SAPIENZA
UNIVERSITÀ DI ROMA

Position tracking in 3D scenes using high order primitives

Master thesis

Edoardo Ghini

ghini.1616301@studenti.uniroma1.it

Department of Computer, Control and Management Engineering Antonio Ruberti
Sapienza University of Rome

Supervisor:
Prof. Dr. Giorgio Grisetti
Cosupervisor:
Irvin Aloise

October 21, 2019

Abstract

The goal of this thesis is to build a system that solves the LIDAR Odometry and Tracking problem using the measurements of a 3D LIDAR sensor mounted on a mobile robot. In particular, we extract high level features - namely points, lines and planes - from the environment to achieve fast performances. The features are generated with normal-based clustering approach. The key idea is that these features capture meaningful information of the environment and their efficient parametrization results in a lightweight optimization problem. We use a unified parametrization based on *degenerate quadrics* to represent the aforementioned geometric primitives, resulting in a less complex implementation. Thanks to this, the robot trajectory is estimated in a effective and efficient way.

Contents

Chapter 1

Introduction

The approach discussed is part of one of the most fundamental problems in robotics, the *simultaneous localization and mapping problem*, also contracted in **SLAM**. This problem has been addressed since the early 1980s. Many possible solutions to the SLAM problem can be found, depending on the environment, the sensors or the application that one wants to address. In this work will be addressed the problem of LIDAR Odometry and Tracking, using high order geometric primitives as features. More in detail, given the 3D raw point-cloud obtained from the sensor, we extracted different geometric features - namely lines and planes - through normal-based clustering.

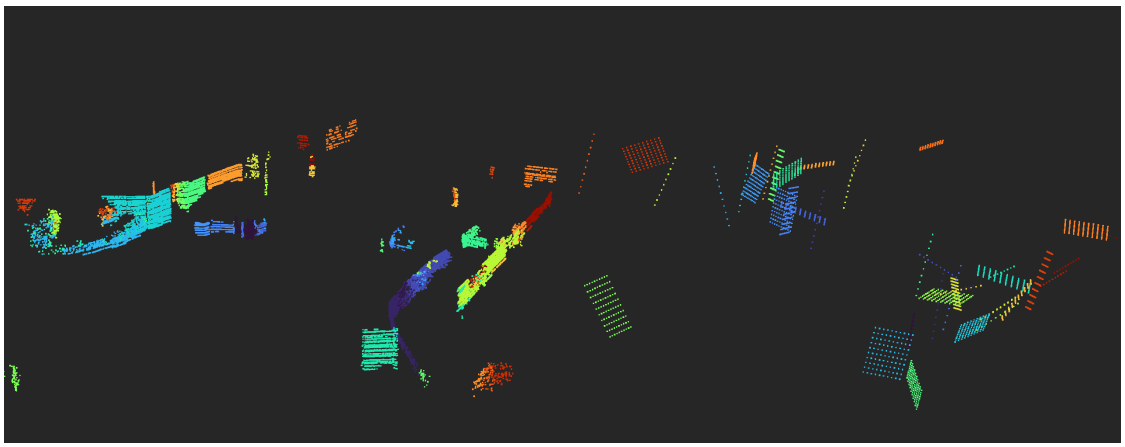


Figure 1.1: High order primitives: on the left clusterized data points, on the right: high order features (lines and planes)

Then, using this features we will compute an odometry estimation between subsequent scans, with a Least-Squares based optimization algorithm. Finally, we will estimate the full trajectory of the robot using only the sensor information. The remaining of this thesis is organized as follows:

- Chapter 2: summary of solutions already proposed to the problem.
- Chapter 3: problem statement and the theoretical basement of this work.
- Chapter 4: in-depth explanation of all the components of the proposed system.
- Chapter 5: presents the results obtained.

- Chapter 6: final considerations and possible future enhancements.

Chapter 2

Related Work

The SLAM problem is well-known since many decades [1] [2]. Early solution were based on Gaussian Filters [3] [4] [5] [6] or Particle Filters [7] [8] [9].

The solutions based on Gaussian Filters are not able to address successfully the non-linearity of the problem. Instead, the particle filter ones imply a problem of a size unsuitable for acceptable results. For this reasons smoothing solutions became more attractive.
In particular graph-based SLAM rapidly became the gold-standard method to solve the problem. In this paradigm, the SLAM system is composed by two parts:

- the front-end that processes sensor data to build a hyper graph containing the map and the robot trajectory.
- the back-end that optimizes the graph to reduce inconsistencies.

The graph-based paradigm was firstly introduced in the work of Lu and Milios [10] in the context of the laser-scan registration. This thesis falls in the graph-SLAM paradigm. In particular, it proposes a 3D-LiDAR based SLAM system. 3D Lidars are common nowadays in autonomous driving applications and the community proposed several approaches that work with this sensor.
A relevant source of this thesis is the work of Zhang et al. [11]. They proposed a LIDAR Odometry solution named LOAM which was fast and efficient, exploiting a 2D LIDAR sensor mounted on a servomotor to get 3D readings of the environment.
In this thesis, we make use of a different sensor called 3D LIDAR which has a faster frame-rate and a better resolution. Subsequently, Zhang et al. in [12] proposed a novel method for LIDAR odometry that combines visual data coming from rgb camera to reach better performances. They used this additional data channel to recognize whether the robot is passing through an already visited place and generate loop closures.
Recently Shan et al. [13] proposed a novel LIDAR odometry solution that takes advantage additional information coming from the ground. It performs feature extraction of edges and planes. Therefore is computationally lighter than LOAM, scoring similar performances.

Deschaud et al. [14] in their system IMLS-SLAM, proposed a solution to the LIDAR Odometry problem that exploits a novel optimization algorithm. They use surface representations of the point clouds to relax the constraints during optimization.

Behley et al. [15] used surfel-based landmarks to perform LiDAR SLAM. In their system, called SuMa, they extract and work with surfels - namely small flat surfaces, which bring more information on the structure of the environment, resulting in a more robust system overall. They also execute loop-closure to recover from the drifting error.

Chapter 3

Basics

In the Robotics community *SLAM* is the problem of learning the trajectory of a robot and the map of the enviroment in which it is moving.
Solving the SLAM problem has been considered quite complex. In the literature, several solutions have been explored but two macro-categories can be distinguished: filtering and smoothing approaches. The former consists of an online state estimation of both the current robot position and of the map structure. This estimate is refined thanks to the new information gathered from the sensors. Some examples of filtering approaches are Kalman filters, information filters and also particle filters. Smoothing approaches, instead, estimate the whole robot trajectory using a set of sensor measurements referring to wider time window that extends in the past. In general, these solutions rely on Least-Squares error minimization to reach convergence. For the fact that smoothing solutions optimize over an entire set of measurements they achieve better performances than filtering alternatives.
In particular, the graph-SLAM paradigm introduced by Lu and Milos [10] is considered the gold standard in the current state-of-the-art SLAM systems. This method is based on the collaboration of two sub modules: front-end and back-end.
In the front-end sub module raw sensor data is processed and an *hyper-graph* is incrementally built. The nodes of this graph represent either the pose of the robot or the position of salient points of the environment. The edges contain spatial constraints between subsets of nodes. Since the sensor is affected by noise, the *hyper-graph* might have some inconsistencies.
In the back-end sub module the information stored in the *hyper-graph* is used to compute the best node configuration according to the constraints embedded in the edges. In the remaining of this chapter, we will introduce the basic components of a Graph-SLAM system. In particular:

- Sec. 3.1 presents the Least-Squares formulation of the SLAM problem.
- Sec. 3.2 analyzes how to perform Data-Association, that is fundamental to generate the edges of the graph.
- Sec. 3.3 proposes an overview of the Iterative Closest Point algorithm, used to compute the motion between 2 point-clouds.
- Sec. 3.4 introduces the problem of LiDAR based odometry, which represents the focus of this work.

3.1 Least-Squares Problem Formulation

In this section is proposed a procedure of Least-Squares state extimation of non-linear stationary systems [16]. The elements that characterize the SLAM problem as a stationary system $\mathcal{W}$ are the following:

- the robot is moving in an unknown environment following a trajectory described by a sequence of random variables $\mathbf{x} = \{\mathbf{x}_1, .. \mathbf{x}_N\}$.
- the robot starts from a position $\mathbf{x}_0$
- while moving the robot gathers Odometry measurements $\mathbf{u} = \{\mathbf{u}_1, .. \mathbf{u}_T\}$
- the state of the system is indirectly observed by a set of sensors from which we obtain measurements $\mathbf{z} = \{\mathbf{x}_1, .. \mathbf{z}_M\}$
- the environment can be parametrized in a map with different degrees of granularity:
  - occupancy grid
  - set of points
  - set of high order geometric primitives

Since the measurements are affected by noise, it is impossible to estimate in closed form the *state* of the system given the measurements. Still, one can compute a distribution over the potential system *states* given these measurements.
In conclusion the objective of this section consists of estimating the posterior probability of the robot trajectory $\mathbf{x}_{1:T}$ and the map $\mathbf{m}$ as in 3.1, given the initial position $\mathbf{x}_0$, the sensors measures $\mathbf{z}_{1:T}$ and the odometry measures $\mathbf{u}_{1:T}$.

$$p(\ \mathbf{x}_{1:T},\ \mathbf{m}\ |\ \mathbf{z}_{1:T},\ \mathbf{u}_{1:T},\ \mathbf{x}_0\ ) \tag{3.1}$$

3.1.1 Gaussian assumption

For simplicity, since in this work odometry measurements are absent, conditional dependence from $\mathbf{u}_{1:T}$ is removed. Then for brevity $\mathbf{x}_0$ is omitted.
Consequently, the solution given by the *Least-Squares* algorithm can be compactly described by 3.2 as the state which maximizes the probability of observing that state given a set of measures.

$$\mathbf{x}^* = \underset{\mathbf{x}}{\operatorname{argmax}}\, p(\mathbf{x}|\mathbf{z}) \tag{3.2}$$

To this end, we compute the optimal state $\mathbf{x}$ that best explains the measurements $\mathbf{z}$.

$$p(\mathbf{x}|\mathbf{z}) = \frac{p(\mathbf{z}|\mathbf{x})p(\mathbf{x})}{\mathbf{z}} \propto p(\mathbf{z}|\mathbf{x}) \tag{3.3}$$

$$= \prod_i p(\mathbf{z}^{[i]}|\mathbf{x}) \tag{3.4}$$

Applying the *Bayes Rule* 3.3 and considering the measurements conditionally independent from each other 3.4 is obtained.
Given the measurement $\mathbf{z}$ with $\mathbf{\Sigma}$ we indicate the covariance matrix associated to the measurement. For convenience, in the remaining of the manuscript we will make use of the information matrix $\mathbf{\Omega} = \mathbf{\Sigma}^{-1}$ to address the noise relative to the measurement $\mathbf{z}$.
Furthermore, we indicate with $\hat{\mathbf{z}} = \mathbf{h}(\mathbf{x})$ the prediction of measurement $\mathbf{z}$, which is a function of the state. Given this, applying the Gaussian assumption one can expand (3.4) as follows:

$$p(\mathbf{z}^{[i]}|\mathbf{x}) = \mathcal{N}(\quad \mathbf{z}^{[i]};\quad \mathbf{h}^{[i]}(\mathbf{x}),\quad \mathbf{\Sigma}^{[i]}\quad) \tag{3.5}$$

$$\propto \exp(-\underbrace{(\mathbf{h}^i(\mathbf{x}) - z^i)}_{\mathbf{e}^i(\mathbf{x})}{}^T \underbrace{\mathbf{\Sigma}_i^{-1}}_{\mathbf{\Omega}^i}(\mathbf{h}^i(\mathbf{x}) - z^i)) \tag{3.6}$$

Consequently substituting 3.6 in 3.3, 3.7 is found and applying the negative log-likelihood operation it follows 3.8 which represents the *cost function* to minimize.

$$\mathbf{x}^* = \underset{\mathbf{x}}{\operatorname{argmax}} \prod \exp(-\mathbf{e}^i(\mathbf{x})^T \mathbf{\Omega}^i \mathbf{e}^i(\mathbf{x})) \tag{3.7}$$

$$= \underset{\mathbf{x}}{\operatorname{argmin}} \sum_i \mathbf{e}^i(\mathbf{x})^T \mathbf{\Omega}^i \mathbf{e}^i(\mathbf{x}) \tag{3.8}$$

3.1.2 Linearization

Given that the $\mathbf{h}(\mathbf{x})$ function is *non-linear*, it is necessary to compute a quadratic approximation of the problem linearizing the error $\mathbf{e}^i(\mathbf{x})$ around the current guess. Thus, the linearization of the error funtion is obtained computing the first-order Taylor expansion of $\mathbf{e}^i(\mathbf{x})$ as in 3.9, where $\mathbf{J}^i$ represent the i^{th} jacobian matrix of the error funtion.

$$\mathbf{e}^i(\mathbf{x}^* + \Delta\mathbf{x}) \simeq \quad \underbrace{\mathbf{e}^i(\mathbf{x}^*)}_{\mathbf{e}^i} + \underbrace{\left.\frac{\partial \mathbf{e}^i(\mathbf{x})}{\partial \mathbf{x}}\right|_{\mathbf{x}=\mathbf{x}^*}}_{\mathbf{J}^i} \Delta\mathbf{x} \tag{3.9}$$

Therefore the perturbation on the *cost function* term gives 3.11 and substituting this result in 3.8, 3.12 is obtained.

$$\mathbf{e}^i(\mathbf{x}^* + \Delta\mathbf{x})^t\mathbf{\Omega}^i\mathbf{e}^i(\mathbf{x}^* + \Delta\mathbf{x}) \simeq \Delta\mathbf{x}^T\mathbf{J}^{iT}\mathbf{\Omega}^i\mathbf{J}^i\Delta\mathbf{x} + 2\mathbf{J}^i\mathbf{\Omega}^i\mathbf{e}^i\Delta\mathbf{x} + \mathbf{e}^{iT}\mathbf{\Omega}^i\mathbf{e}^i \tag{3.10}$$

$$\simeq \Delta\mathbf{x}^T\mathbf{H}^i\Delta\mathbf{x} + 2\mathbf{b}^{iT}\Delta\mathbf{x} + \mathbf{c}^i \tag{3.11}$$

$$\Delta\mathbf{x}^* = \underset{\Delta\mathbf{x}}{\operatorname{argmin}}\, \Delta\mathbf{x}^T\mathbf{H}\Delta\mathbf{x} + 2\mathbf{b}^T\Delta\mathbf{x} + \mathbf{c} \tag{3.12}$$

In order to solve this convex optimization problem, we compute the derivative of 3.12 with respect to $\mathbf{\Delta x}$ and we equal it to 0, as reported in 3.13.

$$0 = \frac{\partial[\Delta\mathbf{x}^T\mathbf{H}\Delta\mathbf{x} + 2\mathbf{b}^T\Delta\mathbf{x} + \mathbf{c}]}{\partial\Delta\mathbf{x}} \tag{3.13}$$

$$-\mathbf{b} = \mathbf{H}\Delta\mathbf{x} \tag{3.14}$$

$$\Delta\mathbf{x} = -\mathbf{H}^{-1}\mathbf{b} \tag{3.15}$$

Solving for $\Delta\mathbf{x}$ the optimal perturbation to apply to the state 3.16 would be found.

$$\mathbf{x}^* \longleftarrow \mathbf{x}^* + \Delta\mathbf{x} \tag{3.16}$$

Besides, since the measurement function is *non-linear*, there is the need to an iterative procedure: the Gauss-Newton (GN) algorithm as it is described in Algorithm 1.

The convergence of GN is not guaranteed, it depends on three main factors:

- the closeness of the initial guess from the solution
- the likelihood of encountering singularities in the optimization
- the smoothness of the error functions

At last is crytical to have a matrix $\mathbf{H}$ with full rank. It it is not the case, other ILS algorithms - like LM (*Levenberg-Marquardt*) can be used to overcome this issue.

3.1.3 Working on manifolds

The gaussian assumption holds if the state is Euclidean. Still, this is rarely the case in SLAM, because angular quantities are necessary to descibe the robot movements. Therefore the state usually belongs to a *smooth manifold.* which is a space that locally can be considered *Euclidean* and can be defined using a local parametrization in $\mathbb{R}^n$. Under those circumstances, if the state includes non-Euclidean quantities, then it is

Algorithm 1 Standard Gauss-Newton minimization algorithm

Require: Initial guess $\breve{\mathbf{x}}$; a set of measurements $\mathcal{C} = \{\langle \mathbf{z}_k, \Omega_k \rangle\}$
Ensure: Optimal solution $\mathbf{x}^\star$

$F_{new} \leftarrow \breve{F}$ ▷ compute the current error
while $\breve{F} - F_{new} > \epsilon$ **do**
  $\breve{F} \leftarrow F_{new}$
  $\mathbf{b} \leftarrow 0$
  $\mathbf{H} \leftarrow 0$
  for all $k \in \{1, ..., K\}$ **do**
    $\hat{\mathbf{z}}_k \leftarrow h_k(\breve{\mathbf{x}})$ ▷ compute prediction
    $\mathbf{e}_k \leftarrow \hat{\mathbf{z}}_k - \mathbf{z}_k$ ▷ compute error
    $\mathbf{J}_k \leftarrow \left.\frac{\partial h_k(\mathbf{x})}{\partial \mathbf{x}}\right|_{\mathbf{x}=\breve{\mathbf{x}}}$ ▷ compute Jacobian
    $\mathbf{H}_k \leftarrow \mathbf{J}_k^T \Omega_k \mathbf{J}_k$ ▷ contribution of $\mathbf{z}_k$ in $\mathbf{H}$
    $\mathbf{b}_k \leftarrow \mathbf{J}_k^T \Omega_k \mathbf{e}_k$ ▷ contribution of $\mathbf{z}_k$ in $\mathbf{b}$
    $\mathbf{H}$ += $\mathbf{H}_k$ ▷ Accumulate contributions in $\mathbf{H}$
    $\mathbf{b}$ += $\mathbf{b}_k$ ▷ Accumulate contributions in $\mathbf{b}$
  end for
  $\Delta\mathbf{x} \leftarrow solve(\mathbf{H}\Delta\mathbf{x} = -\mathbf{b})$ ▷ Solve linear system
  $\breve{\mathbf{x}}$ += $\Delta\mathbf{x}$ ▷ Apply increment
  $F_{new} \leftarrow F(\breve{\mathbf{x}})$ ▷ Update the error
end while
return $\breve{\mathbf{x}}$

necessary to use an extended parametrization on the *manifold* and a minimal one which resembles an *Euclidean* space. Then the minimal can be used to represent increments since they are *locally* defined operations.

An operation carried on a *manifold* is bounded to give a result which does not break the continuity with the respect to the operands. This is always the case if the operands belong to Euclidean spaces, then $\mathbb{R} + \mathbb{R} \to \mathbb{R} \equiv \mathbb{R} \boxplus \mathbb{R} \to \mathbb{R}$.
On the contrary, if the operands are angular quantities:

$$\begin{aligned} &\alpha, \beta, \psi \in SO(2) \\ &\alpha + \beta \to \psi \neq \alpha \boxplus \beta \to \psi \\ &\alpha - \beta \to \psi \neq \alpha \boxminus \beta \to \psi \end{aligned}$$

The symbol $\boxminus$ and similarly $\boxplus$ stands for the equivalent $-$ and $+$ binary operators with the difference that they are executed in the *manifold* space. The *Least-Squares* algorithm takes advantage of a perturbation of the state $\mathbf{x} \boxplus \Delta\mathbf{x}$ which, with a correct definition of the $\boxplus$ operator, will be carried on a manifold. Similarly, if the measurements coming from the sensors are *non-Euclidean* a correctly defined $\boxminus$ operator allows to build a well-defined error function as in 3.17 .
Finally, the update of the GN solution is now carried on the manifold, then 3.16 becomes 3.18.

$$\mathbf{e}(x) = \mathbf{h}(x) \boxminus \mathbf{z} \tag{3.17}$$

$$\mathbf{x}^* \longleftarrow \mathbf{x}^* \boxplus \Delta\mathbf{x} \tag{3.18}$$

3.2 Data Association

Through data association it is possible to find the correspondences between observations and the state variables that causes them. In particular is possible to identify which portion of the environment is responsible for a certain measurement. This knowledge is fundamental to build the graph.
The procedure that finds correspondences should be able to avoid the identification of *false-positives*. These represent outliers of the optimization and, thus, the more they are, the worst the solution computed will be. Therefore the data association requires a structured method to be able to distinguish between landmarks extracted from the environment. Usually the environment in which the robot navigates contains objects that could generate ubiquitous observations:

- indoor environments : corridors, walls, rooms.
- urban outdoor environments : trees, buildings, poles.

The ubiquitousness is caused by the fact that the robot observes them from different poses, and gather sets of measurements that represent the overall shape of this artifacts. However, some intrinsic characteristics of the object might remain unchanged even if observed from two very different poses.
A robust association procedure, independently from the sensor used and from the processing executed on the raw data, should be able to find couples of measurements coming from the same artifact. This could be accomplished by choosing an effective parametrization for the features extracted and using them to compute associations. If a generic description of the features - called descriptor - carries enough information about the portion of the environment that generates them, a robust algorithm such as RANSAC should be able to disambiguate the ubiquitous cases.
Features could be addressed here as sets of landmarks $\mathbf{L}_t$, $\mathbf{L}_{t+1}$ without loss of generality. In practice carrying out data association means that a set of correspondences $\mathbf{C}$ bound features coming from different robot poses to belong to the same environment portion.

$$\mathbf{L}_t : \{\, l_1^t, \dots\, l_J^t \,\}, \quad \mathbf{L}_{t+1} : \{\, l_1^{t+1}, \dots\, l_K^{t+1} \,\}$$
$$\mathbf{C} : \{\, c_1, \dots\, c_N \,\} \qquad c_n \rightarrow \langle l_j^t, l_k^{t+1} \rangle$$

Despite the good performances, RANSAC becomes slow when dealing with large amount of data. This can represent a problem in Robotics applications, which often have to run in real-time. Different ways of computing associations are present in literature, depending on the type of landmark, on how its descriptor is formalized and on the other context of the problem in general. One of the mostly used one is on a tree-based data structure that guarantees fast and accurate performances.

3.2.1 KD-Tree

A *KD-Tree* stands for *k-dimensional binary search tree* (BST) and is a structure that stores *k-dimensional* points. The points are inserted in the tree following a space separation logic that takes advantage of the Gaussian distribution covariance matrix. In fact, at each fork the tree is dividing the *k-dimensional* space in two parts along the principal direction of the covariance.
A KD-Tree full of $\mathbf{n}$ points is useful because, given a *query* point, it is able to find the point nearest to it among all the $\mathbf{n}$ points. This operation will have a computational complexity smaller of $\mathbf{O}(\mathbf{n})$ since the query algorithm will travel directly to the leaf of the three containing the wanted point - namely $\mathbf{O}(\log n)$. Therefore a KD-tree allows to reach real-time preformances in the case of huge quantities of points or in the case of points of high dimensionality.

3.3 ICP

ICP is the acronym for *Iterative Closest Point*. This algorithm is used to recover the transformation that relates two clouds of points. Given that the correspondences

are known thanks to a well-defined data association, the clouds can be seen as a single costellation of constraints.
In practice, iteratively each constraint is explicited in an equation and stacked in a matrix. A constraint bounds points of different clouds. Namely, it is the distance between a point projected with the transformation we need to recover and the corresponding points in the other cloud.
The more the wanted transformation is similar to the real transformation the smaller are the distances expressed in the constraints. The resulting linear system of equations, assuming it is not *under-determined*, could be inverted. Then the optimal transformation between the clouds is recovered.

3.4 LIDAR odometry

A *LIDAR*, which stands for *Laser Image Detection and Ranging*, is a sensor built with arrays of laser range-finders. It perceives the surrounding environment in the form of a cloud of *depth-based* ranges. This clouds of ranges, with the due conversions, become clouds of 2D or 3D points.
One can exploit the point-cloud generated by the sensor to estimate the trajectory of the robot. For example using the registration procedure exposed in Sec. 3.3. This procedure is known as LIDAR Odometry.
In the 2D case, simple registration approaches will suffice. For example ICP and its variants like NICP [17] is enough to deliver an accurate motion estimation in real-time. Conversely, in the 3D case, the amount of data coming from the sensor is overwhelming for the processing required in these algorithms. Thus, it is necessary to abandon the *point-to-point* approaches and to explore solutions that extract features from the point-clouds. In this work we address the LiDAR Odometry extracting high-level features from the cloud and using them to compute the relative motion between subsequent scans. The entire procedure of extraction and registration of these features is reported in Sec. 4.5.

3.4.1 Graph-based SLAM

Until now it has been discussed the odometry estimation only considering information coming from two consecutive scans. This approach generally results in a drift of the solution away from the real trajectory. Since small errors, due to a not fully recovered transformation or noise in the measurements, multiply in magnitude at each subsequent iteration.
As is explained in [18] the optimization problem should include also aggregate information coming the sensors in a previous time window. These additional information are organized in a graph where nodes correspond to poses of the robot at different points in time and edges represent constraints between the poses. In the end the graph is subjected to a Least-Squares optimization procedure as explained above to find the spatial configuration of poses that minimizes the errors on the constraints imposed by the graph edges.

Chapter 4

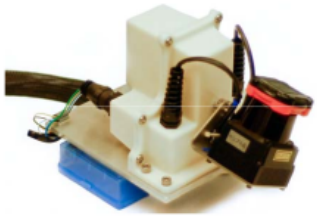
System Setup

In this Chapter we examine in depth each component of the proposed system:

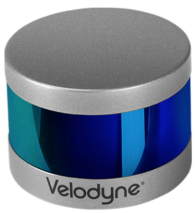
- In Sec. 4.1: the type of sensor adopted.
- In Sec. 4.2: the dataset used.
- In Sec. 4.3: a way to organise measurements data efficiently.
- In Sec. 4.4: an algorithm to clean the data from redundant information.
- In Sec. 4.5: the operations needed to extract the features.
- In Sec. 4.6: the necessary steps for LIDAR Tracking.

4.1 Laser Sensors

Different types of laser sensors have been used in SLAM. All of them are based on an array of high precision laser range-finders that give measure of the distance between the sensor and the point where the laser beam rebounds. Consequently, the ranges can be aggregated and after a full rotation of the array of scanners a complete range based image of the environment could be reconstructed. For example the laser sensor used by [11] is made of a single array of 40 laser scanners with a 180° field of view, fig.4.1A. The sensor is mounted on a servomotor, that rotates around the z axis of the LIDAR, creating a *3-dimensional* scan of the environment.



(A) Laser sensor used in LOAM



(B) 360° Velodine Laserscanner

Figure 4.1: Laser sensors

This sensor allows a rough perception of a 3D environment with a field of view of 180°. In this work, the sensor used is more expensive and much more advanced, a 360° Velodyne Laser scanner, fig.4.1B. Conversely, from the previous, this sensor is composed of an array of 64 laser scanners that is fixed in a vertical orientation. The array, internally rotates at high speed around the vertical axis, generating a much denser *band* of measures. Thus, covering a field of view of 360°.

4.2 Dataset Composition

The data used are coming from an open sourced group of datasets, namely the KITTI Vision Benchmark Suite [19]. After the extraction of the ranges between the sensor and the surfaces of the environment, each range is converted in a three-dimensional coordinate. The point-cloud obtained preprocessing a single sensor reading is illustrated in fig.4.2.

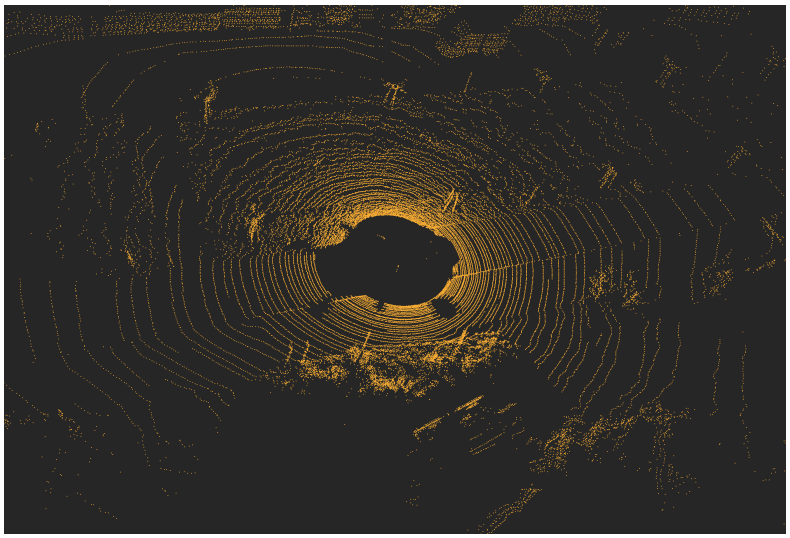


Figure 4.2: Laser scan of KITTI outdoor dataset

Still, since the robot is moving at a considerable speed while acquiring data, the sensor readings will be affected by this motion. More in detail, since the reference frame of the sensor changes during the acquisition of a single 360-degree scan, the acquired data will be distorted and present artifacts due to the motion. In [11] is proposed a procedure to recover this distortion.
Still, if the frame-rate of the sensor is high enough and the motion of the robot is relatively slow, this distortion might be neglected. Thanks to the high frame-rate of the sensor used in this thesis, this undistortion step can be avoided and, thus, the input of our system is simply the raw LiDAR measurement.

4.3 Spherical Conversion

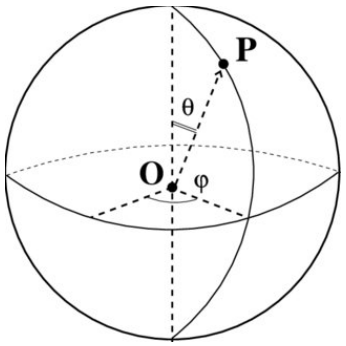


Figure 4.3: Spherical coordinates reference system

Since the cloud is coming from an high precision sensor it can contain a considerable amount of points. Hence, a spatially ordered reorganization of the cloud could improve the performances. The redistribution considered is spherically shaped, as in [20]. This allows the points of the cloud to be described in spherical coordinates 4.3.
For this reason the entire cloud is projected in spherical coordinates $\{ \psi, \theta, \mathbf{r} \}$: where ψ is the azimuth, θ is the elevation and $\mathbf{r}$ is the distance from the origin $\mathbf{O}$ of the Cartesian point $\mathbf{P}$.

$$\psi = \text{atan2}(P_y, P_x) \tag{4.1}$$

$$\theta = \text{atan2}(P_x^2 + P_y^2, P_z) \tag{4.2}$$

$$\mathbf{r} = \sqrt{P_x^2 + P_y^2 + P_y^2} \tag{4.3}$$

During this projection, it could be possible that, some points could lay in similar azimuthal and elevational coordinates. If this is the case, then only the nearest point to the sensor is kept and all the others will be discarded.
This heuristic and the choice of the desired resolution of the sphere in terms of number of rows and columns will determine the result of the unprojection of the cloud.
In particular the result will be a matrix of points in which each point has been inserted in a determinated row and column according to its elevation and azimuth.
Consequently, this spatially ordered structure will be exploited. For example finding all the points near a given one will have linear cost. In fact, it will be enough to take the points with matrix coordinates that are adjacent to the coordinates of the desired point.

4.4 Flat-surface removal

In order to reduce the overall complexity of the following computations we removed the uninformative part of the cloud. This is the case for the portions of the environment which do not present relevant shapes or easy-to-detect edges - e.g. floor, ceiling and all the surfaces that lack of verticality. We exploited the algorithm presented in [20] to

perform to efficiently identify and remove uninformative flat surfaces.
In practice for every point will be considered a circle on the x-y plane. Then will be counted the number of points which vertical projection falls inside that circle. Finally, if number of counted points is smaller than a threshold the point will be discarded.

4.5 Features extraction

Our approach, instead of processing raw points coming from sensor reading, extracts high level features from the point-could, namely lines and planes. In this way we have a more compact representation of the environment, reducing the amount of data that the system has to process. This allows dealing with a smaller optimization problem and also tries to generalize the environment information. Therefore trying to reduce te impact of the noise coming from the sensor measurements.
In [20] is presented a set of LIDAR features which are points, lines and planes. Instead as proposed by Nardi et al. in [21], we will characterize the extracted features as *degenerate quadrics*. Thanks to this formalization we can represent multiple geometric primitives as a unique object. In order to extract lines and planes from a raw point-cloud, we perform normal-based clustering. In the next section, this process is analyzed more in detail.

4.5.1 Normal Computation

As first step, we need to compute the normal for each point of the point-cloud. Hence, for each point we take the group of points that are near it in the *spherical image* (the structure shown in the spherical conversion section) and compute the mean μ_i and covariance $\mathbf{\Sigma}_i$ of each group. Consequently, applying an SVD (*singular value decomposition*) to $\mathbf{\Sigma}_i$ we compute the eigenvalues λ_i and the associated eigenvectors $\mathbf{e}_i$. Finally, the normal for that point will be the eigenvector associated to the smallest eigenvalue of $\mathbf{\Sigma}_i$.
The idea is that points of the environment that belongs to the same artifacts (eg. walls, edges, poles) have a similar normal vector.

4.5.2 Clustering

In this part is explained the strategy used to clusterize the point cloud. Normally the normals are affected by noise. Therefore, before the cluster assignment, we apply a smoothing procedure to the normals of the points. In practice the smoothed normal vectors are the result of an average of their normals and their neighbors normals.
Then the points which are adjacent on the *spherical image* and have a similar normal direction will be assigned to the same cluster (fig.4.4).

Finally, features will be created from each cluster that contains at least a certain number of points. In this work the features $f_i \in \mathbf{F}$ are lines and planes. Every feature f_i is characterized by an *origin* vector μ_i and from a *direction* vector ν_i.

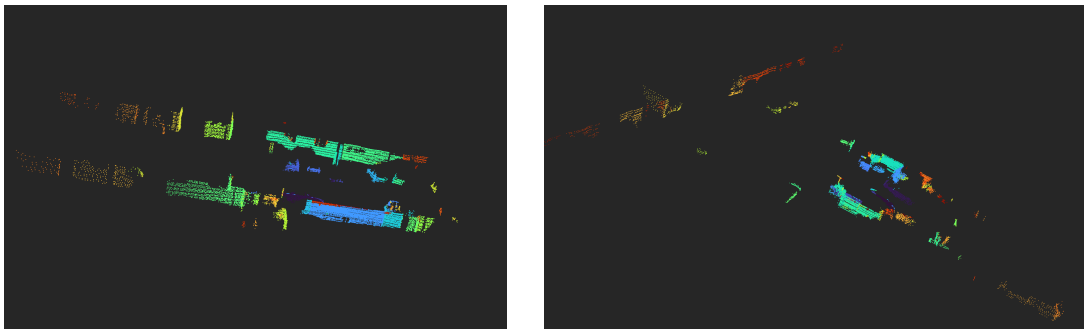


Figure 4.4: 3D representation of clusters

- **lines** *direction* is the eigenvector e corresponding to the greatest eigenvalue λ_{max} of the associated Gaussian distribution. The reason behind this choice is that the line is spatially distributed along a single direction. Therefore the two smaller eigenvalues will be close to zero.
- **planes** *direction* instead is the eigenvector e corresponding to the smallest eigenvalue λ_{min} in order to avoid ubiquitous cases. In fact the other two eigenvalues will be similar in magnitude since the plane extends spatially along two main directions. Thus, a *unique* characterization should be dependant only on the eigenvector associated with λ_{min}.

4.5.3 Eigenvalue-based characteristics

There are structured information that can be inferred by the relationship between eigenvalues λ.
The covariance $\mathbf{\Sigma}$ is a 3x3 matrix, its eigenvalues $\lambda = \{ \lambda_1, \lambda_2, \lambda 3 \}$ will be again computed through SVD. Then as in 4.4 the sum of the eigenvalues Σ_λ is computed.

$$\Sigma_\lambda = \lambda_1 + \lambda_2 + \lambda_3 \tag{4.4}$$

The ξ vector represents the eigenvalues λ normalized, 4.5.

$$\xi_i = \frac{\lambda_i}{\Sigma_\lambda}, \quad for\ i \in \{ 1, 2, 3 \} \tag{4.5}$$

As it has been done in [22] each feature has eight different traits:

$$\begin{aligned}
\textbf{Linearity:} \quad & \mathbf{L}_\lambda = \frac{\xi_1 - \xi_2}{\xi_1} \\
\textbf{Planarity:} \quad & \mathbf{P}_\lambda = \frac{\xi_2 - \xi_3}{\xi_1} \\
\textbf{Scattering:} \quad & \mathbf{S}_\lambda = \frac{\xi_3}{\xi_1} \\
\textbf{Omnivariance:} \quad & \mathbf{O}_\lambda = \sqrt[3]{\xi_1\ \xi_2\ \xi_3} \\
\textbf{Anisotropy:} \quad & \mathbf{A}_\lambda = \frac{\xi_1 - \xi_3}{\xi_1} \\
\textbf{Eigenentropy:} \quad & \mathbf{E}_\lambda = -\sum_{i=1}^{3} \xi_i \ln \xi_i \\
\textbf{Sum of eigenvalues:} \quad & \Sigma_\lambda = \lambda_1 + \lambda_2 + \lambda_3 \\
\textbf{Change of curvature:} \quad & \mathbf{C}_\lambda = \frac{\xi_3}{\xi_1 + \xi_2 + \xi_3}
\end{aligned}$$

We have found that two of these distinctive traits, the scattering and the change of curvature, are consistent indicator of quality of the feature. Thus, each feature that does not respect two upper bounds for $\mathbf{S}_\lambda$ and $\mathbf{C}_\lambda$, will be discarded.

4.6 Motion tracking

In this section we present how the features extracted are used to give an estimation of the *ego-motion* of the robot. In the following steps a *motion-model* has been adopted in order to:

- help the *correspondences finder* with a prior.
- have a good initial guess for the iterative algorithms of the *alignment*.

Finally, we will explain how the *tracking* phase generates the SLAM solution: the estimated robot trajectory and the map structure.

4.6.1 Motion model

A motion model allows to have a rough estimate of the displacement of the robot between two subsequent sensor readings. This displacement can be used as initial guess to perform ICP, enhancing the convergence properties of the process. The displacement of the robot can be described by a transformation $\mathbf{T} = [\mathbf{R}\ \mathbf{t}]$.
In a 3D environment $\mathbf{T}$ belongs to SE(3) space and embeds a translation on the Cartesian axes and a rotation around three angles θ, ϕ, ψ. In this work, we use the constant velocity assumption to generate a guess of the new pose of the robot useful for the following steps.

After the optimization this guess will be substituted with a more precise $\mathbf{T}_t$ that will be used to generate the next guess.

4.6.2 Correspondence finder

The features f_i extracted from point clouds will be used to align two subsequent clouds through an ICP-inspired process. The correspondence finder is a module which will generate a series of c_i correspondences. Each correspondence bounds two features belonging to different clouds $c_i \rightarrow \left\langle f_j^t, f_k^{t+1} \right\rangle$. In order to use a KD-Tree, we should be able to parametrize our matchable features into k-dimensional Euclidean vectors.
Therefore, assuming ν_i as the vector that describes the main direction of the feature f_i.

- A line is parametrized by a vector: $\mathbf{l}_i$:

$$\begin{aligned} \mathbf{l}_i &\in \mathbb{R}^6 \\ \mathbf{l}_i &= \begin{bmatrix} \mathbf{c}_i \\ \nu_i \end{bmatrix} \\ \mathbf{c}_i &= \nu_i \times \mu_i \end{aligned} \tag{4.6}$$

- A plane is parametrized by a vector: $\mathbf{p}_i$:

$$\begin{aligned} \mathbf{p}_i &\in \mathbb{R}^4 \\ \mathbf{p}_i &= \begin{bmatrix} \nu_i \\ d_i \end{bmatrix} \\ d_i &= \nu_i \cdot \mu_i \end{aligned} \tag{4.7}$$

The 4.6 and 4.7 are quantities that describe the *skewedness* of the feature with the respect of the sensor.
In the search for corrispondences, lines $\mathbf{l}_i$ and planes $\mathbf{p}_i$, coming from the cloud $\mathbf{P}_{t-1}$ will be inserted in two different KD-Trees. Then the vector $\mathbf{c}_t$ of correspondences will be filled with the results coming from the queries to the KD-trees using features coming from $\mathbf{P}_t$.

4.6.3 Aligner

In the previous section we explained how we find correspondences c_i between features. Now this correspondences will be used to build an optimization problem and find the transformation between two laser scans. Every correspondence is translated in a constraint called a factor. Then this factors are used to build a factor-graph. This graph represents a Least-Squares problem as shown in Sec. 3.1.
We use the guess from the motion model to initialize the system and an iterative optimization of the error is executed until convergence is reached.
The result of the optimization is a transformation $\mathbf{T}_t$ that minimizes the error in the factor-graph.

4.6.4 Tracker

The traking procedure is responsible to aggregate results coming from the aligner. Therefore it updates the graph with the newly computed edges and nodes.
Inside the graph is contained the entire robot trajectory. In addition, the features extracted at each iteration can be projected in a global map, using the information inside the edges of the graphs. As a result a map of features can be generated.
The advantages in creating a global map of features are:

- the redundant features information can be removed.
- each feature in a global map will be less affected by sensor noise because is likely that it has been observed from multiple robot poses.
- it allows executing offline optimization procedures to recover the drifting error.

Chapter 5

Results

In this chapther we will present the results incrementally:

- In Sec. 5.1 three examples of flat surfaces removal.
- In Sec. 5.2 a 2D and a 3D visualization of the normals directions.
- In Sec. 5.3 a 2D multi level representation of the environment and a 3D clusterization example.
- In Sec. 5.4 an example with a synthetic dataset and another with the KITTI dataset.
- In Sec. 5.5 we will show the correspondences between features of different scans.
- In Sec. 5.6 there are two 3D images of the robot trajectory after the tracking.

5.1 Flat surface removal experiment

In the following it is shown the result of the flat surface removal algorithm of Sec. 4.4. In fig.5.1, fig.5.2 and fig.5.3 is shown on the left side the scan of the environment, and on the right the remaining points after the flat surfaces removal.

5.2 Normal computation experiment

According to Sec. 4.5.1 in fig.5.4 in the upper part there is the projection of the ranges stored in the spherical image (Sec. 4.3) and in the lower part there are the corresponding normals encoded in RGB colors. Points lying on the same wall have similar color.
Then in fig.5.5: on the left side there is the raw point cloud. On the right side there are the normal vectors (starting in green and degrading to yellow) of the points remaining after both the flat surfaces removal and the normals computation.

5.3 Clustering experiment

The clustering procedure (Sec. 4.5.2) applied on the KITTI dataset, is shown in fig.5.6. In particular the third 2D image, the *path image*, indicates which points might

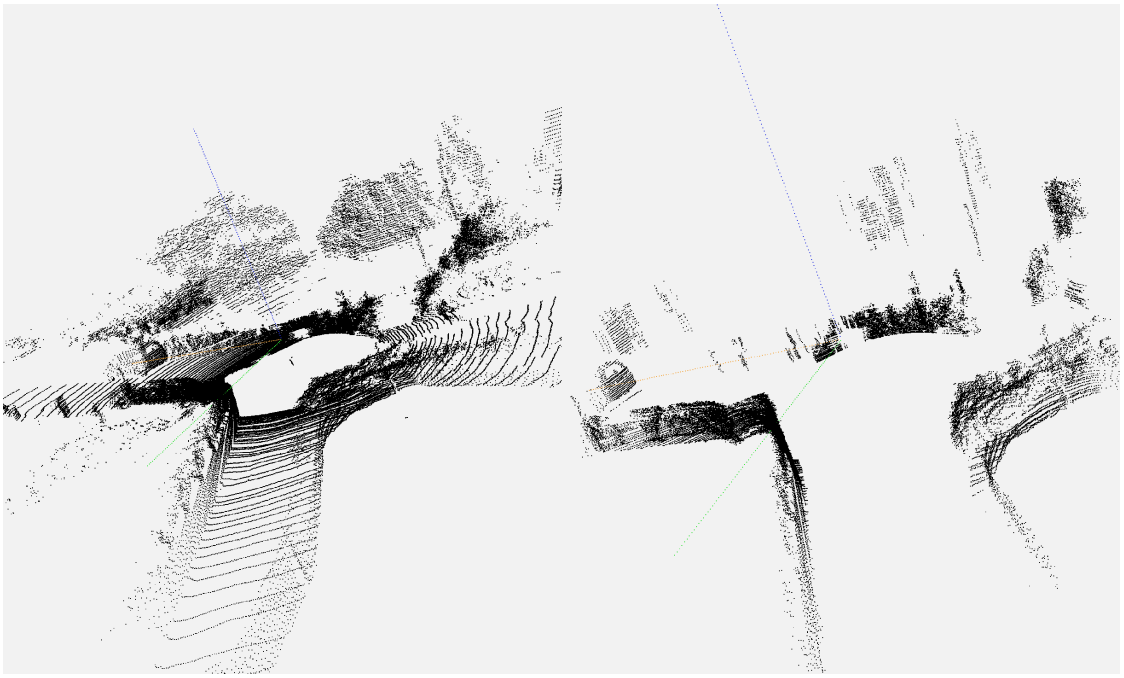


Figure 5.1: Flat surfaces removal in the first section (KITTI dataset outdoor)

Figure 5.2: Flat surfaces removal in the second section (KITTI dataset outdoor)

belongs to the same cluster.
The fourth, the *blurred normals image*, shows the smoothing process to remove the noise

Figure 5.3: Flat surfaces removal in the third section (KITTI dataset outdoor)

Figure 5.4: Scan points and their normals projected in 2d

from the normals.
Finally, in the last image, the clusters of points are distinguishable by their different colors.

Then in the fig.5.7 we can see the final results of the clustering projected on the 3D map. Again each cluster is distinguished by a different color.

5.4 Features extraction experiment

At first in fig.5.8 we use a synthetic dataset made of six planes to verify that these features are extracted properly (Sec. 4.5).

Then in fig.5.9 on the left there are the points already clustered and on the right there are the corresponding generated lines and planes.

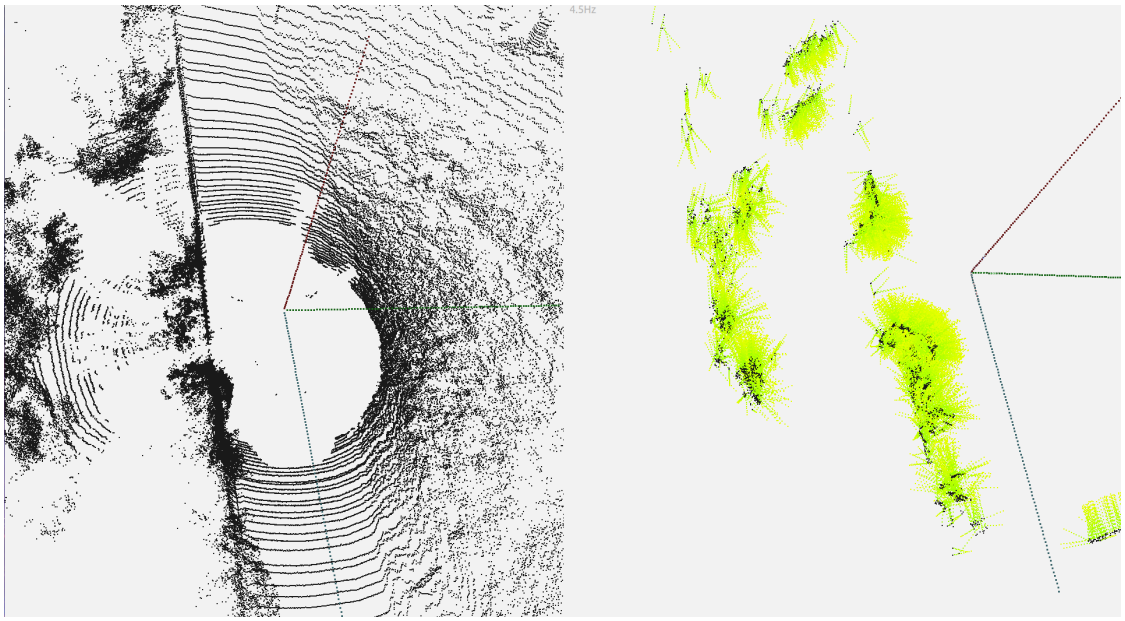


Figure 5.5: Normals directions: vectors starting as green and ending as yellow

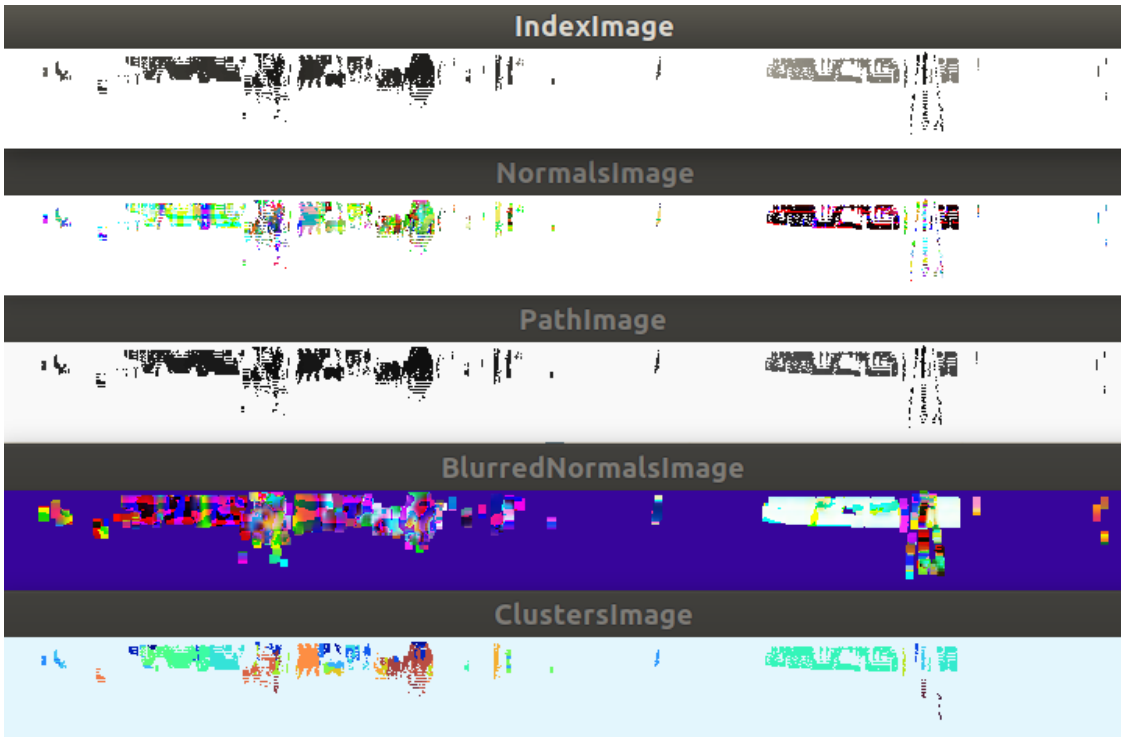


Figure 5.6: Five 2D images that show the preprocessing steps applied to the dataset

5.5 Correspondence finder

In this section we show the result of the correspondence finder module (Sec. 4.6.2). In particular in fig.5.10, from left to right, we have the features $\mathbf{F}_{t-1}$, the features $\mathbf{F}_t$ and

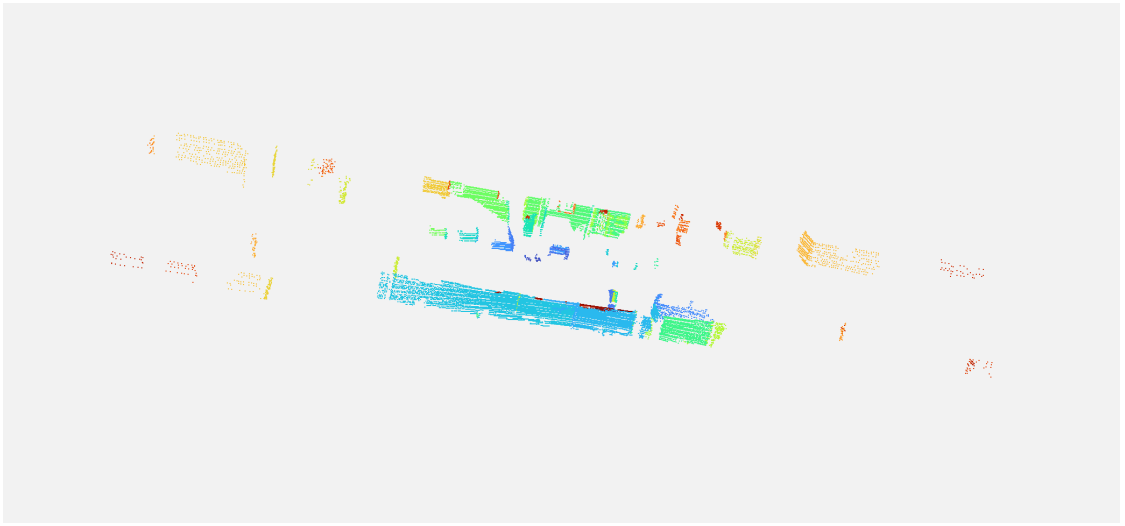


Figure 5.7: Clusters of the KITTI data-sequence projected in 3D

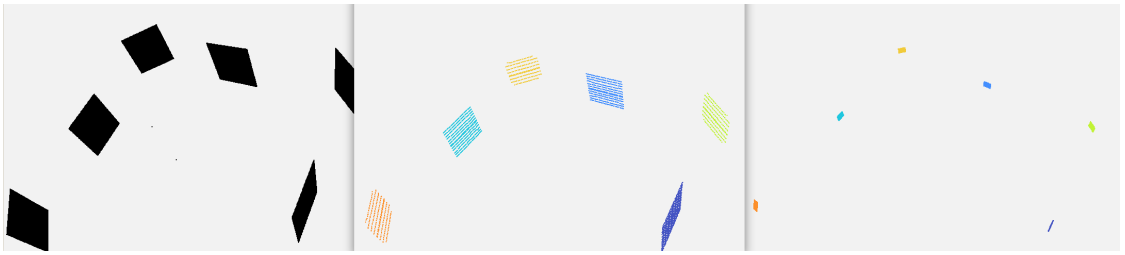


Figure 5.8: Feature extraction of six planes from syntetic data

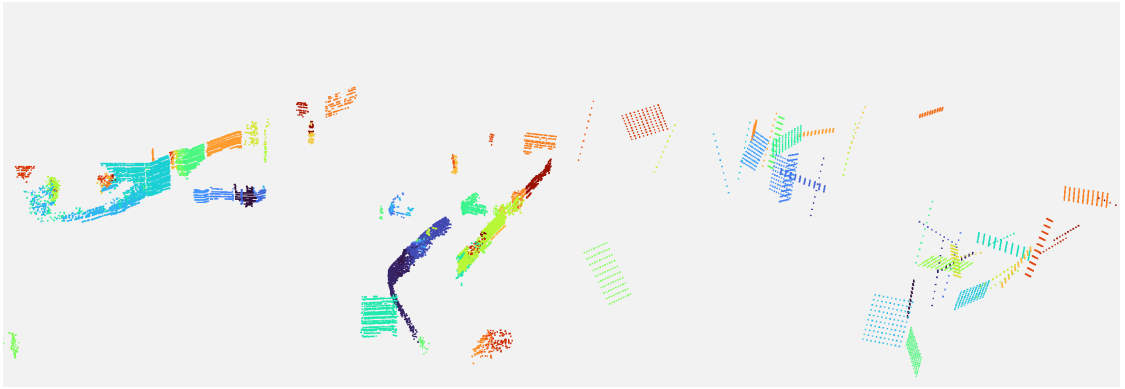


Figure 5.9: Feature extraction from KITTI dataset: on the left clusters, onthe right features

at last the associations found. In the third quadrant the red features belong to $\mathbf{F}_{t-1}$, and the blue features belong to $\mathbf{F}_t$. The black segments indicate the associations, connecting the centers of associated features.

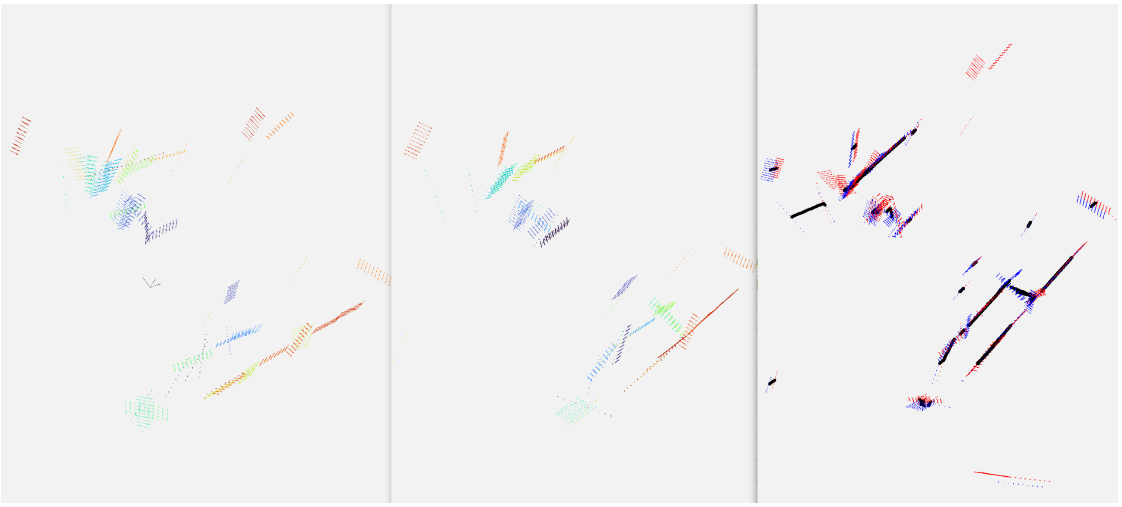


Figure 5.10: Left: previous features, center: current features, right: features correspondences (black segments)

5.6 Full traking experiment

Here we show the results of the full-tracking (Sec. 4.6.4) on the KITTI data-sequence: in the fig.5.11 we can see the global map of the environment and the trajectory of the robot (small black squares).

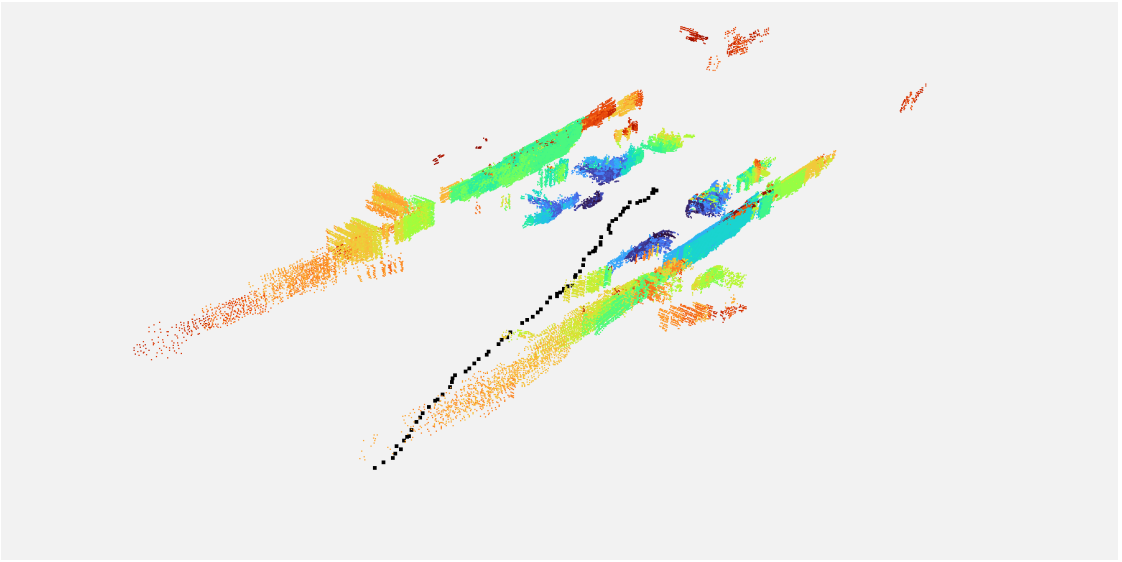


Figure 5.11: Full tracking of the initial section of the KITTI sequence

Chapter 6

Discussion

In conclusion, in this Master's thesis, it has been reached the goal of creating a LIDAR Odometry and Tracking pipeline. The implementation of each section of the project has allowed me to understand deeply the nature of the SLAM problem. The system developed is written in C++ and it is open-source.
In the end the system manages to extract high level features from the laser measurements and tracks successfully the trajectory of the robot.

There are several possible enhancement for this work:

- Introduce offline optimization procedures that works on batches of laser scans in order to recover the drifting error.
- The addiction of the loop closure step in the pipeline in order to recognize the environment sections already visited from the robot.
- Parallelization of the some preprocessing operations that are conducted on each data point.
- The introduction of more flexible high level primitives like for example curved surfaces. That will allow a better characterization of the artifacts in the environment.

Bibliography

[1] Durrant-Whyte, H., Bailey, T.: Simultaneous localization and mapping: part i. IEEE Robotics Automation Magazine **13**(2) (June 2006) 99–110

[2] Bailey, T., Durrant-Whyte, H.: Simultaneous localization and mapping (slam): part ii. IEEE Robotics Automation Magazine **13**(3) (Sep. 2006) 108–117

[3] Grisetti, G., Stachniss, C., Burgard, W., et al.: Improved techniques for grid mapping with rao-blackwellized particle filters. IEEE transactions on Robotics **23**(1) (2007) 34

[4] Dissanayake, M.G., Newman, P., Clark, S., Durrant-Whyte, H.F., Csorba, M.: A solution to the simultaneous localization and map building (slam) problem. IEEE Transactions on robotics and automation **17**(3) (2001) 229–241

[5] Davison, A.J., Murray, D.W.: Simultaneous localization and map-building using active vision. IEEE Transactions on Pattern Analysis and Machine Intelligence **24**(7) (July 2002) 865–880

[6] Se, S., Lowe, D., Little, J.: Mobile robot localization and mapping with uncertainty using scale-invariant visual landmarks. The International Journal of Robotics Research **21**(8) (2002) 735–758

[7] Montemerlo, M., Thrun, S., Koller, D., Wegbreit, B., et al.: Fastslam: A factored solution to the simultaneous localization and mapping problem. Aaai/iaai **593598** (2002)

[8] Montemerlo, M., Thrun, S., Koller, D., Wegbreit, B., et al.: Fastslam 2.0: An improved particle filtering algorithm for simultaneous localization and mapping that provably converges. In: IJCAI. (2003) 1151–1156

[9] Castellanos, J.A., Neira, J., Tardós, J.D.: Limits to the consistency of ekf-based slam. IFAC Proceedings Volumes **37**(8) (2004) 716–721

[10] Lu, F., Milios, E.: Globally consistent range scan alignment for environment mapping. Autonomous Robots **4**(4) (Oct 1997) 333–349

[11] Zhang, J., Singh, S.: Loam: Lidar odometry and mapping in real-time. In: Robotics: Science and Systems. Volume 2. (2014) 9

[12] Zhang, J., Singh, S.: Visual-lidar odometry and mapping: low-drift, robust, and fast. In: 2015 IEEE International Conference on Robotics and Automation (ICRA). (May 2015) 2174–2181

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

[14] Deschaud, J.E.: Imls-slam: scan-to-model matching based on 3d data. In: 2018 IEEE International Conference on Robotics and Automation (ICRA), IEEE (2018) 2480–2485

[15] Behley, J., Stachniss, C.: Efficient surfel-based slam using 3d laser range data in urban environments. In: Robotics: Science and Systems. (2018)

[16] Charnes, A., Frome, E.L., Yu, P.L.: The equivalence of generalized least squares and maximum likelihood estimates in the exponential family. Journal of the American Statistical Association **71**(353) (1976) 169–171

[17] Serafin, J., Grisetti, G.: Nicp: Dense normal based point cloud registration. In: 2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), IEEE (2015) 742–749

[18] Grisetti, G., Kummerle, R., Stachniss, C., Burgard, W.: A tutorial on graph-based slam. IEEE Intelligent Transportation Systems Magazine **2**(4) (2010) 31–43

[19] Geiger, A., Lenz, P., Stiller, C., Urtasun, R.: Vision meets robotics: The kitti dataset. International Journal of Robotics Research (IJRR) (2013)

[20] Serafin, J., Olson, E., Grisetti, G.: Fast and robust 3d feature extraction from sparse point clouds. In: 2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), IEEE (2016) 4105–4112

[21] Nardi, F., Della Corte, B., Grisetti, G.: Unified representation and registration of heterogeneous sets of geometric primitives. IEEE Robotics and Automation Letters **4**(2) (2019) 625–632

[22] Weinmann, M., Jutzi, B., Mallet, C.: Semantic 3d scene interpretation: A framework combining optimal neighborhood size selection with relevant features. ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences **2**(3) (2014) 181