SAPIENZA
UNIVERSITÀ DI ROMA

Sparse LiDAR Odometry using intensity channel: a comparison

Faculty of Information Engineering, Informatics and Statistics
Master of Science in Artificial Intelligence and Robotics

Francesco Starna
ID number 1613660

Advisors
Prof. Giorgio Grisetti
Dr. Leonardo Brizi

Academic Year 2021/2022

Sparse LiDAR Odometry using intensity channel: a comparison
Sapienza University of Rome

This thesis has been typeset by LaTeX and the Sapthesis class.

Author's email: francesco.starna@mail.com

Abstract

In the latest years, computer vision is becoming one of the most important Artificial Intelligence field due to the variety of practical applications, especially the ones concerning robot navigation systems. In these, data are provided by the use of sensors, which may be of different kinds, depending on the accuracy needed, implying different costs. Lately, LiDAR sensors became preferred to cameras, due to the three-dimensional capabilities, and therefore high estimation accuracy. Through the use of these sensors, it is possible to provide to a navigation system the capability of estimating the current position in real-time, which can be useful in many scenarios. This thesis project focuses on the use of LiDAR 3D data for estimating the odometry of a vehicle, giving attention to the different available methods in order to make a comparison.

Contents

1 Introduction **1**

2 Related Work **3**

3 Preliminaries **6**
3.1 Simultaneous Localization and Mapping 6
3.1.1 Mathematical description 7
3.1.2 Localization 7
3.1.3 Mapping 9
3.1.4 Architecture 10
3.2 Least-Squares Optimization 12
3.2.1 Problem formulation 12
3.2.2 Gauss-Newton algorithm 13
3.2.3 Iterative Closest Point 17
3.2.4 Outlier Rejection 20
3.3 Feature Extraction 23
3.3.1 Traditional approach 24
3.3.2 Machine learning approach 29

4 Pipeline **33**
4.1 Projection 33
4.2 Extraction 35
4.3 Tracking 36
4.4 Registration 37
4.5 Odometry Estimation 38

5 Experiments **40**
5.1 System 40
5.1.1 LiDAR hardware 40
5.1.2 Robot Operating System 40
5.2 Dataset 41

5.2.1 GNSS conversion 42
5.3 Results 43
5.4 Comparison 46

6 Conclusions **49**

Bibliography **51**

Chapter 1

Introduction

In recent years, the research on robot navigation systems has increased its attention on more accurate and fast methods. This is mainly due to the coming of autonomous robots, which require real-time and extremely accurate applications in order to guarantee safety and reliability. There are plenty of other fields where a navigation system is a key requirement for the application to work. Smart farming, rescuing, exploration, surveillance, military, are just some of the possible applications which use these systems. In each of these fields, the accomplishment of a task is strictly related to the capability of the robot to localize itself into the environment, build a map of its surrounding, and possibly interact with objects in it. This problem is called Simultaneous Localization and Mapping (SLAM), and its efficiency and accuracy determine the success or not of the subsequent actions. A perfect example is the autonomous car, which task is to localize itself into the road lane, identify objects and estimate their distances, and consequently varying acceleration and steering in order to avoid them, and drive safely. This complex scenario makes clear the reason for the need of accurate applications, starting from reliable and precise position tracking systems, which are called odometers.

Odometry relies on data provided by motion sensors to estimate change in position over time. Sensors may be of many different natures; sound navigation and ranging (SoNAR), monocular or stereo cameras, light detection and ranging (LiDAR) are the most used ones. Nowadays, in autonomous driving systems, LiDAR sensor is the most used, since it provides a rich three-dimensional scan of the surrounding environment which is invariant to light conditions, and therefore more efficient than cameras' systems which reliability depends on the visibility conditions. LiDAR sensors provide a dense environment representation called Point Cloud, where each one has a 3D geometric position and a light intensity measured by the laser beam.

However, despite the density of the representation, the sensor measurements may provide small errors. These are meaningless locally, but acquire significant importance when accumulated over time, leading to large displacement estimations. In order to deal with this problem, one should keep the good measurements (inliers), discard the wrong ones (outliers), and finally derive the best model possible. On the topic of outliers rejection, the research community mainly focuses on two different approaches: (1) consensus methods, which probabilistically estimate a model by iteratively select a minimum solution from the available measurements and seek for the largest set of inliers, (2) robust estimators, which are a generalization of maximum likelihood estimation (MLE), and try to minimize some cost function to bias the model in presence of outliers. Both methods have advantages and disadvantages: the former are typically slower to reach a probabilistically good estimate, but they don't need a guess assumption, while the latter are generally faster, but they should have an initial good estimate in order to converge.

This thesis project focuses on the exploitation of LiDAR information using a classical 2D approach; first the point clouds are projected into a 2D image, then feature extraction techniques are applied in order to extract a bunch of interesting points which can be detected in subsequent LiDAR beams. The points are extracted using a traditional approach, Oriented FAST and Rotated BRIEF (ORB), or a newly machine learning approach called SuperPoint. The process of finding the correspondences between interesting points of previous and next cloud is called tracking. Once the interesting points have been coupled, one can retrieve the 3D correspondences and perform registration techniques for finding the rigid body transformation between the previous and the next cloud, such as Random Sample Consensus (RANSAC) or Iterative Reweighed Least-Squares (IRLS). The transformations are then accumulated in order to estimate the trajectory travelled by the car.

The structure of the text starts with an overview of some recent works on localization and outlier rejection approaches in Chapter 2. Then, a description of SLAM system, with its mathematical derivation and components, feature extraction approaches, with a journey from traditional to machine learning one, and least-squares optimization, with the formulation and the derived algorithms, are summarized in Chapter 3, which are preliminaries concepts needed to reach the Chapter 4, where the pipeline of the process is explained, also highlighting the difference between the methods used, starting from the LiDAR projection and ending with the odometry estimation. In Chapter 5, a summary of the experiments is done, with a detailed comparison between the methods, and finally, in Chapter 6, conclusions and future developments are reported.

Chapter 2

Related Work

Most of the work on localization and mapping applications concern camera and LiDAR sensors, each of which has some advantages and disadvantages. Although camera sensors are cheap and provide rich data for feature extraction, they project the 3D space onto a 2D one, which lead to a loss of information. There are a few techniques to recover the 3D space, such as the exploitation of the Pinhole Camera Model [1], which describes the mathematical relationship between the coordinates of a point in 3D space and its projection onto the image plane. Moreover, systems based on camera sensors, have to deal with changing of light conditions and camera lens distortion, which implies an accurate camera calibration process.

Camera-based SLAM was initially solved by filtering [2]. In that approach, every frame is processed by the filter to jointly estimate the map feature locations and the camera pose. It has the drawbacks of wasting computation in processing consecutive frames with little new information and the accumulation of linearization errors. After some years, the introduction of bundle adjustment using least-squares optimization has led the way to more robust frameworks. A remarkable work on the field is ORB-SLAM [3], which deals with several tasks, such as tracking, mapping, relocalization, place recognition, and loop closure. They used parallel threads for accomplishing the different tasks, all using the same features extracted with ORB, which allows real-time performances without the use of a GPU. The novel idea of the work was the introduction of a feature-based method with respect to a direct method, which instead rely on optimization over all the image pixels. A recent direct method work is Direct Sparse Odometry (DSO) [4]. It optimizes a photometric error defined directly on the images over a window of recent frames, providing robustness to photometric distortions present. In contrast to existing direct methods, they jointly optimize for all involved parameters (camera intrinsics, camera extrinsics, and inverse depth values), performing the equivalent of a photometric bundle adjustment.

On the contrary, LiDAR sensors may be very expensive depending on the accuracy level, frame rate, range, and field of view, but they provide a dense mass of accurate 3D points which are invariant to light changes and can be exploited in SLAM systems. A LiDAR sensor uses laser beams, which can be ultraviolet, infrared, or visible light, for sampling the surrounding space in order to produce precise measurements $p_n = (x_n, y_n, z_n, i_n)$, with i the intensity value of the point. The idea behind the functioning of the sensor is to transmit laser beams to a target and analyze the return reflection. Based on the round trip time, it is possible to determine the distance of the target point from the sensor. Modern LiDAR sensors allow obtaining information about the surface characteristics of the objects hit by the laser beam. In fact, besides measuring the round trip time, it also measures the intensity of the reflected beam, from which it can derive information such as roughness and reflectivity. Many LiDAR sensors also integrate an Inertial Measurement Unit (IMU) which comprises multi axis gyroscopes and accelerometers to provide reliable position and motion recognition. Thanks to this, distortion is prevented by bringing the points measured by the LiDAR into the same reference frame, which may be affected by vibrations and external environment changes. This process is called de-skewing, and it is carried out at sensor firmware level.

Feature extraction on point clouds is not as easy as it is on camera images, because of the lack of 2D information. Moreover, it is not feasible to register all the points, because the size of a point cloud provided by a LiDAR sensor may vary from 10 to more than 100 thousands, and this will slow down the system at the cost of losing real-time performances. Therefore, approaches that attempt to decrease the size of the point clouds or methods based on features extraction have gained more attention. An example of work on this field is LiDAR Odometry and Mapping (LOAM) [5], where the scan region is divided into 4 identical sub-regions, each of which provides 2 edge points and 4 planar points. These points are selected according to a function which evaluates the roughness of the local surface for each point cloud. This method suffers in the case of uneven terrain because of wrong association between these feature points. LeGo-LOAM [6] has been developed in order to overcome this problem. With the introduction of the segmentation module the distance image is separated into multiple clusters and labels are assigned to remove the interference of noisy points.

An accurate review on LiDAR odometry and mapping techniques is reported in [7]. The authors argue that the 3D space is more challenging than the 2D one, because the amount of data is much larger, the feature matching in spatial dimension is more complex than that in a plane, and the demand for positioning accuracy is also higher.

Both camera and LiDAR based systems estimate motion based on registration techniques. According to [8], the works on these follow mainly three branches:

- **Point-based methods**. These methods extract keypoints from the LiDAR frames, establish the keypoint pairs, and estimate the transformation according to the pairs. One of the earliest approaches, introduced in [9], that is still in use for LiDAR odometry applications is the ICP method. In this branch are also included probabilistic methods such as RANSAC.

- **Distribution-based methods**. The key step in these approaches is to convert the input point cloud into equal cell sizes on which the normal distributions are computed. These are then compared with other normal distributions from other point clouds and are assigned to a score. The Normal Distribution Transform (NDT) method, introduced in [10] has been a defining type of approach to tackle the registration problem for LiDAR odometry applications. The advantage is that distributions are generated in all the occupied cells irrespective of the resolution. One of the significant NDT-based approach for odometry estimation is LOAM.

- **Network-based methods**. The network-based approaches employ neural networks in the pose estimation module of the LiDAR odometry pipeline. Approaches in the domain of LiDAR odometry using deep learning are relatively recent when compared to the point and distribution-based approaches. PointNet [11] was proposed for point cloud object classification and part segmentation tasks, and it was the pioneer work on the field.

The authors have compared several methods of the three branches, concluding that, in the future direction of research, fusion-based approaches are suggested for precise LiDAR odometry.

Chapter 3

Preliminaries

In the following chapter, some theoretical and mathematical preliminaries are explained, in order to better understand the content of the main part of the project.

3.1 Simultaneous Localization and Mapping

Simultaneous Localization and Mapping (SLAM) is the computational problem of constructing and keeping updated a map while simultaneously estimating the state of the robot in it. The state could be for instance the pose, namely position and orientation, but it may be more complex including information such as calibration parameters or velocity. SLAM algorithms use computational geometry and computer vision techniques for solving non-linear optimization problems, with the aid of single or multiple sensors which provide internal or external measurements. The latter are further elaborated to avoid noise, which may come from the sensor itself and/or the environment, and to select the most relevant for building the best estimate of the state. The robot should also have a certain *kinematic model*, which includes information about the commands sent to the low level control unit, which is responsible for actuating the motors of the system to make the robot moving into the environment. Unfortunately, using only commands and sensors is not sufficient for a correct localization and mapping, especially for long-term trajectories, since small errors which usually happen, are propagated along the trajectory causing the global error to become eventually large. To solve this problem, a technique called *bundle adjustment* is performed when a loop closure is detected, namely whenever the robot revisits a location that it already knows. A complete SLAM framework is also responsible for relocalization in case of failures.

3.1.1 Mathematical description

The goal of a SLAM system is to estimate the robot trajectory $X_t = \{\mathbf{x}_0, \mathbf{x}_1, ..., \mathbf{x}_t\}$ and the map of the environment M_t, over discrete time steps t, given a series of sensor measurements $Z_t = \{\mathbf{z}_0, \mathbf{z}_1, ..., \mathbf{z}_t\}$. The robot trajectory is the composition of consecutive poses, 4×4 matrices composed by a rotation matrix $\mathbf{R}$, which represents the orientation, and a translation vector $\mathbf{t}$, which represents the position:

$$\mathbf{x} = (\mathbf{R}|\mathbf{t}) = \begin{pmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{pmatrix}$$

The trajectory can be estimated by the odometry, which is computed by integration of a series of controls $U_t = \{\mathbf{u}_0, \mathbf{u}_1, ..., \mathbf{u}_t\}$, given the *transition model* of the robot:

$$\mathbf{x}_{t+1} = f(\mathbf{x}_t, \mathbf{u}_t)$$

The *observation model* of the robot can predict the measurement using, as prior probability, the estimated trajectory and the map obtained so far:

$$\hat{\mathbf{z}}_{t+1} = h(\mathbf{x}_t, M_t)$$

The measurements Z_t from the sensor come unlabeled, so one should associates them with the predictions $\hat{\mathbf{z}}$; this problem is called *data association*, and it is generally solved by means of heuristics and geometrical computation. Finally, having defined all the elements, the aim is to compute the following distribution:

$$P(X_t, M_t|U_t, Z_t)$$

The latter represents the probability of the trajectory and map at time t, given the sensor measurements and control inputs so far. This problem is also known as smoothing according to Markovian probability theory.

3.1.2 Localization

Localization is the problem of determining where the robot is located into the map which is simultaneously created during SLAM execution. The task is really sensitive to small errors because it accumulates them over time, and it generates consistent deviations from the actual values. In the case where the robot initial location is unknown the problem is called *global localization*, and techniques of relocalization and/or *place recognition* in the environment must be applied. However, most of the time, the problem is local, and the initial location is set to the zero reference frame of the system.

When the robot goes through already visited places, for instance in indoor environments, it often happens that same locations have a different trajectory and map, due to the errors accumulated over time. This is the case of *loop closure*, which has to be first detected, evaluated, and used to correct the error along all the trajectory and map built until that moment. There are different methods which can deal with loop closure, according to the available sensor measurements. If there is a camera, one may exploit 2D keypoints descriptors and store them in a tree-structured image database along with a weight according to its relevance; this is called distributed *Bag of Words* (DBoW), which is a technique usually exploited in natural language processing tasks, that was originally implemented in [12] and improved several times thereafter. Another similar method which instead uses a binary search tree structure, improved the performances and added state-of-the-art search correctness and completeness without including point correspondences, and it is known as HBST [13]. If instead a LiDAR sensor is used, one may build a set of 3D keypoints, gather them in a database and match them together with the map. Another possibility is to transform the 3D points in a 2D representation and apply the same 2D techniques used for images [14].

In both cases, robust methods are required to avoid false positive, since a wrong loop closure leads to a complete mismatch from the actual map.

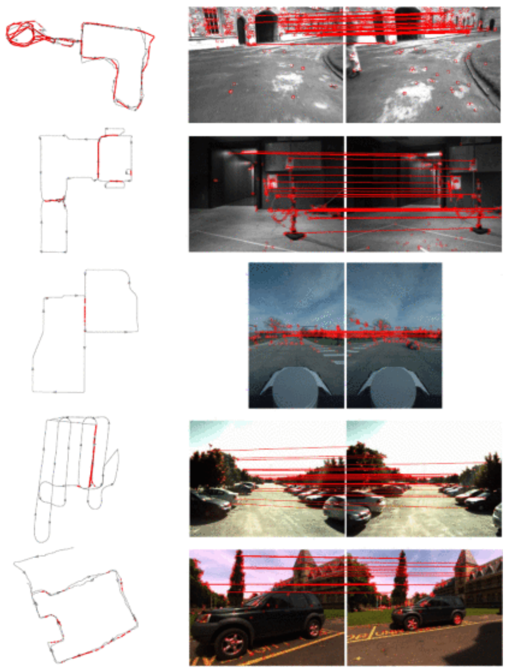

Figure 3.1. DBoW method for visual place recognition. On the right side there are image matching examples, while on the left side the trajectory adjusted after correction. Image courtesy of [12].

3.1.3 Mapping

The map created by a SLAM system may differ in type of representation, depending on the sensor used, and type of data, which allows generating 2D or 3D maps. Classical maps created by SLAM algorithms are based on metric information, using distances from obstacles for building the representation.

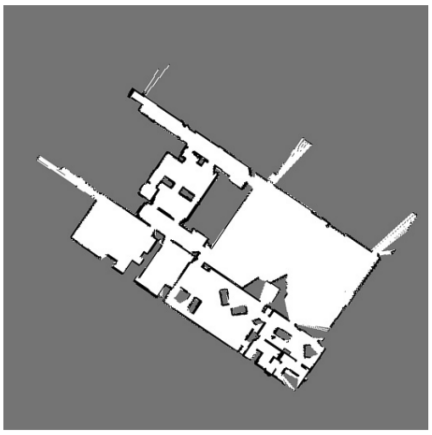

Figure 3.2. 2D metric map created by merging different robot maps evaluated with 2D LiDAR scanners. Image courtesy of [15].

Topological maps are a method of environment representation which capture the connectivity of the environment rather than creating a geometrically accurate map. In contrast, grid maps use arrays of discretized cells to represent the world, and make inferences about which cells are occupied, which are then marked. Typically, the cells are assumed to be statistically independent in order to simplify computation.

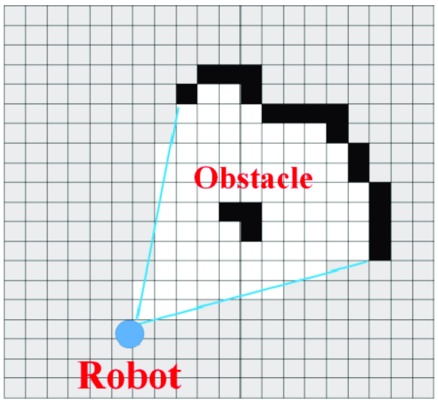

Figure 3.3. 2D grid map obtained by a low cost RGB-D sensor. Image courtesy of [16].

Modern autonomous systems, like self-driving cars, mostly simplify the mapping problem by making extensive use of highly detailed map data collected in advance. This can include map annotations, lines and segments representing the road lane, and visual data such as Google's StreetView. Essentially such systems reduces the SLAM problem to a localization only task, perhaps allowing for moving objects such as cars and people only to be updated in the map at run-time.

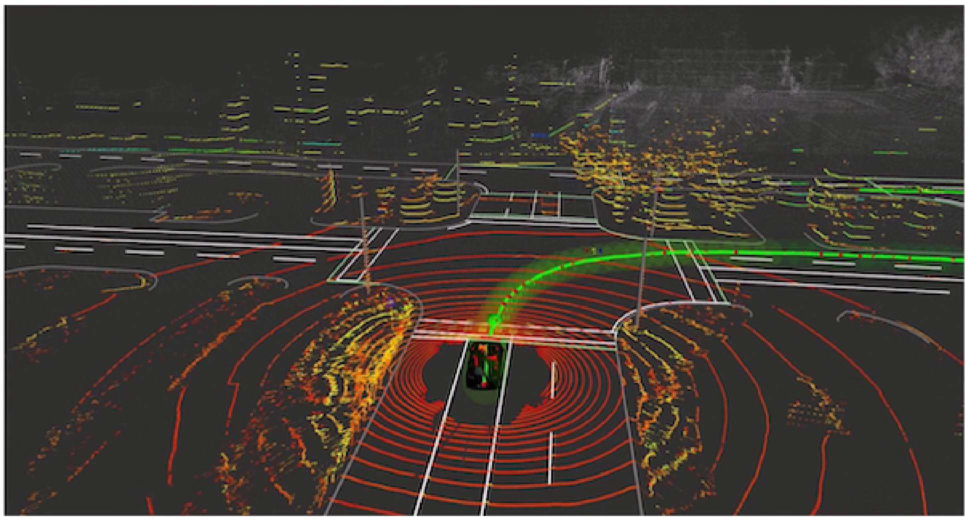

Figure 3.4. Autoware on board embedded system. It builds a 3D map obtained by combining several sensors, which are 3D LiDAR, camera, IMU, and GPS, with pre-processed 3D maps provided by an advanced driver assistance systems (ADAS). Image courtesy of [17].

3.1.4 Architecture

SLAM architecture can be divided into two main components: *front-end* and *back-end*, which structure can be seen in image 3.5.

The front-end can be closely related to computer vision and signal processing. Building a robust observation model is not an easy task, and it strictly depends on the sensor used. Every sensor provides noisy information, and the more the system is sensitive to outliers, the worst is the estimated trajectory. Moreover, sensor data, can't be exploited directly and most of the time it needs a higher level of representation. The front-end is the subsystem responsible for extracting, filtering, and associating data from the sensor, and creating an initial estimate, getting rid as much as possible of these sensor related issues. The data association problem refers to a short-term one, which deals to the association between short instants of time, and a long-term one, which handles associations between new measurements and temporally different instants, necessary for loop closure detection. The topic of the

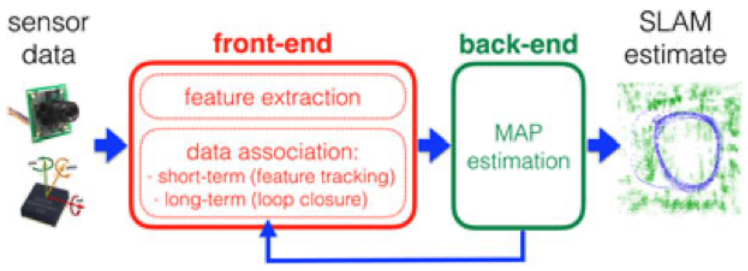

Figure 3.5. SLAM architecture. Image courtesy of [18].

thesis project mainly focuses on the front-end subsystem of SLAM.

The back-end, instead, applies a *Maximum a Posteriori* (MAP) probability estimate every time a loop closure is detected: this operation is aimed to correct the estimate made by the front-end subsystem. The problem can be interpreted in terms of inference over a *factors graph*. The probability measurement terms $P(\mathbf{z}_k|X_k)$ and the prior $P(X)$ are called *factors*, and they encode probabilistic constraints over a subset of nodes. A factor graph is a graphical model that encodes the dependence between the k-th factor (and its measurement $\mathbf{z}_k$) and the corresponding variables X_k. A first advantage of the factor graph interpretation is that it enables an insightful visualization of the problem. Figure 3.6 shows an example of a factor graph underlying a simple SLAM problem. Furthermore, the connectivity of the factor graph in turn influences the sparsity of the resulting SLAM problem. The problem is often referred to *graph-based optimization* or *graph-SLAM* [19] and it is solved as a non-linear least-squares problem as in most problems of interest in robotics, which is introduced in the next section.

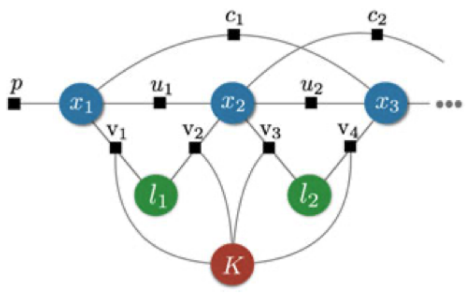

Figure 3.6. SLAM as a factor graph: Blue circles denote robot poses at consecutive time steps, green circles denote landmark positions, red circle denotes the variable associated with the intrinsic calibration parameters (K). Factors are shown as black squares: "u" correspond to odometry constraints, "v" correspond to camera observations, "c" denotes loop closures, and "p" denotes prior factors. Image courtesy of [18].

3.2 Least-Squares Optimization

Iterative Least-Squares solvers are core building blocks of many robotic applications. This technique has been traditionally used for calibration, registration, and global optimization. In the following, the mathematical concepts and derivations have been well stated in [18] and [20].

3.2.1 Problem formulation

Let W be the system with non-observable state variables X, and let Z be the measurements, i.e., a perception of the environment. The state is distributed according to a prior $P(X)$ and the conditional distribution of the measurement given the state $P(Z|X)$ is known, which is also called observation model. We want to estimate the state X by assignment of variables X^* which maximize the probability $P(X|Z)$ of the state, given the sensor measurements, as already seen in section 3.1.1. By applying Bayes theorem we can rewrite

$$X^* = \underset{X}{argmax}\ P(X|Z) = \underset{X}{argmax}\ P(Z|X)P(X) \tag{3.1}$$

In the case where no prior knowledge is available, X becomes a constant and can be dropped by the equation, so MAP reduces to *maximum likelihood estimation*. If we also assume that the measurements Z are independent the equation (3.1) factorizes into

$$\begin{aligned} X^* &= \underset{X}{argmax}\ P(X) \prod_{k=1}^{m} P(\mathbf{z}_k|X) \\ &= \underset{X}{argmax}\ P(X) \prod_{k=1}^{m} P(\mathbf{z}_k|\mathbf{x}_k) \end{aligned} \tag{3.2}$$

where, on the right-hand side, we noticed that $\mathbf{z}_k$ only depends on a subset of variables in X. We consider then the following key assumptions:

- The prior about the state is uniform

$$P(X) = \mathcal{N}(X; \mu_X; \Sigma_X) = \mathcal{N}(X; v_X; \Omega_X) \tag{3.3}$$

- The observation model is affected by Gaussian noise

$$P(\mathbf{z}_k|\mathbf{x}_k) = \mathcal{N}(\mathbf{z}_k; \mu_{\mathbf{z}|\mathbf{x}}; \Omega_{\mathbf{x}}^{-1}) \tag{3.4}$$

Equation 3.3 is expressed using the canonical parameterization of the Gaussian, with the *information matrix*, which is the inverse of the *covariance matrix* $\Omega_X = \Sigma_X^{-1}$, and the information vector $v_X = \Omega_X \mu_X$. In (3.4) the mean is the predicted measurement function $\mu_{\mathbf{z}|\mathbf{x}} = h(\mathbf{x}_k)$. According to the previous definitions we can rewrite equation (3.4) as

$$P(\mathbf{z}_k|\mathbf{x}_k) \propto exp(-||h(\mathbf{x}_k) - \mathbf{z}_k||^2_\Omega) \tag{3.5}$$

where $\mathbf{e}(\mathbf{x}_k) = h(\mathbf{x}_k) - \mathbf{z}_k$ is the error function and we used the notation $||\mathbf{e}||^2_\Omega = \mathbf{e}^T \Omega \mathbf{e}$. Since maximizing the exponential is the same as minimizing the negative log likelihood, we finally obtain

$$\begin{aligned} X^* &= \underset{X}{argmin} - log\left(P(X)\prod_{k=1}^{m} P(\mathbf{z}_k|X)\right) && (3.6)\\ &= \underset{X}{argmin} - log\left(P(X)\prod_{k=1}^{m} exp(-||\mathbf{e}_k||^2_\Omega)\right) && (3.7)\\ &= \underset{X}{argmin} \sum_{k=1}^{m} ||\mathbf{e}_k||^2_\Omega && (3.8)\end{aligned}$$

The equation (3.8) represents a non-linear least-squares problem which can be solved with the well known Gauss-Newton algorithm.

3.2.2 Gauss-Newton algorithm

The algorithm is an iterative minimization of the function $F(\mathbf{x}) = \sum_k ||\mathbf{e}_k||^2_\Omega$. For convenience, we define the stochastic variable $\Delta x = \mathbf{x} - \mu_x$ which we call the *perturbation*. For retrieving a solution of the function $F(\mathbf{x})$ we first compute the joint probability $P(\Delta\mathbf{x}, \mathbf{z})$ using the chain rule, and then we condition on the known measurement $\mathbf{z}$. In the case of a linear measurement function $h(\mathbf{x}) = \mathbf{A}(\mathbf{x} - \mu_x) + \hat{\mathbf{z}}$, the prediction model has the form

$$P(\mathbf{z}|\Delta\mathbf{x}) = \mathcal{N}(\mathbf{z}, \mu_{z|\Delta x}) = \mathbf{A}\Delta\mathbf{x} + \hat{\mathbf{z}}; \Omega^{-1}_{\Delta z|x}) \tag{3.9}$$

Under the Gaussian assumptions we made, the joint distribution has the form

$$P(\Delta\mathbf{x}, \mathbf{z}) = \mathcal{N}(\Delta\mathbf{x}, \mathbf{z}; \mu_{\Delta x,z}; \Omega^{-1}_{\Delta x,z}) \tag{3.10}$$

$$\mu_{\Delta\mathbf{x},\mathbf{z}} = \begin{pmatrix} \mathbf{0} \\ \hat{\mathbf{z}} \end{pmatrix} \quad \Omega_{\Delta\mathbf{x},\mathbf{z}} = \begin{pmatrix} \Omega_{xx} & \Omega_{xz} \\ \Omega^T_{xz} & \Omega_{zz} \end{pmatrix} \tag{3.11}$$

The values in $\Omega_{\Delta\mathbf{x},\mathbf{z}}$ are obtained applying the chain rule to multivariate Gaussian to (3.9) and $P(\Delta\mathbf{x})$, and they result in the following

$$\begin{aligned} \Omega_{xx} &= \mathbf{A}^T \Omega_{z|x} \mathbf{A} + \Omega_x \\ \Omega_{xz} &= -\mathbf{A}^T \Omega_{z|\Delta x} \\ \Omega_{zz} &= \Omega_{z|x} \end{aligned}$$

Since the prior is non-informative we can set $\Omega_x = 0$ and the information vector of the joint distribution is computed as

$$v_{\Delta x,z} = \Omega_{\Delta x|z}\mu_{\Delta x|z} = \begin{pmatrix} -\mathbf{A}^T\Omega_{z|x}\hat{\mathbf{z}} \\ \Omega_{z|x}\hat{\mathbf{z}} \end{pmatrix} \tag{3.12}$$

Integrating the known measurement $\mathbf{z}$ in the joint distribution $P(\Delta\mathbf{x}, \mathbf{z})$ results in a new distribution $P(\Delta\mathbf{x}|\mathbf{z})$. This can be done by conditioning in the Gaussian domain

$$P(\Delta\mathbf{x}|\mathbf{z}) \sim \mathcal{N}(\Delta\mathbf{x}; v_{\Delta x|z}; \Omega_{\Delta x|z}) \tag{3.13}$$

where

$$v_{\Delta x|z} = v_{\Delta x} - (\mathbf{A}^T\Omega_{z|x})\mathbf{z} = \mathbf{A}^T\Omega_{z|x}(\mathbf{z} - \hat{\mathbf{z}}) \tag{3.14}$$

$$\Omega_{\Delta x|z} = \Omega_{zz} = \mathbf{A}^T\Omega_{z|x}\mathbf{A} \tag{3.15}$$

The conditioned mean is retrieved from

$$\mu_{\Delta x|z} = \Omega^{-1}_{\Delta x|z}v_{x|z} = -\mathbf{H}^{-1}\mathbf{A}^T\Omega_{z|x}(\mathbf{z} - \hat{\mathbf{z}}) \tag{3.16}$$

Here we rename $\mathbf{e} = (\mathbf{z} - \hat{\mathbf{z}})$ the *measurement error*, $\mathbf{H} = \mathbf{A}^T\Omega_{z|x}\mathbf{A}$ the information matrix of the estimate, and $\mathbf{b} = \mathbf{A}^T\Omega_{z|x}\mathbf{e}$ the information vector of the estimate. Remembering that $\Delta x = \mathbf{x} - \mu_x$, the Gaussian distribution over the conditioned states has the same information matrix, while the mean is obtained by summing the increment's mean

$$\mu_{x|z} = \mu_x + \mu_{\Delta x|z} \tag{3.17}$$

An important derivation is that besides the optimal solution $\mu_{x|z}$, with the matrix $\mathbf{H} = \Omega_{\Delta x|z}$ we can also estimate its uncertainty $\Sigma_{\Delta x|z} = \Omega^{-1}_{\Delta x|z}$. Integrating multiple independent measurements $\mathbf{z}_{1:k}$ requires stacking them in a single vector. As a result, matrix $\mathbf{H}$ and vector $\mathbf{b}$ are composed by the sum of each measurement's contribution

$$\mathbf{H} = \sum_{k=1}^{m} \mathbf{A}_k^T\Omega_{z_k|x}\mathbf{A}_k \qquad \mathbf{b} = \sum_{k=1}^{m} \mathbf{A}_k^T\Omega_{z_k|x}\mathbf{e}_k \tag{3.18}$$

Equations (3.16), (3.17) and (3.18) allow us to find the exact mean of the conditional distribution, under the assumptions that (i) the measurement noise is Gaussian, (ii) the measurement function is an affine transform of the state, and (iii) both measurement and state spaces are Euclidean. Now we first relax the assumption on the affinity of the measurement function, leading to the common derivation of the Gauss-Newton algorithm.

If the measurement model is a non-linear function of the state, we can approximate the behaviour of the solution $\mu_{x|z}$ with the Taylor's expansion around the mean

$$\mathbf{h}(\check{\mathbf{x}} + \Delta\mathbf{x}) \approx \mathbf{h}(\check{\mathbf{x}}) + \left.\frac{\partial \mathbf{h}(\mathbf{x})}{\partial \mathbf{x}}\right|_{x=\check{x}} \Delta\mathbf{x} = \mathbf{J}\Delta\mathbf{x} + \hat{\mathbf{z}} \tag{3.19}$$

where $\mathbf{J}$ is the Jacobian of the measurement function. Summarizing, at each iteration, the Gauss-Newton algorithm:

- processes each measurement $\mathbf{z}_k$ by evaluating error $\mathbf{e}_k(\mathbf{x}) = \mathbf{h}_k(\mathbf{x}) - \mathbf{z}_k$ and Jacobian $\mathbf{J}_k$ at the current solution $\check{\mathbf{x}}$
- builds a coefficient matrix and coefficient vector for the linear system in Equation 3.16, and computes the optimal perturbation by solving a linear system $\Delta x = \mathbf{H}^{-1}\mathbf{b}$
- applies the computed perturbation to the current state to get an improved estimate $\check{\mathbf{x}} \leftarrow \check{x} + \Delta\mathbf{x}$

The last thing to discuss is the case when the operations of addition and subtraction for computing the error $\mathbf{e}_k$ and apply the increments $\Delta\mathbf{x}$ are not defined in the Euclidean space. This always happens in computer vision and robotics applications, leading to poor results. For this reason, a typical space in which rotational or similarity transformations lie is considered, that is the *Manifold*. A smooth manifold $\mathbb{M}$ is a space that, since it is not homeomorphic to $\mathbb{R}^n$, admits a locally Euclidean parameterization around each element $\mathbf{M}$ of the domain, commonly referred to as *chart*, as illustrated in Figure 3.7.

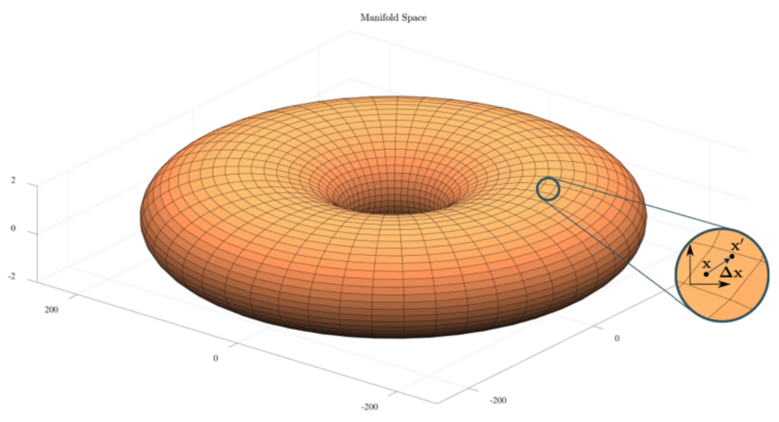

Figure 3.7. Illustration of a manifold space. Since the manifold is smooth, local perturbations i.e., $\Delta\mathbf{x}$ in the illustration, can be expressed with a suitable Euclidean vector. Image courtesy of [20].

A chart computed in a manifold point $\mathbf{M}$ is a function from $\mathbb{R}^n$ to a new point $\mathbf{M}$' on the manifold:

$$chart_{\mathbf{M}}(\Delta\mathbf{m}) : \mathbb{R}^n \rightarrow \mathbb{M} \tag{3.20}$$

Similarly given two points $\mathbf{M}$ and $\mathbf{M}'$ on the manifold, we can determine the motion $\Delta\mathbf{m}$ on the chart constructed on $\mathbf{M}$ that would bring us to $\mathbf{M}'$. Let this operation be the inverse chart, denoted as $chart^{-1}_{\mathbf{M}}(\mathbf{M}')$. The direct and inverse charts allow us to define operators on the manifold that are analogous to the sum and subtraction. Those operators, referred to $\boxplus$ and $\boxminus$, are defined as

$$\mathbf{M} = \mathbf{M} \boxplus \Delta\mathbf{m} \overset{\text{def}}{=} chart_{\mathbf{M}}(\Delta\mathbf{m}) \tag{3.21}$$

$$\Delta\mathbf{m} = \mathbf{M}' \boxminus \mathbf{M} \overset{\text{def}}{=} chart^{-1}_{\mathbf{M}}(\mathbf{M}') \tag{3.22}$$

Once proper operators are defined, we can reformulate our minimization problem in the manifold domain. To this extent, we can simply replace the $+$ with a $\boxplus$ in the computation of the Taylor expansion of Equation (3.19). Since we will compute an increment on the chart, we need to compute the expansion on the chart $\Delta\mathbf{x}$ at the local optimum that is at the origin of the chart itself $\Delta\mathbf{x} = 0$. The same holds in the increment equation, while if the measurement lies on a manifold too, a local $\boxminus$ operator is needed to compute the error.

We finally have all the elements to show the Gauss-Newton algorithm for iterative least-squares optimization

Algorithm 1 Gauss-Newton for Manifold Least Square Optimization

```
Require: initial guess $\check{\mathbf{x}}$, measurements $Z = \langle \mathbf{z}_k, \Omega_k \rangle$, $n$ iterations
Ensure: optimal solution $\mathbf{x}^*$
  for iteration = 1...$n$ do
      $\mathbf{H} \leftarrow \mathbf{0}$, $\mathbf{b} \leftarrow \mathbf{0}$
      for all $k \in \{1...K\}$ do
          $\mathbf{e}_k \leftarrow \mathbf{h}_k(\check{\mathbf{x}}) \boxminus \mathbf{z}_k$
          $\mathbf{J}_k \leftarrow \frac{\partial \mathbf{e}(\check{\mathbf{x}} \boxplus \Delta\mathbf{x})}{\partial \Delta\mathbf{x}}\big|_{\Delta x = 0}$
          $\mathbf{H} \leftarrow \mathbf{H} + \mathbf{J}_k \Omega_k \mathbf{J}_k$
          $\mathbf{b} \leftarrow \mathbf{b} + \mathbf{J}_k \Omega_k \mathbf{e}_k$
      end for
      $\Delta\mathbf{x} \leftarrow solve(\mathbf{H}\Delta\mathbf{x} = -\mathbf{b})$
      $\check{\mathbf{x}} \leftarrow \check{\mathbf{x}} \boxplus \Delta\mathbf{x}$
  end for
  return $\check{\mathbf{x}}$
```

The optimal solution $\mathbf{x}^*$ is ensured if and only if a number of sufficient iterations is executed for converging to the solution. Having defined the general Gauss-Newton algorithm, in order to run it on a specific problem, one just need to:

- define for each type of variable $\mathbf{x}_i$ of the state $\mathbf{x}$ (i) an extended parameterization $\mathbf{X}_i$, (ii) a vector perturbation $\Delta\mathbf{x}_i$, and (iii) a $\boxplus$ operator that computes a new point on the manifold $\mathbf{X}'_i = \mathbf{X}_i \boxplus \Delta\mathbf{x}_i$. If the variable is Euclidean, the extended and the increment parameterization match and thus $\boxplus$ degenerates to vector addition.

- For each type of factor $P(\mathbf{z}_k|\mathbf{x}_k)$ of the measurements, specify (i) an extended parameterization $\mathbf{Z}_i$, (ii) a Euclidean representation $\Delta\mathbf{z}_i$ for the error vector, and (iii) a $\boxminus$ operator such that, given two points on the manifold $\mathbf{Z}_i$ and $\mathbf{Z}'_i$, $\Delta\mathbf{z}_i = \mathbf{Z}'_i \boxminus \mathbf{Z}_i$ represents the motion on the chart that moves $\mathbf{Z}_i$ onto $\mathbf{Z}'_i$. If the measurement is Euclidean, the extended and perturbation parameterization match and, thus, $\boxminus$ becomes a simple vector difference. Finally, it is necessary to define the measurement function $\mathbf{h}_k(\mathbf{X}_k)$ that, given a subset of state variables $\mathbf{X}_k$, computes the expected measurement $\mathbf{Z}_k$.

3.2.3 Iterative Closest Point

A typical example of smooth manifold is the $SE(3)$ domain of 3D homogeneous transformations. We consider two sets of 3D points (point clouds) and we want to find the transform that maximize the overlap between the two, as in Figure 3.8. Let P^f be the one fixed and P^m the one that is moved. The problem is known as *Iterative Closest Point* (ICP) registration which takes an initial guess of the target transformation $\check{\mathbf{X}}$, pairs of point correspondences $\left\langle p^m, p^f \right\rangle$, and computes the optimal transformation $\mathbf{X}^*$.

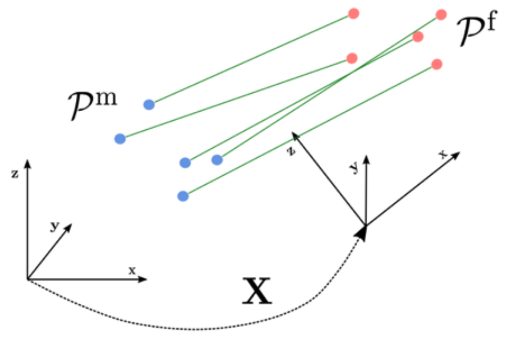

Figure 3.8. Registration of two point cloud through ICP. The red points represent entries of the fixed cloud, while the blue points belong to the moving one. Green lines emphasize the associations between points belonging to the two clouds. Image courtesy of [20].

The problem can be formulated according to the following definitions:

- The state is represented by a rotation matrix and a translation vector, therefore $\mathbf{X} = [\mathbf{R} \mid \mathbf{t}]$.
- An Euclidean parameterization for the chart is $\Delta\mathbf{x} = (\Delta x\ \Delta y\ \Delta z\ \Delta\alpha_x\ \Delta\alpha_y\ \Delta\alpha_z) \in \mathbb{R}^6$, where the first three elements $\mathbf{\Delta t}$ are simply Euclidean quantities, while the *alpha* deltas are Euler angles rotation contributions.
- The addition operator is $\mathbf{X} \boxplus \Delta\mathbf{x} = vector2transform(\Delta\mathbf{x})\mathbf{X}$. The $v2t$ function takes the perturbation vector and builds a transformation matrix including a rotational part, which is the composition of the rotation matrices relative to each Euler angle, and a translational part simply taking the single vector contributions

$$v2t(\Delta\mathbf{x}) = [\mathbf{R}_x(\alpha_x)\mathbf{R}_y(\alpha_y)\mathbf{R}_z(\alpha_z) \mid \Delta t] \tag{3.23}$$

In this problem, we have just one type of factor, which depends on the relative position between a pair of corresponding points, after applying the current transformation $\mathbf{X}$ to the moving cloud. Given a set of points correspondences $\left\{\left\langle p_0^m, p_{j(0)}^f \right\rangle, ..., \left\langle p_K^m, p_{j(K)}^f \right\rangle\right\}$, each fixed point p_j^f constitutes a measurement since its value does not change during optimization. On the contrary, each moving point p_k^m will be used to generate the prediction $\hat{\mathbf{z}}$. Therefore we need the following entities:

- Measurement function: it computes the position of point p_k^m that corresponds to the point $p_{j(k)}^f$ by applying the transformation $\mathbf{X}$, so

$$\mathbf{h}_k^{icp}(\mathbf{X}) = \mathbf{R}^T(p_{j(k)}^m - \mathbf{t}) \tag{3.24}$$

- Error function: since both prediction and measurement are Euclidean, the $\boxminus$ operator degenerates to a simple minus

$$\mathbf{e}_k^{icp}(\mathbf{X}) = \mathbf{h}_k^{icp}(\mathbf{X}) - p_{j(k)}^f \tag{3.25}$$

The only one element missing is the Jacobian which can be computed from the error function as:

$$\begin{aligned}
\mathbf{J}_k^{icp}(\mathbf{X}) &= \left.\frac{\partial \mathbf{e}_k^{icp}(\mathbf{X} \boxplus \Delta\mathbf{x})}{\partial \Delta\mathbf{x}}\right|_{\Delta x=0} \\
&= \left.\begin{pmatrix} \frac{\partial \mathbf{e}_k^{icp}(\cdot)}{\partial \mathbf{\Delta t}} & \frac{\partial \mathbf{e}_k^{icp}(\cdot)}{\partial \Delta\alpha} \end{pmatrix}\right|_{\Delta x=0} \\
&= \left.\begin{pmatrix} \frac{\partial \mathbf{\Delta t}}{\partial \mathbf{\Delta t}} & \frac{\partial \mathbf{R}(\Delta\alpha) p_{j(k)}^m}{\partial \Delta\alpha} \end{pmatrix}\right|_{\Delta x=0} \\
&= \begin{pmatrix} \mathbf{I} & -[p_{j(k)}^m]_\times \end{pmatrix}
\end{aligned}$$

where we used the skew operator of a vector $[v]_\times$ which computes the relative 3×3 skew-symmetric matrix. With this in place, we can instantiate and run the Algorithm 1.

Minimization algorithms like Gauss-Newton or Levenberg-Marquardt (LM) lead to the repeated construction and solution of the linear system $\mathbf{H}\Delta\mathbf{x} = -\mathbf{b}$. In many cases, each measurement $\mathbf{z}_k$ only involves a small subset of state variables. Therefore, in the cases when the number of measurements is proportional to the number of variables such as SLAM, the system matrix $\mathbf{H}$ is *sparse*, symmetric and positive semi-definite by construction. Exploiting these intrinsic properties, we can efficiently solve the linear system in Equation 3.16. In fact, the literature provides many solutions to this kind of problem, which can be arranged in two main groups: (i) iterative methods and (ii) direct methods. The first compute an iterative solution to the linear system by following the gradient of the quadratic form. Iterative methods might require a quadratic time in computing the exact solution of a linear system, but they are required when the dimension of the problem is very large. Direct methods, instead, return the exact solution of the linear system, usually leveraging on some matrix decomposition such as the *Cholesky factorization*, which is the one used in the thesis project.

Linear Relaxation

The ICP problem could be approached without applying iteratively the Gauss-Newton algorithm. If we relax the constraint that $\mathbf{R}$ is a rotation matrix, the affine transformation can be estimated minimizing the error of the 12 parameters vector $\mathbf{X} = (\mathbf{r}_1\ \mathbf{r}_2\ \mathbf{r}_3\ \mathbf{t}^T)$ where $\mathbf{r}_i$ is the i-th row of the rotation matrix $\mathbf{R}$, and $\mathbf{t}$ is the translation vector. This estimation can be formulated as an optimization problem, whose objective function is as follows:

$$\mathbf{e}(\mathbf{R}, \mathbf{t}, c) = \frac{1}{k}\sum_{i=1}^{k} ||\mathbf{p}^f_{j(k)} - (c\mathbf{R}\mathbf{p}^m_k + \mathbf{t})||^2 \tag{3.26}$$

In the equation the constant value c is the scale factor, which is not present in our case, since the measurements come from the same sensor dimensionality, therefore will be set to 1. The algorithm follows from Arun's result [21] based on singular value decomposition, and it is well explained in [22].

Algorithm 2 is valid only under the assumption that all the correspondences between the points of the clouds have the same uncertainty, which is reasonable since the measurements come from the same sensor. Moreover it estimates a rotation matrix $\mathbf{R}$ which not necessarily belongs to the orthonormal group $SO(3)$. This may happen when there are wrong correspondences, in which case the algorithm uses the additional degrees o freedom to (wrongly) reduce the error as much as possible.

Algorithm 2 Closed-form Solution of Least-Squares Optimization

```
Require: point correspondences $\{\langle x_0, y_0\rangle, ..., \langle x_K, y_K\rangle\}$
Ensure: closed-form solution of $\mathbf{R}$ and $\mathbf{t}$
  $\boldsymbol{\mu}_x \leftarrow \frac{1}{k}\sum_{i=1}^{k}\mathbf{x}_i, \quad \boldsymbol{\mu}_y \leftarrow \frac{1}{k}\sum_{i=1}^{k}\mathbf{y}_i$
  $\boldsymbol{\sigma}_x \leftarrow \frac{1}{k}\sum_{i=1}^{k}||\mathbf{x}_i - \boldsymbol{\mu}_x||^2, \quad \boldsymbol{\sigma}_y \leftarrow \frac{1}{k}\sum_{i=1}^{k}||\mathbf{y}_i - \boldsymbol{\mu}_y||^2$
  $\boldsymbol{\Sigma} \leftarrow \frac{1}{k}\sum_{i=1}^{k}(\mathbf{y}_i - \boldsymbol{\mu}_y)(\mathbf{x}_i - \boldsymbol{\mu}_x)^T$
  $\mathbf{UDV}^T \leftarrow SVD(\boldsymbol{\Sigma})$
  if $det(\mathbf{U})det(\mathbf{V}) = 1$ then
      $\mathbf{S} = \mathbf{I}_{3\times3}$
  else
      $\mathbf{S} = diag(1, 1-1)$
  end if
  $\mathbf{R} \leftarrow \mathbf{USV}^T$
  $\mathbf{t} \leftarrow \boldsymbol{\mu}_y - \mathbf{R}\boldsymbol{\mu}_x$
  return $\mathbf{R}, \mathbf{t}$
```

3.2.4 Outlier Rejection

In the previous section, we assumed the point correspondences were correctly associated by some pre-processing mechanism before feeding them as input of the two algorithms. In reality, this is not the case, because sensors acquire measurements with different kind of noises:

- *intrinsic* noise is the one coming from the internal structure of the sensor, such as calibration parameters, lens distortion, or some low-level control process. Since it is repeatable at every cycle, it can be taken into account and solved by calibration techniques.

- *extrinsic* noise does not depend on the sensor itself, by rather from external influence of the environment. This may vary depending on the sensor and robot platform used. In the case of an autonomous vehicle with a LiDAR sensor mounted on top, this could be generated by drive maneuvers, holds on the road, wind action, etc., causing inconsistency between successive point clouds.

For these reasons, wrong associations, called *outliers*, may be present, while good ones are called *inliers*. As already said, small errors are accumulated along the trajectory, causing large displacements at the end. Therefore, in order to reject outliers and obtain the best state estimation possible, one can adapt different strategies depending on the method used.

Robust estimator

Iterative least-square methods compute the error by the difference between prediction and measurement, and they use it to update the information vector of the estimate $\mathbf{b}$. If the error is large it may potentially represent an outlier case, which implies a consistent wrong perturbation vector $\Delta\mathbf{x}$ which is summed up to the estimate. However, there are circumstances where all errors are quite large even if no outliers are present. A typical case occurs when we start the optimization from an initial guess which is far from the optimum. A possible solution to this issue consists of carrying on the optimization under a cost function that grows sub-quadratically with the error. Indicating with $u_k(x) = \sqrt{\mathbf{e}_k(\mathbf{x})^T \Omega_k \mathbf{e}_k(\mathbf{x})}$ the L1 Omega-norm of the error term in Equation 3.8, we can generalize the error equation in term of a scalar function $\rho(u)$ that compute a new error term as a function of the L1-norm:

$$\underset{X}{argmin} \sum_{k=1}^{m} \rho(u_k(x)) \tag{3.27}$$

which is called *robustifier*. The analysis of the gradient of the Equation 3.27 shows that the robustifier function $\rho(u_k(x))$ regulates the magnitude of the error term through a scalar function $\gamma_k(x)$. By absorbing this scalar term at each iteration in a new information matrix $\Omega_k(x) = \gamma_k(x)\Omega_k$, we can rely on the iterative algorithm illustrated in the previous sections to implement a robust estimator. This formalization of the problem is called Iterative Reweighed Least-Squares (IRLS).

Algorithm 3 Iterative Reweighed Least Squares Optimization for Manifold

```
...
for iteration = 1...$n$ do
    ...
    for all $k \in \{1...K\}$ do
        $\mathbf{e}_k \leftarrow \mathbf{h}_k(\check{\mathbf{x}}) \boxminus \mathbf{z}_k$
        $\mathbf{J}_k \leftarrow \frac{\partial \mathbf{e}(\check{\mathbf{x}} \boxplus \Delta\mathbf{x})}{\partial \Delta\mathbf{x}}\big|_{\Delta x=0}$
        $u_k \leftarrow \sqrt{\mathbf{e}_k(\mathbf{x})^T \Omega_k \mathbf{e}_k(\mathbf{x})}$
        $\gamma_k \leftarrow \frac{1}{u_k} \frac{\partial \rho_k(u)}{\partial u}\big|_{u=u_k}$
        $\Omega_k = \gamma_k \Omega_k$
        $\mathbf{H} \leftarrow \mathbf{H} + \mathbf{J}_k \Omega_k \mathbf{J}_k$
        $\mathbf{b} \leftarrow \mathbf{b} + \mathbf{J}_k \Omega_k \mathbf{e}_k$
    end for
    ...
end for
...
```

The use of robust cost functions reduces the contribution of measurements with a higher error. The robust version of the Gauss-Newton algorithm requires to choose a robustifier function $\rho_k(u)$ for each type of factor. The non-robust case is captured by choosing $\rho_k(u) = \frac{1}{2}u^2$.

Random Sample Consensus

Robust estimators rely on the iterative approach for regulating the error, therefore it cannot be taken into account when using closed-form solutions algorithms. For this cases, one may use a non-deterministic method which (i) estimates a model taking randomly a minimum solution from the available data, (ii) evaluates the model by fitting all the data, and (iii) iteratively seeks for the best solution. Such a method is called Random Sample Consensus (RANSAC), which produces an estimation that is more accurate the more iterations are performed. In our specific case, the steps of RANSAC are the following:

1. **Randomly select a minimum amount of data**. A minimum solution for computing a rigid body transformation for aligning two point clouds, is formed by two pairs of three points.
2. **Compute a model**. A rigid body transformation between two pairs of three points can be obtained using the closed-form solution Algorithm 2.
3. **Evaluate the model**. The rigid body transformation is applied to all the points of the moving cloud, then, for each pair, the Euclidean distance between the fixed point and the transformed point is computed. If this result is below a certain threshold, the pair is considered an inlier, otherwise an outlier. The total number of inliers represents the model's score.
4. **Repeat**. The procedure is repeated N times. Each time, if a higher score is found, the best model is updated.

The pseudo-code is reported in Algorithm 4. The model estimated by RANSAC algorithm depends on the number of iterations chosen, therefore there is no guarantee that the solution found is correct. However the correctness of the solution can be estimated according to the number of iterations. Let w be the percentage of inliers present in the dataset, n the minimum number of data to calculate a solution and p the desired probability of success. The number of iterations k necessary to obtain a solution with probability p will then be

$$k = \frac{log(1-p)}{log(1-w^n)} \tag{3.28}$$

Algorithm 4 RANSAC

```
Require: data $\mathbf{D}$, iterations $N$, inliers threshold $t$
Ensure: model $\mathbf{B}$ which best fits the data
  bestModel ← $null$
  bestScore ← 0
  for iteration = $1...N$ do
      randomData ← $randomSelect(\mathbf{D})$
      model ← $fitModel(randomData)$
      score ← $evalute(model, D, t)$
      if score > bestScore then
          bestModel ← $model$
          bestScore ← $score$
      end if
  end for
  return bestModel
```

The advantage of RANSAC is the ability to estimate the parameters of a model robustly, even in presence of several outliers. On the other hand, its disadvantage is that the optimal solution would require an infinite number of iterations, so setting the iterations to a finite number N, one can only reach the optimal solution with probability p.

3.3 Feature Extraction

In computer vision, the task of detecting features and being able to match them in different sequences is of fundamental importance for many applications such as automate object tracking, point matching for computing disparity, stereo calibration, motion-based segmentation, recognition, 3D object reconstruction, robot navigation, image retrieval and indexing, and many others. Features are intended to be informative and non-redundant, and since they are a small subset of the original data, feature extraction is also related to dimensionality reduction. Very often, the input data is too large and may contain redundant information, which slow down the learning process and biases more examples with respect to others. In images, this can be represented by data containing repetitive pixels, which can be reduced by detecting a set of interesting points, called keypoints. A *keypoint* is generally defined as the 2D location within the image, and described by a set of surrounding pixels.

A robust feature extraction technique, must be invariant with respect to geometric transformations such as translations, rotations, and scaling, and concerning photometric transformations, such as brightness and exposure. In recent computer vision applications, it has been observed that approaches using neural networks for features extraction are more robust and able to generalize better than others, as highlighted in the study reported in [23].

3.3.1 Traditional approach

A keypoint is the point which is expressive in texture, i.e. the point at which the direction of the boundary of the object changes abruptly or the intersection point between two or more edge segments. These are represented by a *corner.* The idea behind corner detection is to consider a small patch around each pixel p in an image. We want to identify all such pixel patches that are unique. Uniqueness can be measured by shifting each patch by a small amount in a given direction and measuring the amount of change that occurs in the pixel values. Mathematically this can be formalized as a change function, taking the sum squared difference (SSD) of the pixel values before and after the shift, and identifying patches where the SSD is large in all directions

$$E(u,v) = \sum_{xy} w(x,y)[I(x+u,y+v) - I(x,y)]^2 \tag{3.29}$$

where (u,v) are the (x,y) coordinates of the image patch I. The function $E(u,v)$ has to be maximized for corner detection. Applying Taylor Expansion to the above equation and after some mathematical steps, we get the equation

$$E(u,v) \approx [u,v]\left(\sum \begin{bmatrix} I_x^2 & I_{xy} \\ I_{xy} & I_y^2 \end{bmatrix}\right)\begin{bmatrix} u \\ v \end{bmatrix} \tag{3.30}$$

obtaining the auto-correlation matrix

$$A = \sum_{xy} w(x,y)\begin{bmatrix} I_x^2 & I_{xy} \\ I_{xy} & I_y^2 \end{bmatrix} \tag{3.31}$$

By solving for the eigenvectors of the matrix A, we can obtain the directions for both the largest and smallest increases in SSD. The corresponding eigenvalues λ_1,λ_2 give us the actual value amount of these increases. The score for each patch is calculated as

$$R = det(A) - \alpha(trace(A))^2$$

- if $|R|$ is small, which happens when λ_1 and λ_2 are small, the region is *flat.*
- if $R < 0$, which happens when $\lambda_1 >> \lambda_2$ or vice versa, the region is an *edge.*
- when R is large, which happens when $\lambda_1 >> \lambda_2$ are large and $\lambda_1 \sim \lambda_2$, the region is a *corner.*

This method was first introduced by Chris Harris and Mike Stephens [24] and takes the name of *Harris Corner Detector*. A graphical represention of the criterion can be seen in Figure 3.9.

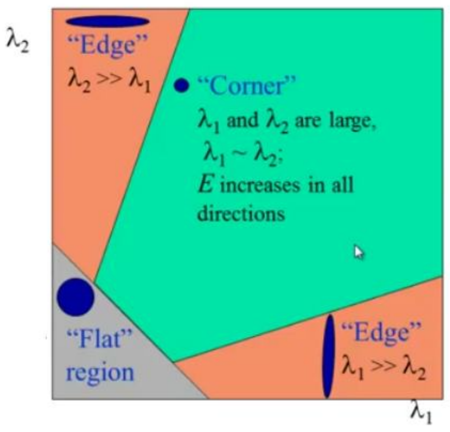

Figure 3.9. Harris Corner Detector Thresholding. Image taken from openCV documentation.

Harris detection is efficient in regular images but can suffer when these are scaled. This problem was solved by D. Lowe which introduced *Scale Invariant Feature Transform* (SIFT) [25]. The algorithm can be divided mainly in four steps:

1. **Scale-space peak selection**. Real world objects are meaningful only at a certain scale, and a scale space attempts to replicate this concept on digital images. This is realized by means of the Difference of Gaussians (DoG), a function $L(x, y, \sigma) = G(x, y, \sigma) * I(x, y)$ that is produced from the convolution of a Gaussian kernel (Blurring) at different scales with the input image. This process is done for different octaves in the Gaussian Pyramid, which can be seen in Figure 3.10.

2. **Keypoint localization**. DoGs are then used for computing Laplacian of Gaussians (LoG), which are scale inviariant. One pixel in an image is compared with its neighbors as well as the pixels in the next and previous scales. If it is a local extrema, it is a potential keypoint. In order to get rid of edges, similarly to Harris corner detector, a hessian matrix H is used to discriminate flats, edges and corners.

3. **Orientation assignment**. In order to make the features rotation invariant, a neighborhood of pixels is taken around the keypoint location depending on the scale, and the gradient magnitude and direction is calculated in that region. An orientation histogram with 36 bins covering 360 degrees is created. The

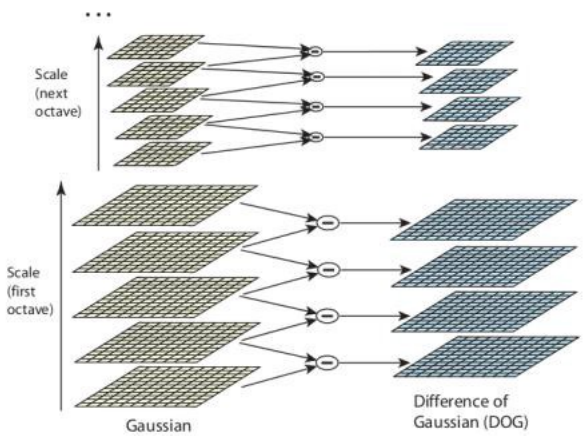

Figure 3.10. DoG computed for the Gaussian pyramid. Image courtesy of [25].

highest peak bin along with others below a certain percentage of it, are taken to compute the orientation.

4. **Keypoint descriptor**. At this point, each keypoint has a location, scale, and orientation. In order to build a descriptor a 16×16 window around the keypoint is considered. For each 4×4 sub-block of the window a 8 bin orientation histogram is created. Achieving orentation and illumination independence is done by (i) subtracting the keypoint's rotation from each orientation, and (ii) normalizing the feature vector with a threshold.

Although SIFT had introduced a robust method for finding invariant keypoints, the process can be very expensive in terms of computational resources and time. Both the detection and description phases were improved in efficiency.

For detection, *Features from Accelerated Segment Test* (FAST) [26] was introduced. The algorithm takes a pixel p in an array and compares the brightness of p to surrounding 16 pixels that are in a small circle around p. The pixels in the circle are then sorted into three classes (lighter than p, darker than p or similar to p). If more than 8 pixels are darker or brighter than p then it is selected as a keypoint. The most promising advantage of the FAST corner detector is its computational efficiency. The detection process is represented in Figure 3.11.

For description, *Binary Robust Independent Elementary Features* (BRIEF) [27] has taken hold. After detecting keypoints the algorithm converts image patches around them into a binary feature descriptor, which contains only 1s and 0s, so that together they can represent an object. BRIEF deals with the image at pixel level, so it is very noise-sensitive. By pre-smoothing the patch, this sensitivity can be reduced, thus increasing the stability and repeatability of the descriptors. This is done applying a Gaussian kernel. Then, the binary feature vector is created

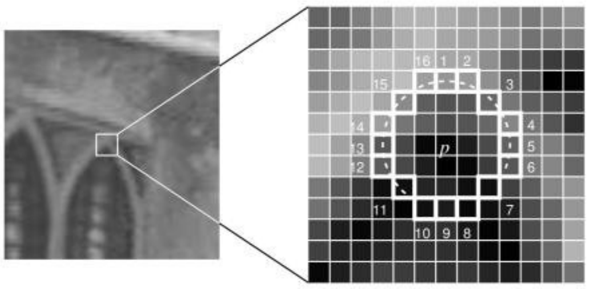

Figure 3.11. 12 point segment test corner detection in an image patch. The highlighted squares are the pixels used in the corner detection. The pixel at p is the center of a candidate corner. Image taken from openCV documentation.

according to a test:

$$\tau(p; x, y) = \begin{cases} 1 & \text{if } p(x) < p(y) \\ 0 & \text{if } p(x) \geq p(y) \end{cases}$$

where $p(x)$ is the intensity of patch p at a point x. The feature is defined as a vector of n binary tests

$$f_n(p) = \sum_{1 \leq i \leq n} 2^{i-1} \tau(p; x_i, y_i) \tag{3.32}$$

Choosing a set of $n(x, y)$-location pairs uniquely defines a set of binary tests. The (x, y) pair is also called random pair which is located inside the patch. In total n random pairs are selected for creating a binary feature vector, and they are chosen from one of five *sampling geometries* which can be seen in Figure 3.12. Each keypoint is described by a feature vector which is 128–512 bits string. BRIEF relies on a relatively small number of intensity difference tests, allowing descriptor construction and matching much faster than other state-of-the-art algorithms.

Oriented FAST and Rotated BRIEF

The corner detection and descriptor mentioned earlier have some problems. FAST detector do not have an orientation component and multiscale features. Therefore, rotating and scaling an image causes a change of keypoints detected. Also, BRIEF suffer from a related problem: the descriptors extracted using this algorithm are not invariant with respect to rotations. A combination of the two techniques which solves these issues is called *Oriented FAST and Rotated BRIEF* (ORB) [28]. For the detection phase, ORB uses a multiscale image pyramid as in SIFT, and then detects keypoints in the image using the FAST algorithm. By detecting keypoints at each level is effectively identify them at a different scale. In this way, ORB is partially scale invariant. After locating keypoints ORB assigns an orientation to each one left

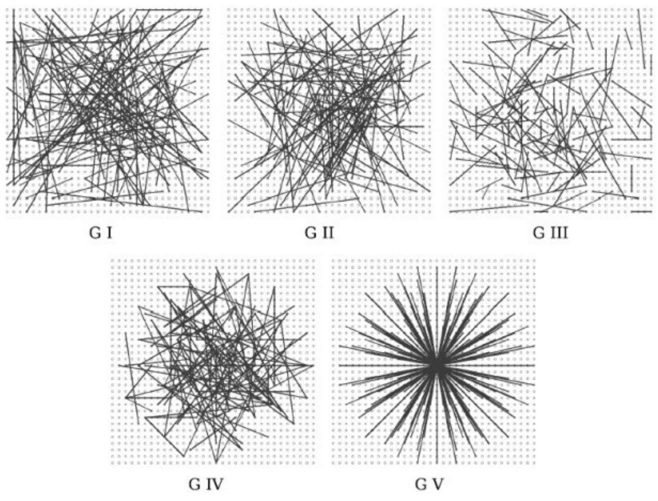

Figure 3.12. Different sampling approaches to choosing the test locations. Image courtesy of [27].

or right facing depending on how the levels of intensity change around that keypoint. For detecting intensity changes ORB uses intensity centroids C, computed from patch *moments*:

$$m_{pq} = \sum_{xy} x^p y^q I(x, y)$$

with $I(x, y)$ the intensity of a certain pixel with (x, y) coordinates. From the patch moments the centroids are computed as

$$C = (\frac{m_{10}}{m_{00}}, \frac{m_{01}}{m_{00}})$$

From the corner's center O to the centroid a vector $\overrightarrow{OC}$ can be constructed. The orientation of the patch is then given by:

$$\theta = atan2(m_{01}, m_{1,0})$$

A visual illustration of this process can be seen in Figure 3.13. Once calculated the orientation of the patch, it can be rotated to a canonical representation and then compute the descriptor, thus obtaining rotation invariance.

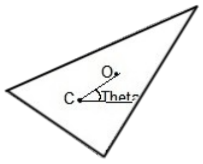

Figure 3.13. ORB computation of patch rotation. Image taken from medium article.

For the description phase, ORB proposes a method to steer BRIEF according to the orientation of the keypoints. It uses the patch orientation θ and the corresponding rotation matrix $\mathbf{R}(\theta)$ to construct a steered version of a $2 \times n$ matrix $\mathbf{S}$, which is built for any feature vector of n binary tests at location (x_i, y_i)

$$\mathbf{S} = \begin{pmatrix} x_1, ..., x_n \\ y_1, ..., y_n \end{pmatrix}$$

The BRIEF operator from Equation 3.32 becomes :

$$f_n(p) = f_n(p)|(x_i, y_i) \in \mathbf{S}_\theta \tag{3.33}$$

A lookup table of precomputed BRIEF patterns is constructed. As long at the keypoint orientation θ is consistent across views, the correct set of points $\mathbf{S}_\theta$ will be used to compute its descriptor. In addition, ORB runs a greedy search among all possible binary tests to find the ones that have both high variance in order to have non-correlation property, since high variance makes a feature more discriminative. The result is called rBRIEF.

Nowadays, ORB keypoint detector and descriptor is still used for many computer vision application since the outperforming efficiency. However, in many cases efficiency can be considered less important than accuracy, depending on time requirements, and some other algorithms are preferred. A very interesting study which shows this trade-off between efficiency and accuracy among the available feature detector-descriptors have been done in [29].

3.3.2 Machine learning approach

Machine Learning (ML) is used in the domain of digital image processing to solve difficult problems (e.g. image colorization, classification, segmentation and detection). In particular, Deep Learning (DL) methods, such as Convolutional Neural Networks (CNNs), mostly improve prediction performance and have pushed the boundaries of what was possible. Problems which were assumed to be unsolvable are now being solved with super-human accuracy. Image classification is a prime example of this. In the feature extraction field, using ML techniques has proven to be very effective compared to the traditional techniques, as we will see also in the experiments' results.

SuperPoint Architecture

The neural network used as alternative to traditional approaches is SuperPoint, a fully-convolutional neural network developed by DeTone et al. in [30] aimed at both detecting interest points and providing their corresponding fixed-length descriptors from an image in a single forward pass. The model has a single, shared encoder to process and reduce the input image dimensionality. After the encoder, the architecture splits into two decoder which learn task specific weights for (1) interest point detection and (2) for interest point description. The architecture can be seen in Figure 3.14.

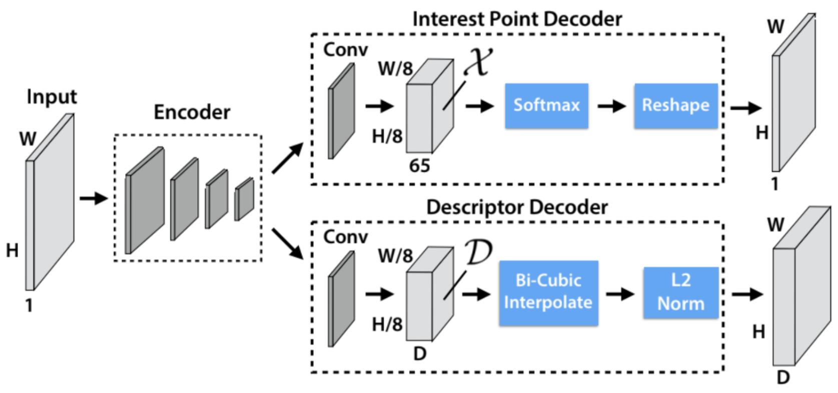

Figure 3.14. SuperPoint Decoders. Image courtesy of [30].

The specific task of each encoder/decoder is summarized below:

- **Shared Encoder**. The encoder consists of convolutional layers, spatial downsampling via pooling and non-linear activation functions. It maps the input image $I \in \mathbb{R}^{H \times W}$ to an intermediate tensor $B \in \mathbb{R}^{Hc \times Wc \times F}$ with smaller spatial dimension and greater channel depth.

- **Interest Point Decoder**. The interest point detector head computes $X \in \mathbb{R}^{Hc \times Wc \times 65}$ and outputs a tensor sized $\mathbb{R}^{H \times W}$. The 65 channels correspond to local, non-overlapping 8×8 grid regions of pixels plus an extra "no interest point" dustbin. After a channel-wise softmax, the dustbin dimension is removed and a reshape from $\mathbb{R}^{Hc \times Wc \times 64}$ to $\mathbb{R}^{H \times W}$ is performed.

- **Descriptor Decoder**. The descriptor head computes $D \in \mathbb{R}^{Hc \times Wc \times D}$ and outputs a tensor sized $\mathbb{R}^{H \times W \times D}$. First a semi-dense grid of descriptors (e.g., one every 8 pixels) is learned. Learning descriptors semi densely rather than densely reduces training memory and keeps the run-time tractable. The decoder then performs bi-cubic interpolation of the descriptor and then L2-normalizes the activations to be unit length.

The final loss is the sum of two intermediate losses: one for the interest point detector, $\mathcal{L}_p$, and one for the descriptor, $\mathcal{L}_d$. A pairs of synthetically warped images is used, which have both (i) pseudo-ground truth interest point locations and (ii)

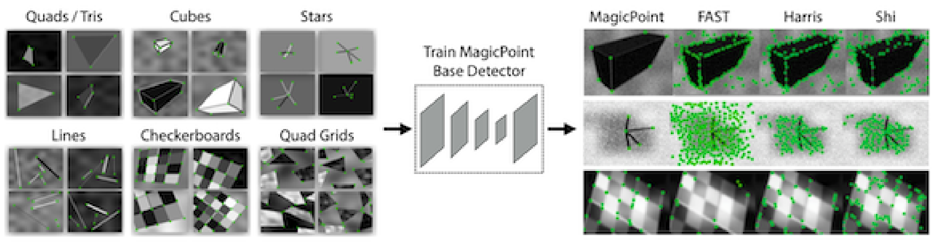

Figure 3.15. Synthetic Shapes dataset is used to train the MagicPoint convolutional neural network, which is more robust to noise when compared to classical detectors. Image courtesy of [30].

the ground truth correspondence from a randomly generated homography $\mathcal{H}$ which relates the two images. This allows to optimize the two losses simultaneously, given a pair of images

$$\mathcal{L}(X, X', D, D'; Y, Y', S) = \mathcal{L}_p(X, Y) + \mathcal{L}_p(X', Y') + \lambda\mathcal{L}_d(D, D', S)$$

where λ is used to balance the final loss and S denotes the entire set of correspondences for a pair of images.

The training phase is performed with a base detector called *MagicPoint* which is used in conjunction with *Homographic Adaptation* to generate pseudo-ground truth interest point labels for unlabeled images in a self-supervised fashion.

Synthetic Pre-Training

Since there are no large datasets of interest point labeled images available, the authors created a large-scale synthetic dataset called *Synthetic Shapes* that consists of simplified 2D geometry via synthetic data rendering of quadrilaterals, triangles, lines and ellipses. In this dataset, it is easy to remove label ambiguity by modeling interest points with simple Y-junctions, L-junctions, T-junctions as well as end points of line segments. Homographic warps is applied to each image to augment the number of training examples, and then the SuperPoint architecture is used to train MagicPoint. Test results on the synthetic dataset overcome with a large gap traditional detectors such as FAST or Harris, as can be seen in Figure 3.15.

Homografic Adaptation

Unfortunately the performance did not generalize well on real world images, and this motivated the self-supervised approach for training on real-world images which is called Homographic Adaptation. At the core of the process is the application of random homographies to warped copies of the input image and combination of the results. Homographies give exact or almost exact image-to-image transformations

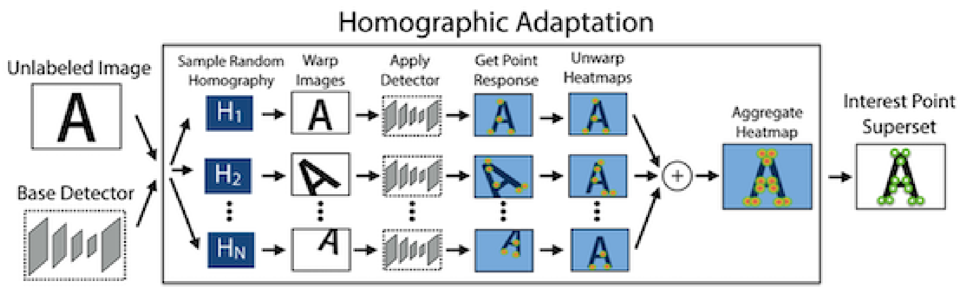

Figure 3.16. Homographic Adaptation is a form of self-supervision for boosting the geometric consistency of an interest point detector trained with convolutional neural networks. Image courtesy of [30].

for camera motion with only rotation around the camera center, scenes with large distances to objects, and planar scenes. Moreover, because most of the world is reasonably planar, a homography is a good model for what happens when the same 3D point is seen from different viewpoints. Because homographies do not require 3D information, they can be randomly sampled and easily applied to any 2D image. The basic idea behind Homographic Adaptation is to perform an empirical sum over a sufficiently large samples of random $\mathcal{H}$'s as in Figure 3.16. Let f_θ represent the initial interest point function to adapt, I the input image, x the resulting interest points and $\mathcal{H}$ a random homography, such that $x = f_\theta(I)$. An ideal interest point operator should be covariant with respect to homographies, namely $\mathcal{H}x = f_\theta(\mathcal{H}(I))$. The resulting aggregation over samples thus gives rise to a new and improved, super-point detector

$$\hat{F}(I, f_\theta) = \frac{1}{N_h} \sum_{i=1}^{N_h} \mathcal{H}_i^{-1} f_\theta(\mathcal{H}_i(I))$$

Homographies are chosen among translation, scale, in-plane rotation, and symmetric perspective distortion using a truncated normal distribution, as illustrated in Figure 3.17.

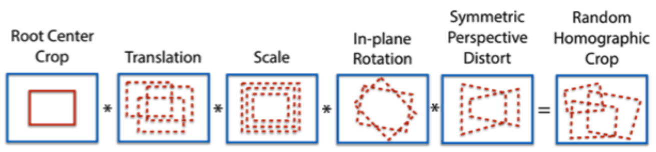

Figure 3.17. Random Homography Generation. The random homographies are generated as the composition of less expressive, simple transformations. Image courtesy of [30].

Performances of SuperPoint have been tested on a patches dataset, comparing repeatability and matching abilities with well known traditional detector and descriptor systems such as SIFT and ORB, showing a consistent improvement.

Chapter 4

Pipeline

The method developed to estimate the odometry of a car equipped with a 3D laser scanner will be described in detail in this chapter. As mentioned, the method is part of a SLAM system, namely the front-end part. The sensor measurements density, i.e. the point clouds, allows reconstructing corresponding two-dimensional high-resolution representations. The latter can be manipulated by computer vision techniques to extract interesting points for proceeding with point cloud registration, which serves to reconstruct the car odometry. The different methods presented in Chapter 3 have been used in order to make a final comparison on performances.

4.1 Projection

The data provided by the sensor requires pre-processing before it can be used for analysis. The goal is to examine the original point cloud in order to extract a sparse one, which contains only features, so that the algorithm used can be more efficient by handling a reduced amount of data. In particular, it is necessary to extract an intensity image from the point cloud provided by the sensor: this is made possible by a projection function $\Pi : \mathbb{R}^3 \to \mathbb{R}^2$, originally proposed in [31]. The spherical projection function Π converts point clouds Cartesian coordinates (x, y, z) into spherical coordinates (r, θ, φ), with radial distance r, azimuth angle θ and polar angle φ:

$$
\begin{aligned}
r &= \sqrt{x^2 + y^2 + z^2} \in \mathbb{R} \\
\theta &= atan2(y, x) \in [-\pi/2, \pi/2] \\
\varphi &= asin(z/r) \in [-\pi, \pi]
\end{aligned}
$$

Assuming that laser beams are uniformly distributed in the vertical field of view (FoV), the spherical coordinates can be projected into image coordinates (u, v) as

$$\begin{pmatrix} u \\ v \end{pmatrix} = \begin{pmatrix} \frac{W}{2}(1 + \frac{\theta}{\pi}) \\ H\frac{f_{up} - \varphi}{f_{up} - f_{down}} \end{pmatrix}$$

where W and H are respectively the width and height of the image and $f = f_{up} + f_{down}$ is the vertical FoV of the sensor. It may happen that multiple points fall in the same image pixel, in which case only the value having the smallest radial distance is retained. In each pixel, the intensity value i_n is normalized and stored.

Another issue is that some pixel could result empty, for instance when the laser beam hits an object which is outside the sensor range. In such cases an empty pixel is set, in order to take it into account for the next step of feature extraction. The correspondences between 2D image points and 3D cloud points are also stored for further needs. An example of LiDAR projection can be seen in Figure 4.1.

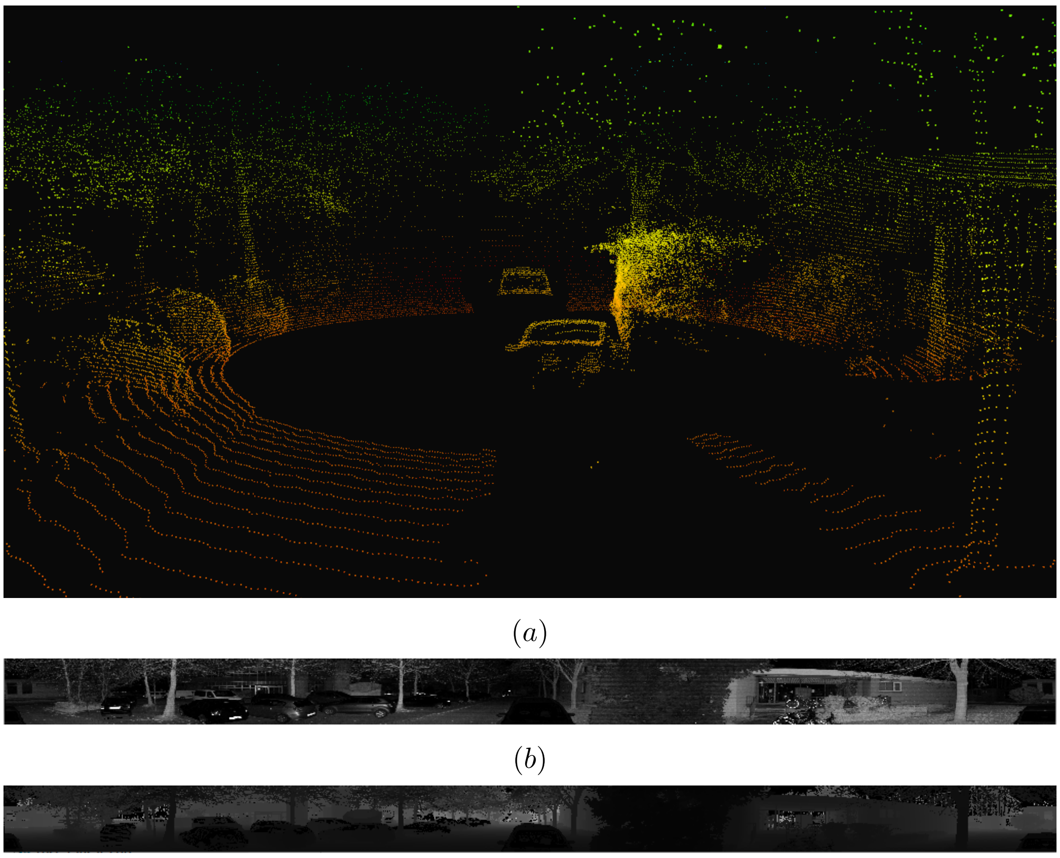

Figure 4.1. 3D point cloud projected onto a 2D image through the change of coordinate formulas. Figure (a) shows the original point cloud visualized into rviz package of ROS. Figure (b) is the intensity of the LiDAR projection, while (c) is the respective depth.

4.2 Extraction

The feature extraction phase involves keypoints detection and description carried out by the different methods presented in Section 3.3.

Detection

Algorithm 5 Non-Maximum Suppression

```
Require: keypoints $K$, image height and width $W, H$, nms distance $nms\_dist$
Ensure: filtered keypoints $\hat{K}$
  $grid \leftarrow 0_{H \times W}, \quad \hat{K} \leftarrow empty$
  for $keypoint\ in\ K$ do
      $grid.at(keypoint) \leftarrow 1$
  end for
  $K.sortByConfidence()$
  for $keypoint\ in\ K$ do
      $x \leftarrow keypoint.x + nms\_dist, \; y \leftarrow keypoint.y + nms\_dist$
      if grid.at(x,y) $= 1$ then
          $grid.block(keypoint) \leftarrow 0$
          $grid.at(x, y) \leftarrow -1$
      end if
  end for
  for $r\ in\ grid.rows$ do
      for $c\ in\ grid.cols$ do
          if $grid.at(r, c) = -1$ then
              $\hat{K}.append(keypoint(c - nms\_dist, r - nms\_dist))$
          end if
      end for
  end for
  return $\hat{K}$
```

The first step requires finding the keypoints in the LiDAR projected 2D image. For this, SuperPoint and the oriented version of FAST were used. The neural network was loaded with pre-trained weights, obtained by training SuperPoint on real world images, as stated in the work of [30]. Being a fully convolutional network is an advantage, because no requirement on image dimensions are requested, even with irregular images like the ones produced in the projection phase, which have a really small height compared to width. This has been of relevant importance, since if a fixed size was requested, scaling and shrinking the projected images, would have led to a loss of information.

A problem that could arise from the SuperPoint neural network is the multiple feature detection, i.e. when the network outputs very close and similar, but distinct, keypoints. In order to avoid this issue, the operation of Non-Maximum Suppression (NMS) has been done, which is shown in Algorithm 5. For each keypoint, the algorithm returns the probability that it is effective by filtering the clusters of keypoints, and selecting only the one that has the highest probability for each cluster. The oriented FAST instead, detects keypoints as described in Section 3.3.1.

Description

In the second step the keypoints' descriptors are computed according to the SuperPoint decoders and rBRIEF described in the previous chapter. They associate for each keypoint a vector of 256 floats, and a vector of 128 bits, respectively. The vector type is crucial for some operations that concern the tracking phase, but it also influences the performance. Some examples of the extraction phase are shown in Figure 4.2.

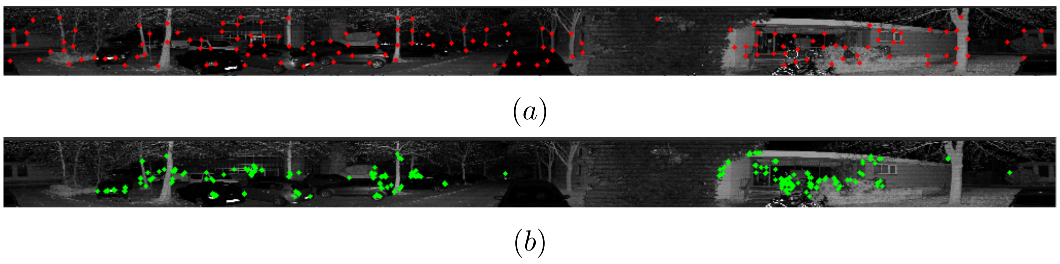

Figure 4.2. Feature extraction on one LiDAR projected image. Figure (a) shows the keypoints detected by the SuperPoint neural network, while figure (b) shows the ones detected by ORB. It can be seen the highest sparsity of the former with respect to the latter. The ORB features are focused on small patches of the image, leading to a more difficult tracking stage.

4.3 Tracking

Once we have obtained the sparse cloud from feature extraction of the current frame, the next step is to estimate the transform between this and the previous one. To this end we need first to solve the data association problem which can be stated as follows: given the current and previous sparse cloud, the goal is to associate the keypoints each other possibly discarding false positives. To solve this, each descriptor of the current and previous sparse cloud have been matched using a *brute-force* approach along with the k-nearest neighbours algorithm, to get the $k = 2$ best matches. After that, for each match, a ratio test is performed in order to choose the good one, as explained by D. Lowe in [25]

$$L(m_{k=1}, m_{k=2}) = \begin{cases} \text{keep} & \text{if } m_{k=1}.distance < m_{k=2}.distance * \text{ threshold)} \\ \text{discard} & \text{otherwise} \end{cases}$$

Lowe's test between two matches checks that the two distances are sufficiently different. If they are not, then the keypoint is eliminated and will not be used for further calculations.

Using a brute-force approach might generally lead to wrong associations, especially with dense measurements, but since the cloud is sparse it is a reasonable choice. This assumption certainly holds for SuperPoint features, but it cannot be said the same for ORB features, because, for the specific images projected from the LiDAR measurements, the keypoints tend to cluster in a few image patches, as can be seen in Figure 4.2.

A second check is done using geometrical information instead of descriptive ones. In particular, the Euclidean 2D norm between the candidate matches is computed. If the norm is lower than a certain threshold then the match is kept, otherwise not. This is particularly helpful when the descriptors are really similar, which may happen in different darker or lighter areas where pixels have a high probability to be matched. While Low's test would probably keep them, the Euclidean one would discard them if they are located in too far coordinates of the image.

Figure 4.3 illustrates how the same features can be tracked in multiple consecutive frames and be uniquely identified.

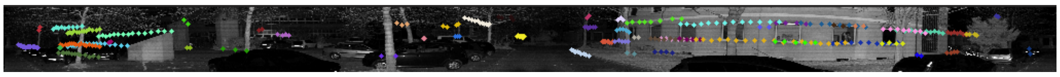

Figure 4.3. SuperPoint feature tracking between consecutive LiDAR frames. Each coloured point represent a keypoint tracked from previous frames and still detected in the current one.

4.4 Registration

Now that we have the 2D associations between consecutive LiDAR frames, they can be registered. It is easy to recover the 3D points from the 2D tracked features, since the 2D-3D correspondences have been previously stored. Given two consecutive sparse point cloud correspondences, $\left\{\left\langle p_0^{prev}, p_{j(0)}^{curr}\right\rangle, ..., \left\langle p_K^{prev}, p_{j(K)}^{curr}\right\rangle\right\}$, of which points have been already matched and tracked, the goal is to estimate a transform $\mathbf{T} = [\mathbf{R}|\mathbf{t}] \in \mathbb{R}^{4\times4}$ such that the sparse point clouds are aligned as $p^{curr} = \mathbf{T}p^{prev}$, where $\mathbf{R}$ and $\mathbf{t}$ denote the rotational and translational part of the transform.

The methods to solve this problem have been discussed in Section 3.2.3. Two different strategies have been followed for estimating the transform:

1. **IRLS**. This is the classical ICP derivation from Gauss-Newton algorithm with robust estimators, which, assuming to have a good initial guess, refines iteratively the transform. The Since the trajectory estimation problem is tackled from a local point of view, it is reasonable to set the initial transform as the identity matrix $\mathbf{I} \in \mathbb{R}^{4\times 4}$, having an identity rotation and zero translation vector. The outlier rejection is carried out by the *Threshold* robust function $\rho(u) = u \cdot threshold$. The procedure is reported in Algorithm 3.

2. **Closed-form LS & RANSAC**. This is the linear relaxation version of the ICP formulation which exploits the SVD decomposition to solve the least-squares problem. No initial guess is required. The outlier rejection is carried out by RANSAC, which iteratively select 3 random point correspondences from the associated sparse clouds and run Algorithm 2. The transform is therefore evaluated, and the inliers score determines its accuracy. The procedure is reported in Algorithm 4.

An example of cloud registration can be seen in Figure 4.4.

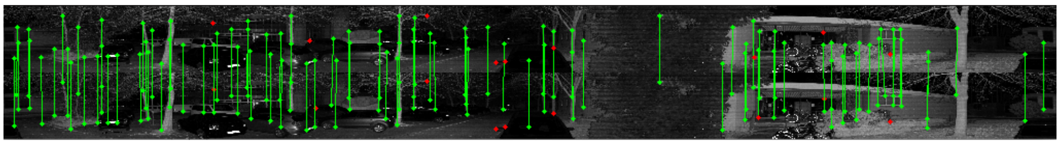

Figure 4.4. Point correspondences between two consecutive sparse point clouds. The green points are selected as inliers and therefore associated with a straight green line. The red points are discarded as identified as outliers. Then the transform is estimated via IRLS or closed-form LS.

4.5 Odometry Estimation

Finally, we come to the goal of the project, the odometry estimation. Since the LiDAR sensor is rigidly equipped on the top of the car, we can assume that changes in orientation and translation are transmitted to the car itself. We have already mentioned that the estimation is local, so the initial transform is the identity $\mathbf{I}_{4\times 4}$. From the second frame the transforms are estimated as soon as a new point cloud measurement comes. The odometry is computed as

$$\mathbf{T}_k^0 = \mathbf{I}_{4\times 4}\ \mathbf{T}_1^0\ \mathbf{T}_2^1\ \dots\ \mathbf{T}_k^{k-1}$$

where $\mathbf{T}_i^j$ is the transform in frame i with respect to frame j. Therefore, the accumulated transform $\mathbf{T}_k^0$ will contain the odometry estimated from the initial frame 0 to the final frame k, i.e. the orientation $\mathbf{R}_k^0$, which represents the steering angle of the car, and the translation vector $\mathbf{t}_k^0$, which represents the trajectory of the car. An example of estimated odometry can be seen in Figure 4.5.

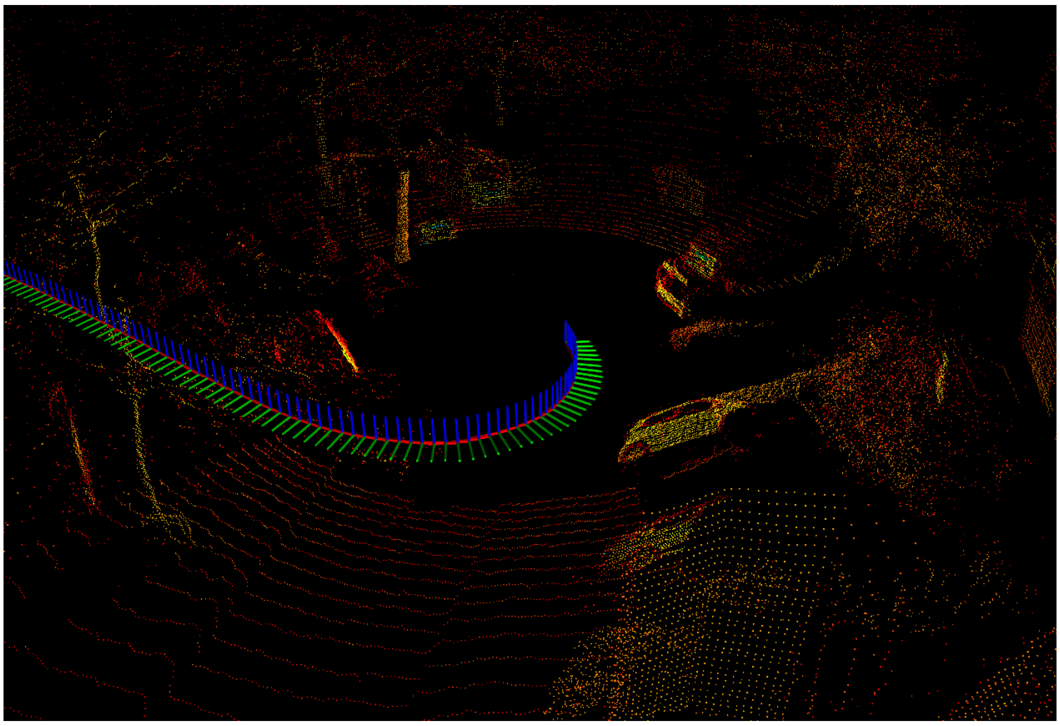

Figure 4.5. An example of estimated odometry. The transform frames accumulated can be clearly seen in the image.

Chapter 5

Experiments

In the following chapter the aim is to validate the presented approach highlighting the differences according to the different methods used, in terms of accuracy, performance, and application contexts.

5.1 System

The project has been developed in C++ on the operating system Ubuntu 20.04. The computer has a 2.3 GHz 8-Core Intel Core i9 processor, a 16 GB 2667 MHz DDR4 memory, and no graphic card has been exploited in order to run the pipeline. However, even though using a neural network, in the Section 5.3, it is shown that real-time performances have been achieved.

5.1.1 LiDAR hardware

The specific Lidar sensor used for the presented work is the Ouster OS1-64: it provides a fixed frame resolution and uses open source drivers. In particular, OS1 has a vertical symmetrical field of view of 33.2° and a horizontal field of view of 360°, with a vertical resolution of 64 channels and horizontal of 1024, which translates into a point cloud of 65536 points per frame. The sensor provides data at 10Hz.

5.1.2 Robot Operating System

The project was developed using the open-source robotics middleware Robot Operating System (ROS) Noetic. It provides services designed for a heterogeneous computer cluster such as hardware abstraction, low-level device control, implementation of commonly used functionality, message-passing between processes, and package management. A typical ROS environment setup includes building a *package*, which is the project itself, collecting a bunch of *nodes*, which communicate each other by publishing or subscribing to some *topics*. In order to simulate real-time sensor measurements acquirement, often datasets are provided (or can be recorded) as *bag* files, which publish different types of information, such as sensor measurements, timestamps, or messages, on different topics.

The ROS architecture of the project is summarized in Figure 5.1.

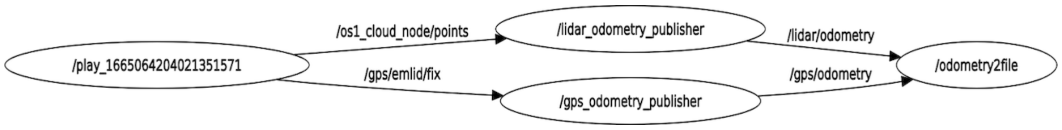

Figure 5.1. ROS graph showing the active nodes and the topics they are publishing/subscribing to.

On the leftmost part, the node *play..*, which is a bag file containing the dataset, publishes (i) on the topic $/os1\_cloud\_node/points$, the OS-1 LiDAR point cloud measurements, and (ii) on the topic $gps/emlid/fix$, the geodetic coordinates (latitude and longitude), at a frequency of $10Hz$. The core node of the project is $lidar\_odometry\_publisher$, which encloses the whole odometry estimation pipeline, it is subscribed to the bag cloud points, and publishes the odometry on the topic $/lidar/odometry$. The other parallel node instead, $gps\_odometry\_publisher$, converts the GPS coordinates into Euclidean ones as explained in the next section, it is subscribed to the bag GPS coordinates and publishes on $/gps/odometry$. Last, the $odometry2file$ node, subscribed to the odometry topics, is responsible for synchronizing the two measurements and aligning the resulting trajectories. Finally, the trajectories are written in a text file and can be used to plot and visualize them.

5.2 Dataset

The method has been tested on the IPB Car dataset, provided by the Institute of Geodesy and Geoinformation of the University of Bonn. The dataset was acquired in an urban environment, specifically in the city of Bonn, using a custom-made sensor system. The latter consists of a vehicle on which are mounted camera, LiDAR equipped with IMU and GNSS sensors. Figure 5.2 shows the described system. There are many dynamic objects in the scene, such as pedestrians, bicycles, and other vehicles, which introduce some noise. The recorded sequences include stopping and accelerating moments, moreover, the dataset contains multiple visits of the same place, to give the possibility to test systems that include loop closure. The dataset includes 18 sequences of approximately one minute each. Each sequence is provided in a binary rosbag format; each file can be played in the ROS system so as to publishing every sensor measurement on different topics.

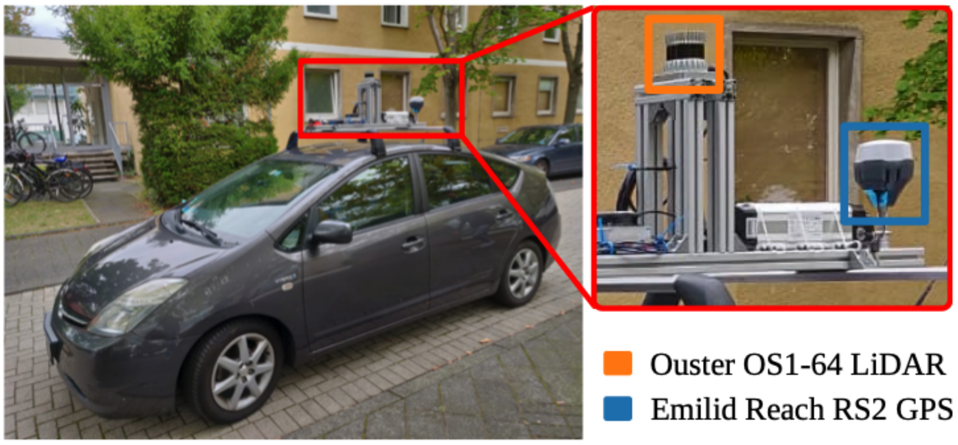

Figure 5.2. Sensor setup used for data recording: Ouster OS1-64 LiDAR sensor plus GNSS information from a Emilid Reach RS2.

5.2.1 GNSS conversion

In order to use the GNSS data for comparison with the trajectory estimated by the system, it is necessary to translate the data provided in *geodetic* coordinates format $(long, lat)$ into Cartesian coordinates (x, y, z). Since the coordinate system is placed in the Heart's center, we are specifically talking about Earth-Centered Earth-Fixed (ECEF) coordinates, also known as the *geocentric* coordinate system. The GPS sensor provide the *geodetic latitude*, which is defined as the angle between the equatorial plane and the surface normal at a point on the ellipsoid. Finding the ECEF coordinates require the *geocentric latitude*, which is defined as the angle between the equatorial plane and a radial line connecting the centre of the ellipsoid to a point on the surface. The difference between the two can be seen in Figure 5.3. Following the World Geodetic System standard WGS84, it is possible to convert the geodesic to geocentric latitude, defined as $\mathbf{gLat}(lat)$ and find the distance of a point on the Heart's surface $\mathbf{P}$ from the Heart's center $\mathbf{O}$, defined as $R_{Eeart}(\mathbf{P})$.

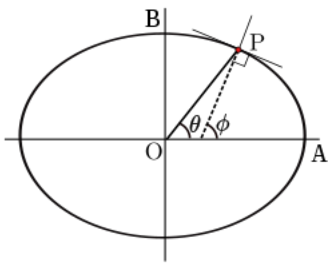

Figure 5.3. The definition of geodetic latitude (ϕ) and geocentric latitude (θ). Image taken from Wikipedia.

With these geometric relationships, the conversion from geodetic to ECEF coordinates is calculated as

$$
\begin{aligned}
x &= R_{Heart}(\mathbf{P}) \cdot cos(long) \cdot cos(\mathbf{gLat}(lat)) \\
y &= R_{Heart}(\mathbf{P}) \cdot sin(long) \cdot cos(\mathbf{gLat}(lat)) \\
z &= R_{Heart}(\mathbf{P}) \cdot sin(\mathbf{gLat}(lat))
\end{aligned}
$$

Since the GPS sensor provides only translational data, the comparisons have been done using only the estimated trajectory.

Feature Extraction			**Tracking**		
SuperPoint	nFeatures	300	BF Matcher	knnThreshold	0.7
	threshold	0.1		normType	L2/H
	nmsBlockSize	6		normThreshold	30
FAST	threshold	30	**Registration**		
ORB	nFeatures	300	IRLS	iterations	1
	scaleFactor	1.1		kernelThreshold	5e-5
	nLevels	8		damping	0.5
	edgeThreshold	15	RANSAC	iterations	30
	patchSize	15		inliersThreshold	20cm

Table 5.1. System parameters of the different methods used. For the feature extraction methods, the parameters have been chosen inspired by [14]. Tracking parameters have been decided according to the best experiment results obtained on the different methods. Norm type L2 and H stand for Euclidean norm in the case of float descriptors (SuperPoint), and Hamming distance, i.e. XOR operation, for binary descriptors (ORB). For the registration parameters, the number of iterations have been chosen according to a convergence criterion: as soon as the new estimate is too close from the previous, the algorithm stops. The average number of iterations reported, have been found after many experiments. The thresholds have been set to a reasonable value in order to reject as much as possible outliers, while not removing too many correspondences which leads to poor estimations.

5.3 Results

The experiments have been done with the first 10 sequences of the IPB Car dataset, evaluating performances in terms of computational time and accuracy. As already mentioned, the GNSS measurements provide only translational information, therefore we can just evaluate the estimated trajectory of the car. To measure the accuracy of the system, we can make use of the Root Mean Square Error (RMSE) as metric. The RMSE is defined as

$$RMSE = \sqrt{\frac{1}{k}\sum_{i=1}^{k} ||\hat{\mathbf{t}}_k - \mathbf{t}_k^{GNSS}||^2}$$

Since the two estimated trajectories are defined with respect to different reference frames, before comparing them it is necessary to perform an initial alignment between the two, with the help of Algorithm 2, which calculates a rigid body transformation to align the two trajectories. The system parameters set for each method, are reported in Table 5.1. The plots of the trajectories are shown in Figure 5.4 and Figure 5.5. Their evaluation are summarized in Table 5.2.

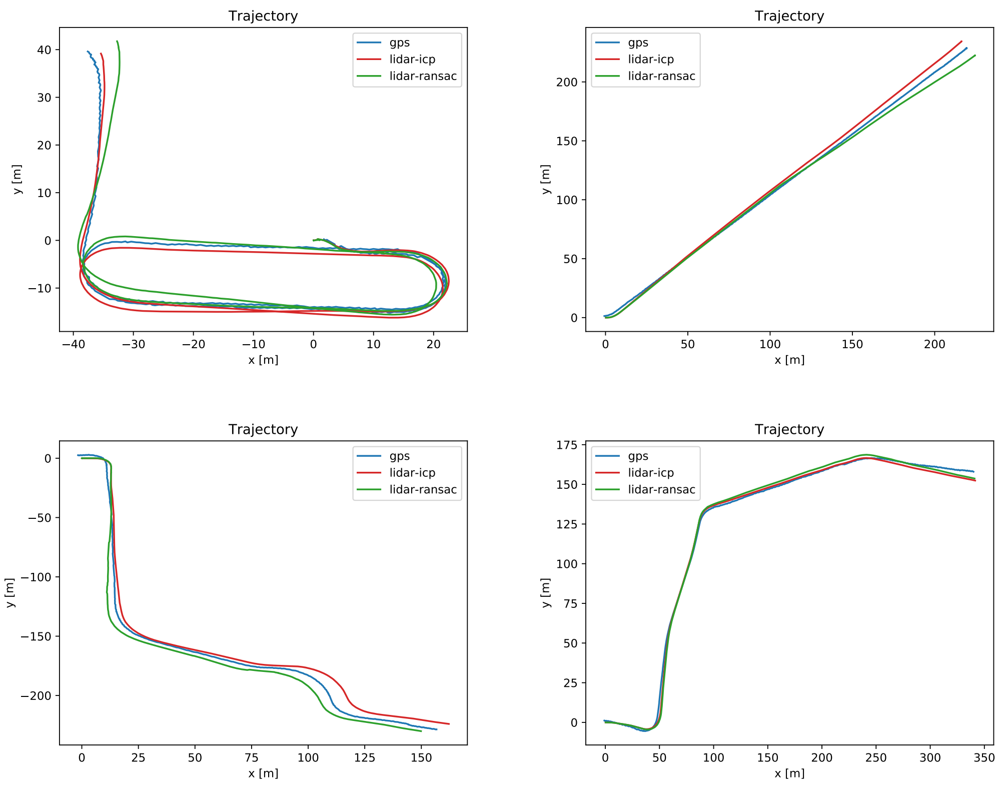

Figure 5.4. Sequences from 1 to 4 of IPB Car Dataset trajectories, with IRLS, RANSAC and GNSS estimations.

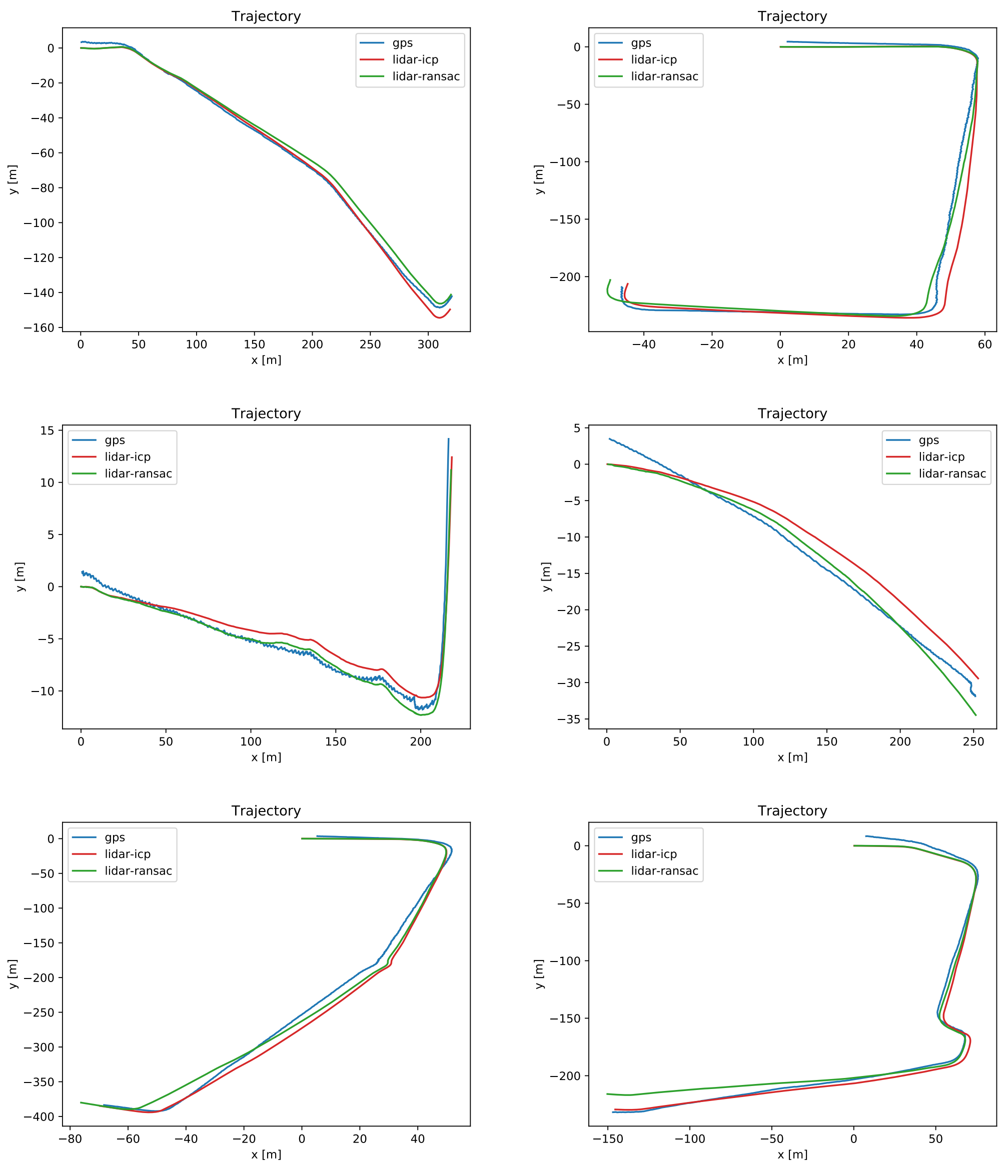

Figure 5.5. Sequences from 5 to 10 of IPB Car Dataset trajectories, with IRLS, RANSAC and GNSS estimations.

	00	**01**	**02**	**03**	**04**	**05**	**06**	**07**	**08**	**09**
IRLS	1.31	1.07	1.21	1.23	1.47	1.51	0.85	1.06	1.85	1.69
RANSAC	1.29	0.88	1.69	1.07	1.06	1.41	0.87	0.87	1.71	1.65

Table 5.2. RMSE of estimated trajectories using SuperPoint as feature extractor, and comparing the registration methods.

5.4 Comparison

Let's first take a look on the differences between the two feature extraction methods, SuperPoint and ORB. Both the trajectories have been estimated using the same registration method, IRLS, in order to enhance the feature extraction part.

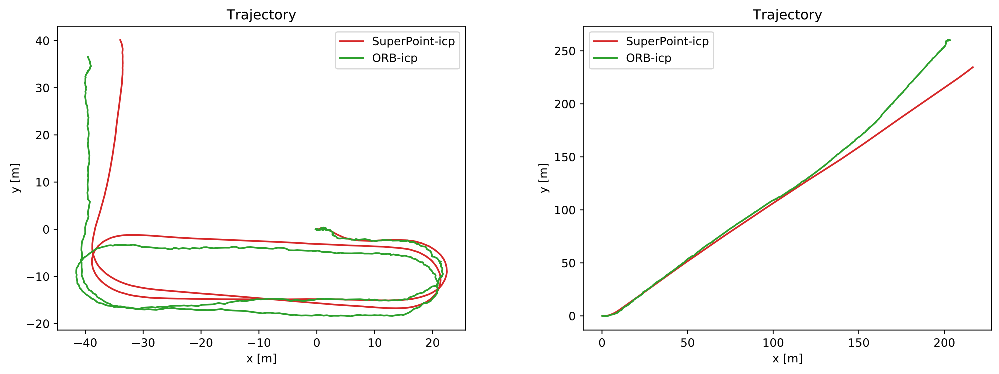

Figure 5.6. Comparison between ORB and SuperPoint trajectories both using IRLS, on IPB Car sequences 00 and 01.

As can be seen in Figure 5.6, SuperPoint clearly overcomes ORB in accuracy. In fact, the RMSE errors are 1.31 (00), 1.07 (01), and 13.56 (00), 8.61 (01), respectively. This gap is due to the nature of the projected LiDAR images. As already seen in Figure 4.2, the keypoints detected by SuperPoint are highly sparse and repeatable frame by frame. On the contrary, ORB detects interesting points in a poorly distributed and concentrated way. With this behavior, distinguishing correspondences between frames becomes difficult, and the tracking system gives very few inliers, resulting in poor estimates and also to the absence of a transformation. The latter, causes the shaking pattern and the distortion of the trajectories. Regarding speed performances instead, the system with SuperPoint architecture runs at 20 Hz, while with ORB runs at 110 Hz. This can be easily explained by the fact that the neural network approach needs a graphic card in order to run at its maximum performance, and, depending on the capacity, it might surpass the traditional approach. Still, remembering that the LiDAR sensor provides measurements at a frequency of 10 Hz, both the systems are valid to process the data, and can be considered equal in performance from a requirements point of view. However, since this project aims to build only the front-end system of a SLAM architecture, including other pieces like a back-end (loop closure) or a mapping construction systems, certainly lead to lack of computational resources and therefore loss of real-time performances. These premises are different for ORB extractor, since its performance allows multiple SLAM systems to run together in the same application with the only use of a CPU, as also demonstrated by the ORB-SLAM architecture [3]. Having said so, and assuming to have a good system which provides a graphic card allowing parallelism in favor of neural networks, the focus in the experiments moved on highlighting the differences between the two registration methods, IRLS and RANSAC, using only SuperPoint.

Iterations on both the algorithms have been chosen according the following convergence criterion:

$$iterations = \begin{cases} \text{continue} & \text{if } d(\mathbf{T}_{k-1}, \mathbf{T}_k) > \delta \\ \text{stop} & \text{otherwise} \end{cases}$$

where d is the distance between the previous and the current transforms:

$$d(\mathbf{A}, \mathbf{B}) = \sum_{i=1}^{n} \sum_{j=1}^{m} |a_{ij} - b_{ij}|$$

and δ is a threshold value. The distance between the two transforms has been used in order to avoid additional iterations, which consume too many resources and get very small improvements, and keeping the real-time performances. The optimal number of iterations have been reported in Table 5.1. The first thing we should notice is that while RANSAC improves over iterations, given its probabilistic nature, IRLS converges in only 1 iteration, raising the performances. In fact, IRLS, with 20 Hz frequency, overcomes RANSAC which runs at 15 Hz. Parameters tuning has been done mainly with the number of iterations for RANSAC, and the kernel threshold for IRLS. In Figure 5.7 are shown some trajectories obtained by varying those parameters.

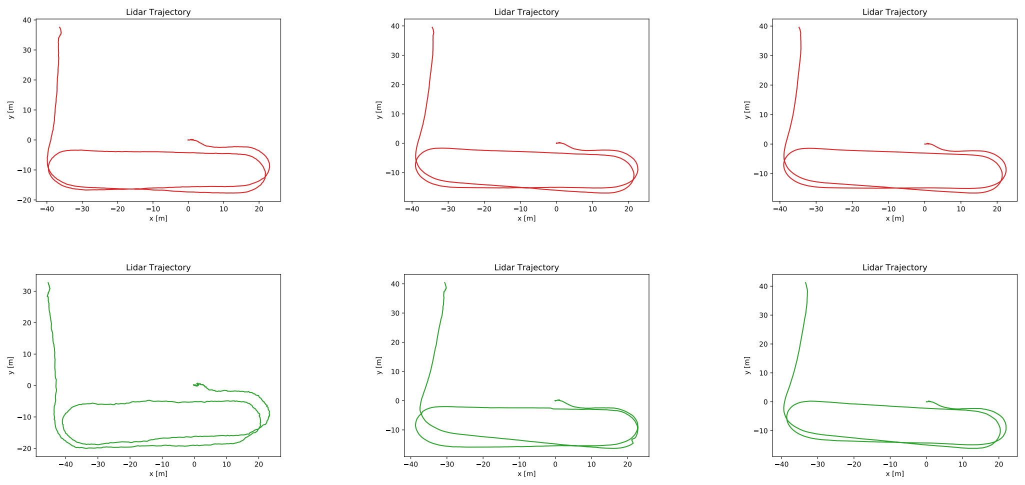

Figure 5.7. IBP car sequence 00 trajectories. In the first row, IRLS with $5e^{-1}$, $5e^{-2}$, and $5e^{-3}$ kernel thresholds. In the last row, RANSAC with 5, 10, and 20 iterations.

Another interesting aspect is that, looking at Table 5.2, RANSAC almost always overcomes IRLS in terms of accuracy. This means that the consensus method, given a sufficient number of iterations, is more effective than the robust estimator. In fact, according to Equation 3.28, given $k = 30$ iterations, $n = 3$ minimum number of data, and $w = 0.7$ inliers ratio, the probability to get a good solution is:

$$p = 1 - (1 - w^n)^k = 0.9996$$

These results are not a surprise; recent works on odometry estimation and SLAM systems have proved the effectiveness of consensus methods not only in terms of accuracy but also in speed. In [32], the authors estimate odometry by using stereo visual sensors, and to reduce the impact of outliers, a minimum sample is not totally randomly generated. The hypotheses are generated on the order of the features' ages and similarities. Features with larger ages and higher similarities are selected first, which makes the process faster and more effective than standard RANSAC. In [32], a dynamic SLAM system using sensors fusion with stereo visual and 2D LiDAR, the authors use an extended Kalman filter (EKF) with an improved version of RANSAC, called multilevel-RANSAC (ML-RANSAC). The proposed algorithm works in conjunction with a region-based CNN (r-CNN) which classifies stationary and moving objects, allowing a more robust data association. These works are a few examples showing how consensus-based methods can be really fast, achieving impressive results on odometry estimation, with respect to least-squares-based methods.

The last consideration is about application contexts. In this scenario, a local tracking assumption, simplifies the IRLS on the first frame, for which initial guess is an identity, that perfectly fits the initial state of the car. In the next frames, the initial guess is always a good representation of the car state, given the initial convergence. In global localization scenario, the optimal convergence of the IRLS method is no longer guaranteed due to uncertainty, therefore losing accuracy and speed (more than 1 iteration is required) performances. On the contrary, RANSAC does not need an initial state estimation, thus it is more suitable. Last, plugging a beck-end system for a complete SLAM framework, requires loop closure detection and bundle adjustment, which might suffer with lest-squares methods in the absence of a good initial guess. Well known frameworks like ORB-SLAM [3] or DSO with loop closure (LDSO) [33], in fact use RANSAC for computing the transform of candidate loops.

Chapter 6

Conclusions

The presented work focused on the development of a front-end architecture for a SLAM system, specifically a method for odometry estimation based on frame-to-frame matching using LiDAR sensor measurements, and a final comparison on registration techniques, focusing on accuracy and speed, has been done. The experiments have shown that, besides using a CPU, a neural network feature extraction approach as SuperPoint, is fast enough to being used in real-time applications such as odometry estimation, moreover outperforming traditional approaches such the widely used Oriented FAST and Rotated Brief (ORB). Regarding registration, Iterative Reweighed Least-Square (IRLS) version of ICP and Random Sample Consensus (RANSAC), have been compared. Based on the experience of this project and on related works research, it can be concluded that:

- IRLS is faster than RANSAC when a local estimation problem is faced, given its convergence properties when a good initial guess is chosen.
- RANSAC is more accurate than IRLS when a good set of inliers is present, given its probabilistic nature.
- When loop closure is needed for a back-end system, IRLS convergence is no longer valid and RANSAC must be used in order to estimate a correct transform.
- If there is not clear distinction between inliers and outliers RANSAC can't be used by definition.
- If there are plenty of parameters, then the number of data points RANSAC needs to select to build a candidate solution will also be large. This makes significantly less likely to find a subset of inliers in a reasonable number of iterations. An example could be bundle adjustment.

There are different possible future works which can be done in order to improve the system. Regarding the hardware available, one could be the integration of the IMU sensor measurements, which provide prior motion, to have an initial estimate of the relative LiDAR displacement. On the feature extraction, in addition to the intensity, one may exploit the depth information in order to validate keypoints' association consistency. Moreover, other neural network approaches techniques, such as Unsuperpoint [34], could be investigated in order to compare extraction performances on LiDAR projected images. Concerning the registration of the transforms, a possible extension could be estimating between non-successive frames; sometimes sharp steering of the vehicle might compromise the sensor measurements, reason why skipping some frames and perform tracking and registration would result in a better estimate. This can be applied also in conjunction with normal frame-by-frame registration, for checking consistency. Finally, loop closure detection with bundle adjustment would improve the overall performance making the system very precise. For this, visual place recognition techniques on LiDAR projected images should be applied, for instance HBST [13] or DBoW [12], using the feature descriptors extracted from every image.

Bibliography

[1] R. Hartley and A. Zisserman, *Multiple View Geometry in Computer Vision.* New York, NY, USA: Cambridge University Press, 2 ed., 2003.

[2] A. J. Davison, I. D. Reid, N. D. Molton, and O. Stasse, "Monoslam: Real-time single camera slam," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 29, no. 6, pp. 1052–1067, 2007.

[3] R. Mur-Artal, J. M. M. Montiel, and J. D. Tardos, "ORB-SLAM: A versatile and accurate monocular SLAM system," *IEEE Transactions on Robotics*, vol. 31, pp. 1147–1163, oct 2015.

[4] J. Engel, V. Koltun, and D. Cremers, "Direct sparse odometry," *CoRR*, vol. abs/1607.02565, 2016.

[5] J. Zhang and S. Singh, "Loam: Lidar odometry and mapping in real-time," 07 2014.

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

[7] L. Huang, "Review on lidar-based slam techniques," in *2021 International Conference on Signal Processing and Machine Learning (CONF-SPML)*, pp. 163–168, 2021.

[8] N. Jonnavithula, Y. Lyu, and Z. Zhang, "Lidar odometry methodologies for autonomous driving: A survey," 2021.

[9] E. Recherche, E. Automatique, S. Antipolis, and Z. Zhang, "Iterative point matching for registration of free-form curves," *Int. J. Comput. Vision*, vol. 13, 07 1992.

[10] P. Biber and W. Strasser, "The normal distributions transform: a new approach to laser scan matching," in *Proceedings 2003 IEEE/RSJ International Confer-*

ence on Intelligent Robots and Systems (IROS 2003) (Cat. No.03CH37453), vol. 3, pp. 2743–2748 vol.3, 2003.

[11] C. R. Qi, H. Su, K. Mo, and L. J. Guibas, "Pointnet: Deep learning on point sets for 3d classification and segmentation," 2016.

[12] D. Galvez-López and J. D. Tardos, "Bags of binary words for fast place recognition in image sequences," *IEEE Transactions on Robotics*, vol. 28, no. 5, pp. 1188–1197, 2012.

[13] D. Schlegel and G. Grisetti, "HBST: A hamming distance embedding binary search tree for visual place recognition," *CoRR*, vol. abs/1802.09261, 2018.

[14] L. Di Giammarino, I. Aloise, C. Stachniss, and G. Grisetti, "Visual place recognition using lidar intensity information," 2021.

[15] S. Carpin, "Fast and accurate map merging for multi-robot systems," *Auton. Robots*, vol. 25, pp. 305–316, 10 2008.

[16] T. Nam, J. Shim, and Y. Cho, "A 2.5d map-based mobile robot localization via cooperation of aerial and ground robots," *Sensors*, vol. 17, p. 2730, 11 2017.

[17] S. Kato, S. Tokunaga, Y. Maruyama, S. Maeda, M. Hirabayashi, Y. Kitsukawa, A. Monrroy, T. Ando, Y. Fujii, and T. Azumi, "Autoware on board: Enabling autonomous vehicles with embedded systems," in *2018 ACM/IEEE 9th International Conference on Cyber-Physical Systems (ICCPS)*, pp. 287–296, 2018.

[18] C. Cadena, L. Carlone, H. Carrillo, Y. Latif, D. Scaramuzza, J. Neira, I. Reid, and J. J. Leonard, "Past, present, and future of simultaneous localization and mapping: Toward the robust-perception age," *IEEE Transactions on Robotics*, vol. 32, no. 6, pp. 1309–1332, 2016.

[19] G. Grisetti, R. Kümmerle, C. Stachniss, and W. Burgard, "A tutorial on graph-based slam," *IEEE Transactions on Intelligent Transportation Systems Magazine*, vol. 2, pp. 31–43, 12 2010.

[20] G. Grisetti, T. Guadagnino, I. Aloise, M. Colosi, B. Della Corte, and D. Schlegel, "Least squares optimization: from theory to practice," 2020.

[21] K. S. Arun, T. S. Huang, and S. D. Blostein, "Least-squares fitting of two 3-d point sets," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PAMI-9, no. 5, pp. 698–700, 1987.

[22] S. Umeyama, "Least-squares estimation of transformation parameters between two point patterns," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 13, no. 4, pp. 376–380, 1991.

[23] K. Arai and S. Kapoor, eds., *Advances in Computer Vision.* Springer International Publishing, 2020.

[24] C. Harris and M. Stephens, "A combined corner and edge detector," in *Proc. AVC*, pp. 23.1–23.6, 1988. doi:10.5244/C.2.23.

[25] D. Lowe, "Distinctive image features from scale-invariant keypoints," *International Journal of Computer Vision*, vol. 60, pp. 91–, 11 2004.

[26] E. Rosten and T. Drummond, "Machine learning for high-speed corner detection," in *Computer Vision – ECCV 2006* (A. Leonardis, H. Bischof, and A. Pinz, eds.), (Berlin, Heidelberg), pp. 430–443, Springer Berlin Heidelberg, 2006.

[27] M. Calonder, V. Lepetit, C. Strecha, and P. Fua, "Brief: Binary robust independent elementary features," vol. 6314, pp. 778–792, 09 2010.

[28] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski, "Orb: an efficient alternative to sift or surf," pp. 2564–2571, 11 2011.

[29] S. A. K. Tareen and Z. Saleem, "A comparative analysis of sift, surf, kaze, akaze, orb, and brisk," in *2018 International Conference on Computing, Mathematics and Engineering Technologies (iCoMET)*, pp. 1–10, 2018.

[30] D. DeTone, T. Malisiewicz, and A. Rabinovich, "Superpoint: Self-supervised interest point detection and description," pp. 337–33712, 06 2018.

[31] J. Behley and C. Stachniss, "Efficient surfel-based slam using 3d laser range data in urban environments," 06 2018.

[32] M. S. Bahraini, A. B. Rad, and M. Bozorg, "Slam in dynamic environments: A deep learning approach for moving object tracking using ml-ransac algorithm," *Sensors*, vol. 19, no. 17, 2019.

[33] X. Gao, R. Wang, N. Demmel, and D. Cremers, "LDSO: direct sparse odometry with loop closure," *CoRR*, vol. abs/1808.01111, 2018.

[34] P. H. Christiansen, M. F. Kragh, Y. Brodskiy, and H. Karstoft, "Unsuperpoint: End-to-end unsupervised interest point detector and descriptor," 2019.