POLITECNICO MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE E DELL'INFORMAZIONE

Improved Graph SLAM with OpenStreetMap Priors

Tesi di Laurea Magistrale in Computer Science and Engineering - Ingegneria Informatica

Authors: **Massimo Del Tufo, Stefano Del Tufo**

Student IDs: 952609, 915994
Advisor: Prof. Matteo Matteucci
Co-advisor: Ing. Matteo Frosi
Academic Year: 2021-22

Abstract

Simultaneous Localization And Mapping (SLAM) consists in the concurrent construction of a model of the environment (the map), and the estimation of the pose of the robot moving within it. To obtain accurate reconstruction of the scene, in the last decade many LiDAR-based SLAM systems have been proposed, exploiting dense point clouds to track the motion of the robot and build a 3D map. Existing systems, however, usually do not include additional information like prior maps, which may prove to be useful in the whole SLAM process. For this reason, we improve an existing system that includes 2D maps information (prior map) to solve the SLAM problem. The maps correspond to the shapes of the buildings of a city and are provided by OpenStreetMap (OSM) [14]. By processing data coming from an on-board LiDAR, the robot perceives the surrounding buildings and tries to match them with the prior map. This way, the robot will simultaneously localize itself and correct the maps while moving. To deal with this problem, an existing Graph-based SLAM system (baseline [4]) is used and improved, as main contribution of this thesis. The system consists of a graph (pose graph) whose nodes correspond to the poses of the robot and the buildings (i.e. their centroids) and whose edges represent constraints between the poses. To improve the baseline system, we developed different algorithms and procedures. First we implemented a procedure to asynchronously download and buffer the maps to deal with the long OSM servers response time. In order to extract the relevant features from the LiDAR point clouds, we implemented an algorithm to extract line segments using RANSAC [15]. Moreover, we present a line segments based scan matching algorithm to extract the map priors along with its extension to generate geometrical constraints in order to prevent the overlapping of the buildings within the map. Finally, leveraging the g2o API [13], we partitioned the pose graph in order to balance the localization and map constraints. We compared the developed system against the baseline using the KITTI dataset [9], showing that our system has greatly improved the baseline system, reducing the robot localization error and generating visually more consistent maps.

Keywords: Robots, Line segment scan matching, Graph SLAM, Map prior, Localization,

Mapping

Abstract in lingua italiana

Simultaneous Localization And Mapping (SLAM) consiste nella costruzione simultanea di un modello dell'ambiente (la mappa) e nella stima della posa del robot che si muove al suo interno. Per ottenere una ricostruzione accurata della scena, nell'ultimo decennio sono stati proposti molti sistemi SLAM basati su LiDAR, che sfruttano dense nuvole di punti (point clouds) per tracciare il movimento del robot e costruire una mappa 3D. I sistemi esistenti, tuttavia, di solito non includono informazioni aggiuntive come l'utilizzo di mappe preesistenti (map priors), che possono rivelarsi utili nell'intero processo SLAM. Per questo motivo, miglioriamo un sistema esistente che include mappe 2D preesistenti per risolvere il problema SLAM. Le mappe corrispondono alle forme degli edifici di una città e sono fornite da OpenStreetMap (OSM) [14]. Elaborando i dati provenienti da un LiDAR, il robot percepisce gli edifici circostanti e cerca di abbinarli alla mappa precedente. In questo modo, il robot simultaneamente si localizzerà e correggerà le mappe mentre si muove. Per affrontare questo problema, viene utilizzato e migliorato un sistema SLAM basato su grafi esistente (baseline [4]), come contributo principale di questa tesi. Il sistema è costituito da un grafo (grafo delle pose) i cui nodi corrispondono alle pose del robot e degli edifici e i cui archi rappresentano vincoli tra le pose. Per migliorare la baseline, abbiamo sviluppato diversi algoritmi e procedure. Inizialmente abbiamo implementato una procedura per scaricare e bufferizzare in modo asincrono le mappe per gestire il lungo tempo di risposta dei server di OSM. Per estrarre le caratteristiche rilevanti dalle point cloud del LiDAR, abbiamo implementato un algoritmo per estrarre le linee usando RANSAC [15]. Inoltre presentiamo un algoritmo di corrispondenza delle point cloud (scan matching) basato su linee per estrarre i vincoli (map priors) dalla mappa di OSM insieme alla sua estensione per generare vincoli geometrici al fine di non far sovrapporre gli edifici nella mappa. Infine, sfruttando l'API g2o [13], abbiamo partizionato il grafo delle pose per bilanciare i vincoli sulla localizzazione e quelli estratti dalla mappa. Abbiamo confrontato il sistema sviluppato con la baseline utilizzando un dataset di KITTI [9], dimostrando che il nostro sistema ha notevolmente migliorato la baseline, riducendo l'errore di localizzazione del robot e generando mappe visivamente più coerenti.

Parole chiave: Robots, Scan matching con linee, Graph SLAM, Map prior, Localizzazione, Mappatura

Contents

Abstract **i**

Abstract in lingua italiana **iii**

Contents **v**

1 Introduction **1**

1.1 Thesis Contributions . . . 2

2 Graph SLAM with Map Priors **5**

2.1 Graph-based SLAM . . . 5

2.2 ROS . . . 6

2.3 System Architecture . . . 8

2.4 HDL Graph SLAM . . . 9

2.5 Baseline system . . . 12

2.5.1 Downloading of the Buildings . . . 12

2.5.2 LiDAR Point Clouds 2D Conversion . . . 13

2.5.3 Map Priors Extraction . . . 13

2.5.4 Inserting Graph Edges and Optimization . . . 14

3 Improvements **15**

3.1 OpenStreetMap XML Buffering . . . 15

3.2 Line Segments Extraction with RANSAC . . . 17

3.3 Line Segment Based Scan Matching . . . 19

3.3.1 Edges Extraction and Matching . . . 21

3.3.2 Choosing the Best Transformation using Edges . . . 21

3.3.3 Choosing the Best Transformation using Lines . . . 22

3.3.4 Specialized Fitness Function . . . 23

3.3.5 Computed Information Matrix . . . 27

3.4 Graph Partitioning using g2o levels 27
3.5 Overlapped Buildings Problem Resolution 31

4 Results **35**
4.1 Test Data and Evaluation Metrics 35
4.1.1 KITTI Odometry Benchmark 35
4.1.2 RPE 39
4.1.3 ATE 40
4.2 Final Results 41
4.2.1 Errors 41
4.2.2 Visual Evaluation 42
4.2.3 Space and Time Efficiency 50

5 Conclusions **51**
5.1 Future Improvements 51

Bibliography **53**

A Appendix A **55**
A.1 Signed Vector Projection Norm 55

List of Figures **57**

List of Tables **59**

Acknowledgements **61**

1 | Introduction

Today we can see many examples of robotics system being used to deal with tasks that were once carried out by man. Robotic systems are proven to be particularly useful to solve problems that are dangerous or difficult for humans. These include mobile robots that deal with automated cleaning, farming and ranching, cargo delivery, search and rescue in hazardous areas, and many more. These mobile robots must work safely and reliably in complex and unknown environments, relying only on their perceptions obtained from the on-board sensors, while also performing path planning. So, to be able to fulfill these tasks, the robot needs an accurate map and an estimate of its pose within it.

By definition, Simultaneous Localization And Mapping (SLAM) addresses the problem of building a model of the environment surrounding the robot, i.e. the map, while simultaneously estimating the state of the robot moving inside it. The state of the robot generally coincides with its pose, which includes position and orientation, while the map includes various features of the surrounding environment, such as landmarks (distinctive map features), obstacles or buildings. In addition to the map, created while the robot is moving, the robot stores a lot of data acquired along the trajectory. This data, perceived through the robot sensors, is used to correct the estimate of the robot pose, in particular by means of the LiDAR scan matching.

In the last 30 years, many solutions have been proposed to solve the SLAM problem. The most famous systems are based on a probabilistic formulation of the problem, for example by using Extended Kalman Filters (EKF) [19], Rao-Blackwellized particle filters [2], and maximum likelihood estimation [5].

In addition to these, which were the most used, other solutions based on cameras and laser range sensors have also been developed. The LiDAR scanners, which are able to provide precise scans of the environment in either 2D or 3D, allow to obtain good localization and mapping results. Finally, it is also worth mentioning another family of algorithms that solve the SLAM problem formed by graph-based methods, also known as Graph SLAM. In these systems a graph is constructed, whose nodes coincide with the robot poses and whose edges represent constraints between nodes, extracted from the perceptions of the

sensors.

Once the graph is built, it is optimized by adjusting its nodes in order to minimize the error given by its edges. The edges encode also the loop closure constraints found during the robot trajectory. When performing a loop closure, the algorithm matches the information between the current sensed environment with previously sensed data. Specifically, when the robot goes by places already seen before and it finds a matching between the current and previous sensed data, it will correct the current pose by adding a special edge in the graph, which represents the information associated to the loop.

Within the system we want to improve [4] (from now on we will refer to this system as the **baseline system**), to increase the SLAM accuracy, pre-computed maps have been introduced. The 2D prior maps allow the robot to enhance its estimated pose with additional information. This is derived from the alignment between the input LiDAR scans and the pre-computed maps.

1.1. Thesis Contributions

In this thesis, our goal is to improve an already existing Graph SLAM system, which combines LiDAR data and prior maps downloaded from OpenStreetMap (OSM) [14]. This algorithm allows the robot to be independent of GPS data, except to estimate the initial robot pose in order to download the first map. The baseline system was created with the intent of dealing with situations where the GPS might not have a stable signal, for example in tunnels or GNSS-denied zones.

The baseline system has multiple issues that compromise its execution and accuracy. One of its issues is related to the downloading process of the maps from the OSM servers. The baseline execution time, since it synchronously waits the response of the OSM servers to download the maps, is constrained and sometimes blocked to wait the OSM servers long latency. Moreover, if the OSM gets temporarily unavailable the whole execution will be compromised. Another notable issue is its sensitivity to outliers within the sensed LiDAR point clouds. In fact, as shown in Chapter 4, the correction of the buildings poses is frequently compromised by the building surroundings, such as fences and vegetation. Due to the high number of map prior constraints, integrated into the pose graph, the localization of the robot was worsened w.r.t. two consecutive poses. In fact the new constraints make the robot poses to be corrected too heavily and sometimes erroneously. This issue can lead to enormous adjustments of the LiDAR estimated odometry, which is really precise on short travelled distances and shouldn't be corrected so much, making the baseline system less robust to map priors erroneously extracted. Lastly, the baseline system does not have

any constraint between the neighbouring buildings. For this reason, when adjusting the building poses, two neighbouring buildings could be overlapped, generating not consistent maps.

To address all these issues, in this thesis we propose an improved version of the system, obtained thanks to many contributions. Our first contribution was to integrate a buffer in the downloading process of the OSM maps. This enabled the proposed system to run asynchronously from the long OSM servers response time.

Then, we enhanced the prefiltering process of the LiDAR point clouds to remove the numerous outliers. Moreover, we implemented an algorithm to extract relevant features of the LiDAR point clouds, i.e., line segments that outline a building.

To follow the novel line segments extraction procedure, we implemented a new line segments-based scan matching algorithm from scratch. This method is particularly suited for the scenario considered, as it extracts the map priors by aligning the extracted line segments of the LiDAR scans with the ones given by the downloaded OSM maps. This algorithm proves to be more robust than the previously used ICP [1] algorithm, which was used by the baseline system to extract the map priors.

The alignments between LiDAR and the OSM map provide the needed constraints in order to fix the poses of both robot and buildings. Each extracted constraint is embedded into the pose graph as an edge, which connects the nodes corresponding to some of the buildings and robot poses along the robot trajectory, respectively. Finally, in order to prevent the overlapping of the buildings within the map, we added a new constraint between two building nodes, which will rigidly constrain the neighboring buildings to not overlap.

2 | Graph SLAM with Map Priors

In this chapter we present the details of the Graph-based SLAM system we want to improve [4], we will refer to this system as the **baseline system**. In Section 2.1 we describe the formulation of the SLAM problem using graphs. The implementation of the system is based on the Robotic Operating System (ROS) [17], which is a set of software libraries and tools that help to build robot applications, as described in Section 2.2. The baseline system is an extension of HDL Graph SLAM [7], and the common aspects of their system architectures is discussed in Section 2.3. To better understand the baseline system, we first describe HDL Graph SLAM in Section 2.4, followed by its extension (i.e., the current baseline) in Section 2.5.

2.1. Graph-based SLAM

As anticipated in Chapter 1, a graph-based SLAM system builds a pose graph mode and optimize the localization and mapping problem. The graph nodes represent the robot poses (position and orientation), while the graph edges represent the constraints (as relative poses) between the nodes.

The poses are expressed as transformation matrices, an handy mathematical formulation to embed rotation and translation in a single structure. The properties of the transformation matrix allow to compose transformations by simple matrix multiplication and calculation of the inverse. The transformation matrices have the following structure:

$$\left\{ \begin{bmatrix} R & t \\ 0 & 1 \end{bmatrix} \mid R \in SO(2), t \in \mathbb{R}^2 \right\} \tag{2.1}$$

Since we are constructing a pose graph in 2D space, t is the 2x1 translation vector (position x and y) and R is the 2x2 rotation matrix of the transformation (orientation θ). SO(2) is the special orthogonal group in which are contained the rotation matrices. In other words, every transformation matrix gives the information of a pose $< x, y, \theta >$, position and orientation, with reference to a selected reference system of coordinates.

Every node and edge of the pose graph is associated to a transformation matrix. The nodes represent the poses of the robot, while the edges (or constraints) contain the information about the relative poses (or relative transformations) between two nodes. All constraints are used to build an error function to be minimized during the optimization of the graph. After this step, the nodes of the graphs are moved from their initial position and orientation, in order to better satisfy the constraints.

The robot, while it is moving, incrementally constructs the pose graph. The nodes of the robot are initialized to the odometry estimated position, while the edges are added based on the sensed data.

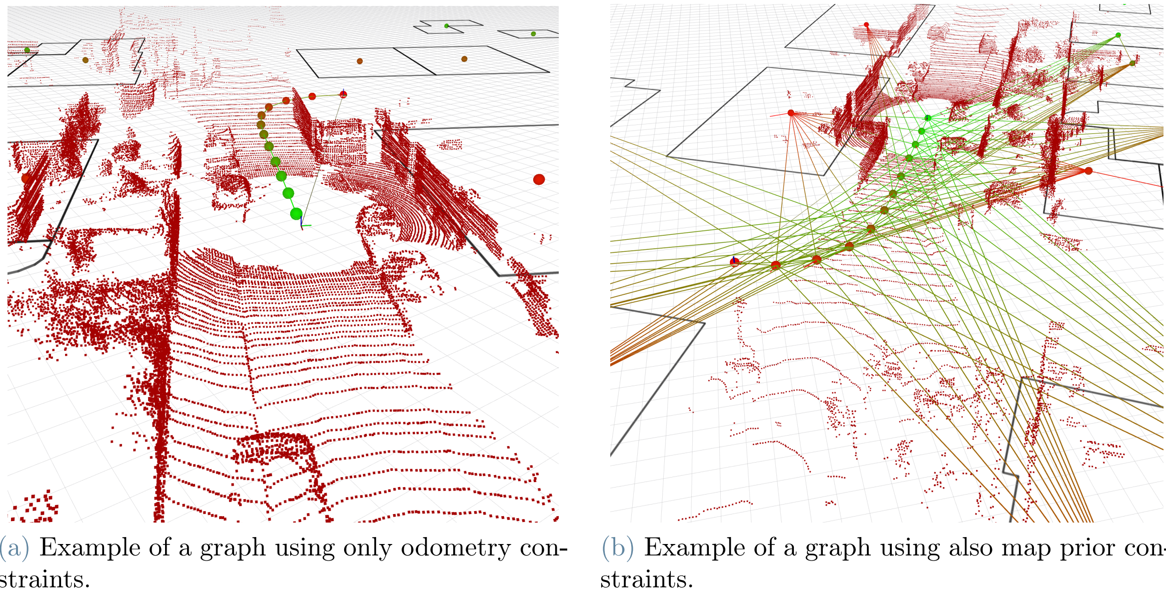
In our case, the odometry estimation is performed via LiDAR point cloud scan-to-scan matching, and the edges connecting two consecutive poses, contain only the corresponding estimated motion of the robot.

As a side note, from a continuous trajectory followed by the robot, only some poses are extracted to create the nodes of the graph. Standard graph-based approaches constantly add new nodes to the graph. As a result, memory and computational requirements grow over time, preventing long-term mapping applications. A continuously growing graph slows down graph optimization and makes it more costly to identify loop closures [12]. To solve this problem, not all the poses are retained, thus only special poses remain along with their corresponding data (such as LiDAR point clouds, GPS, IMU and others) also called keyframes, discarding the other poses and related data.

To adjust the estimation error, when the robot visits places that it has seen before during its trajectory, it adds a new type of edge to account for an existing loop closure. With loop closures, the robot can reduce the trajectory estimation error, which incrementally increases over time. An example of a graph using only odometry constraints is shown in Figure 2.1a.

2.2. ROS

In this section we introduce the Robotic Operating System (ROS) [17] on which our work is based. ROS is an open source set of software libraries and tools that helps to build robot applications. It gives the services of an operating system, such as hardware abstraction, low-level device control, message-passing between processes, and package management. It also provides tools and libraries for writing, managing, and running code across multiple hosts / devices. ROS is based on a peer-to-peer communication network to provide a common interface to all the processing units that usually compose a robotic system.



(a) Example of a graph using only odometry constraints.

(b) Example of a graph using also map prior constraints.

Figure 2.1: Graphs of a graph-based SLAM system, the red clouds are the LiDAR point clouds and the black polygons are the OSM maps. The green/red spheres and lines are the nodes and arcs of the graph respectively.

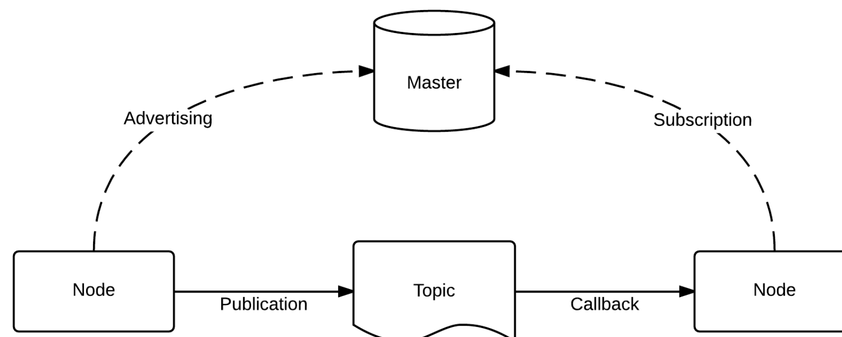


Figure 2.2: ROS common components

ROS comprises the following abstractions: nodes, master, messages, services, topics, nodelets and bags.

Nodes are the processing unit and the building blocks of a robotic system, and they are meant to execute its different parts. Usually, a robotic system is composed by multiple nodes. A ROS node is written using a dedicated library, such as roscpp or rospy (even if not mandatory).

The nodes can communicate using messages, which are defined structures that embed data such as strings, integers, arrays and other structures. Before sending and receiving

messages, each node has to publish or subscribe to a topic on which the communication happens (See Figure 2.2). The communication and orchestration of the whole system is managed by a special node called master. The master provides name registration and lookup features (such as topics, services and more) to all the nodes. To each topic it is associated a name and a type of message used for the communication. When a node subscribes to a topic, it binds a callback function that it is executed anytime a new message arrives.

Since the publish / subscribe model is unidirectional, i.e. there is not a request / response behaviour, services are provided to enable also a direct interaction between nodes. Differently for the topics, the services use two message types, one for the request and another for the response.

Another important concept are the nodelets. They provide the same functionalities of simple nodes but are more efficient thanks to a shared memory communication rather than network based. They are designed to run multiple algorithms in a single machine. Because of the shared memory, there is no cost of copying messages between processes, and for this reason, communication between nodelets is more efficient than the communication between conventional nodes.

Finally bags are a common way to represent dataset in a suitable way to be processed with ROS, they consist of recorded data, for example in a real world execution of a robotic system, that can be replayed anytime to test and help the programming of the nodes. A bag captures and stores all the messages of chosen topics, which can be replayed anytime.

2.3. System Architecture

The baseline system is an extension of HDL Graph SLAM [7] and their architecture shares common modules, represented with the following nodelets:

- prefiltering_nodelet
- scan_matching_nodelet
- floor_detection_nodelet
- hdl_graph_slam_nodelet

In Figure 2.3 there are highlighted the topics (arcs with red lables) used by the nodelets (boxes) to communicate. As we can see, the pipeline takes the LiDAR point cloud as input and generates the map as output. Internally, after prefiltering the point clouds by the prefiltering_nodelet, the filtered point clouds are read by the scan_matching_nodelet which

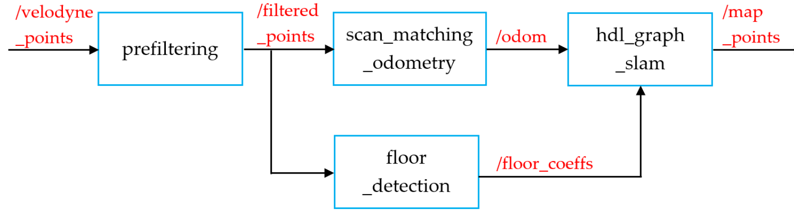


Figure 2.3: HDL Graph SLAM system architecture

estimates the odometry of the robot using a registration algorithm. HDL Graph SLAM uses also a floor_detection_nodelet, which computes the ground plane coefficients and are used by the hdl_graph_slam_nodelet as additional constraints on the pose graph. Since these constraints optimize the graph such that the floor planes (detected by RANSAC [15]) of the pose nodes become the same and it is designed to compensate the accumulated rotation error of the scan matching in large flat indoor environments [7], the floor_detection_nodelet is not used by the baseline system anymore. In fact the baseline application is meant to deal with outdoor and not flat environments, therefore its architecture consists of the three remaining nodelets.

2.4. HDL Graph SLAM

The hdl_graph_slam nodelet is the core of the pipeline. It subscribes to multiple topics other than odom and floor_coeffs, such as GPS and IMU. Each topic has a corresponding callback function, which is executed to process every received message. These messages are dispatched into FIFO queues that are flushed periodically, every 3 seconds, in order to construct the pose graph.

As mentioned in Section 2.1, along the trajectory of the robot, only some poses are chosen to add a node to the pose graph. In hdl_graph_slam, these poses are chosen only if the robot has traveled more than 2 meters than the last chosen pose, or it has rotated more than 115 degrees than the last pose.

The selected poses are stored into keyframes, which are special data structures that store the information acquired from each pose and the pose itself. Keyframes are instantiated by the odometry callback function (binded to the odom topic) and are stored in a queue.

Other queues are used to store additional sensor data, such as GPS and IMU. Using a timer callback, which is executed every 3 seconds, all the queues are flushed. In particular, the previously instantiated keyframes are inserted into the pose graph as nodes, along with

their corresponding edges. An edge between two consecutive keyframes is called odometry constraint. Sensor data, GPS and IMU, is used to add additional edges, called prior edges, which constrain a single node position and orientation respectively.

The timer callback, after flushing all the queues, checks every new keyframe if it can perform a loop closure with older nodes. If it finds a loop, it adds an additional edge between the two matched nodes. Finally, after the insertion of every node and edge, the pose graph is optimized and the robot pose corrected based on the newly added constraints.

Since the gps messages are expressed as latitude, longitude and altitude (global reference system), and the constraints are expressed within a local reference system, the gps messages should be converted into East-North-Up coordinates (ENU) before inserting the corresponding gps prior edges.

Each edge has an associated weight, called information matrix. The information matrix is a diagonal 6x6 matrix, with each component on the diagonal being associated to one of the six degrees of freedom in 3D, independently one from the other:

$$\begin{bmatrix} w_x & 0 & 0 & 0 & 0 & 0 \\ 0 & w_y & 0 & 0 & 0 & 0 \\ 0 & 0 & w_z & 0 & 0 & 0 \\ 0 & 0 & 0 & w_\alpha & 0 & 0 \\ 0 & 0 & 0 & 0 & w_\beta & 0 \\ 0 & 0 & 0 & 0 & 0 & w_\theta \end{bmatrix} \tag{2.2}$$

In particular, the first three elements of the diagonal (starting from the top) weigh the translation and the last three values weigh the orientation of the transformation matrix associated to the edge. Intuitively, if an edge has bigger information matrix than another edge, it would influence the optimization process more, conversely if its information matrix is smaller it would influence the optimization process less.

The information matrix takes into account the uncertainty of the Gaussian approximation of an edge. The information matrix Ω is the inverse of the covariance matrix Σ [19].

$$\Omega = \Sigma^{-1} \tag{2.3}$$

A generic element of the covariance matrix is σ_{ij}^2, that expresses the covariance between the $i-th$ and $j-th$ components. Given that the components of a measurement are

independent one from the other, the co-variances are all equal to zero outside the main diagonal. The elements on the diagonal, corresponding to i = j, are the variances $\sigma_{ii}^2 = \sigma_i^2$ of the i-th component, and σ_i corresponds to the standard deviation. Because of these considerations and the Formula 2.3, the components of an information matrix are in the form $1/\sigma_i$. So the information matrix can be expressed as:

$$\begin{bmatrix} 1/\sigma_x & 0 & 0 & 0 & 0 & 0 \\ 0 & 1/\sigma_y & 0 & 0 & 0 & 0 \\ 0 & 0 & 1/\sigma_z & 0 & 0 & 0 \\ 0 & 0 & 0 & 1/\sigma_\alpha & 0 & 0 \\ 0 & 0 & 0 & 0 & 1/\sigma_\beta & 0 \\ 0 & 0 & 0 & 0 & 0 & 1/\sigma_\theta \end{bmatrix} \tag{2.4}$$

If we select a low variance of a component, which means that the uncertainty of a measurement is believed to be low, the corresponding value on the information matrix is high. Vice versa, if the variance is high, the uncertainty of a measurement is believed to be high, and the corresponding value on the information matrix is low.

In hdl_graph_slam, in order to construct the information matrix, a standard deviation range is specified for the translation and orientation estimation uncertainty, min_σ_i and max_σ_i. Each component of the information matrix is computed, using the Formula 2.3, based on a σ_i value that falls within the specified range. This value is chosen according to a fitness score, which corresponds to the average distance of a source point cloud and a target point cloud, the point clouds are sensed in the two consecutive poses to which we want to add the constraint. The average distance is computed using a point-to-point distance metric. For each point in the source point cloud, it computes the distance from its nearest neighbour of the target point cloud, the fitness score will be the average of all points distances. The fitness score is scaled between 0 and 1 and it is used to choose a value between max_σ_i and min_σ_i to construct the final information matrix associated to the edge to be inserted.

After the last optimization of the graph, hdl_grph_slam provides a ROS service to save the map. The final map is generated by merging all the LiDAR point clouds associated to every keyframe.

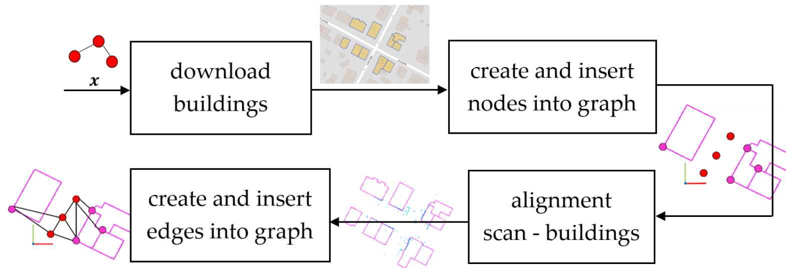


Figure 2.4: Pipeline of the baseline system

2.5. Baseline system

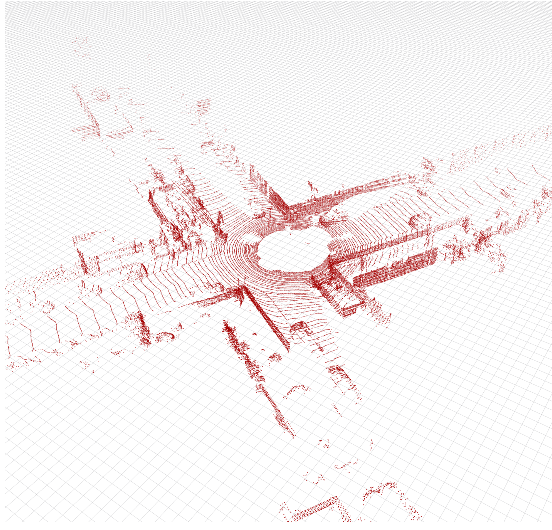
As anticipated, the baseline system is an extension of HDL Graph SLAM. It's goal is to embed map priors into the pose graph in order to better localize the robot and simultaneously fix the maps given by OSM.

The baseline system extends solely the hdl_graph_slam nodelet. In particular, after flushing the keyframe queue, it executes a new function, update_buildings_nodes, that performs all the process steps (As shown in Figure 2.4). In particular, after being called by the optimization timer callback (every 3 seconds) it iterates over all the keyframes that do not have buildings associated with them, performing the operations explained in the following subsections in the order they are presented.

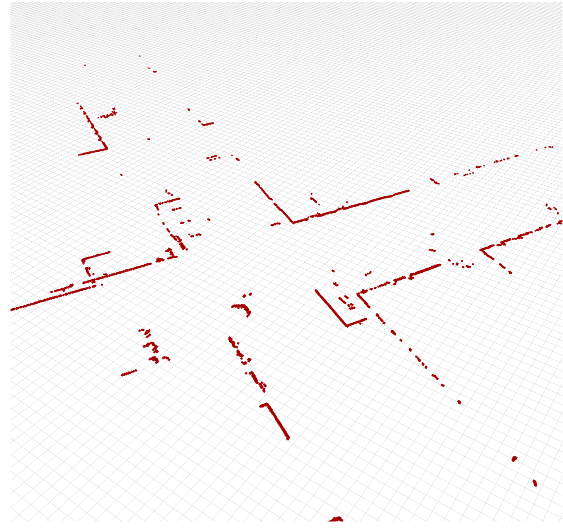
2.5.1. Downloading of the Buildings

In order to download the maps from OSM, update_buildings_nodes has to specify the following parameters: latitude, longitude, radius and host. The OSM host returns the buildings that are within the circular area of the specified radius and centered in the given coordinates. Since the gps is used only during the initialization to download the first map, in the subsequent poses it has to convert the estimated pose of the robot from ENU coordinates into latitude and longitude coordinates, in order to retrieve the subsequent maps. To perform the conversion between the two coordinates systems, it stores the first gps message, latitude and longitude, associated to the origin of the ENU coordinates system and uses it as reference to compute the conversion.

After the request, update_buildings_nodes synchronously waits the response. Once it receives the requested maps, it converts them from XML into its internal data structures, i.e. each building has its own instance and the corresponding fields. The downloaded maps are composed solely by the shapes of the buildings, which are encoded by means of their vertices. Each vertex has it's own coordinates, latitude and longitude, which are



(a) Example of a 3D LiDAR point cloud before the conversion.



(b) Example of a 2D LiDAR point cloud after the conversion.

Figure 2.5: Visualization of the conversion of the point clouds from 3D to 2D.

converted into ENU coordinates before being used. In order to perform scan matching, their shapes are interpolated using points and converted into point clouds.

2.5.2. LiDAR Point Clouds 2D Conversion

Since the LiDAR point clouds are in 3D, they should be converted into 2D in order to be correctly used. Before the flattening process, the points below the LiDAR view point are removed, in order to remove possible outliers, such as cars, bicycles, fences and more. Subsequently, the LiDAR point clouds are flattened by setting to zero all the z components of its points, as shown in Figure 2.5.

2.5.3. Map Priors Extraction

As shown in Figure 2.4, after the download of the buildings and the filtering of the LiDAR point cloud associated to the keyframe that is currently being processed, update_buildings_nodes creates a node for each building found in its range. Then, to extract the map priors, it performs the following matches:

- the **global matching**, which matches the LiDAR point cloud with all the surrounding buildings
- the **local matching**, which matches the point cloud associated to a single building with the LiDAR point cloud, to locally align the building with the LiDAR point

cloud

The global matching is performed using the registration algorithm NDT_OMP [8]. The returned transformation is used as initial guess for another scan matching algorithm, FAST_VGICP [11], which it is used to perform the local matching. This returns a transformation for each building around the keyframe that is currently being processed.

2.5.4. Inserting Graph Edges and Optimization

Subsequently, update_buildings_nodes inserts into the pose graph the edges connecting the current keyframe to all the surrounding buildings, using the transformation found with the local matching and the associated information matrix. If the registration algorithm does not converge to align a building, e.g. if there are not enough LiDAR points to perform local matching, then the corresponding edge will not be inserted into the graph.

To compute the information matrix, update_buildings_nodes uses the same algorithm of HDL Graph SLAM. Since the baseline system is in 2D, it has adapted the algorithm to return a 3x3 information matrix instead of the 6x6 matrix used by HDL Graph SLAM.

After the insertion of the edges, the pose graph is optimized. All the keyframes and buildings nodes are moved and rotated in order to satisfy the newly inserted odometry and map constraints.

3 | Improvements

In Chapter 2, we have seen the workflow of the baseline Graph SLAM system. In this chapter we give further details about some of the many baseline's issues, such as overlapping of map elements and efficiency. With our work, we contributed to overcome these problems, by developing novel algorithms, that will also be detailed in this chapter.

Section 4.1 describes some methods used to improve the efficiency of map retrieval from OpenStreetMap. Follows Section 4.2 and Section 4.3, where we discuss about the procedures of line-features extraction and subsequent line-based scan matching algorithm along with the specific fitness function, that improved the estimation of the transformations expressed in the graph edges. Finally the Section 4.4 describes the HypherGraph feature of g2o [13] used to improve the graph optimization result, while the last Section 4.5 explains the adopted solution to avoid having overlapped buildings in the generated 2D maps.

3.1. OpenStreetMap XML Buffering

As we have seen in Section 2, every LiDAR scan gather by the robot contains the point clouds associated to one or more buildings nearby the robot. As the input point clouds are retained in keyframes, by downloading the map from OpenStreetMap centered in the robot, we are able to align the keyframe point cloud to the set of points extracted from the map. This way we can simultaneously estimate and optimize both the robot and building poses.

The downloaded maps are encoded into an XML string, so we have to parse the XML into a data structure and, with a given OpenStreetMap ID, we can uniquely reference each element of the map (i.e. buildings) as needed. Due to the high latency of the OSM servers (on average 2000 milliseconds from overpass-api.de), the downloading process must be optimized in order to not slow down the execution of the overall SLAM pipeline.

As it is commonly done to handle streams of data, we chose to use a buffer. In particular, while processing a single keyframe and downloading the local map, the requested map radius is larger than the one required for the current keyframe. New keyframes, following

Figure 3.1: Visualization of the buffering algorithm

the latter, would still be able to reuse the buffered map, enabling us to request new maps ahead of time and prevent any possible latency or packet loss from the OSM servers.

Practically, we use a radius of 120 meters to fill the buffer as an XML string, storing also the ENU coordinates of the buffer center (Figure 3.1). For every keyframe, the robot will request the nearby buildings within a radius of 35 meters, therefore it will parse the useful portion of the buffered XML and get the corresponding piece of map.

Whenever the estimated pose of the robot goes outside a specific area (circle of 60 meters radius) from the buffer center, the system will do an asynchronous request to OSM and update the buffer along with its new center. This way, we have still a margin of 25 meters of the buffer to use while waiting the OSM server response.

This solution has considerably improved the processing time of the buildings, in fact the first time we wait the full latency of the OSM server (about 2000 ms) but the following times we will simply process the buffer with an average time of 5 ms. Quite different from the baseline system which, every time it needs new buildings, waits for the response of the OSM servers putting the entire system on hold and making it unstable. Moreover, if the server becomes temporarily unresponsive, any package loss can be tolerated, increasing the overall robustness of the algorithm since it still has a margin of 25 meters into the buffer to be used as explained above.

3.2. Line Segments Extraction with RANSAC

As stated in the previous chapters, the input data consists of point clouds collected using a 3D LiDAR. These clouds accurately represent the surrounding environment of the robot, containing buildings, vehicles, houses and so on. As we are interested only in buildings, since the map priors are extracted by matching their geometrical shapes with the LiDAR scans, the rest of the cloud can be considered noise to remove.

To achieve this goal it is useful to filter the point clouds and remove the outliers. This operations are executed in this order:

- Removal of the points that are underneath the LiDAR horizon to remove possible outliers such as vehicle, bushes, pedestrians, fences and even the road itself
- For every point, remaining from the previous step, we extract a fixed number of its nearest neighbors and interpolate a plane on them. the point is selected only if the normal to the interpolated plane is almost perpendicular (within a fixed threshold) to the ground plane normal. In this way, all the remaining points will belong to vertical local planes, such as walls.
- Point cloud flattening by removing the z component of all points

The order in which we apply these filters is important. Removing the points before computing the local planes is more efficient, moreover the flattening of the point cloud has to be performed as last step since the other steps need the z component. A further improvement is to perform feature extraction from the previously filtered LiDAR data that in our case corresponds to the line segments that outline the buildings. To perform this task, we chose to use Random sample consensus (RANSAC) [15]. The RANSAC algorithm is a learning technique to estimate parameters of a model by random sampling on observed data. The models we want to estimate are lines and the observed data is the LiDAR point cloud.

The RANSAC algorithm for line extraction is essentially composed of two steps that are iteratively repeated:

- In the first step, two sample points are randomly selected from the input point cloud. A fitting line and the corresponding model parameters (line direction and a line point) are computed using only the two samples.
- In the second step, the algorithm checks how many points of the entire input point cloud are consistent with the fitting line obtained from the first step. A point will be considered as an outlier if it does not fit the fitting line within some error threshold, otherwise it will be considered an inlier.

The set of inliers obtained for the fitting model is called the consensus set. The RANSAC algorithm will iteratively repeat the above two steps until the obtained consensus set has enough inliers or it reaches a maximum number of iterations.

The data obtained from RANSAC should be further elaborated to extract only a segment of the line that represent a wall following the pseudo code Algorithm 3.1.

Algorithm 3.1 Line segments extraction with RANSAC

1: $minClusterSize, minSegmentLength$
2: $pointCloud$ = the filtered data produced by LiDAR
3: $extractedSegments \leftarrow [\,]$
4: **while** $pointCloud.size > minClusterSize$ **do**
5: $\quad a, vect, inliers \leftarrow RANSAC(pointCloud)$
6: $\quad cluster \leftarrow clustering(inliers)$
7: $\quad$ **if** $cluster.size > minClusterSize$ **then**
8: $\qquad segment \leftarrow extractSegment(cluster, a, vect)$
9: $\qquad$ **if** $segment.length > minSegmentLength$ **then**
10: $\qquad\quad extractedSegments \leftarrow segment$
11: $\qquad$ **end if**
12: $\quad$ **end if**
13: $\quad pointCloud \leftarrow pointCloud - cluster$
14: **end while**
15: **return** $extractedSegments$

Since the inliers can belong to more than one building, in the specific case of buildings with lined walls, it has to cluster them based on their euclidean distance and select only the cluster with the largest number of points. If the selected cluster has less points than

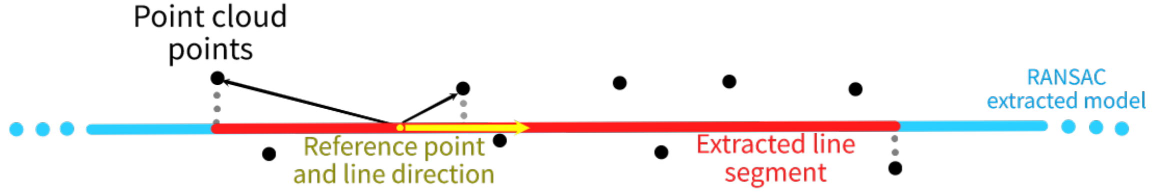


Figure 3.2: Visualization of the line extraction algorithm.

a fixed threshold, the cluster is removed from the point cloud and the process goes to the next iteration, otherwise it continues to the next step.

The points in the cluster are projected on the line by performing the scalar product with the normalized line direction vector (see Figure 3.2 and Appendix A.1), thus it is possible to identify two special points that have the lowest and largest signed vector projection norm. These two points are the endpoints of the extracted line segment.

As a last step of the line detection and extraction procedure, the points of the selected cluster are removed from the point cloud and the extraction process is repeated until the point cloud has less points than the minimum cluster size.

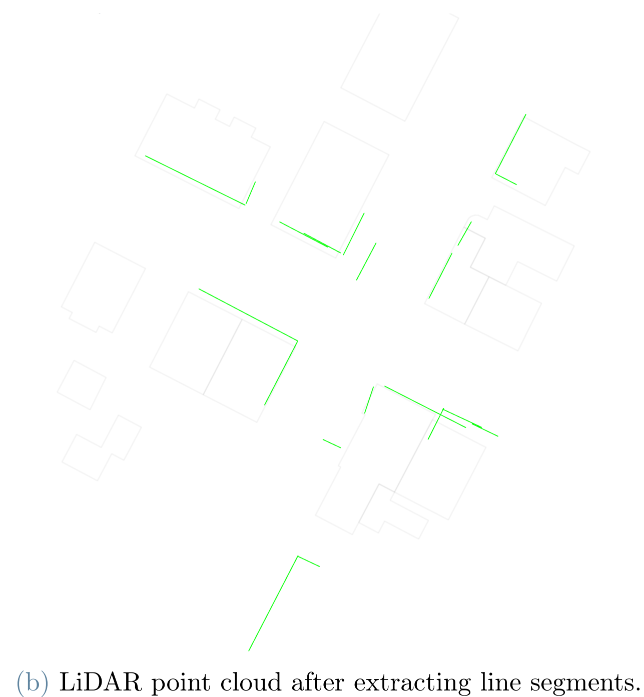
As a final result we obtain much cleaner data about the environment surrounding the robot, which considerably facilitates the subsequent scan matching operation useful to estimate the robot and buildins poses. (Figure 3.3).

3.3. Line Segment Based Scan Matching

The alignment procedure is the most important part of the pipeline, all the map priors embedded into the pose graph are retrieved using a scan matching algorithm.

In the baseline system, the Iterative Closest Point (ICP) [16] family of algorithms was used as registration method to extract the map priors. One of the major drawbacks of these algorithms are the implicit assumption of full overlap of the shapes being matched and the theoretical requirement that the points are taken from a known geometric surface, rather than measured [1]. The first assumption is not met because the buildings point clouds are essentially a set of polygons, more often rectangles, and the LiDAR sees only a portion of them, such as the front part, making the two point clouds to be aligned partially overlapping. Moreover, since our LiDAR point cloud has many outliers such as trees, cars, bicycles, and others, the second assumption is not met either.

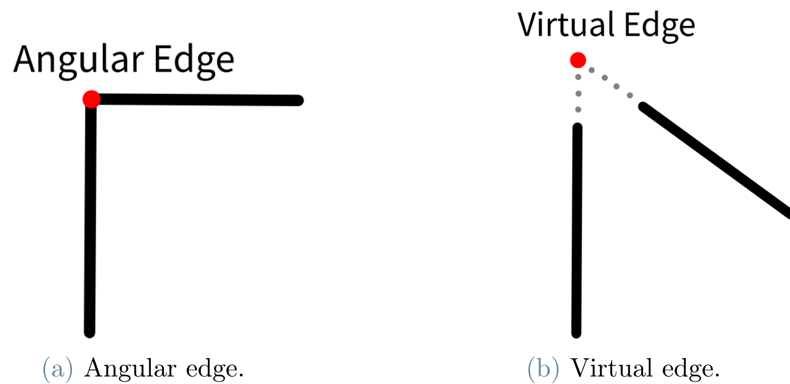
(a) LiDAR point cloud before extracting line segments.



(b) LiDAR point cloud after extracting line segments.

Figure 3.3: Results of line segments extraction

Beside the improved performance to localize the robot, using ICP did not perform as well to fix all buildings' poses. Moreover, all the ICP algorithms suffer of local minima and they cannot give us a reliable transformation. For this reasons, we decided to implement a new algorithm tailoring our specific problem by aligning line segments and edges. Since all the line segments of the buildings can be extracted from OpenStreetMap, we have to



(a) Angular edge.

(b) Virtual edge.

Figure 3.4: The two types of edges

use our line segment extraction algorithm only on the LiDAR point clouds, as explained in Section 3.2. After having extracted all the line segments, we now have enough information to perform line matching, with the goal of finding the rigid 2D transformation that best aligns the two sets of line segments.

3.3.1. Edges Extraction and Matching

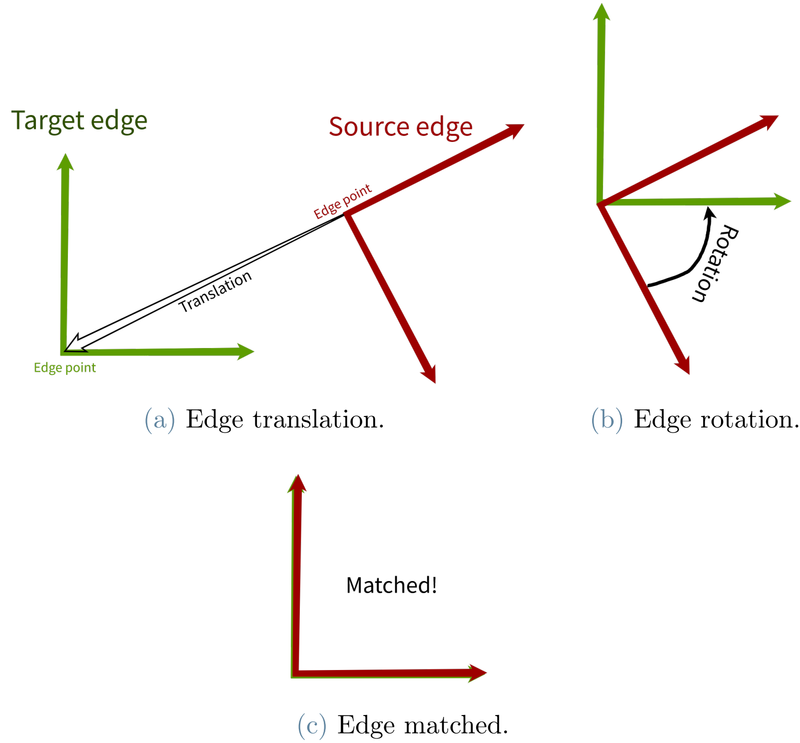
To find the 2D transform that best aligns two sets of line segments, we rely on the edges within these sets. We can define two different types of edges [6] (Figure 3.4):

- Angular edges, which is the intersection of two line segments having a common endpoint
- Virtual edges, which is the intersection of two line segments not having a common endpoint

Note that using an edge within each cloud is sufficient to find a unique transform that fixes all the three planar degrees of freedom (Figure 3.5). When aligning two edges, we find the SE(2) transformation that gives us the translation to overlap the two edge points (the intersection point of two line segments composing the edge), and a rotation that will align the edges' line segments orientations.

3.3.2. Choosing the Best Transformation using Edges

Once extracted the relevant features we can perform the alignment process. The algorithm consists in matching every possible pair of edges between the source point cloud (the one on which the transformation should be applied) and the target point cloud. Among



(a) Edge translation.

(b) Edge rotation.

(c) Edge matched.

Figure 3.5: Visualization of the steps to perform edge matching

all the possible transformations (search space) we choose the one that maximizes our custom fitness score, as described in Section 3.3.4. Even if the alignment does not suffer of local minima, as it tests all possible transformations, we choose to restrict the search space constraining the translation and orientation of the desired transformation, to reduce processing time and avoid possible ambiguities. The maximum allowed translation and orientation can be finely tuned depending on the scenarios in which the algorithm is used. For example, when starting a new sequence of test data, the initial robot pose is chosen arbitrarily, since we do not have this information. For this reason the orientation of the desired transformation is not constrained, in order to estimate every possible initial orientation. Once the robot has found its initial orientation it can constrain the maximum orientation, depending on the current underlying odometry estimation error associated to the SLAM system front-end.

3.3.3. Choosing the Best Transformation using Lines

Sometimes, the number extracted of edges are not enough to find a decent transformation. Moreover, if we align a single edge, it may happen that the other line segments are still displaced (Figure 3.6).

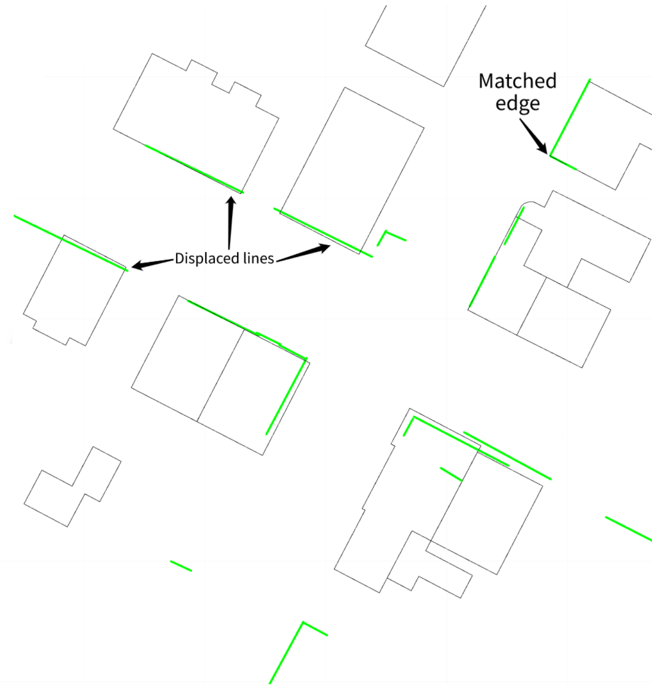


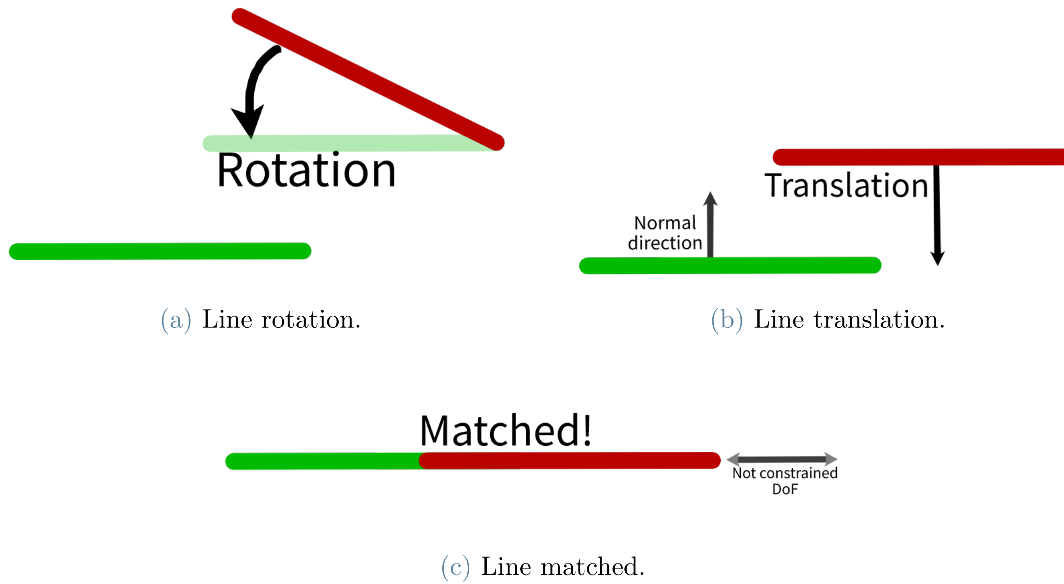
Figure 3.6: Displaced lines when performing line segments alignment using only edges.

For this reason, to increase the accuracy, we can start from the transformation found in Section 3.3.2 and perform a second alignment process based only on line segments (Figure 3.7). This time, given a pair of line segments, we can constrain only two degrees of freedom out of three. In order to fix also the third degree of freedom, we match two line segments by first rotating the source line segment (the one to be aligned) in order to make it parallel to the target line segment. Between the two possible angles to align two line segments, one clockwise and another counter clockwise, we choose the smallest one. Next, we project the source line segment on the target line segment using a translation along the direction of the normal of the target line segment.

Having found a method to align two line segments we can now compute the possible transformations between every pair of line segments between source and target (composing our search space). As in the previous case, we can reduce the search space by aligning the source line segment only to the k nearest neighbours of the target line segments (where k is an arbitrary parameter). The final transformation will be the one with the highest fitness score (see Section 3.3.4) and the final result of the alignment.

3.3.4. Specialized Fitness Function

To select the right transformation from the search space, we have to quantify a fitness score for each of them. Implementing a scan matching algorithm from scratch enabled



(a) Line rotation.

(b) Line translation.

(c) Line matched.

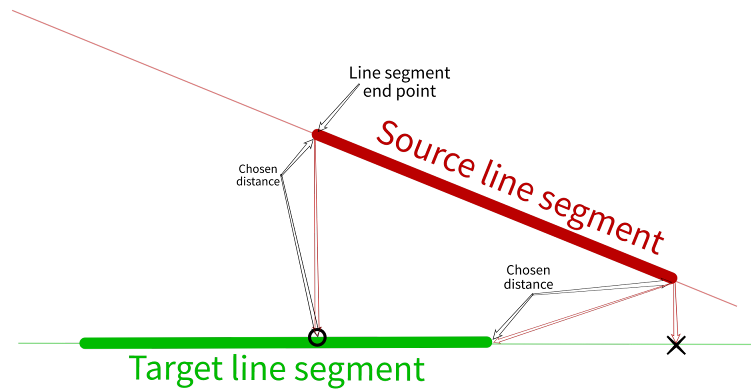
Figure 3.7: Visualization of the steps to do line matching

us to build a fitness function that best suits our needs. Usually, when performing scan matching between point clouds, the point-to-point distance metric is used to quantify the fitness score. In our case, since we are aligning lines, we adopted a line-to-line distance metric. There are multiple ways to compute this value and in our algorithm we use the following methods:

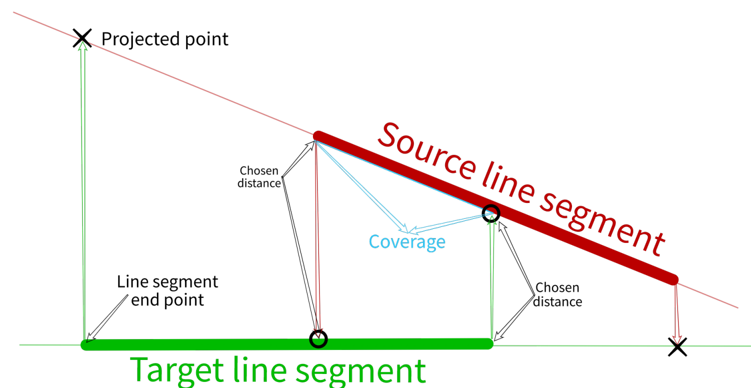
- Line segments distance without coverage
- Line segments distance with coverage

To compute the distance of a source line segment to a target line segment without coverage, we use only the two endpoints of the source line segment. For each endpoint, we compute the projected point onto the line where the target line segment is laid. If the projection point is inside the target line segment, we will use the distance from its projection, otherwise we compute the distance from the nearest endpoint of the target line. The final distance without coverage of the source line segment to the target line segment will be the average of the two source endpoints distances 3.8a.

To compute the distance of a source line segment to a target line segment with coverage, we have to take into account also how much two lines match each other. This method will return two metrics: the distance and the coverage (how much the source line segment would cover the target line segment if matched as described in Section 3.3.3). This time, we compute also the projections of the target line segment endpoints, along its normal direction. As shown in Figure 3.8b, between the four endpoints (source and target), we



(a) Line to line distance without coverage.

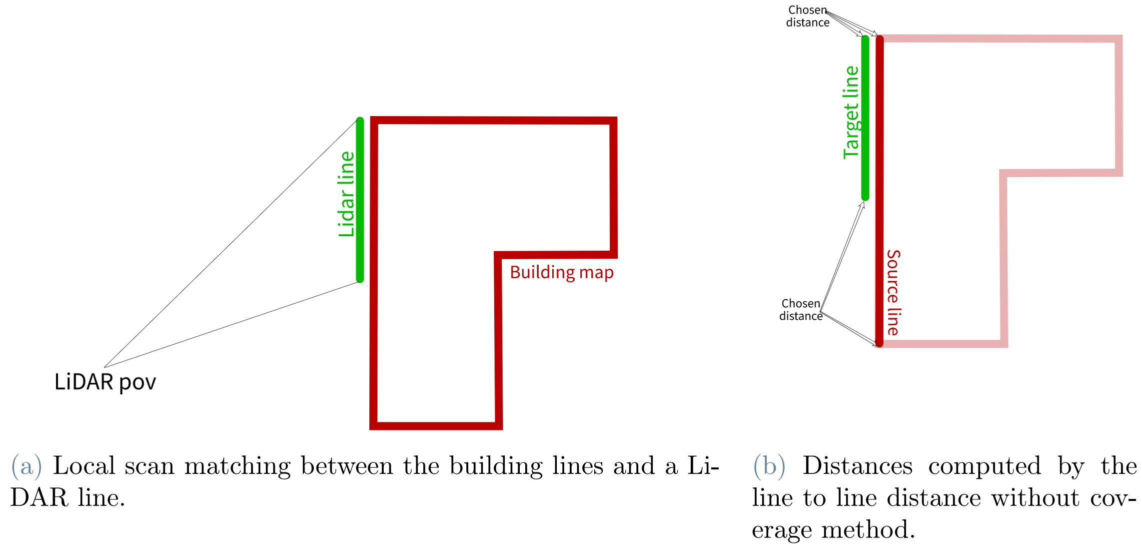


(b) Line to line distance with coverage.

Figure 3.8: Visualization of the line to line distances.

choose the two that have their projection within the other line segment. The final distance of the source line segment to the target line segment will be the average of the distances of the chosen points, while the coverage will be the used portion of the source line, expressed in meters.

The usage of these two different metrics depends on the scenarios in which they are applied. The main difference between the two methods is that the line-to-line distance without coverage takes into account also the lengths of the lines being matched (see Figure 3.9), while the other does not. For example, it gives a grater distance than the second method if the source line segment is bigger than the target line (as shown in Figure 3.9b). This is not a desired behaviour when, trying to fix a building pose (**local matching**), we use the line segments of a building as source and the ones extracted from the LiDAR as target. In the average case, the target line segments will be always smaller than the source ones (since the LiDAR usually sees a building wall partially). On the other hand, while trying to fix the robot pose (**global matching**), when using the line segments extracted



(a) Local scan matching between the building lines and a LiDAR line.

(b) Distances computed by the line to line distance without coverage method.

Figure 3.9: Visualization of the line to line distance without coverage problem.

from the LiDAR as source and the ones of the nearby buildings as target, we can assume that the source line segments are always shorter than the target line segments. If a LiDAR line segment is longer than a building line segment, then it will probably not match with that building. In that case, we cannot use the line to line distance with coverage in the global matching, because it won't take into account the real distances of the LiDAR line segments.

Having explained the distance metrics we can now define our fitness function. To compute a fitness score of a transformation found with the methods described in Sections 3.3.2 and 3.3.3, we first transform all the source line segments using the given transformation. Subsequently, iterating over the transformed line segments, we find, for each of them, the corresponding nearest neighbor of the target line segments. The metric used to find the nearest neighbor is one of the two described above, depending on the type of matching being performed (local or global). The fitness score is then computed using three quantities:

- The average distance of each source line from the nearest neighbour of the target line segments
- The overall coverage of the source line segments, computed by adding the coverage of every source line segment
- The translation norm of the given transformation

In our implementation, the average distance is computed with the line-to-line distance

without coverage for the global matching, while for the local matching we use the line-to-line distance with coverage. Intuitively, the average distance will contribute negatively to the fitness score, because it will choose the transformation having as little average distance as possible. The coverage, computed for every transformation, will contribute to the fitness score to choose the transformation of the source line segments that overlaps them on the target line segments the most. The last property will contribute negatively to the fitness score, as it will give more importance to transformations having smaller translation norm. These contributions, after being scaled to have the same order of magnitude, are weighted and added/subtracted to compose our specialized fitness score.

$$AvgDistanceW = A \cdot \frac{min(MaxAvgDistance, AvgDistance)}{MaxAvgDistance} \cdot 100 \tag{3.1}$$

$$CoverageW = B \cdot \frac{Coverage}{TotalLineSegmentsLength} \cdot 100 \tag{3.2}$$

$$TranslationW = C \cdot \frac{min(MaxTranslation, Translation)}{MaxTranslation} \cdot 100 \tag{3.3}$$

$$FitnessScore = -AvgDistanceW + CoverageW - TranslationW \tag{3.4}$$

The two constants *MaxAvgDistance* and *MaxTranslation* can be chosen arbitrarily depending on the use case.

This fitness score is the result of a great number of attempts, experimentation and considerations to try to solve all the bad alignments we have encountered. After fine tuning all the parameters, our scan matching algorithm is able to outperform the baseline to globally localize the robot and especially to locally fix the buildings poses as described in the results (Chapter 4).

3.3.5. Computed Information Matrix

As a last note, since the information extracted using edges can be considered more reliable, the information matrix (descried in Section 2.4), associated to the graph edges connecting robot nodes with buildings nodes, is chosen differently depending on the utilized feature to extract the constraint. As result the graph optimization will weigh more the edge based constraints than the line segments-based constraints, increasing the accuracy.

3.4. Graph Partitioning using g2o levels

As seen in Chapter 2, the whole system relies on the construction of a pose graph in the 2D space. Each node (also called vertex) represents a pose, embedding the position and ori-

entation of an object (robot or building), while each edge represents a constraint between the nodes on their relative positioning. An edge can constrain one (prior edge) or more nodes simultaneously (hyper-graph edge). This constraint is encoded in homogeneous coordinates using transformation matrices of the special Euclidean group SE(2), instead of the common SE(3) used in 3D spaces. The transformation matrices are expressed as 3x3 matrices:

$$\left\{ \begin{bmatrix} R & t \\ 0 & 1 \end{bmatrix} \mid R \in SO(2), t \in \mathbb{R}^2 \right\} \tag{3.5}$$

Where t is the 2x1 translation vector and R is the 2x2 rotation matrix of the transformation.

Along with the relative transformation, each edge is weighted by an information matrix, explained in the Section 2.4, which will be used during the optimization process of the graph. Intuitively, if an edge has a bigger information matrix than another edge, it would influence the optimization process more, conversely if its information matrix is smaller it would influence the optimization process less. The information matrix is a diagonal 3x3 matrix, and each component of its diagonal weights the three degrees of freedom, independently:

$$\begin{bmatrix} w_x & 0 & 0 \\ 0 & w_y & 0 \\ 0 & 0 & w_\theta \end{bmatrix} \tag{3.6}$$

In practice, w_x is equal to w_y and w_θ has a different value, with the first two being associated to the translational components, and the third weighs the rotation of the transformation matrix associated to the edge.

One of the drawbacks of the baseline system, due to the embedding of the buildings in the pose graph, is the worsening of the relative pose error (RPE) of the robot poses, which is one of the localization performance metrics that describes the accuracy on the relative transformation between two poses, as described in the results (Chapter 4). The higher the value, the more irregular is the trajectory with the nodes arranged in a zig zag pattern.

Each robot pose has only two edges related to odometry constraints, in particular with the previous pose and the next one. These edges are the odometry information retrieved by the LiDAR scan matching, which show little estimation error on short travelled distances. Since each robot node is connected with multiple edges with the near buildings nodes,

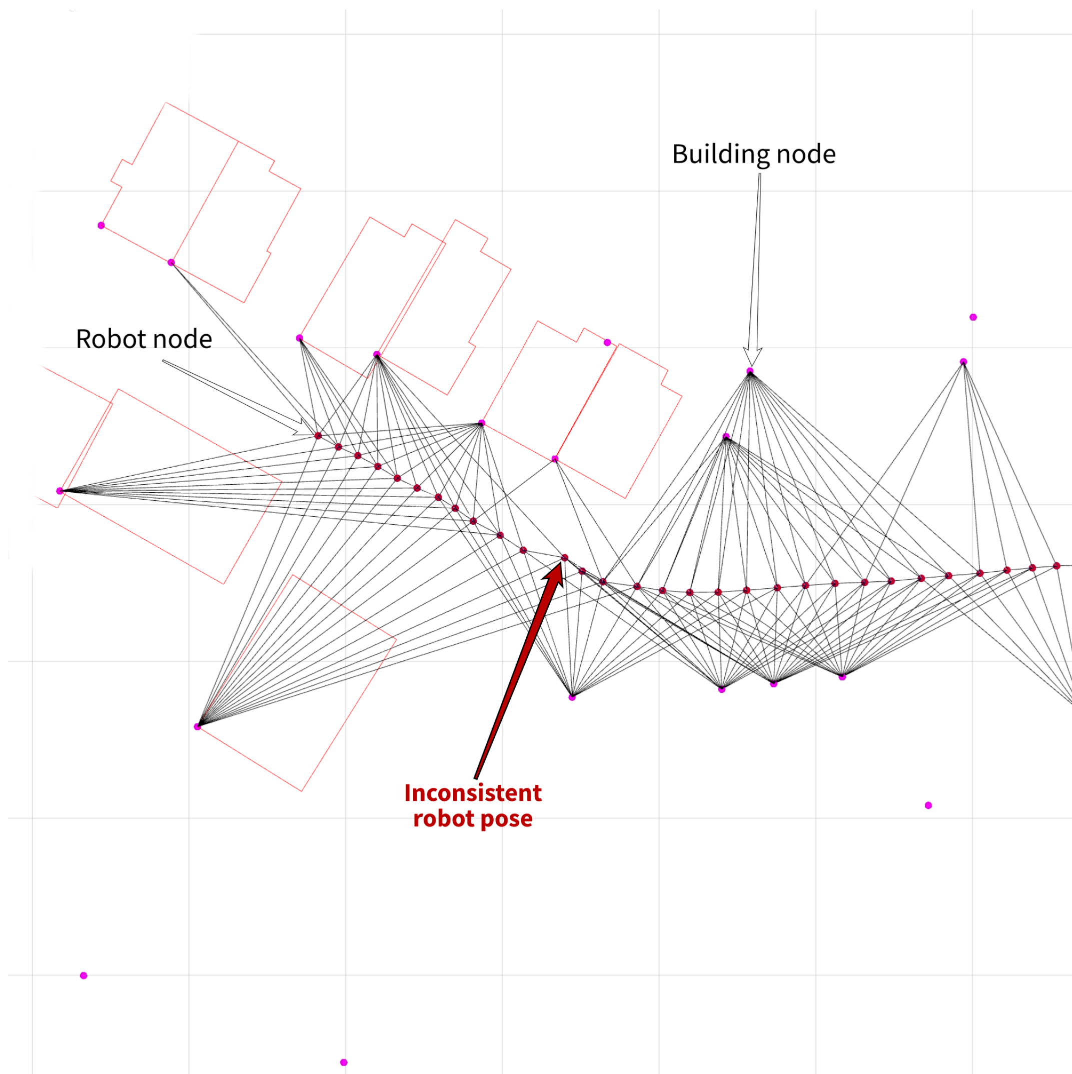


Figure 3.10: How the edges connecting the nearby buildings predominate the odometry constraint in the baseline system.

the odometry information gets predominated by the building edges. Considering that the information matrices used in the baseline have the same order of magnitude for all the edges, any bad local alignment with the buildings would erroneously move the robot node abruptly, making the poses of the robot not relatively consistent with the odometry information gathered by the LiDAR scan matching (as shown in Figure 3.10).

Another problem is the necessity to fix the first building node encountered, arising from the fact that every node can be moved freely by the graph optimization algorithm. In fact, during the initial pose estimation of the robot (which is conventionally chosen facing east), a constraint between a robot node and a building node does not tell which one of the two nodes should be moved. In the baseline system this issue was solved by fixing the

first building node encountered, in this way the graph optimization algorithm will move only the robot node, successfully estimating the initial robot pose.

On the other hand, the pose of the fixed building will not be corrected as the others. In addition, the trajectory estimated by the LiDAR scan matching, especially in small travelled distances, are extremely accurate. For this reason, during the estimation of the initial pose, only the orientation should be corrected, while the translation should be already estimated accurately by the LiDAR scan matching.

As a recap, the issues we aim to solve are:

- Inconsistency of consecutive robot poses
- Not accurate estimation of the initial pose of the robot

To deal with these issues we decided to partition the pose graph, to independently optimize the robot poses and the building poses. To achieve this, we used the feature of g2o to assign an optimization level to every edge. The optimization level is an integer $n \geq 0$ and specifies the edges used in the optimization. In our case we partitioned the pose graph in two parts:

- The edges between two robot nodes have an optimization level equal to 0 (localization partition)
- The edges between a robot node and a building node have an optimization level equal to 1 (mapping partition)

If we specify, an optimization level equal to n to be optimized, all the edges having an optimization level $\leq n$ will be used to optimize the pose graph. Using this feature of g2o, we can divide the optimization process into two phases.

In the first phase, we optimize the localization part, where all the edges between robot poses will be used during the optimization. In this way only the robot nodes will be moved, while the building nodes will remain in their initial pose as if they were disconnected from the graph.

In the second phase, we fix all the robot nodes and then optimize all edges by specifying an optimization level equal to 1. In this case, the edges of level 0, even if used by the optimization algorithm, will not move the robot nodes since we have fixed them before performing the optimization. Therefore, only the mapping partition will be optimized, moving the building nodes to better satisfy their constraints.

With this method, the building nodes will be corrected according to their constraints,

without interfering with the localization. On the other hand, the robot nodes would not be corrected as it was done before. For this reason, with our line segments-based scan matching algorithm (see Section 3.3), we perform a global matching with all the nearby buildings. The prior information extracted from the global matching is embedded into the graph as a prior edge (an edge constraining a single node). This solution adds only a prior edge for each robot node. In addition, the prior edge information matrix can be scaled to reduce its importance with respect to the odometry constraints. If the global matching has a low confidence, based on its fitness score, the prior edge will not be added at all.

To adjust the initial orientation of the robot, we don't have to fix the node of the first building encountered anymore. Instead, the orientation of the first robot node estimate is set, based on the transformation matrix found with the global matching, and fixed in the origin. As described in Chapter 2, the origin of the map coincides with the first gps message. This message is used as reference to convert all the ENU coordinates of the robot reference system into global coordinates, latitude and longitude. For this reason, it is legit fixing the first robot node in the origin, since the robot trajectory starts exactly from it.

Partitioning the graph improved dramatically the RPE and made the robot nodes follow a consistent trajectory without sudden distortions. In addition, it is possible to set the importance ratio between the odometry edges and the map prior edges that will soften up the robot pose corrections during the graph optimization.

3.5. Overlapped Buildings Problem Resolution

Due to the lack of geometrical constraints between neighboring buildings, the baseline system has the notable issue of overlapping some buildings (as shown in Figure 3.11). When aligning sparse point clouds, it may happen that the alignment does not perform as expected, and the constraint to fix a building pose could wrongly change its position instead.

To solve this issue, we managed to add a new type of edge that constrains a building node with another one (geometrical constraint). Specifically, after the second phase of the optimization (see Section 3.4), we remove all the geometrical constraints between the building nodes, and perform the following Algorithm 3.2:

Firstly, we check if all buildings overlap with other structures. For every pair of overlapped buildings, by using our scan matching algorithm, we find the minimal transformation that

Figure 3.11: Example of the generated map by the baseline system, the moved buildings are colored in red, the OSM buildings are black with a less thicker line and the LiDAR map in transparent black.

removes the overlapping by matching their geometrical features (line or edge) as seen in Section 3.3. Finally, we add an edge between the two buildings and optimize the g2o graph. The information matrix associated to the edge has a great value, to overtake the other constraints (hard constraint).

To valuate if two given buildings are overlapped, we check if any of their line segments are intersected. Since the buildings are closed shapes, this will be sufficient to state if they are overlapped or not.

To find the minimal transformation that removes the overlapping, we have implemented a custom method of our scan matching algorithm. In particular, of the all possible transformation found by matching edges and lines, the algorithm chooses the one having the

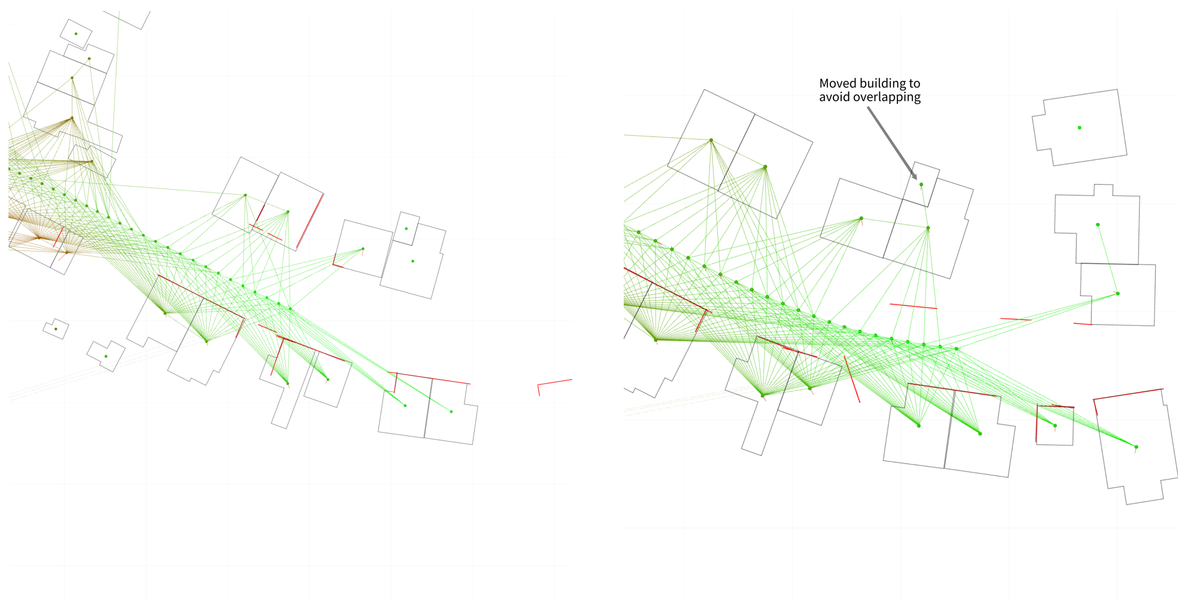
Algorithm 3.2 Algorithm to remove ovelapping from the buildings

1: $maxNumberOfIterations$
2: $infMatrix$
3: **while** $maxNumberOfIterations > 0$ **do**
4: $overlappedBuildings \leftarrow$ getOvelappedBuildings()
5: **if** $overlappedBuildings.size() = 0$ **then**
6: **break**
7: **end if**
8: **for all** $buildingA, buildingB$ **in** overlappedBuildings **do**
9: $transform \leftarrow$ lineBasedScanmatching.align(buildingA, buildingB)
10: $relPose \leftarrow (transform * buildingA.pose).inverse() * buildingB.pose$
11: addEdge(buildingA.node, buildingB.node, relPose, infMatrix)
12: **end for**
13: optimizeGraph()
14: $maxNumberOfIterations \leftarrow maxNumberOfIterations - 1$
15: **end while**

minimal translation norm which also removes the overlapping between the two buildings. In other words, the search space of the transformation matrices is reduced to those that remove the overlapping and, from this set, the scan matching algorithm chooses the one having minimal translation norm. It is worth mentioning that the desired transformation angle is constrained to be less than 60 degrees, as this will ensure that the building will not be rotated heavily.

Considering that the resolution of the building overlap can create a domino effect that moves many neighboring buildings to positions very far from the starting ones, we decided to add a prior edge to all the buildings, to encode the prior information about their pose given by OSM. So, if a building is scattered from its initial pose and it has not any constraint, it will be optimized to its original position.

After this last improvement, our system does not show overlapped buildings anymore. It is still possible that, in some rare case, the algorithm cannot find a configuration that removes the overlapping from all the buildings, but in the average case it successfully removes the overlapping and match the neighbour buildings. In addition, by removing the overlapping, it may happen that some buildings are re-positioned even if not directly seen by the LiDAR as shown in Figure 3.12 and this is important because keeps relative positions between buildings registered on OSM.



(a) Example of buildings before being overlapped.

(b) Example of buildings after being overlapped, and correctly re-positioned.

Figure 3.12: Visualization of one building being moved to avoid overlapping, even if not seen by the LiDAR directly.

4 | Results

In this chapter, we present the test data and evaluation metrics used as comparison with the baseline system. To evaluate the localization performance, in Subsections 4.1.2 and 4.1.3, we describe two commonly used metrics in SLAM benchmarks: Absolute Trajectory Error (ATE) and Relative Pose Error (RPE).

The ATE measures the error of the robot position with respect to the ground truth, while the RPE measures the error of the transformation between two consecutive robot poses with respect to the given ground truth relative transformation.

Conversely to the robot poses, the buildings' poses cannot be evaluated numerically, since there is not a ground truth for them. Instead, as previously done, we choose to visually evaluate the map corrections and their consistency with the map generated by the LiDAR point clouds, as explained in Section 4.2.

4.1. Test Data and Evaluation Metrics

In the following subsections, to better understand the results, we describe the test data and how we computed the evaluation metrics.

4.1.1. KITTI Odometry Benchmark

The KITTI dataset has been recorded from a moving car equipped with sensors [3]. It includes camera images, LiDAR scans, high-precision GPS measurements and IMU accelerations from a combined GPS/IMU system.

All the trajectories of the dataset are stored as sequences in a broad category called Raw Data. In addition, each dataset falls into a subcategory, namely: "Road", "City", "Residential", "Campus" and "Person".

Using the Raw Data sequences they made a dataset called "Visual Odometry / SLAM Evaluation 2012" (in short Odometry dataset), which contains only the data related to the cameras (gray and color) and the LiDAR scans. In addition to the data extracted from

Nr.	Sequence name	Start	End
00:	2011_10_03_drive_0027	000000	004540
01:	2011_10_03_drive_0042	000000	001100
02:	2011_10_03_drive_0034	000000	004660
03:	2011_09_26_drive_0067	000000	000800
04:	2011_09_30_drive_0016	000000	000270
05:	2011_09_30_drive_0018	000000	002760
06:	2011_09_30_drive_0020	000000	001100
07:	2011_09_30_drive_0027	000000	001100
08:	2011_09_30_drive_0028	001100	005170
09:	2011_09_30_drive_0033	000000	001590
10:	2011_09_30_drive_0034	000000	001200

Table 4.1: Table to map Odometry sequences to Raw Data sequences.

the Raw Data sequences, in order to perform benchmarking, also the ground truth poses have been integrated. The Odometry dataset consists of 22 sequences: 11 sequences (00-10) with ground truth trajectories for training and 11 sequences (11-21) without ground truth for evaluation.

In order to allow the usage of the GPS and IMU data for the training data as well, they provided the mapping of the training set (sequences 00-10) to the corresponding Raw Data sequences of the KITTI dataset. The Table 4.1 lists the name, start and end frame of the Raw sequence that has been used to extract each Odometry training sequence.

Since our SLAM system extracts map priors, it fits best within an urban scenario. For this reason, to test our algorithm, we used the "Residential" sequence 07 named "2011_09_30_drive_0027" (see Figure 4.1). Of this sequence we've used the following data:

- The LiDAR (Velodyne HDL-64E) point clouds.
- The Inertial Navigation System (OXTS RT 3003) data (GPS/IMU)
- The calibration data of the relative positioning of the sensors (Camera, Camera-to-GPS/IMU, Camera-to-Velodyne)
- (From the Odometry dataset) The ground truth poses (trajectory) of the **camera gray left** in its coordinate system (i.e., z pointing forwards)
- The timestamps for all the data

The sequence can be downloaded from the KITTI web page [9]. All of its data is stored as binary and text files that should be converted into a ROS bag in order to be used by

Figure 4.1: Overview of the sequence used map from OpenStreetMap and its trajectory. A square represent a 10 × 10 m^2 area. The trajectory is distributed on a total area of approximately 230 × 180 m^2 , for a total length, estimated by the scan matching, of 687.116 m.

our systems. For this task, we use a tool called kitti2bag [10], to convert the Raw Data sequence into a ROS bag. To integrate the additional ground truth data given by the Odometry dataset, we had to extend this tool to also add the camera gray left ground truth poses.

The generated bag will have a frame associated to each sensor, an additional *base_link* frame at ground level in the center of the car and a *gt_camera_gray_left* which is the frame used to retrieve the ground truth poses (see Figure 4.2).

The frame associated to our robot is *base_link*, so the localization and maps are generated based on this frame. To correctly estimate the ground truth poses of the robot we have to transform the *gt_camera_gray_left* poses into *base_link* poses to retrieve the ground truth trajectory of the robot frame. Remember that the camera gray left is rigidly attached to the car and their relative position is known.

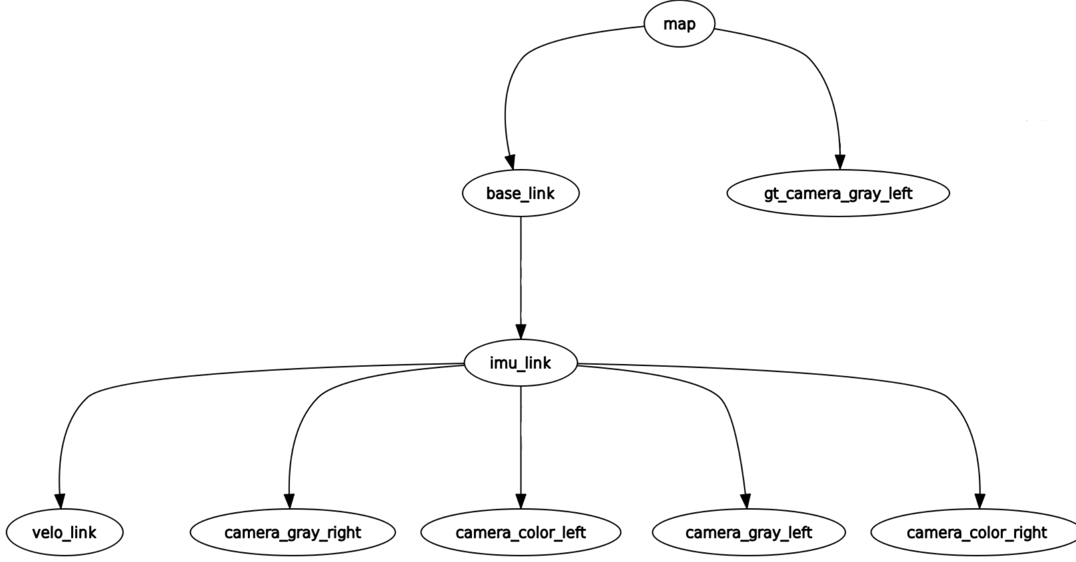


Figure 4.2: Tree frames of the generated bag.

To transform the poses of the *gt_camera_gray_left* into the poses of *base_link* we use the following formula:

$$gt_bl = initial_orient * bl_to_cgl_trans * gt_cgl * bl_to_cgl_trans.inv() \tag{4.1}$$

where:

- *gt_bl* is the *base_link* ground truth pose we want to derive within the *map* reference frame
- *initial_orient* is the initial orientation of the whole trajectory estimated by the IMU sensor, since it is not given by the ground truth
- *bl_to_cgl_trans* is the static transformation from *base_link* to *camera_gray_left*
- *gt_cgl* is the given ground truth pose of *gt_camera_gray_left*
- *bl_to_cgl_trans.inv()* is the inverse of the *bl_to_cgl_trans* transformation matrix

This computation is needed because the given ground truth pose are of a different object and in a different reference frame (namely *gt_camera_gray_left* and its reference frame). Using Figure 4.3 as reference, the given ground truth pose of *gt_camera_gray_left* is tf2 (in red its trajectory) in reference frame 2 (different than frame1 = *map*), so its

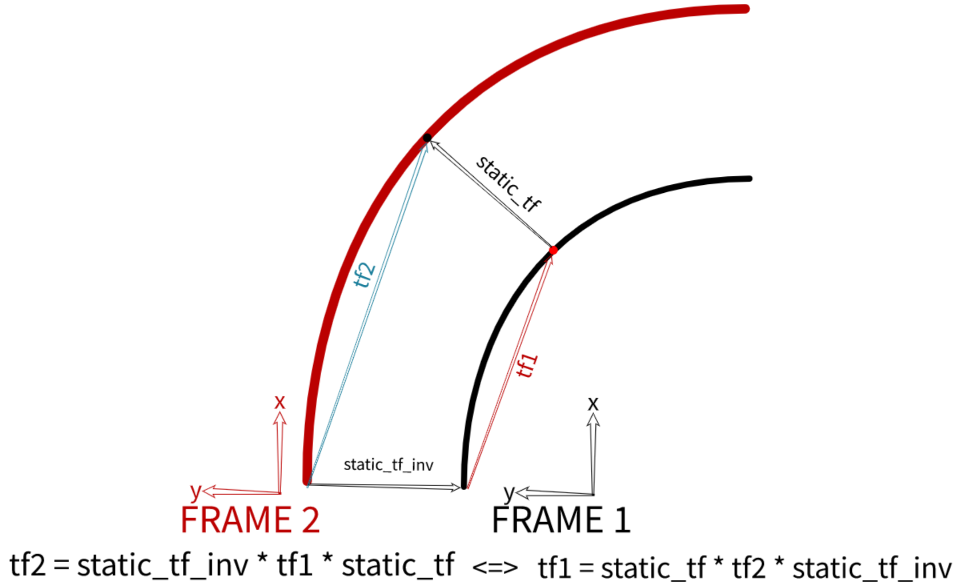


Figure 4.3: Visualization of ground truth poses transformation.

coordinates stems from the origin of frame 2. The *base_ link* ground truth pose we want to find is tf1, which is in the reference frame 1 (corresponding to *map* frame). The static transformation *bl_ to_ cgl_ trans* between *base_ link* and *camera_ gray_ left* corresponds to the static_tf transformation in the Figure 4.3.

By visual proof we can see that, to extract the *base_ link* ground truth pose, we have to transform the *gt_ camera_ gray_ left* pose from *map* to its real reference frame i.e. by left multiplying *gt_ cgl* by *bl_ to_ cgl_ trans*. After the last multiplication, we find the *camera_ gray_ left* pose in the *map* frame. Finally, we can retrieve the *base_ link* ground truth pose by right multiplying the *camera_ gray_ left* pose found previously by the static transform bl_to_cgl_trans.inv().

To compute the localization errors, the *gt_ camera_ gray_ left* ground truth poses are transformed into the *base_ link* ground truth poses using the Formula 4.1.

4.1.2. RPE

As already stated, the RPE, Relative Pose Error, measures the local accuracy of the trajectory between consecutive robot poses [18].

Considering two consecutive poses of the robot, T_{i-1} and T_i, expressed as SE(2) transformations with respect to a global frame of coordinates, the relative pose transformation

between them is computed as:

$$T_{x_rel_i} = (T_{i-1})^{-1} * T_i \tag{4.2}$$

Using the same formula, it computes also the relative pose transformation between two consecutive poses of the ground truth obtaining $T_{gt_rel_i}$.

Given $T_{x_rel_i}$ and $T_{gt_rel_i}$, corresponding to the same pair of poses, the difference between the relative pose transformations is computed as:

$$T_{delta_i} = (T_{gt_rel_i})^{-1} * T_{x_rel_i} \tag{4.3}$$

where T_{delta_i} indicates how much the i'th robot relative pose transformation differs from the ground truth one. A high value of this error, divided in two components (translation and rotation), corresponds to a high error on the estimate of transformation between the pair of poses considered.

In particular, the translational RPE is computed as:

$$t_{RPE} = \frac{1}{n} * \sum_{i=1}^{n} trans(T_{delta_i}) \tag{4.4}$$

where trans(T_{delta_i}) is the translational component of T_{delta_i} and n is the number of couples of robot poses in the sequence.

The rotational RPE is computed as:

$$r_{RPE} = \frac{1}{n} * \sum_{i=1}^{n} rot(T_{delta_i}) \tag{4.5}$$

where $rot(T_{delta_i})$ is the rotational component of T_{delta_i}.

4.1.3. ATE

The ATE, Absolute Trajectory Error, measures the accuracy of the estimated robot pose within a fixed reference frame, comparing the absolute poses between the estimated and ground truth trajectories [18].

Taken the ground truth pose T_{gt_i} of the trajectories, transformed as seen in Subsection

4.1.1, and the corresponding estimated robot pose T_{x_i}, the delta transformation between them is computed as:

$$T_{delta_i} = T_{gt_i}^{-1} * T_{x_i} \tag{4.6}$$

The ATE is computed only on the translational component, and it is:

$$ATE = \frac{1}{m} * \sum_{i=1}^{m} trans(T_{delta_i}) \tag{4.7}$$

where $trans(T_{delta_i})$ is the translational component of T_{delta_i} and m is the number of poses in the sequence.

4.2. Final Results

We present now the results before and after applying the improvements on the baseline system. In Subsection 4.2.1 we show the estimated numerical errors about the robot localization. In Subsection 4.2.2 we propose a visual comparison of the alignment between the map point cloud and the estimated buildings.

4.2.1. Errors

We present the computed numerical errors, namely ATE and RPE, for the following systems:

- hdl_graph_slam with only odometry and loop closures
- the baseline system before applying any improvement
- the improved system with all the solutions described in Chapter 3

Table 4.2 shows the results for the three systems. The values for ATE and t_RPE are expressed in meters, while r_RPE is expressed in radians. all the values are expressed as mean ± standard_deviation.

Before any consideration about the results, it's worth mentioning that two consecutive poses of the robot are, in average, 2 meters apart. So the t_RPE values are expected to be much smaller than 2 meters as well as we expect small values for the r_RPE for the relative orientation error between two consecutive poses.

From the Table 4.2 we can tell that hdl_graph_slam, without any additional sensor, be-

	ATE [m ± m]	t_RPE [m ± m]	r_RPE [rad ± rad]
hdl_graph_slam	1.682 ± 1.002	0.04556 ± 0.04196	-1.241e-05 ± 0.001097
baseline	1.542 ± 0.9293	0.2361 ± 0.3290	0.001864 ± 0.03288
improved baseline	1.223 ± 0.7027	0.04358 ± 0.04261	-3.074e-05 ± 0.001079

Table 4.2: ATE and RPE errors

side the Velodyne LiDAR, has already low ATE and RPE. This proves that the underlying system should be considered reliable and the robot localization should be corrected just slightly.

The baseline system, integrating map priors, lowers the ATE and localizes the robot better than hdl_graph_slam as expected. On the other hand, as anticipated in Section 3.4, the RPE is much worse than hdl_graph_slam. This is mainly caused by the high number of map prior constraints for each robot node that makes them move abruptly, worsening the RPE between consecutive robot poses.

Finally, after all the improvements described in Chapter 3, our system improved the ATE by 44 centimeters with respect to hdl_graph_slam, and 32 centimeters with respect to the baseline. Considering that the error is already low, we should express them in percentages, so the ATE was reduced by:

- **27.3%** with respect to hdl_graph_slam
- **20.7%** with respect to the baseline system

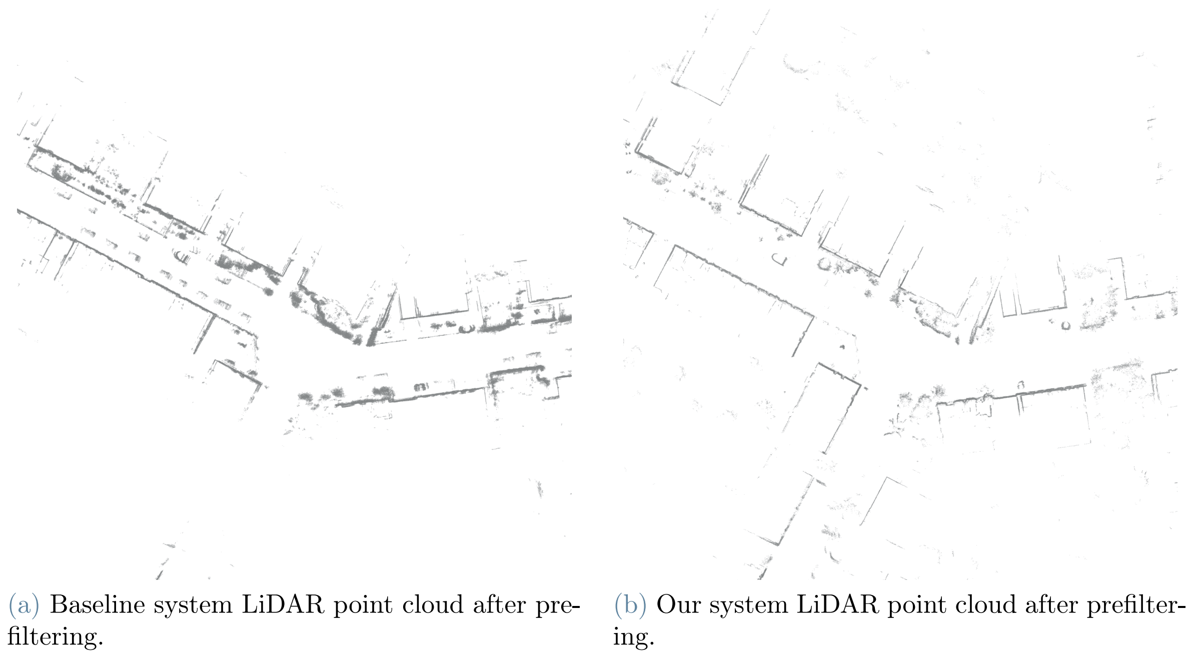
Both the RPEs of our system improved with respect to hdl_graph_slam, while the baseline issue has been solved as explained in Section 3.4.

The maximum absolute value of r_RPE for hdl_graph_slam is 0.0011096941, while for our system is 0.0011095379, reducing the r_RPE by just 0.01%. While our system's t_RPE improved by **4.37%** with respect to hdl_graph_slam.

4.2.2. Visual Evaluation

We provide a visual evaluation of the generated maps of the baseline system and the ones of our system as a comparison. In particular,the whole map is generated using the Sequence 07 of the KITTI Odometry dataset and we will show the portions of it as the robot passed through them. In each figure, we show on the left the baseline map, on the right our map. Every figure has three overlapped maps:

- The map generated by merging all the LiDAR point clouds (transparent black)



(a) Baseline system LiDAR point cloud after prefiltering.

(b) Our system LiDAR point cloud after prefiltering.

Figure 4.4: LiDAR point clouds after prefiltering.

- The map of the buildings before being aligned (less thicker black lines)
- The map of the buildings after being aligned (baseline: red, our: green)

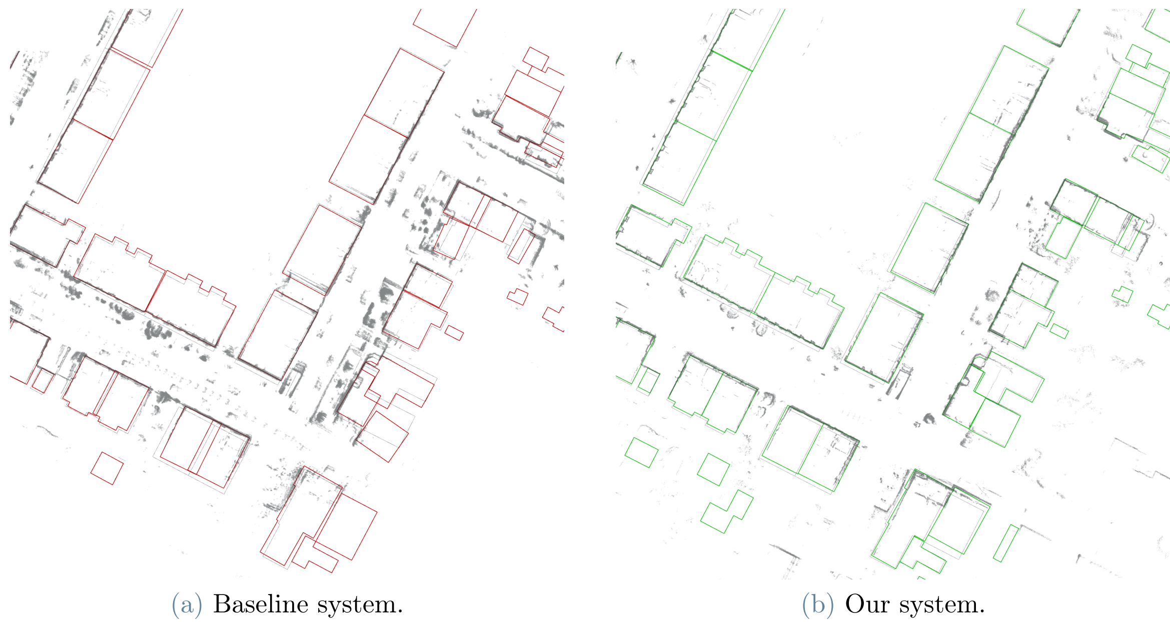
Before showing the aligned maps, we want to highlight that our prefiltering algorithm removes more outliers than the baseline. Consequently, the LiDAR maps are visibly cleaner, as shown in Figure 4.4.

To visually evaluate the maps, we should assume that the LiDAR generated maps are the target used to correct the OSM maps. Therefore we expect that the colored buildings to be overlapped as better as possible with the LiDAR maps.

In Figure 4.5 we can see that our system is more robust to the LiDAR noise. Thanks to our line extraction algorithm (see Section 3.2), only the relevant information is extracted from the LiDAR point clouds.

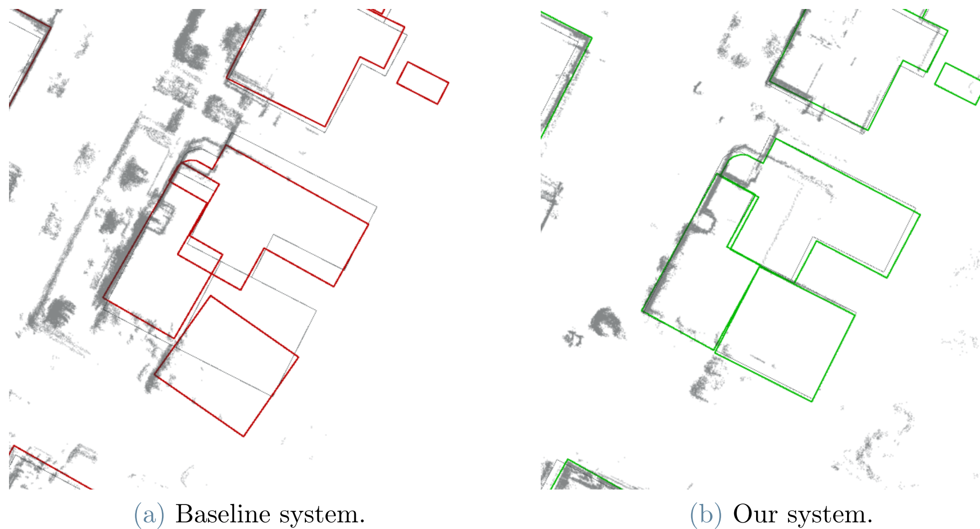
We have to mention that our algorithm assumes that the buildings are formed by lines and edges, but sometimes this assumption is not met, as with the building in Figure 4.6, nonetheless the results are still acceptable.

In Figure 4.7, since the Sequence was recorded in 2011, we can see a newly constructed building showed by the OSM map (in 2022) but not sensed by the LiDAR. Our system, thanks to the non-overlapping constraints, can successfully prevent the newly constructed building to overlap other buildings (as shown in Figure 4.8).



(a) Baseline system. (b) Our system.

Figure 4.5: Map results: frame 1.



(a) Baseline system. (b) Our system.

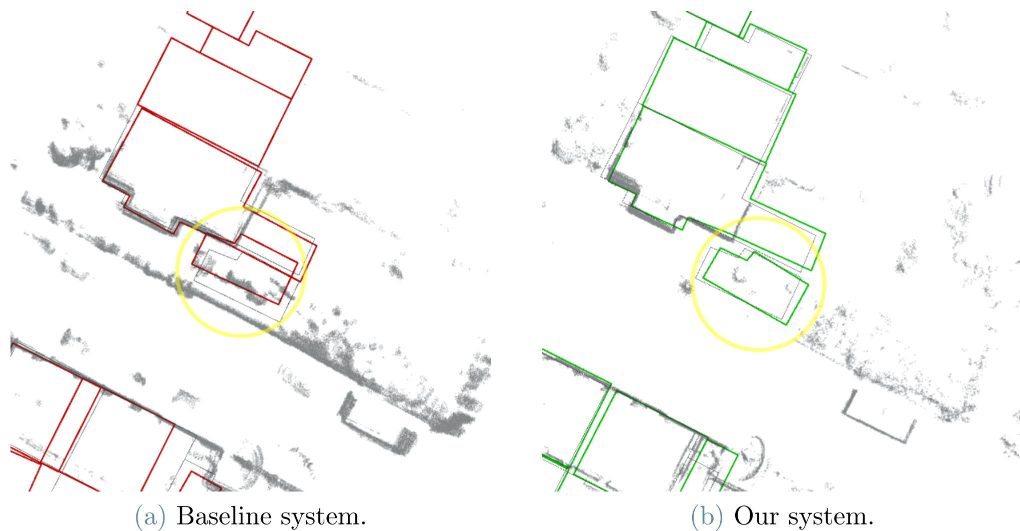
Figure 4.6: Example of a circular shaped building.

In Figure 4.9 we can see how the overlapping between the buildings is totally removed, compared to the baseline system. Moreover, the lines of the buildings are perfectly aligned with the LiDAR map (Figure 4.10).

Despite the algorithm is designed to be applied in urban environments, it proved to be robust also in environment with few buildings. In Figure 4.11 the robot runs by a park and the global matching, used to localize the robot, performs as expected relying only on the buildings on his left.

(a) Baseline system. (b) Our system.

Figure 4.7: Map results: frame 2.



(a) Baseline system. (b) Our system.

Figure 4.8: Example of a newly constructed building.

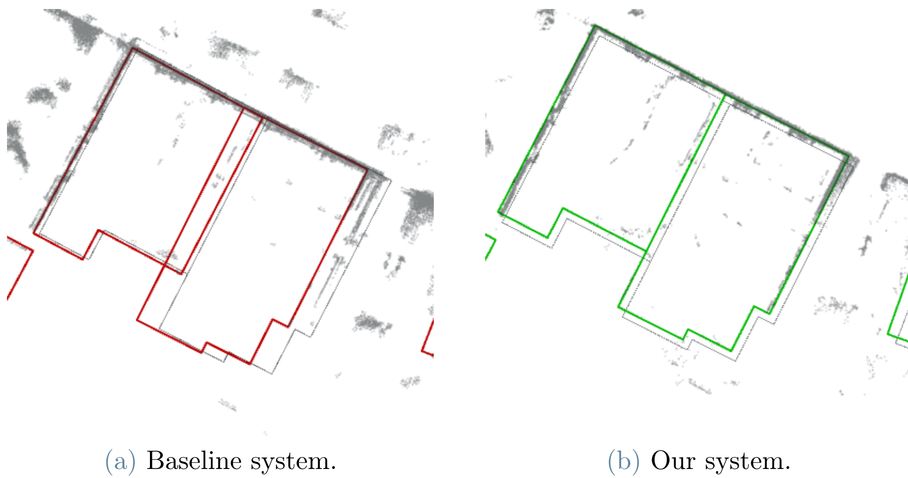
Sometimes, the LiDAR lasers can go through windows and map the interior of the buildings. Our algorithm may see some additional and unexpected edges that compromise the building correction, as shown in Figure 4.12.

In Figure 4.13, our system LiDAR can see more buildings than the baseline system, probably due to an issue in the baseline prefiltering process.

In Figure 4.14 we can see that sometimes the OSM buildings are not shaped correctly.

(a) Baseline system. (b) Our system.

Figure 4.9: Map results: frame 3.

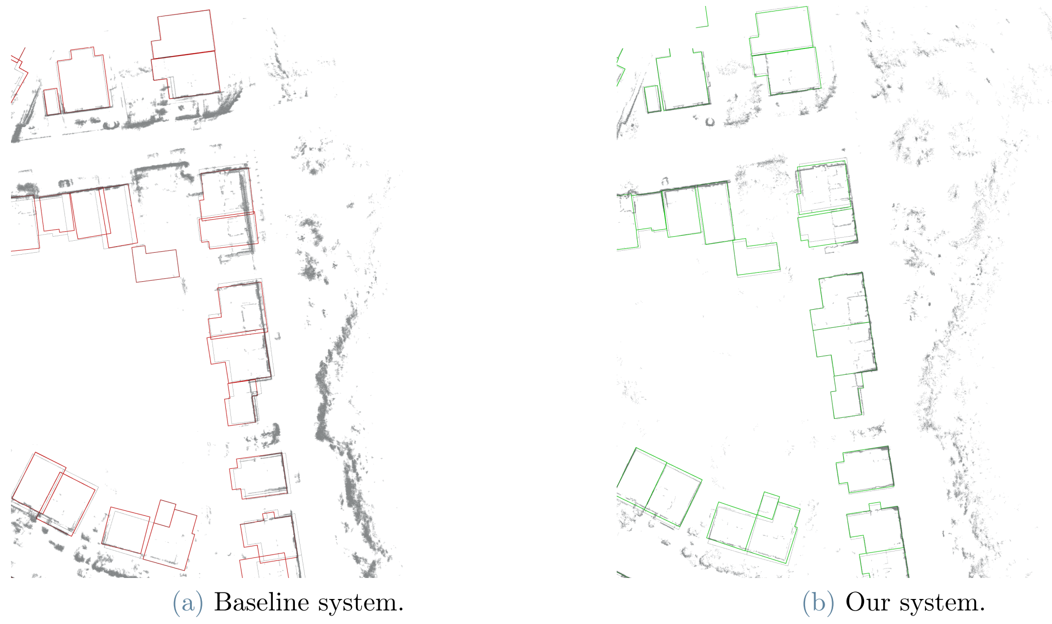


(a) Baseline system. (b) Our system.

Figure 4.10: Example of two overlapped buildings.

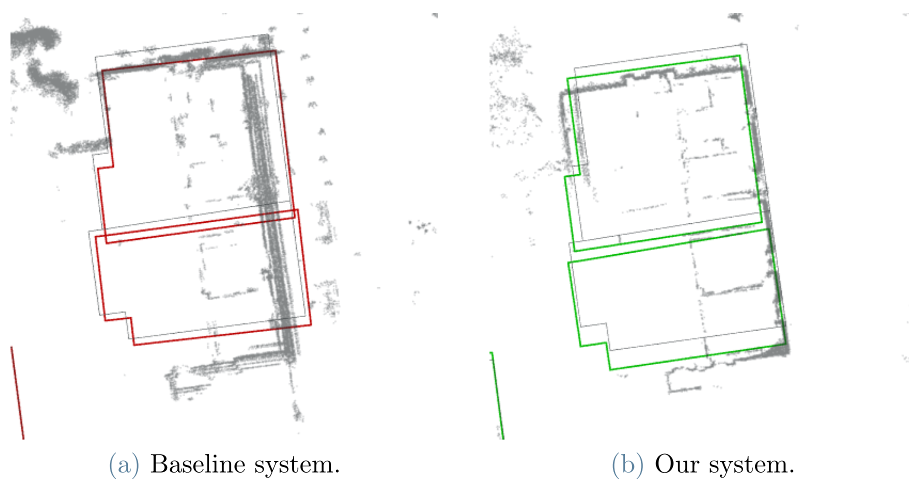
Nevertheless, our system can correctly position the building in the center, if it is seen from multiple angles as shown in Figure 4.15.

Lastly in Figure 4.16, closing the loop, we can see that our system can correctly handle the "corridor" like situations in which the buildings are all bordering, without overlapping any of them.



(a) Baseline system. (b) Our system.

Figure 4.11: Map results: frame 4.

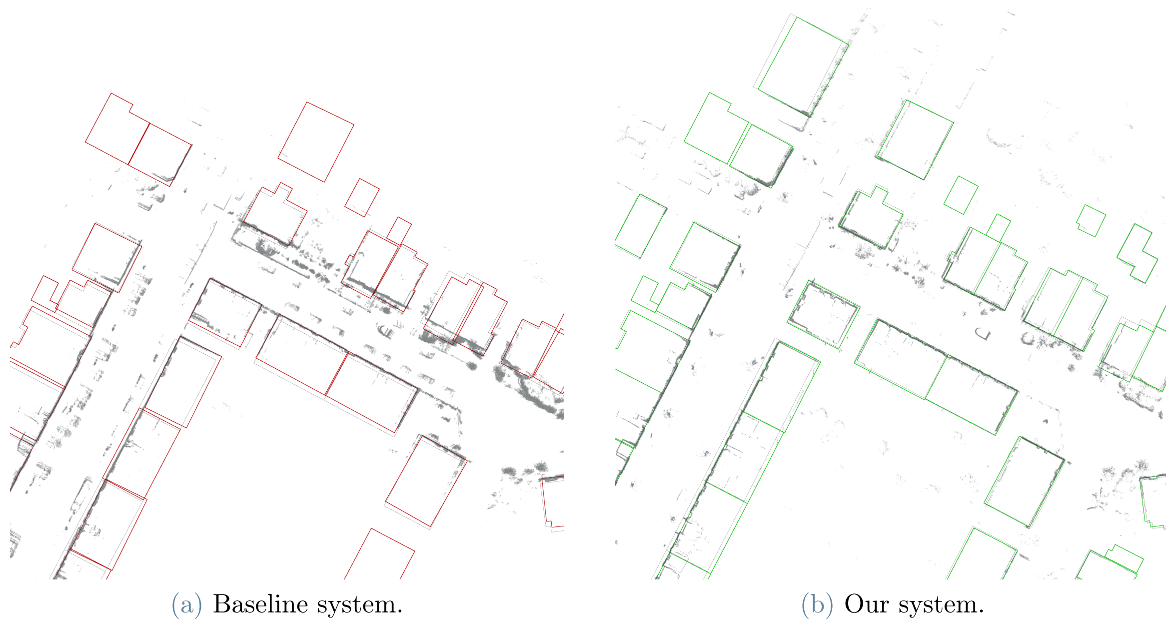


(a) Baseline system. (b) Our system.

Figure 4.12: Example of the LiDAR seeing inside the buildings.

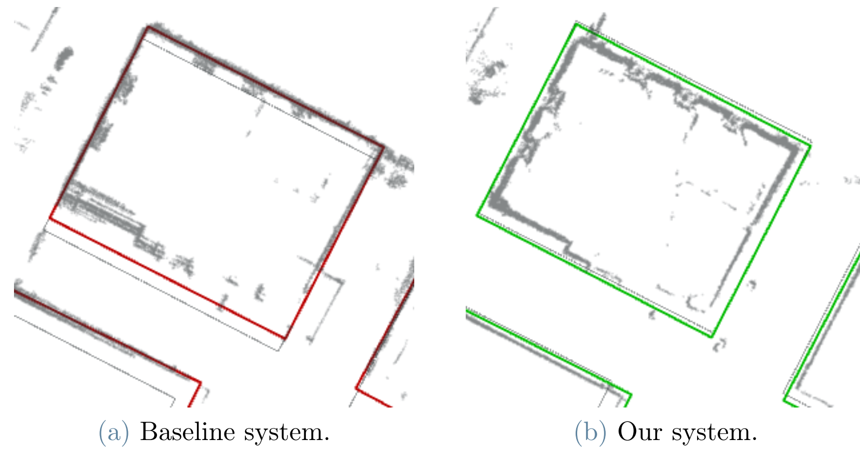
(a) Baseline system. (b) Our system.

Figure 4.13: Map results: frame 5.



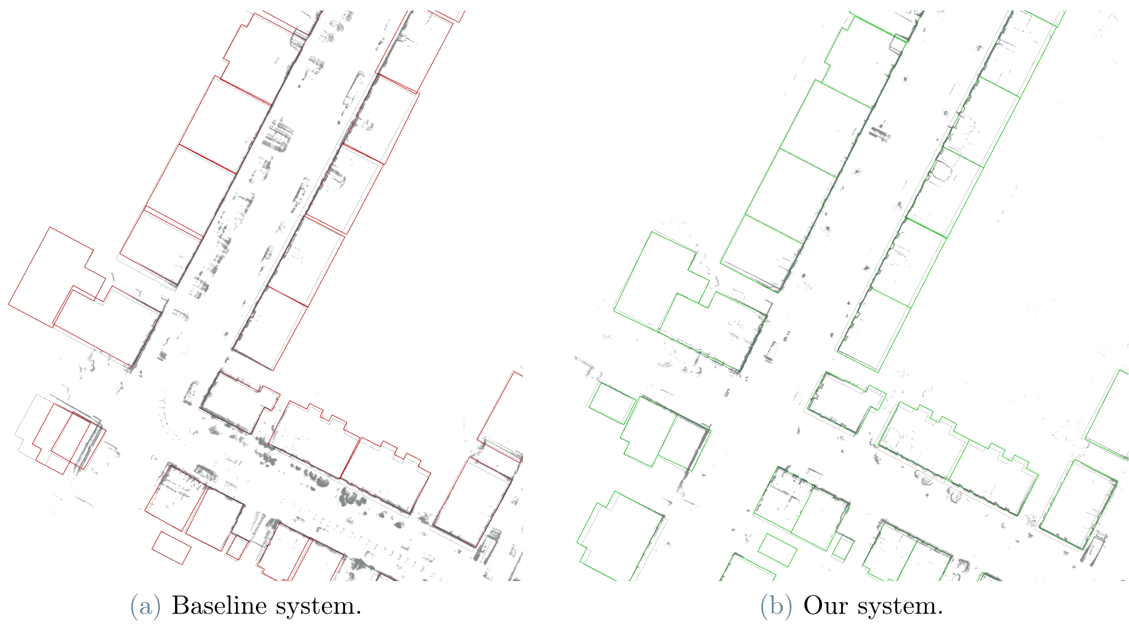
(a) Baseline system. (b) Our system.

Figure 4.14: Map results: frame 6.



(a) Baseline system. (b) Our system.

Figure 4.15: Example of a badly shaped building from OSM.



(a) Baseline system. (b) Our system.

Figure 4.16: Map results: frame 7.

	Execution time [ms]
avg_line_segments_extraction_time	463.749
avg_global_line_matching_time	996.182
avg_global_fgicp_matching_time	351.508
avg_local_line_matching_time	0.796748
avg_local_fgicp_matching_time	55.2481

Table 4.3: Global and local matching execution times

4.2.3. Space and Time Efficiency

In this last subsection we want to briefly discuss about our system overall efficiency. During the development, we re-implemented some algorithms, to reduce time and space complexity.

Compared to the maps generated by the baseline system as explained in the Section 2.5, our maps have a density of points three times smaller.

On the other hand, our line segments-based scan matching algorithm, since it has a search space that grows quadratically with the number of lines and edges, is slower than the algorithm used by the baseline (FastGICP) when performing global matching, while it is much faster when performing local matching, with a low number of line segments, as shown in Table 4.3.

Notice that we divided the global line matching in two parts, to highlight the execution time to extract the line segments from the LiDAR point clound and the execution time to perform the actual global matching.

The global matching takes 1.5 seconds instead of 0.35 seconds of the FastGICP scan matching. Meanwhile, the local scan matching is 50 times faster than FastGICP, since a single building has a low number of geometrical features to be matched. This means that, when performing local matching, the execution time is improved by 50 ms for each nearby building. So, with 10 buildings to perform local matching, we regain 500 seconds of computation time, partially compensating the global scan matching long execution time.

5 | Conclusions

In this thesis we presented some improvements to an existing graph-based SLAM with map priors system [4]. Our aim was to deal with the issues left in the original system, by implementing dedicated algorithms. In particular, the implemented line segments-based scan matching algorithm can reliably extract the map priors, solving the majority of the issues related to robot localization and, especially, map correction.

The geometrical constraints solved the overlapping buildings problem and, thanks to the adopted solution, the downloading process of the maps from the OSM servers has become more robust.

The localization of the robot was improved and the drifting of the LiDAR scan matching is successfully corrected, even in long travelled distances.

The system, given a high precision localization of the robot, can be solely used to correct the maps. A possible application could be to contribute and speed up the validation process of the open database of OpenStreetMap. Conversely, given a really precise map, the system can be solely used to perform a reliable robot localization.

Our system has also a new set of parameters that can be finely tuned and adapted to extend its possible applications which include different environments and to create maps based on different landmarks.

5.1. Future Improvements

Despite the great number of issues solved, there is still room for further improvements.

A notable problem left to solve is related to the map priors encoded into the graph. As we have seen, sometimes the extracted constraints can only limit two degrees of freedom of the plane. For example, if the robot sees only a line segment of a building, it cannot tell the translation along the line direction to correctly determine its pose. In these cases, to guess the building pose, we use the projection of a line segment as seen in Section 3.3.3. In our system, the used map prior constraint uses the guessed pose, that may be wrong.

To solve this, the g2o framework, used to construct the pose graph, can be extended to make the graph edges constrain only some degrees of freedom.

Another improvement could be to enlarge the robot vision and extract the map priors using more than one consecutive LiDAR point cloud. This would increase the chances to see a building from multiple angles, increasing the number of edges that are essential to perform global and local matching.

In out system, the line segments are extracted using the 2D LiDAR point cloud. It may be possible to use directly the 3D point cloud to extract planes rather than lines. We believe that it can improve the feature extraction algorithm and reduce the outliers dramatically.

The last improvement we can think of is to use additional data. For example, by using the cameras, we can perform semantic segmentation of the point clouds. This may help to enhance the prefiltering process to only choose the points belonging to the buildings.

Bibliography

[1] P. J. Besl and N. D. McKay. A method for registration of 3-d shapes. *IEEE Trans. Pattern Anal. Mach. Intell.*, 14(2):239–256, 1992. URL http://dblp.uni-trier.de/db/journals/pami/pami14.html#BeslM92.

[2] A. Doucet, N. de Freitas, K. Murphy, and S. Russell. Rao-blackwellised particle filtering for dynamic bayesian networks, 2013. URL https://arxiv.org/abs/1301.3853.

[3] A. Geiger, P. Lenz, and R. Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012.

[4] V. Gobbi. Simultaneous localization and mapping in urban scenarios with non-rigid openstreetmap priors. Master's thesis, Politecnico di Milano, 2021.

[5] G. Grisetti, S. Grzonka, C. Stachniss, P. Pfaff, and W. Burgard. Efficient estimation of accurate maximum likelihood maps in 3d. In *2007 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3472–3478, 2007. doi: 10.1109/IROS.2007.4399030.

[6] Y. Iskender. Line segment based range scan matching without pose information for indoor environments. Master's thesis, Bilkent University, 2008. URL http://repository.bilkent.edu.tr/handle/11693/15381.

[7] Kenji Koide. hdl_graph_slam, 2022. URL https://github.com/koide3/hdl_graph_slam.

[8] Kenji Koide. ndt_omp, 2022. URL https://github.com/koide3/ndt_omp.

[9] KITTI contributors. KITTI Raw Data sequences. https://www.cvlibs.net/datasets/kitti/raw_data.php, 2012. Accessed: 2022-09-12.

[10] kitti2bag contributors. kitti2bag convert kitti dataset to ros bag file the easy way! https://github.com/tomas789/kitti2bag, 2019. Accessed: 2022-09-12.

[11] K. Koide, M. Yokozuka, S. Oishi, and A. Banno. Voxelized gicp for fast and accurate 3d point cloud registration. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 11054–11059, 2021. doi: 10.1109/ICRA48506.2021.9560835.

[12] H. Kretzschmar and C. Stachniss. Information-theoretic compression of pose graphs for laser-based slam. *The International Journal of Robotics Research*, 31:1219 – 1230, 2012.

[13] R. Kümmerle, G. Grisetti, H. Strasdat, K. Konolige, and W. Burgard. G2o: A general framework for graph optimization. In *G2o: A general framework for graph optimization*, pages 3607 – 3613, 06 2011. doi: 10.1109/ICRA.2011.5979949.

[14] OpenStreetMap contributors. Planet dump retrieved from https://planet.osm.org . `https://www.openstreetmap.org`, 2017.

[15] R. B. Rusu and S. Cousins. 3D is here: Point Cloud Library (PCL). In *IEEE International Conference on Robotics and Automation (ICRA)*, Shanghai, China, May 9-13 2011. IEEE.

[16] A. Segal, D. Hähnel, and S. Thrun. Generalized-icp. In J. Trinkle, Y. Matsuoka, and J. A. Castellanos, editors, *Robotics: Science and Systems*. The MIT Press, 2009. ISBN 978-0-262-51463-7. URL `http://dblp.uni-trier.de/db/conf/rss/rss2009.html#SegalHT09`.

[17] Stanford Artificial Intelligence Laboratory et al. Robotic operating system, 2022. URL `https://www.ros.org`.

[18] J. Sturm, N. Engelhard, F. Endres, W. Burgard, and D. Cremers. A benchmark for the evaluation of rgb-d slam systems. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 573–580, 2012. doi: 10.1109/IROS.2012.6385773.

[19] S. Thrun, W. Burgard, and D. Fox. Probabilistic robotics (intelligent robotics and autonomous agents), 2001.

A | Appendix A

A.1. Signed Vector Projection Norm

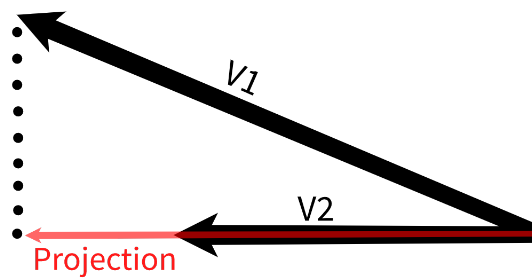


Figure A.1: Visualization of a vector projection.

Using as reference the Figure A.1, we can compute the signed norm of the projection of a vector V1 on a second vector V2 by using the scalar product:

$$\frac{V1 \cdot V2}{\|V2\|} = cos(\angle(V1, V2)) \cdot \|V1\| \tag{A.1}$$

List of Figures

2.1 Graphs of a graph-based SLAM system 7
2.2 Ros Components 7
2.3 HDL Graph SLAM system architecture 9
2.4 Pipeline of the baseline system 12
2.5 Visualization of the conversion of the point clouds from 3D to 2D. 13

3.1 Buffering visualization 16
3.2 Line extraction 19
3.3 Results of line segments extraction 20
3.4 Edge types 21
3.5 Edge matching 22
3.6 Displaced lines only edges 23
3.7 Line matching 24
3.8 Line to line distances 25
3.9 Line to line distance without coverage problem 26
3.10 Predominated odometry constraint 29
3.11 Baseline overlapping buildings problem 32
3.12 Building moved to avoid overlapping. 34

4.1 Sequence 07 overview 37
4.2 Tree frames of the generated bag. 38
4.3 Visualization of ground truth poses transformation. 39
4.4 LiDAR point clouds after prefiltering. 43
4.5 Map results: frame 1. 44
4.6 Example of a circular shaped building. 44
4.7 Map results: frame 2. 45
4.8 Example of a newly constructed building. 45
4.9 Map results: frame 3. 46
4.10 Example of two overlapped buildings. 46
4.11 Map results: frame 4. 47

4.12 Example of the LiDAR seeing inside the buildings. 47
4.13 Map results: frame 5. 48
4.14 Map results: frame 6. 48
4.15 Example of a badly shaped building from OSM. 49
4.16 Map results: frame 7. 49

A.1 Visualization of a vector projection. 55

List of Tables

4.1 Mapping Table to Raw Data 36
4.2 ATE and RPE errors 42
4.3 Global and local matching execution times 50

Acknowledgements

We would like to thank all the contributors who have supported us in this master's degree thesis. A special thanks goes to our parents and friends who relieved and encouraged us throughout our studies.