

Later on publishers and subscribers are initialed.
As every topic working in the solution, the names are preceded by the prefix *"/ld_slam/"*. Considering the odometry node running on generic robot α, here it is considered as *"robot_0/"*.
To summarize, every name of topics in nodes has the prefix *"/robot_numRobot/ld_slam/"*.
Many subscription topics are used in solution:

- **initial_pose_sub_**:
  - **Name**: prefix + "/initial_pose"
  - **Massage type**: geometry_msgs::msg::PoseStamped.
  - **Callback functionalities**: collects initial pose and updates current pose.
- **imu_sub_**:
  - **Name**: prefix + "/imu"
  - **Massage type**: sensor_msgs::msg::Imu.
  - **Callback functionalities**: collects imu information, imu data are converted into euler angles representation and **lidar_undistortion_** object

is initialed (the object associated to *lidar_undistortions.h* library functionalities.

- **input_cloud_sub_**:
  - **Name**: prefix + "/point_cloud_raw"
  - **Massage type**: sensor_msgs::msg::PointCloud2.
  - **Callback functionalities**: core of the odometry algorithm, it is descripted in the following subsection.

Information computed are shared using the publishers, here a subset is presented:

- **pose_pub_**
  - **Name**: prefix + "/current_pose"
  - **Massage type**: geometry_msgs::msg::PoseStamped.
  - **Message content**: current pose estimates as output of odometry computation.
- **map_pub_**:
  - **Name**: prefix + "/map"
  - **Massage type**: sensor_msgs::msg::PointCloud2.
  - **Message content**: the node evaluates a first version of the map exploiting odometry information, the map is provided using this publisher.
- **odom_pub_**:
  - **Name**: prefix + "/odom"
  - **Massage type**: nav_msgs::msg::Odometry.
  - **Message content**: odometry informations are converted in the message format supported by nav2 stack, in order to perform navigation.

Main algorithm implementation

As mentioned before, **input_cloud_sub** associated callback is the location of main algorithms implementation.
Having new lidar scan as input:

1. If it is acquired imu data, **adjustDistortion** method of the object *lidar_undistortion_*| is called, so in the lidar scan natural distortion is recovered.

2. Lidar scan is filtered by a **voxel_grid-based** filter, a downsample technique necessary to reduce computational time [38] (during testing phase, it is shown that this filtering can not be neglected otherwise computational time becomes of time seconds order).

3. **Source** point clouds used in scan matching algorithms is set by downsampled point cloud in the **receiveCloud** method, called later on a phase of initialization of the **target** with the first point cloud coming from the sensor. The next target will become the current source point cloud.
   Before every source and target initialization, to the point cloud is previously applied a transformation matrix referred to the last pose of the robot to perform an alignment with current position.

4. Scan matching algorithm can perform since source and target are set. From the results it is taken the **transformation matrix**, useful to extract updated poses of the robot, and the **fitnessScore** used as debug and performance result.

5. Updated map and pose are published calling **publishMapAndPose** method: here the estimations are all published to the topics referred in the initialization phase. In this method, at the final instance, the target point cloud is set ready to the next scan matching step.

4.2.1 Testing

Signal Processing module is tested comparing results coming from the two types of scan matching algorithms discussed in chapter 3: NDT and GICP.
Considering many different parameters, it is shown that the NDT approach performs better than the other: the drift on estimations is contained, the algorithm can be executed a long time giving a first consistent noisy localization estimation. GICP-based odometry estimations must be passed through optimization algorithms quickly so the noise model can be applied to robot poses between a point cloud and the successive.
In this section only results coming from NDT-based odometry are presented. The solution is tested using different **resolution** parameters showing not only the first path of robot to be optimized, but also results in terms of mean ICP-score and mean time to perform estimation.
The path is taken using **RVIZ tool** publishing *Path* messages.

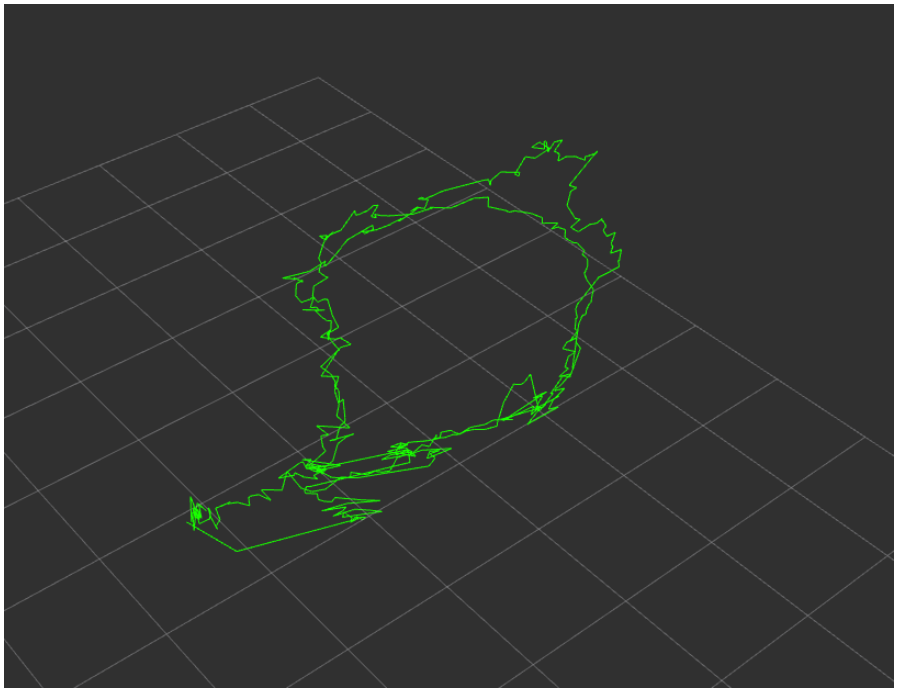

Figure 4.2: Odometry estimation using resolution set to 5.0.

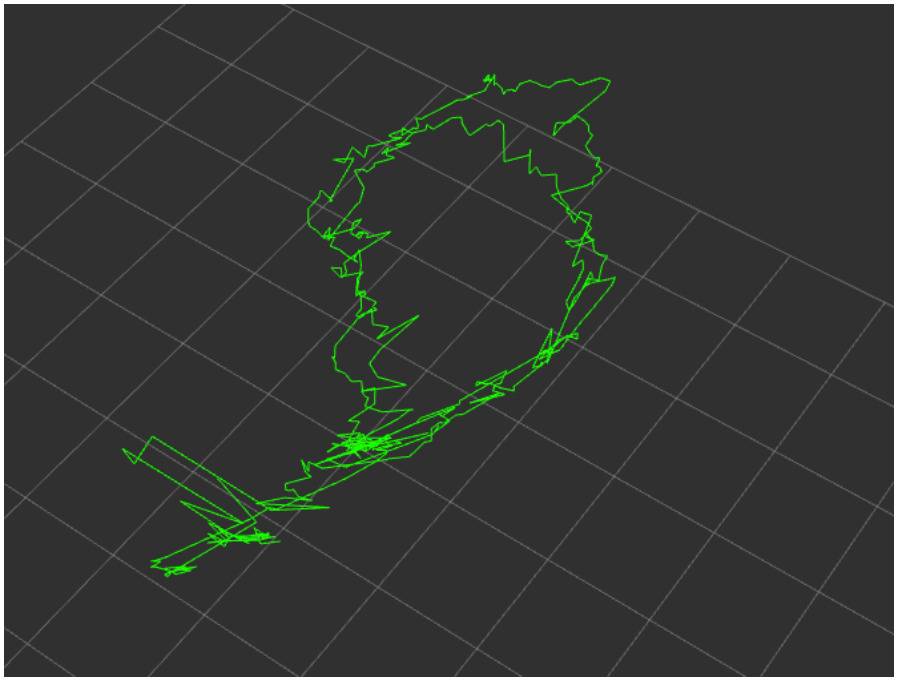

Figure 4.3: Odometry estimation using resolution set to 4.0.

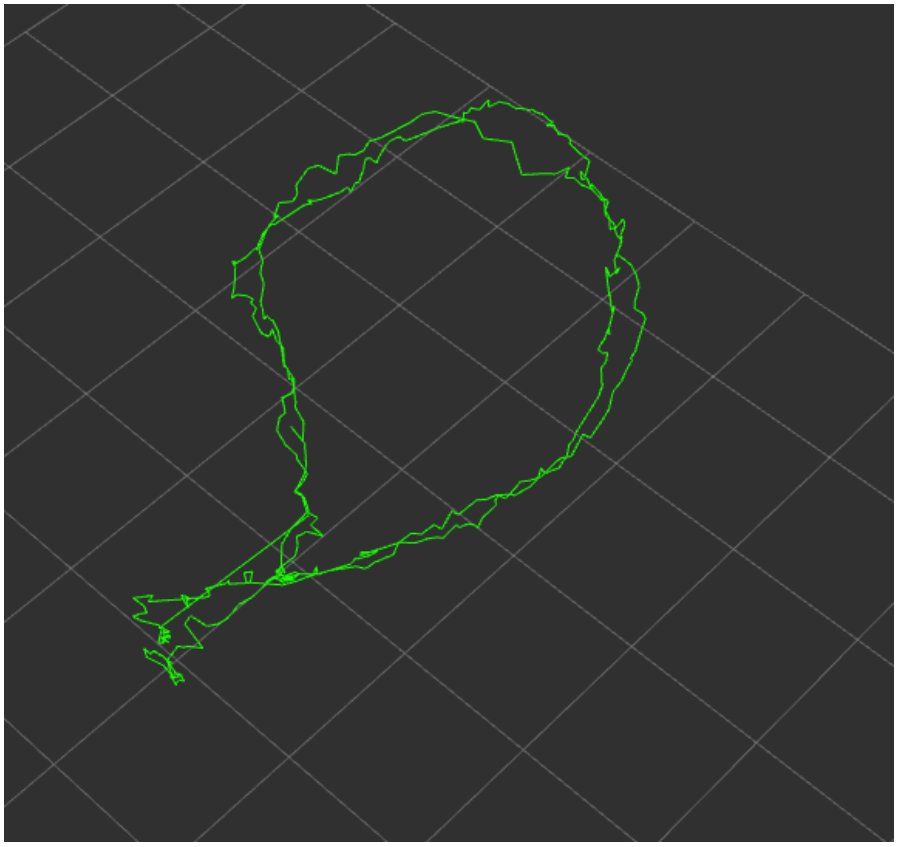

Figure 4.4: Odometry estimation using resolution set to 2.0.

Table 4.1: Signal Processing module results

resolution	mean score	mean time (ms)
5.0	5.53	130
4.0	4.39	135
2.0	4.09	127

From the above table and figure 4.4, it is determined experimentally that lower resolution produces better results in less time. Resolution can not be reduces less than 0.8 since the algorithm performance decreases.

4.2.2 Feature extraction for future upgrades

Referring to successive modules discussed later in this work thesis many improvements can be done, in order to save much more computational and communication resources.

One of the most important is regarding point cloud representation, the idea is to reduce intrinsic redundancy without suffering from low quality information. It is important to underline that downsampling is already done in the actual solution (it is necessary for ICP algorithm) but the method can be seen as "blind" since it is not dependent on the environment.
Feature extraction formed by LIO-SAM odometry [39] can be a solution to the problem: from a deskew version of point clouds (**imageProjection** node), it is calculated surface and corner features (**featureProjection** node). Combining these two results, it is possible to have a consistent representation of original raw information.
The development of nodes are taken from open-source library referred to [40]. These two nodes are tested exploiting KITTY dataset [41]. In the following, three images of the same view are presented: the first is taken from raw dataset, the second and the last is taken calculating **surface** and **corner** features.

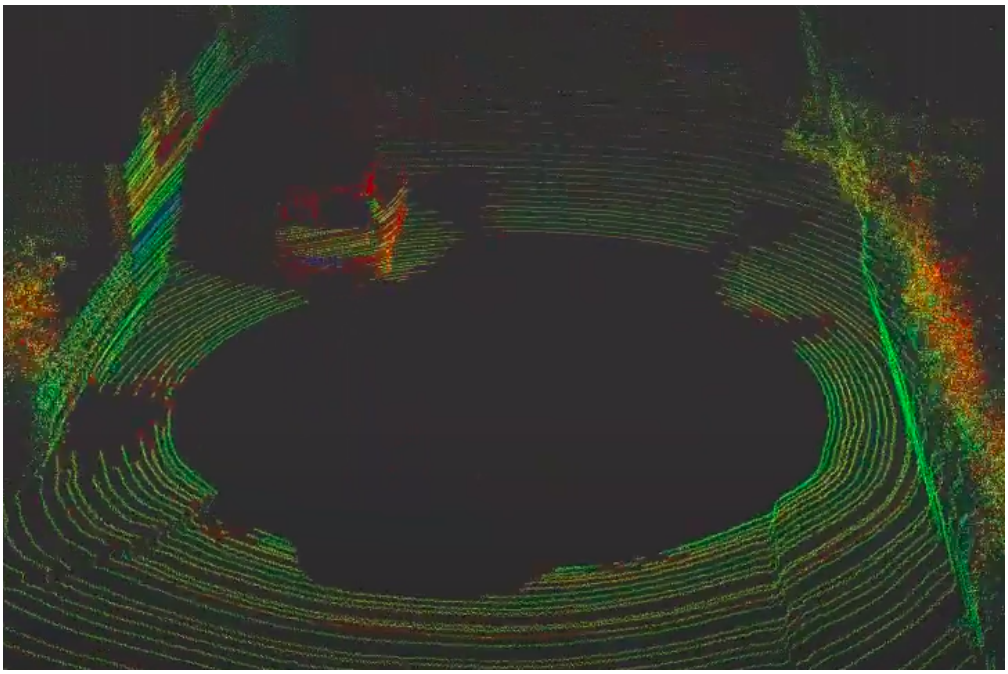

Figure 4.5: Raw KITTI dataset representation.

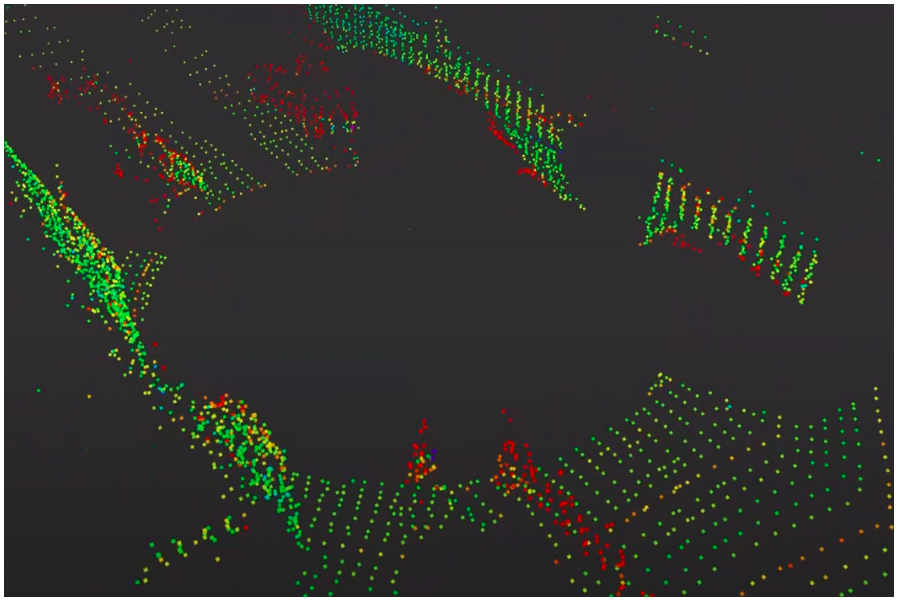

Figure 4.6: Surface features calculated.

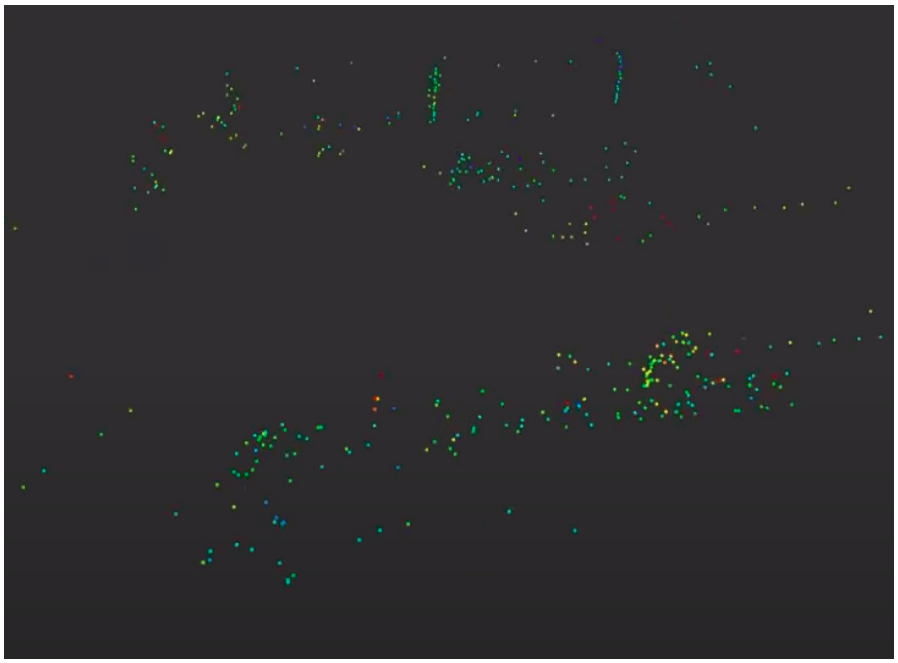

Figure 4.7: Corner features calculated.

4.3 Intra-Robot Loop Closure detection

Intra-robot loop closure detection exploits scan context descriptors to detect previously visited locations by robots.
The implementation is inspired by **SC-LeGO-LOAM** [42], a complete SLAM solution developed in ROS1 where the loop closure module is scan-context dependent. Here many changes are applied other than a porting of the code to ROS2.
The code shares the same structure with inter-robot loop closure detection module since code separation would produce further message exchanges between topics.
The structure follows:

```
ld_slam/
├── include/
│   └── ld_slam/
│       ├── loopClosureDetection.h
│       └── Scancontext.h
└── src/
    ├── loopClosureDetection.cpp
    └── Scancontext.cpp
```

In the *ld_slam/include/* subdirectory:

- **loopClosureDetection.h**: object **loopClosureDetection** with relative support informations is declared.
- **Scancontext.h**: **ScManager** object declarations with relative scan context creation and manipulation methods and parameters.

In the *ld_slam/src/* subdirectory:

- **loopClosureDetection.cpp**: algorithms implementation locations. As mentioned before, it contains code for intra- and inter-robots versions.
- **Scancontext.cpp**: code about creation and manipulation of scan contexts referred to scan context open source library [16] for the local version. Inter-robot API are created following same patterns of the library authors. It is also provided a debug API useful to enable to test the solution not in run-time.

Initialization process

Loop closure detection algorithm is called with fixed frequency, object in the ROS2 standard libray **TimerBase** are used in this context. At every frequency peak the method **performLoopClosure** is called.
Other than timer, even publisher and subscribers must be initialed. Regarding the prefix in the name of topics, it is referred what discussed in 4.2.
Many subscription topics are used in solution:

- **cloud_raw_sub_**:
  - **Name**: prefix + "/point_cloud_raw"
  - **Massage type**: sensor_msgs::msg::PointCloud2.
  - **Callback functionalities**: the point cloud is collected into a vector and **makeAndSaveScancontext** method of *scManager* is called: scan context is created as discussed in 3.2.2 and saved inside the object into a vector.
- **odom_sub_**:
  - **Name**: prefix + "/odom"
  - **Massage type**: geometry_msgs::msg::PoseStamped.
  - **Callback functionalities**: odometry estimations are collected into a vector. Robots poses are useful to align point cloud to prepare ICP-based scan matching algorithm.
- **input_cloud_sub_**:
  - **Name**: prefix + "/point_cloud_raw"
  - **Massage type**: sensor_msgs::msg::PointCloud2.
  - **Callback functionalities**: core of the odometry algorithm, it is descripted in the following subsection.

Main algorithm implementation

As mentioned before, the starting point of algorithm is the method call at frequency by *TimerBase* object. It is assumed that at least 20 measurements are collected (both coming from raw clouds and odometry estimation) in order to proceed with algorithms.
The implementation is splitted in two sub methods:

1. **detectLoopClosure**: a method of *scManager*, **detectLoopClosureID**, is called. In this method, provided by the library, it is searched the nearest descriptor from the last scan context inserted into the vector. Since scan context is a heavy representation (matrix of double of dynamic dimension), this algorithm is done in two phases: at first from the 2D dimension matrix is extracted a single row vector, where the entries are the result of an encoding function applied to the correspective row, called **ring-key**. The encoding function is:

$$\phi(r_i) = \frac{\|r_i\|_0}{N_s} \tag{4.1}$$

where r_i is the entries of vector, $\|\cdot\|$ the norm and N_s the number of sectors. Then this resultant vector is used to construct a **KD tree** to perform a **KD-tree search**, in order to find the nearest descriptors to the query (the last calculated). From all the descriptors, it is calculated the distances from each one of the set of nearest and it is selected whose has minimum value. The correspondent index is returned to the method's caller.
Whether an index is returned (and it is different to -1), a couple of indexes are formed called **loop closure candidates**.

2. Now the next part of the algorithm can be performed: using this couple of indexes, it is extracted from the vector the relative point cloud entries and those are used to set an ICP algorithm, in order to check if the two point clouds are near enough. As always in the presence of an ICP step, the target point cloud is aligned by applying the transformation matrix extracted from odometry on the source entries.
The ICP step is performed and fitnessScore is compared to a threshold: lower value means candidate loop as **valid** and relative transformation matrix can be inserted into pose-graph structure.

4.3.1 Testing

Considering the path calculated by the Signal Processing module in figure 4.4, it is shown that the robot retraces last routes to come back to the origin position. This dataset characteristic can be exploited to test both intra- and inter- loop closure algorithms.
In the testing of local version, no other additional debug methods are developed. Results are summurized in the following table, where NDT and GICP scan matching algorithm are compared both from mean ICP-scores and mean time of calculation. In particular, time of calculation is referred to the second part of the algorithm (where it is prepared and evaluated the scan matching algorithm) since the other time referred to **detectLoopClosureID** can be neglected.
From the results, it is shown that NDT-based algorithm are preferred to perform

matching algorithm	**mean score**	**mean time (s)**
NDT	0.81	42
GICP	077	51

Table 4.2: Intra-Robot Loop Closure detection module results.

scan matching also in this module. Furthermore it is evident that matching of two non-consequent point clouds is an computational expensive tasks using both

approaches. This is the real motivation to find better techniques to reduce point cloud representation as described in (4.2).

4.3.2 Future upgrades

Future upgrades of modules involved the second step of the algorithm: at the moment ICP-based algorithm preparation is done using raw data or downsampled point clouds based on voxel-grid based filtering technique. The idea is to exploit point cloud features of surface and corner to represent raw information, in order to recover computational resources.
These features are used in the preparation phase of the ICP algorithm: later on scan context matching a couple of indexes are calculated, one referred to source point cloud and the other to the target. Since the single combination of descriptors is not enough (information is not well descriptive), target point cloud must be prepared creating an history point cloud starting from the target index: the idea is to take not only the sum of corner and surface referred to it but also a subset of neighbor point clouds.
The window length is fixed, in the solution is taken as 15 precedent indexes (figure 4.8).

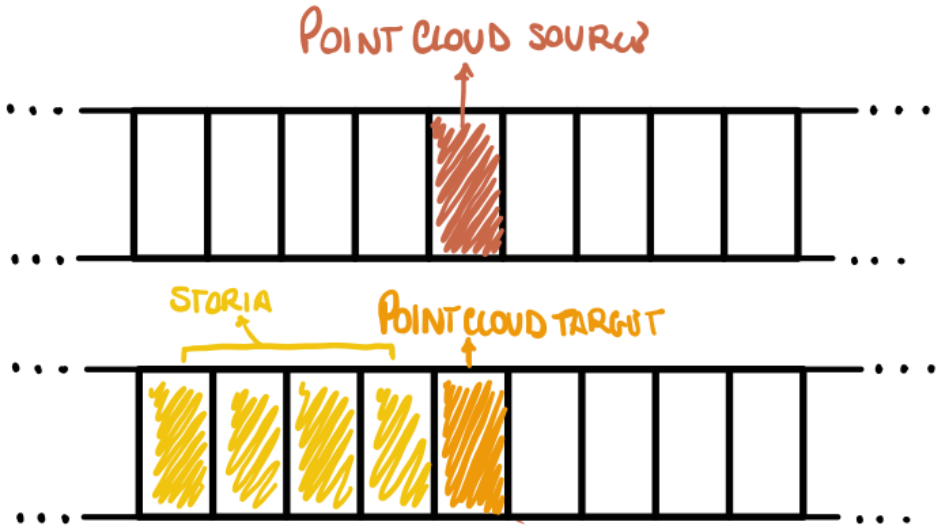

Figure 4.8: History point clouds preparation on target. Source is prepared using raw data from left index in the couple evaluated by scan context matching.

4.4 Inter-Robot Loop Closure detection

Inter-robot loop closure detection module is activated once two robots are in communication range, so they are enabled to exchange information.
Performing an algorithm based on scan context descriptors it can be possible to execute this critical module limiting amount of heavy data involved in the communication.
The code structure follows the same locations of the local version so the files contained into directories are not explained again.

Initialization process

Communication between robots is regulated by messages sharing through topics exploiting an well-managed asynchronous approach.
The initialization process is fundamental to enable topic communication: during the discovery of a new neighbor both robots must properly subscribe to a set of topics referred with correct names, in order to visualize information intended to be shared.
Since discovery was not the most important algorithm aspect of the module, the development is done considering only two robots in the environment. In the case of more robots involved, subscribers must be saved in proper structure referred with robot id (vector of subscribers can be an idea).
Since the modules of intra- and inter- submodules are referred to same source file, those data collected by local algorithm through topics discussed before are ready to use: in particular, point cloud raw data and odometry calculated.
Consider being in robot α (topic name prefix *"robot_0/"*) which have to communicate with robot β (topic name prefix *"robot_1/"*), it subscribes to topics:

- **sc_context_sub_**:
  - **Name**: "robot_1/ld_slam/loop_closure/sc_context"
  - **Massage type**: std_msgs::msg::Float64MultiArray.
  - **Callback functionalities**: scan context evaluated by robot β during its movement are published, the subscriber takes the descriptors in the format of a single vector and unpack it reconstructing the original matrix (this operation is done using index parameters set by robot β before publishing process) to be saved by **scManager**. In the library is added a method **saveOtherScancontextAndKeys** that collects neighbor's descriptors in order to perform algorithm discussed later. Even this method is developed to work only with two robots, future version must to overcome this limitation: rather than collects one single vector of descriptors, these can be stored into a map referred by robot id.

- **loop_id_sub_**:
  - **Name**: "robot_1/ld_slam/loop_closure/loop_id"
  - **Massage type**: std_msgs::msg::Int32MultiArray.
  - **Callback functionalities**: robot β requests point clouds of robot α to proceed with inter-loop detection. Those request is expressed by indexes shared with this message. Whether the point clouds referred by indexes entries are contained into correspondent vector, then those can be published as message using a publisher.
- **loop_cloud_sub_**:
  - **Name**: "robot_1/ld_slam/loop_closure/cloud"
  - **Massage type**: sensor_msgs::msg::PointCloud2.
  - **Callback functionalities**: it is inserted to the workflow associated with *loop_id_sub_*, later on a request comes from indexes and successive publishing of point clouds, this subscriber collects the results.

Otherwise the publisher are defined as:

- **sc_context_pub_**
  - **Name**: "robot_0/ld_slam/loop_closure/sc_context"
  - **Massage type**: std_msgs::msg::Float64MultiArray.
  - **Message content**: in the case of an communication interface developed, those descriptors are shared at rendezvous instant. Here descriptors are published at fixed frequency calling **publishScancontext** method: all the scan contexts are recovered from *scManager* using **getScanContext** API, then each descriptor is packed in a single vector saving index information to be used in the unpacking phase by robot β.
- **loop_id_pub_**
  - **Name**: "robot_0/ld_slam/loop_closure/loop_id"
  - **Massage type**: std_msgs::msg::Int32MultiArray.
  - **Message content**: the robot α sends a point clouds request building vector of indexes and packing into a messagge passing through this topic.
- **loop_cloud_pub_**
  - **Name**: "robot_0/ld_slam/loop_closure/cloud"
  - **Massage type**: sensor_msgs::msg::PointCloud2.
  - **Message content**: This publisher is called during loop id callback, for each of the indexes a point cloud is published through this topic.

Main algorithm implementation

Elaboration of information, previously given by the neighbor of a robot, is developed asynchronously: two timers (object of TimerBase) are instantiated during initialization, the aim is to call a method at a fixed frequency whose execution is regulated by flags.
The first method (**performInterLoopClosure**) execution proceeds only if two condition are fulfilled:

- there was exchange of information (a flag is set when the robot receives scan context into the topic)
- No other inter-loop detection is on execution at the moment.

If execution proceeds, it is called **detectInterLoopClosureID** method on *scManager* object. The method is implemented as an extension of the **detectLoopClosureID** used in the local version of the algorithm.
Two vectors of descriptors are compared: the one referred to robot α and the "other" referred to the neighbor. Later on KD-tree creation suitably updated after that a fixed number of iteration is executed (this is done in order to not unbalance KD-tree since tree structure may collapse into a linear vector, limiting the general performance), for each of the other descriptors a KD-tree search is performed.
If the search gives back a result, the indexes (corresponding to other and robot α vectors) are added into a vector of resultant indexes (**candidate loops**), previously initialized with entries equal to -1, which is returned to the caller. The vector is then filtered of those entries equal to -1 and the second step of the algorithm can be prepared.
Exactly as the local version, a scan matching algorithm must be set with point clouds corresponding to indexes previously found. A vector of point clouds source is created by robot α point clouds, the target vector clouds is then created when the robot β responds to a query formed by indexes corresponding to those point clouds needed. The query is done using the corresponding topic. When robot β receives the query it responds back with point clouds to the topic referred to last subsection.
In the callback of subscriber of this topic a flag is plugged on enabling the execution of the second method (**performInterLoopClosure_callback**) called at a given frequency by *TimerBase* object. In the method vector of target point clouds is formed so an ICP-based algorithm can be executed: corresponding robot α odometry estimation gives a transformation applied for each couple of sources and targets, the target is downsampled with voxel-grid based approach, then scan-matching algorithm can be performed.
If the ICP score is below a threshold then an inter-robot loop is detected, the corresponding couple of index candidates gives a relative transformation matrix to

be added to pose-graph.
Then all the flags are unplugged to enable successive detections.

4.4.1 Testing

Since the communication interface is not developed yet, the module is not tested at run time. The analysis is focussed only on the single module functionalities.
In the testing process, dataset characteristics are stressed identifying two subsets in the path crossed by the robot: one regarding to the point clouds referred to the first trajectory and the others referred to the inverse part (figure 4.9). Those two subset of point clouds are published in different topics simulating classical behavior of robots during an algorithm run.

In the topic's callback referred to the second part of the trajectory, the point

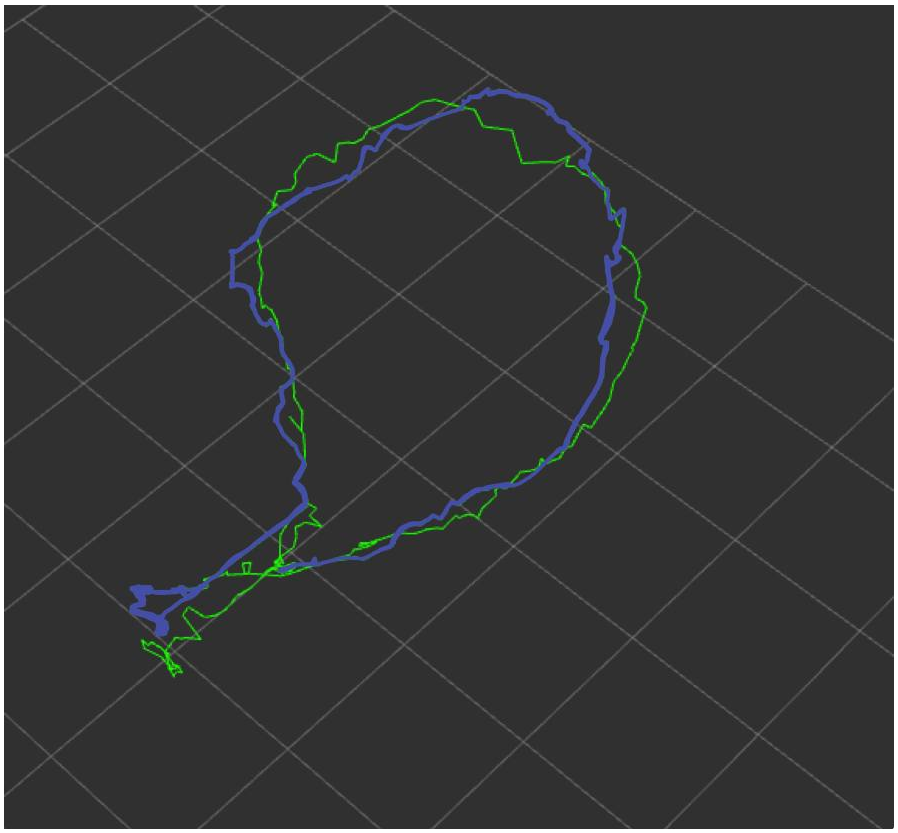

Figure 4.9: Two path considered in the testing process (blue is the reversed trajectory).

clouds are firstly stored into a vector and then passed to a method developed only for debugging purpose: **makeAndSaveOtherScancontextAndKeys** is a

method of *SCManager* object, the main subject is to calculate and store scan context associate descriptor.
Since the two elements provided to a robot by its neighbor during a possible rendezvous are stored and calculated (vector of neighbor point clouds and scan context into *SCManager* object), the inter-robot loop closure detection algorithm can be performed.
The testing results are quite similar to the local version of the algorithm in terms of mean scores and mean time for elaboration. The two matching algorithm considered before are compared.
Exactly as discussed in the previous section, time of elaboration is taken considering

matching algorithm	**mean score**	**mean time (s)**
NDT	0.93	33
GICP	0.64	58

Table 4.3: Inter-Robot Loop Closure detection module results.

only the second part of the algorithm related to matching of two non-subsequent point clouds since the other algorithm contribution (coming from communication interface in the majority) can not be evaluated. Elaboration time is high as previous results, the introduction of point clouds descriptors as corner and surface can be an important improvement to overcome this problem.

4.4.2 Future upgrades

The natural upgrade is to create target point cloud not from raw information shared by robot α but requesting an history point cloud created using surface and corner descriptors, as discussed in the previous subchapter. The amount of informations shared during communication decreases saving bandwidth.

Chapter 5

Conclusion

Collaborative Simultaneous Localization and Mapping (SLAM) problem is quite hard to accomplish since many fields are involved: from point cloud elaboration of information to communication challenges. By choosing one suite of sensors with respect to another, many of these components might be reformulated, discouraging modularity of solutions.
In this thesis work a possible distributed solution based on lidar sensing is designed, the design is not provided only in terms of what is necessary to add but also giving an theoretical overview where libraries rest.
Later on this documentation work, a first development of sensing modules is provided: Signal Processing module and both Intra- and Inter-robot Loop closure detection modules. In particular, the Inter-robot Loop Closure detection module reveals to be a possibly good approach to resolve computational and resource constraints.
After that it is provided possible upgrade ideas all based on downsample feature-based representation of raw data already provided in Signal Processing.

Future work

Complete solution must be developed, since only a first not-filtered pose graph structure is created at the end of the work. Outlier Rejection and Distributed Mapper modules must be implemented exploiting open-source libraries and theoretical aspects referred into Chapter 3.
After that many improvements to the already modules implemented can be done following suggestions discussed in the relative subchapters.

Acknowledgements

Sono alla fine di questo percorso e di questi anni, in questo piccolo spazio vorrei riempirmi la bocca di doverosi **grazie**, poiché se sono qui oggi non è solo per mio merito.
Il primo è rivolto alla professoressa Marina Indri, per i numerosi consigli e gli incoraggiamenti che mi hanno accompagnato durante tutto il percorso di elaborazione e stesura di questo lavoro. Ringrazio Gianluca Prato, David Pangcheng ed Enrico Ferrera per la pazienza di ascoltarmi a cadenza bi-settimanale in tutti questi mesi, donandomi parte della risorsa che più di tutte è preziosa: il tempo. Mi avete insegnato a individuare valore quando si è accecati e offuscati da aspettative.
Ringrazio Fondazione LINKS per avermi permesso, con questa opportunità, di muovere i primi passi in un settore spesso elitario come la robotica.
Ringrazio la mia famiglia: mamma, papà, Serena e nonna Angela. Grazie per la vostra vicinanza e perenne supporto.
Ringrazio Alessandro e Nikole, per avermi fatto sentire a casa anche fuori dal nido.
Ringrazio i miei amici tutti, vecchi e nuovi, per accettarmi per quello che sono, senza filtri e maschere.

Bibliography

[1] Brianna Wessling. *100,000+ mobile robots shipped in 2021*. Link (cit. on p. 1).

[2] U. Frese, Wagner, and R. Röfer. «A SLAM Overview from a User's Perspective». In: 24 (2010), pp. 1–2 (cit. on p. 1).

[3] MHL Staff. *Order Fulfillment Mobile Robots Start to Deliver*. (cit. on p. 2).

[4] Robotics Online Marketing Team. *Service Robots Improve Defense Industry's Safety Standards*. (cit. on p. 2).

[5] Pierre-Yves Lajoie, Benjamin Ramtoula, Fang Wu, and Giovanni Beltrame. «Towards Collaborative Simultaneous Localization and Mapping: a Survey of the Current Research Landscape». In: *Field Robotics* 2.1 (Mar. 2022), pp. 971–1000. DOI: 10.55417/fr.2022032. URL: https://doi.org/10.55417%2Ffr.2022032 (cit. on pp. 3, 5, 7, 8).

[6] Zhi Yan, Luc Fabresse, Jannik Laval, and Noury Bouraqadi. «Team Size Optimization for Multi-robot Exploration». In: Oct. 2014, pp. 438–449. ISBN: 978-3-319-11899-4. DOI: 10.1007/978-3-319-11900-7_37 (cit. on p. 5).

[7] Tixiao Shan, Brendan Englot, Drew Meyers, Wei Wang, Carlo Ratti, and Rus Daniela. «LIO-SAM: Tightly-coupled Lidar Inertial Odometry via Smoothing and Mapping». In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2020, pp. 5135–5142 (cit. on p. 9).

[8] Kamak Ebadi et al. *LAMP: Large-Scale Autonomous Mapping and Positioning for Exploration of Perceptually-Degraded Subterranean Environments*. 2020. DOI: 10.48550/ARXIV.2003.01744. URL: https://arxiv.org/abs/2003.01744 (cit. on pp. 12, 13).

[9] P. Lajoie, B. Ramtoula, Y. Chang, L. Carlone, and G. Beltrame. «DOOR-SLAM: Distributed, Online, and Outlier Resilient SLAM for Robotic Teams». In: *IEEE Robotics and Automation Letters* 5.2 (2020), pp. 1656–1663 (cit. on pp. 15, 17, 43–45).

[10] Francisco Martín Rico. *A Concise Introduction to Robot Programming with ROS2*. CRC Press/Chapman Hall, 2022. ISBN: 9781032267203; 1032267208; 9781032264653; 1032264659; 9781003289623; 1003289622 (cit. on pp. 18, 19).

[11] The Robotics Back-End. *ROS1 vs ROS2, Practical Overview For ROS Developers*. Link (cit. on p. 19).

[12] Open Robotics. *ROS2 Foxy Documentation: Concepts*. Link (cit. on pp. 19, 20).

[13] Steven Macenski, Francisco Martin, Ruffin White, and Jonatan Ginés Clavero. «The Marathon 2: A Navigation System». In: *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2020 (cit. on pp. 20, 21).

[14] Michele Colledanchise and Petter Ögren. *Behavior Trees in Robotics and AI*. CRC Press, July 2018. DOI: 10.1201/9780429489105. URL: https://doi.org/10.1201%2F9780429489105 (cit. on p. 22).

[15] Radu Bogdan Rusu and Steve Cousins. «3D is here: Point Cloud Library (PCL)». In: *IEEE International Conference on Robotics and Automation (ICRA)*. Shanghai, China: IEEE, May 2011 (cit. on pp. 24, 27, 56).

[16] 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)*. 2018, pp. 4802–4809. DOI: 10.1109/IROS.2018.8593953 (cit. on pp. 24, 40–42, 64).

[17] Open Robotics. *Factor Graphs and GTSAM*. Link (cit. on p. 25).

[18] Frank Dellaert et al. *borglab/gtsam*. Version 4.2a7. May 2022. DOI: 10.5281/zenodo.5794541. URL: https://doi.org/10.5281/zenodo.5794541 (cit. on p. 25).

[19] Dirk Holz, Alexandru E. Ichim, Federico Tombari, Radu B. Rusu, and Sven Behnke. «Registration with the Point Cloud Library: A Modular Framework for Aligning in 3-D». In: *IEEE Robotics Automation Magazine* 22.4 (2015), pp. 110–124. DOI: 10.1109/MRA.2015.2432331 (cit. on pp. 27, 28, 30, 32, 35, 36).

[20] Marius Muja and David G. Lowe. «Fast Approximate Nearest Neighbors with Automatic Algorithm Configuration». In: *International Conference on Computer Vision Theory and Applications (VISAPP'09)*. 2009 (cit. on p. 29).

[21] Aleksandr Segal, Dirk Hähnel, and Sebastian Thrun. «Generalized-ICP». In: June 2009. DOI: 10.15607/RSS.2009.V.021 (cit. on p. 33).

[22] Peter Biber and Wolfgang Straßer. «The Normal Distributions Transform: A New Approach to Laser Scan Matching». In: vol. 3. Nov. 2003, 2743–2748 vol.3. ISBN: 0-7803-7860-1. DOI: 10.1109/IROS.2003.1249285 (cit. on p. 36).

[23] Feng Lu and Milios. «Robot pose estimation in unknown environments by matching 2D range scans». In: *1994 Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*. 1994, pp. 935–938. DOI: 10.1109/CVPR.1994.323928 (cit. on p. 37).

[24] Joshua G. Mangelson, Derrick Dominic, Ryan M. Eustice, and Ram Vasudevan. «Pairwise Consistent Measurement Set Maximization for Robust Multi-Robot Map Merging». In: *2018 IEEE International Conference on Robotics and Automation (ICRA)*. 2018, pp. 2916–2923. DOI: 10.1109/ICRA.2018.8460217 (cit. on p. 45).

[25] Randall Smith, Matthew Self, and Peter C. Cheeseman. «Estimating Uncertain Spatial Relationships in Robotics». In: *CoRR* abs/1304.3111 (2013). arXiv: 1304.3111. URL: http://arxiv.org/abs/1304.3111 (cit. on p. 45).

[26] Wikipedia. *Mahalanobis distance*. Link (cit. on p. 46).

[27] Bharath Pattabiraman, Md. Mostofa Ali Patwary, Assefaw Hadish Gebremedhin, Wei-keng Liao, and Alok N. Choudhary. «Fast Algorithms for the Maximum Clique Problem on Massive Graphs with Applications to Overlapping Community Detection». In: *CoRR* abs/1411.7460 (2014). arXiv: 1411.7460. URL: http://arxiv.org/abs/1411.7460 (cit. on pp. 47, 48).

[28] Siddharth Choudhary, Luca Carlone, Carlos Nieto, John Rogers, Henrik I. Christensen, and Frank Dellaert. *Distributed Mapping with Privacy and Communication Constraints: Lightweight Algorithms and Object-based Models*. 2017. DOI: 10.48550/ARXIV.1702.03435. URL: https://arxiv.org/abs/1702.03435 (cit. on pp. 48, 49).

[29] Akash Sharma Siddharth Choudhary Antonella Wilby. *Distributed-Mapper*. Link (cit. on pp. 48, 53).

[30] Luca Carlone, David Rosen, Giuseppe Calafiore, John Leonard, and Frank Dellaert. *Lagrangian Duality in 3D SLAM: Verification Techniques and Optimal Solutions*. 2015. DOI: 10.48550/ARXIV.1506.00746. URL: https://arxiv.org/abs/1506.00746 (cit. on p. 50).

[31] Wikipedia. *von Mises distribution*. Link (cit. on p. 50).

[32] Luca Carlone, Roberto Tron, Kostas Daniilidis, and Frank Dellaert. «Initialization techniques for 3D SLAM: A survey on rotation estimation and its use in pose graph optimization». In: *2015 IEEE International Conference on Robotics and Automation (ICRA)*. 2015, pp. 4597–4604. DOI: 10.1109/ICRA.2015.7139836 (cit. on p. 50).

[33] Rick Wicklin. *Least-squares optimization and the Gauss-Newton method*. Link (cit. on p. 50).

[34] Wikipedia. *Exponential map (Lie theory)*. Link (cit. on p. 51).

[35] ROS Wiki. *Bags*. Link (cit. on p. 54).

[36] Ryohei Sasaki. *lidarslam_ros2*. Link (cit. on pp. 55, 56).

[37] koide3. *ndt_omp*. Link (cit. on p. 56).

[38] PCL documentation. *Downsampling a PointCloud using a VoxelGrid filter*. Link (cit. on p. 59).

[39] 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)*. 2020, pp. 5135–5142. DOI: 10.1109/IROS45743.2020.9341176 (cit. on p. 62).

[40] Tixiao Shan. *LIO-SAM*. Link (cit. on p. 62).

[41] Andreas Geiger, Philip Lenz, Christoph Stiller, and Raquel Urtasun. «Vision meets Robotics: The KITTI Dataset». In: *International Journal of Robotics Research (IJRR)* (2013) (cit. on p. 62).

[42] RPM Robotics Lab. *SC-LeGO-LOAM*. Link (cit. on p. 64).