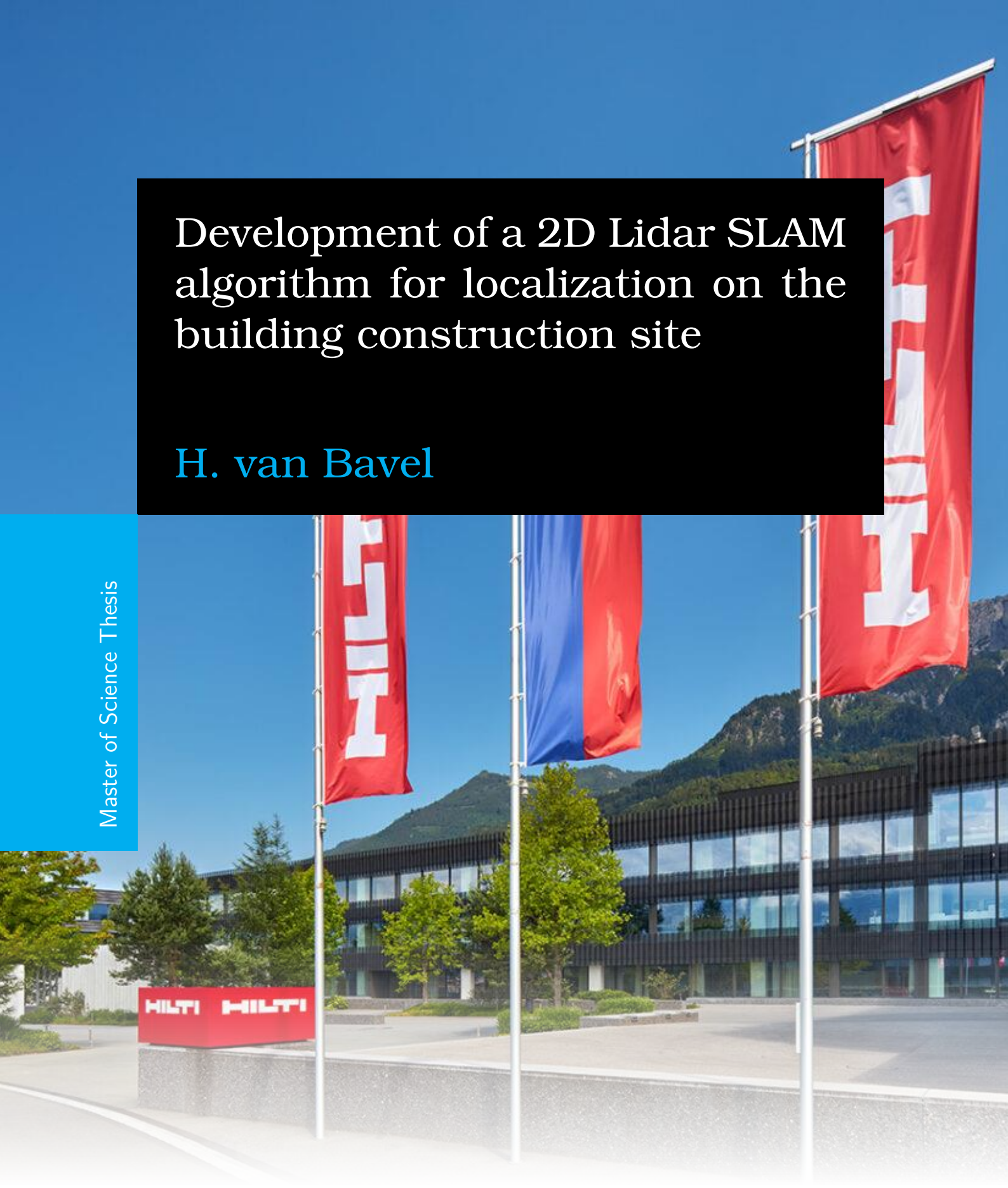
Development of a 2D Lidar SLAM algorithm for localization on the building construction site

H. van Bavel

Master of Science Thesis



TU Delft Delft University of Technology

Delft Center for Systems and Control

Development of a 2D Lidar SLAM algorithm for localization on the building construction site

Master of Science Thesis

For the degree of Master of Science in Systems and Control at Delft University of Technology

H. van Bavel

January 31, 2022

Faculty of Mechanical, Maritime and Materials Engineering (3mE) · Delft University of Technology

The work in this thesis was supported by Hilti Corporation. Their cooperation is hereby gratefully acknowledged.

Abstract

In order to improve the autonomy of construction robots, a Simultaneous Localization And Mapping (SLAM) solution is needed that localizes the robot using solely on-board sensing with sub-5 mm accuracy. As pointed out by the results of the Hilti SLAM Challenge, the state-of-the-art in SLAM currently does not offer a solution that comes close to this requirement. The winning algorithm FAST-LIO2 achieved an accuracy typically around 4-10 cm. When considering the literature, it can be noted that the state-of-the-art SLAM algorithms are designed with generality in mind, and limited attention has been paid to SLAM that focuses on increasing accuracy by targeting a specific scene structure.

In order to improve SLAM accuracy, this thesis proposes a novel 2D Lidar SLAM algorithm focused on indoor environments, hereafter called Ray-SLAM. The algorithm is centered around the assumption that walls are available, which is valid in many cases on the building construction site. Ray-SLAM introduces several novelties which are (1) a sparse map representation to facilitate a joint pose-map optimization scheme, (2) observation-to-map alignment using a non-iterative procedure (contrary to the Iterative Closest Point algorithm) and (3) a stop-and-go strategy to prevent Lidar motion distortion to corrupt the map.

The algorithm was extensively tested using six real-world indoor datasets recorded with the Ouster OS0 Lidar in rooms with various shapes and sizes and with a total trajectory length of 307 m. The motion was constrained to the horizontal plane and a stop-and-go motion pattern was applied. Ground-truth was recorded at static points to evaluate the accuracy. In the majority of the test cases, Ray-SLAM was able to estimate the trajectory successfully. Failure cases led back to either a lack of unoccluded walls in the scene or violated model assumptions about these walls (e.g. incorrect referencing to clutter close to the wall or to a door being opened).

Ray-SLAM's accuracy was compared with two 3D Lidar SLAM algorithms, LOAM and FAST-LIO2 respectively. The compared algorithms solved a harder 6DOF estimation problem, but had access to 64 Lidar scan rings instead of one and (optionally) the Inertial Measurement Unit (IMU). An overall Mean Absolute Error of 5.7 mm is reported by Ray-SLAM in the successful cases, which is 5.0× more accurate than LOAM and 2.4× more accurate than FAST-LIO2. Further research is suggested to improve the robustness of Ray-SLAM and extend it to a full 3D SLAM algorithm.

Table of Contents

1 Introduction **1**
- 1-1 Research motivation . . . 1
- 1-2 Contributions . . . 5
- 1-3 Thesis Outline . . . 6

2 Related Work **9**
- 2-1 Existing work on sensor fusion algorithms for pose estimation . . . 9
- 2-2 Existing work on point cloud handling . . . 11
  - 2-2-1 Normal estimation and local cluster analysis . . . 12
  - 2-2-2 Edge- and planar feature point extraction . . . 13
  - 2-2-3 Point cloud registration using the ICP algorithm . . . 14
- 2-3 Existing work on Lidar-centric localization algorithms . . . 16
  - 2-3-1 Building model based localization . . . 16
  - 2-3-2 Simultaneous localization and mapping . . . 17
- 2-4 Design goals of the proposed localization algorithm . . . 19

3 A novel 2D Lidar SLAM algorithm: Ray-SLAM **21**
- 3-1 State space model and map representation . . . 22
  - 3-1-1 Dynamic model . . . 22
  - 3-1-2 Map representation . . . 23
  - 3-1-3 Lidar measurement model . . . 26
  - 3-1-4 Finding correspondences between the Lidar observations and the map elements . . . 27
- 3-2 Problem formulation and algorithm implementation details . . . 28
  - 3-2-1 Loosely coupled estimation of the map and state . . . 28
  - 3-2-2 Nonlinear estimation of the virtual pose measurement and map. . . . 31
  - 3-2-3 Linear estimation of the state. . . . 34
  - 3-2-4 Spawning new map elements. . . . 35
  - 3-2-5 Finding correspondences between the Lidar observations and the map elements. . . . 41
  - 3-2-6 Summary of the Ray-SLAM algorithm . . . 42

4 Results **45**
4-1 Measurement setup . . . 45
4-2 Datasets . . . 46
4-3 Experiment results and robustness analysis . . . 49
4-3-1 Parameter tuning . . . 49
4-3-2 Analysis of the experiment results . . . 50
4-3-3 Opening a door . . . 54
4-3-4 Conclusions of the experiment results and robustness analysis . . . 55
4-4 Accuracy analysis . . . 56
4-4-1 Trajectory/waypoint alignment procedure . . . 57
4-4-2 Accuracy results . . . 58
4-4-3 Ablation study . . . 60
4-4-4 Remainder of the error on successful datasets . . . 61
4-5 Tuning of beam angle discount factor . . . 63
4-6 Effect of the anchor map elements . . . 64

5 Conclusions and Recommendations **67**
5-1 Conclusions . . . 67
5-2 Future Work . . . 69
5-2-1 Improve Ray-SLAM as 2D SLAM algorithm . . . 69
5-2-2 Extend Ray-SLAM to a full 3D SLAM algorithm . . . 70

A Rotations and transformations **71**

B Ouster OS0 technical specifications **73**

C Ouster OS0 noise properties **75**

Bibliography **77**

Glossary **83**
List of Acronyms . . . 83
List of Symbols . . . 84

List of Figures

1-1 The Hilti Jaibot drilling robot in action. Localization is done using a Total Station, which is shown in the right. 2

1-2 **Left**. The Ouster OS0 Lidar generates 131072 distance measurements per scan, which together form a *point cloud*. This Lidar scans at 10 Hz (see Appendix B). **Middle**. A visualization of the point cloud measurement of a Lidar. **Right**. An Inertial Measurement Unit (IMU). 3

1-3 A high-level flow chart description of the novel Ray-SLAM algorithm that is proposed in this thesis. The steps 'Optimize the map' and 'Extend the map' (marked in green) are carried out only if the robot is perceived to be stationary. 6

2-1 In this toy-Lidar example, five individual laser range measurement sensors are located inside the Lidar with different fixed pitch angles. The five measurements at a single azimuth angle α_t are carried out at the same time and considered a *scan line*. The five range sensors rotate internally within the Lidar and generate 80 scan lines sequentially over time in this example. A *scan ring* is the collection of observed points resulting from one range sensor during one rotation. The observations from all range sensors during one rotation are referred to as a *scan*. The Ouster OS0 Lidar was used for the experiments in this thesis, which has 64 scan rings and 2048 observations per ring (see Appendix B). 12

2-2 Many Lidar-based SLAM algorithms use LOAM-based feature points. On the left the planar feature point is shown, where the arrow points in the normal direction of the local surface. On the right, the edge feature point is shown, where the arrow points into the principal direction of the line that represents the edge. 13

2-3 Different map representations that can be found in existing SLAM works. 19

3-1 A typical scene where a construction robot could be deployed to automate taks such as drilling holes [1]. Walls are available that the robot can use as reference for localization. The environment is cluttered with construction materials. In this scene particular since, a 2D SLAM algorithm can be an appropriate solution since the floor is flat and the walls are upright. 24

3-2 A toy example of a building's cross section. A wall may have curvature and the flat surface can be interrupted by e.g. a fire extinguisher. Line segments (green) are used to model the wall and are referred to as *map elements*. The map elements are placed away from the edge of the wall, the fire extinguisher and each other such that only the (nearly) flat surfaces are used as a references for localization. . 24

3-3 Parameterization of the map elements. The i^{th} map element $\mathcal{M}_t^i$ is shown as a green line segment. The element has a fixed origin $p^i_{\mathcal{M},x}, p^i_{\mathcal{M},y}$ and heading φ^i. The position offset $d^i_{\text{pos},t}$ and heading offset $d^i_{\varphi,t}$ in blue are (unknown) model parameters. All map elements have the same radius s_r for simplicity. 25

3-4 Schematic overview of the reference frames of the SLAM problem and their connections. An example indoor scene is displayed. 26

3-5 **Left.** Schematic representation of the Lidar observations. **Right.** Visual representation of the ray casting function, which provides the measurement prediction $\tilde{y}$ by taking the Euclidean distance between the Lidar's position and the predicted intersection point between the laser beam and map element. 27

3-6 A flow chart description of the Ray-SLAM algorithm using the algorithmic components. Once the initial map is constructed, the algorithm carries out SLAM in a 10 Hz loop. The color green indicates steps that are carried out only if the Lidar is considered to be stationary. The front-end functions $m_{\text{init}}(\cdot)$, $m(\cdot)$ and $k(\cdot)$ have direct access to the raw sensor data $\mathcal{Y}_t$. The estimation problems indicated by $p(\cdot)$ are the back-end of Ray-SLAM. The asterisk at $\mathcal{M}^*_{\sigma,t}$ is placed to indicate a diagonal matrix approximation. 29

3-7 The ray casting laser beam error model vs. the conventional point-to-line error model. The difference between these error terms depend on the beam angle β_t^i. . 32

3-8 The weighting function $\cos(\beta)^{s_\beta}$ allows the user to reduce or increase the weight of observations with a steep inclination angle β using the tuning parameter s_β. Setting the weighting factor $s_\beta = 1.0$ results in the point-to-line error metric, where each measurement residual is weighted such that an observed point results the same residual regardless of incidence angle β. 33

3-9 Visual representation of the linear descriptor value $l_{t,i}$, which can be intuitively interpreted as the width of the smallest oval shape that fits around the local cluster of adjacent points. A cluster size of $N_{\text{clust}} = 5$ is used in this example. The example Lidar scan shows two walls meeting in a corner point, with the north-sided wall being partly occluded. 37

3-10 The selection of three anchor map elements is shown. These elements have a position offset that is fixed to zero. As a result, the SLAM problem becomes anchored in the global reference frame. 40

3-11 Visualization of the process to establish correspondences. Map elements facing away from the Lidar are excluded. The angles γ_1, γ_2 are used to pre-select correspondence candidates. The correspondences in green are accepted with $|\tilde{y} - y| < s_{\text{corr}}$. 42

3-12 **Left.** A visualization of the ICP algorithm. It can be seen that 3 incorrect correspondences (red) are made due to an incorrect initial pose guess (see §2-2-3). **Right**. The proposed Non-Iterative Closest Point (NICP) observation-to-map alignment procedure considers the Lidar's perspective using ray casting and aligns the Lidar observations with sparse linear map features using (3-19). These features are placed away from edges and irregularities in order to avoid incorrect correspondences. No iterations are required in this alignment process. 43

4-1 **Left.** The Ouster OS0 Lidar device is mounted rigidly onto a wheeled tripod. Five reflective markers are attached for tracking. **Right.** The reflective markers are detected by the motion capturing system. A rigid body is attached to the five markers. 46

4-2 The setup with a reflective prism mounted on the top of the Lidar. This prism is detected by the Hilti PLT-300 Total Station (left) and is able to record coordinates of stationary points as a ground-truth source. 46

4-3 Motion Capture Lab. 47

4-4 VGN office floor 1. 48
4-5 VGN floor 2. 48
4-6 VGN building central. 48
4-7 VGN building top. 49
4-8 The estimated map and trajectory of datasets *VGN floor 1(a)* and *VGN floor 1(b)* 51
4-9 **Left.** Just before failure at waypoint 12 of dataset *VGN floor 1(b)* the Lidar is running out of local horizontally oriented map elements. **Right.** The algorithm starts to diverge due to incorrect correspondences and lack of map elements. . . 51
4-10 The estimated map and trajectory of datasets *VGN floor 2* and *VGN top*. 52
4-11 The estimated map and trajectory of datasets *Motion capture lab* and *VGN central*. 52
4-12 An example of a situation with multiple correspondence errors. The Lidar observes a wall in the room with the PLT tripod and a person in front of it. Two of the three PLT legs and the person were associated with the map element that represent the wall, introducing a significant error in the SLAM process. 53
4-13 The clutter in front of the wall prevents map elements to be found at the wall. . 54
4-14 Only vertically oriented map elements are visible to the Lidar. The Lidar's pose is not properly constrained due to the lack of horizontally oriented map elements, causing the algorithm to fail. 54
4-15 A door is opened during the recording of a dataset. A map element was placed at the closed position of the door, and starts to introduce errors as the door is being opened. 55
4-16 Plot of 5 map variables $d_{\text{pos},i}, d_{\varphi,i}$ over time of dataset *Motion capture lab* with small map elements. Only five out of nine map elements are shown to keep the figure readable. 62
4-17 **Left.** The 16 map elements with $r = 0.18$ m of dataset *Motion capture lab* are shown that were found during initialization. **Right.** Accuracy comparison when excluding various reflective planes from the SLAM process. 62
4-18 Effect of tuning the Beam Angle Discount Factor s_β. 63
4-19 **Left.** Singular values of the Jacobian matrix J on a logarithmic scale when no anchor map elements were selected. The Lidar remained stationary over the 5 second period. **Right.** Singular values when selecting 3 independent anchor map elements. 64

A-1 **Left.** Cartesian and polar coordinates. **Middle.** Two rotated frames. **Right.** Two frames with relative rotation and translation are related through transformation. . 71

C-1 **Left.** Least-squares plane fitted through a point cloud that represents a straight wall. **Right.** Distribution of the point-to-plane projection distance. The distance between the Lidar and the wall was 1 meter. 75
C-2 **Left.** Noise distribution, Lidar-to-wall distance of 5 m. **Right.** Noise distribution, Lidar-to-wall distance of 9 m. 76

List of Tables

1-1 The top ten contenders of the Hilti SLAM challenge. The table was adapted from [2]. For each contender, six trajectory estimates were compared with ground truth at 82 waypoint locations using a Total Station. Available sensors were two Lidars (L), three IMU's (I) and five cameras (C). 3

2-1 A selection of SLAM algorithms. 18

4-1 Overview of datasets. 49

4-2 Overview of tuning parameters. 50

4-3 Error comparison at the static waypoint locations. Failure cases are marked red. Two cases of dataset *VGN floor 1(b)* were partially successful (see §4-3) and are marked purple. The errors of these datasets are based on the first 11 out of 13 waypoints. The values between brackets $(\cdot)$ indicate the Max-AE, i.e. the largest outlier in the waypoint error sequence. 59

4-4 Error comparison summary, considering only the (partially) successful cases of Ray-SLAM using small map elements. These cases contain 69 out of 82 waypoints that were recorded over the six trajectories. The values between brackets $(\cdot)$ indicate the largest outlier in the error sequence (Max-AE). 59

4-5 Ablation study results. The relative error increases over the available datasets are shown with respect to the original Ray-SLAM algorithm in three ablation cases. *RS(L)* represents Ray-SLAM with map elements of $r = 0.35$ m and *RS(S)* represents Ray-SLAM with map elements of $r = 0.18$ m. The values between brackets $(\cdot)$ indicate how much the largest outlier in the error sequence has increased (Max-AE). 60

5-1 Error comparison summary, considering only the (partially) successful cases of Ray-SLAM with small map elements of $s_r = 0.18$. These cases contain 69 out of 82 waypoints that were recorded over the six trajectories. The values between brackets $(\cdot)$ indicate the largest outlier in the error sequence (Max-AE). 68

B-1 Ouster OS0 Lidar technical specifications, adapted from [3]. 73

B-2 Ouster OS0 IMU technical specifications, adapted from [3]. 73

Preface

The decision to take on the challenge of a masters study in Systems and Control was quite a leap of faith for me, as I was already working for over seven years in the industry after finishing the B.Eng. study in mechanical engineering. I had settled into a job that gave me financial prosperity and little to worry about in life. However, I had this feeling that there was so much more to learn and explore, and the more complex scientific topics I found intriguing were out of reach. Long story short, I decided to just go for it, knowing it would not be an easy task. Looking back now, I am very happy to have made this choice. It was a wonderful journey, being exposed to the fascinating fields of control, robotics, sensor fusion and more. The people I have met both at TU Delft and Hilti inspired me to think critically and pushed me to learn and grow.

For this, I would like to express my gratitude to university supervisors dr. Manon Kok and dr. Sander Wahls, TU Delft department of Systems and Control and my company supervisors Michele Crabolu, Hilti - TPA, Michael Helmberger, Hilti - TPR and Kristian Morin, Hilti - TPR for their invaluable support and feedback.

Second, I would like to thank my girlfriend Oriane de Vries and my cousin Bart Jacobsz Rosier. Oriane, for always being there for me and having the patience and willingness to hear me out on all the abstract topics. And Bart, for the invaluable mentoring and support when I needed it.

Delft, University of Technology — H. van Bavel
January 31, 2022

Chapter 1

Introduction

1-1 Research motivation

Robotics is widely applied nowadays in sectors such as health care, agriculture and manufacturing, to name just a few. Also in the building construction sector, robots are used for off-site prefabrication of building materials. In these processes, robots are located next to assembly lines and the environment is highly controlled in order to mass produce the product of interest. Automation can thrive in this context, because the robots and workpieces are on fixed locations and the conditions remain constant over time. It can be seen, however, that this trend of automation using robots has still not diffused into in-situ fabrication as of today.

Robotization for in-situ fabrication on the building construction site is a much more challenging topic. In-situ fabrication poses a variety of new challenges: the environment is large and complex, the environment changes over time, clutter may be present and the real-world building is inherently different from the designed building (*as-built errors*). There are many unsolved problems that need to be addressed in order for a robot to act 'sensibly' and 'get things done' under such circumstances. A more in-depth analysis about the progression and challenges ahead for automation on the building construction site is given by Buchli et al. [4].

In general, an autonomous in-situ fabrication robot has to solve three high-level algorithmic steps [4]: (1) localization of the robot itself, the destination and potential obstacles, (2) planning the optimal action(s) and (3) execution of the actions in a robust and predictable manner. Several attempts for (semi) automated in-situ fabrication can be found in the literature, for example in [5] where accurate autonomous brick laying is demonstrated, and [6] shows the construction of a steel reinforced concrete wall with a complex curved shape. In [7], a wheeled robotic platform with a manipulator arm is demonstrated. Under the right circumstances, this robot can localize itself on the building construction site and navigate autonomously to destinations to perform tasks such as drilling holes.

This thesis research was conducted at the robotics research department of the company Hilti[1]. Hilti develops, manufactures, and markets products for the construction industry. Hilti is pur-

[1] https://www.hilti.com/

Figure 1-1: The Hilti Jaibot drilling robot in action. Localization is done using a Total Station, which is shown in the right.

suing the goal to increase productivity and improve job site safety by automating dangerous and heavy construction tasks by automating in-situ labor. As a first step towards realizing this goal, Hilti released the Jaibot in late 2020, see Figure 1-1. The Jaibot is a semi-automated tracked drilling robot aimed to alleviate workers from overhead drilling. Doing this work manually is strenuous, challenging, and time-consuming. The Jaibot needs to be steered manually to a location from which the robot arm can reach the target, after which the robot can carry out the drilling task by itself. The interested reader can view a YouTube[2] video to see the set-up and operation of the Jaibot.

In this thesis study, the focus lies on the localization part of a robot such as the Jaibot. In order to comply to the strict metric architectural requirements, a localization accuracy of sub-5 mm is required. Currently the Jaibot uses a Total Station (Hilti PLT-300) [8] for localization. The Total Station is shown in the right of Figure 1-1. This expensive device needs to be set up manually near the Jaibot by an operator by detection of reference points in the building, which is a tedious and time consuming procedure. The Total Station can then detect a reflective prism on the Jaibot and reliably localize the robot with sub-5 mm accuracy. The Total Station needs to be repositioned every time the line-of-sight is broken between the robot and the Total Station.

For the purpose of increasing autonomy and reducing cost of the Hilti Jaibot drilling robot, a solution is required that solely relies on on-board sensing, and as such eliminate the dependency on a Total Station or any other external infrastructure. This means that the position and orientation (together referred to as *pose*) are to be estimated. Existing methods in the literature for robot localization using on-board sensing can be divided into two strategies: (1) building model based localization, where the architectural building model acts as the primary

[2] https://www.youtube.com/watch?v=yi5jBeMQkwg

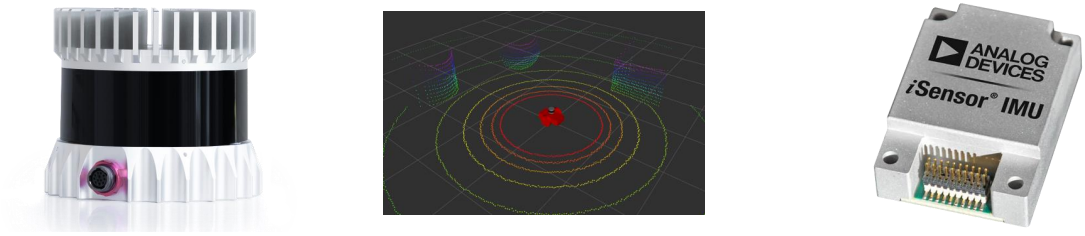


Figure 1-2: Left. The Ouster OS0 Lidar generates 131072 distance measurements per scan, which together form a *point cloud*. This Lidar scans at 10 Hz (see Appendix B). **Middle**. A visualization of the point cloud measurement of a Lidar. **Right**. An Inertial Measurement Unit (IMU).

Table 1-1: The top ten contenders of the Hilti SLAM challenge. The table was adapted from [2]. For each contender, six trajectory estimates were compared with ground truth at 82 waypoint locations using a Total Station. Available sensors were two Lidars (L), three IMU's (I) and five cameras (C).

#	Team	Algorithm	Sensors	<1 cm	<10 cm	<100 cm	>100 cm
1	Megvii	FAST-LIO2 [11]	L+I	2	67	13	0
2	Bosch Research	*Proprietary*	L+I	1	68	13	0
3	Vision & Robotics	*Proprietary*	L+I	1	51	30	0
4	GeoSLAM	*Proprietary*	L+I	2	55	13	12
5	Oxford Robotics	VILENS [13]	L+C+I	0	44	38	0
6	Nanyang University	VIRAL SLAM [14]	L+C+I	1	32	48	1
7	Tsingsens	CMU DOOM	L+I	0	37	37	8
8	ETH Zürich	Maplab, OKVIS [15]	L+C+I	0	22	52	8
9	NPM3D Team	CT-ICP [16]	L+I	0	30	31	21
10	UC San Diego	*Proprietary*	L+I	0	37	12	0

reference and (2) Simultaneous Localization And Mapping (SLAM) where an observation-based map is constructed on-the-go, assuming no prior map information is available.

When considering the literature, it can be seen that building model based localization is not able to realize the required sub-5 mm accuracy [7, 9], and SLAM is typically not designed and/or tested with use cases in mind that require sub-cm accuracy [10, 11]. Hilti's need for a solution to this problem is highlighted by the Hilti SLAM Challenge [12] which was held during September 2021. A total of twelve datasets were published that represent the challenges of a building construction site to review the state-of-the-art and push this field of research forward. The robot was equipped with five vision (camera) sensors, three Inertial Measurement Unit (IMU) sensors and two Light Detection and Ranging (Lidar) sensors. An IMU sensor consists of a gyroscope and accelerometer, and the Lidar generates a large amount of range measurements within its field of view, see Figure 1-2. Accurate ground-truth information at static points was recorded but only published for six datasets, the other six datasets were used to evaluate and rank the competitors. No architectural model of the environment was provided as reliance on such a model is considered a limitation; the model may be inaccurate or not available at all. Contenders were therefore required to utilize the SLAM strategy. A total of 30 contenders worldwide, both companies and universities, submitted their trajectory estimates. The results of top ten contenders are shown in Table 1-1.

Based on these results, the following conclusion can be drawn about the state-of-the-art in SLAM: (1) the best performing SLAM algorithms typically obtain an absolute position error between 1 cm and 10 cm at most waypoints, with some outliers between 10 and 100 cm and (2) localization using Lidar and IMU dominate the top ten. A possible explanation for (2) is that the Lidar sensor measures distance directly and reliably, whereas pixel intensities from cameras are subjected to complex processing pipelines before geometric relations can be established [17], potentially increasing localization errors. This observation was also made during the literature review that was conducted prior to this thesis, and it is for this reason that cameras are not considered in this thesis. The Lidar and/or cameras are often combined with the IMU because the IMU can accurately detect motion at a high sampling rate, which helps to increase both accuracy and robustness in scenarios with aggressive motion. Furthermore, when taking a closer look at the methods that have been deployed, there is the general trend that Lidar-centric SLAM algorithms are designed with generality in mind, rather than of aiming for improved accuracy in a specific scenario. Altogether, it becomes clear that there is a significant gap between the performance of the state-of-the-art and the sub-5 mm accuracy requirement that a construction robot needs to meet, which motivates the need for further research. As more accurate laser-based sensing hardware comes with significant increase of cost, this thesis will focus on algorithmic improvements instead. The main question of this thesis is therefore formulated as:

How can the state-of-the-art in Lidar-centric SLAM be improved to achieve sub-5 mm accuracy on a building construction site?

With many Lidar-centric SLAM algorithms being available as open-source implementations, an important strategic choice for the thesis project is as follows. Should one:

(A) take an existing open-source SLAM algorithm as basis and improve it. *Advantages:* build on top of existing functionality. *Disadvantages:* inherit (some of) the assumptions and/or limitations tied to the algorithm's overall design philosophy.

(B) build a SLAM algorithm from the ground-up. *Advantages:* no imposed restrictions and/or prior assumptions. Achieve a high level of control in all parts of the algorithm. *Disadvantages:* the scope of the final algorithm is to be reduced significantly.

A SLAM algorithm consists of many parts that coherently work together as we will see in Chapter 2, together forming the SLAM pipeline. The approach of this thesis project is to develop a Lidar SLAM algorithm that introduces multiple interdependent novelties at various levels of this pipeline, and as such building on top of an existing implementation would be cumbersome. This thesis therefore pursues strategy (B), with the goal of realizing a full SLAM pipeline with carefully designed components that work together in synergy. It will therefore be important for this project to simplify the research question in acceptable ways, make assumptions and focus on a limited amount of sensor modalities. Since fusing the IMU and Lidar sensor data is a complex process that requires a significant amount of effort, it was chosen to focus only on the Lidar sensor. With this in mind, the following sub questions are defined to answer the main research question:

- ***How can a Lidar SLAM algorithm exploit the specific scene structure that can be expected on a building construction site?***

Rather than focusing on general scenes, an algorithm is to be developed that is tailored at one specific scene type. The algorithm should work on real-world datasets. The algorithm may be simplified in certain ways, such that the results of this thesis serve as a proof-of-concept of the novelties that it introduces.

- ***How robust is the novel algorithm in representative real-world scenarios with various levels of complexity and moving obstacles?***

  The complexity of the real world is not to be underestimated, and potential problems may arise in unexpected ways that cause the SLAM process to fail. Realizing a SLAM method that has the ability to act robustly to the many challenges that arise in the real world is therefore considered paramount to its usability. Investigating the robustness of the novel algorithm is one key questions to be answered.

- ***What is the accuracy of the novel SLAM algorithm, and how does it compare with other state-of-the-art Lidar-centric SLAM algorithms?***

  Similar to the Hilti SLAM Challenge, the accuracy of the novel SLAM algorithm is to be evaluated and compared against the state-of-the-art.

1-2 Contributions

Prior to this thesis work, a literature survey was conducted. The results of this survey, and later confirmed by the results of the Hilti SLAM Challenge, show that Lidar-centric SLAM yields the most accurate results when compared to camera-centric SLAM. It was furthermore found that at the core of these Lidar-centric algorithms, several assumptions were made about what the world looks like and how the robot's observations relate to its internal representation of this world. These assumptions were put in place for *general* environments, and introduce multiple (small) sources of error. It was hypothesized how a different approach with more accurate assumptions about the scene structure of a building construction site can yield better results. The main contributions of this thesis are presented as follows:

- ***Development of a novel 2D Lidar SLAM algorithm that focuses on indoor scenes and accuracy.***

  In this thesis, a novel 2D Lidar SLAM algorithm is proposed, hereafter called Ray-SLAM. Various novelties are introduced by the algorithm that have the common goal of improving the localization accuracy on indoor scenes similar to a building construction site. Figure 1-3 provides a high-level overview of the algorithm.

- ***Extensive testing and performance evaluation of the proposed algorithm on representative real-world datasets.***

  Real-world datasets have been recorded with an Ouster OS0 Lidar in scenes that represent locations where Hilti envisions deployment of construction robots. The datasets were less complex with respect to the Hilti SLAM Challenge datasets such that (1) the motion was restricted to the horizontal plane, (2) the robot moved in a stop-and-go manner and (3) outdoor areas were avoided. The five indoor scenarios were selected

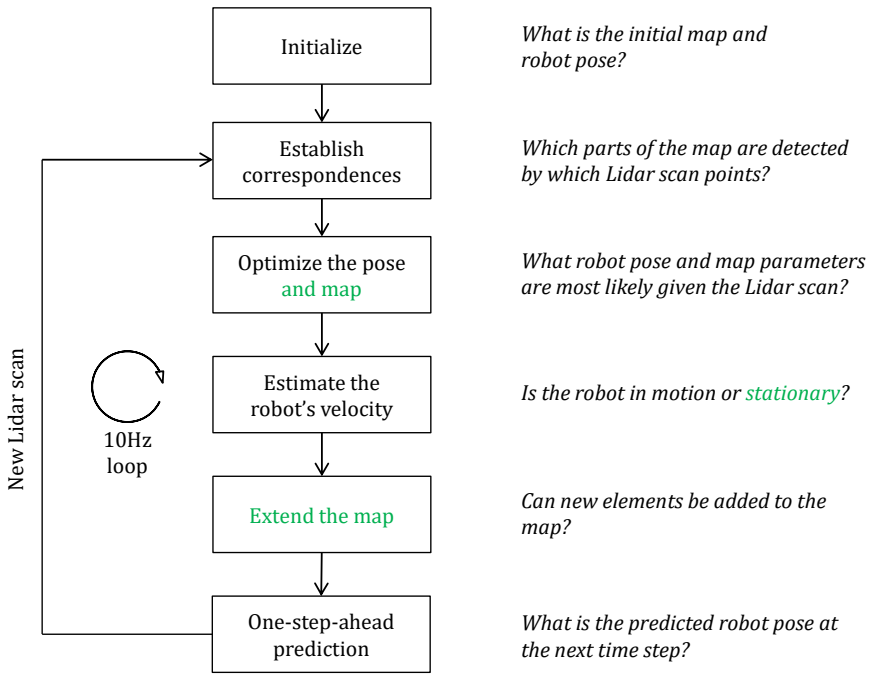


Figure 1-3: A high-level flow chart description of the novel Ray-SLAM algorithm that is proposed in this thesis. The steps 'Optimize the map' and 'Extend the map' (marked in green) are carried out only if the robot is perceived to be stationary.

with rooms of various shapes and sizes in order to get an understanding of how Ray-SLAM performs under various real-world circumstances. Ground-truth positions were recorded to assess the accuracy. In-depth analysis of the failure cases was conducted to asses the algorithm's robustness.

- ***Comparison of the proposed algorithm's accuracy with the state-of-the-art.***

  The performance of Ray-SLAM was compared with state-of-the-art algorithms, i.e., LOAM [18] and FAST-LIO2 [11] by comparing the accuracy of these algorithms on the same datasets. Similar to the Hilti SLAM Challenge, the accuracy was only evaluated at static waypoint locations. The ground-truth information was obtained using either the Total Station (Hilti PLT-300) or a motion capturing system (OptiTrack). The Mean Absolute Error (MAE) error metric is used to obtain an overall performance metric. In order to obtain a measure of resilience to outliers, the Maximum Absolute Errror (Max-AE) metric is used.

1-3 Thesis Outline

Having defined the research questions and thesis contributions, Chapter 2 will focus on the related work in the fields of sensor fusion, processing of Lidar sensory data and SLAM. For completeness, also several localization algorithms are discussed that utilize the architectural

building model. This chapter will introduce the concept the algorithm's *back-end* and *front-end*, which will help to get a better understanding of how the lower level building blocks of a localization algorithm are related.

Chapter 3 will consequently propose the Ray-SLAM algorithm that was developed during this thesis. The simplifications and assumptions are detailed on which the algorithm builds, followed by the novelties that Ray-SLAM introduces. Chapter 3 then continues with a detailed description of the individual components that together form the SLAM algorithm.

In Chapter 4, the real-world experiments are described that have been conducted in order to assess the accuracy and robustness of the Ray-SLAM algorithm. The pose estimates of Ray-SLAM will be compared against the ground-truth positions that have been measured with a Total Station or motion capturing system. The algorithm's accuracy is compared against two state-of-the-art Lidar-centric SLAM algorithms in order to assess if the novelties bring an improvement. Special attention will be paid in this Chapter to failure cases and cases where the accuracy degraded significantly.

Chapter 5 will conclude this thesis. The research questions that have been posed in this chapter will be answered, and the strengths and limitations of the algorithm will be pointed out. As this thesis presents a proof-of-concept for a novel strategy in SLAM, this chapter will detail the next steps that need to be taken to transform the ideas of Ray-SLAM into a full 3D SLAM algorithm that can potentially be among the top performers in the Hilti SLAM Challenge.

Chapter 2

Related Work

This chapter will familiarize the reader with the related work on the topic of Lidar-centric localization algorithms. The limitations and questions left unanswered by the existing work are emphasized to motivate the need for the novel approach that this thesis proposes. Based on the limitations that have been found, the design goals are introduced that this thesis will pursue in order to realize a highly accurate localization algorithm.

General notation. Unless stated otherwise, small letters represent a scalar, bold letters a vector, capital letters a matrix, small Greek letters an angle and calligraphy letters a set. The i^{th} element in a set is denoted in superscript. The notation t in subscript denotes the time point of a time-variant variable, relative to the time that the estimation process was started at $t = 0$. The time points $t = 0, 1, 2, ...$ coincide with the Lidar scan time points, which are time-stamped roughly 0.1 sec apart since the Lidar scans at typically 10 Hz. Reference frames are denoted by capital letters in superscript. Descriptive terms about a quantity are written in subscript where needed. As a fictitious example, the Cartesian point vector $\mathbf{p}_{\text{ref},0}^{L,i} \in \mathbb{R}^3$ denotes the i^{th} point in the set of points $\mathcal{P}_{\text{ref},0}^{L}$ at $t = 0$ in the L frame, with the description 'ref'. Furthermore $\hat{\cdot}$ describes an estimate, $\check{\cdot}$ a one step ahead prediction and $\tilde{\cdot}$ a measurement prediction. The transpose operation is denoted by $(\cdot)^\top$.

2-1 Existing work on sensor fusion algorithms for pose estimation

In the context of SLAM, the literature often distinguishes the *front-end* and *back-end* parts of the algorithm [19, 20]. The front-end is considered the part of the algorithm that has access to the sensor information. It interprets this information to establish relationships between the sensor information, map and robot pose(s). These relationships form the basis of the optimization problem. The back-end is consequently responsible for estimating the actual state(s) by solving the optimization problem that incorporates these constraints. This thesis uses the principle of front-end and back-end separation to explain the functioning of localization algorithms in a more general sense.

First, we consider the state-of-the-art in general back-end sensor fusion algorithms. A general framework to model a dynamic system such as a robot is the *state space* [21] representation. The state vector $\mathbf{x}_t$ contains e.g. the robot's position and some parameterization of its orientation [22] at time t. In 3D, the orientation is often expressed by the quaternion or Euler angles while in 2D one can simply use the heading angle. Often the robot's velocity and the Inertial Measurement Unit (IMU) biases are also included. A nonlinear *dynamics* model $f(\cdot)$ uses *input measurements* $\mathbf{u}_t$ to relate the state $\mathbf{x}_t$ to the next state $\mathbf{x}_{t+1}$. The dynamics are affected by the process noise term $\mathbf{w}_t$. The nonlinear *observation* model $h(\cdot)$ on the other hand relates the robot state to the *update measurements* $\mathbf{y}_t$ at a specific time instance and is affected by the noise term $\mathbf{e}_t$. These models together form the state space representation of the system and are denoted by

$$\mathbf{x}_{t+1} = f(\mathbf{x}_t, \mathbf{u}_t, \mathbf{w}_t), \qquad \mathbf{w}_t \sim \mathcal{N}(0, Q), \tag{2-1a}$$

$$\mathbf{y}_t = h(\mathbf{x}_t, \mathbf{e}_t), \qquad \mathbf{e}_t \sim \mathcal{N}(0, P_y), \tag{2-1b}$$

where the noise parameters $\mathbf{w}_t, \mathbf{e}_t$ are modeled as zero mean Gaussian noise with covariances Q and P_y respectively. The state estimation algorithms that are described below share the common objective of estimating $\mathbf{x}_t$, making use of available sensory information $\mathbf{u}$ and $\mathbf{y}$ up until time point t and the initial state prior $\check{\mathbf{x}}_0$.

Kalman filtering

A SLAM algorithm becomes increasingly useful if it can be deployed on a robot that is able to run the algorithm in real-time. Estimating the entire map and trajectory at once, known as the *smoothing* problem, quickly becomes unfeasible in real-time as time progresses and the mapped area grows larger. Under the assumption of the Markov property (i.e. all the system's information up to the current time t is contained in the state $\mathbf{x}_t$), one can use (a version of) the Kalman Filter (KF) [23] to solve SLAM incrementally, significantly reducing the computational demand of a SLAM algorithm. The Kalman filter is a powerful state estimation algorithm that has withstood the test of time since it was introduced in 1960. It can be understood as a two-step process: a time update that uses the model of (2-1a) to predict $\mathbf{x}_t$ at time step $t-1$, and a measurement update that updates the predicted state estimate using the newly available measurement $\mathbf{y}_t$, resulting in $\hat{\mathbf{x}}_t$.

For linear state space models and assuming Gaussian additive noise, the Kalman filter gives an unbiased minimum variance estimate of the state $\mathbf{x}_t$. Many variants and improvements over the traditional Kalman filter exist in the literature, mainly focusing on improving the performance for nonlinear state space models [21, 24]. The Extended Kalman Filter (EKF) linearizes the state space model using a first order Taylor expansion around the current estimate, and takes one Gauss-Newton optimization step with a step length of one to obtain the estimate $\hat{\mathbf{x}}_t$ for the measurement update [25]. Iterating this process until convergence leads to the Iterated Extended Kalman Filter (IEKF). Variations of the Kalman Filter can be found as the basis of various localization and SLAM algorithms such as [11, 21, 26]. The main benefit of Kalman Filter based methods is their relative low computational overhead, low memory requirements and ease of implementation.

Moving horizon estimation

Moving Horizon Estimation (MHE) is a more advanced state estimation strategy over Kalman

Filtering and is considered as the state-of-the-art in general sensor fusion frameworks. The main benefit of MHE is the flexibility to re-linearize of past states based on new information, which improves the accuracy and robustness to measurement outliers. An example of a state-of-the-art MHE-based algorithm is ConFusion [27]. ConFusion provides a sensor fusion framework where any sensor modality can be incorporated in a modular fashion. It provides the flexibility to (1) add *static parameters* to the estimation problem to optimize parameters such as extrinsic calibrations, (2) use the built-in implementation of the IMU measurement model that is based on the *preintegration*[1] technique and (3) jointly estimate the entire state trajectory as a smoothing problem. ConFusion is found for example in the autonomous construction robot as described in [7]. ConFusion, however, is a large code base and requires increased computational resources with respect to a Kalman Filter based approach. ConFusion is furthermore designed for more general sensor fusion problems rather than SLAM in specific. It does not contain a Lidar measurement model, and the current implementation of cameras is restricted to detection of QR codes that need to be placed in the scene manually.

Factor graph optimization

Formulating the state estimation and SLAM problems as a *factor graph* [19, 29] has received significant attention lately and is representative of the state of the art in SLAM. The most popular open-source factor graph back-end algorithms are GTSAM [30] and g2o [31]. The factor graph provides a flexible structure well suited to SLAM such that the user is free to define how the robot poses and observed landmarks (*variables*) are connected through process and update measurement constraints or prior information (*factors*). The factor graph algorithm then estimates the poses of the robot and landmarks by solving an optimization problem. Benefits of pose graph optimization are (1) a convenient abstraction layer to model and visualize the SLAM problem, (2) ability to solve loop closures in round-trip trajectories and (3) ability to optimize the complete state trajectory in real-time using the technique of incremental smoothing and mapping. The downside of a factor graph is that the user is now forced to work on a higher level of abstraction by the use of the graph structure.

2-2 Existing work on point cloud handling

This section takes a closer look at the existing work in the front-end of Lidar based localization. The problem of localization with the Lidar has a strong focus on geometry. The Lidar device observes the world from the constantly changing perspective of the robot, and one needs to make assumptions about the large and complex outside world in order to obtain a simplified representation of this world that a robot can cope with for the purpose of localization. Figure 2-1 takes a closer look at the structure of the Lidar observations, and clarifies which collection of observations form the *scan line* and *scan ring* respectively.

We will now consider the state-of-the-art techniques for handling point cloud data that form the basis of the assumptions that the robot makes when perceiving the world. Small error sources introduced by such techniques eventually add up over time and cause the SLAM algorithm to drift [10].

[1] IMU preintegration is an advanced technique to reduce the high-rate IMU output to lower rate relative pose information, taking into account the IMU biases and the full manifold $SO(3)$ of transformations in $\mathbb{R}^3$ [28].

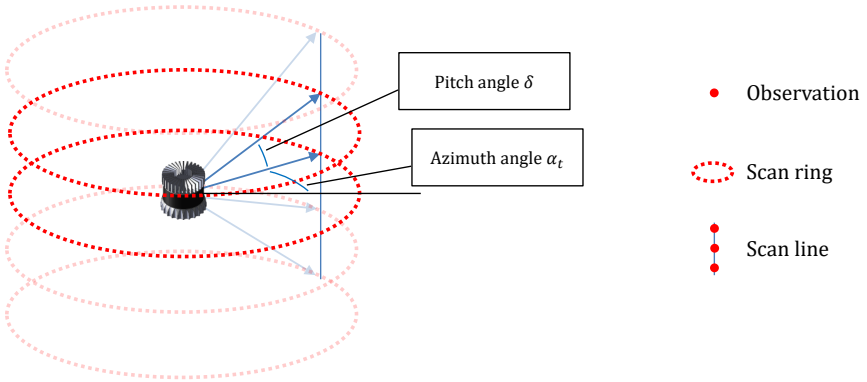


Figure 2-1: In this toy-Lidar example, five individual laser range measurement sensors are located inside the Lidar with different fixed pitch angles. The five measurements at a single azimuth angle α_t are carried out at the same time and considered a *scan line*. The five range sensors rotate internally within the Lidar and generate 80 scan lines sequentially over time in this example. A *scan ring* is the collection of observed points resulting from one range sensor during one rotation. The observations from all range sensors during one rotation are referred to as a *scan*. The Ouster OS0 Lidar was used for the experiments in this thesis, which has 64 scan rings and 2048 observations per ring (see Appendix B).

The following techniques are now discussed that form the back bone of many Lidar-centric localization algorithms in the literature, respectively (1) normal estimation and local cluster analysis, (2) edge- and planar feature point extraction (3) the Iterative Closest Point (ICP) algorithm to align two point clouds. Special attention is paid to potential error sources that can be introduced by these techniques.

2-2-1 Normal estimation and local cluster analysis

We start by introduction of the *point cloud*, which is the collection of Cartesian point coordinates $\mathbf{p}^i \in \mathbb{R}^3$ such that the point cloud $\mathcal{P} = \{\mathbf{p}^1, \mathbf{p}^2, \ldots, \mathbf{p}^N\}$. Since the points in the Lidar's observed point cloud are ordered by azimuth- and pitch angles (see Figure 2-1), one can easily extract a local set of observed points centered around $\mathbf{p}^i$ in order to analyze the geometry of the locally observed scene. This local set of N_{clust} points around $\mathbf{p}^i$ is denoted by $\breve{\mathcal{P}}_i \subseteq \mathcal{P}$. The *covariance tensor* G [32] is defined by

$$G = \frac{1}{N_{\text{clust}}} \sum_{\mathbf{p}^j \in \breve{\mathcal{P}}_i} \left(\mathbf{p}^j - \overline{\mathbf{p}}^i\right)\left(\mathbf{p}^j - \overline{\mathbf{p}}^i\right)^\top, \qquad \overline{\mathbf{p}}^i = \frac{1}{N_{\text{clust}}} \sum_{\mathbf{p}^j \in \breve{\mathcal{P}}_i} \mathbf{p}^j. \tag{2-2}$$

The point $\overline{\mathbf{p}}^i$ is the mean point coordinate of cluster $\breve{\mathcal{P}}_i$. Let the eigenvectors $\mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3$ correspond to the eigenvalues $\lambda_1 \geq \lambda_2 \geq \lambda_3 \geq 0$ of G. The *Plane PCA* method as discussed in [33] takes N_{clust} between 10 and 50 in order to obtain the surface normal $\mathbf{n}^i = \mathbf{e}_3^i$ that corresponds to the point $\mathbf{p}^i$. The eigenvalues λ can be utilized to calculate useful properties such the cluster's linearity, planarity and curvature [32].

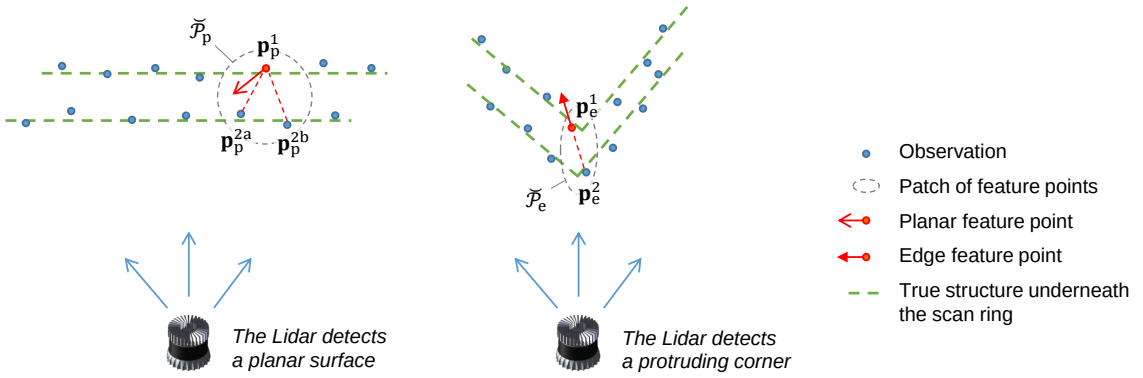


Figure 2-2: Many Lidar-based SLAM algorithms use LOAM-based feature points. On the left the planar feature point is shown, where the arrow points in the normal direction of the local surface. On the right, the edge feature point is shown, where the arrow points into the principal direction of the line that represents the edge.

2-2-2 Edge- and planar feature point extraction

A very influential technique in Lidar SLAM is the concept of an edge- and planar feature point extraction as introduced by the LOAM [18] algorithm, which has since then become the *de-facto* standard of how the front-end of a Lidar centric SLAM algorithms handle point cloud data, resulting in popular follow-up works such as [10, 34, 35, 36].

The key concept is that instead of considering a full Lidar scan, only a limited number of *feature points* is kept that carry the most important information. The approach considers a collection of points on two adjacent scan rings and is further explained below together with Figure 2-2. The smoothness c of point $\mathbf{p}^i$ on one scan ring is considered by the (simplified) expression

$$c = \left\| \sum_{\mathbf{p}^j \in \breve{\mathcal{P}}, j \neq i} \left(\mathbf{p}^i - \mathbf{p}^j \right) \right\|_2, \tag{2-3}$$

where $\breve{\mathcal{P}}$ is a collection of j points, with both half of the points in $\breve{\mathcal{P}}$ lying on each side of $\mathbf{p}^i$. The smoothness c tends to go to zero when $\mathbf{p}^i$ lies in the middle of $\breve{\mathcal{P}}$, which is the case when $\breve{\mathcal{P}}$ represents a line. Points are sorted by c, and edge points can be chosen with the highest c and planar points with the lowest c. A planar patch $\breve{\mathcal{P}}_p$ is formed by three planar points on two adjacent scan rings such that $\breve{\mathcal{P}}_p = \{\mathbf{p}_p^1, \mathbf{p}_p^{2a}, \mathbf{p}_p^{2b}\}$. The local surface normal can now be determined. One of these three points, together with the surface normal, is kept as the planar feature. The edge patch $\breve{\mathcal{P}}_e$ is found by considering two close edge points on two adjacent scan rings, such that $\breve{\mathcal{P}}_e = \{\mathbf{p}_e^1, \mathbf{p}_e^2\}$. The principal direction of the edge patch can now be determined. The edge feature contains one of the two points and the direction vector. This process is repeated for all scan lines to collect features in the whole scan. Voxel grid discretization[2] is used to reduce the amount of feature points.

Three important limitations of the LOAM feature extraction principle are (1) the direction of

[2] A *voxel grid* is a grid of cube-shaped volumes (cells) in $\mathbb{R}^3$ with an identical size, and can be understood as the 3D generalization of square pixels on the 2D plane. Voxel grid discretization is the process of reducing the amount of points in $\mathcal{P}$ such that each cell contains up to one point.

the planar feature point's normal is sensitive to measurement noise because only three points are used that are relatively close together, (2) the edge feature points represent a surface close to the edge, not the edge itself and (3) objects with a complex geometry, e.g. trees, often generate a lot of edge features. The select amount edge features representing such an object are a very incomplete description of this object. As such, the point cloud containing LOAM feature points is only a coarse approximation of the actual underlying geometry, reducing the overall accuracy of a SLAM algorithm that uses these features. Errors resulting from this coarse description are typically neglected.

2-2-3 Point cloud registration using the ICP algorithm

In order to explain the *point cloud registration* problem, let us consider the following example. Let the collection of points $\mathcal{P}^W$ represent a map of the scene in the world frame W. Now let $\mathcal{Q}^L$ be a set of points in the Lidar frame L that was collected by a Lidar sensor located somewhere in this particular scene. The registration problem is concerned about finding the transformation T^{WL} by alignment of the two sets of points, effectively localizing the Lidar in the map. See Appendix A for the definition of the transformation matrix and the related mathematical operations. Besides localization of the Lidar in a map, one could also infer relative motion by registration of $\mathcal{Q}^L_{t+1}$ to $\mathcal{Q}^L_t$. As such, point cloud registration is often a central part in a Lidar-centric localization algorithm. It is a substantial problem to which many authors have contributed.

A robust way of solving the registration problem was proposed by Besl et al. [37] in 1992, using the ICP algorithm. Lidar-centric localization algorithms typically rely on some variant of the ICP algorithm to compare the latest Lidar scan either with a (local) map or with a previous scan to infer the pose. ICP is generally considered as a robust and accurate method for this purpose [38]. One iteration of the algorithm can be described by executing the following 3 steps:

1. Establish correspondences: every point in set $\mathcal{Q}$ is matched with its nearest neighbour in set $\mathcal{P}$ based on the Euclidean distance.
2. Outlier rejection: correspondences with a distance larger than a certain threshold value $d_{\max}$ are considered outliers and are removed.
3. Optimization: find the transformation that minimizes the cost function using

$$\hat{T} = \underset{T}{\operatorname{argmin}} \sum_{i=1}^{N} \left\| \mathbf{e}^i_{\text{po}\to\text{po}} \right\|_2^2 \tag{2-4}$$

   where $T \in SE(3)$ represent the relative transformation between the sets of points. The term $\mathbf{e}^i_{\text{po}\to\text{po}} = \mathbf{p}^i - T\mathbf{q}^i$ represents the *point-to-point* error metric.

The process above is repeated until a termination condition is satisfied, resulting in a transformation that aligns $\mathcal{Q}$ to $\mathcal{P}$.

Because the points in both sets are often sampled differently (the Lidar points to different parts of the scene when e.g. slightly rotated), more accurate results can be achieved using the

point-to-line and/or *point-to-plane* error metrics. As can be seen in Figure 2-2, the LOAM feature points attempt to locally approximate the structure of the scene by planes and lines. Rather than minimizing the Euclidean distance between $\mathbf{q}^i$ and $\mathbf{p}^i$, one can minimize the projection distance of the observed point $\mathbf{q}^i$ to the feature point $\mathbf{p}^i$ using

$$\mathbf{e}^i_{\text{po}\to\text{li}} = \mathbf{v}^i_{\mathcal{P}} \times \left(\mathbf{p}^i - T\mathbf{q}^i\right), \tag{2-5a}$$

$$\mathbf{e}^i_{\text{po}\to\text{pl}} = \mathbf{n}^{i\top}_{\mathcal{P}} \left(\mathbf{p}^i - T\mathbf{q}^i\right), \tag{2-5b}$$

in the optimization step of ICP. The vectors $\mathbf{v}^i_{\mathcal{P}}$ and $\mathbf{n}^i_{\mathcal{P}}$ represent the principle and normal directions of the edge and planar feature points in $\mathcal{P}$ respectively as visualized in Figure 2-2. The SLAM algorithms based on the LOAM feature point extraction technique rely on an ICP process that minimizes the point-to-plane and point-to-line errors for all the corresponding features that have been found.

A relevant adaptation of the ICP algorithm for indoor localization is presented in the work of Blum et al. [9]. An ICP-based method is proposed to localize the robot directly in the architectural building model by sampling the available 3D building model to a point cloud with normal directions attached to each point. The point-to-plane ICP method is used to localize the robot.

Nearest Neighbor detection. The step of establishing correspondences in ICP requires special attention because a brute force search strategy quickly becomes computationally too expensive for larger point clouds [39]. Approximate search techniques have therefore been developed to speed up this process. Most notably the kD-tree and octree structures are well established techniques for this purpose [40, 41]. These tree structures organize the 3D point cloud by splitting space hierarchically into smaller subspaces until each subspace contains one point. The search for a nearest neighbor can now be done by starting at the root of the tree and systematically going up the tree, significantly reducing the search space. This approach imposes a risk of returning a sub-optimal neighbor, but the redundancy between points typically prevents large accuracy degradation.

Limitations of the ICP process. As this thesis aims to bring improved accuracy in localization, we will take now a closer look at potential error sources in the ICP algorithm:

- Because of high point densities in both $\mathcal{P}$, $\mathcal{Q}$, small initial misalignments between the sets can result in a large amount of both false positive and false negative correspondences, pulling the transformation estimate away from its true value. Iteratively re-establishing these correspondences reduces these errors, but its success can be unpredictable and depends on the sensor resolution, type of scene and initial transformation guess.
- The nearest neighbor detection step using an approximate search technique may result in sub-optimal correspondences.
- Point coordinates in both $\mathcal{P}$, $\mathcal{Q}$ are often extracted directly from the Lidar's point cloud and therefore corrupted with noise, limiting the accuracy of ICP.
- In the particular case of LOAM [18], the ICP process is based of edge- and planar feature points that both contain errors as explained in §2-2-2.

Undistorting a Lidar scan

Due to the rotating nature of a Lidar sensor, the scan output (typically 10 Hz) contains motion distortion when the robot moves because the scan does not originate from a single robot pose. *Undistorting* the Lidar scan is the process of adjusting the position of the points in the point cloud such that the points are congruent with a singular robot pose. The state-of-the-art technique to overcome this problem is to establish tightly coupled sensor fusion between the Lidar and IMU, and use the trajectory estimate of the IMU to undistort the Lidar scan. LIO-SAM [10] and FAST-LIO2 [11] are examples of such tightly coupled Lidar-IMU SLAM algorithms. The IMU trajectory estimate can be very accurate on the short term if the gyroscope and accelerometer biases are known. These SLAM algorithms therefore estimate the biases online and apply the technique of preintegration to transform high-rate IMU measurements (often 200-1000 Hz) to Lidar-rate (10 Hz) relative pose transformations [28]. Limitations of such an undistortion process are the added complexity and risk of errors due to saturation of the IMU signal and incorrect subtraction of the gravity vector.

2-3 Existing work on Lidar-centric localization algorithms

Now that both the back- and front end parts of Lidar based localization algorithms have been discussed, we proceed to take a closer look at state-of-the-art localization algorithms and see how these works implement these principles. For completeness, this chapter discusses both building model based localization algorithms and SLAM algorithms.

2-3-1 Building model based localization

Several works in the literature leverage the availability of a preexisting map, either in the form of a 2D floor plan or a full 3D model. Boniardi et. al. [42] present a hybrid 2D Lidar localization algorithm that overlays a SLAM map (occupancy grid) on top of the building floor plan under the assumption that the actual building's cross section and floor plan are identical. The occupancy grid is used to model objects that are not included in the floor plan such as clutter. The algorithm demonstrates to cope with increased amounts of clutter, but fails when the as-built errors get too large. In the best test case, the algorithm reports an impressive average error of 12 mm $\pm$ 10 mm. An average error of 42 mm $\pm$ 32 mm was reported in a scenario with increased clutter. No open-source implementation of this algorithm is available.

The work of Gawel et al. [7] shows a method where a construction robot first localizes itself roughly by comparing the Lidar observations with the building model using ICP, and fusing the respective output with the IMU and wheel odometry using ConFusion [27]. Since this process is of limited accuracy, the robot then refines its pose estimate by a measurement procedure using three laser range meters with mm-level accuracy mounted on the end of the robot's arm. The range meters were aimed at building features that were considered relevant to the task, i.e. walls close to the robot. This work obtains an accuracy of sub-4 cm. It was found that the accuracy was highly dependent on the robot's location. Errors were introduced due to an inaccurate orientation estimate of the laser range meter due to either (a) errors in their extrinsic calibration, magnified by the large (>10 m) distance measurements or (b) an inclined floor that was not captured in the building model.

Blum et al. [9] improve the matching between the Lidar and building model by using visual information to filter out Lidar observations that do not directly observe the building structure. The localization procedure then selects three surrounding walls near the building task to which the robot references itself using the point-to-plane ICP algorithm. Special attention is paid to as-built errors and cluttered environments. The method demonstrates an accuracy of around 20 cm.

An inherent limitation of relying on the building model is that this model is never perfect due to as-built errors and clutter present on the scene. As-built errors in practice can be in the order of 10 cm. These aspects limit the accuracy and robustness of such methods.

2-3-2 Simultaneous localization and mapping

The field of Lidar-centric SLAM is a very active field of research, with many works being published every year. The use cases for SLAM can vary significantly (e.g. search and rescue, autonomous driving, domestic robots), and as such it can be seen that existing SLAM algorithms are designed with different design goals in mind. The interested reader is referred to [43] for a more in-depth background on SLAM. In this section, a selection of works is highlighted which stand out in terms of performance and/or unique features. Special attention is payed to works that utilize a novel map representation, because this is at the core of how the robot assumes that the world looks like. Different map types may be better suited for different purposes, and as such the ideal map representation of a SLAM algorithm depends on the use case. It was found that significantly less works have recently been published in the field of 2D Lidar-centric SLAM algorithms with respect to 3D methods. For these 2D methods, the literature has largely consolidated on the application of *occupancy grid* map types. Popular works such as Hector-SLAM, KartoSLAM, GMapping [44] and Cartographer [45] all use such a map type.

A natural place to start with Lidar-centric SLAM algorithms is LOAM [18]. This work was published by Zhang et al. in 2014 and is still considered the benchmark algorithm today by newer SLAM works. The feature point extraction technique of the LOAM algorithm was already discussed in detail in §2-2, including its limitations. The LOAM algorithm achieves an efficient way of compressing the point cloud, resulting in a favorable trade-off between computational efficiency, robustness and accuracy. This is the primary reason why the LOAM feature point selection technique is at the core of many follow-up SLAM algorithms.

The Cartographer [45] 2D SLAM algorithm by Hess et al. is a very popular work for use on ground robots with a 2D Lidar. The occupancy grid map type is well suited for tasks such as obstacle avoidance. The accuracy of the algorithm, however, is inherently limited by the grid size. A grid size of 5 cm is proposed to enable real-time performance and reasonable accuracy. Reducing the grid size to below 5 mm increases the map density by more than two orders of magnitude, indicating the limitations in scalability of computational needs of this approach. The 3D generalization of such a map is proposed in [46], but this map representation is rarely used in 3D SLAM algorithms, also due to limited scalability.

Kimera [47] was designed primarily for depth-camera sensors, and stands out by its use of a *triangular mesh* to represent the world. A mesh can describe the world continuously by connected triangular surfaces. A potential benefit is that large planar surfaces, as often found on a construction site, can be effectively modeled. Limitation of the mesh structure

are (1) non-flat structures can only be approximated and (2) complex topology related to the interconnections between triangles.

The LIO-SAM [10] algorithm takes the LOAM feature point extraction technique as a basis of the front-end, and implements a modern factor graph back-end in order to facilitate tightly coupled fusion between the Lidar and IMU and loop closures. The IMU is used to (1) undistort the Lidar scan, (2) provide an initial guess of the pose at the next time point and (3) provide a relative pose constraint between two successive time points in the factor graph. Many works exists that, similar to LIO-SAM, add extra features on top of the principles of LOAM. As such, these works inherit the same limitations as described in §2-2.

LIC-FUSION 2.0 [48] proposes an interesting addition on top of the LOAM principles. Plane features using the 3DOF *closest point* parameterization are tracked within the state vector of the *Multi-State Constraint Kalman Filter*. Contrary to the other discussed works, no open-source implementation of this algorithm is available. The work claims improved accuracy over LOAM. As can be seen in the YouTube video[3], the tracked planes are only short-lived by a few seconds, and typically no more than two planes are tracked simultaneously.

The FAST-LIO2 [11] algorithm breaks the trend of using the LOAM feature point extraction technique by using the raw point cloud directly to construct a dense map. Two main improvements are implemented such that this dense point cloud map can be accommodated. A more efficient *ikd*-tree is proposed to structure the point cloud map such that correspondence finding process can be done with more points in the cloud. Furthermore, a novel strategy for calculating the IEKF's Kalman gain is implemented such that the (larger) ICP optimization problem of (2-4) can be solved efficiently. The result is a SLAM strategy that maintains a dense global map to which new points are directly registered. The FAST-LIO2 algorithm is the highest ranking algorithm of Hilti SLAM Challenge.

The different map representations are visualized in Figure 2-3. The selection of existing works in SLAM is summarized in Table 2-1. A key observation is that the existing works attempt to model largely the entire scene. This can be cumbersome from an accuracy perspective, as a construction site is often cluttered with objects that have complex geometry (e.g. a pile of pallets). Accurately describing such complex geometry is challenging due to noisy sensor data and a limited map resolution. The resulting coarse representation of such complex geometry may consequently reduce localization accuracy.

Table 2-1: A selection of SLAM algorithms.

Name	Back-end	Map type	Highlight
LOAM [18]	Custom	Feature points	Highly influential work in Lidar SLAM.
Cartographer [45]	Custom	2D occupancy grid	Popular 2D SLAM work.
Kimera [47]	Factor graph	Mesh	Unique map representation.
LIO-SAM [10]	Factor graph	Feature points	Tight IMU coupling, loop closure.
LIC-FUSION 2.0 [48]	KF-based	Feature points, planes	Integration of plane tracking.
FAST-LIO2 [11]	KF-based	Dense point cloud	Hilti SLAM Challenge winner.

[3] https://www.youtube.com/watch?v=waE5nepxD-Q

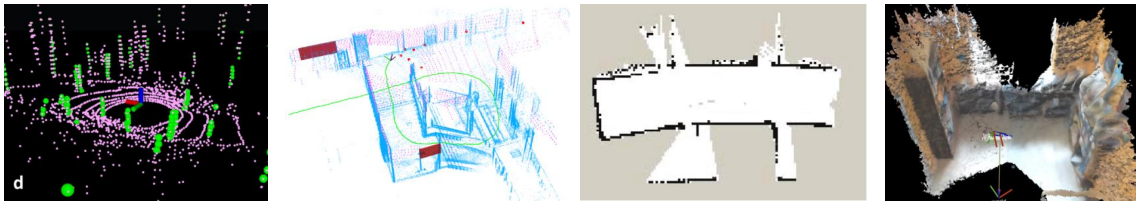


Figure 2-3: Different map representations that can be found in existing SLAM works.

2-4 Design goals of the proposed localization algorithm

We have now taken a closer look at the state-of-the-art in Lidar centric localization algorithms, their components and limitations. It is concluded that currently no method exists that comes close to meeting the requirement of sub-5 mm accuracy. The algorithms discussed in this chapter typically achieve an accuracy of around 5-10 cm, and this level of accuracy can only be maintained in certain scenarios. The localization strategies that utilize the building model are inherently limited by as-built errors and cluttered environments. Altogether, this motivates the need to explore how the accuracy of a localization algorithm can be improved. The main design goals of the novel localization algorithm are summarized below.

- **Ability to work independent of an existing building model.** Building model based localization methods suffer from as-built errors and increased amount of clutter in the scene, or such a model may not be available. Therefore this thesis aims to rely on a map which is built from observations (SLAM) that is independent of this model.
- **Focus on improved accuracy in indoor scenes.** Existing works in Lidar SLAM design their algorithm in such a way that these algorithms can be deployed in general scenes. Limited attention is paid in the literature to a SLAM strategy that is specifically targeted at indoor scenes in order to leverage the specific structure that can be expected at such a scene to improve accuracy.
- **Static accuracy is prioritized over accuracy while the robot is in motion.** A construction robot for tasks such as drilling holes will carry out this task while stationary. Therefore, accurate localization while the robot is stationary is prioritized.
- **Use sequential estimation to keep the size of the problem tractable.** This thesis sets the goal to utilize a sequential estimation technique such as the Kalman filter to solve the SLAM problem incrementally, rather than estimating the entire trajectory and map at once (i.e. smoothing).
- **Ability to cope with moving obstacles and cluttered environments.** People may be present on a construction site and various amounts of clutter can be expected. The localization algorithm should be robust enough to cope with such challenges.

With these design goals stated, Chapter 3 will propose the novel Lidar SLAM algorithm that is aimed at eliminating the limitations of the related work mention in this chapter and realize these design goals.

Chapter 3

A novel 2D Lidar SLAM algorithm: Ray-SLAM

This chapter proposes the Ray-SLAM algorithm, a 2D Lidar Simultaneous Localization And Mapping (SLAM) algorithm. Since the SLAM problem is large and complex, this chapter starts by introducing several simplifications. Then, the state-space model of the system is introduced, with special attention paid to the map. The SLAM problem is introduced formally, which is split-up in several subproblems. Then, the Ray-SLAM algorithm's components are described that aim to solve each of these subproblems such that these components work together for the purpose of simultaneous localization and mapping.

General notation. Tuning parameters are denoted by the letter s, e.g. s_{vel} represents a velocity threshold that can be set by the user. Boolean variables are denoted by the letter b.

As stated in Chapter 1, the SLAM problem is concerned about estimation of the robot's trajectory and constructing a map of the scene, and no prior information is assumed to be available (e.g. an existing map). For clarity, this chapter refers to the Lidar's trajectory as an actual robot is not present. To reduce the scope of the problem, we start by stating which simplifications are put in place.

Simplification 1: reduce the 3D problem to 2D. This work establishes an algorithm that focuses on the 2D simplification of the 3D problem. In other words, we care about the Lidar's 2D position and its heading in the horizontal plane. An accurate pose estimate in the horizontal plane can be an effective solution for many use cases for robots such as the Jaibot.

Furthermore, this thesis study can be considered as a proof-of-concept and a first step towards 3D SLAM. The design choices for the Ray-SLAM algorithm are made with 3D generalization in mind. If the 2D algorithm shows to be successful, then 3D generalization can be considered as a follow-up project. Generalization to 3D is discussed further in Section 5-2.

Simplification 2: repetitive static conditions are present the Lidar motion. Due to the rotating nature of a Lidar sensor, the 10 Hz scan output contains motion distortion when the Lidar moves because the scan does not originate from a single pose (see §2-2-3).

Undistorting the Lidar scans can be done using an IMU, but is considered outside the scope of this thesis. The algorithm therefore does SLAM when the robot is stationary, and localizes only within the latest map when the robot moves. This ensures that errors due to motion distortion are not incorporated into the map. However, the Lidar is now required to be stationary during initialization and stop on a regular basis in order to extend the map. It is therefore not recommended to deploy the algorithm on e.g. a drone, where zero velocity is harder to accomplish.

3-1 State space model and map representation

This section builds up the state space model of the system by defining the state vector, dynamic model and the Lidar measurement model. Special attention is paid to the map which is an integral part of the SLAM problem. The parameterization of the map, how the map is built and maintained, and how the map interacts with the state vector are key ingredients of the Ray-SLAM algorithm that will be explained in detail this chapter.

State vector. The state description of the system $\mathbf{x}_t$ contains the Lidar's pose and its linear- and angular velocities. The pose $\mathbf{x}_{\text{pose},t} \subseteq \mathbf{x}_t$ at time point t contains the position-heading parameterization of transformation of the Lidar frame L to the SLAM frame S, denoted by $T_t^{SL} \in SE(2)$ (see Appendix A for the definition of a transformation matrix and the related mathematical operations). The location of SLAM frame is chosen at $t = 0$ using T_0^{SL}. It can be placed at an arbitrary location by the SLAM algorithm: expressing the map and trajectory from different reference frames yield equally valid solutions to the SLAM problem [19]. The user-provided initial transformation T_0^{SL} is denoted by $\breve{\mathbf{x}}_{\text{pose},0}$. The SLAM frame is often placed at the starting pose of the Lidar such that $\breve{\mathbf{x}}_{\text{pose},0} = \mathbf{0}$. As such, the initial pose is defined rather than estimated. Throughout this thesis, when the reference frame is omitted from the notation, the SLAM frame S is assumed as reference frame for the Lidar pose and map features, and the Lidar frame L is assumed for the observations made by the Lidar.

It was chosen to include the Lidar's linear- and angular velocities into the state vector. An estimate of these quantities is helpful to (1) detect static conditions and (2) predict the Lidar's pose at time $t + 1$ more accurately. The state vector can now be defined as

$$\mathbf{x}_t = \begin{pmatrix} p_{\text{x},t} & p_{\text{y},t} & \theta_t & v_{\text{x},t} & v_{\text{y},t} & \omega_t \end{pmatrix}^\top \tag{3-1}$$

where $p_{\text{x},t}, p_{\text{y},t}$ denote the robot's Cartesian coordinates, θ_t the heading angle in radians, $v_{\text{x},t}, v_{\text{y},t}$ the robot's linear velocity in m/s, and ω_t the robot's angular velocity in rad/s.

In construction, the *world* frame W is typically attached to a specific part of the architectural building model, such as the corner of a room. The rigid transformation between the SLAM frame and a world frame is denoted by T^{WS}. Estimating this transformation is referred to as the *global registration* problem and is considered outside the scope of this thesis [49], as this work does not utilize an a-priori known map.

3-1-1 Dynamic model

The dynamic model relates the robot state $\mathbf{x}_t$ to $\mathbf{x}_{t+1}$ using the inputs of the system $\mathbf{u}_t$. Such input measurements can come from an Inertial Measurement Unit (IMU) or wheel odometry

for example. In order to reduce the scope of this thesis, a *constant velocity* model is used instead, which is a simpler model that uses no input measurements. The continuous-time constant velocity model is

$$\dot{\mathbf{x}}_t = A_{\text{ct}}\mathbf{x}_t, \qquad A_{\text{ct}} = \begin{pmatrix} 0_{3\times3} & \mathcal{I}_3 \\ 0_{3\times3} & 0_{3\times3} \end{pmatrix}, \tag{3-2}$$

and follows from the fact that the state elements $v_{x,t}, v_{y,t}, \omega_t$ are the derivatives of state elements $p_{x,t}, p_{y,t}, \theta_t$ respectively. Using the zero-order hold discretization method, the discrete-time dynamics matrix A is obtained which is denoted by

$$A = \begin{pmatrix} \mathcal{I}_3 & c_{\text{dt}}\mathcal{I}_3 \\ 0_{3\times3} & \mathcal{I}_3 \end{pmatrix}, \tag{3-3}$$

where c_{dt} denotes the time between each Lidar scan, which is 0.1 sec in this thesis. We model the Lidar's velocity as a random-walk process by adding the noise term $\mathbf{w}_t$ to the velocity in order to accommodate for model inaccuracies. The discrete-time dynamic model can now be defined as

$$\mathbf{x}_{t+1} = A\mathbf{x}_t + \mathbf{w}_t, \qquad \mathbf{w}_t \sim \mathcal{N}(\mathbf{0}, Q), \tag{3-4a}$$

$$Q = \begin{pmatrix} 0_{3\times3} & 0_{3\times3} \\ 0_{3\times3} & Q_{\text{vel}} \end{pmatrix}. \tag{3-4b}$$

The process noise covariance matrix Q_{vel} is a tuning parameter that describes how much the constant velocity assumption is trusted. The model becomes inaccurate when the Lidar undergoes acceleration, but this is not considered a problem as pose estimation accuracy at static waypoints is the primary focus of this thesis. Note that the model is linear.

3-1-2 Map representation

The process of building and maintaining a map is an integral part of SLAM. The map representation plays a vital role in the Lidar measurement model, and is therefore discussed first. The core idea of Ray-SLAM is to add information about the scene to the SLAM problem by assuming that walls are available. This work attempts to realize improved accuracy by focusing on indoor scenes specifically rather than on general scenes. Figure 3-1 shows an indoor scene where a construction robot could be deployed to give the reader a general sense of the considered use cases. Typical challenges become visible such as the large amount of clutter in the scene and (partly) obscured walls. Furthermore, we let go of the need to model the scene in its entirety, and only model enough parts (i.e. walls) of the scene to enable localization. As such, we can avoid to attempt modeling objects in the scene with complex geometry (e.g. a pile of pallets). This is beneficial, because inaccurate models of such objects inevitably introduce errors in the localization process. This design philosophy differentiates Ray-SLAM from other SLAM algorithms, as other algorithms aim for applicability to general scenes where the assumption that walls are present may not hold (e.g. any natural environment), and try to capture mostly the entire scene in its map (see §2-3-2).

To further specify the 2D map representation that is suitable for the design objective of accurate indoor localization, the following considerations were made: (1) Walls can easily

Figure 3-1: A typical scene where a construction robot could be deployed to automate taks such as drilling holes [1]. Walls are available that the robot can use as reference for localization. The environment is cluttered with construction materials. In this scene particular since, a 2D SLAM algorithm can be an appropriate solution since the floor is flat and the walls are upright.

be modeled by line segments in 2D. It was chosen therefore to build the map from 2D line segments. (2) Walls may have discontinuities that interrupt the flat surface (e.g. due to a fire extinguisher mounted onto the wall) and slight curvature may exist on a cm-level. To localize accurately using walls as reference, it is chosen to use disjoint line segments with a fixed length (for example 0.5 m) to represent the walls. A large wall can now be modeled by multiple line segments to capture potential curvature. The line segments are to be placed away from discontinuities such as the edge of the wall or objects attached onto the wall. The modeling of walls by such line segments is illustrated in Figure 3-2. (3) In order to optimize the line segment that represents the wall, the line segment should be parameterized in a minimal way such that these parameters can be estimated by solving an optimization problem.

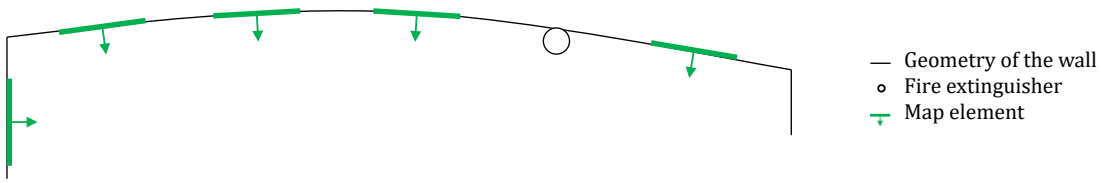


Figure 3-2: A toy example of a building's cross section. A wall may have curvature and the flat surface can be interrupted by e.g. a fire extinguisher. Line segments (green) are used to model the wall and are referred to as *map elements*. The map elements are placed away from the edge of the wall, the fire extinguisher and each other such that only the (nearly) flat surfaces are used as a references for localization.

Based on these considerations the *map element* is now introduced, which is the main feature in the map representation used by the Ray-SLAM algorithm. One map element $\mathcal{M}_t^i$ is a line segment that is parameterized by (1) a fixed initial pose (position and heading), (2) a time varying heading deviation and (3) a time varying position deviation in the direction of the map element's heading. These deviations are referred to as the map element *offsets*. The map element's length is fixed to $2s_r$, where radius s_r is a tuning variable. The time invariant initial pose is denoted by $\mathcal{M}_{\text{pose}}^i = (p_{\mathcal{M},x}^i \quad p_{\mathcal{M},y}^i \quad \varphi^i)$ and can be interpreted as a coarse initial estimate of the wall's local geometry based on the observations in a single Lidar scan. The heading- and position offsets are denoted by $d_{\text{pos},t}^i, d_{\varphi,t}^i$. The total heading of the map element is equal to $\varphi^i + d_{\varphi,t}^i$ such that $d_{\varphi,t}^i$ remains close to zero. The map element is visualized in Figure 3-3.

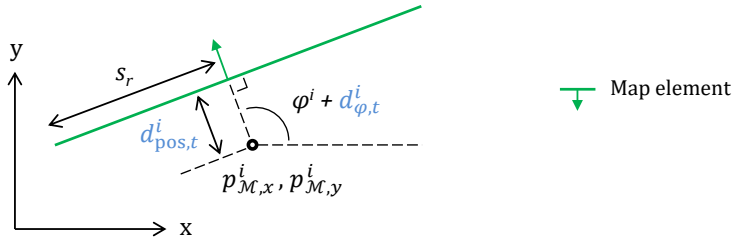


Figure 3-3: Parameterization of the map elements. The i^{th} map element $\mathcal{M}_t^i$ is shown as a green line segment. The element has a fixed origin $p_{\mathcal{M},x}^i, p_{\mathcal{M},y}^i$ and heading φ^i. The position offset $d_{\text{pos},t}^i$ and heading offset $d_{\varphi,t}^i$ in blue are (unknown) model parameters. All map elements have the same radius s_r for simplicity.

The heading- and position offsets of the map elements are considered (unknown) model parameters, and the objective of the estimation algorithm will be to estimate these offsets as accurately as possible given the available sensor data. Only two out of three degrees of freedom are captured as model parameters such that the map element cannot move parallel to the wall. The uncertainty of the estimates $\hat{d}_{\text{pos},t}^i, \hat{d}_{\varphi,t}^i$ are denoted by $\sigma_{d,\text{pos},t}^i, \sigma_{d,\varphi,t}^i$.

The notation $\mathcal{M}_{\text{d},t}$ is used to denote all map offsets $d_{\text{pos},t}^i, d_{\varphi,t}^i \in \mathcal{M}_t$. Similarly, $\mathcal{M}_{\sigma,t}$ denotes all map offset uncertainties $\sigma_{d,\text{pos},t}^i, \sigma_{d,\varphi,t}^i \in \mathcal{M}_t$. The Gaussian posterior distribution of the map element offsets is written as $p(\mathcal{M}_{\text{d},t}^i)$. To collect all information of one map element, we denote it by

$$\mathcal{M}_t^i = (\underbrace{p_{\mathcal{M},x}^i \quad p_{\mathcal{M},y}^i \quad \varphi^i}_{\mathcal{M}_{\text{pose}}^i} \quad \underbrace{d_{\text{pos},t}^i \quad d_{\varphi,t}^i}_{\mathcal{M}_{\text{d},t}^i} \quad \underbrace{\sigma_{d,\text{pos},t}^i \quad \sigma_{d,\varphi,t}^i}_{\mathcal{M}_{\sigma,t}^i}). \tag{3-5}$$

The map $\mathcal{M}_t$ can now be defined as a collection of $N_{\mathcal{M}}$ map elements, i.e.

$$\mathcal{M}_t = \left\{ \mathcal{M}_t^1, \mathcal{M}_t^2, \ldots, \mathcal{M}_t^{N_{\mathcal{M}}} \right\}. \tag{3-6}$$

A visual overview is given in Figure 3-4, showing the Lidar's trajectory, map elements and geometry of the building.

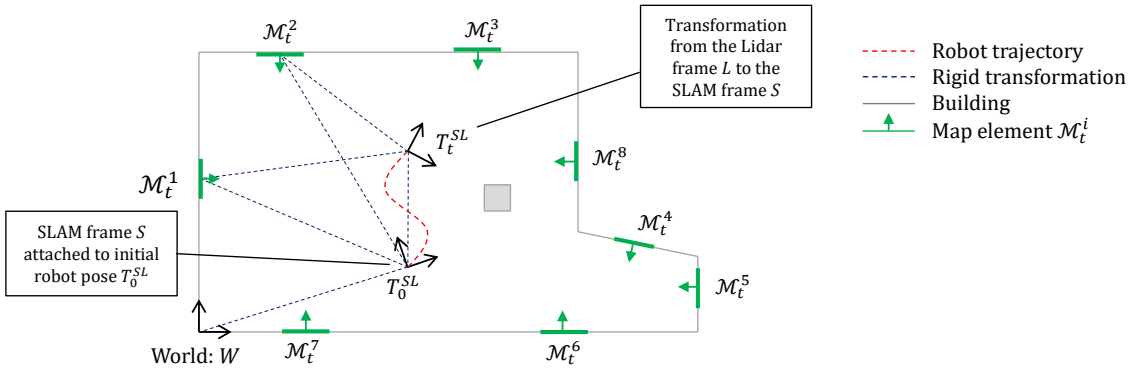


Figure 3-4: Schematic overview of the reference frames of the SLAM problem and their connections. An example indoor scene is displayed.

It should be noted that the map representation can be generalized to 3D, see Section 5-2. The hypothetical 3D map element could for example be a disc-shaped surface with radius s_r and origin $\mathbf{p} \in \mathbb{R}^3$, and would have three degrees of freedom: roll, pitch and position offset in the direction of the surface normal.

3-1-3 Lidar measurement model

Since the SLAM problem is simplified to the 2D case, only the Lidar's scan ring is used which has its pitch angle closest to zero (see Figure 2-1). Each range measurement has associated azimuth angle. We define the set of range measurements and azimuth angles from a single scan rotation by

$$\mathcal{Y}_t = \left\{ y_t^1, y_t^2, \ldots, y_t^{N_l} \right\}, \tag{3-7a}$$

$$\mathcal{A}_t = \left\{ \alpha_t^1, \alpha_t^2, \ldots, \alpha_t^{N_l} \right\}, \tag{3-7b}$$

where N_l denotes the number of Lidar observations in a scan. Note that the time stamp t is identical for each observation, neglecting 'rolling shutter' effect of the rotating Lidar, which is common practice in Lidar SLAM where no process is implemented to undistort a scan. The azimuth values are obtained directly from the rotary encoder of the Lidar sensor, and may change over time due to a slight shift in sampling within the interval of $0 \leq \alpha_t^i < 2\pi$. The Lidar's pose together with the Lidar observations are illustrated in Figure 3-5 (left).

Contrary to existing works that typically compare two point clouds using the Iterative Closest Point (ICP) algorithm to infer a pose, the approach was chosen to model each range measurement from the Lidar directly in order to prevent ICP related errors (see §2-2-3). Making use of the map representation from §3-1-2, we can introduce the range measurement model as

$$y_t^i = h(\mathbf{x}_t, \alpha_t^i, \mathcal{M}_t^j) + e_{\mathrm{r},t}^i, \qquad e_{\mathrm{r},t}^i \sim \mathcal{N}(0, \sigma_r^2), \tag{3-8}$$

where j represents the index value of the map element that was detected by the i^{th} range measurement a time point t (matching i and j is discussed in the next section). Recall that α_t^i is the azimuth angle of the Lidar. The noise term $e_{\mathrm{r},t}^i$ is modeled as zero-mean additive

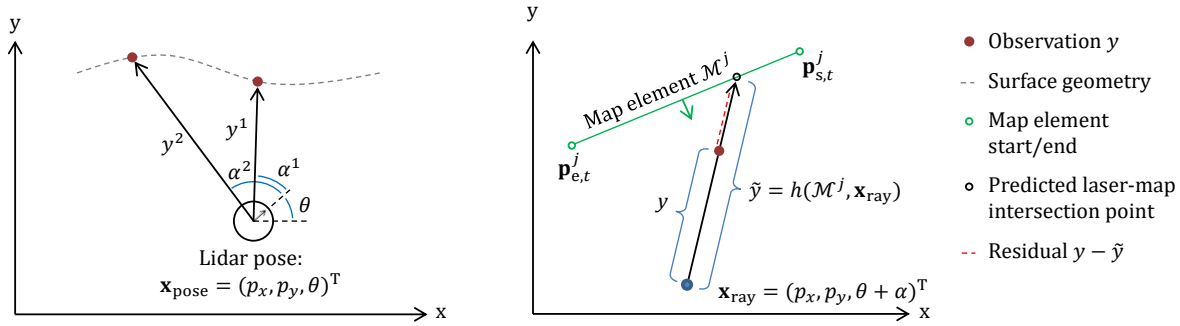


Figure 3-5: Left. Schematic representation of the Lidar observations. **Right.** Visual representation of the ray casting function, which provides the measurement prediction $\tilde{y}$ by taking the Euclidean distance between the Lidar's position and the predicted intersection point between the laser beam and map element.

Gaussian noise with a standard deviation of σ_r. In reality, the measurement error $e^i_{\text{r},t}$ may have a bias and covariance that depend on e.g. the objects shape, reflectivity and measurement inclination angle. It is standard practice in Lidar SLAM however to assume that the error signal is a zero mean Gaussian noise signal with a standard deviation of σ_r, and it is up to the sensor manufacturer to calibrate the Lidar device such that the bias is minimized. The validity of the Gaussian noise assumption is studied in Appendix C. It is shown that the range measurement noise indeed follows a Gaussian distribution, but the experiment was unable to reveal the mean of the error (biasedness).

The measurement prediction function $h(\cdot)$ used in this thesis is defined by the 2D simplification of the Ray/Triangle Intersection algorithm [50], and is referred to as *ray casting*. Ray casting can intuitively be interpreted as the Euclidean distance between the Lidar's position and the predicted intersection point between the laser beam and map element, see Figure 3-5 (right). Let $\mathbf{p}^j_{\text{s},t}, \mathbf{p}^j_{\text{e},t}$ represent the Cartesian start- and end point coordinates of map element $\mathcal{M}^j_t$ and let $\mathbf{x}_{\text{pos},t}$ represent the Lidar's Cartesian coordinates in the SLAM frame S. Furthermore let $\mathbf{r}^i_t$ represent the normalized direction vector of the laser beam using

$$\mathbf{r}^i_t = \begin{pmatrix} \cos(\alpha^i_t + \theta_t) & \sin(\alpha^i_t + \theta_t) \end{pmatrix}^\top \tag{3-9}$$

in the SLAM frame. The predicted range measurement $\tilde{y}^i_t$ of a single laser beam can now be modeled by

$$h(\mathbf{x}_t, \alpha^i_t, \mathcal{M}^j_t) = \frac{(\mathbf{p}^j_{\text{s},t} - \mathbf{x}_{\text{pos},t}) \times (\mathbf{p}^j_{\text{e},t} - \mathbf{p}^j_{\text{s},t})}{\mathbf{r}^i_t \times (\mathbf{p}^j_{\text{e},t} - \mathbf{p}^j_{\text{s},t})} \tag{3-10}$$

where $\times$ denotes the two-dimensional cross product. Note that the function $h(\cdot)$ is non-linear.

3-1-4 Finding correspondences between the Lidar observations and the map elements

It can be seen that the model of (3-8) introduces a challenge: it is not known a-priori which observation i detects which map element j. This problem is referred to as *correspondence*

finding. Some of these measurements may detect a map element, and others may detect clutter in the scene or a part of a wall that is not represented by a map element. The result of the correspondence finding process yields the following three sets:

$$\mathcal{Y}_{\mathcal{C},t} = \left\{ y^1_{\mathcal{C},t}, y^2_{\mathcal{C},t}, \dots, y^{N_\mathcal{C}}_{\mathcal{C},t} \right\} \subseteq \mathcal{Y}_t, \tag{3-11a}$$

$$\mathcal{A}_{\mathcal{C},t} = \left\{ \alpha^1_{\mathcal{C},t}, \alpha^2_{\mathcal{C},t}, \dots, \alpha^{N_\mathcal{C}}_{\mathcal{C},t} \right\} \subseteq \mathcal{A}_t, \tag{3-11b}$$

$$\mathcal{N}_{\mathcal{C},t} = \left\{ n^1_{\mathcal{C},t}, n^2_{\mathcal{C},t}, \dots, n^{N_\mathcal{C}}_{\mathcal{C},t} \right\}. \tag{3-11c}$$

where $\mathcal{Y}_{\mathcal{C},t}, \mathcal{A}_{\mathcal{C},t}$ only contain Lidar measurements that detected a map element. The amount of correspondences is denoted by $N_\mathcal{C}$. The integer variable $n^i_{\mathcal{C},t} \in \mathcal{N}_{\mathcal{C},t}$ represent the map element index that was detected by the i^{th} Lidar observation respectively, and as such $\mathcal{N}_{\mathcal{C},t}$ acts as a link table. The correspondence object $\mathcal{C}_t$ is furthermore defined by

$$\mathcal{C}_t = \{\mathcal{Y}_{\mathcal{C},t}, \mathcal{A}_{\mathcal{C},t}, \mathcal{N}_{\mathcal{C},t}\} \tag{3-12}$$

as the collection of the correspondence information. The function $k(\cdot)$ is introduced which has the objective of finding the correspondences and has the form of

$$\mathcal{C}_t = k(\mathbf{x}_t, \mathcal{M}_t, \mathcal{Y}_t, \mathcal{A}_t). \tag{3-13}$$

The implemented solution of the correspondence finding function $k(\cdot)$ is detailed in §3-2-5.

3-2 Problem formulation and algorithm implementation details

The previous section has laid the foundation of the Ray-SLAM algorithm by introducing the state-space model and map representation. For the SLAM algorithm to function, we need the following three sub-problems to be solved: (1) estimation of the state and map element offsets, (2) spawning new map elements and (3) finding correspondences between the Lidar observations and map elements. The implemented solutions to these sub-problems are referred to as Ray-SLAM's *components*. Components (2) and (3) have direct access to the raw Lidar measurements and are considered the front-end of Ray-SLAM, while component (1) is considered the back-end. In order to get a better overview of how these components work together, a flow chart description is given in Figure 3-6, which is closely related to the flow chart of Figure 1-3. This section now proceeds with formalizing these sub-problems and describes the algorithmic components to solve them in detail.

Parameter initialization. The *stationary indicator* boolean variable b_{stat} is introduced to specify whether the Ray-SLAM algorithm should switch to SLAM or localization only. This variable is initialized by $b_{\text{stat}} = \text{True}$. The state initializes by setting $\mathbf{x}_0 = (\breve{\mathbf{x}}^\top_{\text{pose},0} \quad 0 \quad 0 \quad 0)^\top$ based on the assumption that the Lidar is stationary during start-up. The initial pose $\breve{\mathbf{x}}_{\text{pose},0}$ is provided by the user.

3-2-1 Loosely coupled estimation of the map and state

We now consider the problem of estimating the state trajectory $\mathbf{x}_{1:t}$ and map $\mathcal{M}$ given the sensory data $\mathcal{Y}_{0:t}$ and initial robot pose $\mathbf{x}_0$. This thesis assumes the Markov property between

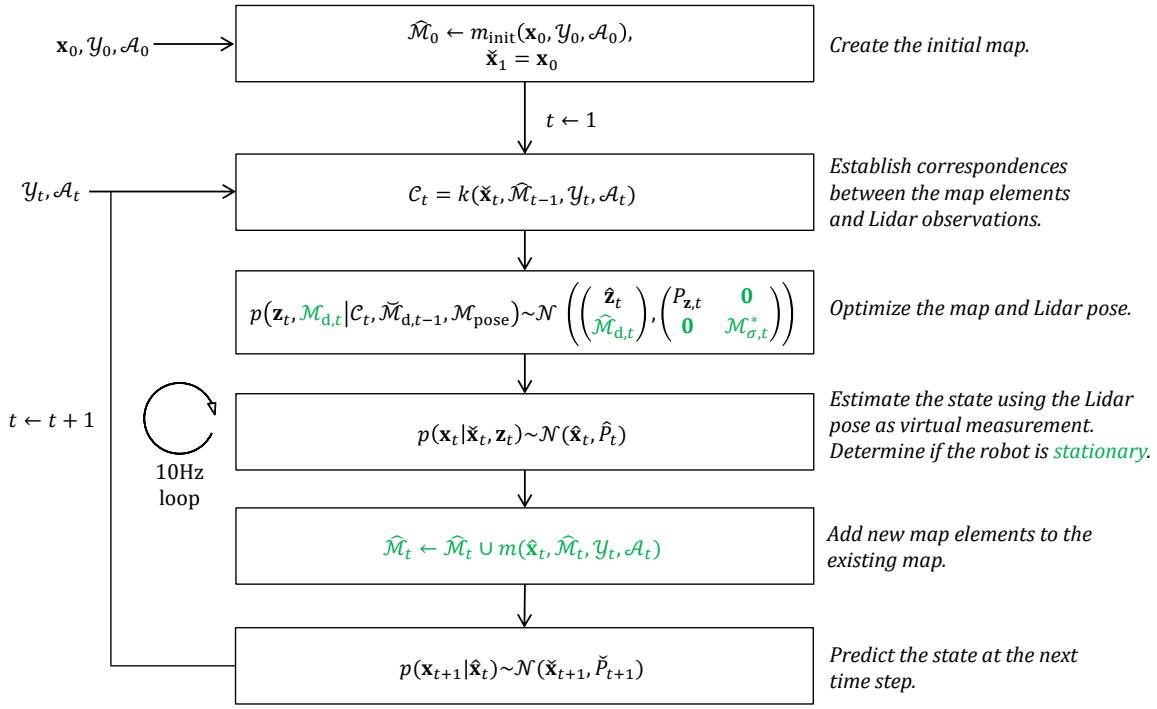


Figure 3-6: A flow chart description of the Ray-SLAM algorithm using the algorithmic components. Once the initial map is constructed, the algorithm carries out SLAM in a 10 Hz loop. The color green indicates steps that are carried out only if the Lidar is considered to be stationary. The front-end functions $m_{\text{init}}(\cdot)$, $m(\cdot)$ and $k(\cdot)$ have direct access to the raw sensor data $\mathcal{Y}_t$. The estimation problems indicated by $p(\cdot)$ are the back-end of Ray-SLAM. The asterisk at $\mathcal{M}^*_{\sigma,t}$ is placed to indicate a diagonal matrix approximation.

successive states $\mathbf{x}_{t-1:t}$, which is common practice in pose estimation [21], such that the problem can be solved incrementally. The map offset estimates at $t-1$ are used as a prior in order to incrementally improve these estimates based on new observations. The map offset prior is denoted by $\breve{\mathcal{M}}_{\mathrm{d},t-1}$.

The incremental pose- and map estimation problem is now introduced as the following conditional probability density function:

$$p\left(\mathbf{x}_t, \mathcal{M}_{\mathrm{d},t} \mid \mathcal{C}_t, \mathbf{x}_{t-1}, \breve{\mathcal{M}}_{\mathrm{d},t-1}, \mathcal{M}_{\mathrm{pose}}\right) \sim \mathcal{N}\left(\begin{pmatrix} \hat{\mathbf{x}}_t \\ \hat{\mathcal{M}}_{\mathrm{d},t} \end{pmatrix}, \begin{pmatrix} P_t & 0 \\ 0 & P_{\mathcal{M},t} \end{pmatrix}\right) \tag{3-14}$$

where the posterior is approximated by a Gaussian distribution. Note that $\mathcal{C}_t$ represents the encapsulated Lidar observations. Rather than considering the optimization problem of (3-14), this thesis decouples the problem into two subproblems:

$$p\left(\mathbf{z}_t, \mathcal{M}_{\mathrm{d},t} \mid \mathcal{C}_t, \breve{\mathcal{M}}_{\mathrm{d},t-1}, \mathcal{M}_{\mathrm{pose}}\right), \tag{3-15a}$$

$$p\left(\mathbf{x}_t \mid \mathbf{x}_{t-1}, \mathbf{z}_t\right), \tag{3-15b}$$

where $\mathbf{z}_t$ is introduced as a placeholder variable that is directly related to $\mathbf{x}_{\mathrm{pose},t}$. Subproblem (3-15a) uses the Lidar observations in $\mathcal{C}_t$ and the prior map offsets $\breve{\mathcal{M}}_{\mathrm{d},t-1}$ to jointly estimate the Lidar pose $\mathbf{z}_t$ as a *virtual measurement* and map offsets $\mathcal{M}_{\mathrm{d},t}$ while disregarding the temporal relation between $\mathbf{x}_{t-1}$ and $\mathbf{x}_t$. The virtual pose measurement is distributed as $\mathbf{z}_t \sim \mathcal{N}(\mathbf{x}_{\mathrm{pose},t}, P_{\mathbf{z},t})$ and is part of $SE(2)$. Subproblem (3-15b) consequently estimates the full state $\mathbf{x}_t$ using $\mathbf{x}_{t-1}$ and $\mathbf{z}_t$ of (3-15a) while considering the system's dynamics.

Such a decoupled approach for state estimation is often referred to in the literature as *loosely coupled*. Using this approach, it is important that when summarizing the measurement- and map information into $\mathbf{z}_t$ in (3-15a), no information gets lost that is relevant for estimating $\mathbf{x}_t$ in (3-15b). Since the Lidar range measurements can only measure distance (and not velocity), this is assumed to be the case. The decoupled approach furthermore assumes that the subproblems are statistically independent, which is believed to be true since the measurement noise $e^i_{\mathrm{r},t}$ acting on (3-15a) and the process noise $\mathbf{w}_t$ acting on (3-15b) are unrelated processes.

This loosely coupled approach was chosen for several reasons. First, we note that the virtual pose measurement model takes the shape of

$$\mathbf{z}_t = C\mathbf{x}_t + \mathbf{e}_{\mathbf{z},t}, \quad \mathbf{e}_{\mathbf{z},t} \sim \mathcal{N}(0, P_{\mathbf{z},t}), \tag{3-16}$$

with the observation matrix $C = (\mathcal{I}_3 \quad 0_{3\times 3})$, which is a linear model. This observation matrix follows from the fact that $\mathbf{z}_t$ and $\mathbf{x}_{\mathrm{pose},t}$ are directly related. Since the dynamics of the system in (3-4) are also described by a linear model, we can now use a linear Kalman filter [23] to solve the estimation problem of (3-15b). The linear Kalman filter is well understood, computationally efficient and easy to implement. Furthermore, solving the nonlinear estimation problem of (3-15a) may rely on a solver that implements automatic (analytical and/or numerical) differentiation. As such, implementation of e.g. an iterated extended Kalman filter can be avoided, and there is no need to manually establish the Jacobian matrix of the optimization problem.

3-2-2 Nonlinear estimation of the virtual pose measurement and map.

Problem formulation. We now establish the derivation of the nonlinear least squares form of the estimation problem of (3-15a). The nonlinear least squares problem is a widely studied problem class, and efficient solver algorithms are available to solve such a problem [25, 51]. First, we use the conditioning rule to establish that

$$p\left(\mathbf{z}_t, \mathcal{M}_{\mathrm{d},t} \mid \mathcal{C}_t, \breve{\mathcal{M}}_{\mathrm{d},t-1}, \mathcal{M}_{\mathrm{pose}}\right) = p\left(\mathcal{M}_{\mathrm{d},t} \mid \breve{\mathcal{M}}_{\mathrm{d},t-1}\right)\ p\left(\mathbf{z}_t \mid \mathcal{C}_t, \mathcal{M}_{\mathrm{d},t}, \mathcal{M}_{\mathrm{pose}}\right). \tag{3-17}$$

Independent conditioning terms have been dropped. We now split up the set of correspondences $\mathcal{C}_t$ into its three components $\mathcal{Y}_{\mathcal{C},t}, \mathcal{A}_{\mathcal{C},t}, \mathcal{N}_{\mathcal{C},t}$. Using Bayes rule on the probability term of $\mathbf{z}_t$, we can establish that

$$p(\mathbf{z}_t \mid \mathcal{Y}_{\mathcal{C},t}, \mathcal{A}_{\mathcal{C},t}, \mathcal{N}_{\mathcal{C},t}, \mathcal{M}_{\mathrm{d},t}, \mathcal{M}_{\mathrm{pose}}) \propto p\left(\mathcal{Y}_{\mathcal{C},t} \mid \mathbf{z}_t, \mathcal{A}_{\mathcal{C},t}, \mathcal{N}_{\mathcal{C},t}, \mathcal{M}_{\mathrm{d},t}, \mathcal{M}_{\mathrm{pose}}\right) \tag{3-18}$$

where $\propto$ denotes proportionality. We now expand the statistically independent range measurements in $\mathcal{Y}_{\mathcal{C},t}$ to obtain that (3-17) is proportional to

$$p\left(\mathcal{M}_{\mathrm{d},t} \mid \breve{\mathcal{M}}_{\mathrm{d},t-1}\right) \prod_{i=1}^{N_{\mathcal{C}}} p\left(y^i_{\mathcal{C},t} \mid \mathbf{z}_t, \alpha^i_{\mathcal{C},t}, \mathcal{M}^{n^i_{\mathcal{C},t}}_{\mathrm{d},t}, \mathcal{M}^{n^i_{\mathcal{C},t}}_{\mathrm{pose}}\right). \tag{3-19}$$

Recall that $N_{\mathcal{C}}$ denotes the amount of correspondences, and the term $n^i_{\mathcal{C},t} \in \mathcal{N}_{\mathcal{C},t}$ in superscript contains the index value of the map element to which the i^{th} Lidar range measurement corresponds. We can now see the probabilistic Lidar measurement model of (3-8) on the right-hand side respectively. In practice, rather than maximizing the probability of (3-19), we typically minimize the negative log likelihood [21], resulting in the optimization problem of

$$\left\{\hat{\mathbf{z}}_t, \hat{\mathcal{M}}_{\mathrm{d},t}\right\} = \underset{\mathbf{z}_t, \mathcal{M}_{\mathrm{d},t}}{\arg\min} - \log p\left(\mathcal{M}_{\mathrm{d},t} \mid \breve{\mathcal{M}}_{\mathrm{d},t-1}\right) - \log \sum_{i=1}^{N_{\mathcal{C}}} p\left(y^i_{\mathcal{C},t} \mid \mathbf{z}_t, \alpha^i_{\mathcal{C},t}, \mathcal{M}^{n^i_{\mathcal{C},t}}_{\mathrm{d},t}, \mathcal{M}^{n^i_{\mathcal{C},t}}_{\mathrm{pose}}\right). \tag{3-20}$$

Since the range measurement noise $e^i_{\mathrm{r},t}$ and the map offset prior $\breve{\mathcal{M}}_{\mathrm{d},t}$ have a Gaussian distribution, the optimization problem (3-20) reduces to a nonlinear least squares problem. By denoting the measurement prediction $\tilde{y}^i_{\mathcal{C},t} = h(\mathbf{z}_t, \alpha^i_{\mathcal{C},t}, \mathcal{M}^{n^i_{\mathcal{C},t}}_{\mathrm{d},t}, \mathcal{M}^{n^i_{\mathcal{C},t}}_{\mathrm{pose}})$ and using the previous map offset estimates $\hat{d}^i_{t-1}$ and their uncertainties $d^i_{\sigma,t}$ as the map offset prior, this nonlinear least squares optimization problem is formulated as

$$\left\{\hat{\mathbf{z}}_t, \hat{\mathcal{M}}_{\mathrm{d},t}\right\} = \underset{\mathbf{z}_t, \mathcal{M}_{\mathrm{d},t}}{\arg\min} \sum_{d^i_t \in \mathcal{M}_{\mathrm{d},t}} (\sigma^i_{\mathrm{d},t-1})^{-2}(d^i_t - \hat{d}^i_{t-1})^2 \ + \ \sigma_r^{-2} \sum_{i=1}^{N_{\mathcal{C}}} (y^i_{\mathcal{C},t} - \tilde{y}^i_{\mathcal{C},t})^2. \tag{3-21}$$

The compact notation $d^i_t, \sigma^i_{\mathrm{d},t}$ refers to both the position- and heading offsets in $\hat{\mathcal{M}}^i_t$ together. Note that the squared map prior and observation residuals are weighted by their inverse variance $(\sigma^i_{\mathrm{d},t-1})^{-2}, \sigma_r^{-2}$ respectively. The Lidar scan may be corrupted by motion distortion. When the estimated velocity $||\hat{\mathbf{x}}_{\mathrm{vel},t}||_2$ exceeds some threshold, the optimization problem of (3-21) is reduced to pose estimation only using

$$\hat{\mathbf{z}}_t = \underset{\mathbf{z}_t}{\arg\min}\, \sigma_r^{-2} \sum_{i=1}^{N_{\mathcal{C}}} \left(y^i_{\mathcal{C},t} - \tilde{y}^i_{\mathcal{C},t}\right)^2 \tag{3-22}$$

with the modification that the measurement prediction $\tilde{y}^i_{\mathcal{C},t}$ in (3-22) is based on $\hat{\mathcal{M}}_{\mathrm{d},t-1}$. Note that the optimization problems of (3-21) and (3-22) are not the final optimization problems implemented by Ray-SLAM, this will follow in the next section.

Implementation details. In order to arrive at the specific nonlinear least squares optimization problem as implemented in Ray-SLAM, the following steps are carried out: (1) Introduction of the beam angle discount factor. (2) Reduction of the optimization problem's size by introduction of the map optimization vector.

Introduction of the beam angle discount factor. In model based optimization, it is important to define the error term (such as $y^i_{\mathcal{C},t} - \tilde{y}^i_{\mathcal{C},t}$ in (3-21)) accurately to describe the mismatch between the data and the model. The optimization algorithm relies on this error term to find the unknown parameters $\mathbf{z}_t, \mathcal{M}_{\mathrm{d},t}$ that minimizes the sum of squared error terms, resulting in $\hat{\mathbf{z}}_t, \hat{\mathcal{M}}_{\mathrm{d},t}$ with the least model mismatch. Depending on how properly the error term is modeled, the (local) minimum at $\hat{\mathbf{z}}_t, \hat{\mathcal{M}}_{\mathrm{d},t}$ can be either close or far away from its true value. This section takes a closer look at different error terms related to Lidar SLAM.

Conventional Lidar SLAM algorithms project the observed laser point $\{y^i_t, \alpha^i_t\}$ onto the feature to which it is associated, resulting in the point-to-feature error term [18]. Depending on the 3D feature type the point-to-plane or point-to-line error is obtained. Since this thesis is concerned with the 2D case, we now take a closer look at the point-to-line error term e_{p2l}, which is shown in Figure 3-7. This error term implicitly assumes an orthogonal beam angle $\beta^i_t = 0$ for each Lidar observation, where the beam angle β^i_t is defined as the angle of the laser beam with respect to the map feature normal. This assumption is not true in general, as each measurement comes in at a different angle up to $\beta^i_t < 90°$ depending on the orientation of the feature with respect to the Lidar's position.

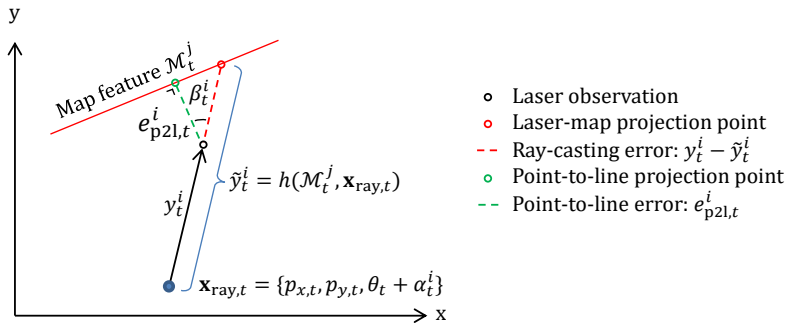


Figure 3-7: The ray casting laser beam error model vs. the conventional point-to-line error model. The difference between these error terms depend on the beam angle β^i_t.

This thesis uses a different formulation of the error term, which is based on ray casting respectively and is denoted by

$$e^i_{\mathrm{rc},t} = y^i_{\mathcal{C},t} - \tilde{y}^i_{\mathcal{C},t} \tag{3-23}$$

which is defined as the difference between the measurement and the prediction based on (3-10). Note that the error term $e^i_{\mathrm{rc},t}$ is different from $e^i_{\mathrm{p2l},t}$, and this difference depends on β^i_t. As can be seen in Figure 3-7, the ray-casting and point-to-line error terms are geometrically related using

$$e^i_{\mathrm{p2l},t} = \cos(\beta^i_t) e^i_{\mathrm{rc},t}. \tag{3-24}$$

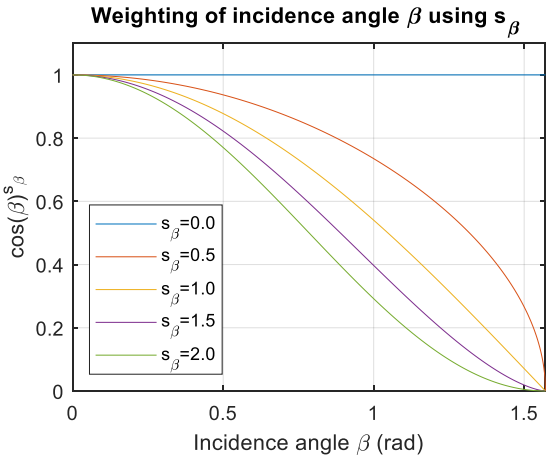


Figure 3-8: The weighting function $\cos(\beta)^{s_\beta}$ allows the user to reduce or increase the weight of observations with a steep inclination angle β using the tuning parameter s_β. Setting the weighting factor $s_\beta = 1.0$ results in the point-to-line error metric, where each measurement residual is weighted such that an observed point results the same residual regardless of incidence angle β.

In order to implement both the ray-casting and point-to-line error metrics into Ray-SLAM, the generalized laser beam error term $e^i_{\text{gen},t}$ is introduced as

$$e^i_{\text{gen},t} = \cos(\beta^i_t)^{s_\beta}(y^i_{\mathcal{C},t} - \tilde{y}^i_{\mathcal{C},t}) \tag{3-25}$$

where s_β is introduced as the *beam angle discount factor* tuning parameter. Raising the term $\cos(\beta^i_t)$ to the power of s_β sets the error term $e^i_{\text{gen},t}$ to ray-casting for $s_\beta = 0$, and to point-to-line for $s_\beta = 1$ respectively. It should be noted that s_β is not restricted to just 0 or 1, see Figure 3-8. This thesis will investigate which $s_\beta \in \mathbb{R}$ results in pose estimation with the highest accuracy with respect to the ground-truth.

Introduction of the map optimization vector. The nonlinear optimization problem of (3-21) attempts to optimize all the map offsets (position offsets and/or heading offsets) $d^i_t \in \mathcal{M}_{\text{d},t}$. In practice, however, not all of the map element offsets need to be optimized because map elements may not be observed due to occlusion. Also, map elements can be excluded from optimization by setting $\sigma^i_{\text{d},t} = 0$ as a flag to reduce the size of the optimization problem (see the next paragraph for more details). Therefore, we introduce the map optimization vector $\mathbf{d}_{\text{opt},t}$, which the subset of all map offsets that take part in optimization such that

$$\mathbf{d}_{\text{opt},t} \subseteq (d^1_{\text{pos},t} \quad d^1_{\varphi,t} \quad \dots \quad d^{N_\mathcal{M}}_{\text{pos},t} \quad d^{N_\mathcal{M}}_{\varphi,t})^\top. \tag{3-26}$$

The resulting length of vector $\mathbf{d}_{\text{opt},t}$ is denoted by $N_{\mathbf{d},\text{opt}}$. The full optimization vector consists of both the virtual pose measurement $\mathbf{z}_t$ and $\mathbf{d}_{\text{opt},t}$. Map offsets that are excluded from optimization are: (1) map offsets with a standard deviation $\sigma^i_{\text{d},t} = 0$ as stored in $\hat{\mathcal{M}}_t$ and (2) map offsets related to map features without any correspondences as indicated by $\mathbf{b}_{\mathcal{M},t}$. Furthermore, if movement was detected by $b_{\text{stat}} = \text{False}$, all map offsets are excluded such that $N_{\mathbf{d},\text{opt}} = 0$. This reduces the optimization problem from SLAM to localization only. The

optimization problem as implemented in Ray-SLAM is

$$\left\{\hat{\mathbf{z}}_t, \hat{\mathbf{d}}_{\text{opt},t}\right\} = \underset{\mathbf{z}_t, \mathbf{d}_{\text{opt},t}}{\arg\min} \sum_{d_t^i \in \mathbf{d}_{\text{opt},t}} (\sigma_{\text{d},t-1}^i)^{-2} (d_t^i - \hat{d}_{t-1}^i)^2 \; + \; \sigma_r^{-2} \sum_{\mathcal{C}_t^i \in \mathcal{C}_t} \left(\cos(\beta_t^i)^{s_\beta} (y_{\mathcal{C},t}^i - \tilde{y}_{\mathcal{C},t}^i)\right)^2. \tag{3-27}$$

As initial guess, we use the predicted pose $\check{\mathbf{x}}_{\text{pose},t}$ and the map element offsets $\hat{\mathcal{M}}_{\text{d},t-1}$. The Levenberg-Marquardt method is selected as the solver algorithm of MATLAB's `lsqnonlin`$(\cdot)$ function to find $\{\hat{\mathbf{z}}_t, \hat{\mathbf{d}}_{\text{opt},t}\}$ that minimizes the cost of (3-27). The solver uses numerical differentiation to obtain the required Jacobian matrices.

Extract the pose- and map covariance matrices and update the map. Besides the solution of (3-27), we are also interested in the uncertainty (covariance) of this solution, which is extracted using a standard procedure from the literature [21, 51] and works as follows. The function `lsqnonlin`$(\cdot)$ provides the Jacobian matrix J at the (local) minimum of the cost function (3-27), which is calculated using numerical differentiation. We now take the upper-left 3×3 block of $J^\top J$ and invert it to obtain the pose covariance matrix $P_{\mathbf{z},t}$. The lower-right block $J^\top J$ of size $N_{\mathbf{d},\text{opt}} \times N_{\mathbf{d},\text{opt}}$ is inverted to obtain the covariance matrix of the map optimization vector $P_{\mathbf{d},\text{opt},t}$.

The map element offsets and their uncertainties in $\hat{\mathcal{M}}_{t-1}$ are overwritten by $\hat{\mathbf{d}}_{\text{opt},t}$ and the diagonal values of $P_{\mathbf{d},\text{opt},t}$ respectively in order to obtain the improved map offset estimates $\hat{\mathcal{M}}_{\text{d},t}$ and their uncertainties $\mathcal{M}_{\sigma,t}$. It was chosen to neglect the off-diagonal covariance values of $P_{\mathbf{d},\text{opt},t}$ because this allowed the code to be simplified. During the SLAM process, quite often the Lidar is at a position where not all the map elements are observed. Using a full covariance matrix to describe the uncertainty of the map variables would mean that only specific parts of this covariance matrix need to be extracted and updated, depending on which of the map elements are observed. Implementing the full covariance matrix to describe the map variable uncertainties is suggested as future work.

As a way to keep the size of the optimization problem tractable, Ray-SLAM provides the option to set threshold values on $\sigma_{d,\text{pos},t}^i, \sigma_{d,\varphi,t}^i$. The map offsets in $\hat{\mathbf{d}}_{\text{opt},t}$ always have a lower (or equal) uncertainty than those in $\hat{\mathcal{M}}_{t-1}$ because information was added to the map. If the newly obtained map offset uncertainties have dropped below the user-defined thresholds $s_{\sigma,\text{pos}}, s_{\sigma,\varphi}$ then the respective map variable uncertainty will be set to zero. This has the effect that at the next time step $t+1$ the corresponding map element offset will not become a part of the map optimization vector $\mathbf{d}_{\text{opt},t+1}$.

3-2-3 Linear estimation of the state.

Problem formulation. Recall $p(\mathbf{x}_t \mid \mathbf{x}_{t-1}, \mathbf{z}_t)$ as the linear estimation problem of (3-15b). The linear Kalman filter decouples this estimation problem further into a *measurement update* and a *time update* respectively by

$$p(\mathbf{x}_t \mid \check{\mathbf{x}}_{t-1}, \mathbf{z}_t) \sim \mathcal{N}\left(\hat{\mathbf{x}}_t, \hat{P}_t\right), \tag{3-28a}$$

$$p(\mathbf{x}_{t+1} \mid \hat{\mathbf{x}}_t) \sim \mathcal{N}\left(\check{\mathbf{x}}_{t+1}, \check{P}_{t+1}\right) \tag{3-28b}$$

where $\check{\mathbf{x}}_{t+1}$ denotes the one-step-ahead prediction of the time update.

Implementation details. The estimation problems of (3-28) are solved using the linear Kalman filter equations following [23]. The Kalman filter in Ray-SLAM serves two purposes, which are: (1) estimation of the Lidar's velocity to reliably detect stationary conditions and (2) predict the pose at $t+1$ using the time update of the Kalman filter. The pose prediction accuracy is of significant importance for the robustness of the correspondence function $k(\cdot)$ of (3-13), and hence Ray-SLAM's overall robustness.

A less important byproduct is that the pose estimate $\hat{\mathbf{x}}_{\text{pose},t}$ can be more accurate than $\hat{\mathbf{z}}_t$ when the Lidar is stationary by adding the constant velocity assumption. This effect is only small however because of the high process noise Q assigned to the constant velocity model. We now proceed to describe the linear Kalman filter implementation using the dynamics models of (3-4) and the (virtual) observation model of (3-16) to solve the linear state estimation problem.

Measurement Update. The virtual measurement $\hat{\mathbf{z}}_t$ and the associated covariance $P_{\mathbf{z},t}$ from (3-27) are used to update the predicted state $\check{\mathbf{x}}_t$ using

$$K_t = \check{P}_t C^\top (C\check{P}_t C^\top + P_{\mathbf{z},t})^{-1}, \tag{3-29a}$$

$$\hat{\mathbf{x}}_t = \check{\mathbf{x}}_t + K_t(\hat{\mathbf{z}}_t - C\check{\mathbf{x}}_t), \tag{3-29b}$$

$$\hat{P}_t = (\mathcal{I} - K_t C_t)\check{P}_t, \tag{3-29c}$$

where K_t represents the time-varying Kalman gain, $\check{P}_t$ the predicted state covariance and $\hat{P}_t$ the updated state covariance.

Check if the Lidar is stationary. After establishing the state estimate $\hat{\mathbf{x}}_t$ using (3-29), it is now estimated if the Lidar is stationary using

$$b_{\text{stat}} = \left\| \begin{pmatrix} \hat{v}_{t,x} & \hat{v}_{t,y} & \hat{\omega}_t \end{pmatrix} \right\|_2 < s_{\text{vel}} \tag{3-30}$$

where s_{vel} is a threshold value set by the user. If $b_{\text{stat}} = \text{True}$, then the map optimization vector $\mathbf{d}_{\text{opt},t+1}$ of (3-26) is set to a length of $N_{\mathbf{d},\text{opt}} = 0$ at the next time step.

Time Update. The time update simply extrapolates the Lidar's pose at the current time step by using the constant velocity assumption by

$$\check{\mathbf{x}}_{t+1} = A\hat{\mathbf{x}}_t, \tag{3-31a}$$

$$\check{P}_{t+1} = A\hat{P}_t A^\top + s_Q^2 \mathcal{I} \tag{3-31b}$$

where s_Q is a tuning parameter that dictates how much the linear velocity assumption is to be trusted.

3-2-4 Spawning new map elements.

Problem formulation. Recall that one map element is denoted by

$$\mathcal{M}_t^i = (\underbrace{p^i_{\mathcal{M},x} \quad p^i_{\mathcal{M},y} \quad \varphi^i}_{\mathcal{M}^i_{\text{pose}}} \quad \underbrace{d^i_{\text{pos},t} \quad d^i_{\varphi,t}}_{\mathcal{M}^i_{\text{d},t}} \quad \underbrace{\sigma^i_{d,\text{pos},t} \quad \sigma^i_{d,\varphi,t}}_{\mathcal{M}^i_{\sigma,t}}), \tag{3-32}$$

and the map $\mathcal{M}_t$ as a collection of $N_{\mathcal{M}}$ map elements, i.e.

$$\mathcal{M}_t = \left\{ \mathcal{M}_t^1, \mathcal{M}_t^2, \ldots, \mathcal{M}_t^{N_{\mathcal{M}}} \right\}. \tag{3-33}$$

As no prior map of the scene is available, the SLAM problem start with an empty set of map elements $\mathcal{M}_0 = \varnothing$ at $t = 0$. Therefore, we need a map initialization function $m_{\text{init}}(\cdot)$ at $t = 0$ in the form of

$$\hat{\mathcal{M}}_0 \leftarrow m_{\text{init}}(\mathbf{x}_0, \mathcal{Y}_0, \mathcal{A}_0) \tag{3-34}$$

to spawn the initial map elements. Furthermore, an important part of the SLAM problem is that the robot discovers new areas as it traverses the scene. This can be seen in Figure 3-4, where the robot can observe map elements $\mathcal{M}^{1:7}$ at $t = 0$, but not (yet) $\mathcal{M}^8$. At time point t, $\mathcal{M}^8$ becomes visible. It should be noted that while we write $\hat{\mathcal{M}}$ to indicate that the map is being estimated, the 'true' map $\mathcal{M}$ does, strictly speaking, not exist because walls are never perfectly straight, which is suggested by the linear map element. At $t > 0$, we need to add new map elements to the map that do not overlap existing map elements. The map extension function using $m(\cdot)$ at $t > 0$ is therefore in the form of

$$\hat{\mathcal{M}}_t \leftarrow \hat{\mathcal{M}}_t \cup m(\hat{\mathbf{x}}_t, \hat{\mathcal{M}}_t, \mathcal{Y}_t, \mathcal{A}_t). \tag{3-35}$$

Implementation details. We start by establishing an intermediate *single-scan map* $\bar{\mathcal{M}}_t$ from the Lidar scan $\{\mathcal{Y}_t, \mathcal{A}_t\}$, populated with map elements from this scan only. This map $\bar{\mathcal{M}}_t$ is used in both functions $m_{\text{init}}(\cdot)$ and $m(\cdot)$. The following three steps describe the process of extracting the single-scan map $\bar{\mathcal{M}}_t$ from $\{\mathcal{Y}_t, \mathcal{A}_t\}$: (1) determine for each Lidar observation if it lies on a line, (2) extract lines from the scan and (3) establish the map elements based on these lines. The textual explanation that follows goes together with Algorithm 1 and 2.

Extract $\bar{\mathcal{M}}_t$ from $\mathcal{Y}_t$. (1) Determine for each Lidar observation if it lies on a line.

We start by converting a Lidar scan $\{\mathcal{Y}_t, \mathcal{A}_t\}$ from polar coordinates to Cartesian coordinates using basic trigonometry to obtain the point cloud $\mathcal{Q}_t = \{\mathbf{q}_t^1, \mathbf{q}_t^2, \ldots, \mathbf{q}_t^{N_l}\}$ in the Lidar frame L (see Figure 3-5). The points $\mathbf{q}_t^i$ are transformed to the SLAM frame using the transformation T_t^{SL} which is obtained from $\mathbf{x}_{\text{pose},t}$ or $\breve{\mathbf{x}}_{\text{pose},0}$.

The *local cluster* $\breve{\mathcal{Q}}_{t,i}$ is introduced as a set of N_{clust} point coordinates. For a particular point $\mathbf{q}_t^i \in \mathcal{Q}_t$, its local cluster the set of $N_{\text{clust}} = 13$ points, six of each side of $\mathbf{q}_t^i$ by leveraging the fact that the points in $\mathcal{Q}_t$ are ordered in clockwise direction. The cluster size N_{clust} is a tuning parameter[1]. Similar to other works in SLAM such as [52], the *PCA* method is used to analyze the the shape of the local cluster. As we can recall from (2-2), a cluster's covariance tensor $G_{t,i}$ [33] is calculated as

$$G_{t,i} = \frac{1}{N_{\text{clust}}} \sum_{\mathbf{q}_t^j \in \breve{\mathcal{Q}}_{t,i}} \left(\mathbf{q}_t^j - \overline{\mathbf{q}}_t^i\right)\left(\mathbf{q}_t^j - \overline{\mathbf{q}}_t^i\right)^\top, \qquad \overline{\mathbf{q}}_t^i = \frac{1}{N_{\text{clust}}} \sum_{\mathbf{q}_t^j \in \breve{\mathcal{Q}}_{t,i}} \mathbf{q}_t^j, \tag{3-36}$$

where $\bar{\mathbf{q}}_t^i$ denotes the mean coordinate of the cluster. The linear descriptor value $l_{t,i}$ associated with point $\mathbf{q}_t^i$ is defined by the smallest eigenvalue of $G_{t,i}$, and as such $l_{t,i}$ represent the cluster's variance in minimum-variance direction (i.e. the cluster's flatness). This process is carried out for every point $\mathbf{q}_t^i \in \mathcal{Q}_t$. The linear descriptor value $l_{t,i}$ is visualized in Figure 3-9. It can be seen that $l_{t,i}$ increases when the local cluster of points do not lie on a line.

Extract $\bar{\mathcal{M}}_t$ from $\mathcal{Y}_t$. (2) Extract lines from the scan. The *linear cluster* $\bar{\mathcal{Q}}_{t,j}$ is now introduced, which is a set of at least N_{clust} points. This collection of points is considered to

[1] Tuning of N_{clust} was done empirically. Large N_{clust} results in less false positives, while small N_{clust} increases the algorithm's ability to place map elements far away (>5 m) where observations are more spread out.

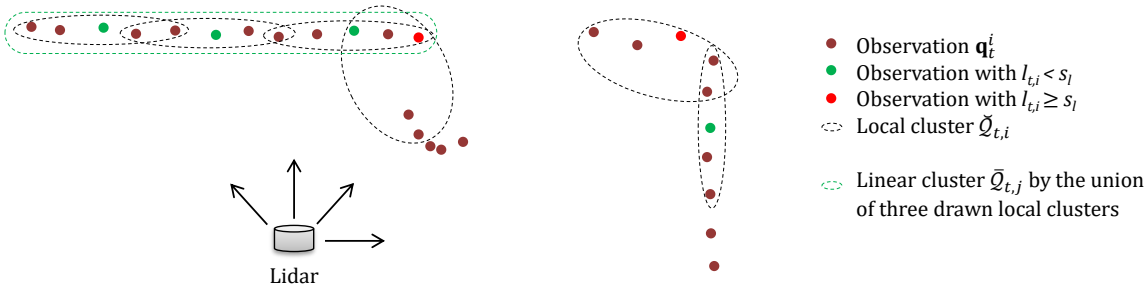


Figure 3-9: Visual representation of the linear descriptor value $l_{t,i}$, which can be intuitively interpreted as the width of the smallest oval shape that fits around the local cluster of adjacent points. A cluster size of $N_{\text{clust}} = 5$ is used in this example. The example Lidar scan shows two walls meeting in a corner point, with the north-sided wall being partly occluded.

lie on a line, and we therefore assume that these points represent a wall in the scene. Starting with $l_{t,1}$, we look for observed points $\mathbf{q}_t^i$ with an associated $l_{t,i} < s_l$, where s_l is introduced as a tuning variable. When a point $\mathbf{q}_t^i$ is found that meets this criterion, the first linear cluster $\bar{\mathcal{Q}}_{t,j}$ with index $j = 1$ is initialized by adding N_{clust} points to it by $\bar{\mathcal{Q}}_{t,j} \leftarrow \breve{\mathcal{Q}}_{t,i}$. Adjacent points in the local clusters $\{\breve{\mathcal{Q}}_{t,i+1}, \breve{\mathcal{Q}}_{t,i+2}, ...\}$ are added to the set $\bar{\mathcal{Q}}_{t,j}$ until $l_{t,i} > s_l$. This process is repeated to find all linear clusters $j = 1, 2, ...$ in one scan $\mathcal{Q}_t$. Figure 3-9 shows an example of a linear cluster.

For each cluster, a line is then fitted through the set of points using the total least squares method of [53]. This results in an unbounded line, parameterized by its normal direction $\phi_{\text{lin},j}$ and orthogonal distance to the origin $r_{\text{lin},j}$. The outer two points in $\bar{\mathcal{Q}}_{t,j}$ are projected onto this unbounded line, resulting in two points $\{\mathbf{p}_{\text{s},j}, \mathbf{p}_{\text{e},j}\}$ that describe the line segment that fits through this linear cluster.

Extract $\bar{\mathcal{M}}_t$ from $\mathcal{Y}_t$. (3) Establish the map elements of $\bar{\mathcal{M}}_t$.

The final step of the map element extraction is to create line segments in the format of (3-32). Zero, one or two map elements are placed on each line based on the line's length (Algorithm 2 only describes the process of placing one map element for brevity). Here, an important aspect is that Ray-SLAM places the map elements with a margin of $0.25s_r$ towards the end of the line, which can only be done if the length of the line $\|\mathbf{p}_{\text{s},j} - \mathbf{p}_{\text{e},j}\|_2 > 2.5s_r$. Recall that s_r is the map element radius (see Figure 3-3). This prevents association of the adjacent non-linear structure to this respective line in correspondence function $k(\cdot)$ due to a slightly incorrect initial guess of the Lidar's pose. Contrary to existing works, iteratively re-establishing the correspondences at a single time point is therefore not necessary.

The map offsets $d^i_{\text{pos},t}, d^i_{\varphi,t} \in \bar{\mathcal{M}}_{\text{d},t}$ are initialized at zero and their uncertainty parameters $\sigma^i_{d,\text{pos},t}, \sigma^i_{d,\varphi,t} \in \bar{\mathcal{M}}_{\sigma,t}$ are initialized at 1 (zero would fix the map offsets permanently). As soon as the map offsets are optimized, their uncertainty parameters will follow naturally from the solution of the optimization problem that is discussed later in this chapter (§3-2-2).

Construct the initial map $\hat{\mathcal{M}}_0$.

Initializing the Ray-SLAM algorithm starts with constructing the initial map $\hat{\mathcal{M}}_0$ using $m_{\text{init}}(\cdot)$, a process which requires special attention. As studied by [54], the SLAM problem is not per se a well-posed problem due to the lack of absolute information that places the

Algorithm 1: Extract map $\bar{\mathcal{M}}_t$

Input : Lidar pose $\hat{\mathbf{x}}_{\text{pose},t}$, Lidar scan $\mathcal{Y}_t, \mathcal{A}_t$, tuning variables $s_l, s_r, N_{\text{clust}}$
Output: Intermediate map $\bar{\mathcal{M}}_t$

1. Convert $\mathcal{Y}_t, \mathcal{A}_t$ to Cartesian point coordinates $\mathcal{Q}_t$ in the Lidar frame.
2. Transform all $\mathbf{q}_t^i \in \mathcal{Q}_t$ from the Lidar frame to the SLAM frame using $\mathbf{q}_t^{S,i} = T_t^{SL}\mathbf{q}_t^i$, where T_t^{SL} is the transformation matrix of $\hat{\mathbf{x}}_{\text{pose},t}$ using (A-4). The SLAM frame S is dropped onwards for notational convenience.
3. Calculate the linear descriptor value $l_{t,i}$ for all N_l points $\mathbf{q}_t^i \in \mathcal{Q}_t$:

   for $i = 1, 2, \ldots, N_l$ **do**

   (a) Collect the local cluster $\breve{\mathcal{Q}}_{t,i}$ around $\mathbf{q}_t^i$ of size N_{clust}.

   (b) Calculate the local cluster covariance tensor

   $$G_{t,i} = \frac{1}{N_{\text{clust}}} \sum_{\boldsymbol{q}_t^j \in \breve{\mathcal{Q}}_{t,i}} \left(\mathbf{q}_t^j - \overline{\mathbf{q}}_t^i\right)\left(\mathbf{q}_t^j - \overline{\mathbf{q}}_t^i\right)^\top, \qquad \overline{\mathbf{q}}_t^i = \frac{1}{N_{\text{clust}}} \sum_{\mathbf{q}_t^j \in \breve{\mathcal{Q}}_{t,i}} \mathbf{q}_t^j. \quad (3\text{-}37)$$

   (c) Set $l_{t,i}$ = smallest eigenvalue of $G_{t,i}$.

   end for
4. Set $i = 1$ (measurement index), $j = 1$ (linear cluster index).
5. Extract linear clusters of points $\bar{\mathcal{Q}}_{t,j}$:

   while $i \leq N_l$ **do**

   (a) **if** $l_{t,i} > s_l$ **then** set $i \leftarrow i + 1$, **continue**

   (b) Add points to the linear cluster $\bar{\mathcal{Q}}_{t,j}$:

   if $\bar{\mathcal{Q}}_{t,j} = \varnothing$ **then**

   Add all the points in $\breve{\mathcal{Q}}_{t,i}$ using $\bar{\mathcal{Q}}_{t,j} \leftarrow \breve{\mathcal{Q}}_{t,i}^{1:N_{\text{clust}}}$.

   else

   Add only the last point in $\breve{\mathcal{Q}}_{t,i}$ using $\bar{\mathcal{Q}}_{t,j} \leftarrow \bar{\mathcal{Q}}_{t,j} \cup \breve{\mathcal{Q}}_{t,i}^{N_{\text{clust}}}$.

   if $l_{t,i+1} > s_l$ **then** set $j \leftarrow j + 1$.

   end if

   (c) Set $i \leftarrow i + 1$.

   end while
6. Set $N_{\text{lin}} = j - 1$ (the amount of linear clusters), $k = 1$ (map element index).

Algorithm 2: Extract map $\bar{\mathcal{M}}_t$ (continued)

Input : Lidar pose $\hat{\mathbf{x}}_{\text{pose},t}$, Lidar scan $\mathcal{Y}_t, \mathcal{A}_t$, tuning variables $s_l, s_r, N_{\text{clust}}$
Output: Intermediate map $\bar{\mathcal{M}}_t$

7. Extract line segments from the linear clusters:

   for $j = 1, 2, \ldots, N_{\text{lin}}$ **do**

   (a) $\{r_{\text{lin},j}, \phi_{\text{lin},j}\} \leftarrow$ Fit the unbounded total least squares line [53] to $\bar{\mathcal{Q}}_{t,j}$.

   (b) $\{\mathbf{p}_{\text{s},j}, \mathbf{p}_{\text{e},j}\} \leftarrow$ Project the outer two points in $\bar{\mathcal{Q}}_{t,j}$ onto the line of $\{r_{\text{lin},j}, \phi_{\text{lin},j}\}$.

   end for

8. Establish the map $\bar{\mathcal{M}}_t$:

   for $j = 1, 2, \ldots, N_{\text{lin}}$ **do**

   (a) Calculate the line segment's center by $\mathbf{p}_{\text{c},j} = 0.5(\mathbf{p}_{\text{s},j} + \mathbf{p}_{\text{e},j})$.

   (b) Add a map element to $\bar{\mathcal{M}}_t$ only if the line is long enough:
   if $\|\mathbf{p}_{\text{s},j} - \mathbf{p}_{\text{e},j}\|_2 > 2.5 s_r$ **then**

$$\bar{\mathcal{M}}_t^k = (\underbrace{p_{\text{c},j,x} \quad p_{\text{c},j,y} \quad \phi_{\text{lin},j}}_{\bar{\mathcal{M}}_{\text{pose}}^k} \quad \underbrace{0 \quad 0}_{\bar{\mathcal{M}}_{\text{d,t}}^k} \quad \underbrace{1 \quad 1}_{\bar{\mathcal{M}}_{\sigma,t}^k}), \tag{3-38}$$

   Set $k \leftarrow k + 1$.
   end if

   end for

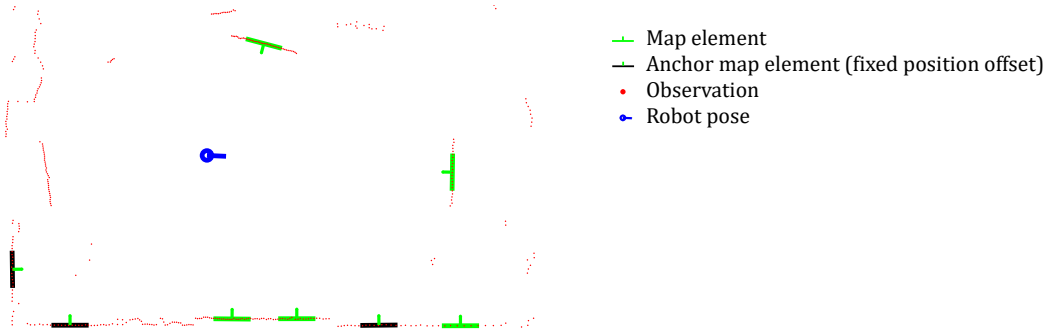


Figure 3-10: The selection of three anchor map elements is shown. These elements have a position offset that is fixed to zero. As a result, the SLAM problem becomes anchored in the global reference frame.

robot and map in a global reference frame. As argued by [55], at least two a-priori known reference point observations are needed for the planar SLAM system to be *locally weakly observable*. These reference points eliminate the three degrees of freedom of the system where the robot pose and map move together.

In order to improve the observability of the SLAM problem, Ray-SLAM implements a process of defining a set of *anchor map elements* during initialization. The user sets three map element position offsets $\sigma^i_{d,\text{pos},t} = 0$ as a flag to exclude these map offsets from an optimization procedure, and as a consequence the corresponding map position offset is fixed to $d^i_{\text{pos},t} = 0$. As a result, the local reference frame in which the Lidar perceives its environment becomes anchored within the global reference frame or world frame W. Defining the anchor map elements is implemented as a manual step, shown in Figure 3-10. In order to properly constrain the SLAM system, the three map elements should be chosen such that the elements are not all parallel to constrain translation and their normals do not line up in a single point to constrain rotation.

In summary, the function $m_{\text{init}}(\cdot)$ carries out the following steps to obtain $\hat{\mathcal{M}}_0$: (1) extract $\bar{\mathcal{M}}_0$ from $\mathcal{Y}_0$ and (2) fix the position offset of three anchor map elements.

Append the map $\hat{\mathcal{M}}_t$.

We introduce $\mathbf{x}_{\mathcal{M}} \in \mathbb{R}^2$ as the last Lidar location where map elements have been added. In order to save computational resources, appending the map is only carried out if the Lidar has moved more than the threshold distance $s_{\mathcal{M}}$ with respect to the last map update location $\mathbf{x}_{\mathcal{M}} \in \mathbb{R}^2$ by checking the condition

$$\|\hat{\mathbf{x}}_{\text{pos},t} - \mathbf{x}_{\mathcal{M}}\|_2 > s_{\mathcal{M}}. \tag{3-39}$$

If this condition holds, we update $\mathbf{x}_{\mathcal{M}}$ using $\mathbf{x}_{\mathcal{M}} \leftarrow \mathbf{x}_{\text{pos},t}$ and proceed with appending the map. We start by extracting $\bar{\mathcal{M}}_t$ from $\mathcal{Y}_t$. The map $\bar{\mathcal{M}}_t$ is likely to contain map elements at locations that had already been mapped in $\hat{\mathcal{M}}_t$. Therefore, in order to prevent overlapping and/or duplicate map elements in $\hat{\mathcal{M}}_t$, we remove the map elements in $\bar{\mathcal{M}}_t$ where the condition holds of

$$\left\| (p^i_{\bar{\mathcal{M}},x} \quad p^i_{\bar{\mathcal{M}},y}) - (p^j_{\mathcal{M},x} \quad p^j_{\mathcal{M},y}) \right\|_2 < 2.2 s_r. \tag{3-40}$$

The remainder of $\bar{\mathcal{M}}_t$ now represents new parts of the scene, which is the result of the function $m(\cdot)$. Removing these duplicate/overlapping map elements also prevents mapping the same

area twice after a loop closure. Recall that map is now appended by

$$\hat{\mathcal{M}}_t \leftarrow \hat{\mathcal{M}}_t \cup m(\hat{\mathbf{x}}_t, \hat{\mathcal{M}}_t, \mathcal{Y}_t). \tag{3-41}$$

To provide more context on the map building technique of Ray-SLAM, we now briefly explain other commonly used map building strategies by Lidar-centric SLAM algorithms. The LOAM algorithm executes a mapping step at 1 Hz continuously, where new feature points are added to the map and voxel grid binning is applied such that only one point is kept per voxel. Such a map is considered a *global* map, as the entire map is considered each time ICP-based scan alignment is carried out. LIO-SAM takes a different approach by the use of *key-frames*. Key-frames are defined at Lidar poses after a certain amount of movement was detected, which is set either 1 m or 10° in the work. Only the edge- and planar feature points at key-frames are kept as the map. To increase the computational efficiency, a moving-horizon of the last N key-frames is considered during the ICP-based scan alignment, the collection of these key-frames is considered a *local* map. Key-frames that fall outside the horizon of the local map are kept in the growing factor graph to facilitate loop closures.

3-2-5 Finding correspondences between the Lidar observations and the map elements.

Problem formulation. Recall that in §3-1-4, the correspondence finding function $k(\cdot)$ was introduced as

$$\mathcal{C}_t = k(\mathbf{x}_t, \mathcal{M}_t, \mathcal{Y}_t, \mathcal{A}_t) \tag{3-42}$$

where $\mathcal{C}_t$ contains the subset of Lidar observations $\mathcal{Y}_{\mathcal{C},t} \subseteq \mathcal{Y}_t, \mathcal{A}_{\mathcal{C},t} \subseteq \mathcal{A}_t$ that have detected a map element, and the linking table $\mathcal{N}_{\mathcal{C},t}$ with the respective map element index values:

$$\mathcal{C}_t = \{\mathcal{Y}_{\mathcal{C},t}, \mathcal{A}_{\mathcal{C},t}, \mathcal{N}_{\mathcal{C},t}\}. \tag{3-43}$$

Obtaining accurate correspondences is critical to the performance of the SLAM algorithm because the algorithm will assume onward that the observation actually represents the structure to which a correspondence has been established. If the observation represents something close to this structure instead, an error is introduced that pulls the pose estimate away from its true value.

Implementation details. The first step of establishing correspondences is to identify the map elements that face away from the Lidar's position and exclude these from the process since the face of a wall cannot be observed from behind. This prevents erroneous correspondences with thin walls after the Lidar has traversed around this wall. Then, the following process is executed for each map element that faces towards the Lidar, which is also illustrated in Figure 3-11:

- Angles $\gamma^j_{1,t}, \gamma^j_{2,t}$ are determined, which are the heading angles of the start- and end points of map element $\hat{\mathcal{M}}^j_t$ with respect to the Lidar in the SLAM frame. Only Lidar measurements where the measurement direction $\theta_t + \alpha^i_t$ falls in the interval of $\gamma^j_{1,t} < \theta_t + \alpha^i_t < \gamma^j_{2,t}$ could detect this map element, these measurements are thus selected as a correspondence candidates.

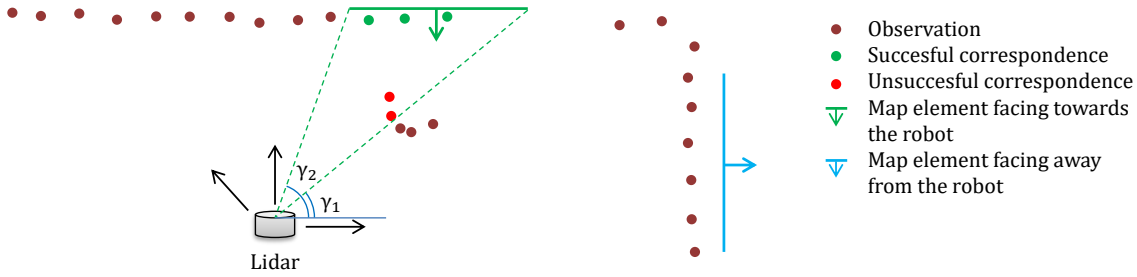


Figure 3-11: Visualization of the process to establish correspondences. Map elements facing away from the Lidar are excluded. The angles γ_1, γ_2 are used to pre-select correspondence candidates. The correspondences in green are accepted with $|\tilde{y} - y| < s_{\text{corr}}$.

- For each correspondence candidate, the ray casting measurement prediction $\tilde{y}_t^i$ is subtracted from the actual measured distance y_t^i. Only if the magnitude of $\tilde{y}_t^i - y_t^i$ does not exceed the threshold s_{corr}, the correspondence is accepted.

Furthermore, not all map elements may be detected by the Lidar due to occlusion. The boolean vector $\mathbf{b}_{\mathcal{M},t} \in \mathbb{R}^{N_\mathcal{M}}$ is collected that indicates for each of the $N_\mathcal{M}$ map elements if at least one correspondence has been established. Using $\mathbf{b}_{\mathcal{M},t}$, we can eliminate undetected map elements from the pose- and map estimation process.

It is interesting to note how this process differs from correspondence finding methods in other state-of-the-art Lidar SLAM algorithms, where correspondence finding is typically integrated into an ICP process. As explained in §2-2-3, ICP relies on the Euclidean distance between points in the map and the observed point cloud respectively. All points in the observed point cloud need to find their nearest neighbor in the map, and since both sets of points can be large, this problem can be expensive to solve. Ray-SLAM reduces the required computational efforts for correspondence finding by (1) reducing the density of the map and (2) pre-selection of correspondence candidates using the heading angles γ_1, γ_2 of the map elements to reduce the search space. It is hypothesized that this technique will become increasingly valuable in a potential 3D generalization of Ray-SLAM.

3-2-6 Summary of the Ray-SLAM algorithm

So far in this chapter, the state space model of the SLAM system has been established in combination with the map representation. Detailed descriptions were given about the sub-problems of Ray-SLAM's approach to 2D Lidar SLAM, including the implementation of their solutions. We now summarize Ray-SLAM's assumptions and novelties that were introduced in this chapter, and provide an algorithmic overview.

Assumptions. In summary, the Ray-SLAM algorithm assumes that: (1) The Lidar range measurements are corrupted by zero mean Gaussian noise signals. (2) Walls are present in the scene. (3) The Lidar motion is constrained to the horizontal 2D plane and does not exhibit roll, pitch and heave motion. (4) Repetitive static conditions are present in the robot motion. (5) The state description of the SLAM system possesses the Markov property.

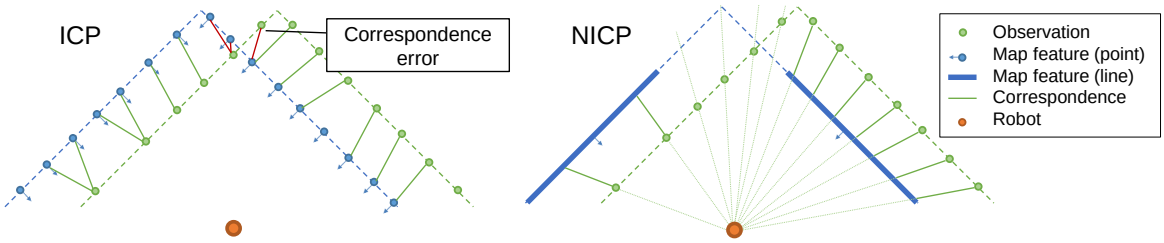


Figure 3-12: Left. A visualization of the ICP algorithm. It can be seen that 3 incorrect correspondences (red) are made due to an incorrect initial pose guess (see §2-2-3). **Right**. The proposed NICP observation-to-map alignment procedure considers the Lidar's perspective using ray casting and aligns the Lidar observations with sparse linear map features using (3-19). These features are placed away from edges and irregularities in order to avoid incorrect correspondences. No iterations are required in this alignment process.

Novelties. The outlined approach to Lidar SLAM differs with respect to the state-of-the-art on various levels. Existing works in Lidar SLAM often build the map of dense (feature) points [18, 56] in an attempt to model the scene in its entirety. The map is appended when new areas are explored, and no attempt is made to improve the existing map features by accumulating statistical information about these features. Some form of the ICP method [39] is typically used to align new Lidar observations with this map for localization. The ICP algorithm is required to iteratively re-establish the correspondences because small misalignments in the initial pose guess lead to a large amount of false positive and negative correspondences. Contrary to this approach, this thesis proposes: **(1)** Ray-SLAM adds additional knowledge about the scene's structure to the SLAM problem by using a sparse map representation where walls are used as primary reference. Since walls can easily be modeled in 2D by line segments, a poor map quality is avoided at parts of the scene with complex geometry. **(2)** Since the map elements are placed away from non-flat objects in the scene (see Figure 3-2), correspondence errors due to small misalignments in the initial pose guess can largely be avoided. Hence, the steps of correspondence finding (3-13) and pose estimation (3-19) need to be carried out only once, contrary to the ICP algorithm that is often used in Lidar SLAM. This process is introduced therefore as Non-Iterative Closest Point (NICP). Aligning the observations to the map using ICP and NICP is compared in Figure 3-12. **(3)** The map accuracy is improved over time in a probabilistic fashion.

Algorithm summary. The Ray-SLAM algorithm is summarized in Algorithm 3.

Algorithm 3: Ray-SLAM

Input : Initial pose $\check{\mathbf{x}}_{\text{pose},0} \in SE(2)$, measurements $\mathcal{Y}_{0:T}, \mathcal{A}_{0:T}$ and all tuning variables s
Output: Trajectory estimate $\hat{\mathbf{x}}_{1:T}$ and map $\hat{\mathcal{M}}_T$

1. Set $\mathbf{x}_0 = (\check{\mathbf{x}}_{\text{pose},0}^\top \quad 0 \quad 0 \quad 0)^\top$, $\check{\mathbf{x}}_1 = \mathbf{x}_0$, $\hat{\mathcal{M}}_0 = \varnothing$ and $b_{\text{stat}} = \text{True}$.
2. Extract the initial map using $\hat{\mathcal{M}}_0 \leftarrow m_{\text{init}}(\mathbf{x}_0, \mathcal{Y}_0, \mathcal{A}_0)$ as described in §3-2-4. Three map elements are selected by the user as anchor map elements by setting $\sigma^i_{d,\text{pos},0} = 0$, resulting in initial map $\hat{\mathcal{M}}_0$.
3. **for** $t = 1, 2, \ldots, T$ **do**
   (a) Establish correspondences between the observations and the map elements using $\mathcal{C}_t = k(\check{\mathbf{x}}_t, \mathcal{M}_{t-1}, \mathcal{Y}_t, \mathcal{A}_t)$ as described in §3-2-5. Obtain map element observation boolean vector $\mathbf{b}_{\mathcal{M},t}$, which indicates if the map elements have been observed.
   (b) **if** $b_{\text{stat}} = \text{False}$ **then** set $\mathbf{b}_{\mathcal{M},t} \leftarrow \mathbf{0}$ such that the map is not optimized when motion is detected.
   (c) Define the map optimization vector $\mathbf{d}_{\text{opt},t} \subseteq (d^1_{\text{pos}} \quad d^1_\varphi \quad \ldots \quad d^{N_\mathcal{M}}_{\text{pos}} \quad d^{N_\mathcal{M}}_\varphi)^\top$ using $\mathbf{b}_{\mathcal{M},t}$ as described in §3-2-2. Solve the optimization problem

$$\begin{aligned}\left\{\hat{\mathbf{z}}_t, \hat{\mathbf{d}}_{\text{opt},t}\right\} = \underset{\mathbf{z}_t, \mathbf{d}_{\text{opt},t}}{\arg\min} \sum_{d^i_t \in \mathbf{d}_{\text{opt},t}} (\sigma^i_{\text{d},t-1})^{-2}(d^i_t - \hat{d}^i_{t-1})^2 \\ + \sigma_r^{-2} \sum_{i=1}^{N_\mathcal{C}} \left(\cos(\beta^i_t)^{s_\beta}(y^i_{\mathcal{C},t} - \tilde{y}^i_{\mathcal{C},t})\right)^2\end{aligned} \tag{3-44}$$

   where the measurement prediction $\tilde{y}^i_{\mathcal{C},t}$ depends on $\mathbf{z}_t, \alpha^i_t, \hat{\mathcal{M}}_t$ and is obtained by (3-10). Extract $P_{\mathbf{z},t}$ and $\sigma^i_{\text{d},t} \in \mathcal{M}_{\sigma,t}$ at the local minimum of (3-44) and update the map $\hat{\mathcal{M}}_{t-1}$ to $\hat{\mathcal{M}}_t$ as described in §3-2-2.
   (d) Measurement update

$$K_t = \check{P}_t C^\top (C\check{P}_t C^\top + P_{\mathbf{z},t})^{-1}, \tag{3-45a}$$
$$\hat{\mathbf{x}}_t = \check{\mathbf{x}}_t + K_t(\hat{\mathbf{z}}_t - C\check{\mathbf{x}}_t), \tag{3-45b}$$
$$\hat{P}_t = (\mathcal{I} - K_t C_t)\check{P}_t. \tag{3-45c}$$

   (e) Set $b_{\text{stat}} \leftarrow (\|(\hat{v}_{t,x} \quad \hat{v}_{t,y} \quad \hat{\omega}_t)\|_2 < s_{\text{vel}})$.
   (f) Time update

$$\check{\mathbf{x}}_{t+1} = A\hat{\mathbf{x}}_t, \tag{3-46a}$$
$$\check{P}_{t+1} = A\hat{P}_t A^\top + s_Q^2 \mathcal{I}. \tag{3-46b}$$

   (g) **if** $b_{\text{stat}} = \text{True}$ **then** append the map using $\hat{\mathcal{M}}_t \leftarrow \hat{\mathcal{M}}_t \cup m(\hat{\mathbf{x}}_t, \hat{\mathcal{M}}_t, \mathcal{Y}_t), \mathcal{A}_t)$ as described in §3-2-4 to obtain the appended map $\hat{\mathcal{M}}_t$.

end for

Chapter 4

Results

This chapter describes the real-world experiments that have been conducted to evaluate the performance of the Ray-SLAM algorithm. The measurement setup was deployed in multiple indoor scenes with various shapes and sizes in order to get a realistic perspective on the algorithm's strengths and limitations. Special attention is paid to the accuracy, robustness and influence of various tuning parameters. The algorithm's performance is furthermore compared to two state-of-the-art open source SLAM algorithms.

4-1 Measurement setup

All experiments have been carried out with the Ouster OS0-64 Lidar device. It has a specified range accuracy of ±3 cm for diffusely reflecting (matte) surfaces and ±10 cm for reflective surfaces. The measurement precision ranges between $\sigma = 1.0$ cm and $\sigma = 5.0$ cm depending on the measurement distance and surface reflectivity. The Lidar has a vertical resolution of 64 beams spread equally over 90°. Because of the even number of beams, the two central beams have an angle of $-0.7°, +0.7°$ with respect to the horizontal axis, of which the upper one was used. The Lidar rotates internally at 10 Hz with a horizontal resolution of 2048 measurements per rotation. The Lidar has an integrated IMU of the model InvenSense ICM-20948 with a sampling rate of 100 Hz. The IMU clock is synchronized with the Lidar clock up to 1 ms. More detailed specifications can be found in Appendix B.

The measurement setup consists of the Lidar device mounted onto a wheeled tripod to ensure that the device moves within the horizontal plane, see Figure 4-1. A water level meter was used to ensure that the Lidar was leveled.

For use with a motion capture system, five reflective markers were attached for tracking. A rigid body is attached to the five markers in order to track the device (Figure 4-1, right). The x, y coordinates of the top reflective marker is combined with the heading θ of the rigid body in order to obtain the Lidar's pose in $SE(2)$. Since the top marker is above the optical center, the extrinsic calibration between the Lidar and motion capture frame consist only of a

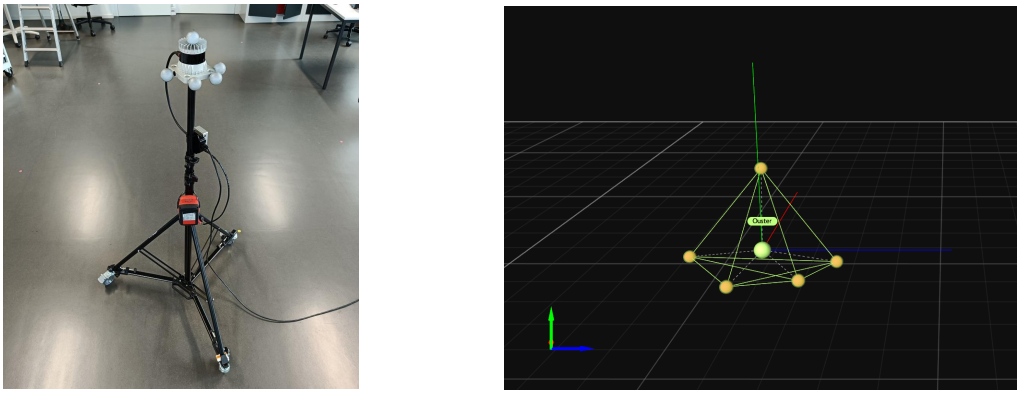


Figure 4-1: Left. The Ouster OS0 Lidar device is mounted rigidly onto a wheeled tripod. Five reflective markers are attached for tracking. **Right.** The reflective markers are detected by the motion capturing system. A rigid body is attached to the five markers.

Figure 4-2: The setup with a reflective prism mounted on the top of the Lidar. This prism is detected by the Hilti PLT-300 Total Station (left) and is able to record coordinates of stationary points as a ground-truth source.

heading difference. The OptiTrack motion capture system is a reliable source of ground-truth information with sub-mm accuracy.

At scenes other than the motion capture lab, the PLT-300 Total Station was used as a source of ground-truth. The reflective prism is mounted on top of the Lidar as shown in Figure 4-2. The prism is carefully aligned centrally above the Lidar so that there is no extrinsic calibration required between the two in a 2D simplification. The PLT-300 has sub-3mm accuracy and measures only the position of the reflective prism in stationary conditions.

4-2 Datasets

Six data sets with ground truth information have been recorded at five different locations at the Hilti campus located in Schaan, Liechtenstein. The locations were selected to represent a wide variety of room shapes and sizes in order to get an understanding of how Ray-SLAM

performs under various real-world circumstances. The large rooms in particular are scenarios in which the Jaibot could be deployed for operation. As discussed in Chapter 1, utilizing Ray-SLAM on the Jaibot is one of the primary use cases.

Lidar and windows. Most buildings contain a significant amount of windows in them, and windows were not avoided when seeking suitable locations to test Ray-SLAM in real-world scenarios. Windows are typically not detected directly, but act more like a mirror (depending on how clean the window is). Lidar SLAM algorithms, Ray-SLAM included, consequently create a mirrored version of the room in their map behind the window. This is typically not a problem because the mirrored features behave consistently as the Lidar is moved. Errors can be introduced however when the windows are not coplanar, causing the mirrored feature to behave inconsistent across different windows. Addressing this potential error source is considered outside the scope of this thesis. Section 5-2 will go into more detail about possible ways to eliminate this error source.

Motion capture lab.

The motion capture lab at the Innovation Center building on the Hilti campus is a relatively small room of approximately 60 m^2. The room has a rectangular shape and no obstacles in the middle of the room that can occlude parts of this room, see Figure 4-3. The measurement setup stays within this room during data capture, and as such no new areas are discovered. The dataset *Motion capture lab* was recorded at this scene. Challenging aspects are several reflective surfaces (whiteboards) that are used as reference and the large amount of windows. The available OptiTrack motion capture system provides a reliable source of ground-truth information.

Figure 4-3: Motion Capture Lab.

VGN floor 1.

The VGN building is a large office building on the Hilti campus. It was emptied at the time of writing since it was put out of use. The large office spaces make an ideal environment to carry out testing, as a building construction site is often relatively empty as well. The first floor office section is roughly 350 m^2, see Figure 4-4. A part of this office section was found cluttered with a large amount of scrapped office materials (left in the image). Two data sets were captured, *VGN floor 1(a)* contains a trajectory in the non-cluttered part and *VGN floor 1(b)* contains a trajectory through the clutter. A large amount of windows is present.

Figure 4-4: VGN office floor 1.

VGN floor 2.

The office at the 2^{nd} floor of the VGN building is similar in shape of the 1^{st} floor but is smaller (220 m^2), more open and less cluttered as can be seen in Figure 4-5. One data set was captured with a trajectory that covers most of the scene.

Figure 4-5: VGN floor 2.

VGN central.

The central part of the VGN building contains a meeting room that is connected to several small neighboring rooms via door openings, see Figure 4-6. Some clutter is present next to the door openings, providing a challenging scenario for Ray-SLAM that will need to connect the rooms at both side of the door openings. The recorded data set contains a trajectory that connects four different rooms.

Figure 4-6: VGN building central.

VGN top.
The top floor of the VGN building is a large room of roughly 250 m^2. The majority of the room periphery consist of windows as can be seen in Figure 4-7. A significant section of the room periphery is very irregular due to three elevator entrances and four doors without large planar surfaces in between. The recorded data set covers a trajectory though the room and exits the room into an adjacent hallway at the right of the image.

Figure 4-7: VGN building top.

The datasets are summarized in Table 4-1.

Table 4-1: Overview of datasets.

Scene	Room size	Traj. length	Waypoints
Motion capture lab	60 m^2	21 m	8
VGN floor 1(a)	350 m^2	65 m	15
VGN floor 1(b)	350 m^2	45 m	13
VGN floor 2	220 m^2	85 m	20
VGN central	120 m^2	39 m	11
VGN top	250 m^2	52 m	15

4-3 Experiment results and robustness analysis

This section discusses the map and trajectory reconstruction processes of Ray-SLAM using the datasets as discussed in §4-2. Failures and large errors ($>$ 10 cm) are investigated in detail in order to assess the robustness of the Ray-SLAM algorithm.

4-3-1 Parameter tuning

An overview of the tuning parameters that have been used during the experiments is given in Table 4-2. The author had access to the ground-truth information during the tuning process, and no strict separation between tuning- and evaluation datasets was applied. Therefore a risk exist that the tuning parameters are overfitted to the datasets. The tuning process employed by the author was mainly focused on prevention of failures cases, rather than minimizing the localization error statistics. It was found that in particular the parameters s_r, N_{clust}, s_l and s_{corr} required careful tuning. The tuning was carried out for the most part on dataset *VGN*

Table 4-2: Overview of tuning parameters.

Parameter	Unit	Value	Description
σ_y	meter	0.025	Lidar range noise standard deviation
s_{vel}	m/s	0.05	Velocity threshold
s_r	meter	0.18-0.35	Map element radius
s_β		1.0	Beam angle discount factor
N_{clust}		13	Map element extraction cluster size
s_l		0.01	Map element extraction linearity threshold
s_{corr}	meter	0.25	Correspondence distance threshold
s_Q		0.025	Process noise standard deviation
$s_{\sigma,\text{pos}}$		0.0004	Map element position offset uncertainty threshold
$s_{\sigma,\varphi}$		0.0015	Map element heading offset uncertainty threshold
$s_{\mathcal{M}}$	meter	0.5	Map extension distance threshold

floor 1(a), which generalized to most other datasets but resulted in failure on datasets *VGN central* and *VGN top*. Changing the map element radius from $s_r = 0.35$ to $s_r = 0.18$ allowed successful processing of *VGN top*. After that, only minor tuning was carried out such that both small ($s_r = 0.18$) and large ($s_r = 0.35$) map elements worked robustly. This chapter considers the results of both the small and large map elements, and the other parameters were kept constant.

The Lidar's initial position always starts at $(p_{x,0} \quad p_{y,0})^\top = \mathbf{0}$. The initial heading θ_0 was chosen such that the map aligns with the plot axes. The beam angle discount factor s_β was set to 1.0 by default such that the error metric represents the point-to-line distance. Tuning of s_β is investigated in detail in §4-5.

4-3-2 Analysis of the experiment results

Dataset VGN floor 1(a) and VGN floor 1(b). In Figure 4-8 the reconstruction of dataset *VGN floor 1(a)* and *VGN floor 1(b)* are shown using the large $r = 0.35$ m map elements. At three locations it can be seen that map elements have been placed on windows. Ray-SLAM succesfully reconstructed the map and trajectory of dataset *VGN floor 1(a)* for both map element sizes. In dataset *VGN floor 1(b)*, only 11 out of 13 waypoints were successfully estimated as the algorithm failed just before reaching the 12^{th} waypoint. The waypoint estimate where the Lidar was stopped can often be recognized by a sharp corner in the continuous trajectory. The ground truth positions were transformed to the SLAM frame (see §4-4-1) to visualize that the estimate typically aligns closely with the ground truth. Due to the scale of the plots, the mm-level errors are often not visible.

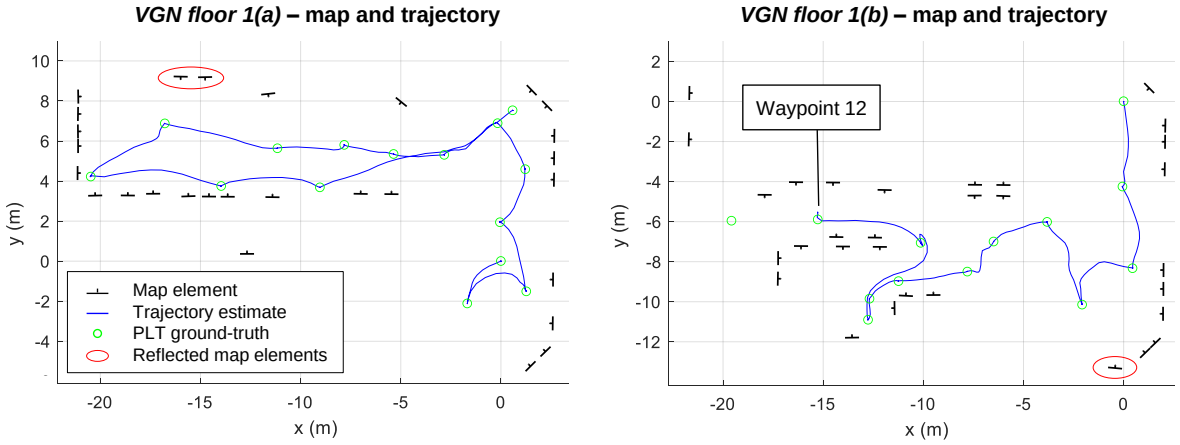


Figure 4-8: The estimated map and trajectory of datasets *VGN floor 1(a)* and *VGN floor 1(b)*

The cause for failure is shown in Figure 4-9, highlighting the SLAM process just before failure. The green circle illustrates the last location where the map was updated. At this location, two horizontally oriented walls at the left were not yet visible to the Lidar, and Ray-SLAM was unable to place map elements there. A little later, the algorithm runs out of horizontally oriented correspondences and fails.

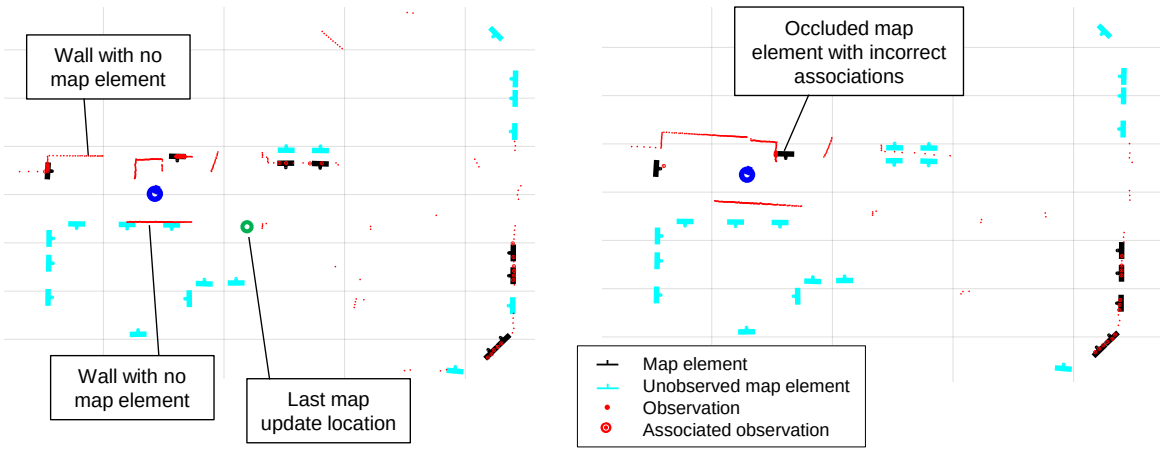


Figure 4-9: Left. Just before failure at waypoint 12 of dataset *VGN floor 1(b)* the Lidar is running out of local horizontally oriented map elements. **Right.** The algorithm starts to diverge due to incorrect correspondences and lack of map elements.

Failure could probably have been prevented in this scenario if the measurement setup had stopped more often. An identical failure occured when using the small map elements of $r = 0.18$ m.

Dataset VGN floor 2 and VGN Top. In Figure 4-10, the reconstruction of dataset *VGN floor 2* and *VGN top* are shown. Ray-SLAM placed several map elements on walls that were reflected through the windows, these are marked in red. In dataset *VGN floor 2* all waypoints were successfully estimated with both map element sizes. Processing of dataset *VGN top* only

succeeded using the smaller $r = 0.18$ m map elements because of the lack of large walls in the majority of the scene as seen in Figure 4-7.

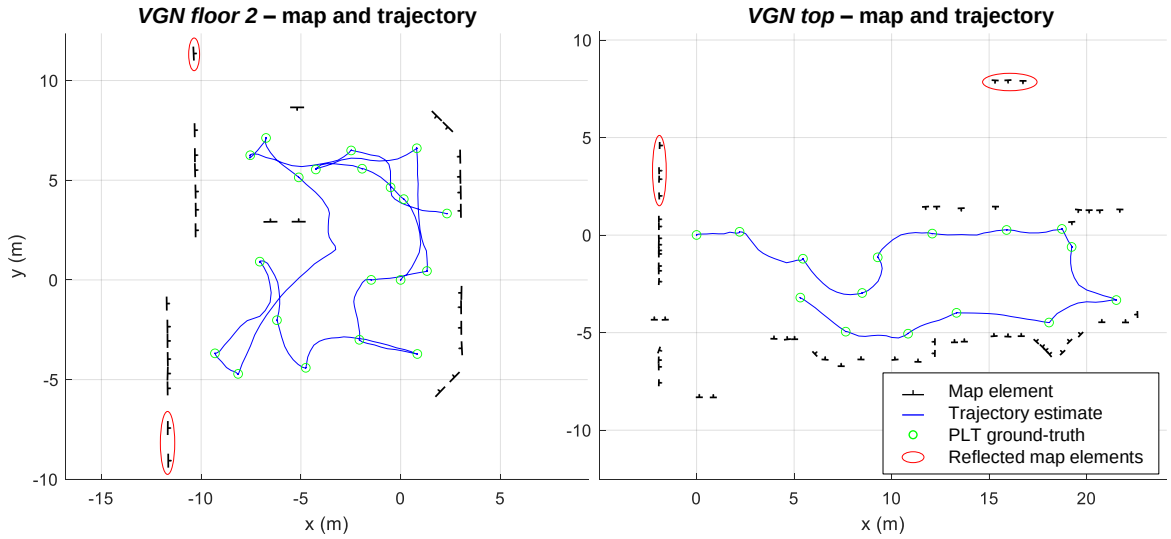


Figure 4-10: The estimated map and trajectory of datasets *VGN floor 2* and *VGN top*.

Dataset Motion capture lab and VGN central. In Figure 4-11 the reconstruction of datasets *Motion capture lab* and *VGN central* are shown.

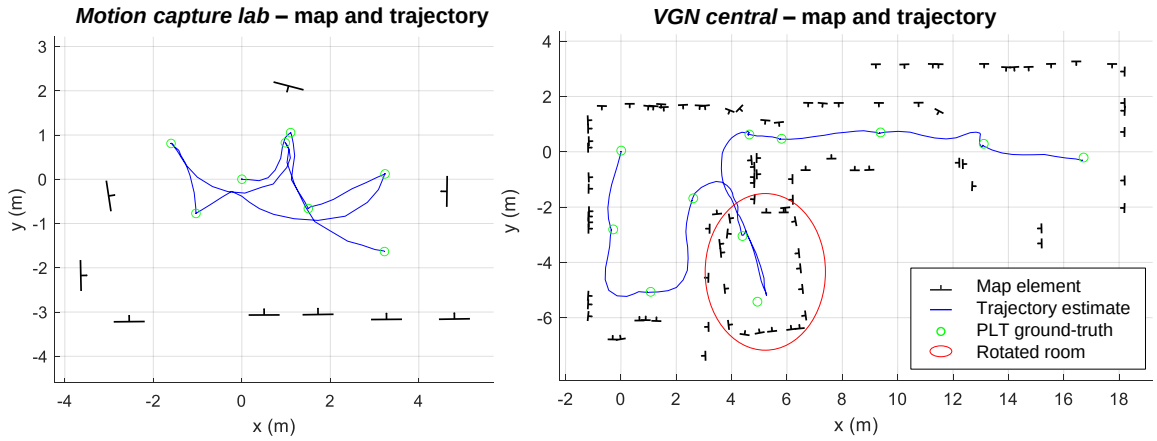


Figure 4-11: The estimated map and trajectory of datasets *Motion capture lab* and *VGN central*.

Ray-SLAM successfully reconstructed a map and trajectory with the *Motion capture lab* dataset for both map element sizes. It can be seen that Ray-SLAM encountered problems in *VGN central* with small map elements when entering the small room that is marked in red. The trajectory estimate in this room contain significant errors of almost 40 cm. When using large map elements, the small room was reconstructed successfully but the algorithm failed later when successively entering the hallway. Both cases are discussed now in detail.

The *VGN central* dataset with small map elements was problematic due to *correspondence*

errors. A correspondence error occurs when a map element that represents a wall is linked to an observation of something else than this wall. This association introduces an error in (3-27) that pulls the pose estimate away from its true value because the algorithm now assumes that the observation actually represents the wall. Figure 4-12 shows how the correspondence error originated in detail. The three legs of the PLT-300 tripod and the person standing in front of the wall (eleven observations in total) are wrongly associated with the map elements representing this wall. As a result, the newly discovered room in the bottom-right corner was rotated counter clockwise.

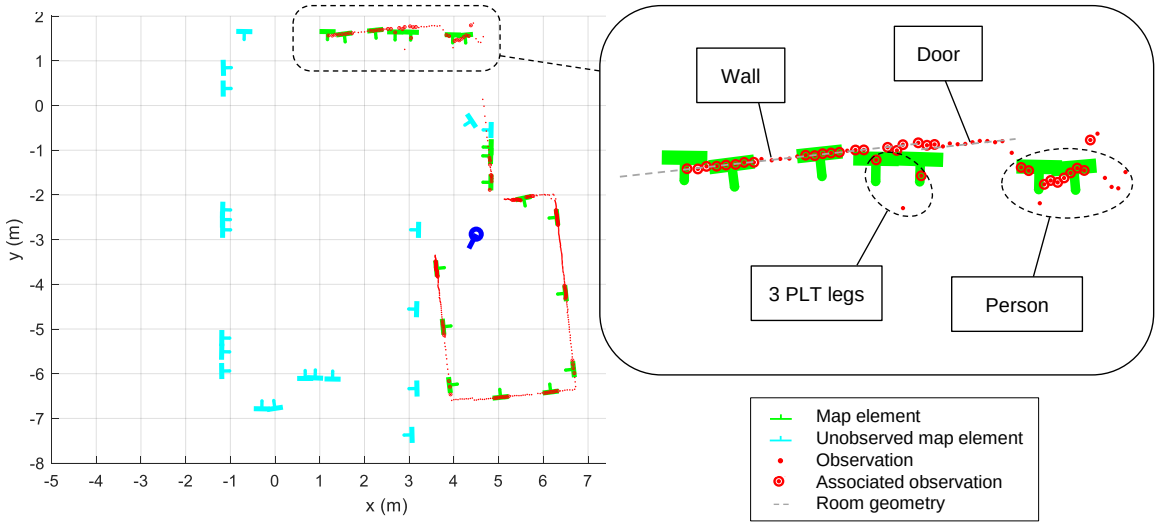


Figure 4-12: An example of a situation with multiple correspondence errors. The Lidar observes a wall in the room with the PLT tripod and a person in front of it. Two of the three PLT legs and the person were associated with the map element that represent the wall, introducing a significant error in the SLAM process.

When the large map elements were selected in the *VGN central* dataset, the association errors as shown in Figure 4-12 did not occur when the Lidar entered the small room. This room was reconstructed successfully without the rotation. However, map reconstruction problems occurred when the Lidar entered the hallway as shown in Figure 4-13. Because the straight wall in the newly discovered area was interrupted by clutter in front of it, the linear descriptor value of the local clusters exceeded the threshold (see §3-2-4).

Next, the algorithm runs out of horizontally oriented correspondences as illustrated in Figure 4-14. This Figure shows the last SLAM iteration before failure. The existing horizontally oriented correspondences are occluded, and no new ones have been found in the uncharted area.

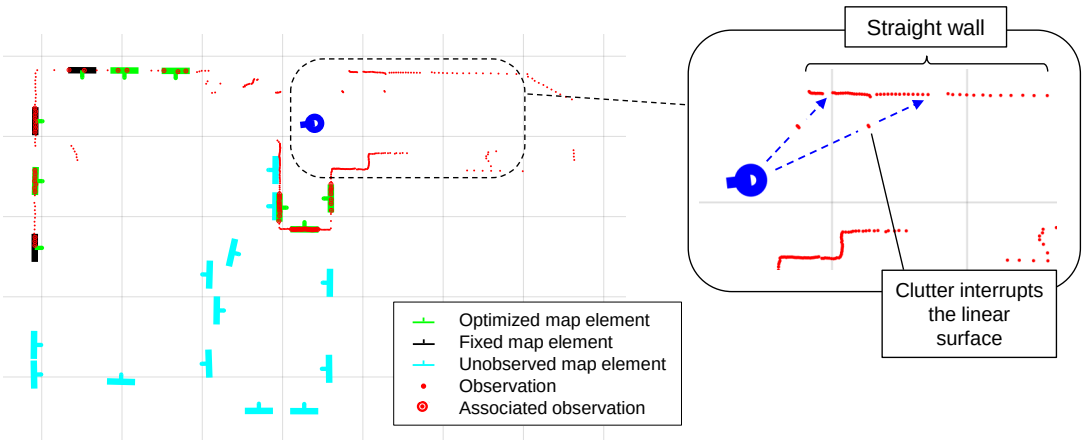


Figure 4-13: The clutter in front of the wall prevents map elements to be found at the wall.

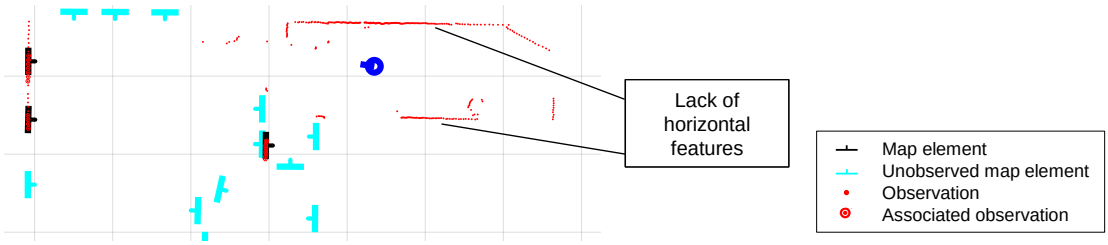


Figure 4-14: Only vertically oriented map elements are visible to the Lidar. The Lidar's pose is not properly constrained due to the lack of horizontally oriented map elements, causing the algorithm to fail.

4-3-3 Opening a door

A separate dataset was recorded with no ground truth information in the basement of the Innovation Center building at the Hilti campus to evaluate the scenario of opening a door in front of the Lidar. It was found that Ray-SLAM has a high probability of failure when a door is opened in front of the Lidar, and the measurement setup next moves through the doorway. This process is illustrated in Figure 4-15.

In the figure it can be seen that one map element represents the the door in closed position. The map element is unable to follow the movement of the door because the algorithm had already accumulated confidence in map element parameters over the last few seconds and the assumption was made that there is no movement in the structure. Correspondence errors can be observed with both the partially opened door and the person in front of the door who opens it. Ray-SLAM's ability to remain accurate when moving through such a door opening depends highly on the amount of available map elements around the Lidar to prevent the SLAM process from collapsing.

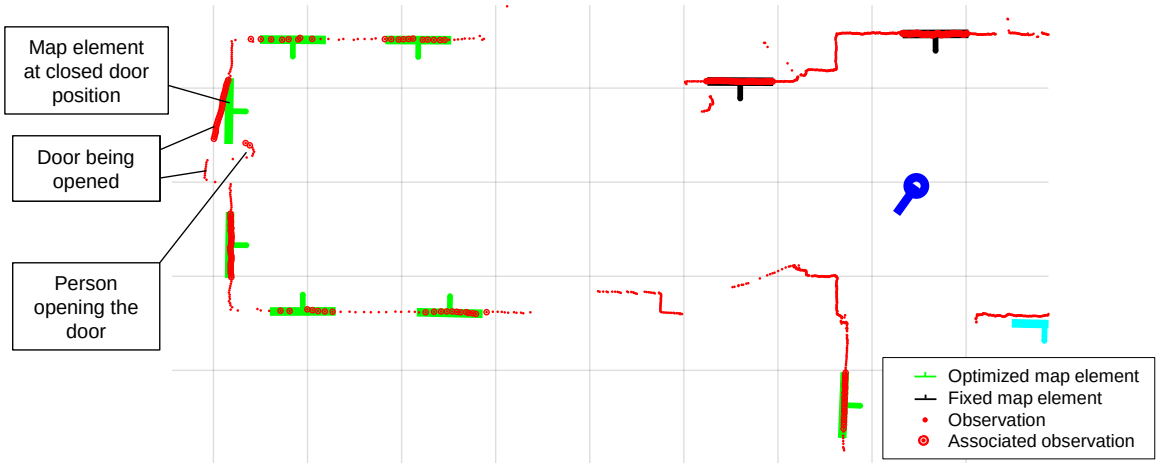


Figure 4-15: A door is opened during the recording of a dataset. A map element was placed at the closed position of the door, and starts to introduce errors as the door is being opened.

4-3-4 Conclusions of the experiment results and robustness analysis

In summary, a total of six datasets with ground-truth have been processed by the Ray-SLAM algorithm using both small and large map elements, resulting in twelve test cases. In seven of these cases, Ray-SLAM reconstructed the map and trajectory successfully. In two cases, Ray-SLAM was partially successful and estimated 11 out of 13 waypoints. Three test cases resulted in failure. One additional dataset was recorded to specifically assess the scenario of opening a door in front of the Lidar. The causes for failure that were observed, including potential improvements, are listed as follows:

- **Map building errors.** Ray-SLAM requires a multitude of map elements to be observed in the scene for localization. Whenever these map elements become unavailable, Ray-SLAM is likely to fail. There can be different reasons for why the map elements become unavailable, which are (1) the scene contains too few walls to which map elements can be attached. Alternatively the walls may be irregular or cluttered, preventing the map element extraction process to generate the map elements. (2) The repetitive static conditions are not sufficiently present in the Lidar's motion (note that the SLAM algorithm does not control the movement of the measurement setup). When the Lidar is moved into an uncharted area, existing map elements may become occluded due to the room's geometry. Adding new map elements is therefore necessary. The Lidar should be stopped at locations where current map elements *and* potential areas for new map elements are visible at the same time, so that the map elements of the new area can be properly connected to the existing map. Potential solutions to these problems would be: (1) increase the map density, (2) establish closed-loop control, i.e. Ray-SLAM actively stops the robot when needed, (3) undistort Lidar scans using the IMU such that static conditions are not needed and (4) initialize the map $\hat{\mathcal{M}}_0$ using the architectural building model.
- **Correspondence errors.** One example with eleven prominent correspondence errors was highlighted in Figure 4-12. Cases where only one or two correspondence errors are

made locally happened more often, but typically go unnoticed because the redundant amount of map elements around the Lidar prevent large errors. Correspondence errors could be mitigated in various ways. Tighter tuning of s_{corr} is one option. Recall that observations with a prediction error above $y - \tilde{y} < s_{\text{corr}}$ are rejected as correspondences. However by doing so, problems can be introduced elsewhere, e.g. in larger scenes a higher s_{corr} may be needed to prevent missing correspondences. A more robust way would be to add object recognition using visual information to the SLAM pipeline, similar to the work of Blum et al. [9]. This work utilizes camera images and a robust neural segmentation network to distinguish structure from clutter on a construction site. The classification is then attached to Lidar observations in order to robustify the process of establishing correspondences.

- **Moving objects.** It was noted that moving objects within the Lidar's line of sight were often not a problem for Ray-SLAM, because (1) moving objects such as persons do not have a linear shape and their observations therefore do not spawn new map elements, and (2) moving objects are typically further away than s_{corr} from a map element such that no correspondence will be established. All datasets recorded in the VGN building contain up to three persons that continuously walk around the scene within the line of sight of the Lidar. Moving objects do become a problem when the object is of linear shape, because then Ray-SLAM will attach a map element to this object and consequently assume that this object does not move. Such a situation occurs when opening a door for example. Dealing with such problems could be done by disabling certain map elements when the residuals associated with this element become too high.

As discussed in Chapter 2, a SLAM engine can be divided into a front-end and a back-end. In the case of Ray-SLAM, it was found that errors were only introduced by the front-end as a result of challenging geometry of the scene. Once the front-end part of the problem was set up correctly, the back-end part consistently generated estimates of the pose and map with a high degree of accuracy.

A different approach to increasing the robustness of a SLAM algorithm is to adapt the back-end side of the algorithm in such a way that the impact of errors in the optimization problem is smaller, by introduction of robust loss functions to model more heavy-tailed distributions with respect to the normal distribution. A state-of-the-art method for doing so is proposed in [57]. This work proposes an adaptive robust kernel for nonlinear least squares optimization that is specifically designed to deal with outliers in the context of Lidar SLAM.

4-4 Accuracy analysis

In this section, the accuracy of Ray-SLAM is analyzed and compared with other state-of-the-art algorithms.

Comparing Ray-SLAM with 2D SLAM works. As pointed out in §2-3-2, the state-of-the-art in 2D Lidar SLAM is based on coarse (typically 5 cm) occupancy grid type maps that have limited ability to scale the map resolution. The accuracy of such an algorithm is limited to its grid size. The six datasets were processed with both Cartographer [45] and

Hector SLAM[1]. In summary, it was found that: (1) Cartographer was able to estimate a trajectory and map using the default 5 cm grid size. Rotation of the measurement setup often resulted in mapping errors, such as a duplicate of a wall 20 cm behind it, even though the IMU was utilized. (2) Cartographer failed on all datasets when the grid size was reduced to 2 cm. (3) Hector SLAM failed on five out of six datasets using the default 5 cm grid size. Failure typically happened when the measurement setup was rotated. Given these results, the 2D algorithms were not considered relevant for comparison in the context of high-accuracy SLAM.

Comparing Ray-SLAM with 3D SLAM works. Instead, the 2D Ray-SLAM algorithm is compared directly with two 3D SLAM algorithms. The LOAM [18] and FAST-LIO2 [56] were selected for accuracy comparison. LOAM is one of the most influential works in Lidar SLAM as discussed in Chapter 2, which was renamed to ALOAM in an open-source implementation[2]. FAST-LIO2 was chosen because it is the highest ranking competitor on the Hilti SLAM Challenge data sets[12].

It is important to be aware of the limitations of the 2D-to-3D comparison, because the respective 3D algorithms were designed to solve an extended problem under a different set of assumptions using additional sensor data:

- The 3D SLAM algorithm needs to estimate three more degrees of freedom of the Lidar's pose: height, roll and pitch.
- The 3D SLAM algorithms do not assume repetitive static conditions.
- The 3D SLAM algorithm has access to the full 3D point cloud of 64 horizontal scan rings instead of 1 scan ring for 2D SLAM.
- The 3D SLAM algorithm may optionally use the IMU data.

Despite these differences, both 2D and 3D SLAM methods generate a trajectory using Lidar and (optionally) IMU sensor recordings. The comparison of these trajectories is still considered meaningful if the Lidar's motion is constrained to the 2D plane.

ALOAM was configured using the recommended settings for indoor scenes, which feature an increased map resolution. For FAST-LIO2, the default configuration was used for Ouster Lidars. The 'blind' parameter was changed from 4 m to 2 m. This parameter specifies the Lidar cropping distance, i.e. only measurements with a distance larger than the blind parameter are kept. This was done to allow FAST-LIO2 to work better in small rooms.

4-4-1 Trajectory/waypoint alignment procedure

Ray-SLAM provides an estimate of the Lidar's trajectory in the SLAM frame, denoted by $\hat{T}_{0:T}^{SL}$, as was illustrated in Figure 3-4. Similar to the Hilti SLAM Challenge [12], the ground-truth information was recorded at static points only using the PLT-300 Total Sation, which was shown in Figure Figure 4-2 (except for the *Motion Capture Lab* dataset). The PLT measures only the position of the reflective marker, and not its orientation. We denote the

[1] http://wiki.ros.org/hector_slam
[2] https://github.com/HKUST-Aerial-Robotics/A-LOAM

marker by M, and the i^{th} waypoint position of the marker in the PLT's world frame W by $\mathbf{p}_{\text{wp},i}^{WM}$. To align the Ray-SLAM trajectory estimate with the ground-truth marker positions of the PLT, we use the 2D simplification of the method that was used in the Hilti SLAM Challenge [12]. This method is explained now in detail.

We start with extracting the waypoint transformations of the Ray-SLAM trajectory estimate by averaging the pose estimates $\hat{\mathbf{x}}_{\text{pose},t}$ at locations where $b_{\text{stat}} = \text{True}$. By use of (A-4), Ray-SLAM's waypoint estimates $\hat{T}_{\text{wp},i}^{SL}$ at the waypoints are obtained. The waypoint estimates are now transformed to the world frame W in which the ground-truth waypoints were recorded using

$$\hat{T}_{\text{wp},i}^{WM} = T^{WS}\hat{T}_{\text{wp},i}^{SL}T^{LM} \tag{4-1}$$

where $T^{LM} = \mathcal{I}_3$ since the marker/prism is positioned above the optical center of the Lidar. Finding T^{WS} that aligns the two trajectories[3] is done by solving the optimization problem

$$\{\hat{x}, \hat{y}, \hat{\theta}\} = \underset{x,y,\theta}{\arg\min} \sum_{i=1}^{N_{\text{wp}}} \left\| \mathbf{p}_{\text{wp},i}^{WM} - (T^{WS}(x,y,\theta)\hat{T}_{\text{wp},i}^{SL}T^{LM})_{\text{pos}} \right\|_2^2 \tag{4-2}$$

where $(\cdot)_{\text{pos}}$ represent the linear translation components of the transformation matrix and $T(x,y,\theta)$ represents the conversion (A-4) from position/heading to a transformation matrix in $SE(2)$. The problem is solved using MATLAB's `lsqnonlin`$(\cdot)$. The estimation error at a waypoint $e_{\text{wp},i} \in \mathbb{R}$ is now given by the residual of (4-2):

$$e_{\text{wp},i} = \left\| \mathbf{p}_{\text{wp},i}^{WM} - (\hat{T}_{\text{wp},i}^{WM})_{\text{pos}} \right\|_2 . \tag{4-3}$$

4-4-2 Accuracy results

In order to analyze the accuracy of the estimate, the Mean Absolute Error (MAE) and the Maximum Absolute Error (Max-AE) metrics at the waypoint locations are obtained from (4-3). These error metrics are compared against those of ALOAM and FAST-LIO2. The MAE gives a general metric on the accuracy of the SLAM algorithm, while the Max-AE describes the largest error in the error sequence, quantifying the algorithm's robustness to estimate outliers. Table 4-3 summarizes these error metrics that were obtained from the 6 experiments. *AL* represents ALOAM, *FL2* represents FAST-LIO2, *RS(L)* represents Ray-SLAM with map elements of $r = 0.35$ m and *RS(S)* represents Ray-SLAM with map elements of $r = 0.18$ m. Tuning of s_β was set to 1 for the point-to-line error metric.

[3] In §4-3, several plots were shown of Ray-SLAM's trajectory estimate alongside the ground-truth marker positions. The reverse transformation T^{SW} was used to transform the marker positions to the SLAM frame to be able to generate these plots.

Table 4-3: Error comparison at the static waypoint locations. Failure cases are marked red. Two cases of dataset *VGN floor 1(b)* were partially successful (see §4-3) and are marked purple. The errors of these datasets are based on the first 11 out of 13 waypoints. The values between brackets (·) indicate the Max-AE, i.e. the largest outlier in the waypoint error sequence.

Dataset	MAE (Max-AE) [mm]			
	AL	FL2	RS(L)	RS(S)
Motion capture lab	16.5 (31.4)	10.2 (22.1)	**2.8 (3.8)**	4.4 (9.3)
VGN floor 1(a)	35.6 (64.3)	15.1 (45.2)	6.7 (14.3)	**6.2 (13.7)**
VGN floor 1(b)	28.0 (81.9)	11.3 (18.4)	**6.9 (13.3)**	7.4 (11.9)
VGN floor 2	24.2 (53.7)	15.4 (43.0)	4.9 (8.9)	**4.2 (11.2)**
VGN central	36.3 (62.6)	**25.7 (67.0)**	fail	111.9 (389.3)
VGN top	38.2 (84.1)	16.6 (44.8)	fail	**6.2 (8.8)**

As indicated in §4-3-1, no strict separation was applied between tuning- and evaluation datasets. Therefore it should be noted that overfitting the algorithm to the datasets cannot be ruled out. In order to quantify the potential increase in accuracy, Table 4-4 compares only the (partially) successful cases of Ray-SLAM with small map elements with ALOAM and FAST-LIO2.

Table 4-4: Error comparison summary, considering only the (partially) successful cases of Ray-SLAM using small map elements. These cases contain 69 out of 82 waypoints that were recorded over the six trajectories. The values between brackets (·) indicate the largest outlier in the error sequence (Max-AE).

Metric	MAE (Max-AE)		
	AL	FL2	RS(S)
[mm]	28.5 (63.1)	13.7 (34.7)	5.7 (11.0)
Relative	5.0× (5.7×)	2.4× (3.2×)	1.0×

The following conclusions can be drawn from the accuracy analysis:

- The overall trend is that in the (partially) successful cases, Ray-SLAM outperforms ALOAM and FAST-LIO2 with a significant margin. In these (partially) successful cases, the overall improvement with respect to FAST-LIO2 is ×2.4 for the MAE and ×3.2 for the Max-AE. With respect to ALOAM, the overall improvement is ×5.0 for the MAE and ×5.7 for the Max-AE.
- It depends on the dataset whether the small or large map elements result in higher accuracy. Small map elements seem more robust against challenging scenes where large walls are not available.
- Although the (partially) successful cases show a significant increase in accuracy, the proposed solution is not yet accurate enough for applications such as the Jaibot that require consistent sub-5 mm accuracy. Outliers with an error > 10 mm were typically present.

4-4-3 Ablation study

In order to gain a better understanding of why the results of Ray-SLAM show an increase in accuracy with respect to the compared algorithms, an *ablation study* was carried out. Such a study comprises of systematically disabling certain functionality the algorithm in order to gain insight into which parts contributed to the end result, and in what extent. The study was limited to the (partially) successful datasets. The following ablation cases are considered:

1. No map optimization. The optimization vector contains only the Lidar's pose and no map variables, and as such the map element poses are not fine tuned over time.
2. Disregard the map building strategy that only appends the map when the robot is stationary. Instead, do continuous map extension: attempt to add new map elements every 2.0 seconds (typical Lidar SLAM algorithms append the map continuously). The map variables are optimized at every SLAM loop iteration.
3. Continuous map extension (similar to case 2), but with no map optimization.

One novelty that could not be a part of the ablation study is the map representation using line segments, as this feature is hard to adapt to the commonly used point-cloud map representation. The results are shown in Table 4-5 below.

Table 4-5: Ablation study results. The relative error increases over the available datasets are shown with respect to the original Ray-SLAM algorithm in three ablation cases. *RS(L)* represents Ray-SLAM with map elements of $r = 0.35$ m and *RS(S)* represents Ray-SLAM with map elements of $r = 0.18$ m. The values between brackets $(\cdot)$ indicate how much the largest outlier in the error sequence has increased (Max-AE).

Ablation case	MAE (Max-AE)	
	RS(L)	RS(S)
1. No map optimization.	1.3× (1.4×)	2.9× (2.9×)
2. Continuous map extension.	1.8× (3.1×)	2.2× (2.6×)
3. Continuous map extension + no map optimization.	1.8× (1.7×)	2.6× (2.5×)

The following conclusions can be drawn from the ablation study:

- Both the map optimization strategy and extending the map at waypoints only bring an overall improvement in the accuracy of Ray-SLAM. Map optimization successfully improves the map accuracy by using more observations, and the map extension procedure at static waypoints successfully prevents motion distorted observations to corrupt the map.
- Small map elements suffer significantly more from disabling the map optimization than large map elements. This can be explained by the fact that only few observations are used to spawn these elements due to their limited length, and the placement is therefore more sensitive to noise. Enabling map optimization filters out this noise by using more measurements to determine the map element poses.

The overall trend that can be observed is that using more (undistorted) observations to establish the map brings a higher accuracy.

4-4-4 Remainder of the error on successful datasets

In the robustness analysis (§4-3) the error sources have been discussed that caused either a failure or a large (> 10 cm) error. This section takes a closer look at potential error sources during processing of the successful datasets that did not lead to large errors or failure. Only for some of these error sources it was possible to carry out a specific experiment to quantify its impact. The list of error sources in this section should be considered non-exhaustive.

Imperfect Lidar range measurements. The range measurements produced by the Lidar are corrupted with noise and may be biased. Both the bias and noise of the sensor may depend on the beam angle β and surface type of the detected object. These aspects were not modeled by the Ray-SLAM algorithm.

Imperfect ground-truth information. The PLT-300 Total Station is specified to have an accuracy of sub-3 mm, and the motion capturing system sub-0.5 mm. Furthermore the optical center of the reflective prism/marker may not be perfectly above the optical center of the Lidar, resulting in errors in the ground-truth information.

Violation of the 2D assumption. The setup of Figure 4-2 may exhibit small amounts of roll, pitch and heave motion due to an uneven floor or slight misalignment in the leveling. During the experiments, ± 5 mm heave and $\pm 0.5^\circ$ was typically observed by the Motion Capturing system and by ground plane estimation of the Lidar data. Another violation of the 2D assumption is that the Lidar scan ring is oriented 0.7° upwards, meaning that a different height of the wall is detected as the Lidar is moved away from this wall. In the search for evidence of the violated 2D assumption, we consider the evolution of map variables $d_{\text{pos},i}, d_{\varphi,i}$ over time as shown in Figure 4-16. This figure shows that at different static waypoints, the map variables want to converge to different values. This is most prominent in $d_{\text{pos},2}$ between $t = 5$ sec and $t = 55$ sec, showing 10 mm difference. This can be interpreted as disagreement about where the wall is perceived to be from different viewpoints, which could originate from a (small) violation of the 2D assumption.

Placement of map elements on reflective surfaces. Ray-SLAM does not distinguish between matte and reflective surfaces. Matte surfaces are detected with 3 cm accuracy, and reflective surfaces with 10 cm accuracy as specified by the manufacturer. Map elements placed on windows or whiteboards for example are currently weighted equally as map elements placed on walls in the optimization problem of (3-27), which is cumbersome. When adding additional information about the type of object that the map element represents, each measurement residual could be weighted appropriately.

The dataset *Motion capture lab* has a large amount of reflective surfaces present, allowing to quantify the effect of the reflective surfaces on the accuracy. In order to do so, processing of this dataset was carried out whilst excluding certain map elements in the SLAM process. The algorithm was modified for this experiment such that only the map elements were used that had been found during initialization as shown in Figure 4-17, and no new map elements were added.

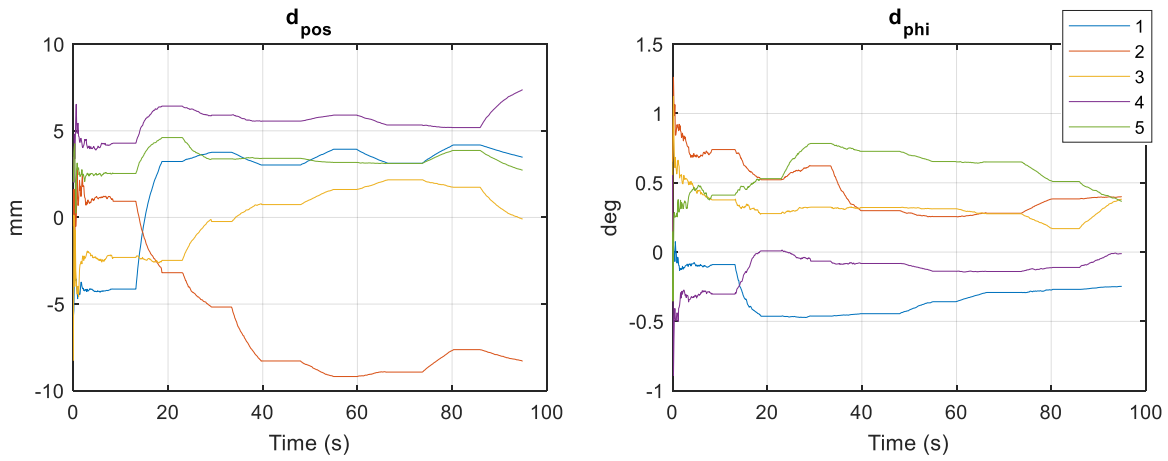
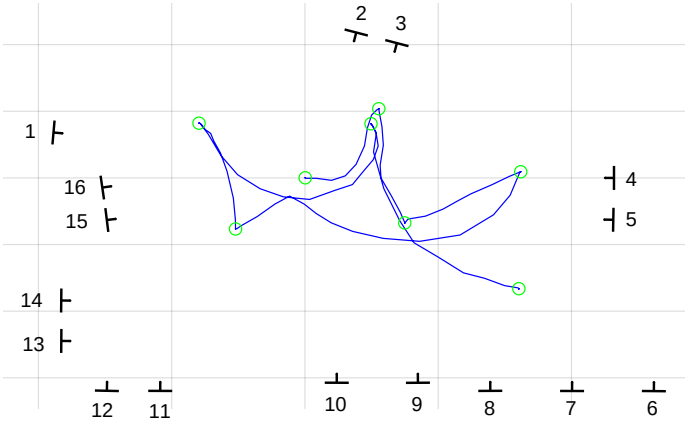


Figure 4-16: Plot of 5 map variables $d_{\text{pos},i}, d_{\varphi,i}$ over time of dataset *Motion capture lab* with small map elements. Only five out of nine map elements are shown to keep the figure readable.



Exclusion	MAE	Max-AE
∅	3.4	5.4
2, 3	3.5	4.0
4, 5	3.9	5.6
9, 10	3.5	5.9
15, 16	3.5	5.8
2, 3, 15, 16	3.7	5.1
4, 5, 15, 16	4.0	7.8
2, 3, 4, 5, 15, 16	4.5	10.4
Original	*4.4*	*9.3*

Figure 4-17: Left. The 16 map elements with $r = 0.18$ m of dataset *Motion capture lab* are shown that were found during initialization. **Right.** Accuracy comparison when excluding various reflective planes from the SLAM process.

Map element 6, 7, 8, 11, 12, 13 and 14 represent a concrete wall. Map element 4, 5, 15 and 16 represent a reflective whiteboard. Map element 9 and 10 represent the partially reflective calibration poster. Map element 2 and 3 represent a TV, and map element 1 represents a painted door respectively. Map elements 8, 11 and 14 were chosen as anchor map elements. The table in the right of Figure 4-17 shows the MAE and Max AE after excluding different sets of reflective surfaces.

Based on these results, it can be concluded that only excluding the TV as reference improves the accuracy. The data suggests that the reflective whiteboards act as reliable references. Exclusion of whiteboards from the SLAM process results in increased errors. It should further be noted that the original algorithm has an increased error over the version that only used the initialization map elements, suggesting that errors were introduced by map elements that were added after initialization. This error source was not further investigated.

Near-flat surfaces. Ray-SLAM's ability to distinguish flat from non-flat surfaces depends highly on the tuning parameter s_l as described in §3-2-4. Recall that s_l represents a threshold value on the linear descriptor value l, which represents the flatness of a local cluster of points. The threshold s_l was set slightly higher than a typical linear descriptor value l for an observed flat wall with normal sensor noise levels. This has the effect that uneven surfaces can only be distinguished from walls if the irregularity of the surface is larger than the noise level of the sensor. Objects such as closed drawers and transitions between a wall and a closed door or poster may therefore mistakenly be interpreted as a wall. Such errors were common during the processing of various successful datasets.

4-5 Tuning of beam angle discount factor

This section takes a closer look at the tuning of the Beam Angle Discount Factor s_β. Only three datasets were considered and only large map elements were used to limit the amount of processing time required for this study. The tuning parameter s_β was changed from 0 to 2 with 0.2 increments. Increasing s_β results in down weighing the Lidar observations with a steep beam angle β as shown in Figure 3-8 in the optimization problem of (3-27). Tuning of $s_\beta = 0$ represents the ray casting error term, and $s_\beta = 1$ represents the point-to-line error term which is typically used in the literature. The result of this process is shown in Figure 4-18.

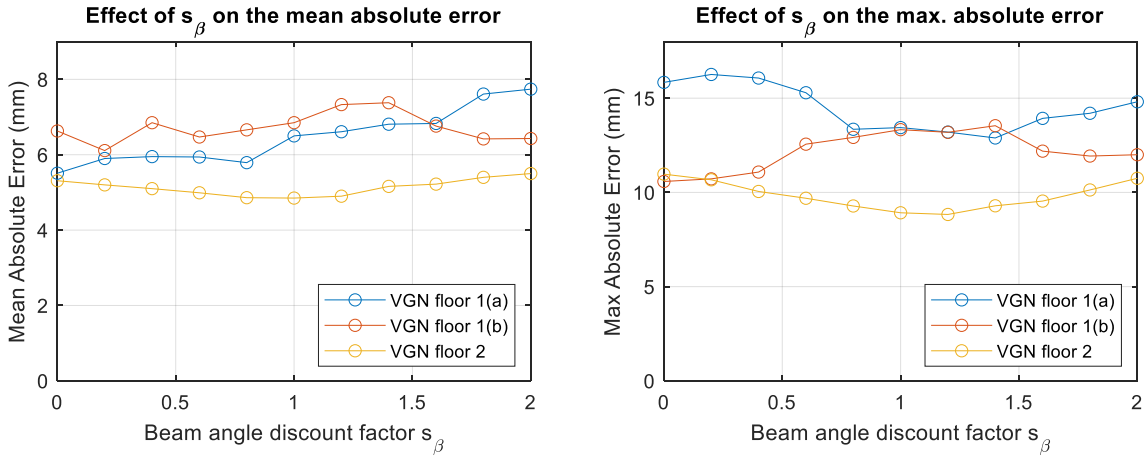


Figure 4-18: Effect of tuning the Beam Angle Discount Factor s_β.

The data suggests that there does not seem to be a particular value of s_β that performs consistently better on the given datasets, and the trend between the datasets contradict to a certain extent. Depending on the dataset, a different s_β might be more favorable, resulting in only a small improvement of the error. In practice, however, one does not have the ground truth available and choosing the right s_β for the particular scene is not an option. The point-to-line configuration of $s_\beta = 1.0$ seems a good all-round performer. Further investigation of the beam angle discount factor is suggested to see if using s_β other than 1 can be justified. For example, one could investigate (1) if there is correlation between the scene type and the effect of s_β, by considering the histogram of the inclination angle β of all observations, or (2)

consider the largest waypoint estimate outlier, and try to identify the reason how s_β affects the accuracy of this estimate. These topics are considered outside the scope of this thesis.

4-6 Effect of the anchor map elements

This section investigates the impact of the anchor map elements on the performance of Ray-SLAM using the dataset *Motion capture lab* as an example. For the purpose of improving the system's observability, the manual step of defining three anchor map elements was introduced in §3-2-4. The observability of the state $\mathbf{x}_t$ together with the identifiability of the map offsets $\mathbf{d}_{\text{opt},t}$ of the estimation problem (3-27) will be further analyzed. Since $\mathbf{x}_t$ and $\mathbf{d}_{\text{opt},t}$ are jointly optimized, the observability/identifyability[4] study is combined. This section proceeds with the term observability whilst referring to both properties. It should be noted that the measurement model of (3-10) is nonlinear. Therefore, instead of studying the observability directly, the *local observability* [58] is studied by considering the linearized version of (3-27), making use of the Jacobian J. If J is full rank, the optimization vector is considered to be *locally observable*.

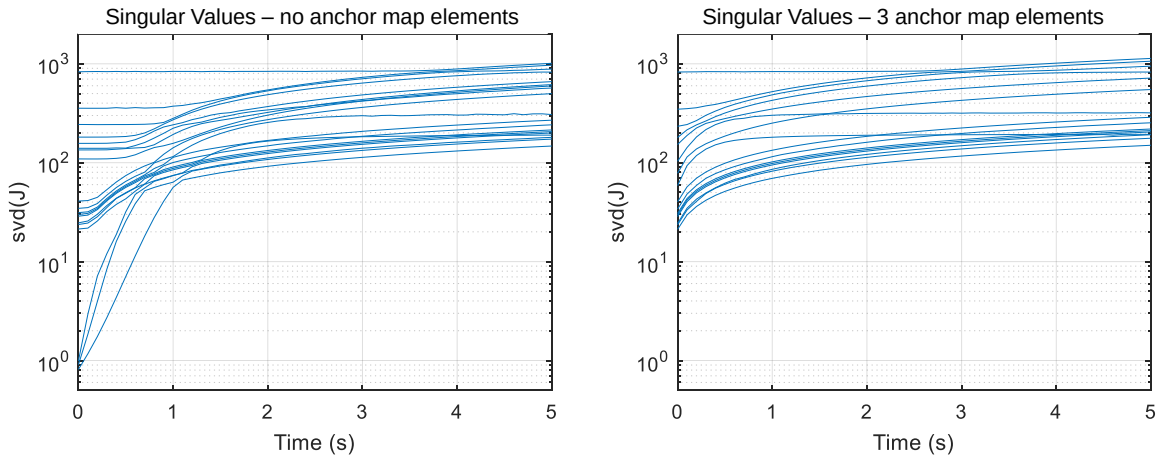


Figure 4-19: Left. Singular values of the Jacobian matrix J on a logarithmic scale when no anchor map elements were selected. The Lidar remained stationary over the 5 second period. **Right.** Singular values when selecting 3 independent anchor map elements.

Figure 4-19 shows the rank analysis of J by considering its singular values over time. An interval of 5 seconds is shown starting at $t = 0$. On the left, the SLAM problem without any defined anchor map elements is shown. It can be seen that three singular values are significantly lower (roughly $\times 20$) at $t = 0$. However the singular values remain far away from zero, which would indicate a rank deficiency. By applying the technique of defining three anchor map elements, it can be seen on the right that at $t = 0$ the lowest singular value is increased by a factor of roughly 20, significantly improving the local observability of the estimation problem. However, when not defining the anchor lines, the low singular values

[4] As pointed out in §3-1-2, the SLAM system may have unobservable modes where the Lidar pose and map move together. These unobservable modes would not be revealed when the study of state observability is decoupled from map parameter identifyability.

increase quickly such that after 0.8 second the local observability is similar to the starting condition of the case with three anchor map elements.

In practice it was found that processing the datasets with and without defining the anchor map elements resulted in similar ($\pm 2\%$) pose estimation accuracy. It can therefore be questioned whether the process of defining anchor map elements is necessary.

Chapter 5

Conclusions and Recommendations

This concluding chapter answers the research questions that have been posed in the introductory chapter. Furthermore, the recommendations for future work are given.

5-1 Conclusions

The main goal of this thesis was to develop a SLAM algorithm that would improve the localization accuracy for construction robots such as the Jaibot, preferably to sub-5 mm accuracy. The algorithm was required to work independently of an architectural building model and rely solely on on-board sensing. As demonstrated by the Hilti SLAM Challenge [12], the state-of-the-art in SLAM currently provides an accuracy of around 5 cm with outliers that vary per scene, which motivates the need for further research. The main question of this thesis was therefore formulated as:

How can the state-of-the-art in Lidar-centric SLAM be improved to achieve sub-5 mm accuracy on a building construction site?

To answer this research question, three sub-questions were posed in the introductory chapter. This thesis project led to the following answers.

How can a Lidar SLAM algorithm exploit the specific scene structure that can be expected on a building construction site?

This thesis proposes Ray-SLAM, a 2D Lidar SLAM algorithm which is centered around the assumption walls are available in the scene, which is typically the case on a building construction site. The algorithm builds and maintains a novel sparse map representation that consists of line segments, and uses this map consequently for navigation. The position- and heading offset parameters of the map are optimized jointly with the Lidar's pose, in such a way that the map accuracy is increased over time. This sparse map representation allows the introduction of a novel Non-Iterative Closest Point (NICP) observation-to-map alignment procedure, which yields reduced correspondence errors and increased computational efficiency with respect to traditional Iterative Closest Point (ICP). A novel stop-and-go strategy is

Table 5-1: Error comparison summary, considering only the (partially) successful cases of Ray-SLAM with small map elements of $s_r = 0.18$. These cases contain 69 out of 82 waypoints that were recorded over the six trajectories. The values between brackets $(\cdot)$ indicate the largest outlier in the error sequence (Max-AE).

Metric	MAE (Max-AE)		
	AL	FL2	RS(S)
[mm]	28.5 (63.1)	13.7 (34.7)	5.7 (11.0)
Relative	5.0× (5.7×)	2.4× (3.2×)	1.0×

utilized to only improve the map while the Lidar remains stationary to prevent motion-distorted Lidar scans corrupting the map. These novelties together (joint pose-and-map optimization scheme, NICP, stop-and-go strategy) distinguish Ray-SLAM from related works in Lidar-centric SLAM such as LOAM [18] and FAST-LIO2 [11] (winner of the Hilti SLAM Challenge). Such works are based on dense point cloud maps and the ICP algorithm.

How robust is the novel algorithm in representative real-world scenarios with various levels of complexity and moving obstacles?

The algorithm was extensively tested using real-world datasets. Data was recorded using the Ouster OS0 Lidar and InvenSense ICM-20948 IMU. A total of six indoor trajectories were captured with a total traveled distance of 307 m, covering rooms of various shapes and sizes, and each with different amounts of clutter. The trajectories were compared at 82 waypoint locations. Each dataset was processed with a map element radius of both $s_r = 0.18$ m and $s_r = 0.35$ m, resulting in twelve experiments. It was observed that Ray-SLAM was successful in seven out of twelve cases, and partially successful in two cases where Ray-SLAM was only able to estimate 11 out of 13 waypoints. Failures of Ray-SLAM can be traced back to either (1) limited ability to add new map elements appropriately, resulting in lack of references later on, (2) correspondence errors due to complex scene geometry or (3) opening a door in front of the Lidar. The algorithm did not suffer from people who moved around the scene. Section §5-2 provides recommendations on what steps can be taken to further improve Ray-SLAM's robustness. The compared algorithms ALOAM [18] and FAST-LIO2 [11] were successful on all datasets.

What is the accuracy of the novel SLAM algorithm, and how does it compare with other state-of-the-art Lidar-centric SLAM algorithms?

The accuracy of Ray-SLAM was evaluated and compared with ALOAM and FAST-LIO2 by analyzing the absolute position error at static waypoint locations. Ground-truth positions were recorded at these waypoints using a Total Station (Hilti PLT-300) or a motion capturing system (OptiTrack). Both ALOAM and FAST-LIO2 estimate a 3D trajectory and map using all 64 Lidar scan rings and (optionally) an IMU, whereas Ray-SLAM estimates a 2D trajectory and map using only one Lidar scan ring. The comparison should therefore be considered of limited validity. An overall Mean Absolute Error of 5.7 mm is reported by Ray-SLAM in the (partially) successful cases, which is 5.0× more accurate than LOAM and 2.4× more accurate than FAST-LIO2. The accuracy comparison considering only these (partially) successful cases is shown in Table 5-1. No strict separation between tuning- and evaluation datasets was applied, therefore a risk exist that the tuning parameters are overfitted to the datasets.

In summary, it can be concluded that the proposed SLAM algorithm obtains an increased localization accuracy in indoor scenes with respect to the state-of-the-art, given that the scene does not pose major difficulties. Despite the improvement, Ray-SLAM does not provide consistent sub-5 mm accuracy in the successful cases. More research effort will be required to realize this target accuracy. Since the Lidar that was used has a specified range accuracy of ± 3 cm, it is recommended to also seek for improvement of the sensing hardware.

5-2 Future Work

When considering the opportunities for future work, it should be noted that both 2D and 3D SLAM algorithms have their value and potential applications. This section therefore discusses both further improvements of Ray-SLAM as a 2D SLAM algorithm and the generalization of Ray-SLAM to a full 3D SLAM algorithm.

5-2-1 Improve Ray-SLAM as 2D SLAM algorithm

Improve the robustness.

Improving the robustness of Ray-SLAM can be done using various strategies. The following three suggestions are made to improve the robustness:

1. Increase the map density. Multiple failure cases of Ray-SLAM were traced back to an inability to place map elements at walls in certain scenarios, resulting in a lack of references at a later point. It is recommended to investigate how the map density can be increased such that the algorithm is less likely to run out of references, for example by implementing map elements of variable size. Introduction of additional shape primitives such as an edge feature would also help to increase the map density. Alternatively, the map could be initialized using the architectural building model.

2. Down-weigh the impact of outliers and errors. Use the technique of robust adaptive loss functions as proposed by Chebrolu et. al. [57] to accommodate for outliers and errors. This work establishes a novel strategy to accommodate for outliers that is specifically targeted at the ICP algorithm in the context of SLAM.

3. Integrate cameras. Estimate for each Lidar observation if structure or clutter is observed using cameras in order to filter out correspondence errors [9].

Quantify the reliability of the estimate.

In practice, one is not only interested in the estimated pose, but also in the reliability of the respective estimate. Some quantification of the reliability could be established based on e.g. the residuals and covariance matrices of the optimization problem of (3-27) [59]. Such functionality would allow the user to know when to trust the algorithm.

Ability to allow 6DOF motion.

To be able to model violations of the 2D assumption, it is recommended to extend Ray-SLAM to 6DOF pose estimation whilst using the 2D map. This means that the height, roll and pitch

need to be estimated, which can be done for example by Lidar ground plane detection [60] and (optionally) an IMU [21].

Ability to function without repetitive static conditions.

To allow the algorithm to remain more accurate when in motion, it is recommended to implement tightly-coupled IMU fusion [28] such that the motion estimate of the IMU can be used to undistort the Lidar scan [10]. This would enable the algorithm to not require static conditions to perform accurate SLAM.

C++ implementation.

In order to allow Ray-SLAM to run seamlessly on existing hardware platforms and increase the computational efficiency, it is recommended to port the algorithm to the C++ language and implement it as a ROS node [61]. This would allow integration of Ray-SLAM into the modular ROS framework. A high-performance nonlinear least-squares solver algorithm such as Ceres [51] is recommended.

5-2-2 Extend Ray-SLAM to a full 3D SLAM algorithm

The design choices for the 2D Ray-SLAM algrotithm have been made with 3D generalization in mind. In order to do this, it is recommended to take a modern 3D sensor fusion or SLAM algorithm as a basis and implement the 3D generalization of Ray-SLAM as additional functionality. The 3D generalization of the map elements would consist of planar segments (e.g. discs or rectangles) with 3 degrees of freedom per map element: position offset and two degrees of rotational offset freedom. Two suggested basis algorithms are discussed below on which 3D Ray-SLAM can be built on top:

- Use a factor-graph SLAM algorithm such as LIO-SAM [10]. The factor graph structure allows convenient modeling of the planar map elements using landmarks. Functionality such as IMU preintegration, undistorting the Lidar scan and real-time smoothing are incorporated already within the algorithm. LIO-SAM uses an ICP process based on the LOAM [18] feature extraction module, which may serve as useful additional constraints in the SLAM problem in the case that the Ray-SLAM planar map elements become too sparse.
- Use a more general moving-horizon estimator back-end algorithm such as ConFusion [27]. Such an approach would require additional efforts with respect to the first option, as ConFusion does currently not have any functionality specific to SLAM. Modules for undistorting the Lidar scan and loop closure detection for example would have to be developed from scratch. In return, the user will gain the increased flexibility of ConFusion to incorporate static parameters into the optimization problem.

It is hypothisized that such a generalized version of Ray-SLAM can potentially be among the top performers in the Hilti SLAM Challenge, based on the 2D proof-of-concept that this thesis provides.

Appendix A

Rotations and transformations

In this appendix the notation is established to describe the position and orientation of points and rigid bodies in 2D space, and formalize the concepts of rotation and transformation [22, 59].

First, the position vector $\mathbf{p}^A \in \mathbb{R}^2$ is introduced which describes the relative position of point $\mathbf{p}$ with respect to the origin of frame $\mathbf{A}$ in 2D space. Figure A-1 (left) shows the Cartesian x, y and polar r, α representation of point $\mathbf{p}$ in frame A, where point $\mathbf{a}$ is at the origin of this frame. The unit vectors $\mathbf{e}_x^A, \mathbf{e}_y^A$ represent the orthonormal basis of frame A in $\mathbb{R}^2$.

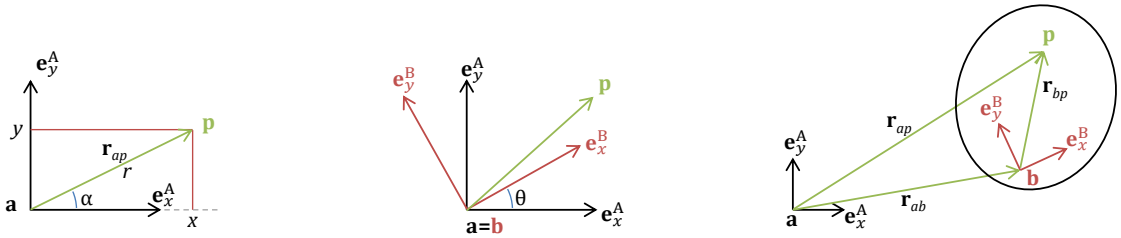


Figure A-1: Left. Cartesian and polar coordinates. **Middle.** Two rotated frames. **Right.** Two frames with relative rotation and translation are related through transformation.

Rotations can have two different interpretations: *passive* and *active* rotations. A passive rotation will represent point $\mathbf{p}$ from a current frame to a new frame (the observer rotates), while an active rotation rotates the point $\mathbf{p}$ within the same reference frame. For this thesis only passive rotations are relevant. An example is shown in Figure A-1 (middle), where point $\mathbf{p}$ is expressed in frame B, which is rotated by angle θ with respect to frame A. The Cartesian coordinates of $\mathbf{p}$ in the rotated frames are related through the rotation matrix R^{BA} that rotates the point from B to A and is defined by

$$R^{AB} = \begin{pmatrix} \cos(\theta^{AB}) & -\sin(\theta^{AB}) \\ \sin(\theta^{AB}) & \cos(\theta^{AB}) \end{pmatrix}. \tag{A-1}$$

The rotation of $\mathbf{p}$ from frame B to A can now be expressed by

$$\mathbf{p}^A = R^{AB}\mathbf{p}^B. \tag{A-2}$$

Note that the origin of frame A and B coincide. The rotation matrix has the properties

$$\det(R) = 1, \quad RR^\top = R^\top R = \mathcal{I}_2, \quad R^{-1} = R^\top. \tag{A-3}$$

Rotation matrices in 2D space belong to the Special Orthogonal group (SO) of $SO(2)$.

When two frames A and B have a relative rotation and a position offset, they are related through their relative *transformation*, see Figure A-1 (right). Transformation matrix T^{AB} is defined as

$$T^{AB} = \begin{pmatrix} R^{AB} & \mathbf{r}_{ab}^B \\ \mathbf{0}_{1\times 2} & 1 \end{pmatrix}, \tag{A-4}$$

and point $\mathbf{p}$ can be transformed from frame B to A using

$$\begin{pmatrix} \mathbf{p}^A \\ 1 \end{pmatrix} = T^{AB} \begin{pmatrix} \mathbf{p}^B \\ 1 \end{pmatrix}. \tag{A-5}$$

The transformation matrix in 2D space is part of the Special Euclidean Lie group (SE) of $SE(2)$. Both rotation matrices and transformation matrices can be chained conveniently. The position and orientation together is referred to as the *pose* of the rigid body. Generalizations of rotation and transformation to 3D are detailed in [22].

Alternatively to the transformation matrix, a transformation from B to A can be parameterized using the more compact *position-heading* parameterization using

$$\mathbf{t}^{AB} = (r_{ab,x}^B \quad r_{ab,y}^B \quad \theta^{AB})^\top. \tag{A-6}$$

Appendix B

Ouster OS0 technical specifications

Table B-1: Ouster OS0 Lidar technical specifications, adapted from [3].

Range (80% diffuse reflectivity)	45 m with > 90% detection probability
Range (10% diffuse reflectivity)	15 m with > 90% detection probability
Minimum range	0.3 m
Range accuracy, diffusely reflective target	±3 cm
Range accuracy, non-diffusely reflective target	±10 cm
Precision, 0.3-1.0m	$\sigma = 2.0$ cm
Precision, 1-10m	$\sigma = 1.0$ cm
Precision, 10-15m	$\sigma = 1.5$ cm
Precision, > 15m	$\sigma = 5.0$ cm
Vertical resolution and field of view	64 channels from $-45°$ to $45°$
Horizontal resolution	2048
Points per second	1310720
Angular sampling accuracy, horizontal	±0.01°
Angular sampling accuracy, vertical	±0.01°
False positive rate	1/10000
Rotation rate	10 Hz
Laser wavelength	865 nm
Data per point	Range, signal, reflectivity, near-infrared, channel, azimuth angle, timestamp

Table B-2: Ouster OS0 IMU technical specifications, adapted from [3].

IMU type	InvenSense ICM-20948
Sampling rate	100 Hz
Data per sample	3 axis gyroscope, 3 axis accelerometer
Accelerometer range	±16 g
Gyroscope range	±200°/s
IMU/Lidar clock synchronisation error	< 1 ms

Appendix C

Ouster OS0 noise properties

This appendix takes a closer look at the noise properties of the Ouster OS0 Lidar. For this, this thesis relies on an internal study that was carried out at Hilti by Michael Helmberger. An experiment was conducted by placing the Lidar 1 m away from a flat wall and recording measurement data for a duration of two hours. The point-cloud data is cropped such that only the points that represent the wall are left over. A plane is now fitted through this set of points by finding the plane that minimizes the squared point-to-plane projection distance [53]. The result of the plane fitting process is seen in Figure C-1 (left), and the distribution of the residuals in Figure C-1 (right).

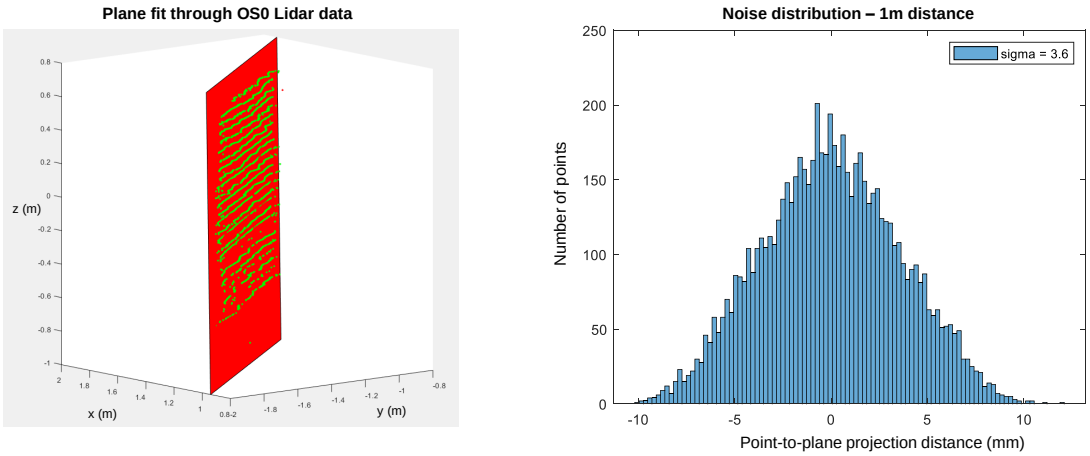


Figure C-1: Left. Least-squares plane fitted through a point cloud that represents a straight wall. **Right.** Distribution of the point-to-plane projection distance. The distance between the Lidar and the wall was 1 meter.

This experiment was repeated for distances of 5 m and 9 m between the Lidar and the wall. The results can be seen in Figure C-2.

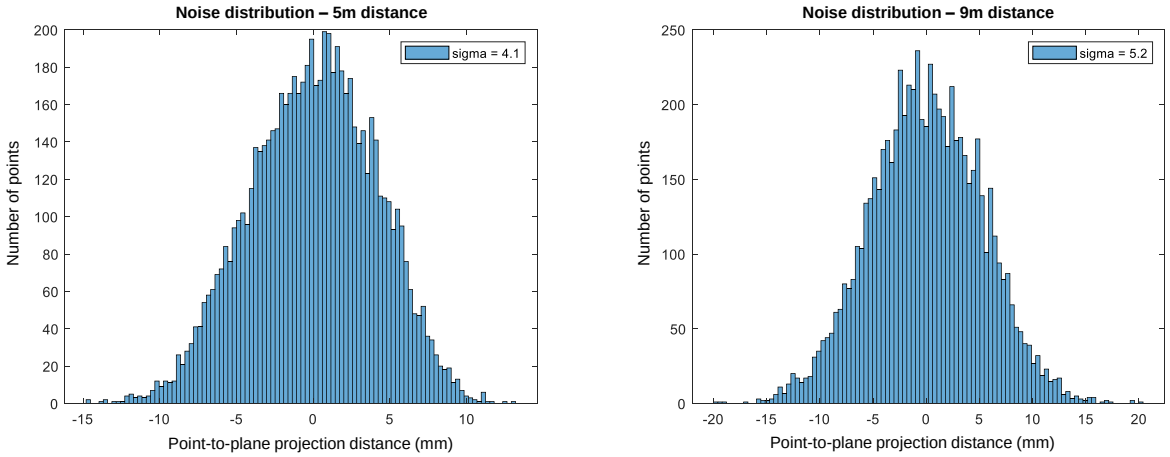


Figure C-2: Left. Noise distribution, Lidar-to-wall distance of 5 m. **Right.** Noise distribution, Lidar-to-wall distance of 9 m.

It can be seen that the distribution of the point-to-plane projection distance follows the typical shape of a Bell curve. Therefore, it can be concluded that the assumption of normally distributed noise is reasonable.

Determination of the bias was not included in this experiment, because obtaining a ground-truth value for each measurement direction requires a significantly harder and more time-consuming experiment. For this, it would be required to know the ground-truth Lidar position and orientation with respect to the wall with very high accuracy. Interpretation of the error in such an experiment would be hard, because it is difficult to know if the error comes from sensor bias or misalignment.

Bibliography

[1] Maintenance Specialists, "What's The Latest On Office Renovations During COVID-19?," [Online]. Available: https://callmsi.com/office-renovation-during-covid-19/ (accessed: November 29, 2021).

[2] M. Helmberger, "Hilti SLAM Challenge," [Online]. Available: https://hilti-challenge.com/ (accessed: November 01, 2021).

[3] Ouster Inc., "Ouster OS0 Datasheet," [Online]. Available: https://data.ouster.io/downloads/datasheets/datasheet-rev05-v2p1-os0.pdf (accessed: November 01, 2021).

[4] J. Buchli, M. Giftthaler, N. Kumar, M. Lussi, T. Sandy, K. Dörfler, and N. Hack, "Digital in situ fabrication - Challenges and opportunities for robotic in situ fabrication in architecture, construction, and beyond," *Cement and Concrete Research*, vol. 112, pp. 66–75, 2018.

[5] T. Sandy, M. Giftthaler, K. Dorfler, M. Kohler, and J. Buchli, "Autonomous repositioning and localization of an in situ fabricator," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 2852–2858, IEEE, 2016.

[6] N. Kumar, N. Hack, K. Dorfler, A. N. Walzer, G. J. Rey, F. Gramazio, M. Daniel Kohler, and J. Buchli, "Design, development and experimental assessment of a robotic end-effector for non-standard concrete applications," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1707–1713, IEEE, 2017.

[7] A. Gawel, R. Siegwart, M. Hutter, T. Sandy, H. Blum, J. Pankert, K. Kramer, L. Bartolomei, S. Ercan, F. Farshidian, M. Chli, and F. Gramazio, "A Fully-Integrated Sensing and Control System for High-Accuracy Mobile Robotic Building Construction," in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 2300–2307, IEEE, 2019.

[8] S. D. Chekole, "Surveying with GPS, total station and terresterial laser scaner: a comparative study," MSc Thesis, School of Architecture and the Built Environment, Royal Institute of Technology (KTH) Stockholm, Sweden, 2014.

[9] H. Blum, J. Stiefel, C. Cadena, R. Siegwart, and A. Gawel, "Precise Robot Localization in Architectural 3D Plans," *arXiv preprint arXiv:2006.05137*, 2020.

[10] T. Shan, B. Englot, D. Meyers, W. Wang, C. Ratti, and D. Rus, "LIO-SAM: Tightly-coupled Lidar Inertial Odometry via Smoothing and Mapping," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 5135–5142, IEEE, 2020.

[11] W. Xu, Y. Cai, D. He, J. Lin, and F. Zhang, "FAST-LIO2: Fast Direct LiDAR-inertial Odometry," *arXiv preprint arXiv:2107.06829*, pp. 1–19, 2021.

[12] M. Helmberger, K. Morin, N. Kumar, D. Wang, Y. Yue, G. Cioffi, and D. Scaramuzza, "The Hilti SLAM Challenge Dataset," *arXiv preprint arXiv:2109.11316*, 2021.

[13] D. Wisth, M. Camurri, S. Das, and M. Fallon, "Unified multi-modal landmark tracking for tightly coupled lidar-visual-inertial odometry," *IEEE Robotics and Automation Letters*, vol. 6, no. 2, pp. 1004–1011, 2021.

[14] T.-M. Nguyen, S. Yuan, M. Cao, T. H. Nguyen, and L. Xie, "VIRAL SLAM: Tightly Coupled Camera-IMU-UWB-Lidar SLAM," *arXiv preprint arXiv:2105.03296*, 2021.

[15] T. Schneider, M. Dymczyk, M. Fehr, K. Egger, S. Lynen, I. Gilitschenski, and R. Siegwart, "Maplab: An Open Framework for Research in Visual-Inertial Mapping and Localization," *IEEE Robotics and Automation Letters*, vol. 3, no. 3, pp. 1418–1425, 2018.

[16] P. Dellenbach, J.-E. Deschaud, B. Jacquet, and F. Goulette, "CT-ICP: Real-time Elastic LiDAR Odometry with Loop Closure," *arXiv preprint arXiv:2109.12979*, 2021.

[17] D. Scaramuzza and F. Fraundorfer, "Visual odometry Part I: The First 30 Years and Fundamentals," *IEEE Robotics and Automation Magazine*, vol. 18, no. 4, pp. 80–92, 2011.

[18] J. Zhang and S. Singh, "LOAM: Lidar Odometry and Mapping in Real-time," *Robotics: Science and Systems*, vol. 2, no. 9, 2014.

[19] G. Grisetti, R. Kummerle, C. Stachniss, and W. Burgard, "A tutorial on graph-based SLAM," *IEEE Intelligent Transportation Systems Magazine*, vol. 2, no. 4, pp. 31–43, 2010.

[20] N. Sünderhauf and P. Protzel, "Towards a robust back-end for pose graph SLAM," in *IEEE international conference on robotics and automation*, pp. 1254–1261, IEEE, 2012.

[21] M. Kok, J. D. Hol, and T. B. Schön, "Using inertial sensors for position and orientation estimation," *Foundations and Trends in Signal Processing*, vol. 11, no. 1-2, pp. 1–153, 2017.

[22] Unknown (2018), "Robot Dynamics Lecture Notes," ETH Zürich [Lecture Notes]. Available: https://ethz.ch/content/dam/ethz/special-interest/mavt/robotics-n-intelligent-systems/rsl-dam/documents/RobotDynamics2018/RD_HS2018script.pdf (accessed: November 01, 2021).

[23] G. Welch and G. Bishop, "An Introduction to the Kalman Filter," *Proc of SIGGRAPH*, vol. 8, no. 41, pp. 27599–23175, 2001.

[24] M. A. Skoglund, G. Hendeby, and D. Axehill, "Extended Kalman filter modifications based on an optimization view point," in *18th International Conference on Information Fusion*, pp. 1856–1861, IEEE, 2015.

[25] K. Madsen, H. B. Nielsen, and O. Tingleff, *Methods for non-linear least squares problems*. 2004.

[26] C. Qin, H. Ye, C. E. Pranata, J. Han, S. Zhang, and M. Liu, "LINS: A Lidar-Inertial State Estimator for Robust and Efficient Navigation," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 8899–8905, IEEE, 2020.

[27] T. Sandy, L. Stadelmann, S. Kerscher, and J. Buchli, "ConFusion: Sensor Fusion for Complex Robotic Systems Using Nonlinear Optimization," *IEEE Robotics and Automation Letters*, vol. 4, no. 2, pp. 1093–1100, 2019.

[28] C. Forster, L. Carlone, F. Dellaert, and D. Scaramuzza, *IMU preintegration on manifold for efficient visual-inertial maximum-a-posteriori estimation*. Georgia Institute of Technology, 2015.

[29] F. Dellaert and M. Kaess, "Factor Graphs for Robot Perception," *Foundations and Trends in Robotics*, vol. 6, no. 1-2, pp. 1–139, 2017.

[30] F. Dellaert, "Factor graphs and GTSAM: A hands-on introduction," Technical Report GT-RIM-CPR-2012-002. Georgia Institute of Technology, 2012.

[31] G. Grisetti, R. Kummerle, Hauke Strasdat, and K. Konolige, "g2o: A general Framework for (Hyper) Graph Optimization," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA) Shanghai, China*, pp. 9–13, 2012.

[32] T. Hackel, J. D. Wegner, and K. Schindler, "Fast Semantic Segmentation of 3D Point Clouds With Strongly Varying Density," *ISPRS Annals of Photogrammetry, Remote Sensing and Spatial Information Sciences*, vol. 3, pp. 177–184, 2016.

[33] K. Klasing, D. Althoff, D. Wollherr, and M. Buss, "Comparison of surface normal estimation methodsfor range sensing applications," in *IEEE International Conference on Robotics and Automation*, pp. 3206–3211, IEEE, 2009.

[34] T. Shan and B. Englot, "LeGO-LOAM: Lightweight and Ground-Optimized Lidar Odometry and Mapping on Variable Terrain," *IEEE International Conference on Intelligent Robots and Systems*, pp. 4758–4765, 2018.

[35] H. Ye, Y. Chen, and M. Liu, "Tightly coupled 3d lidar inertial odometry and mapping," in *International Conference on Robotics and Automation (ICRA)*, pp. 3144–3150, IEEE, 2019.

[36] K. Li, M. Li, and U. D. Hanebeck, "LiLi OM: Towards High-Performance Solid-State-LiDAR-Inertial Odometry and Mapping," *IEEE Robotics and Automation Letters*, vol. 6, no. 3, pp. 5167–5174, 2020.

[37] P. J. Besl and N. D. McKay, "A method for registration of 3D shapes," in *Sensor fusion IV: control paradigms and data structures*, vol. 1611, International Society for Optics and Photonics, 1992.

[38] F. Pomerleau, F. Colas, R. Siegwart, and S. Magnenat, "Comparing ICP variants on real-world data sets: Open-source library and experimental protocol," *Autonomous Robots*, vol. 34, no. 3, pp. 133–148, 2013.

[39] F. Pomerleau, F. Colas, and R. Siegwart, "A Review of Point Cloud Registration Algorithms for Mobile Robotics," *Foundations and Trends in Robotics*, vol. 4, no. 1, pp. 1–104, 2015.

[40] S. Arya, D. M. Mount, N. S. Netanyahu, R. Silverman, and A. Y. Wu, "An optimal algorithm for approximate nearest neighbor searching in fixed dimensions," *Journal of the ACM (JACM)*, vol. 45, no. 6, pp. 891–923, 1998.

[41] D. Meagher, "Geometric modeling using octree encoding," *Computer graphics and image processing*, vol. 2, no. 19, pp. 129–147, 1982.

[42] F. Boniardi, T. Caselitz, R. Kummerle, and W. Burgard, "Robust LiDAR-based localization in architectural floor plans," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 3318–3324, IEEE, 2017.

[43] 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.

[44] G. Grisetti, C. Stachniss, and W. Burgard, "Improved Techniques for Grid Mapping with Rao-Blackwellized Particle Filters," *IEEE transactions on Robotics*, vol. 23, no. 1, pp. 34–46, 2007.

[45] W. Hess, D. Kohler, H. Rapp, and D. Andor, "Real-time loop closure in 2D LIDAR SLAM," *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1271–1278, 2016.

[46] A. Hornung, K. M. Wurm, M. Bennewitz, C. Stachniss, and W. Burgard, "OctoMap: An efficient probabilistic 3D mapping framework based on octrees," *Autonomous Robots*, vol. 34, no. 3, pp. 189–206, 2013.

[47] A. Rosinol, M. Abate, Y. Chang, and L. Carlone, "Kimera: An open-source library for real-time metric-semantic localization and mapping," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1689–1696, IEEE, 2020.

[48] X. Zuo, Y. Yang, P. Geneva, J. Lv, Y. Liu, G. Huang, and M. Pollefeys, "LIC-Fusion 2.0: LiDAR-inertial-camera odometry with sliding-window plane-feature tracking," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 5112–5119, IEEE, 2020.

[49] N. Gelfland and N. J. Mitra, "Robust Global Registration," in *Eurographics Symposium on Geometry Processing*, vol. 2, p. 5, 2005.

[50] T. Möller and B. Trumbore, "Fast, minimum storage ray/triangle intersection," *ACM SIGGRAPH 2005 Courses, SIGGRAPH 2005*, no. 1, pp. 1–7, 1997.

[51] S. Agarwal and K. Mierle, "Ceres Solver," [Online]. Available: http://ceres-solver.org (accessed: September 01, 2021).

[52] S. Zhao, H. Zhang, P. Wang, L. Nogueira, and S. Scherer, "Super Odometry: IMU-centric LiDAR-Visual-Inertial Estimator for Challenging Environments," *arXiv preprint arXiv:2104.14938*, 2021.

[53] L. R. Muñoz, M. G. Villanueva, and C. G. Suárez, "A tutorial on the total least squares method for fitting a straight line and a plane," *Revista de Ciencia e Ingen. del Institute of Technology, Superior de Coatzacoalcos*, vol. 1, pp. 167–173, 2014.

[54] K. W. Lee, W. S. Wijesoma, and I. G. Javier, "On the observability and observability analysis of SLAM," *IEEE International Conference on Intelligent Robots and Systems*, pp. 3569–3574, 2006.

[55] J. Andrade-Cetto and A. Sanfeliu, "The effects of partial observability in SLAM," in *IEEE International Conference on Robotics and Automation*, no. 1, pp. 397–402, 2004.

[56] W. Xu and F. Zhang, "FAST-LIO: A fast, robust LiDAR-inertial odometry package by tightly-coupled iterated kalman filter," *IEEE Robotics and Automation Letters*, vol. 6, no. 2, pp. 3317–3324, 2021.

[57] N. Chebrolu, T. Labe, O. Vysotska, J. Behley, and C. Stachniss, "Adaptive Robust Kernels for Non-Linear Least Squares Problems," *IEEE Robotics and Automation Letters*, vol. 6, no. 2, pp. 2240–2247, 2021.

[58] H. Nijmeijer and A. J. Van der Schaft, *Nonlinear dynamical control systems*. New York: Springer-verlag, vol. 175 ed., 1990.

[59] T. Barfoot, *State estimation for robotics*. Cambridge University Press, 2021.

[60] S. Choi, J. Park, J. Byun, and W. Yu, "Robust ground plane detection from 3D point clouds," in *14th International Conference on Control, Automation and Systems (ICCAS 2014)*, pp. 1076–1081, Institute of Control, Robotics and Systems (ICROS), 2014.

[61] M. Quigley, B. Gerkey, K. Conley, J. Faust, and T. Foote, "ROS: an open-source Robot Operating System," *ICRA workshop on open source software*, vol. 3, 2009.

Glossary

List of Acronyms

EKF	Extended Kalman Filter
ICP	Iterative Closest Point
IEKF	Iterated Extended Kalman Filter
IMU	Inertial Measurement Unit
KF	Kalman Filter
Lidar	Light Detection and Ranging
MAE	Mean Absolute Error
MHE	Moving Horizon Estimation
Max-AE	Maximum Absolute Errror
NICP	Non-Iterative Closest Point
SE	Special Euclidean Lie group
SLAM	Simultaneous Localization And Mapping
SO	Special Orthogonal group

List of Symbols

α	Laser beam measurement direction
β	Laser beam incidence angle
γ	Map element end point angle
λ	Laser beam heading
ω	Robot angular velocity
σ	Standard deviation
θ	Robot heading
φ	Map element heading
$\mathcal{C}$	Set of correspondences
$\mathcal{M}$	Set of map elements
$\mathcal{Q}$	Set of points
$\mathcal{Y}$	Set of measurements
A	Dynamics matrix
B	Body frame
b	Boolean variable
C	Observation matrix
c	Smoothness characteristic of a point
d	Map element offset parameter
e	Error term
e	Measurement noise
G	Cluster covariance matrix
i	Index value
J	Jacobian matrix
K	Kalman gain matrix
l	Linearity descriptor value
N	Number of elements in a set
n	Map element index value
P	Covariance matrix
p	Cartesian position vector
p	Probability density function
Q	Process noise covariance matrix
q	Cartesian coordinate vector of a point observation
R	Rotation matrix
r	Map element radius
S	SLAM frame
s	Tuning parameter
T	Transformation matrix
v	Velocity vector

W	World frame
w	Process noise
x	Cartesian coordinate in the x-direction
x	State vector
y	Cartesian coordinate in the y-direction
y	Distance measurement
$(\cdot)^{-1}$	Inverse
$(\cdot)^{\top}$	Transpose
$(\cdot)^{\circ}$	Degrees
$\breve{x}$	Prior on x
$\check{x}$	One-step-ahead prediction of x given a transition model
$\hat{x}$	Estimate of x
$\tilde{x}$	Prediction of x given a prediction model
$\mathbb{R}^n$	Set of real numbers in n dimensions
$\mathcal{I}_n$	Identity matrix of size n by n
$0_{n\times m}$	Matrix of zeros of size n by m
$\mathcal{N}(0, P)$	Zero-mean Gaussian noise with covariance P
$\pm$	Plus minus
Σ	Summation
$\sim$	Is distributed as
$\propto$	Is proportional to
sin	Sine
cos	Cosine
$\subseteq$	Is a subset of
$\in$	Is an element of
$\times$	Cross product
$\leftarrow$	Is overwritten by
R^{AB}	Rotation matrix expressing the orientation of frame B in frame A
T^{AB}	Transformation defining frame B measured from and expressed in frame A
$\lvert\cdot\rvert$	Absolute value
$\lVert\cdot\rVert_2$	Magnitude
$\lVert\cdot\rVert_P^2$	Squared Mahalanobis distance
$<$	Less than
$\leq$	Less than or equal to
$=$	Equal to
$\geq$	Larger than or equal to
$>$	Larger than
$\triangleq$	Is defined by