Improving Light Detection and Ranging based Simultaneous Localization and Mapping with advanced Map Representations

Thesis submitted in accordance with the requirements of
the University of Liverpool for the degree of Doctor in Philosophy by

Joscha-David Fossel

June 2018

Contents

Illustrations

List of Figures

List of Tables

Notations

The following abbreviations are found throughout this thesis:

SLAM	simultaneous localization and mapping
LiDAR	light detection and ranging
DoF	degrees of freedom
ICP	iterative closest point
SDF	signed distance function
3 DoF	x, y translation and yaw rotation
6 DoF	x, y, z translation and roll, pitch, yaw rotation
Voxel	cubic volume
UAV	unmanned aerial vehicle
ToF	time of flight
IMU	inertial measurement unit
RGB(D)	red green blue (depth)

If not defined otherwise, math symbols are notated using the following naming convention:

`Typewriter`	vector
Bold	matrix
x	horizontal translation in direction of the entity, i.e. forward/backward
y	horizontal translation sideways to the entity, i.e. left/right
z	vertical translation, i.e. up/down
α	roll, i.e. rotation around x axis
β	pitch, i.e. rotation around y axis
γ	yaw, i.e. rotation around z axis
$\mathbf{S}$	pose matrix, i.e. rotation and translation
S	pose set, i.e. rotation and translation
$\mathtt{s}$	pose vector, i.e. rotation and translation
$\mathbf{R}$	rotation matrix
$\mathbf{T}$	translation matrix
$\mathtt{t}$	translation vector
M	reference, model set; “map”
D	source, data set; “sensor measurements”
E	error

Preface

This thesis is primarily my own work. The sources of other materials are identifed.

Abstract

In this thesis we investigate simultaneous localization and mapping (SLAM) with light detection and ranging (LiDAR) based sensors. In particular, we propose three novel SLAM algorithms, *OctoSLAM*, *2D-SDF-SLAM*, and *NOctoSLAM*, that leverage advanced map representations to advance SLAM performance.

OctoSLAM is designed for airborne 2D LiDAR SLAM and produces 3D maps of the environment that are also used for localization. We use the multi-resolution capability of octree based maps to interpolate map gradients for efficient Gauss-Newton minimization based registration. To cope with the mismatched 2D sensor and 3D map dimensions we employ a heuristic that assumes limited similarity of objects over height. In a series of experiments we empirically show that using a 3D map yields significant improvements over traditional 2D map approaches in environments that feature elements with a relatively inhomogeneous structure along height, e.g. tilted walls.

2D-SDF-SLAM is a signed distance function (SDF) based 2D map SLAM approach for ground based robots. The SDF map representation allows us to capture sub grid cell sized details of the environment, and to accurately interpolate map values and gradients. Consequently, efficient Gauss-Newton minimization is used for registration. We empirically evaluate 2D-SDF-SLAM and compare it to Hector SLAM, a SLAM system that uses a similar registration technique but a different map representation. The experiments show that 2D-SDF-SLAM achieves a better performance than Hector SLAM.

NOctoSLAM is a computationally superior alternative to point cloud based SLAM with 3D LiDAR sensors mounted on airborne robots. We propose an octree based map representation that uses the tree structure to efficiently approximate map surface normals and to perform correspondence search, both required for 6 DoF point-to-plane registration. We empirically evaluate our approach against a comparable point cloud based approach, the ETHZ ICP Mapping tool. Our experiments show that *NOctoSLAM* significantly outperforms the ETHZ ICP Mapping tool computationally, without sacrificing accuracy. If the near real-time constraint required for online SLAM front-ends is enforced, *NOctoSLAM* also significantly outperforms the ETHZ ICP Mapping tool in terms of accuracy.

Acknowledgements

First and foremost, I would like to express my sincere gratitude to my supervisor Karl Tuyls. Thank you Karl for your advice and guidance, both professionally and personally. I greatly appreciate you having my back and the opportunities you provided.
Second, I would like to thank my advisors Michael Fisher and Wiebe van der Hoek for providing feedback and giving helpful suggestions to improve my research over the last years.
I would also like to thank Clare Dixon and Guido de Croon for evaluating this work.

Many thanks as well to Daniel Claes and Bastian Broecker, it was a pleasure studying, living, traveling and working with you for nearly a decade. Hennes, among many things, thank you for introducing me to Karl all those years ago, I wouldn't be here, doing this, otherwise. To my other friends I made during my PhD studies, thank you for your part in keeping my sanity moderately intact: Daan, Frans, Haitham, Michael, Sjriek, Benjamin, David, Eric, Elisa, Ipek, Matula, Maryam, Paul, Richard$\times$2 and Sven. Special thanks to my former housemates Tobi, Phil and Lukas: living with you probably added a year to the duration of my studies and had the inverse effect on life expectancy, but I would do it all again. To my friends I grew up with, Jens, Steve, Sebastian, Kristian, Sven, Henrik, Markus, and Matthias: thank you for always being there even after intermittent radio silence from my end.

Last but not least, my parents, Thi-Dong and Siggi. You did not only give me leeway to follow my passions, but always encouraged and supported my endeavors without exceptions. You are incredible, thank you!

P.S. these paragraphs are inconclusive, and there are more people and more things you guys did that I'm grateful for. However, I'd rather express my gratitude in person over a beer or two than here, so bear with me.

Chapter 1

Introduction

In this chapter we introduce the subject of this thesis, i.e. light detection and ranging (LiDAR) based simultaneous localization and mapping (SLAM) for both ground based and airborne robots. An impression of a LiDAR based SLAM output is given in Figure 1.1. We describe the context of the SLAM problem, and state problems with traditional approaches. This is followed by posing research questions corresponding to the problem statement. We then proceed to give a summary of related work, and an outline for the rest of the thesis.

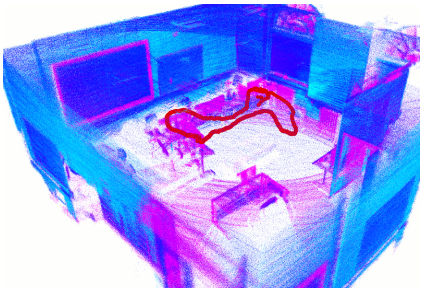


FIGURE 1.1: 3D map SLAM example. Red represents the sensor trajectory.

1.1 Context

A prerequisite for many useful applications in robotics is the ability for a robot to autonomously create a map of the environment and to reliably and accurately localize itself in it. This allows the robot to navigate in previously unknown environments, and is the key for efficient path planning and complex behaviors such as autonomous exploration. Thrun et al. state that "the SLAM problem is generally regarded as one of the most important problems in the pursuit of building truly autonomous mobile robots" [1].

Simultaneous localization and mapping is a challenging problem because of the mutual dependency between the quality of the map and the pose estimation accuracy: an inaccurate pose estimate leads to an imprecise map, and vice versa. If for example, objects are mistakenly added to the map due to sensor noise, the robot might subsequently align itself with the non existent object, deteriorating the pose estimate even further. Such a cascading effect can also be triggered by a misfitting or overly simplified map representation that makes it impossible to get an accurate localization, possibly leading to catastrophic failure. Increasing difficulty is the fact that if deployed on airborne robots, such as multirotor unmanned aerial vehicles (UAVs), the SLAM algorithm should run in near real-time, since the pose estimate might be required to for example keep the robot from drifting into objects.

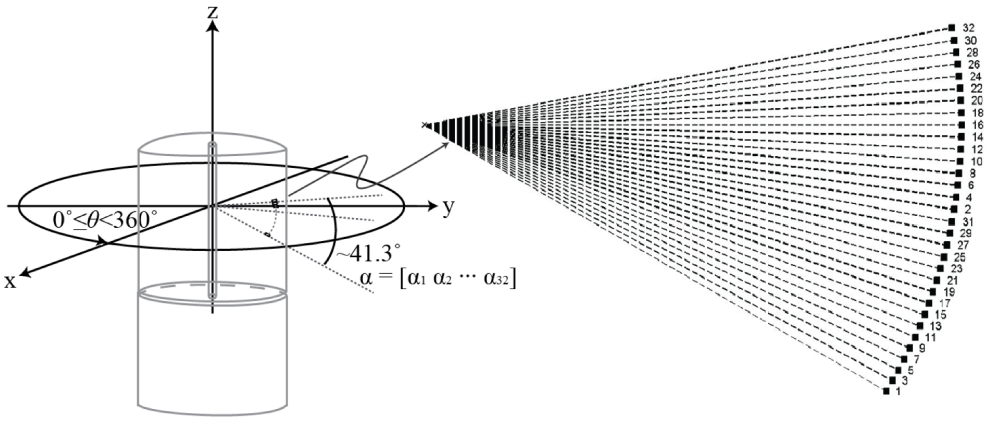


FIGURE 1.2: This drawing shows a mechanical multi-line 3D LiDAR sensor (Velodyne HDL-32E) that features 32 emitters positioned at angular offsets spread over 41.3°, which spin around the z axis resulting in 3D range measurements along multiple planes [8].

Localization in SLAM can be tackled by scan registration/matching, i.e. by aligning scans provided by sensors with a map [2, 3]. While typically used scan registration algorithms, e.g. [2, 4–6] are not guaranteed to converge to the global minimum error, they have been shown to work sufficiently well in practice.

In this dissertation we investigate how using organized map representations benefits light detection and ranging (LiDAR) based simultaneous localization and mapping. Our work focuses on SLAM front-ends, i.e. providing pose estimates and updating the map in near real-time. In contrast, SLAM back-ends perform a posteriori optimization, e.g. loop closure triggered pose graph optimization [7]. LiDAR sensors illuminate objects with pulsed laser light and measure the reflection, using the lights time of flight (ToF) to calculate the distance to the object. Mechanical 2D LiDAR sensors designed for mobile robotics use a rotating mirror to reflect the emitted light, resulting in planar distance measurements. Because of the high refresh rate of the sensor, planar distance measurements can be achieved at ample frequencies, typically around 10Hz to 40Hz. So called 'multi-line' 3D LiDAR sensors use multiple spinning emitters to measure vertically offset points, thus producing measurements along multiple tilted planes, as illustrated in Figure 1.2 [8]. We investigate the three sensor, localization and map dimension permutations listed in Table 1.1, i.e. combinations of 2D or 3D mapping with three or six degrees of freedom (DoF) localization. The 6 DoF translations and rotations are shown in Figure 1.3. We define 3 DoF as horizontal translation along the x and y axis (left/right, forward/back), and rotation around the z axis (yaw). In case of 6 DoF localization, vertical translation along the z axis (up/down), as well as rotation around the x axis (roll) and y axis (pitch) are added.

OctoSLAM tackles 3 DoF registration and 3D mapping on a 6 DoF motion performing robot equipped with a 2D LiDAR sensor. Traditionally, non-actuated 2D LiDAR sensor based SLAM approaches [2, 11, 12] are restricted to 2D maps. However, 3D mapping with 2D LiDAR sensors can be achieved by using panning [13] or spring-mounted [9] sensors in combination with industrial-grade inertial measurement units (IMU). Nevertheless, such specialized mounts may not be an option on certain robots, e.g. on weight restricted UAVs. OctoSLAM uses a

TABLE 1.1: SLAM approaches for different LiDAR dimensions, mapping dimensions, and localization degrees of freedom. Bold font marks algorithms introduced in this dissertation.

		Registration	
		3 DoF	6 DoF
Mapping	2D	**2D-SDF-SLAM**	/
		Hector SLAM [2]	/
	3D	**OctoSLAM**	**NOctoSLAM**
		zebedee [9]	libpointmatcher [10]
		2D	3D
		LiDAR sensor	

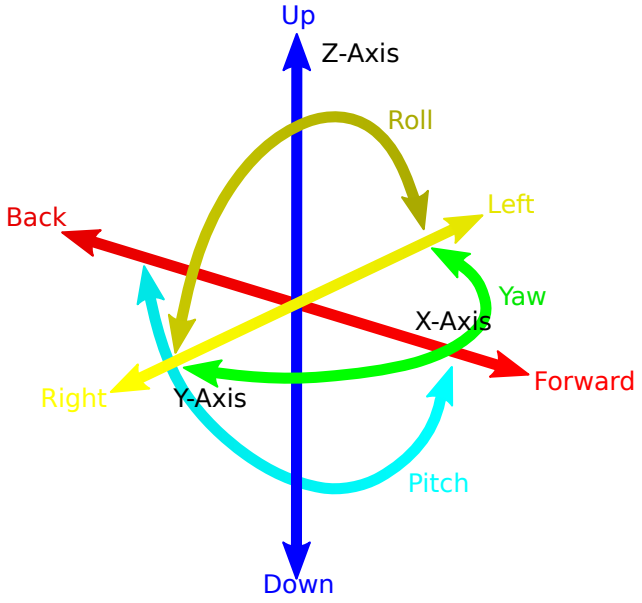
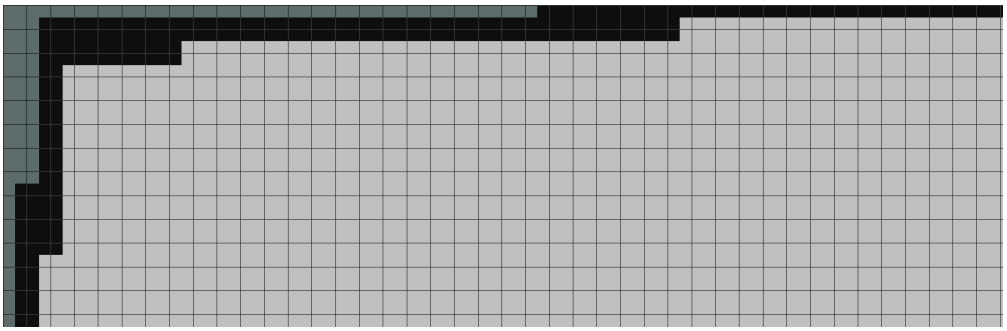
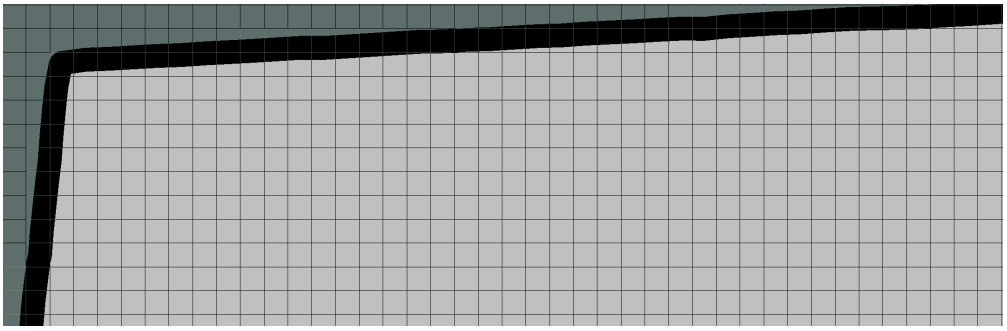


FIGURE 1.3: This figure shows the six degrees of freedom for motion in 3D space and the corresponding naming convention used throughout the thesis.

consumer-grade IMU to determine the roll and pitch angle of the 2D LiDAR sensor, and a down-facing range sensor or an atmospheric pressure sensor to determine the altitude of the robot. To represent the environment, an octree based occupancy map is used. OctoSLAM utilizes the octree parent-child relation to approximate map gradients for localization. Generally speaking, traditional simultaneous 3 DoF localization and 2D mapping approaches are not well suited if the robot performs significant 6 DoF motion, e.g. in a multirotor UAV scenario. While 2D map SLAM approaches [14, 15] work if the environment is homogeneous along the z axis, they may become infeasible if this is not the case. The combination of small overlap between LiDAR readings and an oversimplifying map representation (3D environment to 2D map) decreases localization accuracy and can lead to catastrophic failure. Moreover, 2D maps are unable to



(A) Grid cell maps discretize the environment.



(B) Signed distance function based map can capture details below cell size.

FIGURE 1.4: This figure shows the same environment being represented by an occupancy grid cell map (A) and a signed distance function based map (B). While both use the same underlying grid cell size, the occupancy grid cell map is grainy, while the signed distance function map produces smooth walls.

capture certain environmental details that might be mission critical, for instance underpasses that could be traversed. We empirically show that 3D map OctoSLAM can deal with inhomogeneous environments to a certain degree, outperforming a 2D map but otherwise comparable SLAM algorithm, thus enabling SLAM on 6 DoF motion performing robots with less powerful sensors.

For 2D LiDAR sensor equipped ground based robots performing only 3 DoF movement, 2D map SLAM algorithms [2, 11, 12] are well suited. Such algorithms typically represent the environment as a 2D occupancy grid cell map [16]. As the name suggests, such a representation is limited to storing occupancy in cells, and thus discretizes the environment as shown in Figure 1.4a. We propose 2D-SDF-SLAM, a SLAM algorithm that uses a signed distance function (SDF) based map [17], instead. By storing the distance to the next object in a cell instead of occupancy, objects can be captured with sub-grid cell size precision (Figure 1.4b). Additionally, SDFs are differentiable over large parts of the map, enabling efficient map gradient approximation, and thus allowing for efficient minimization techniques, e.g. Gauss-Newton, to be used for updating the pose estimate. We empirically show that by using a superior map representation, 2D-SDF-SLAM significantly outperforms its occupancy grid cell counterpart Hector SLAM [2] in terms of localization accuracy.

The third scenario we are considering are airborne 6 DoF motion performing robots equipped with a 3D LiDAR sensor. 3D LiDAR sensors allow for 6 DoF localization because measurements are distributed over all three axes. A popular approach to represent 3D environments are

point clouds, i.e. basically storing the measurements in a list. However, because of the high number of points per second provided by 3D LiDAR sensors, point clouds require significant data point filtering in order to adhere to the near real-time constraint. With NOctoSLAM, we devise an octree based map representation that uses the parent-child relation for approximate nearest neighbor search (NNS) on points contained in the map, in order to estimate surface normal vectors (SFN). Map SFNs allow for point-to-plane scan registration, which is well suited for multi-line LiDAR sensors which only produce sparse readings over the vertical axis. The NOctoSLAM map representation also implicitly provides source to reference matching, i.e. pairing newly acquired measurements with the map. We compare NOctoSLAM to a point cloud based point-to-plane SLAM algorithm and empirically show that NOctoSLAM is superior in terms of computational efficiency, while achieving similar accuracy. This discards the necessity for filters, which might be challenging to configure, and which might depend on the type of environment in which the robot is deployed. If filtering is enabled to allow for near real-time performance of the point cloud based SLAM approach, NOctoSLAM achieves higher accuracy in comparison.

1.2 Problem Statement

Simultaneous localization and mapping is fundamental for autonomous robotics. While extensive research on SLAM over the last two decades has already yielded promising results for both 3 DoF [2, 3, 18–20] and 6 DoF SLAM [21–23], new application scenarios such as (multirotor) UAVs, autonomous cars, and improved sensors require further advancements in the field. Accurate, reliable SLAM that runs in near real-time for both 3 DoF and 6 DoF motion performing robots is required. Thus, the following problems are investigated.

- Regarding 2D maps used with 6 DoF motion performing robots, e. g. multirotor UAVs equipped with 2D LiDAR sensors, the following two problems arise:

  (i) 2D map SLAM **does not perform well in environments that are not mostly homogeneous over height,** ranging from bad accuracy to catastrophic failure.

  (ii) 2D maps **omit object height**, so for instance obstacles with low height that UAVs might be able to traverse, become defacto impassable.

- While 2D occupancy grid cell maps are desirable from a computational viewpoint, and have been shown to work well for 2D map SLAM, they inherently lack features beneficial to SLAM algorithms.

  (i) **Environmental details below grid cell size precision cannot be captured** by occupancy grid approaches, possibly limiting the maximum achievable localization accuracy.

  (ii) Occupancy grid cells maps do **neither natively provide nearest neighbor search**, which is required for ICP, **nor map gradients** for Gauss-Newton minimization for registration.

- While the point cloud map representation is commonly used for simultaneous 6DoF localization and 3D mapping, it introduces several problems.

  (i) Each sweep, modern 3D LiDAR sensors provide a **large amount of measurements, overtaxing the capabilities of point cloud based SLAM algorithms**. In particular, to adhere to the near real-time constraint, **filtering out relatively large portions of the acquired data is required**. Also, configuring appropriate **filters requires expert knowledge as well as information about the environment**, and can be **computationally expensive**.

  (ii) Operations required for point-to-plane ICP, i.e. surface normal approximation and source to reference matching, introduce an overhead, as **additional data structures need to be maintained**.

1.3 Research Questions

In regards to the problems stated in the previous section, we issue the following research questions:

- How is 3D map SLAM achievable on a multirotor UAV equipped with a rigidly mounted 2D LiDAR sensor?
- To what extent does 2D LiDAR sensor based 3D map SLAM outperform its 2D map counterpart in terms of accuracy?
- How can we adopt SDF based 2D mapping for 2D LiDAR based SLAM, that uses Gauss-Newton minimization via map gradients for registration?
- To what extent can we improve 2D LiDAR sensor based 2D map SLAM accuracy by using a SDF instead of an occupancy grid cell based map?
- How can we develop a 3D map representation suitable to point-to-plane scan registration (i.e. supporting SFN approximation and NN search), increasing the computational performance whilst maintaining accuracy?

1.4 Related Work

Extensive research has been committed to solving the SLAM problem since it was introduced to the field of robotics in 1986 by Cheeseman et al. [24]. From that time on, a plethora of approaches have emerged that use various types and combinations of sensors for underwater [25–27], ground-based [21, 28, 29] and airborne [15, 30, 31] SLAM. Sensor technologies used for SLAM include sonar, infrared, radar, laser, (depth) vision (RGB(D) cameras), wheel encoder, inertial measurements (accelerometer, gyroscopes), and GNSS (GPS).

SLAM approaches consist of a front-end and optionally a back-end, in combination with a mapping component that can be attached to front-end or back-end. The front-end updates

the pose estimate of the robot online and in near real-time, while the back-end continuously improves the accuracy of these pose estimates. For example, lets assume a robot equipped with wheel encoders and a 2D LiDAR sensor. The revolutions of the wheels can be used to determine the distance traveled, and is used as the front-end. Based on the information provided by the front-end, the robot now drives in a perfect circle and should thus arrive in the exact starting position. However, due to encoder noise or wheel slippage, the robot actually ends up in the vicinity of the starting position, not on it. Using the LiDAR sensor, the back-end now recognizes that this is the case and corrects the pose estimate accordingly.

A popular way to tackle mapping are probabilistic approaches, first realized in 1985 by Moravec & Elfes [32]: occupancy grid cell maps. As the name suggest, such maps divide the environment in grid cells, where each cell stores its probability of being occupied. Analogously, that principle can be transferred to 3D [33]. However, while negligible in the 2D case, the low memory efficiency of this approach restricts 3D occupancy grids to small environments or coarse map resolutions.

An alternative to 3D occupancy grids are given by Herbert et al. in form of elevation maps [34], where each cell in a 2D occupancy grid also stores height. Such a 2.5D approach can be sufficient for ground vehicles, e.g. if they are allowed to traverse obstacles that are below a certain height. However, possibly traversable environmental features such as windows, underpasses, or overhangs cannot be captured appropriately by this approach. Pfaff et al. [35, 36] improve the 2.5D approach to accommodate overhanging objects by classifying them as such in the map. Another way to alleviate the shortcomings of elevation maps is presented by Triebel et al. with Multi-Level Surface Maps [37], where each cell can store multiple surfaces. To enable capturing free space, Dryanovski et al. propse Multi-Volume Occupancy Grids [38], which are similar to Multi-Level Surface Maps with the addition that each cell can not only store multiple surfaces, but also multiple free spaces.

First proposed by Meagher [39], and later optimized by Wurm et al. [40], is the use of octrees for mapping. In this approach, each octree node represents a cubic volume of equal length, width and depth (voxel), and is divided into 8 smaller voxels by its child nodes. As nodes do not need to be pre-initialized, this approach is more efficient than grid based methods.

A non-probabilistic, but also non-discrete map representation are point clouds, i.e. direct storage of coordinates. The downside of this approach is the low memory efficiency, the inability to represent free and unknown space, and the lack of geometric information required for e.g. path planning and obstacle detection. To deal with the low memory efficiency in practice, it is common to use filters, e.g. a density filter where points are only added to the map if the distance to the nearest neighbor is larger than a certain threshold.

In contrast, signed distance function based maps alleviate discretization to a certain degree by storing the minimum distance to objects in cells/voxels. This way, sub grid resolution sized details can be captured while still preserving memory efficiency. While the idea of using SDFs for range data has already been proposed in 1996 by Curless & Levoy [17], they have only become popular with the introduction of KinectFusion [41] in 2011.

If powerful back-ends are used, the front-end can be quite simple, e.g. odometry generated by wheel encoders. Another popular front-end option is the iterative closest point (ICP) algorithm [4], proposed by Besl & McKay in 1992. ICP uses nearest neighbor search to find correspondence between points from two different data sets, called source and reference. These data sets could be successive 2D LiDAR range measurements or pictures taken by a camera. Corresponding source and reference points are then evaluated via a point-to-point error metric. By minimizing said error a pose update is generated. However, both wheel odometry and basic ICP suffer severely from sensor noise, and thus cannot in practice be used for SLAM on their own. Nevertheless, many modern SLAM system rely on improved variants of the original ICP algorithm.

Chen & Medioni [42] introduce an ICP variant for 3D modeling that uses a point-to-line/plane error metric instead the original point-to-point error metric. By registering source points with the line (2D case) or plane (3D case) connecting adjacent reference points, error induced by low sensor resolution is reduced. Censi [6] refines the point-to-line ICP variant by linearizing the registration minimization equation and solving it via linear least-squares. Similarly, Low [43] provides a linear least-squares approximation to the traditionally used nonlinear least-squares method that minimizes the point-to-plane error. This approximation works under the assumption that the difference in relative orientation between source and reference is small. With generalized-ICP [44], Segal et al. extend the idea of point-to-plane ICP by modeling planar surfaces not only in the reference, but also in the source data set, therefore introducing a plane-to-plane ICP variant.

Diosis & Kleemans Polar Scan Matcher [5] aims to improve ICP by using polar coordinates to match points with the same bearing, thus performing faster correspondence search.

SPM-ICP [3], proposed by Holz & Behnke, is a 2D LiDAR 2D map SLAM approach that alleviates the sensor drift issue of ICP by performing incremental scan registration. Instead of registering successive scans with each other, SPM-ICP merges registered scans into a single data set, effectively creating a map of the environment, which is used as reference for future registration. To allow for up to medium-sized indoor environments, Sparse Point Maps (SPMs) are used. In SPMs, less probable measurements are removed according to a probabilistic grid map.

With Hector SLAM [2], Kohlbrecher et al. present an alternative to ICP based 2D map SLAM with 2D LiDAR sensors. They use a Gauss-Newton based scan registration algorithm, which is a variation of the stereo image registration technique proposed in [45]. In Hector SLAM, beam end points are aligned using approximated map gradients. As no correspondence search is performed in this approach, it requires lower computational resources than ICP-based alternatives. However, for good results, their approach requires maintaining multiple grid map versions in different resolutions.

Early solutions to 3D mapping on ground based vehicles use multiple sensors, for example a horizontally mounted 2D LiDAR sensor used for SLAM, and a vertical 2D LiDAR sensor for 3D mapping [46–48]. Another approach is to use panning 2D LiDAR sensors to create 3D scans, and using these for 3 DoF [49] or 6 DoF localization [21, 50, 51]. However, panning

2D sensors are slow, and to achieve a consistent 3D scan they require either correction via e.g. wheel odometry, or a "stop-and-scan" approach.

Bosse et al. tackle 2D LiDAR 3D mapping 6 DoF localization SLAM with zebedee [9]. Here, a 2D LiDAR sensor and an industrial-grade IMU are mounted on a spring, therefore extending the sensors field of view to additional scanning planes. In contrast to other panning 2D LiDAR 3D map SLAM approaches, localization is performed continuously, and not with consistent assembled 3D scans.

Pomerleau et al. devise libpointmatcher [10], a modular library supporting different 3 DoF and 6 DoF ICP variations intended for direct comparison of various registration and error measurement approaches. For fast nearest neighbor search, it makes use of libnabo [52], a optimized kd-tree implementation. It furthermore features various finely tunable pre- and post-processing point filtering techniques, reducing the amount of data processed in order to lower computation costs, allowing libpointmatcher to be used with modern 3D LiDAR sensors. libpointmatcher has been deployed in various scenarios, e.g. on boats and on ground based vehicles in both urban environments and in uneven terrain.

An alternative way to address the high computational costs of handling large 3D point clouds returned by 3D LiDARs is presented by Zhang & Singh with LOAM [23]. LOAM effectively separates the 6 DoF localization from the mapping operation, performing them at different rates with different levels of accuracy. Pose estimation is performed at high frequency, but with low fidelity, while mapping is done less often but investing a higher amount of computational resources to find a more precise result. In order to achieve good pose estimates despite low fidelity, LOAM uses intricate feature detection algorithms to preserve only meaningful data points.

In KinectFusion [41], Newcombe et al. use a SDF based map representation for 6 DoF localization and 3D mapping, achieving high accuracy in near real-time. Originally designed for the use with the Microsoft Kinect RGBD camera, the Kinect Fusion approach is generalized for various sensors in [53]. While [41, 53] rely on ICP for camera tracking, later work showed that SDFs can be employed for direct camera tracking [54, 55].

Popular back-end techniques are graph-based optimization and particle filtering. Particle filter back-ends are probabilistic approaches that work well in low-dimensional spaces, where the number of required particles is manageable. The idea is that multiple particles are maintained simultaneously, where each particle represents a hypothesis of the state. These particles are generated and updated according to a vehicle model, which is applied to the odometry provided by the front-end. To tackle the "kidnapped robot" problem, i.e. creating localization uncertainty by carrying the robot to a different localization, particles can be spawned randomly. Using sensor data, particles are evaluated and resampled: unfit ones are replaced by more fit ones, thus reducing the error in localization and mapping. FastSLAM by Montermerlo et al. [12, 19] and GMapping by Grisetti et al. [11, 18] are examples for range sensor 2D map 3 DoF localization SLAM approaches that successfully apply Rao-Blackwellized particle filters [56] to increase localization and mapping performance. Particle filters are also applied on vision based systems, e.g. by Sim et al. [57], or in conjunction with 3D maps generated by a panning 2D LiDAR

by Welle et al. [58]. Törnqvist et al. [59] introduce a variant that enables higher dimensional vehicle models, in particular a helicopter UAV.

In graph optimization back-ends, a graph consisting of poses and corresponding map segments is optimized, where pose estimates are provided by a front-end, e.g. wheel encoders or ICP. Constraints between poses, e.g. given by revisiting a position after performing a loop, are resolved and thus the mapping error is reduced. Graph optimization can be seen as sparse spring-mass system, where masses represent poses and springs represent transformations between poses. The first working graph-based SLAM algorithm has been developed by Lu & Milios [60] in 1997. Since then, graph optimization has been applied on various systems, e.g: feature-based monocular SLAM [61]; feature-based RGBD SLAM [62]; semi-dense monocular SLAM [31]; semi-dense RGBD SLAM [63]; 2D LiDAR 2D map SLAM [64]; and panning 2D LiDAR 3D map large-scale environments SLAM [65]. Many of these approaches ([31, 61–64]) use g2o [66], an open-source framework for optimizing graph-based nonlinear error functions, developed by Kümmerle et al.

For among others SLAM implementations on actual robots, the robot operating system (ROS) [67] serves as a widely used powerful framework. ROS is a middle-ware that provides libraries and tools to help software developers create robot applications. It provides hardware abstraction, device drivers, libraries, visualizers, message-passing, package management, and more. It is fully open source and designed to support code reuse. ROS consists of independent self contained nodes which communicate via a one-to-many model implemented through the TCP/IP protocol. This allows easy implementation of distributed systems.

For further reading, Thrun & Leonard give a more thorough introduction to particle filters and graph optimization in [1], and an extensive review of point cloud registration algorithms is provided by Pomerleau et al. in [68].

1.5 Thesis Outline

In Chapter 2 we summarize research from other authors that is fundamental to our work. In particular, we discuss 3 DoF and 6 DoF registration, as well as 2D and 3D mapping techniques.

In Chapter 3 we present OctoSLAM, a SLAM system designed for multirotor UAVs equipped with 2D LiDAR sensors. OctoSLAM combines measurements from auxiliary sensors to achieve 3 DoF registration with 3D mapping in near real-time. In contrast to other 2D LiDAR 3D mapping approaches, neither multiple LiDAR, nor panning sensors that provide periodic sweeps, are required.

Afterwards, in Chapter 4, we investigate improving 2D LiDAR based simultaneous 3 DoF localization and 2D mapping by using SDF based maps instead of occupancy grid cell maps. Therefore, we are able to capture environmental details that would have been discarded otherwise. We develop a novel map update scheme that deals with problematic steep incident angles that are likely to occur when updating SDF based maps with wide angle 2D LiDAR measurements. Instead of using ICP based registration, we leverage the map gradients provided by SDF based maps for efficient Gauss-Newton minimization.

Chapter 5 explores 3D LiDAR 6 DoF registration SLAM using point-to-plane ICP. We develop an octree based map representation that caters to the registration algorithm, making commonly used additional auxiliary data structures required for correspondence search and SFN approximation obsolete. Alongside being more efficient than the standard point cloud based representation used for point-to-plane ICP, octree based maps also feature geometric information that could be used for e.g. obstacle avoidance.

Finally, we revisit the problem statement and research questions asked to draw conclusions in Chapter 6. Here, we also discuss future work and provide critical notes for the introduced algorithms.

1.6 List of Contributions

- Telepresence Robots as a Research Platform for AI [69]
  Spring Symposium: Designing Intelligent Robots, AAAI

- OctoSLAM: a 3D Mapping Approach to Situational Awareness of Unmanned Aerial Vehicles (Demonstration) [70]
  International Conference on Adaptive Agents and Multiagent Systems (AAMAS), ACM
  (Used in Chapter 3)

- Development of an Autonomous RC-car [71]
  International Conference on Intelligent Robotics and Applications (ICIRA), Springer

- OctoSLAM: a 3D Mapping Approach to Situational Awareness of Unmanned Aerial Vehicles [72]
  International Conference on Unmanned Aircraft Systems (ICUAS), IEEE
  (Used in Chapter 3)

- How to win robocup @work? The Swarmlab@Work Approach Revealed [73]
  Lecture Notes in Computer Science (LNCS), Springer

- Applied Robotics: Precision Placement in RoboCup@Work [74]
  International conference on Autonomous agents and multi-agent systems (AAMAS), ACM

- smARTLab at work Championpaper RoboCup 2014 Joao Pessoa [75]
  Lecture Notes in Computer Science (LNCS), Springer

- 2D-SDF-SLAM: a signed distance function based SLAM front-end for laser scanners [76]
  Intelligent Robots and Systems (IROS), IEEE
  (Used in Chapter 4)

- NOctoSLAM: fast octree surface normal mapping and registration [77]
  Intelligent Robots and Systems (IROS), IEEE
  (Used in Chapter 5)

Chapter 2

Background

In this chapter we present related work that our research builds upon. We go into more detail than in Section 1.4, and also provide references for further reading. The chapter is divided into two sections, the first discussing scan registration techniques, and the second map representations.

2.1 Registration Algorithms

In this section we give a summary of the iterative closest point algorithm in the original point-to-point correspondence variant. We also present the associated closed-form solution for the 3 DoF ICP registration problem. This is followed by examining 6 DoF ICP registration using the point-to-plane error metric, for which currently no closed-form solution exists. We instead provide a linear approximation that works sufficiently well in practice. Finally, we introduce Gauss-Newton minimization based registration as performed in Hector SLAM, which offers a computationally superior alternative to ICP based registration.

2.1.1 Point-to-Point Iterative Closest Point Registration

The iterative closest point (ICP) algorithm [4] registers two data sets with each other, i.e. it finds a rigid-body transformation that maximizes the overlap of the two sets. In its original form, the ICP algorithm consists of three steps that are repeated until a certain termination criterion is met. Typical termination criteria are a maximum number of iterations, a sufficiently small error, or convergence. In this subsection we present the closed-form solution as proposed for 6 DoF in [78], and as applied to 3 DoF in [3]. The three ICP steps are as follows:

1. Given two data sets M (reference, model set; "map") and D (source, data set; "sensor input"), determine the correspondence of the two data sets.

   Correspondence of $M = (\mathtt{m}_0, \mathtt{m}_1, .., \mathtt{m}_{|M|})$ and $D = (\mathtt{d}_0, \mathtt{d}_1, .., \mathtt{d}_{|D|})$ can be determined by the lowest distance of every element in D to an element in M. Commonly used are the point-to-point and point-to-line metric in combination with Euclidean distance.

2. Determine the error E according to the correspondence between M and D.

To do so, we use the point-to-point error and the Euclidean distance between corresponding points.

3. Minimize E by updating rigid transformation $S = (\mathbf{R}(\gamma), \mathtt{t})$, in this case consisting of rotation around the vertical axis and horizontal translations. Then repeat from the first step on, or stop if termination criteria is met.

For 3 DoF registration, a rigid transformation is defined as $S = (\mathbf{R}(\gamma), \mathtt{t})$, where $\mathtt{t} = (t^x, t^y)^T$ and t_x, t_y describe forward/backward and left/right movement, i.e. the translation along the x and y axis, respectively. $\mathbf{R}(\gamma)$ is the rotation matrix for yaw γ, i.e. rotation around the z axis, and is defined as follows:

$$\mathbf{R}(\gamma) = \begin{pmatrix} \cos\gamma & -\sin\gamma \\ \sin\gamma & \cos\gamma \end{pmatrix}. \tag{2.1}$$

For ease of notation we define $\mathbf{R} = \mathbf{R}(\gamma)$ and $\Delta\mathbf{R} = \mathbf{R}(\Delta\gamma)$, where Δ annotates the desired transformation.
If the point-to-point error metric with Euclidean distance is used, error E is defined as follows:

$$E(\mathbf{R}, \mathtt{t}) = \sum_{i}^{|D|} \sum_{j}^{|M|} w_{ij} \cdot (\|\mathtt{m}_j - (\mathbf{R} \cdot \mathtt{d}_i + \mathtt{t})\|)^2, \tag{2.2}$$

where w_{ij} is 1 if point $\mathtt{m}_j$ corresponds to point $\mathtt{d}_i$, i.e. if $\mathtt{m}_j$ is the closet point in M to $\mathtt{d}_i$, and 0 otherwise.
Given this set of K pairs of corresponding points $\{(\check{\mathtt{d}}_k, \check{\mathtt{m}}_k)\}$, E is minimized by the transformation ΔS consisting of optimal rotation $\Delta\gamma$ and optimal translation $\Delta\mathtt{t}$:

$$\Delta S = (\mathbf{R}(\Delta\gamma), \Delta\mathtt{t}) = \underset{(\Delta\mathbf{R}, \Delta\mathtt{t})}{\arg\min} \sum_{k=1}^{K} (\|\check{\mathtt{m}}_k - (\Delta\mathbf{R} \cdot \check{\mathtt{d}}_k + \Delta\mathtt{t})\|)^2. \tag{2.3}$$

Obtaining translation $\Delta\mathtt{t}$ and rotation $\Delta\mathbf{R}$ can be decoupled from each other by considering corresponding points from model set M and data set D translated by their centroids

$$\check{\mathtt{c}}_M = \frac{\sum_{i=1}^{K} \check{\mathtt{m}}_i}{K}, \check{\mathtt{c}}_D = \frac{\sum_{i=1}^{K} \check{\mathtt{d}}_i}{K}. \tag{2.4}$$

Assuming that the translated sets $\hat{M} = \{\hat{\mathtt{m}}_i \mid \hat{\mathtt{m}}_i = \check{\mathtt{m}}_i - \check{\mathtt{c}}_m\}$ and $\hat{D} = \{\hat{\mathtt{d}}_i \mid \hat{\mathtt{d}}_i = \check{\mathtt{d}}_i - \check{\mathtt{c}}_d\}$ describe the same environment (to a sufficiently high degree), Equation 2.2 can be reformulated into

$$E(\Delta\mathbf{R}, \Delta\mathtt{t}) = \sum_{i=1}^{K} (\|\hat{\mathtt{m}}_i - \Delta\mathbf{R} \cdot \hat{\mathtt{d}}_i - (\Delta\mathtt{t} - \check{\mathtt{c}}_M + \Delta\mathbf{R} \cdot \check{\mathtt{c}}_D)\|)^2 \tag{2.5a}$$

$$= \sum_{i=1}^{K} (\|\hat{\mathtt{m}}_i - \Delta\mathbf{R} \cdot \hat{\mathtt{d}}_i\|)^2 \tag{2.5b}$$

$$- 2(\Delta\mathtt{t} - \check{\mathtt{c}}_M + \Delta\mathbf{R} \cdot \check{\mathtt{c}}_D) \cdot \sum_{i=1}^{K} (\hat{\mathtt{m}}_i - \Delta\mathbf{R} \cdot \hat{\mathtt{d}}_i) \tag{2.5c}$$

$$+ \sum_{i=1}^{K} (\|(\Delta\mathtt{t} - \check{\mathtt{c}}_M + \Delta\mathbf{R} \cdot \check{\mathtt{c}}_D)\|)^2. \tag{2.5d}$$

Because Term 2.5d is minimal for $\Delta\mathtt{t} - \check{\mathtt{c}}_M + \Delta\mathbf{R} \cdot \check{\mathtt{c}}_D = 0$, the optimal translation is:

$$\Delta\mathtt{t} = \check{\mathtt{c}}_m - \Delta\mathbf{R} \cdot \check{\mathtt{c}}_d. \tag{2.6}$$

Term 2.5c is zero because per definition of centroids $\sum_{i=1}^{K} \hat{\mathtt{m}}_i = 0$ and $\sum_{i=1}^{K} \hat{\mathtt{d}}_i = 0$:

$$\sum_{i}^{K} \hat{\mathtt{d}}_i = \sum_{i=0}^{K} (\check{\mathtt{d}}_i - \check{\mathtt{c}}_d) \tag{2.7a}$$

$$= \sum_{i=0}^{K} \check{\mathtt{d}}_i - \sum_{i=0}^{K} \frac{\sum_{j=0}^{K} \check{\mathtt{d}}_j}{K} \tag{2.7b}$$

$$= \sum_{i=0}^{K} \check{\mathtt{d}}_i - \sum_{i=0}^{K} \frac{1}{K} \cdot \sum_{j=0}^{K} \check{\mathtt{d}}_j \tag{2.7c}$$

$$= \sum_{i=0}^{K} \check{\mathtt{d}}_i - 1 \cdot \sum_{i=0}^{K} \check{\mathtt{d}}_i = 0, \tag{2.7d}$$

Therefore, to minimize Equation 2.5a, given the optimal translation so that $\mathtt{t} - \check{\mathtt{c}}_M + \Delta\mathbf{R} \cdot \check{\mathtt{c}}_D = 0$, Term 2.5b which does not depend on Δt has to be minimized:

$$E(\Delta\mathbf{R}) = \sum_{i=1}^{K} (\|\hat{\mathtt{m}}_i - \Delta\mathbf{R} \cdot \hat{\mathtt{d}}_i\|)^2 \tag{2.8a}$$

$$= \sum_{i=1}^{K} ((\|\hat{\mathtt{m}}_i\|)^2 - 2 \cdot \hat{\mathtt{m}}_i \cdot (\Delta\mathbf{R} \cdot \hat{\mathtt{d}}_i) + (\|\Delta\mathbf{R} \cdot \hat{\mathtt{d}}_i\|)^2) \tag{2.8b}$$

$$= \sum_{i=1}^{K} (\|\hat{\mathtt{m}}_i\|)^2 \tag{2.8c}$$

$$- 2 \sum_{i=1}^{K} (\hat{\mathtt{m}}_i \cdot (\Delta\mathbf{R} \cdot \hat{\mathtt{d}}_i)) \tag{2.8d}$$

$$+ \sum_{i=1}^{K} (\|\Delta\mathbf{R} \cdot \hat{\mathtt{d}}_i\|)^2 \tag{2.8e}$$

Rotation is a length preserving operation, so for Term 2.8e holds $(\|\Delta\mathbf{R}\cdot\hat{\mathtt{d}}_i\|)^2 = (\|\hat{\mathtt{d}}_i\|)^2$. Using Equation 2.1, we can resolve Term 2.8d to:

$$\begin{aligned}\sum_{i=1}^{K}(\hat{\mathtt{m}}_i\cdot\Delta\mathbf{R}\cdot\hat{\mathtt{d}}_i) &= \sum_{i=1}^{K}\left[\begin{pmatrix}\hat{m}_i^x\\ \hat{m}_i^y\end{pmatrix}\cdot\begin{pmatrix}\cos\Delta\gamma & -\sin\Delta\gamma\\ \sin\Delta\gamma & \cos\Delta\gamma\end{pmatrix}\begin{pmatrix}\hat{d}_i^x\\ \hat{d}_i^y\end{pmatrix}\right]\\ &= \sum_{i}^{K}(\cos\Delta\gamma\cdot\hat{m}_i^x\hat{d}_i^x+\sin\Delta\gamma\cdot\hat{m}_i^y\hat{d}_i^x-\sin\Delta\gamma\cdot\hat{m}_i^x\hat{d}_i^y+\cos\Delta\gamma\cdot\hat{m}_i^y\hat{d}_i^y) \qquad (2.9)\\ &= \cos\Delta\gamma\sum_{i=1}^{K}(\hat{m}_i^x\hat{d}_i^x+\hat{m}_i^y\hat{d}_i^y)+\sin\Delta\gamma\sum_{i=1}^{K}(\hat{m}_i^y\hat{d}_i^x-\hat{m}_i^x\hat{d}_i^y)\end{aligned}$$

The yaw rotation $\Delta\gamma$ in Equation 2.9 can be resolved to:

$$\Delta\gamma = \arctan\left(\frac{\sum_{i=1}^{K}(\hat{m}_i^y\hat{d}_i^x-\hat{m}_i^x\hat{d}_i^y)}{\sum_{i=1}^{K}(\hat{m}_i^x\hat{d}_i^x+\hat{m}_i^y\hat{d}_i^y)}\right) \qquad (2.10)$$

Furthermore, reiterating Equation 2.6, the 2D translation $\Delta\mathtt{t} = (t^x, t^y)$ is given by:

$$\Delta t = \begin{pmatrix}\breve{c}_m^x\\ \breve{c}_m^y\end{pmatrix}\cdot\begin{pmatrix}\cos\Delta\gamma & -\sin\Delta\gamma\\ \sin\Delta\gamma & \cos\Delta\gamma\end{pmatrix}\begin{pmatrix}\breve{c}_d^x\\ \breve{c}_d^y\end{pmatrix} \qquad (2.11)$$

With regard to the three ICP steps listed in the beginning of this subsection, methods to determine and to minimize the error, and to find correspondences are required. While the first two have been tackled by Equation 2.2, 2.9, 2.10, 2.11, we still have to determine how to find correspondence. In case the point-to-point metric is used, finding corresponding points in data set D and M translates to finding the nearest neighbor $m_j \in M$ for each $\mathtt{d}_i \in D$.

The brute force approach to finding the nearest neighbor has a time complexity of $O(|D|\,|M|)$. By using KD-tree based nearest neighbor algorithms, e.g. FLANN [79] or libnabo [52], the time complexity can be reduced to $O(|D|\log|M|)$. The correspondence error is the aggregated distance between all the correspondence point pairs determined before. It is mathematically defined in Equation 2.2. This straight forward approach is of time complexity $O(|D|)$.

Algorithm 1 gives a pseudo code formulation for 3 DoF ICP scan registration as previously discussed. The termination criteria for this formulation are fulfilled when either n_{max} iterations, a sufficiently low error ε_E, or convergence has been reached. Note that an actual implementation might require additional steps, such as filtering source and reference points, and rejecting correspondences, e.g. by a density filter [3] and by maximum distance, respectively.

Algorithm 1 ICP point-to-point Scan Registration

Require: model set $M = \{\mathtt{m}_i\}$,
data set $D = \{\mathtt{d}_j\}$,
initial pose estimate $S_0 = (\mathbf{R}_0, \mathtt{t}_0) = (\mathbf{R}(\gamma_0), \mathtt{t}_0)$,
minimum accepted error ε_E
convergence step size $\varepsilon_{\Delta E}$
maximum number of iterations n_{max}

function REGISTERSCAN($M, D, S_0, \varepsilon_E, \varepsilon_{\Delta E}, n_{max}$)
  $\mathtt{t} \leftarrow \mathtt{t}_0$
  $\mathbf{R} \leftarrow \mathbf{R}_0$
  $n \leftarrow 0$
  $E \leftarrow \infty$
  $\Delta E \leftarrow \infty$
  $D^* \leftarrow D$
  while $(E > \varepsilon_E)$ and $(\Delta E > \varepsilon_{\Delta\varepsilon})$ and $(n < n_{max})$ **do**
    $n \leftarrow n + 1$
    for all $\mathtt{d}_j \in D$ **do**
      $\mathtt{d}_j^* \leftarrow R \cdot \mathtt{d}_j + \mathtt{t}$
    end for
    $E^* \leftarrow 0$
    for all $\mathtt{d}_j^* \in D^*$ **do**
      $e \leftarrow \infty$
      for all $\mathtt{m}_i \in M$ **do**
        if $((||\mathtt{m}_i - \mathtt{d}_j^*||)^2)$ smaller e **then**
          $\check{\mathtt{m}}_j \leftarrow \mathtt{m}_i$
          $e \leftarrow (||\mathtt{m}_i - \mathtt{d}_j^*||)^2$
        end if
      end for
      $E^* \leftarrow E^* + e$
    end for
    if E^* larger ε_E **then**
      $(\Delta\mathbf{R}, \Delta\mathtt{t}) \leftarrow \arg\min_{(\Delta\mathbf{R}, \Delta\mathtt{t})} \sum_k (||\check{\mathtt{m}}_k - (\Delta\mathbf{R} \cdot \check{\mathtt{d}}_k + \Delta\mathtt{t})||)^2$
      $\mathbf{R} \leftarrow \mathbf{R} + \Delta\mathbf{R}$
      $\mathtt{t} \leftarrow \mathtt{t} + \Delta\mathtt{t}$
    end if
    $\Delta E \leftarrow E - E^*$
    $E \leftarrow E^*$
  end while
end function

2.1.2 Point-to-Plane Iterative Closest Point Registration

In structured environments, such as buildings, one can make reasonable assumptions of the terrain layout in-between adjacent measurements. Point-to-plane ICP [42] presents an extension to the traditional ICP algorithm [4] by using an error metric from each source point to the tangent plane of the corresponding reference points, thus relaxing the requirement for exact point matches. This error metric has been shown to converge faster in general [80], especially for few or distant points. As multi-line 3D LiDAR sensors produce sparse data along the vertical axis, point-to-plane ICP is a reasonable choice for such sensors. The approach can be pictured as allowing source points to "slide" along tangent lines/planes defined by the reference points, see Figure 2.1. In practice, as M and D are sets of points, the tangent lines/planes are calculated using the nearest neighbors around the respective points.

Unfortunately, at the time of writing, there exists no closed-form solution for the point-to-plane ICP algorithm, and using standard nonlinear least-squares tends to be quite slow. However, there does exist a standard linear least-squares approximation to the nonlinear optimization problem, proposed in [43], that we will present here. The system of linear equations approximation of the original nonlinear system assumes a low angular offset between source and reference, and can be solved via singular value decomposition (SVD) [81]. In practice, the linear approximation has been shown to work for rather large angles of up to 30°.

The primary steps for the point-to-plane ICP algorithm are essentially the same as the ones for point-to-point ICP registration presented in Subsection 2.1.1. Namely, (i) finding correspondence between source and reference points, (ii) determining the resulting error, and (iii) minimizing the error by finding adequate pose updates.

Similar to the 3 DoF ICP approach presented previously, we define the reference data set as $M = (\mathtt{m}_0, \mathtt{m}_1, .., \mathtt{m}_{|M|})$ and the source data set as $D = (\mathtt{d}_0, \mathtt{d}_1, .., \mathtt{d}_{|D|})$, with $\mathtt{m}_i = (m_i^x, m_i^y, m_i^z, 1)^T$ and $\mathtt{d}_i = (d_i^x, d_i^y, d_i^z, 1)^T$. That is, each $\mathtt{m}_i$, $\mathtt{d}_i$ stores a x-, y-, and z-position, denoted by the according

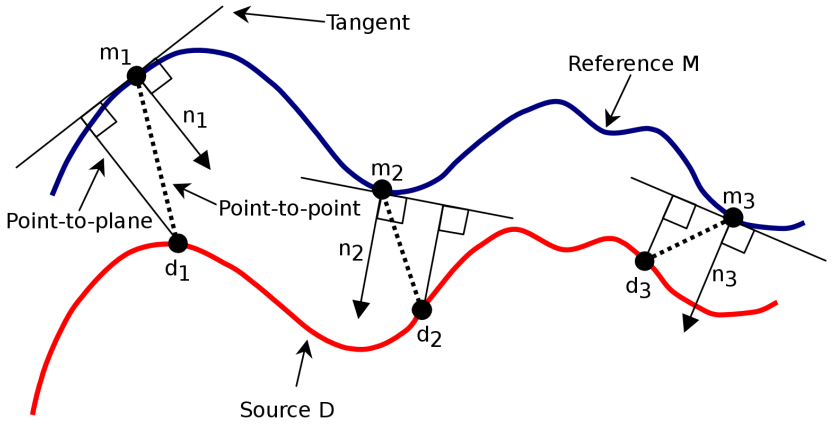


FIGURE 2.1: This figure depicts the difference between point-to-point and point-to-plane correspondence for ICP registration [43]. If the latter is used, the source points are aligned with the tangent through the reference points.

superset character. In contrast to point-to-point registration, we define the error E by the point-to-plane distance between source points and corresponding plane given by the reference points:

$$E(\mathbf{S}) = \sum_{i=1}^{|D|} \sum_{j=1}^{|M|} w_{ij}((\mathbf{S}\cdot \mathtt{d}_i - \mathtt{m}_j)\cdot \mathtt{n}_j)^2), \tag{2.12}$$

where $\mathtt{n}_j = (n_j^x, n_j^y, n_j^z, 0)^T$ is the unit normal vector at point $\mathtt{m}_j$, and $w_{ij} = 1$ if point $\mathtt{d}_i$ corresponds to the plane given by $\mathtt{m}_j$ and $\mathtt{n}_j$, and $w_{ij} = 0$ otherwise. $\mathbf{S}$ is a 3D rigid-body transformation given by the 4×4 matrix consisting of rotation $\mathbf{R}(\alpha,\beta,\gamma)$ and translation $\mathbf{T}(t^x,t^y,t^z)$:

$$\mathbf{S} = \mathbf{T}(t^x,t^y,t^z)\cdot \mathbf{R}(\alpha,\beta,\gamma), \tag{2.13}$$

$$\mathbf{T}(t^x,t^y,t^z) = \begin{pmatrix} 1 & 0 & 0 & t^x \\ 0 & 1 & 0 & t^y \\ 0 & 0 & 1 & t^z \\ 0 & 0 & 0 & 1 \end{pmatrix}, \tag{2.14}$$

and

$$\mathbf{R}(\alpha,\beta\,\gamma) = \mathbf{R}^z(\gamma)\cdot \mathbf{R}^y(\beta)\cdot \mathbf{R}^x(\alpha) \tag{2.15}$$

$$= \begin{pmatrix} \cos\gamma\cos\beta & -\sin\gamma\cos\alpha + \cos\gamma\sin\beta\sin\alpha & \sin\gamma\sin\alpha + \cos\gamma\sin\beta\cos\alpha & 0 \\ \sin\gamma\cos\beta & \cos\gamma\cos\alpha + \sin\gamma\sin\beta\sin\alpha & -\cos\gamma\sin\alpha + \sin\gamma\sin\beta\cos\alpha & 0 \\ -\sin\beta & \cos\beta\sin\alpha & \cos\beta\cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}, \tag{2.16}$$

where α, β, and γ refer to roll, pitch, yaw rotation, and $\mathbf{R}^x$, $\mathbf{R}^y$, and $\mathbf{R}^z$, are the basic rotation matrices around the x, y and z axis, respectively:

$$\mathbf{R}^x(\alpha) = \begin{pmatrix} \cos\alpha & -sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{pmatrix}, \tag{2.17}$$

$$\mathbf{R}^y(\beta) = \begin{pmatrix} \cos\beta & 0 & sin\beta \\ 0 & 1 & 0 \\ -\sin\beta & 0 & \cos\beta \end{pmatrix}, \tag{2.18}$$

$$\mathbf{R}^z(\gamma) = \begin{pmatrix} \cos\gamma & -\sin\gamma & 0 \\ \sin\gamma & \cos\gamma & 0 \\ 0 & 0 & 1 \end{pmatrix}. \tag{2.19}$$

We seek a transformation $\Delta\mathbf{S}$ that minimizes the error E defined in Equation 2.12, i.e.

$$\Delta\mathbf{S} = \Delta\mathbf{T}\cdot\Delta\mathbf{R} = \arg\min_{\Delta\mathbf{S}} \sum_{i=1}^{|D|}((\Delta\mathbf{S}\cdot \mathtt{d}_i - \mathtt{m}_i)\cdot \mathtt{n}_i)^2, \tag{2.20}$$

which requires determining $\Delta T = T(\Delta t^x, \Delta t^y, \Delta t^z)$ and $\Delta R = R(\Delta\alpha, \Delta\beta, \Delta\gamma)$. Note that here and in the following d_i, m_i refer to corresponding sensor measurements and map values.
Unfortunately, as $\Delta\mathbf{R}$ consists of nonlinear trigonometric functions, linear least-squares methods are not applicable. Hence, to achieve a linear approximation, we assume α, β, $\gamma \approx 0$ because $\sin\theta \approx \theta$ and $\cos\theta \approx 1$ if angle $\theta \approx 0$. Thus, we assume that the difference in orientation between source and reference points is small. While this is not strictly the case, the linear approximation has been shown to work well in practice. Furthermore, the iterative nature of the approach alleviates this restriction to a certain degree. Equation 2.16 is therefore approximated as follows:

$$\mathbf{R}(\alpha,\beta,\gamma) \approx \begin{pmatrix} 1 & \alpha\beta-\gamma & \alpha\gamma+\beta & 0 \\ \gamma & \alpha\beta\gamma+1 & \beta\gamma-\alpha & 0 \\ -\beta & \alpha & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{2.21}$$

$$\approx \begin{pmatrix} 1 & -\gamma & \beta & 0 \\ \gamma & 1 & -\alpha & 0 \\ -\beta & \alpha & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} = \tilde{R}(\alpha,\beta,\gamma). \tag{2.22}$$

Accordingly, $\tilde{\mathbf{S}} \approx \mathbf{S}$ is given by:

$$\tilde{\mathbf{S}} = \mathbf{T}(t^x, t^y, t^z) \cdot \tilde{\mathbf{R}}(\alpha,\beta,\gamma) \tag{2.23}$$

$$= \begin{pmatrix} 1 & -\gamma & \beta & t^x \\ \gamma & 1 & -\alpha & t^y \\ -\beta & \alpha & 1 & t^z \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{2.24}$$

Subsequently, we seek $\Delta\tilde{\mathbf{S}} = \mathbf{T}(\Delta t^x, \Delta t^y, \Delta t^z) \cdot \tilde{\mathbf{R}}(\Delta\alpha, \Delta\beta, \Delta\gamma)$ according to Equation 2.12:

$$\Delta\tilde{\mathbf{S}} = \arg\min_{\Delta\tilde{\mathbf{S}}} \sum_{i=1}^{|D|} (\underbrace{(\Delta\tilde{\mathbf{S}} \cdot \mathtt{d}_i - \mathtt{m}_i) \cdot \mathtt{n}_i}_{c_i})^2, \tag{2.25}$$

where the term describing c_i can be resolved to

$$c_i = \left[\Delta\tilde{S} \cdot \begin{pmatrix} d_i^x \\ d_i^y \\ d_i^z \\ 1 \end{pmatrix} - \begin{pmatrix} m_i^x \\ m_i^y \\ m_i^z \\ 1 \end{pmatrix} \right] \cdot \begin{pmatrix} d_i^x \\ d_i^y \\ d_i^z \\ 1 \end{pmatrix} \tag{2.26a}$$

$$= [\Delta\alpha \cdot (n_i^z d_i^y - n_i^y d_i^z) + \Delta\beta \cdot (n_i^x d_i^z - n_i^z d_i^x) + \Delta\gamma \cdot (n_i^y d_i^x - n_i^x d_i^y) + \Delta t_x \cdot n_i^x + \Delta t_y \cdot n_i^y + \Delta t_z \cdot n_i^z] \tag{2.26b}$$

$$- [n_i^x m_i^x + n_i^y m_i^y + n_i^z m_i^z - n_i^x d_i^x - n_i^y d_i^y - n_i^z d_i^z]. \tag{2.26c}$$

Note that Term 2.26b can be rewritten in matrix notation:

$$\begin{pmatrix} n_i^z s_i^y - n_i^y s_i^z & n_i^x s_i^z - n_i^z s_i^x & n_i^y s_i^x - n_i^x s_i^y & n_i^x & n_i^y & n_i^z \end{pmatrix} \cdot \begin{pmatrix} \Delta\alpha & \Delta\beta & \Delta\gamma & \Delta t^x & \Delta t^y & \Delta t^z \end{pmatrix}^T, \tag{2.27}$$

and therefore

$$c_i = \begin{pmatrix} n_i^z s_i^y - n_i^y s_i^z \\ n_i^x s_i^z - n_i^z s_i^x \\ n_i^y s_i^x - n_i^x s_i^y \\ n_i^x \\ n_i^y \\ n_i^z \end{pmatrix}^T \cdot \underbrace{\begin{pmatrix} \Delta\alpha \\ \Delta\beta \\ \Delta\gamma \\ \Delta t^x \\ \Delta t^y \\ \Delta t^z \end{pmatrix}}_{\Delta x} - n_i^x m_i^x + n_i^y m_i^y + n_i^z m_i^z - n_i^x d_i^x - n_i^y d_i^y - n_i^z d_i^z. \tag{2.28}$$

Reiterating Equation 2.20 and Equation 2.25, it holds that

$$\Delta\mathbf{S} \approx \Delta\tilde{\mathbf{S}} = \arg\min_{\Delta\tilde{\mathbf{S}}} \sum_{i=1}^{|D|} ((\Delta\tilde{\mathbf{S}} \cdot \mathtt{d}_i - \mathtt{m}_i) \cdot \mathtt{n}_i)^2 \tag{2.29}$$

$$= \min_{\Delta\mathtt{x}} |\mathbf{A}\Delta\mathtt{x} - \mathtt{b}|^2, \tag{2.30}$$

where $\Delta\mathtt{x}$ is defined in Equation 2.28, and for $0 < i < |D|$

$$\mathbf{A} = \begin{pmatrix} n_1^z s_1^y - n_1^y s_1^z & n_1^x s_1^z - n_1^z s_1^x & n_1^y s_1^x - n_1^x s_1^y & n_1^x & n_1^y & n_1^z \\ n_2^z s_2^y - n_2^y s_2^z & n_2^x s_2^z - n_2^z s_2^x & n_2^y s_2^x - n_2^x s_2^y & n_2^x & n_2^y & n_2^z \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ n_{|D|}^z s_{|D|}^y - n_{|D|}^y s_{|D|}^z & n_{|D|}^x s_{|D|}^z - n_{|D|}^z s_{|D|}^x & n_{|D|}^y s_{|D|}^x - n_{|D|}^x s_{|D|}^y & n_{|D|}^x & n_{|D|}^y & n_{|D|}^z \end{pmatrix}, \tag{2.31}$$

and

$$\mathtt{b} = \begin{pmatrix} n_1^x d_1^x + n_1^y d_1^y + n_1^z d_1^z - n_1^x s_1^x - n_1^y s_1^y - n_1^z s_1^z \\ n_2^x d_2^x + n_2^y d_2^y + n_2^z d_2^z - n_2^x s_2^x - n_2^y s_2^y - n_2^z s_2^z \\ \vdots \\ n_{|D|}^x d_{|D|}^x + n_{|D|}^y d_{|D|}^y + n_{|D|}^z d_{|D|}^z - n_{|D|}^x s_{|D|}^x - n_{|D|}^y s_{|D|}^y - n_{|D|}^z s_{|D|}^z \end{pmatrix}. \tag{2.32}$$

The approximate optimum rotation and translation $\Delta\mathtt{x}$ can be calculated by solving Equation 2.30 via SVD [81], see [43, 82]. Be aware that the rotation approximation $\tilde{\mathbf{R}}(\Delta\alpha, \Delta\beta, \Delta\gamma)$ may not be a valid rotation matrix, thus $\mathbf{R}(\Delta\alpha, \Delta\beta, \Delta\gamma)$ should be used for the final 3D rigid-body transformation, instead.

Finding correspondences in 6 DoF point-to-plane ICP registration can be achieved in the same manner as in the 3 DoF point-to-point case (see Subsection 2.1.1), i.e. by KD-tree based nearest neighbor algorithms of time complexity $O(|D| \log |M|)$. However, as the point-to-plane error metric requires surface normals for every point in the model set, every map update requires finding at least three nearest neighbors for each point $\mathtt{m}_i \in M$. As naive point cloud approaches

lack geometric information, it is impossible to restrict updates to areas that are impacted by the most recent sensor measurements. Therefore, adding new points to the map is of time complexity $O(|M|\log|M|)$. In contrast to $|D|$, $|M|$ grows over time, thus updating the map will require significant computational resources at some point, hence filtering M becomes mandatory. As mentioned earlier, determining the approximately optimum pose update requires SVD of A. Exact SVD of a $m \times n$ matrix SVD has a time complexity of $O(\min(mn^2, m^2n)$ [83], i.e. $O(|D|^3)$ in this case. Accordingly, D is often subject to filtering, as well.

2.1.3 Gauss-Newton Minimization based Registration

In this subsection we present scan registration based on Gauss-Newton minimization via map gradients, as performed by Hector SLAM [2]. Hector SLAM scan registration is used for fast online learning of occupancy grid cell maps [32], i.e a discrete map representation. The algorithm registers 2D LiDAR sensor input with a map in order to determine the 3 DoF pose, and is designed to require low computational resources. To calculate pose updates, a Gauss-Newton based scan registration algorithm, which is a variation of the stereo image registration technique proposed in [45], is used. The Hector SLAM scan registration algorithm consists of two steps that are repeated until the maximum number of iterations is reached. In this subsection we summarize the Hector SLAM approach as presented in [2]. The two Hector SLAM steps are as follows:

1. Given a rigid transformation $S = (\mathbf{R}(\gamma), \mathtt{t})$ for yaw rotation γ and horizontal translation $\mathtt{t} = (t^x, t^y)^T$, an occupancy grid map M and a set of source points $D = (\mathtt{d}_1, \mathtt{d}_2, .., \mathtt{d}_{|D|})$, determine the map gradients $\nabla M(\mathtt{d}_i)$ around each source point $\mathtt{d}_i$ for $0 < i < |D|$.

2. Find a pose update $\Delta S = (\mathbf{R}(\Delta\gamma), \Delta\mathtt{t})$ for the initial pose S that optimizes error E, or stop if the maximum number of iterations is reached.

As mentioned before, Hector SLAM operates on an occupancy grid map. This map representation is implemented using a two dimensional array, which stores the occupancy probability of the corresponding space. However, the discrete nature of this approach limits precision and does not allow for direct computation of map gradients. Therefore, an interpolation scheme that approximates occupancy probabilities for continuous coordinates is employed. For sake of simplicity, we assume a grid cell size of one in the following. Given a continuous coordinate $\mathtt{p}_m = (p^x_m, p^y_m)$, the four surrounding discrete coordinates $P_{00..11}$, with

$$\mathtt{p}_{00} = (p^x_{00}, p^y_{00}) = (\lfloor p^x_m \rfloor, \lfloor p^y_m \rfloor), \tag{2.33a}$$

$$\mathtt{p}_{01} = (p^x_{01}, p^y_{01}) = (\lfloor p^x_m \rfloor, \lceil p^y_m \rceil), \tag{2.33b}$$

$$\mathtt{p}_{10} = (p^x_{10}, p^y_{10}) = (\lceil p^x_m \rceil, \lfloor p^y_m \rfloor), \tag{2.33c}$$

$$\mathtt{p}_{11} = (p^x_{11}, p^y_{11}) = (\lceil p^x_m \rceil, \lceil p^y_m \rceil), \tag{2.33d}$$

are used for linear interpolation of the map value along x and y axis:

$$\begin{aligned} M(\mathtt{p}_m) \approx (p_m^y - p_{00}^y) \cdot [(p_m^x - p_{11}^x) \cdot M(\mathtt{p}_{00}) + (p_{10}^x - p_m^x) \cdot M(\mathtt{p}_{01})] \\ + (p_{01}^y - p_m^y) \cdot [(p_m^x - p_{10}^x) \cdot M(\mathtt{p}_{11}) + (p_{10}^x - p_m^x) \cdot M(\mathtt{p}_{00})]. \end{aligned} \tag{2.34}$$

Using the four surrounding discrete coordinates, the derivatives for the map gradient, i.e. the slope towards occupied map cells, can also be approximated:

$$\frac{\partial M}{\partial x}(\mathtt{p}_m) = (p_m^y - p_{00}^y) \cdot [M(P_{11}) - M(P_{01})] + (p_{01}^y - p_m^y) \cdot [M(P_{10}) - M(P_{00})], \tag{2.35}$$

$$\frac{\partial M}{\partial y}(\mathtt{p}_m) = (p_m^x - p_{00}^x) \cdot [M(P_{11}) - M(P_{10})] + (p_{11}^x - p_m^x) \cdot [M(P_{01}) - M(P_{00})], \tag{2.36}$$

where $M(P)$ returns the occupancy probability of the map at point $\mathtt{p} = (p^x, p^y)$.

From Subsection 2.1.1 we recall that in the 3 DoF case $\mathbf{R}(\gamma)$ is the rotation matrix for yaw (γ), which is defined as

$$\mathbf{R}(\gamma) = \begin{pmatrix} \cos\gamma & -\sin\gamma \\ \sin\gamma & \cos\gamma \end{pmatrix}, \tag{2.37}$$

and will be denoted as $R = R(\gamma)$, $R^* = R(\gamma^*)$, and $\Delta R = R(\Delta\gamma)$ in the following.
We define the error E via the overlap of scan endpoints $\mathtt{d}_i$ with map M:

$$E(\mathbf{R}, \mathtt{t}) = \sum_{i=1}^{|D|} [1 - M(\mathbf{R} \cdot \mathtt{d}_i + \mathtt{t})]^2. \tag{2.38}$$

Thus, optimally aligning the beam endpoints in D with map M translates to finding a transformation S^* that minimizes the error E:

$$S^* = (\mathbf{R}^*, \mathtt{t}^*) = \arg\min_{S^*} \sum_{i=1}^{|D|} [1 - M(\mathbf{R}^* \cdot \mathtt{d}_i + \mathtt{t}^*)]^2. \tag{2.39}$$

We seek to estimate a transformation update $\Delta S = (\Delta R, \Delta t)$ that decreases the error:

$$\sum_{i=0}^{|D|} [1 - M(\mathbf{R}(\gamma + \Delta\gamma) \cdot \mathtt{d}_i + t + \Delta\mathtt{t})]^2 \to 0. \tag{2.40}$$

For ease of notation we define $S \otimes \mathtt{d}_i$ as applying rotation and translation given by S to point $\mathtt{d}_i$:

$$S \otimes \mathtt{d}_i = \begin{pmatrix} \cos(\gamma) & -\sin(\gamma) \\ sin(\gamma) & cos(\gamma) \end{pmatrix} \begin{pmatrix} d_i^x \\ d_i^y \end{pmatrix} + \begin{pmatrix} t^x \\ t^y \end{pmatrix}. \tag{2.41}$$

Equation 2.40 can then be reformulated to:

$$\sum_{i=0}^{|D|}[1 - M((S+\Delta S)\otimes \mathtt{d}_i)]^2 \to 0. \tag{2.42}$$

Using a first order Taylor expansion developing around the map occupancy value $M((S+\Delta S)\otimes \mathtt{d}_i)$ leads to the following error term:

$$\left[1 - M(S\otimes \mathtt{d}_i) - \nabla M(S\otimes \mathtt{d}_i)\frac{\partial(S\otimes \mathtt{d}_i)}{\partial S}\Delta S\right]^2. \tag{2.43}$$

Here, $\nabla M(S\otimes \mathtt{d}_i)$ is the map gradient at the position given by $S\otimes \mathtt{d}_i$:

$$\nabla M(S\otimes \mathtt{d}_i) = \left(\frac{\partial M}{\partial x}(S\otimes \mathtt{d}_i), \frac{\partial M}{\partial y}(S\otimes \mathtt{d}_i)\right), \tag{2.44}$$

and can be approximated via Equation 2.35 and Equation 2.36.
To minimize Equation 2.43 the partial derivative with respect to ΔS is set to zero:

$$2\sum_{i=1}^{|D|}\left[\nabla M(S\otimes \mathtt{d}_i)\frac{\partial(S\otimes \mathtt{d}_i)}{\partial S}\right]^T\left[1 - M(S\otimes \mathtt{d}_i) - \nabla M(S\otimes \mathtt{d}_i)\frac{\partial(S\otimes \mathtt{d}_i)}{\partial S}\Delta S\right] = 0. \tag{2.45}$$

Solving this equation for ΔS leads to the following Gauss-Newton equation for the minimization problem:

$$\Delta S = H^{-1}\sum_{i=1}^{|D|}\left[\nabla M(S\otimes \mathtt{d}_i)\frac{\partial(S\otimes \mathtt{d}_i)}{\partial S}\right]^T[1 - M(S\otimes \mathtt{d}_i)] \tag{2.46}$$

with

$$H = \sum_{i}^{|D|}\left[\nabla M(S\otimes \mathtt{d}_i)\frac{\partial(S\otimes \mathtt{d}_i)}{\partial S}\right]^T\left[\nabla M(S\otimes \mathtt{d}_i)\frac{\partial(S\otimes \mathtt{d}_i)}{\partial S}\right]. \tag{2.47}$$

Using Equation 2.41 $\frac{\partial(S\otimes d_i)}{\partial S}$ can be resolved to:

$$\frac{\partial(S\otimes \mathtt{d}_i)}{\partial S} = \begin{pmatrix} 1 & 0 & -\sin\gamma\cdot d_i^x & -\cos\gamma\cdot d_i^y \\ 0 & 1 & cos\gamma\cdot d_i^x & -sin\gamma\cdot d_i^y \end{pmatrix}. \tag{2.48}$$

Equation 2.46 can now be evaluated via $\frac{\partial(S\otimes \mathtt{d}_i)}{\partial S}$ and $\nabla M(S\otimes \mathtt{d}_i)$ (Equation 2.44), yielding the desired transformation update that optimizes the scan to map alignment.

Unfortunately, with this non-smooth linear approximation of the map gradient, local quadratic convergence of the scan registration algorithm cannot be guaranteed. In practice however, the algorithm works with sufficient accuracy [2], while requiring little computational resources. No costly correspondence search is required, and H is a $4x4$ matrix, i.e. calculating the inverse of H is negligible in terms of computational costs.

2.2 Map Representations

In this section we first present occupancy grid cell mapping, a fundamental approach to maps created using range sensors. Afterwards, we introduce octomap, another probabilistic map type. In contrast to occupancy grid cell mapping, octomap uses an efficient tree based data structure that enables 3D mapping of relatively large sized environments. Finally, a non-probabilistic approach based on signed distance functions is presented.

2.2.1 Occupancy Grid Cell Maps

Occupancy grid cell maps, as introduced in [32], were originally designed for sonar sensor based mapping. Sonar sensors provide rather inaccurate conic measurements, so multiple readings are used to gain a reasonable model of the environment. Nowadays, they are commonly used in combination with 2D LiDAR based 2D mapping 3 DoF localization SLAM, e.g. in Hector SLAM [2], SPM-ICP [3], and gmapping [11]. Figure 2.2 shows a map created by gmapping, a particle filter based SLAM approach. Such maps are rigid in structure, requiring the map size to be determined in before hand if costly copy operations are to be avoided, where memory intensity depends on grid resolution and size. The aim is to represent the environment by estimating the posterior probability $p(M|d_{1:N}, s_{1:N})$ of the map M, given a set of measurements $d_{1:N}$ and corresponding positions $s_{1:N}$, taken over time $1:N$. However, as estimating such a posterior is of exponential complexity dependant on the size of the map, i.e. $2^{|M|}$, it rapidly becomes computationally infeasible, even for relatively small maps. Thus, the problem is broken down into manageable sub-problems by dividing the environment into cells of equal size. We summarize the introduction to occupancy grid cell mapping given in [84].

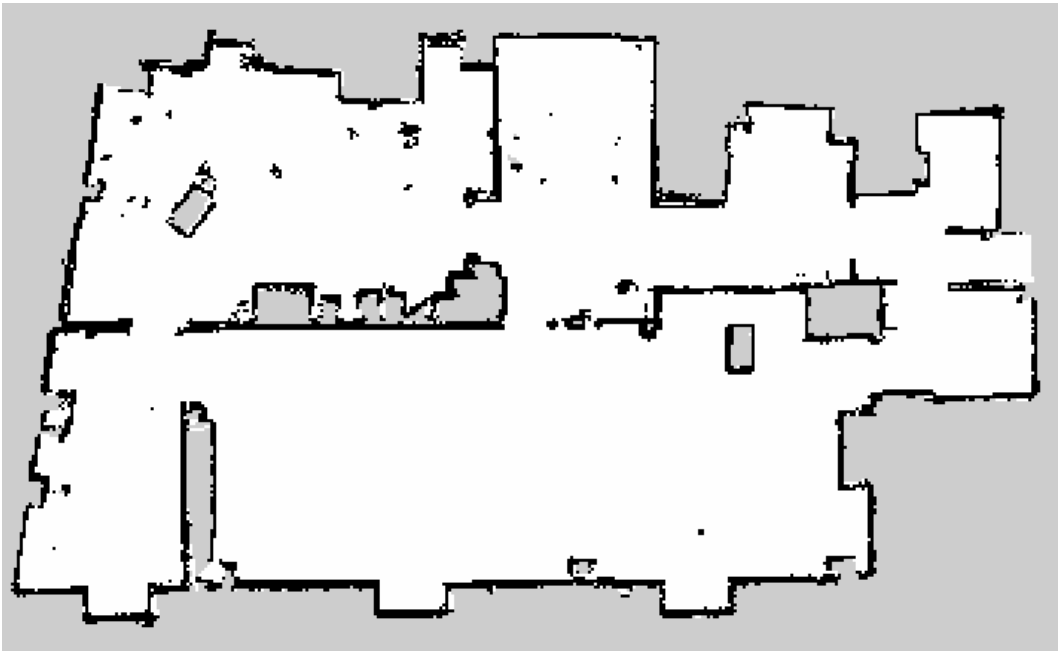


FIGURE 2.2: A rendered occupancy grid cell map for an approximately 15m wide area. Black cells represent occupied, white free, and gray unknown space.

Each occupancy grid map cell stores a binary random variable that stores the probability of being occupied, thus the problem now presents as $p(m_i|d_{1:N}, s_{1:N})$ for each cell m_i, hence

$$p(M|d_{1:N}, s_{1:N}) \approx \prod p(m_i|d_{1:N}, s_{1:N}), \tag{2.49}$$

reducing the complexity to $2|M|$. We assume the state to be static, and a cell is occupied if $p(m_i) = 1$, free if $p(m_i) = 0$, and unknown if $p(m_i) = 0.5$. Note that, as there is no dependency between individual cells, breaking down the original problem discarded modeling dependencies between adjacent cells.
To update the map, the Markov assumption $p(d_t|x, d_{1:N-1}) = p(d_t|x)$, and the Bayes' Rule, i.e.

$$p(a|b) = \frac{p(b|a)p(a)}{p(b)}, \tag{2.50}$$

$$p(a|b,c) = \frac{p(b|a,b,c)p(a|b,c)}{p(b|b,c)}, \tag{2.51}$$

are used repeatedly:

$$p(m_i|d_{1:N}, s_{1:N}) = \frac{p(d_t|m_i, d_{1:N-1}, s_{1:N})p(m_i|d_{1:N-1}, s_{1:N})}{p(d_t|d_{1:N-1}, s_{1:N})} \tag{2.52}$$

$$= \frac{p(d_t|m_i, s_t)p(m_i|d_{1:N-1}, s_{1:N-1})}{p(d_t|d_{1:N-1}, s_{1:N})} \tag{2.53}$$

$$= \frac{p(m_i|d_t, s_t)p(d_t|s_t)p(m_i|d_{1:N-1}, s_{1:N-1})}{p(m_i|s_t)p(d_t|z1:N-1, s_{1:N})} \tag{2.54}$$

$$= \frac{p(m_i|d_t, s_t)p(d_t|s_t)p(m_i|d_{1:N-1}, s_{1:N-1})}{p(m_i)p(d_t|d_{1:N-1}, s_{1:N})}. \tag{2.55}$$

$p(m_i|d_{1:N}, s_{1:N})$ is now based on $p(m_i|d_t, s_t)$, i.e. the inverse sensor model.
The same holds for m_i not being occupied:

$$p(\neg m_i|d_{1:N}, s_{1:N}) = \frac{p(\neg m_i|d_t, s_t)p(d_t|s_t)p(\neg m_i|d_{1:N-1}, s_{1:N-1})}{p(\neg m_i)p(d_t|d_{1:N-1}, s_{1:N})}. \tag{2.56}$$

To gain a recursive update rule, the probability of a cell being occupied is divided by the probability of not being occupied:

$$\frac{p(m_i|d_{1:N}, s_{1:N})}{p(\neg m_i|d_{1:N}, s_{1:N})} = \frac{p(m_i|d_t, s_t)}{1 - p(m_i|d_t, s_t)} \frac{p(m_i|d_{1:N-1}, s_{1:N-1})}{1 - p(m_i|d_{1:N-1}, s_{1:N-1})} \frac{1 - p(m_i)}{p(m_i)}. \tag{2.57}$$

Since we canceled out $p(d_t|s_t)$ and $p(d_t|d_{1:N-1}, s_{1:N})$, we can now update map cells with the probability of being occupied. However, to avoid multiplication, the log odds notation i.e.

$$l(x) = \log \frac{p(x)}{1 - p(x)}, \tag{2.58}$$

and vice versa

$$p(x) = 1 - \frac{1}{1 + \exp l(x)}, \tag{2.59}$$

can be used to turn the product in Equation 2.57 into a sum:

$$l(m_i|d_{1:N}, s_{1:N}) = l(m_i|d_t, s_t) + l(m_i|d_{1:N-1}, s_{1:N-1}) - l(m_i) \tag{2.60}$$

Therefore, updating the map is highly efficient, as it requires only summation.

2.2.2 Octree maps

In this subsection we present the octree based mapping [39] approach octomap [85], a highly optimized framework for volumetric 3D mapping. Similar to occupancy grid approaches (see Subsection 2.2.1), the environment is represented using random binary variables. Therefore, octomap is able to cope with noisy measurements and allows for multiple sensor integration. However, due to the underlying data structure it is superior to occupancy grid approaches in many ways, while maintaining efficient geometric accessibility, which point clouds lack. Octomap is designed to allow for *full 3D mapping*, while being *updatable*, *flexible* and *compact*.
Full 3D mapping means that no prior assumptions about the environment are required. Therefore mapping of arbitrary environments is possible. Octomap can capture free, unknown and occupied space using probabilistic occupancy estimation.
Updatable refers to information encoded in the map being incrementally updatable, which is achieved by providing efficient map access and probabilistic occupancy mapping.
Flexibility relates to a dynamic map size, which does not need to be defined in advance. In addition, the map provides multiple resolutions without overhead. It is however necessary to define the maximum resolution, i.e. the minimum voxel size, in beforehand. If the task at hand requires variable maximum resolutions, octree hierarchies [86] can be used.
Compactness requires the map to be both memory and disk-space efficient, i.e. optimized for random and sequential access memory.

The underlying data structure of octomap, i.e. an octree, is a tree structure where non-leaf nodes have eight descendants. Every node represents a cubic volume (voxel) and stores the corresponding probability of being occupied. Descendant nodes split the voxel represented by their parent node in eight sub-voxels, depicted in Figure 2.3 [40]. Therefore, by descending from root to leaf nodes resolution is increased. Sensor measurements inserted into the octomap are stored in the octrees leaf nodes, which represent voxels of minimum supported size. Nodes are generated on demand when inserting new information. This is both memory efficient, and allows the absence of nodes to encode unknown space. If all siblings encode the same information, the corresponding tree branch can be pruned, as information contained in child nodes is aggregated in parent nodes.

The downside of the octree based data structure for mapping is that direct map access, in contrast to e.g. array based grid maps, is not possible. To retrieve map values, the tree has generally to be descended entirely from root to the leaf node that holds the desired information.

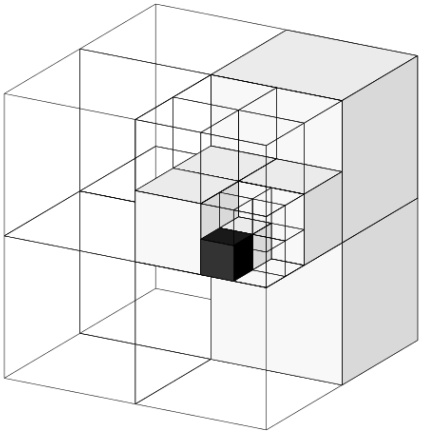


FIGURE 2.3: This figure illustrates the octomap map representation [40]. It shows how voxels are divided into sub-voxels if sensor measurements are inserted. Free space is represented by transparent voxels, occupied space by black voxels, and free space by white voxels. Note how memory is saved by not performing unnecessary branching, e.g. in case of the nodes representing free space.

The exception being if the desired leaf node has been pruned and thus the information being available at a more shallow tree depth, which does not occur frequently. However, as the maximum available map size grows exponentially with tree depth, relatively shallow octrees suffice for relatively large maps. For example, an octomap with a tree depth of 16 and a leaf node resolution of 0.05m can cover over $1600m^3$.

To store occupancy probabilities, the log odd notation is used. This way, updating the map is performed in a computationally efficient manner. See Subsection 2.2.1 for more details on updating the occupancy probability stored in the nodes. After updating the log odds occupancy value of a leaf node, the maximum values are propagated up to the root node. This allows direct rendering of the map in different resolutions, and also provides additional information beneficial to algorithms using the octomap map representation. Figure 2.4 gives an example for the same map being rendered in 4 different resolutions. Path planning for example might only require information about large unoccupied voxels, while still having a fine grain map is beneficial to the localization algorithm.

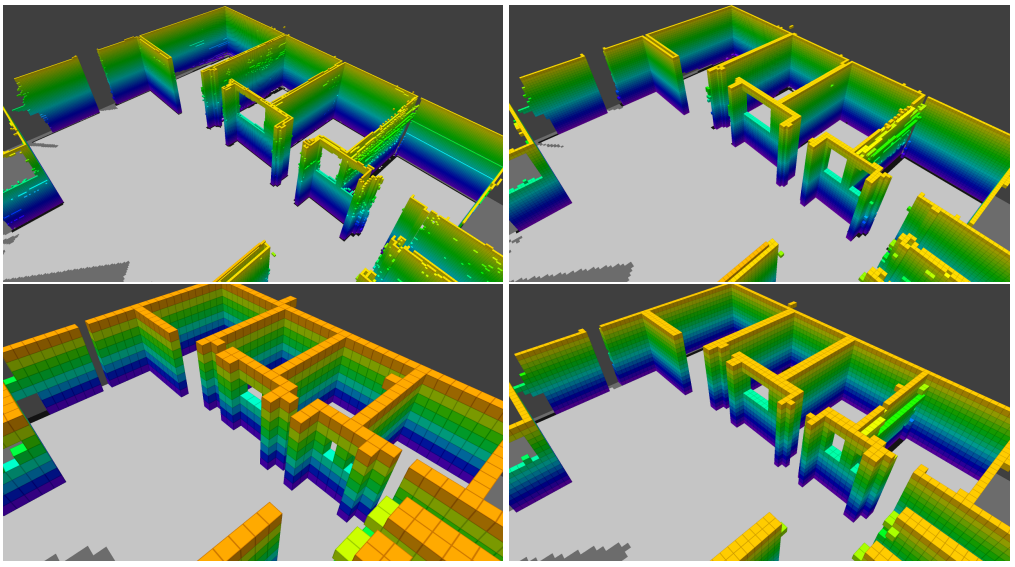


FIGURE 2.4: This figure shows different tree depths of the same octomap map being rendered, where voxel color encodes height. On the top left, leaf nodes that represent a 0.05m resolution are rendered. In clockwise direction the octree is ascended, resulting in voxel resolutions of 0.10m, 0.20m and 0.40m.

2.2.3 Signed Distance Function Maps

Signed distance function (SDF) based maps, as originally proposed in [17], follow a fundamentally different approach than occupancy based maps do. As the name suggest, SDF maps store the signed distance to objects for each position, instead of an occupancy value. A positive distance is used for free space, and a "negative distance" for occupied space. Figure 2.5 gives an example for a SDF based map implemented through a grid structure. It illustrates the principle for a 2D map, where vertical height of points reflects the distance value stored in the corresponding grid cell, with zero distances representing unmapped space.

To update SDF maps, ray tracing [87] from the sensor location to the measured point is used. Positions along the ray between sensor and measured point are updated with the weighted positive distance to the measured point. In contrast, positions on the ray but after the measured point are updated with the negative distance to the measured point. The intersection between positive and negative distance, i.e. the outline of the detected object, is called zero crossing.

To prevent inflating the size of mapped objects, the weight function is chosen in such a way that it tapers off after the zero crossing. As little negative distance as possible should be mapped, because inflating the size of objects becomes problematic if the object is mapped from multiple sides. On the other hand, if the weight function tapers off too early, zero crossings might become compromised due to a lack of sign changes in the map. In practice, the attenuation is usually chosen depending on the accuracy rating of the sensor used for mapping. Additionally, the weight function also tapers off towards the sensor on the other side of the zero crossing, because positive distances become less meaningful the further away they are from objects.

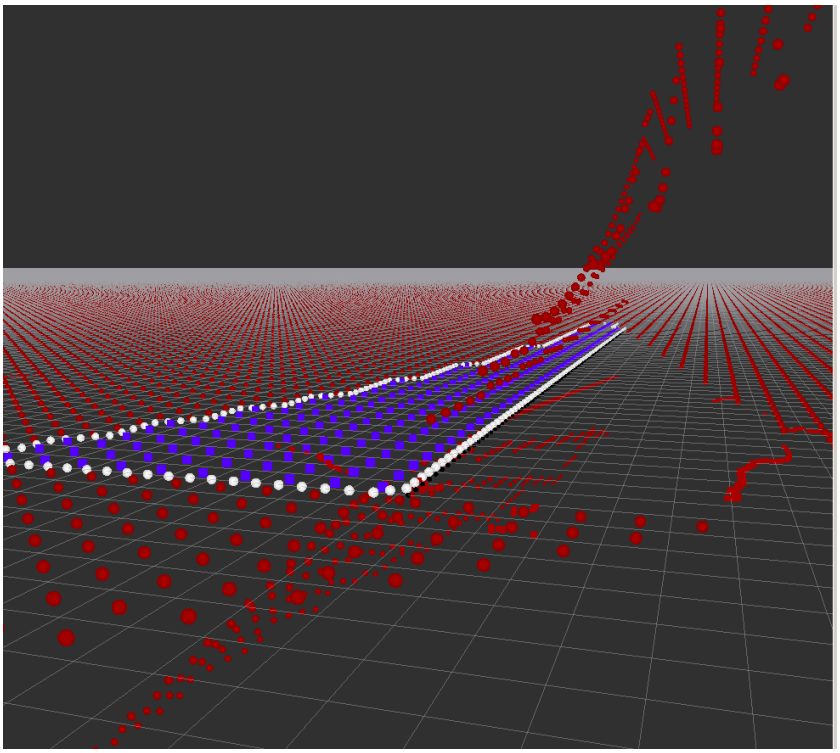


FIGURE 2.5: This figure shows a 3D illustration of a 2D signed distance function based map. The altitude of red points indicate the distance to objects stored in the corresponding cells, where a height of zero is used to represent unknown space. Negative distances, i.e. objects, are represented by blue dots. White dots mark the zero crossing, i.e. a sign change in distances (if zero distances are considered positive for the purpose of generating object outlines).

To demonstrate the principle, we consider mapping a signed distance function M with an immobile 1D range sensor that takes I measurements along the x axis:

$$M(x) = \frac{\sum_{i=1}^{I} w_i(x) d_i(x)}{\underbrace{\sum_{i=1}^{I} w_i(x)}_{W(x)}}, \tag{2.61}$$

where $d_i(x)$ is the distance between sensor and measurement i at position x, and $w_i(x)$ describes the corresponding weight function.
To perform incremental instead of batch updates of the map, the following formula is used:

$$M_{i+1}(x) = \frac{W_i(x) M_i(x) + w_{i+1}(x) d_{i+1}(x)}{W_i(x) + w_{i+1}(x)}, \tag{2.62}$$

with

$$W_{i+1}(x) = W_i(x) + w_{i+1}(x). \tag{2.63}$$

Here, $M_i(x)$ and $W_i(x)$ are the cumulative SDFs and weight functions calculated for the previous update.

Memory and computational efficiency of SDF based maps depend on the implementation. A

naive array based implementation is comparable to occupancy grid cell maps: while sufficiently efficient for 2D maps, representing 3D environments quickly becomes infeasible with size and depending on resolution. To enable a sufficiently efficient SDF based representation of 3D environments, octrees [88] or point clouds [41] can be used.

Chapter 3

Octree based 3D map 3 DoF registration SLAM

This chapter is based on work published in [70, 72]. It focuses on SLAM for airborne robots capable of 6 DoF motion, in particular multirotor unmanned aerial vehicles, equipped with 2D LiDAR sensors. We propose OctoSLAM: combining 2D LiDAR, altitude, and attitude (roll, pitch, yaw) sensor measurements in order to perform 3D map SLAM. To represent the 3D environment an octree based map is used. Our scan registration algorithm is derived from Hector SLAM [2]. We evaluate the performance of our system in simulation and on a real multirotor unmanned aerial vehicles equipped with a 2D LiDAR sensor, a consumer grade inertial measurement unit, and an altitude sensor. The experimental results show significant improvement in the localization and representation accuracy over current 2D map SLAM methods.

3.1 Introduction

Multi-rotor Unmanned Aerial Vehicles (UAVs) are popular yet complex systems which have become widely available to the research community. A common vision is that such airborne agents have a broad scope of applications and can be beneficial in numerous scenarios, for example for coastal [89], mountainous [90], and urban [91] search and rescue. Further civilian examples include geographical mapping, site inspection, and use in agriculture [92]. In such scenarios, (semi-)autonomous exploration of large, inaccessible or hazardous environments can be of use. A prerequisite to such autonomous behavior is the ability of the robot to perceive environmental elements and to localize itself.

SLAM for UAVs can be tackled with 2D time of flight (e.g. 2D LiDAR [15]), 3D time of flight (e.g. 3D LiDAR [77]), visual (e.g. RGB [31]), or hybrid (e.g. RGBD [93]) sensors. Each sensor type has advantages and disadvantages. On the one hand, LiDAR based sensors typically provide more accurate distance measurements and have a higher range than RGBD sensors, while RGB cameras do not provide any distance data at all. On the other hand, LiDAR sensors are generally heavier and more expensive than RGBD sensors, which in turn are outperformed by RGB cameras in that regard. Then again, LiDAR sensors are less susceptible to ambient

lighting or the lack thereof. While 3D LiDAR sensors outperform 2D LiDAR sensors in nearly all aspects, they are also not as commonly available, heavier and a lot more expensive.

In this chapter we will focus on 2D LiDAR based SLAM for 6 DoF motion performing robots. The established practice in this scenario is to project the LiDAR measurements onto 2D maps. This assumes that objects look the same regardless of the observation height. On the one hand, if this assumption holds, 2D map SLAM [94], e.g. SPM-ICP [3], suffices for UAV SLAM. On the other hand, if the assumption does not hold, using 2D maps in the UAV domain may become impracticable. Therefore, we develop a 2D LiDAR based SLAM algorithm that operates on a 3D representation of the environment. We furthermore investigate whether and to what extent using 3D maps for SLAM outperforms the use of 2D maps.

The remainder of this chapter is structured as follows. We introduce our approach, called OctoSLAM, which advances the state-of-the-art low resource requiring 2D LiDAR SLAM frameworks for robots exhibiting 6D motion. This is followed by both simulated and real robot experiments. The chapter then concludes with a brief discussion of our findings.

3.2 OctoSLAM

To achieve 3D map 3 DoF registration SLAM we combine and extend both octomap [40], an octree representation of the environment, and Hector SLAM [2], an algorithm for fast online learning of occupancy grid maps. To implement the proposed SLAM system, the robot operating system [67] framework is used.

OctoSLAM determines the 6D pose of the UAV by 3 DoF registration combined with direct sensor input. In our setup, an inertial measurement unit (IMU) provides roll α and pitch β, while altitude t^z is measured by an actuated downward facing distance sensor. The remaining three dimensions, i.e. translation in x (t^x), y (t^y) direction and yaw rotation γ, are tracked through registration. At time k the position determined by scan registration is given by $\mathtt{s}_k = (t_k^x, t_k^y, \gamma_k)^T$. The algorithm iteratively computes the most recent pose change $\Delta\mathtt{s}_k = \mathtt{s}_k - \mathtt{s}_{k-1} = (\Delta t_k^x, \Delta t_k^y, \Delta\gamma_k)^T$ by registering a set of polar scan endpoints $D = (\mathtt{d}_0, \ldots, \mathtt{d}_{|D|})$ with the map M.

The first step is transforming scan endpoints $\mathtt{d}_i = (r_i, \theta_i)^T$ into Cartesian coordinates in the map frame via the current 6D pose estimate $\mathtt{s}_k^+ = (t_k^x, t_k^y, t_k^z, \alpha_k, \beta_k, \gamma_k)^T$. For ease of notation, we denote transforming scan endpoint $\mathtt{d}_i$ with 6D pose $\mathtt{s}_k^+$ as $\mathtt{s}_k^+ \otimes \mathtt{d}_i$:

$$\mathtt{s}_k^+ \otimes \mathtt{d}_i = \mathbf{R}(\alpha_k, \beta_k, \gamma_k) \cdot (r \cdot \cos\theta, r \cdot sin\theta, 0)^T + (t_k^x, t_k^y, t_k^z)^T. \tag{3.1}$$

The general 3D rotation matrix $\mathbf{R}(\alpha_k, \beta_k, \gamma_k)$ is defined as follows:

$$\mathbf{R}(\alpha, \beta\, \gamma) = \mathbf{R}^z(\gamma) \cdot \mathbf{R}^y(\beta) \cdot \mathbf{R}^x(\alpha), \tag{3.2}$$

where $\mathbf{R}^x$, $\mathbf{R}^y$, and $\mathbf{R}^z$, are the basic rotation matrices around the x, y and z axis, respectively:

$$\mathbf{R}^x(\alpha) = \begin{pmatrix} \cos\alpha & -sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{pmatrix}, \tag{3.3}$$

$$\mathbf{R}^y(\beta) = \begin{pmatrix} \cos\beta & 0 & sin\beta \\ 0 & 1 & 0 \\ -\sin\beta & 0 & \cos\beta \end{pmatrix}, \tag{3.4}$$

$$\mathbf{R}^z(\gamma) = \begin{pmatrix} \cos\gamma & -\sin\gamma & 0 \\ \sin\gamma & \cos\gamma & 0 \\ 0 & 0 & 1 \end{pmatrix}. \tag{3.5}$$

Therefore,

$$\mathbf{R}(\alpha,\beta\,\gamma) = \begin{pmatrix} \cos\gamma\cos\beta & -\sin\gamma\cos\alpha+\cos\gamma\sin\beta\sin\alpha & \sin\gamma\sin\alpha+\cos\gamma\sin\beta\cos\alpha \\ \sin\gamma\cos\beta & \cos\gamma\cos\alpha+\sin\gamma\sin\beta\sin\alpha & -\cos\gamma\sin\alpha+\sin\gamma\sin\beta\cos\alpha \\ -\sin\beta & \cos\beta\sin\alpha & \cos\beta\cos\alpha \end{pmatrix}. \tag{3.6}$$

The vector produced by $\mathtt{s}_k^+ \otimes \mathtt{d}_i$ describes the scan endpoint position in map frame coordinates, which is requiered for inserting a point into, and retrieving occupancy values from the octree.

We follow the Hector SLAM registration approach, which uses occupancy grid map gradients ∇M to align the scan D with the current map. Similarly, OctoSLAM computes interpolated map gradients based on occupancy probabilities of the target and surrounding nodes of an octree based map. The equation for the pose update $\Delta\mathtt{s}_k$ at time k is derived from the Gauss-Newton Equation 2.46 presented in Subsection 2.1.3:

$$\Delta\mathtt{s}_k = \mathbf{H}^{-1}\sum_{i=1}^{|D|}\left[\nabla M(\mathtt{s}_k^+ \otimes \mathtt{d}_i)\frac{\partial(\mathtt{s}_k^+ \otimes \mathtt{d}_i)}{\partial \mathtt{s}_k}\right]^T \cdot\ [1 - M(s_k^+ \otimes d_i)]. \tag{3.7}$$

We recall that the Hessian H is computed as follows:

$$\mathbf{H} = \left[\nabla M(\mathtt{s}_k^+ \otimes \mathtt{d}_i)\frac{\partial(\mathtt{s}_k^+ \otimes \mathtt{d}_i)}{\partial \mathtt{s}_k}\right]^T \cdot \left[\nabla M(\mathtt{s}_k^+ \otimes \mathtt{d}_i)\frac{\partial(\mathtt{s}_k^+ \otimes \mathtt{d}_i)}{\partial \mathtt{s}_k}\right], \tag{3.8}$$

where $\nabla M(\mathtt{s}_k^+ \otimes \mathtt{d}_i)$ are the map gradients at position $\mathtt{s}_k^+ \otimes \mathtt{d}_i$:

$$\nabla M(\mathtt{s}_k^+ \otimes \mathtt{d}_i) = \left(\frac{\partial M}{\partial x}(\mathtt{s}_k^+ \otimes \mathtt{d}_i), \frac{\partial M}{\partial y}(\mathtt{s}_k^+ \otimes \mathtt{d}_i)\right). \tag{3.9}$$

The partial derivative $\frac{\partial(\mathtt{s}_k^+ \otimes \mathtt{d}_i)}{\partial \mathtt{s}_k}$ is computed by:

$$\frac{\partial(\mathtt{s}_k^+ \otimes \mathtt{d}_i)}{\partial \mathtt{s}_k} = \begin{pmatrix} 1 & 0 & -\sin(\gamma)\cdot d_i^x & -\cos(\gamma)\cdot d_i^y \\ 0 & 1 & cos(\gamma)\cdot d_i^x & -sin(\gamma)\cdot d_i^y \end{pmatrix}. \tag{3.10}$$

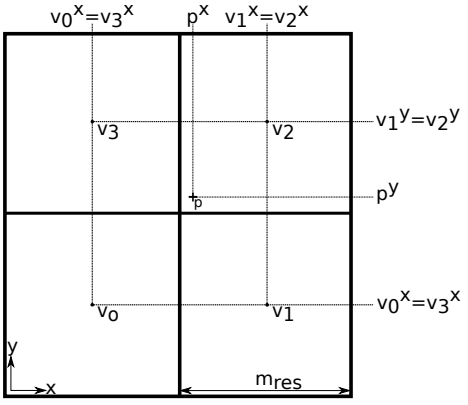


FIGURE 3.1: Simplified 2D visualization for the required voxels surrounding a sensor reading $\mathtt{p}$. Voxels $\mathtt{v}_0(p), \mathtt{v}_1(p), \mathtt{v}_2(p), \mathtt{v}_3(p)$ are used to determine the map value and gradients of $\mathtt{p}$.

In contrast to Hector SLAM, which uses an array based occupancy grid, we use an octree based map. Therefore, calculating the interpolated map values $M(s_k^+ \otimes d_i)$ and gradients $\nabla M(s_k^+ \otimes d_i)$ is slightly different, as unlike with grid approaches, the voxel size throughout the octree is not constant but depends on the tree level. Given a target coordinate $\mathtt{p} = (p^x, p^y, p^z)^T$, the voxels $\mathtt{v}_0(p), \mathtt{v}_1(p), \mathtt{v}_2(p), \mathtt{v}_3(p)$ surrounding $\mathtt{p}$ have to be determined, as illustrated in Figure 3.1.

To determine these voxels, $\mathtt{p}$ is rounded down to the next voxel center coordinate which we define as voxel $\mathtt{v}_0(\mathtt{p})$:

$$\mathtt{v}_0(\mathtt{p}) = \begin{pmatrix} v_0^x(\mathtt{p}) \\ v_0^y(\mathtt{p}) \\ v_0^z(\mathtt{p}) \end{pmatrix} = \begin{pmatrix} m_{res} \oplus p^x \\ m_{res} \oplus p^y \\ m_{res} \oplus p^z \end{pmatrix} \tag{3.11}$$

where m_{res} is the distance between the voxel centers, i.e. the resolution of the octomap which depends on the minimum octree voxel size m_{res}^- and on the tree search depth n.

$$m_{res} := m_{res}(n, m_{res}^-) = 2^{n-1} \cdot m_{res}^- \tag{3.12}$$

The operator $m_{res}\oplus$ is defined as rounding down to the next voxel center. Algorithm 2 gives a pseudo code formulation for $m_{res}\oplus$. In correspondence to voxel $v_0(p)$, voxels $v_1(\mathtt{p})$, $v_2(\mathtt{p})$, and $v_3(\mathtt{p})$ are defined as follows:

$$\mathtt{v}_1(\mathtt{p}) = \begin{pmatrix} v_1^x(\mathtt{p}) \\ v_1^y(\mathtt{p}) \\ v_1^z(\mathtt{p}) \end{pmatrix} = \begin{pmatrix} v_0^x + m_{res} \\ v_0^y \\ v_0^z \end{pmatrix} \tag{3.13a}$$

$$\mathtt{v}_2(\mathtt{p}) = \begin{pmatrix} v_2^x(\mathtt{p}) \\ v_2^y(\mathtt{p}) \\ v_2^z(\mathtt{p}) \end{pmatrix} = \begin{pmatrix} v_1^x \\ v_1^y + m_{res} \\ v_1^z \end{pmatrix} \tag{3.13b}$$

$$\mathtt{v}_3(\mathtt{p}) = \begin{pmatrix} v_3^x(\mathtt{p}) \\ v_3^y(\mathtt{p}) \\ v_3^z(\mathtt{p}) \end{pmatrix} = \begin{pmatrix} v_2^x - m_{res} \\ v_2^y \\ v_2^z \end{pmatrix}. \tag{3.13c}$$

Algorithm 2 Round to Voxel Center

Require: Voxel resolution m_{res},
input coordinate *in*

```
1: function ROUND(m_res, in)
2:     voxelCenter ← 0
3:     mod ← in modulo m_res
4:     if mod larger m_res/2 then
5:         voxelCenter ← in + m_res − mod − m_res/2
6:     else
7:         voxelCenter ← in − mod − m_res/2
8:     end if
9:     return voxelCenter
10: end function
```

The voxels $\mathtt{v}_0(\mathtt{p})..\mathtt{v}_3(\mathtt{p})$ are used to compute the interpolated map value $M(\mathtt{p})$ as described in the previous chapter, with the addition of scaling with m_{res}, which is not necessary when using an array based map with cell size one. We opt to use the same m_{res} for $\mathtt{v}_0(\mathtt{p})..\mathtt{v}_3(\mathtt{p})$, thus limiting the map resolution by the most coarse of the four voxels. The map value $M(\mathtt{p})$ is hence defined as:

$$\begin{aligned} M(\mathtt{p}) = &\frac{v_3^y(\mathtt{p}) - p^y}{m_{res}} \cdot \frac{v_1^x(\mathtt{p}) - p^x}{v_1^x(\mathtt{p}) - v_2^x(\mathtt{p})} \cdot M^*(v_0(\mathtt{p})) \\ &+ \frac{v_3^y(\mathtt{p}) - p^y}{m_{res}} \cdot \frac{p^x - v_2^x(\mathtt{p})}{v_1^x(\mathtt{p}) - v_2^x(\mathtt{p})} \cdot M^*(v_1(\mathtt{p})) \\ &+ \frac{p^y - v_0^y(\mathtt{p})}{m_{res}} \cdot \frac{v_1^x(\mathtt{p}) - p^x}{v_1^x(\mathtt{p}) - v_2^x(\mathtt{p})} \cdot M^*(v_2(\mathtt{p})) \\ &+ \frac{p^y - v_0^y(\mathtt{p})}{m_{res}} \cdot \frac{p^x - v_2^x(\mathtt{p})}{v_1^x(\mathtt{p}) - v_2^x(\mathtt{p})} \cdot M^*(v_3(\mathtt{p})), \end{aligned} \tag{3.14}$$

where $M^*(\mathtt{v}_i)$ refers to the occupancy probability of position $\mathtt{v}_i$ stored in the corresponding octree node.

Map access using octree based maps is not as trivial as for grid based maps. To access the node corresponding to a certain voxel, the octree is traversed from the root node. As explained in Subsection 2.2.2, every octree node has eight children that divide the cubic space into eight equally sized sub-cubes. Since the order which determines which child represents which sub-cube is fixed, a direct path to the target node can be calculated from the target coordinates. This path is then traversed, and every node encountered is stored in a list. If the desired search depth is reached, this node list is returned. Alternatively, if an uninitialized node is encountered during traversal two possibilities arise. (i) The remaining sub-tree might have been pruned, in which case the current node has no children. If this is the case, the current node represents the target voxel and the node list is returned. (ii) The current node does have at least one child, but not the one that lies on the path. In this case the node representing the target coordinates has not been inserted yet, and a "unknown space" marker is added to the list and the list is returned. A pseudo-code formulation for this algorithm is given in Algorithm 3. The map gradients can be

Algorithm 3 Octomap Search

Require: target coordinates tC,
search depth tD,
octree $tree$

```
1:  function OCTOSEARCH(tC, tD, tree)
2:      list ← {}
3:      path ← calculatePath(tC)
4:      curNode ← tree.getRoot()
5:      i ← tree.getMaxTreeDepth()-1
6:      for i down to 0 do
7:          list ← {list, curNode}
8:          if i + 1 equals tD then
9:              return list
10:         end if
11:         nextNode ← path.getNode(i)
12:         if curNode.hasChild(nextNode) then
13:             curNode ← nextNode
14:         else
15:             if curNode.hasNoChildren() then
16:                 return list
17:             else
18:                 list ← {list, −1}
19:                 return list
20:             end if
21:         end if
22:     end for
23:     return list
24: end function
```

found analogously to the map value:

$$\begin{aligned}\frac{\partial M}{\partial x}(\mathtt{p}) = &\frac{v_3^y(\mathtt{p}) - p^y}{m_{res}} \cdot (M(\mathtt{v}_0(\mathtt{p})) - M(\mathtt{v}_2(\mathtt{p}))) \\ &+ \frac{p^y - v_0^y(\mathtt{p})}{m_{res}} \cdot (M(\mathtt{v}_1(\mathtt{p})) - M(\mathtt{v}_3(\mathtt{p})))\end{aligned} \tag{3.15}$$

$$\begin{aligned}\frac{\partial M}{\partial y}(\mathtt{p}) = &\frac{v_1^x(\mathtt{p}) - p^x}{m_{res}} \cdot (M(\mathtt{v}_0(\mathtt{p})) - M(\mathtt{v}_1(\mathtt{p}))) \\ &+ \frac{p^x - v_2^x(\mathtt{p})}{m_{res}} \cdot (M(\mathtt{v}_2(\mathtt{p})) - M(\mathtt{v}_3(\mathtt{p})))\end{aligned} \tag{3.16}$$

Using the map values and gradients of the map for each scan endpoint, Equation 3.7 can be resolved to:

$$\Delta T = \left[\sum_{i=0}^{|D|} \begin{pmatrix} \sqrt{\frac{\partial M}{\partial y}(\mathtt{s}_k^+ \oplus \mathtt{d}_i)} & \frac{\partial M}{\partial y}(\mathtt{s}_k^+ \oplus \mathtt{d}_i) \cdot \frac{\partial M}{\partial x}(\mathtt{s}_k^+ \oplus \mathtt{d}_i) & \frac{\partial M}{\partial y}(\mathtt{s}_k^+ \oplus \mathtt{d}_i) \cdot \gamma_i' \\ \frac{\partial M}{\partial y}(\mathtt{s}_k^+ \oplus \mathtt{d}_i) \cdot \frac{\partial M}{\partial x}(\mathtt{s}_k^+ \oplus \mathtt{d}_i) & \sqrt{\frac{\partial M}{\partial x}(\mathtt{s}_k^+ \oplus \mathtt{d}_i)} & \gamma' \cdot \frac{\partial M}{\partial x}(\mathtt{s}_k^+ \oplus \mathtt{d}_i) \\ \frac{\partial M}{\partial y}(\mathtt{s}_k^+ \oplus \mathtt{d}_i) \cdot \gamma' & \frac{\partial M}{\partial y}(\mathtt{s}_k^+ \oplus \mathtt{d}_i) \cdot \frac{\partial M}{\partial x}(\mathtt{s}_k^+ \oplus \mathtt{d}_i) & \sqrt{\gamma_i'} \end{pmatrix} \right]^{-1} \cdot \sum_{i=0}^{|D|} \left[\left(1 - M(\mathtt{s}_k^+ \oplus \mathtt{d}_i)\right) \begin{pmatrix} \frac{\partial M}{\partial y}(\mathtt{s}_k^+ \oplus \mathtt{d}_i) \\ \frac{\partial M}{\partial x}(\mathtt{s}_k^+ \oplus \mathtt{d}_i) \\ \gamma_i' \end{pmatrix} \right], \tag{3.17}$$

with

$$\begin{aligned} \gamma_i' = (-\sin\gamma \cdot d_i^x - \cos\gamma \cdot d_i^y) \cdot \frac{\partial M}{\partial y}(\mathtt{d}_i) \\ + (\cos\gamma \cdot d_i^x - \sin\gamma \cdot d_i^y) \cdot \frac{\partial M}{\partial x}(\mathtt{d}_i). \end{aligned} \tag{3.18}$$

If a scan endpoint is located in such a way that all surrounding voxels are unoccupied or unknown, no gradients or map values can be determined. While this is not necessarily critical, registration may become too inaccurate if no values can be determined for a sufficiently large number of points. OctoSLAM features three optional heuristics to tackle this situation.
(i) Scan to scan ICP, as introduced in Subsection 2.1.1, is used to achieve a better initial pose $\mathtt{s}_k$. The measurements taken at time k and $k-1$, transformed via pose $\mathtt{s}_k^+$ and $\mathtt{s}_{k-1}^+$, and projected onto the ground plane, are used as source D and reference M for the ICP registration.
(ii) Both map M and transformed measurements $\mathtt{s}_k^+ \oplus \mathtt{d}_i$ are projected onto the ground plane and used for registration. This heuristic is similar to the first one, with the exception that it uses all previous measurements, not just the one at time $k-1$.
(iii) The voxel size m_{res} used for determining the interpolated map value and gradients is increased by reducing the search depth n in Equation 3.12. Since parent nodes in octree based maps aggregate the information stored in their child nodes, increasing voxel size neither increases memory nor computational requirements. By using this heuristic the coverage along the x, y as well as the z dimension is increased. Thus, the maximum pose change from time k to $k-1$ is not limited by a fixed map resolution, as is the case with gradient based grid cell map registration.
Figure 3.2 gives two examples where a heuristic is required to get an inital scan to map coverage. Note that all three heuristics operate under the assumption that objects look similar over height. However, (ii) requires this assumption to hold over the entire height, whereas (i) requires the assumption to hold only between the last two scans, while (iii) incrementally increases the size for which the assumption has to hold. Therefore we deem (iii) to be the most desirable of the three heuristics.

A pseudo code formulation for OctoSLAM registration is given in Algorithm 4. Here the function *getMapValueGradients* refers to $\nabla M(\mathtt{s}_k^+ \otimes \mathtt{d}_i)$. The functions *updateH* and *updateDet* relate respectively to the first and second factor of the multiplication in Equation 3.17. Using these three functions, $\Delta\mathtt{s}_k$ is approximated and subsequently added to $\mathtt{s}_k$ in every iteration.

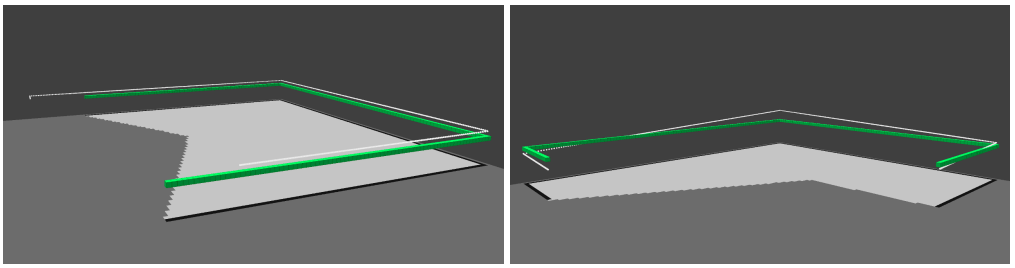


FIGURE 3.2: This figure shows two situations where the robots altitude or attitude change between timesteps has lead to an insufficient overlap between map (voxels) and sensor measurements (white dots). This can be tackled by (i) increasing the voxel size, (ii) using non-incremental point-to-point ICP registration, or (iii) projecting the map (black cells) and measurements onto the ground plane for the current registration iteration.

Algorithm 4 Octomap Scan Registration

Require: Scan endpoints *scan*,
initial pose estimate *pose*,
number of iterations *maxIter*

```
1:  function MATCH(scan, pose, maxIter)
2:      s ← pose
3:      n ← 0
4:      for n to maxIter do
5:          H ← 0
6:          det ← 0
7:          i ← 0
8:          for i to scan.size() do
9:              d ← scan.get(i)
10:             d ← s ⊕ d
11:             M ← getMapValueGradients(d)
12:             H ← H+updateH(M)
13:             det ← det+updateDet(M)
14:         end for
15:         s ← s + (H^-1 * det)
16:     end for
17:     return s
18: end function
```

3.3 Empirical Evaluation

In this section we evaluate the performance of OctoSLAM by conducting simulated and real robot experiments.

For simulated experiments we use Gazebo [95], a simulator capable of simulating articulated robots in three dimensional environments. It generates realistic sensor feedback and supports various simulated sensors, such as 2D LiDAR, IMU, and ultrasonic sensors. The simulated LiDAR sensor we use for our experiments is modeled after the Hokuyo-URG04LX, which has a range of 5m and a field of view of 240 deg. The rated accuracy of the Hokuyo-URG04LX is $\pm$0.03m below 1m range and 3% otherwise. We add noise to the sensor measurements generated

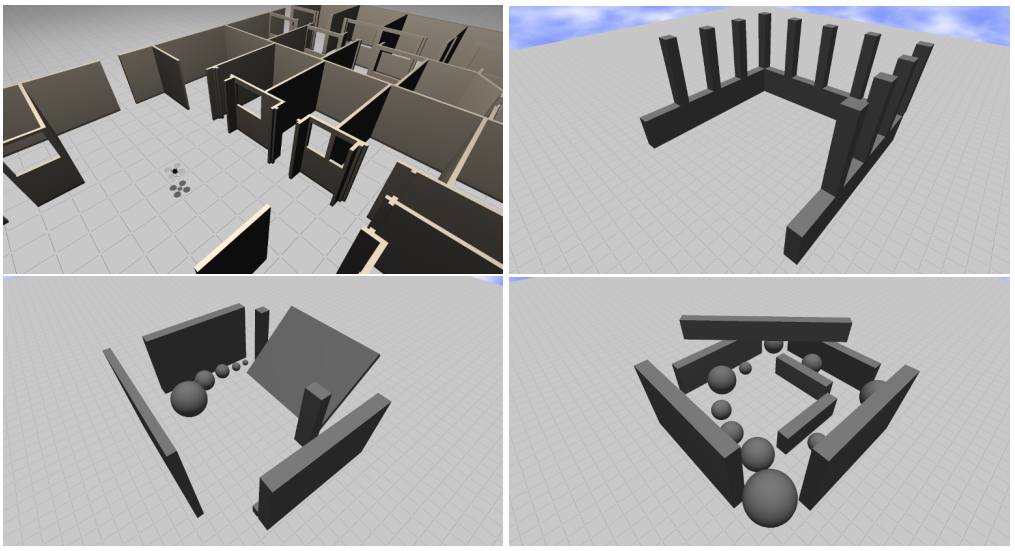


FIGURE 3.3: This figure shows the environments used in the simulated experiments. From the top left in clockwise direction the "Willow", "corridor", "sphere" and "tilted wall" environment is shown. Note that the first two only consist of orthogonal elements, whereas the latter two introduce more complex elements.

by the Gaussian probability distribution with $\mu = 0$ and standard deviation $\sigma = 0.01$:

$$f(x|\mu,\sigma) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}. \tag{3.19}$$

We test OctoSLAM in four simulated environments with different levels of similarity along altitude, shown in Figure 3.3. To measure performance, the root mean squared error (RMSE) of the average Euclidean distance between n number of localized (l_i) and ground truth (g_i) poses is calculated for each experiment episode:

$$RMSE(g,l) = \sqrt{\frac{\sum_{i=1}^{n} (||g_i - l_i||)^2}{n}}. \tag{3.20}$$

Every episode consists of recording the sensor data of a manually controlled flight with an approximate duration of two minutes. Hector and OctoSLAM are then applied to the recorded sensor data, both using a map resolution of 0.05m. OctoSLAM is executed with both 3D and 2D map registration, where in the latter case the map is projected onto the ground plane. In the 2D mode, both "threshold" and "constant" mapping is tested. Threshold mapping refers to updating the map only if the pose has changed by a certain amount and is commonly employed when using 2D maps. Constant mapping in contrast refers to updating the map with every obtained sensor reading, which is necessary to accumulate sufficient sensor measurements to build a 3D map from 2D LiDAR sensor data. The RMSEs for all experiments/approaches are plotted in a bar graph. When comparing multiple bar graphs, be aware that the RMSE axis is scaled for best readability, and is therefore not the same for all bar graphs. In this graph the central line within each bar marks the mean. The inner box around the mean represents the 95% confidence interval for the mean under the assumption that the observations are sampled from a normal distribution.

Therefore, non-overlapping confidence intervals indicate a significant difference for a p-value of 5%. The outer box marks one standard deviation.

To test the SLAM approaches on a real robot, we use a *Mikrokopter Oktokopter XL*. We modify the stock version by adding a *Hokuyo URG-04LX* 2D LiDAR sensor, a downward facing *Parallax PING* ultrasonic sensor and a *Pololu MiniIMU-9* inertial measurement unit. As the ground truth pose is not available to us in the real robot experiments, no quantitative analysis of the results can be given. Instead we perform a qualitative analysis on maps generated by 2D and 3D map SLAM.

3.3.1 Heuristics

As explained in the previous section, OctoSLAM uses up to three heuristics to deal with insufficient scan to map overlap: "multi res.", i.e. variable map resolutions, "var. downproject", i.e. projection of the map onto the ground plane , and ICP registration. Figure 3.4 shows the results for using either of the first two. While the "var. downproject" heuristic works well in the "corridor" environment, introducing non-orthogonal elements to the map decreases the heuristics performance. The multi-resolution heuristic in contrast results in a better performance if more complex elements are introduced to the environment. Therefore we opt to use the multi-resolution heuristic from this point forward.

We found that the ICP heuristic most of the time does not significantly impact the performance. However, if "drastic pose changes" occur, it can be used to increase the performance. Such a case is presented in Figure 3.5, which shows the RMSE over time for a "sphere" environment episode with the ICP heuristic enabled and disabled. Figure 3.6 shows the resulting map for this episode. Here, we can observe that the increase in RMSE is due to yaw localization error. In future experiments, scan to scan ICP is used to improve the initial pose estimate used for scan to map registration.

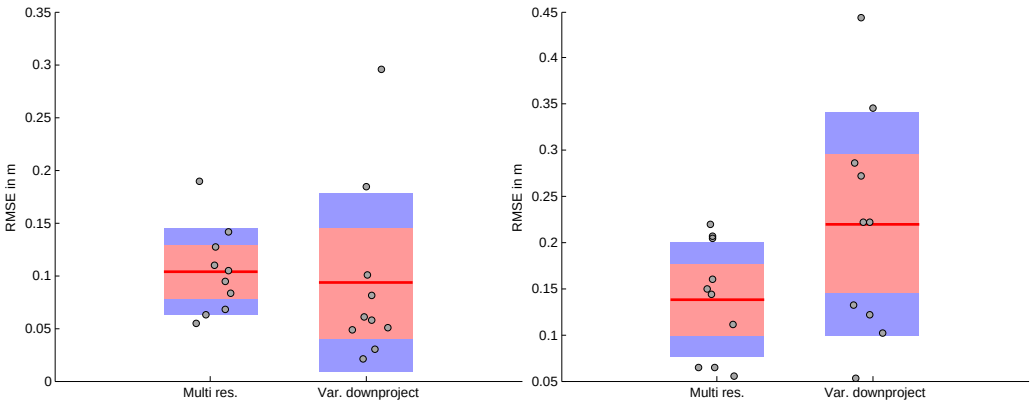


FIGURE 3.4: This figure compares the performance of the multi-resolution heuristic to the variable down-projection heuristic in the "corridor" environment on the left, and the "tilted wall" environment on the right.

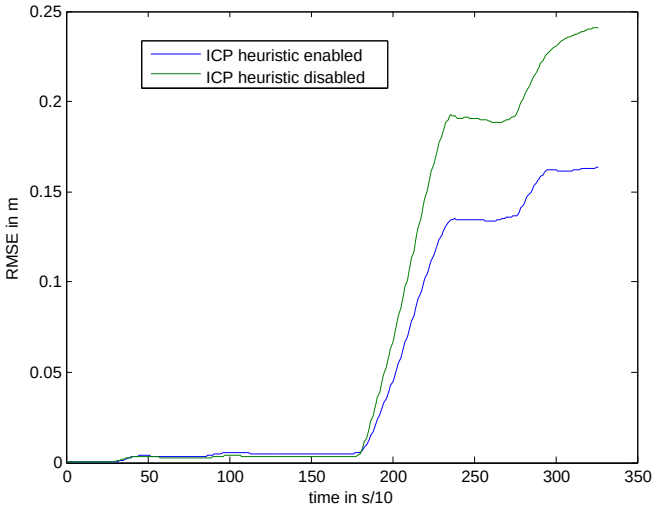


FIGURE 3.5: This graph shows the RMSE for a "sphere" world episode, with enabled (lower line) and disabled (upper line) ICP initial guess heuristic.

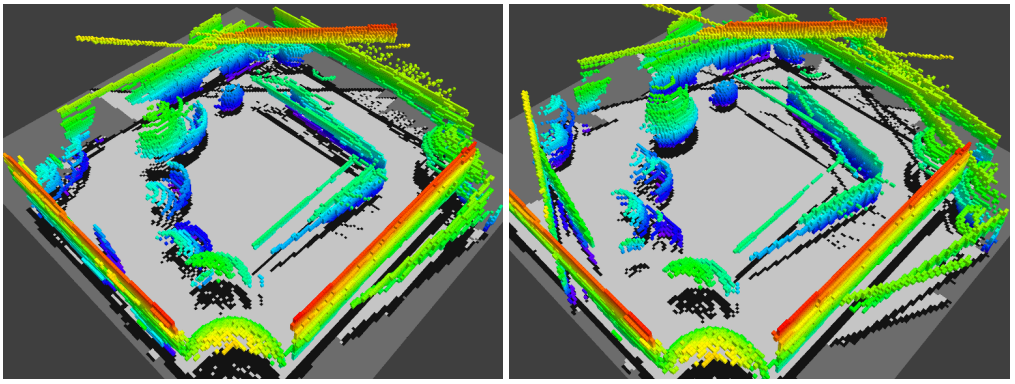


FIGURE 3.6: This figure shows two maps, one generated with the ICP heuristic enabled (left), and one with the heuristic disabled (right).

3.3.2 Simulated Localization on a given Map

To evaluate the localization performance using 3D maps compared to using 2D maps, we generate 0.05m resolution maps of the simulated environments using the ground truth pose. Those maps are then used for localization without mapping. In addition to the results presented in this subsection, the results in tabular form can be found in Appendix A.1.1, and the results for localization without sensor noise are given in Appendix A.1.2.

Figure 3.7 shows the results for the "Willow" environment. On the left the map used for localization, and on the right the results for 3D and 2D map OctoSLAM, are shown. Using a 3D map yields an approximately 11% lower average RMSE. However, as can be seen in the figure, the 95% confidence intervals overlap. Hence, using localization on a 3D map does not give a significant advantage over using a 2D map, in this environment. As this map mostly consists of orthogonal walls reaching from floor to ceiling, one would intuitively expect this result. Furthermore, because the voxel size of the octomap is 0.05m, we consider a localization RMSE of 0.05m to 0.06m to be quite accurate, as the representation resolution itself can introduce inaccuracies of this magnitude.

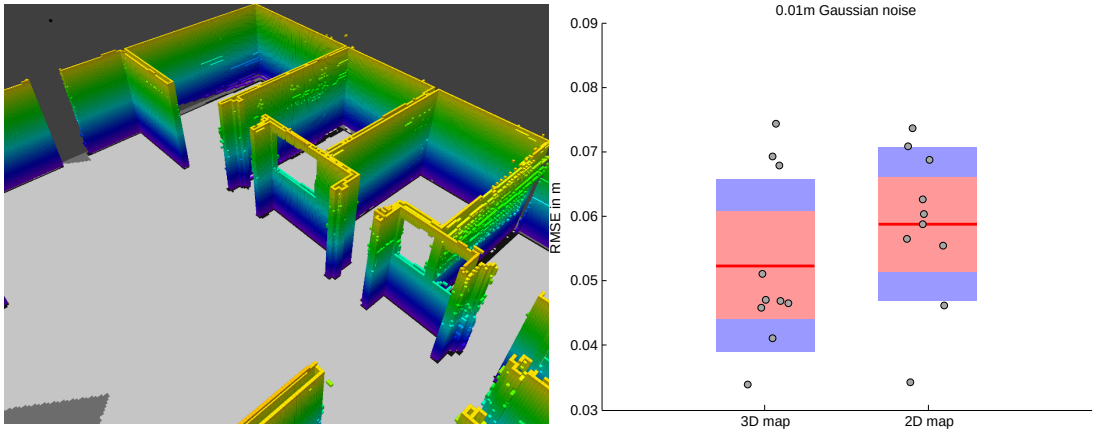


FIGURE 3.7: This figure shows the map of the "Willow" environment used for localization on the left, and on the right the results for localization without mapping. 2D map refers to using the 3D map projected onto the ground plane for registration.

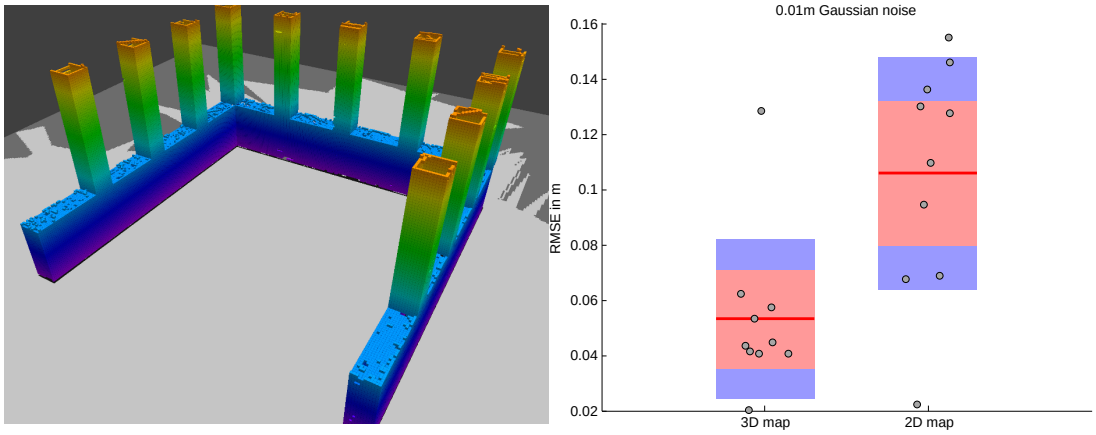


FIGURE 3.8: This figure shows the map of the "corridor" environment used for localization on the left, and on the right the results for localization without mapping. 2D map refers to using the 3D map projected onto the ground plane for registration.

Figure 3.8 shows the results for the "corridor" environment, which introduces slightly more variation over the vertical axis. Similar to the first environment, the average RMSE for 3D map localization ($\sim$0.05m) is approximately equal to the voxel size and can be considered a good result. Using a 3D map yields an around 51% lower average RMSE. The confidence intervals do not overlap, i.e. using a 3D map for localization yields significantly better results than using a 2D map. This result can be explained by the fact that a 2D map makes it hard to distinguish whether a scan hits a pillar or the lower wall. To illustrate the issue, Figure 3.9 shows a 2D map of this environment.

Figure 3.10 shows the results for the "sphere" environment, which as the name suggest introduces curved elements to the environment. Here, the performance of both OctoSLAM variants is very similar. While the result seems unexpected, it can be argued that in this environment the spheres are actually not used for registration too frequently, because the narrow map sizes restricts the movement of the robot.

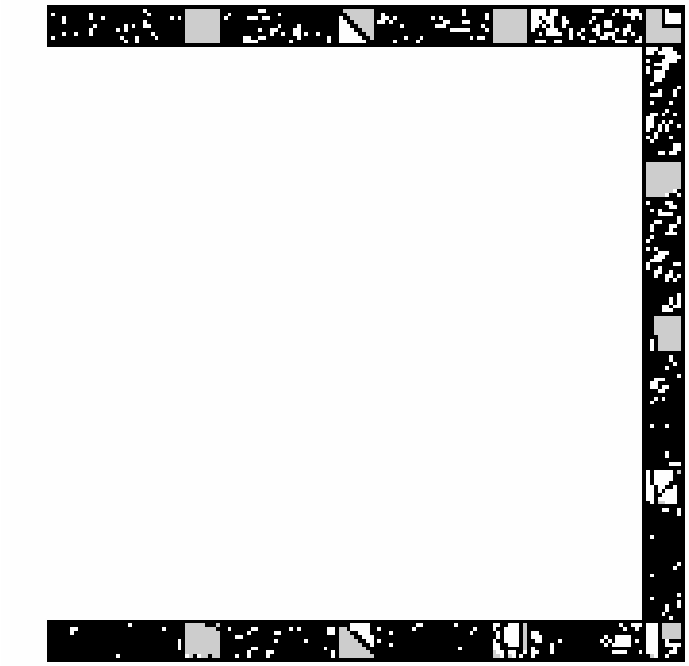


FIGURE 3.9: This figure shows the "corridor" map with voxel size 0.05m projected to 2D.

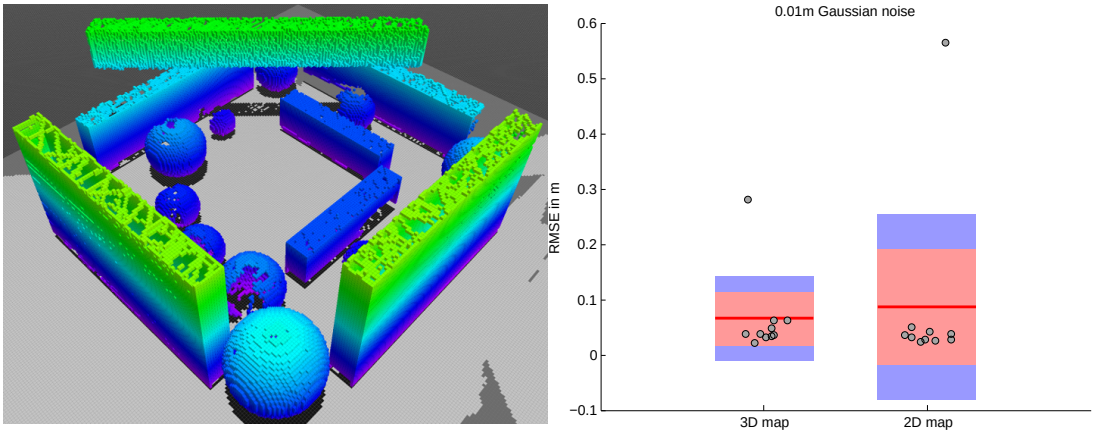


FIGURE 3.10: This figure shows the map of the "sphere" environment used for localization on the left, and on the right the results for localization without mapping. 2D map refers to using the 3D map projected onto the ground plane for registration.

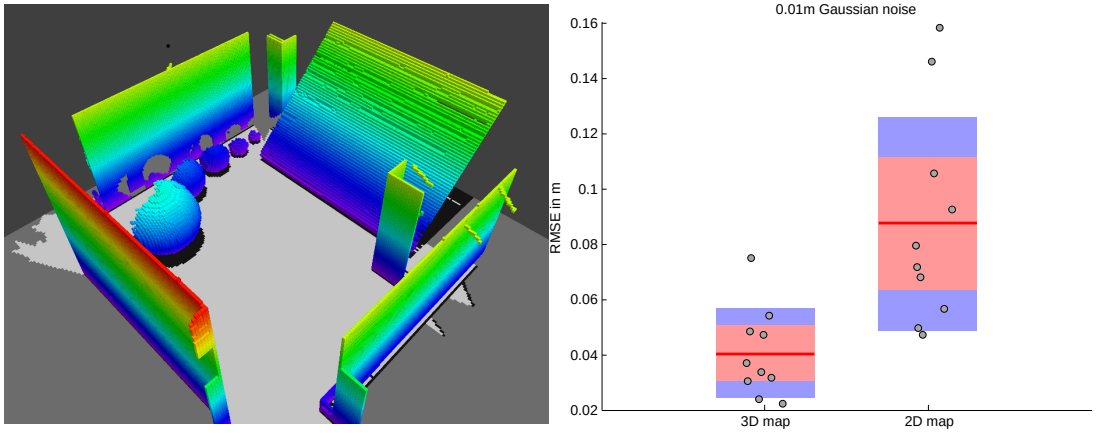


FIGURE 3.11: This figure shows the map of the "tilted wall" environment used for localization on the left, and on the right the results for localization without mapping. 2D map refers to using the 3D map projected onto the ground plane for registration.

The results for the "tilted wall" environment and the corresponding results are given in Figure 3.11. In this experiment 3D map localization yields an approximately 51% lower average RMSE than 2D map localization. As visualized in the figure, this is a significant improvement. This improvement results from the fact that if facing the tilted wall in the top right, 3D map localization can determine the position reliably as it can utilize the height of scan endpoints. 2D map localization on the other hand can not benefit from the height information of the scan endpoints, and hence can not extract reliable map values corresponding to the tilted wall.

To summarize, we found that using a 3D over a 2D map for localization can yield a significant advantage. This is the case, if projecting the environment to 2D leads to large enough parts of the map becoming indistinguishable from each other in areas likely to be in the vicinity of sensor measurements. If facing just a tilted wall for example, the sensor measurements will always be a line, regardless of altitude. If registered with a 2D map, this line could be matched with any part of the tilted wall. In contrast, if the altitude of measurements can be used, the number of possible matches reduces.

3.3.3 Simulated Simultaneous Localization and Mapping

We test simultaneous localization and mapping with three OctoSLAM variants, i.e. 3D, 2D (A), and 2D (B), as well as Hector SLAM. Variant 2D (B) refers to using a 2D map for registration with constant map updates, and (A) refers to using threshold triggered map updates after 0.4m distance traveled or 20° yaw rotation. The same thresholds are used for Hector SLAM. The RMSEs for all SLAM experiments are listed in Appendix A.2.1, and bar graphs for SLAM with no sensor noise are provided in Appendix A.2.2.

The results for SLAM in the "Willow" environment are given in Figure 3.12. While both OctoSLAM 2D A and B perform better on average than Octo SLAM 3D and Hector SLAM, the difference is not significant. It is however interesting to note that the SLAM RMSE of OctoSLAM nearly doubled in comparison to the "localization only" experiments, while the RMSE for OctoSLAM 2D did not. This can be explained by the fact that using 3D maps in

this environment barely provides an advantage over using 3D maps, as the environment mostly consists of walls perpendicular to the ground. However, while the simulated 2D LiDAR sensor captures a consistent 2D map for the 5m long 240 deg wide cone in front of the robot, the 3D map consists of many such scans stitched together via the pose estimates. Thus, the pose error accumulates until sufficient parts of the environment have been inserted into the 3D map. That being said, as the RMSE is below 0.10m for all SLAM approaches, we consider them all adequate for the "Willow" world. The episode used to generate the 3D map has a RMSE of 0.08m. Unfortunately, there is quite some clutter along the walls. However, keep in mind that even using ground truth data for mapping (Figure 3.7) does not produce a completely clean map of this environment. This can be explained by the fact that the walls in this environment line up exactly with the map cell boarders.

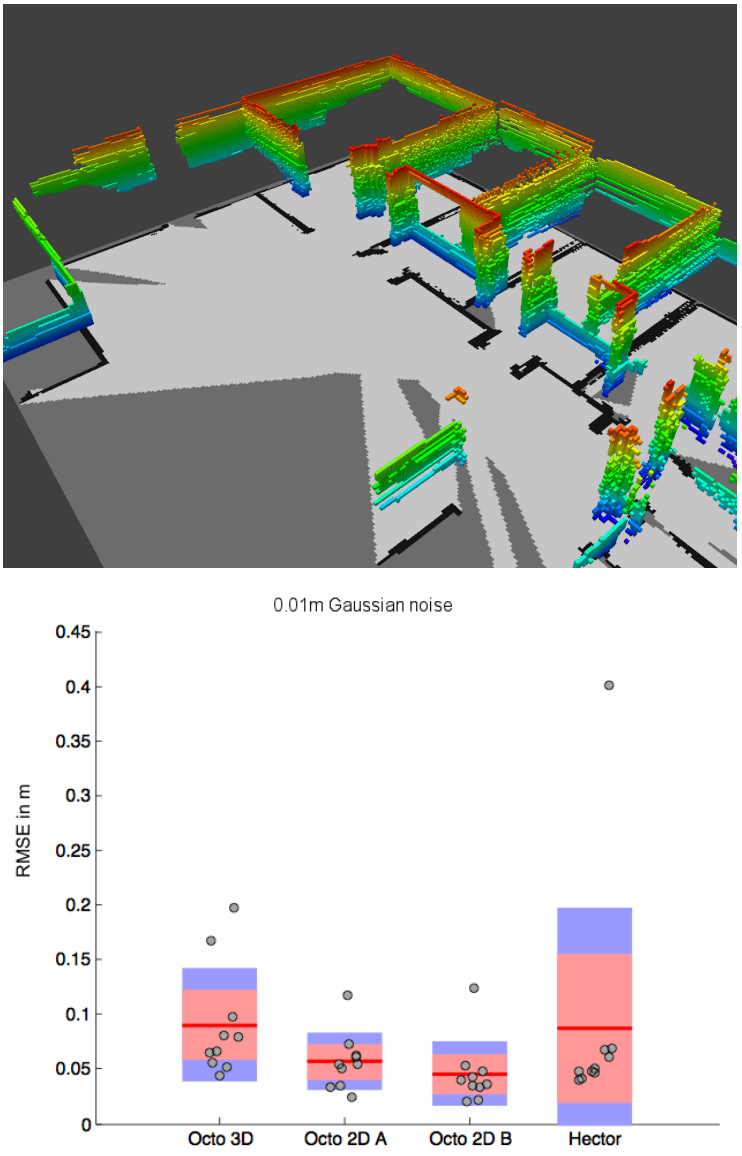


FIGURE 3.12: This figure shows the SLAM results for the "Willow" environment. On the top the 3D map generated by the OctoSLAM 3D median episode can be seen. On the bottom a graph of the RMSEs for SLAM in this environment is shown.

Figure 3.13 visualizes the results for SLAM in the "corridor" environment. Similar to the localization results for this environment, 3D map Octo SLAM significantly outperforms the 2D map variants. It scores an average RMSE of 0.11m, while other approaches suffer from at least twice the RMSE. 3D map OctoSLAM also has the lowest standard deviation of 0.04m, while the 2D map approaches deviate at least thrice as much. The octomap shown is generated by an episode with a RMSE of 0.11m. The walls in this map are rather clean, and the pillars are clearly visible and distinguishable from the lower wall.

The results for SLAM in the "sphere" world are given in Figure 3.14. In this environment, no approach has a significant advantage over the others in terms of average RMSE. Though, the high RMSEs, e.g. 0.35m for 3D map OctoSLAM, stand out. In contrast, in the "localization only" experiments 3D map OctoSLAM did not perform significantly worse here than in

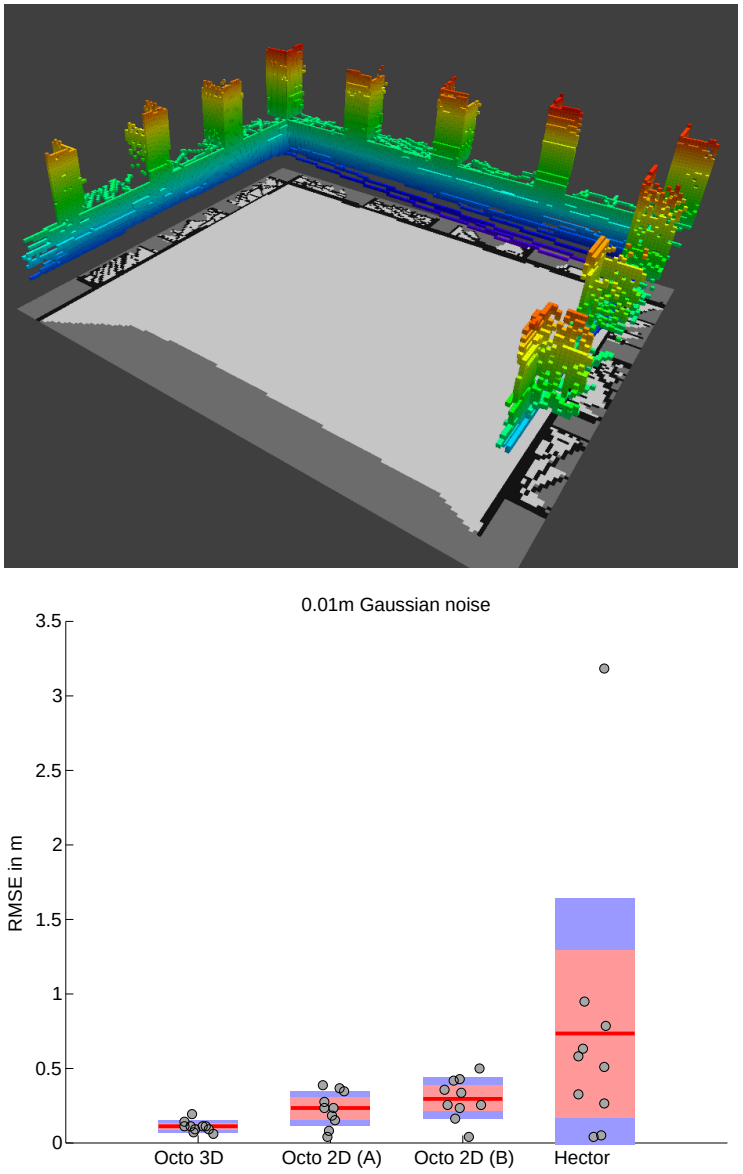


FIGURE 3.13: This figure shows the SLAM results for the "corridor" environment. On the top the 3D map generated by the OctoSLAM 3D median episode can be seen. On the bottom a graph of the RMSEs for SLAM in this environment is shown.

other environments. Also noteworthy is the high standard deviation of 0.33m, brought about by four episodes in which 3D map OctoSLAM has a RMSE over 0.5m. When investigating these episodes we found that ascending in the middle of the map can instantaneously change most of the sensor readings when facing the two small inner walls. If the outer elements are not mapped at that point, the lack of anchor points for the scan registration results in an increased localization error. This is exacerbated by the lack of space for maneuvering, making robot attitudes from which both parts of the inner and outer elements are simultaneously picked up by the sensor less likely. This environment shows the fragility of using a 2D sensor for a 6 DOF motion performing robot. While the median OctoSLAM episode with a RMSE of 0.15m, used for generating the 3D map shown in Figure 3.14, results in an acceptable map, this is not the case for 40% of all episodes.

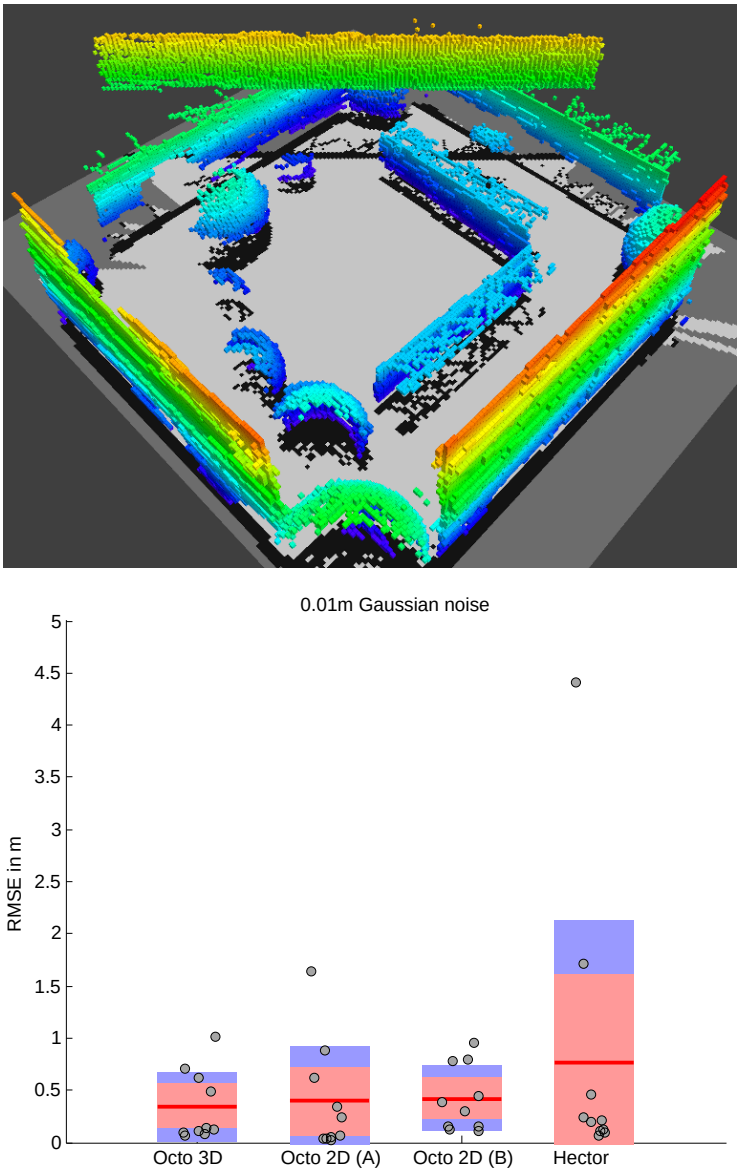


FIGURE 3.14: This figure shows the SLAM results for the "sphere" environment. On the top the 3D map generated by the OctoSLAM 3D median episode can be seen. On the bottom a graph of the RMSEs for SLAM in this environment is shown.

Figure 3.15 shows the results for the "tilted wall" environment. OctoSLAM 3D significantly outperforms the other three SLAM approaches with an RMSE of 0.13m, compared to RMSEs ranging from 0.260m to 0.844m for the 2D map approaches. Hector SLAM scores the worst RMSE average and also has the highest standard deviation. The episode used for mapping has a RMSE of 0.15m.

In general, the results show the same trend as the previous experiments for localization on a given map: 3D map OctoSLAM significantly outperforms the 2D map approaches in the "corridor" and "tilted wall" environment. In the other environments neither approach performs significantly better than the others. It should however be noted that 2D map OctoSLAM does on average perform slightly better in the "Willow" environment, where a 3D map barely provides additional information over a 2D map. Thus, if one can expect such an environment there is no

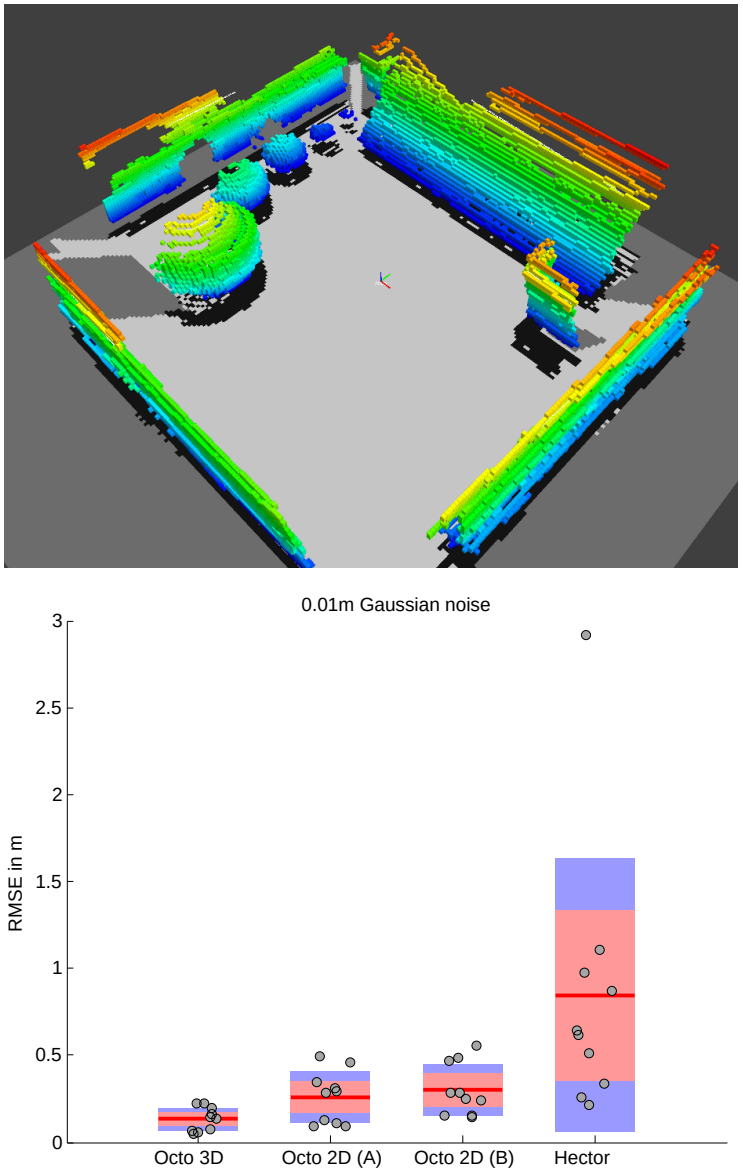


FIGURE 3.15: This figure shows the SLAM results for the "tilted wall" environment. On the top the 3D map generated by the OctoSLAM 3D median episode can be seen. On the bottom a graph of the RMSEs for SLAM in this environment is shown.

reason to use a 3D map SLAM approach. Concerning “constant” vs. “threshold” mapping, i.e. OctoSLAM (A) vs (B), we found that there is no significant difference in performance between the two.

3.3.4 Real Robot Simultaneous Localization and Mapping

The environment for testing SLAM with the UAV is shown in Figure 3.16. It consists of an easy part, in which most elements are straight walls (i.e. upright tables), and a more challenging part, which features elements distinctively different along the z axis.

Figure 3.17 shows the maps generated by 3D map OctoSLAM, 2D map OctoSLAM (B) and Hector SLAM, for a flight through the easy part of the environment. For ease of comparison the 3D maps generated by OctoSLAM are projected onto the ground plane. As indicated by the simulated experiments, all three SLAM approaches perform sufficiently well. Figure 3.18 shows the 3D maps generated by both OctoSLAM variants, where there is barely any difference between the two maps.

FIGURE 3.16: The left picture shows the easy part, the right one the more challenging part. While the easy setting mostly consists of straight walls, many challenging elements are placed in the top half of the right picture.

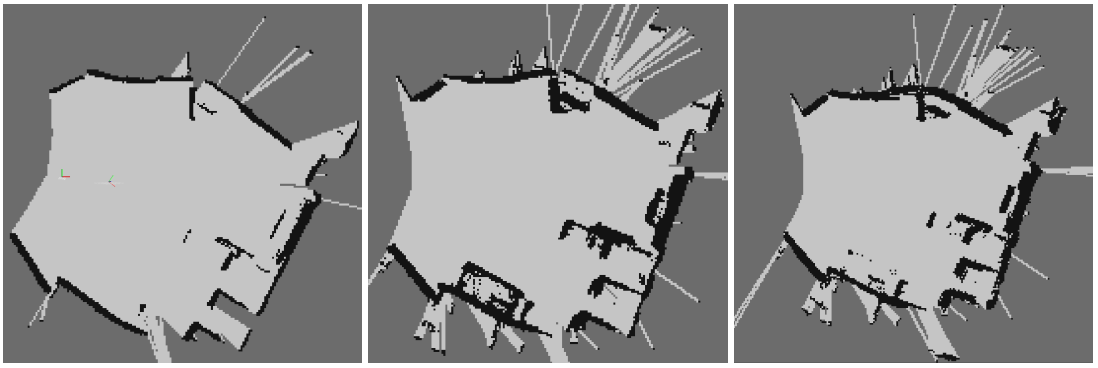


FIGURE 3.17: This figure shows the 2D maps resulting from applying Hector SLAM (left), OctoSLAM 2D (middle) and OctoSLAM 3D (right) in the easy part of the test environment.

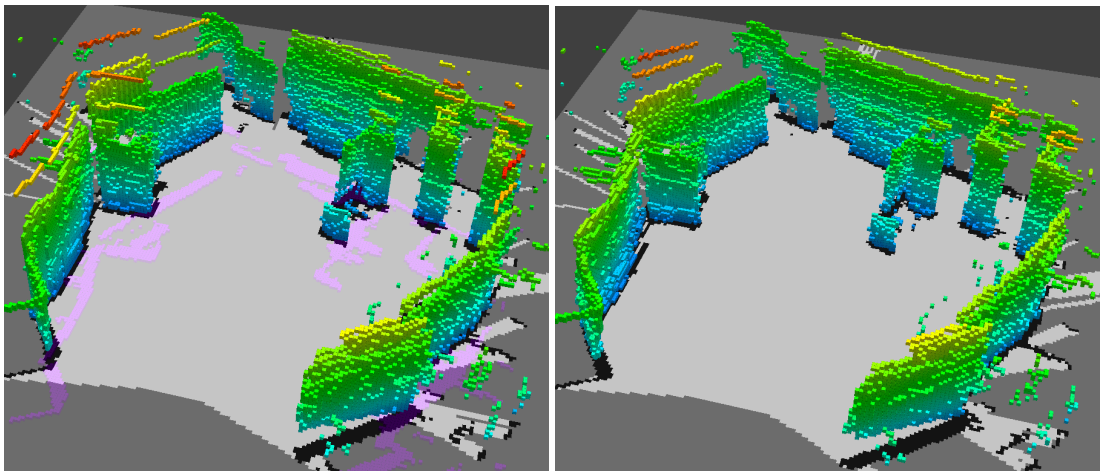


FIGURE 3.18: This figure shows 3D maps of the easy environment, generated by OctoSLAM 2D (left) and 3D (right). Both seem to be of very similar quality.

Moving to the more challenging part of the environments, Figure 3.19 shows the results for SLAM in the entire environment[1]. As Hector SLAM did not produce viable results in this environment, its results have been moved to Appendix A.3. For better comparability to 3D map OctoSLAM, the map for 2D map OctoSLAM was created in 3D, but the localization is performed on a projection of that map onto the ground plane. As can be seen, there are several severe errors in the map created by 2D map OctoSLAM. A good indicator for the low performance is that the right wall has been mapped twice. Partly vertical as it is in the real world, and partly rotated counter clockwise. Additionally, there is quite some horizontal offset, and the wall seperating the upper and lower parts of the environment is mapped very inaccurately. Overall, this map could not be used well, as for example the passage between the easy and challenging part of the environment has been blocked by a wrongly mapped obstacle. In contrast, the map generated by OctoSLAM using a 3D map for pose estimation, looks a lot better overall. While 2D map SLAM has failed here, using a 3D map returns a feasible result. Relatively small features, e.g. the horizontal table legs in the center of the map, can be recognized. The wall to the right is also approximately vertical as in the real world. There is however some horizontal offset, as the lower part seems to be shifted slightly to the left. Nevertheless, this result shows that in certain environments using a 3D map for localization outperforms using a 2D map, making the difference between being able to generate a usable map or not. For closer inspection, both maps from a top-down perspective including the robots trajectory are provided in Appendix A.3.

[1] `https://www.youtube.com/watch?v=XHhbOnLRKIk`

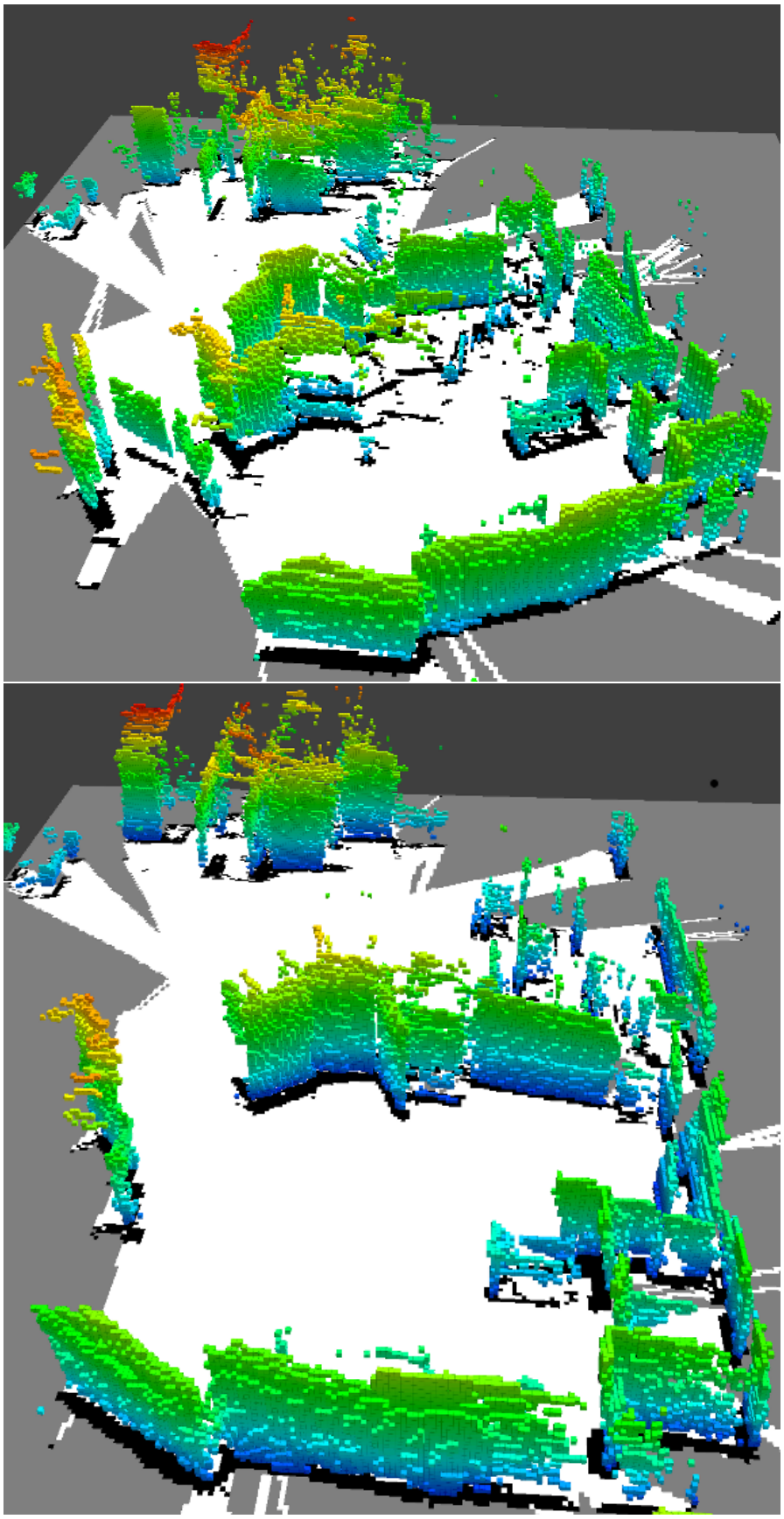


FIGURE 3.19: This figure shows the resulting 3D maps of the complete environment for 2D map (top) and 3D map (bottom) OctoSLAM. While 2D map OctoSLAM produces a 3D map, it only uses the map projected onto the ground plane for registration.

3.4 Conclusions

We have presented OctoSLAM, a novel SLAM approach inspired by octomap and Hector SLAM. OctoSLAM fuses various sensor data, in order to allow for 3D mapping from 2D planar scans and simultaneously performing localization. This requires the use of heuristics which introduce inaccuracy to the localization, which in return decreases the quality of the generated map. Nevertheless, the conducted experiments show that using 3D instead of 2D maps can significantly improve the SLAM performance in the UAV domain, if the environment consists of elements that lose features when viewed from above. In simulation, we found that in environments which tend not to differ over height, 2D map OctoSLAM can perform better than 3D map OctoSLAM, however, not by a significant margin. In contrast, if the environment does not mostly consist of straight walls, 3D map OctoSLAM performs significantly better than 2D map OctoSLAM. The real robot experiments, while not evaluated quantitatively, reinforce these findings. We have not tested OctoSLAM outdoors, but we expect a similar performance if sufficient features are in range (e.g. in urban environments). In case of deployment on open fields however, we expect OctoSLAM to be infeasible, due to a lack of usable features. Overall, the presented experimental results clearly demonstrate the effectiveness and applicability of 3D map OctoSLAM in the indoor UAV domain. The increased performance gained by using 3D maps can enable SLAM in environments where regular 2D map methods fail. We found environments to be infeasible for 2D map SLAM if it is likely that the sensor primarily measures non-orthogonal elements, or if a sufficient number of elements look different over height.

Chapter 4

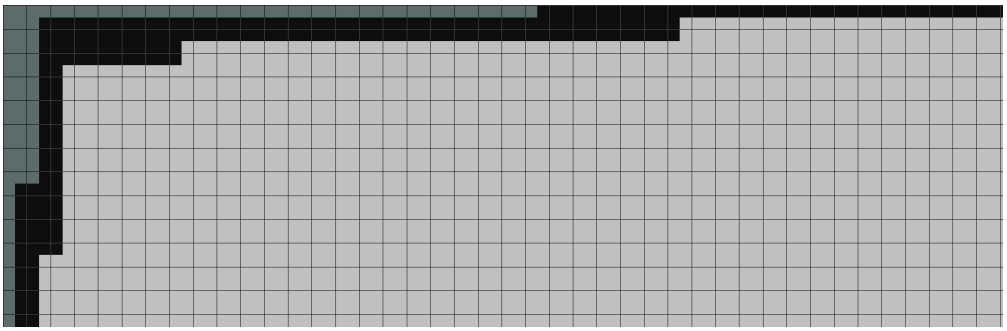
Signed Distance Function based 2D Map 3 DoF registration SLAM

This chapter is based on work published in [76]. We introduce a novel approach to simultaneous localization and mapping for robots equipped with a 2D LiDAR sensor. In particular, we propose a fast scan registration algorithm that operates on 2D maps represented as a signed distance function (SDF). Using SDFs as a map representation has several advantages over existing approaches: while classical 2D scan matchers employ brute-force matching to track the position of the robot, signed distance functions are differentiable on large parts of the map. Consequently, efficient minimization techniques such as Gauss-Newton can be applied to find the minimum error between scan and map. In contrast to occupancy grid maps, the environment can be captured with sub-grid cell size precision, which leads to a higher localization accuracy. Furthermore, SDF maps can be triangulated to polygon maps for efficient storage and transfer. In a series of experiments, conducted both in simulation and on a real physical platform, we demonstrate that SDF tracking is more accurate and efficient than a different Gauss-Newton minimization based approach [2] which uses occupancy grid cell maps. We outperform scan matching on occupancy maps in simulation by ~343% in terms of mean error with a ~63% lower standard deviation. In the real robot experiments, we obtain an average performance advantage of ~10% with a ~25% lower standard deviation.

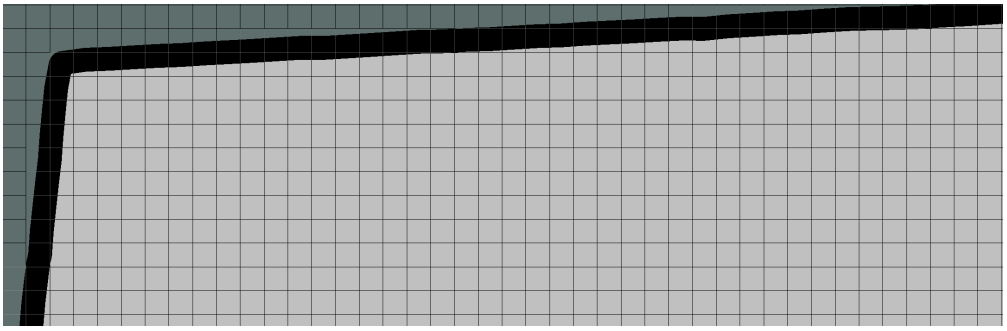
4.1 Introduction

In this chapter we present (i) a scan matching approach based on signed distance functions (SDFs) and (ii) a mapping technique that integrates new laser scans into the map. In combination, this allows us to track the 3 DoF pose of a mobile robot, i.e. horizontal translations and rotation around the vertical axis. We call this approach *2D-SDF-SLAM*.

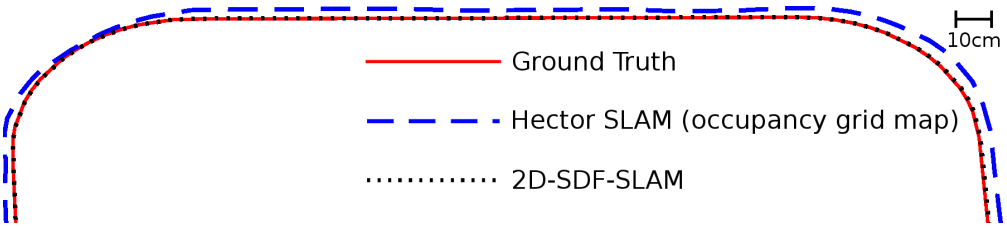
The goal of scan matching is to achieve the best overlap between scan and map by determining a pose update. The novelty of our approach stems from using a SDF to represent the map in conjunction with map gradient based 2D LiDAR sensor registration. SDFs have gained in popularity due to approaches such as KinectFusion [41, 54, 55] targeted at 3D reconstruction



(A) Occupancy grid maps lead to strong discretization.



(B) Same environment as in (a) represented by 2D-SDF-SLAM, i.e. by a signed distance function (SDF) map.



(C) 2D-SDF-SLAM provides more accurate trajectories.

FIGURE 4.1: Instead of representing the map as an occupancy grid map (A), we show in this paper that signed distance functions lead to more accurate SLAM maps (B) and a significantly higher localization accuracy (C).

using depth cameras. We adopt and extend these algorithms for 2D LiDAR data and demonstrate that using SDFs can lead to significantly higher accuracies in comparison to occupancy grid maps. SDFs enable us to capture details of the environment (Figure 4.1b) that otherwise could only be roughly approximated by e.g. commonly used occupancy grid map representations (Figure 4.1a). Figures 4.1a and 4.1b show an example where the discrete occupancy grid map representation is unable to capture fine details. Here, diagonal lines become discretized as "zig-zag" lines, and thus environment information smaller than the map resolution is essentially discarded. In contrast, SDF based maps alleviate these problems to a certain degree. Furthermore, they also inherently allow for approximating more descriptive map gradients, as each cell stores the distance to the next object. These superior map gradient approximations are used to improve the accuracy of the scan registration process (Figure 4.1c).

By representing the map as a signed distance function, we are able to localize the robot more

accurately than by using occupancy grid maps. Although our algorithm is inspired by Kinect-based approaches, we significantly modified it to increase performance with 2D laser scan data. In particular, we developed a novel update scheme that deals better with steep incident angles, which are less frequent in Kinect data. Also, instead of using ICP based registration, we leverage the map gradients provided by SDF based maps for efficient Gauss-Newton minimization. Therefore, no correspondence search for pairing LiDAR measurements with map points is required.

We evaluate the algorithm both in simulation and on a physical platform, and demonstrate that 2D-SDF-SLAM is more accurate than a state-of-the-art occupancy grid based frontend.

The remainder of this chapter is structured as follows: we introduce 2D-SDF-SLAM in Section 4.2 and continue with an empirical evaluation in Section 4.3. We conclude in Section 4.4.

4.2 2D-SDF-SLAM

In this section we explain the workings of the 2D-SDF-SLAM approach. First, we will discuss how a map is created, i.e. how the SDF is updated given a pose estimate and input from a LiDAR sensor. Afterwards, we will present the scan registration algorithm that uses the gradient estimates provided by the SDF map to update the pose estimate. The system is implemented using the Open Source Robotic Foundations Robot Operating System [67].

4.2.1 Mapping

In our approach the map $M = \{m_{xy}\}$ consists of a 2D grid, with the signed distance to the next object being stored in each map cell m_{xy}. Map cells in free space have positive values, while occupied cells have negative values. We assume that the current robot pose $\mathtt{s} = (t^x, t^y, \gamma)^T$ is known, where γ is the rotation around the vertical axis and t^x, t^y are the horizontal translations along x and y axis, respectively. Given a scan $D' = (\mathtt{d}'_0, \mathtt{d}'_1, .., \mathtt{d}'_I)$ that provides us with I scan endpoints $\mathtt{d}'_i = (d_i'^x, d_i'^y)^T$, the map can be updated. To do so all $\mathtt{d}'_i \in D'$ are transformed into the global coordinate system:

$$\begin{pmatrix} d_i^x \\ d_i^y \end{pmatrix} = \mathtt{s} \otimes \mathtt{d}'_i = \begin{pmatrix} \cos\gamma & -\sin\gamma \\ \sin\gamma & \cos\gamma \end{pmatrix} \begin{pmatrix} d_i'^x \\ d_i'^y \end{pmatrix} + \begin{pmatrix} t^x \\ t^y \end{pmatrix}. \tag{4.1}$$

The resulting set of scans is denoted as $D = (\mathtt{d}_0, .., \mathtt{d}_\mathtt{I})$.

A common way to update SDFs, is following the ray from the sensor origin s to all sensor endpoints d_i, where the cells that are intersected by the ray are updated with the positive distance between cell and endpoint. Afterwards, the ray is extended beyond $\mathtt{d}_i$, and cells which are intersected are updated with the negative distance to $\mathtt{d}_i$. However, we do not employ this technique, as especially for wide angle field of view sensors, such as LiDAR sensors, this approach is not optimal: steep incident angles and relatively coarse map resolutions can lead to undesirable map updates. An example for this behavior is given in Figure 4.2a. Here, cells which are both positive and negative are in conflict as they are updated with both positive and negative distances, which do not tend to cancel each other out. The figure also shows that inaccurate map

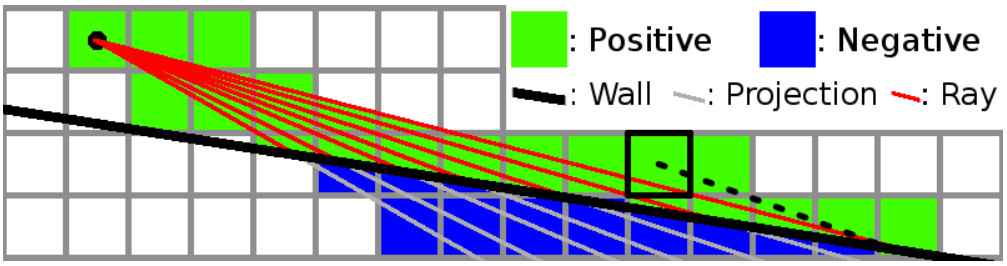
updates can stem from cells being updated with the distance from cell to scan endpoint, while the shortest distance is actually much smaller. For instance, the top/right ray in the example induces highly positive distance updates to cells that are in fact less than a cell size away from the wall, e.g. the cell marked with black borders. Thus, we instead use regression to determine which vertices to upgrade, illustrated in Figure 4.2b. The regression line is used to describe the detected objects outline by a linear function, which is subsequently used to update surrounding cell center points. Thus, we determine a regression line $f(x)$ describing scan endpoints $\mathtt{d}_0$, $\mathtt{d}_1$, and $\mathtt{d}_2$. In addition to m_0, m_1, m_2 and m_3, we also update vertices in the grey area, i.e. if the projection of the vertex onto the regression line fall within the line segment given by $f(x)$ with $x = [2,3]$. Additionally, to be eligible for updates, vertices have to be within a certain distance, denoted by K, of the cell center in which $\mathtt{d_i}$ are located.

The general approach is as follows: we seek a regression line describing all $\mathtt{d_i}$ that are located in the same square $m'_{\lfloor x \rfloor \lfloor y \rfloor}$, i.e. all scan endpoints $\mathtt{d_i}$ for which it holds that $d_i^x > m_0^x$, $d_i^x < m_3^x$, $d_i^y > m_1^y$ and $d_i^y < m_2^y$. Therefore, orthogonal Deming regression [96] is performed on all d_i that fulfill the above conditions. In contrast to simple linear regression, Deming regression accounts for errors on both x-axis and y-axis. The resulting line $f(x) = \beta_0 + \beta_1 x$ minimizes

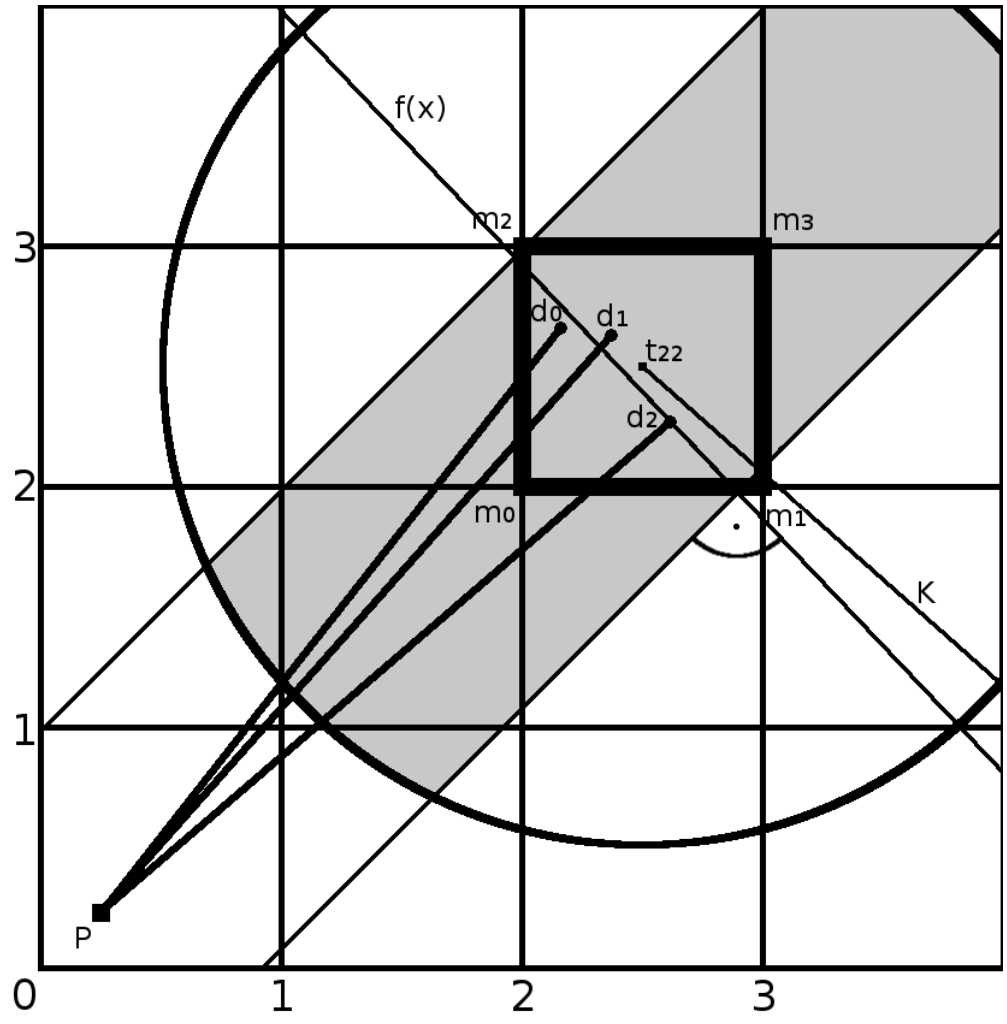
$$\frac{1}{\sigma^2} \sum_{i=1}^{I} \left((d_i^y - \beta_0 - \beta_1 x_i)^2 + (d_i^x - x)^2 \right), \tag{4.2}$$

where σ is the variance of the y error. If there are not enough points available in $m'_{\lfloor x \rfloor \lfloor y \rfloor}$, we either use scan endpoints located in adjacent cells if available, or assume a wall perpendicular to the projection of pose $\mathtt{s}$ onto $f(x)$. We assume both a minimum free space in front, and occupied space behind any detected object in the environment, defined as K. Thus, we update all vertices m_{xy} that are within K distance of $m'_{\lfloor x \rfloor \lfloor y \rfloor}$, and which can be orthogonally projected onto the line segment $f(x)$ with $x = [m_0^x, m_3^x]$. This is visualized by the grey area in Figure 4.2b. If the distance between $\mathtt{s}$ to $f(x)$ along the line given by $\mathtt{s}$ and m_{xy} is larger than the distance between the latter, m_{xy} is updated with the positive point to line distance between m_{xy} and $f(x)$, and with the negative distance otherwise. It follows that the outlines of objects are encoded in the SDF with sub-cell size precision, which in turn is beneficial for the scan registration accuracy.

A common update rule for SDFs is taking the average for all measurements over all time-steps. This way, mapping error induced by Gaussian sensor noise is effectively reduced. However, as every scan not only updates the cell it hits, but also some surrounding cells, averaging at all cell updates can introduce noise to the map, as previously explained in Figure 4.2a. In our approach, averaging within a time-step is already present as we perform regression over multiple scan endpoints. Updating over multiple time-steps is handled by the following heuristic: we introduce a priority level for every cell, which stores the distance to the scan endpoint that induced its update. This distance is used as a quality criteria of the update. We define the priority level $p(x', y')$ as zero for vertices adjacent to the vertices spanning $m'_{\lfloor x \rfloor \lfloor y \rfloor}$, and increase it by one for every further layer of adjacent vertices, where x', y' are the coordinates of the vertex being



(A) Steep incident angles lead to problems in KinectFusion-style map updates. The distance between cells and the point where the corresponding ray intersects with the wall is often larger than the shortest distance between cell and wall. The map cell with black borders for example is updated by the top/right ray. As indicated by the length of the dotted line, the distance stored in that cell will be much larger than the actual distance to the wall.



(B) We locally approximate the scan with a regression line $f(x)$, which leads to consistent map updates even under steep incident angles. Vertices in the gray area are updated based on the scans $\mathtt{d}_i$.

FIGURE 4.2: This figure shows problems of ray based map updates in (a). We present a solution to the problem in (b).

updated and x, y are the coordinates of $m'_{\lfloor x \rfloor \lfloor y \rfloor}$ inducing the update. Thus,

$$p(x',y') = \max(0, \min(\min(|d_0^{\lfloor x \rfloor} - x'|, |d_0^{\lceil x \rceil} - x'|), \min(|d_0^{\lfloor y \rfloor} - y'|, |d_0^{\lceil y \rceil} - y'|))). \tag{4.3}$$

If $p(x',y') = p(x,y)$, measurements from previous time steps are averaged with the current update. In case $p(x',y') < p(x,y)$, the previous distances are discarded. Vice versa, if $p(x',y') > p(x,y)$, the current update is ignored.

4.2.2 Scan Registration

To visualize the proposed registration approach, Figure 4.3 shows a 2D SDF map rendered in 3D. The idea is to use the slope given by the distances stored in the map cells to match LiDAR measurements with zero crossings, i.e. points where the sign of the distances stored in the map changes.

The overall scan registration algorithm we employ consists of two steps: (i) given a pose estimate from the previous time step, $\mathbf{s}'$, and a set of scan endpoints D, determine the map gradient ∇M, and (ii) given ∇M, determine $\Delta\mathbf{s}'$ so that $\mathbf{s}' + \Delta\mathbf{s}$ reduces the alignment error of scan to map. These two steps are repeated until the maximum number of iterations are reached or until the error is sufficiently small.

In order to align laser scans with the existing map, we need to find the robot pose $\mathbf{s}^* = (t^x, t^y, \gamma)^T$ that best aligns the current laser scan $D' = (d'_0, d'_1, .., d'_I)$ with the map M, i.e.,

$$\mathbf{s}^* = \arg\min_{\mathbf{s}} \sum_{i=0}^{I} (M(\mathbf{s} \otimes \mathbf{d}'_i))^2. \tag{4.4}$$

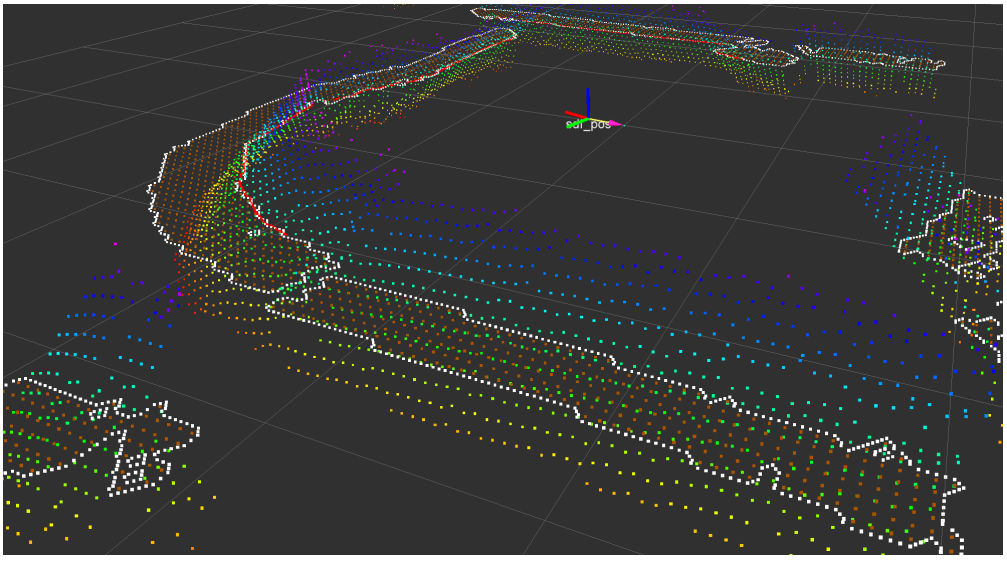


FIGURE 4.3: This figure shows an example for a 2D SDF based map. The red dots mark areas where the distances are negative. The white dots mark the zero crossing, i.e. points where the sign of the distances changes. The other dots represent the distances stored in the cells below/above them, where color encodes height. Blue color indicates a larger distance to walls.

As in Subsection 2.1.3, we tackle this non-linear minimization problem by applying the Gauss-Newton method. Hence, to find the minimum we iteratively linearize the cost function and set its derivative to zero, which requires computing the map gradients. The map gradient approximation is given by the sum of gradients for each scan endpoint, i.e

$$\nabla M = (\sum_{i=0}^{I} \frac{\partial M}{\partial x}(\mathtt{d_i}), \sum_{i=0}^{I} \frac{\partial M}{\partial y}(\mathtt{d_i})). \tag{4.5}$$

Thus we inspect the cells surrounding each scan endpoint d_i, analog to updating the map (Figure 4.2b). For each $d_i = (d_i^x, d_i^y)$ with $i = 0..I$ we inspect the following cells m of size V:

$$m_0^{xy} \; with \; (x = \lfloor d_i^x \rfloor), \; (y = \lfloor d_i^y \rfloor), \tag{4.6}$$
$$m_1^{xy} \; with \; (x = \lfloor d_i^x \rfloor + V), \; (y = \lfloor d_i^y \rfloor), \tag{4.7}$$
$$m_2^{xy} \; with \; (x = \lfloor d_i^x \rfloor + V), \; (y = \lfloor d_i^y + V \rfloor), \tag{4.8}$$
$$m_3^{xy} \; with \; (x = \lfloor d_i^x \rfloor), \; (y = \lfloor d_i^y + V \rfloor). \tag{4.9}$$

Next, we determine if there are zero, two or four sign changes between adjacent cells. Zero sign changes translate to the scan endpoint not being within grid cell size range of an object. In this case, we use the slope given by the distances stored in the cells to approximate the map gradient by linear interpolation:

$$\frac{\partial M}{\partial x}(\mathtt{d_i}) \approx y \cdot (m_2 - m_3) + (V - y) \cdot (m_1 - m_0), \tag{4.10}$$
$$\frac{\partial M}{\partial y}(\mathtt{d_i}) \approx x \cdot (m_2 - m_1) + (V - x) \cdot (m_3 - m_0), \tag{4.11}$$

with $x = d_i^x - m_0^x$ and $y = d_i^y - m_0^y$. The map value for d_i is approximated in a similar manner:

$$M(d_i) \approx |y \cdot (m_2 \cdot x + m_3 \cdot (V - x)) + (V - y) \cdot (m_0 \cdot x + m_1 \cdot (1 - x))| \tag{4.12}$$

If there are two sign changes on the other hand, an object is within grid cell range of the scan endpoint. In this case we retrieve the linear function $g(r)$ describing the object as follows. Let m_{j+} and m_{j-} be the pairs of map cells with $m_{j+} > 0$ and $m_{j-} \leq 0$. We determine the points $\mathtt{p_0}, \mathtt{p_1} = (x_j, y_j)^T$ that define $g(r) = \mathtt{p_0} + r(\mathtt{p_1} - \mathtt{p_0})$:

$$\mathtt{p_j} = \begin{pmatrix} m_{j+}^x \\ m_{j+}^y \end{pmatrix} + (\frac{m_{j+}}{m_{j+} - m_{j-}}) \begin{pmatrix} m_{j-}^x - m_{j+}^x \\ m_{j-}^y - m_{j+}^y \end{pmatrix}. \tag{4.13}$$

The map gradient approximation in x and y direction is the distance along the respective axis between scan endpoint $\mathtt{d_i}$ and its orthogonal projection $\mathtt{q}$ onto $g(r)$:

$$\mathtt{q} = \mathtt{p_0} + \frac{(\mathtt{d_i} - \mathtt{p_0})(\mathtt{p_1} - \mathtt{p_0})}{(\mathtt{p_1} - \mathtt{p_0})^2} (\mathtt{p_1} - \mathtt{p_0}), \tag{4.14}$$

and

$$\begin{pmatrix} \frac{\partial M}{\partial x}(\mathtt{d_i}) \\ \frac{\partial M}{\partial y}(\mathtt{d_i}) \end{pmatrix} \approx \mathtt{q} - \mathtt{d_i}. \tag{4.15}$$

The map value is approximated by the shortest distance between $\mathtt{d_i}$ and $\mathtt{q}$:

$$M(d_i) \approx |\mathtt{q} - \mathtt{d_i}|. \tag{4.16}$$

We can now proceed with registering the scan to the map, which translates to finding a pose update Δs that minimizes the offset between map and current scan. Therefore, following the approach proposed in [2, 54] we aim to satisfy

$$\sum_{i=1}^{I} [M((\mathtt{s}' + \Delta\mathtt{s}') \otimes \mathtt{d}_i')]^2 \to 0. \tag{4.17}$$

To introduce map gradients to the previous equation, we apply a first order Taylor expansion developing around the scaled distance value $M(\mathtt{d}_i' \otimes (\mathtt{s}' + \Delta\mathtt{s}'))$:

$$\left[M(\mathtt{d}_i^*) - \nabla M(\mathtt{d}_i^*) \frac{\partial(\mathtt{d}_i^*)}{\partial \mathtt{s}'} \Delta\mathtt{s}' \right]^2, \tag{4.18}$$

with $\mathtt{d}_i^* = \mathtt{s}' \otimes \mathtt{d}_i'$. Seeking the minimum alignment error, we set the partial derivative with respect to $\Delta\mathtt{s}'$ to zero:

$$0 = \sum_{i=1}^{N} \left[\nabla M(\mathtt{d}_i^*) \frac{\partial(\mathtt{d}_i^*)}{\partial \mathtt{s}'} \right]' + \left[M(\mathtt{d}_i^*) - \nabla M(\mathtt{d}_i^*) \frac{\partial(\mathtt{d}_i^*)}{\partial \mathtt{s}'} \Delta\mathtt{s}' \right]. \tag{4.19}$$

To determine $\Delta\mathtt{s}'$, Equation 4.19 is reformulated to the following Gauss-Newton equation that describes the minimization problem:

$$\Delta\mathtt{s} = \mathbf{H}^{-1} \sum_{i=1}^{N} \left[\nabla M(\mathtt{d}_i^*) \frac{\partial(\mathtt{d}_i^*)}{\partial \mathtt{s}'} \right]' \cdot [M(\mathtt{d}_i^*)] \tag{4.20}$$

with

$$\mathbf{H} = \left[\nabla M(\mathtt{d}_i^*) \frac{\partial(\mathtt{d}_i^*)}{\partial \mathtt{s}'} \right]' \cdot \left[\nabla M(\mathtt{d}_i^*) \frac{\partial(\mathtt{d}_i^*)}{\partial \mathtt{s}'} \right]. \tag{4.21}$$

We then evaluate $\frac{\partial(d_i^*)}{\partial T}$ using Equation 4.1:

$$\frac{\partial(\mathtt{d}_i^*)}{\partial \mathtt{s}'} = \begin{pmatrix} 1 & 0 & -\sin(\gamma) d_i^{*x} & -\cos(\gamma) d_i^{*y} \\ 0 & 1 & cos(\gamma) d_i^{*x} & -sin(\gamma) d_i^{*y} \end{pmatrix}, \tag{4.22}$$

thus determining the pose update $\Delta\mathtt{s}'$.

4.3 Empirical Evaluation

We evaluate 2D-SDF-SLAM both in simulation and on a ground based KUKA youBot robot equipped with a 2D LiDAR sensor. The goal of our experiments is (i) to verify that 2D-SDF-SLAM reliably provides accurate pose estimates, and (ii) to show that 2D-SDF-SLAM generates accurate maps.

For the simulated experiments we use the *Simple Two Dimensional Robot Simulator*[1] (STDR) in combination with a simple random walk algorithm. In contrast to Gazebo, the 3D simulator used for the OctoSLAM evaluation in Section 3.3, the STDR simulator is limited to 2D maps. However, as 2D-SDF-SLAM is designed for ground based robots equipped with a rigid 2D LiDAR sensor, a 2D map simulator suffices. Furthermore, using a 3D simulator leads to unnecessary overhead in terms of computational requirements and ease of use. We forgo comparing 2D-SDF-SLAM with OctoSLAM, because OctoSLAM utilizes attitude changes along the roll and/or pitch axis, which do not occur on ground based robots. To evaluate the real world experiments we gather ground truth data using an *Optitrack*[2] motion capture system. To compare 2D-SDF-SLAM to a similar frontend we perform experiments for both 2D-SDF-SLAM and Hector SLAM and compare the results.

To determine the absolute trajectory error (ATE) we use the method of Besl et al. [4] to align the SLAM trajectory with the ground truth trajectory. This way, error induced by inconsistent initial poses, for instanced caused by an offset between motion capture marker and sensor, is reduced. Additionally, we believe that this increases the quality of the error measure, as static offsets between sensor and map origin do not decrease the quality of the map. Both the aligned SLAM and ground truth trajectory are then used to transform all scan endpoints collected during an episode. We use the average Euclidean distance between the sets of transformed scan endpoints to measure the SLAM performance for each episode.

The map update thresholds are set to 0.4 meter for distance and 0.9 radians for rotation, and the map resolution is set to 0.05 meter grid cell size for both SLAM approaches.

To give a notion on runtime behavior: one Hector SLAM registration iteration takes approximately 0.0021 seconds using a i7-4770 3.4GHZ CPU in the experiments, compared to 0.0023 seconds per iteration for 2D-SDF-SLAM.

4.3.1 Simulated Experiments

We use the STDR simulator to roll out the simulated experiments. The simulated robot performs a simple random walk which consists of repeating (i) driving forward until hitting a wall, and (ii) turning in an arbitrary direction for 0.5 to 1.5 seconds. The translational velocity is set to 0.5 meter per second, and the turning speed to 0.5 radians per second. Each episode continues for a duration of 300 seconds, and begins with the last robot pose of the previous episode. We perform 190 episodes on a simple map shown in Figure 4.4a. The map size is 10x10 meter, and

[1] `http://wiki.ros.org/stdr_simulator`
[2] `http://optitrack.com`

TABLE 4.1: Performance metrics in meter for simulated and real robot SLAM experiments.

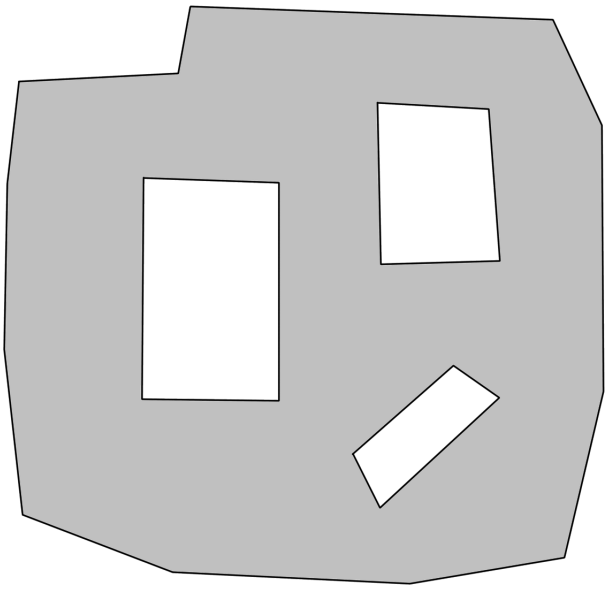
		Median	Mean	σ
Sim.	Hector SLAM	0.0250	0.0310	0.0180
	2D-SDF-SLAM	0.0057	0.0070	0.0067
	% relative change	-338.6	-342.9	-62.9
Real	Hector SLAM	0.0399	0.0339	0.0188
	2D-SDF-SLAM	0.0350	0.0307	0.0141
	% relative change	-12.2	-9.5	-24.6

the simulated LiDAR sensor provides 667 rays per scan with a range of 0.05 to 5.6 meter at 10 Hz.

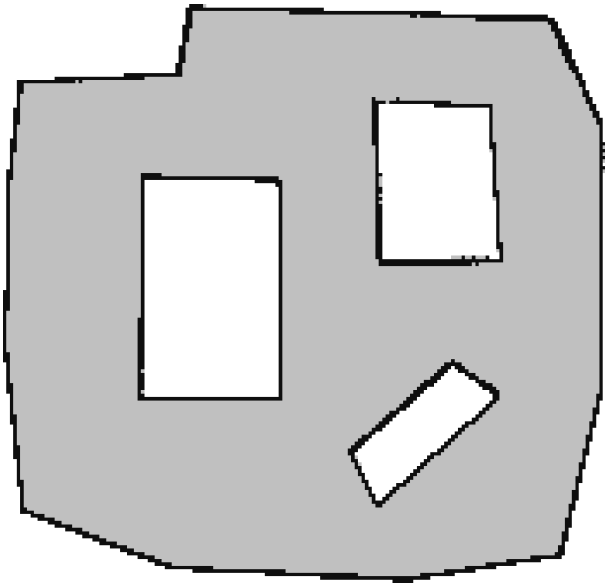
The ground truth aligned trajectories for SDF and Hector SLAM, as well as the ground truth trajectory for a simulator episode are shown in Figure 4.5. Note that in this example the simulated robot does not follow a random route through the environment, as in every other episode. Instead it is manually teleoperated to achieve a visually pleasing result. The mean error in this episode is ~0.0039 meter for 2D-SDF-SLAM, versus ~0.0290 meter for Hector SLAM. This means that 2D-SDF-SLAM yields a ~87% lower trajectory error than Hector SLAM. The error of Hector SLAM and the ground truth trajectory is also clearly visible in Figure 4.5, while the 2D-SDF-SLAM trajectory precisely matches the true trajectory.

Figure 4.4b gives the corresponding map generated by Hector SLAM. The limitations of occupancy grid maps can be clearly seen here, as walls which are not parallel to coordinate axis are mapped as either straight or as ”zig-zag” lines. The black blocks representing the walls are of grid cell size, visualizing the limited accuracy of map values and gradients obtained from such a map, which in turn limits the accuracy of the scan registration algorithm. With SDFs on the other hand, walls can be encoded with sub grid cell size precision as shown in Figure 4.4c. For ease of comparison, the walls are drawn with a grid cell sized line while they can be visualized in any desired resolution.

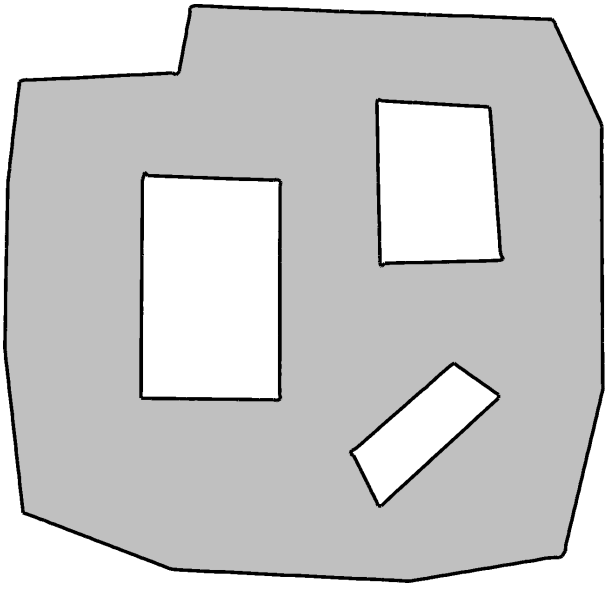
Figure 4.9a shows a boxplot of the overall results for of all 190 episodes, where the central marker represents the median. The box borders represent the 25th-75th percentiles, and the whiskers the range in which data points are not considered outliers. Outliers are plotted individually and do not influence the median indicated in the boxplot. The box plot shows that in simulation 2D-SDF-SLAM clearly performs better and more consistently than Hector SLAM, as is reinforced by the error metrics given in Tab 4.1. 2D-SDF-SLAM outperforms Hector SLAM by ~343% in terms of mean error with a ~63% lower standard deviation of the mean error over all runs. In absolute numbers, 2D-SDF-SLAM achieves an average error of 0.006m, i.e. a magnitude lower than the used map resolution of 0.05m.



(A) Original map.



(B) Hector SLAM map.



(C) 2D-SDF-SLAM map.

FIGURE 4.4: Top down view of the original simulator map, as well as maps generated by 2D-SDF-SLAM and Hector SLAM (both at a resolution of 0.05m).

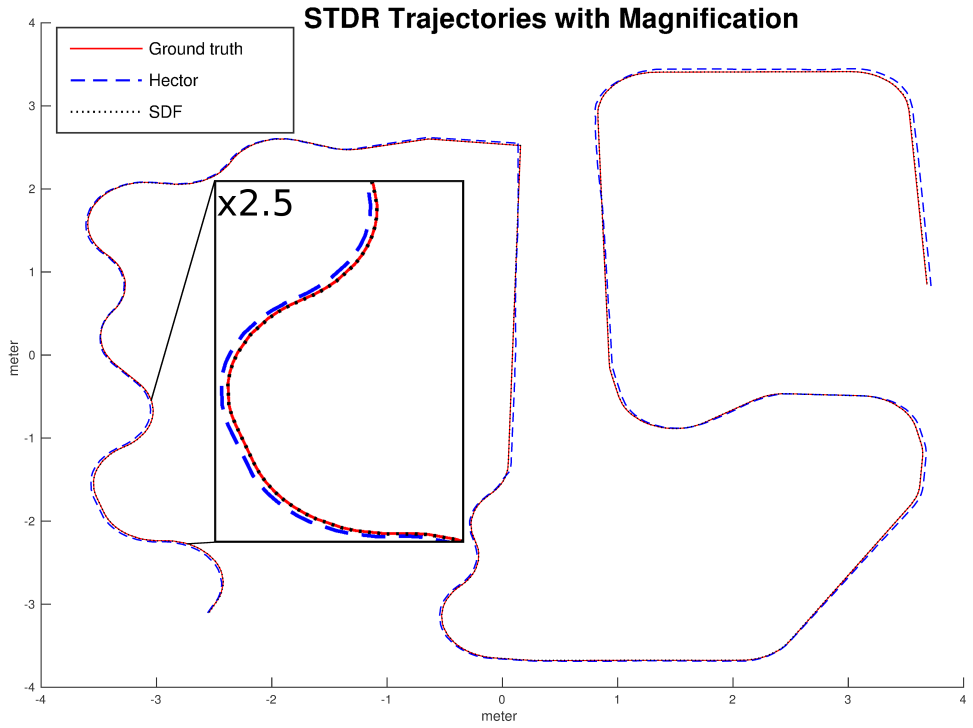


FIGURE 4.5: Ground truth, Hector and 2D-SDF-SLAM trajectories for a simulated episode. The robot is controlled manually.

4.3.2 Real Robot Experiments

For the real robot experiments we use a Hokuyo URG-04LX-UG01 LiDAR sensor mounted on a teleoperated, omnidirectional KuKa youBot. The LiDAR sensor has a field of view of 240 degrees, provides 667 scan endpoints at 10 Hz and is rated with an accuracy of ± 0.03 meters below 1 meter and $\pm 3\%$ otherwise, and thus belongs to the low-end in terms of accuracy, update rate and range. The angular velocity of the robot is limited to 0.5 rad/s, and the translational velocity to 1 m/s. We perform 42 episodes, each approximately 60 seconds long in a roughly 5x5 meter area. The environment used for the experiments and the youBot are shown in Figure 4.6[3]. Only the central laser sensor on the youBot has been used in the experiments. It is elevated so that it can detect the desks on the right, and to prevent occlusion of the motion capture markers that have to be visible to the cameras mounted at the ceiling to guarantee smooth ground truth trajectories. The wall segments in front of the desks in the middle of the room serve to reduce interference from human factors.

The aligned trajectories for a real robot episode are shown in Figure 4.7. In this episode the mean error for 2D-SDF-SLAM is ~0.0290 meter, and ~0.0321 meter for Hector SLAM, i.e. 2D-SDF-SLAM outperforms Hector SLAM by ~10%.

The corresponding maps generated by both SLAM approaches are given in Figure 4.8. As expected, both maps contain more noise than in simulation. Especially the complex structure

[3] `https://www.youtube.com/watch?v=j1vs0sUXAQc`

FIGURE 4.6: A photo of the environment the real robot experiments take place in, including motion capturing system and youBot.

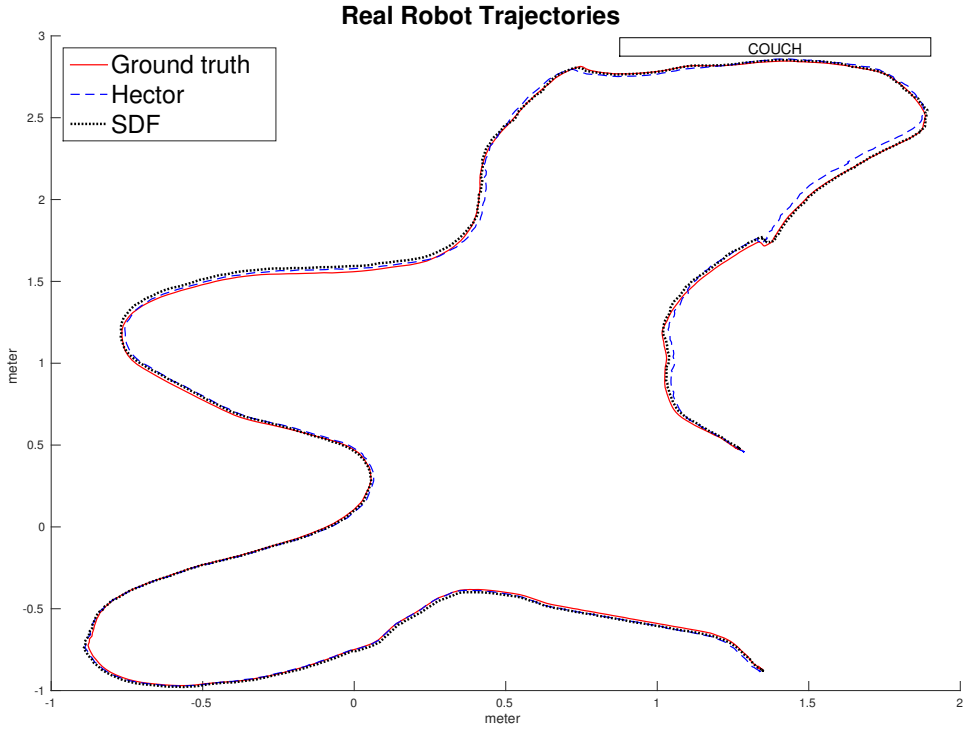
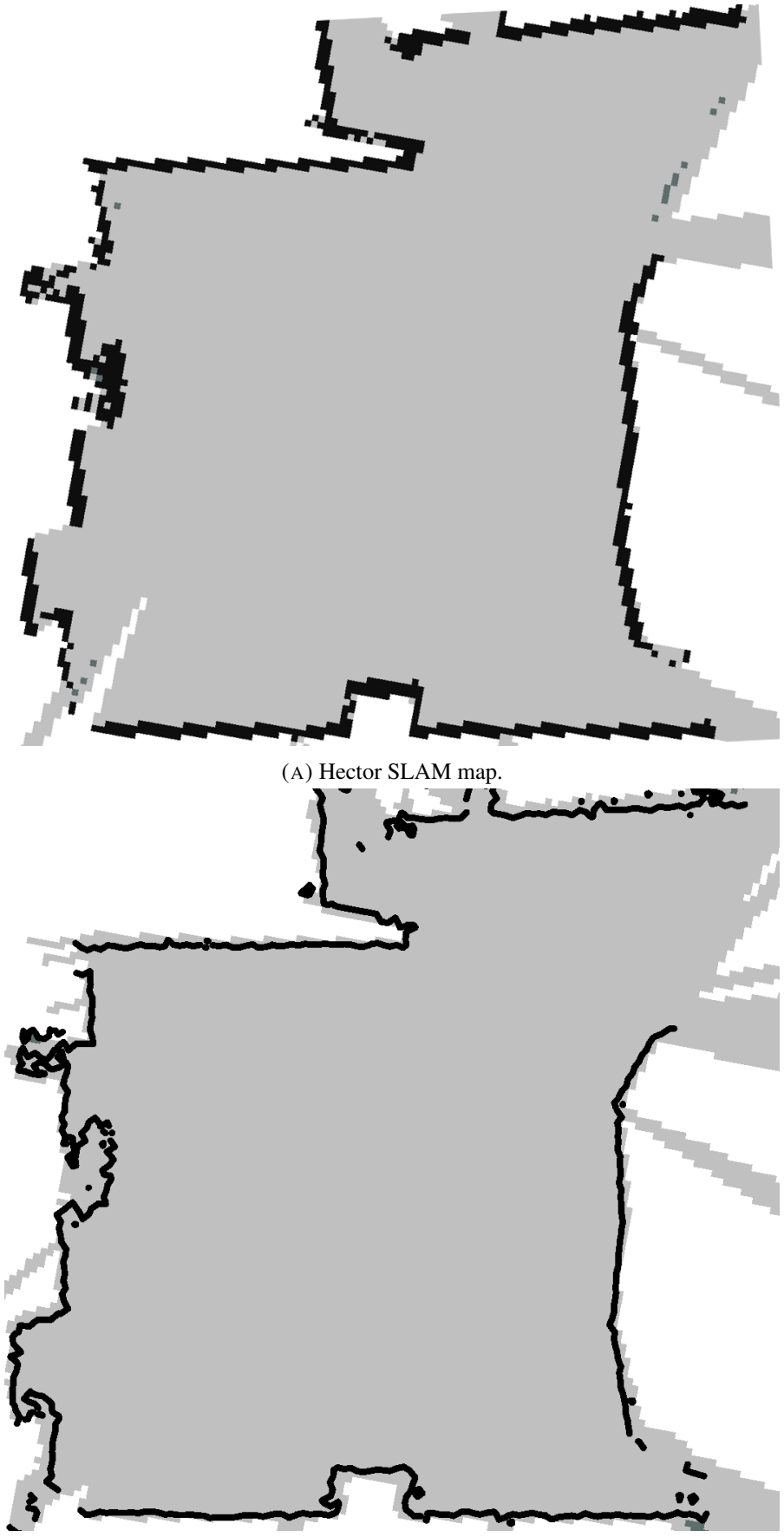


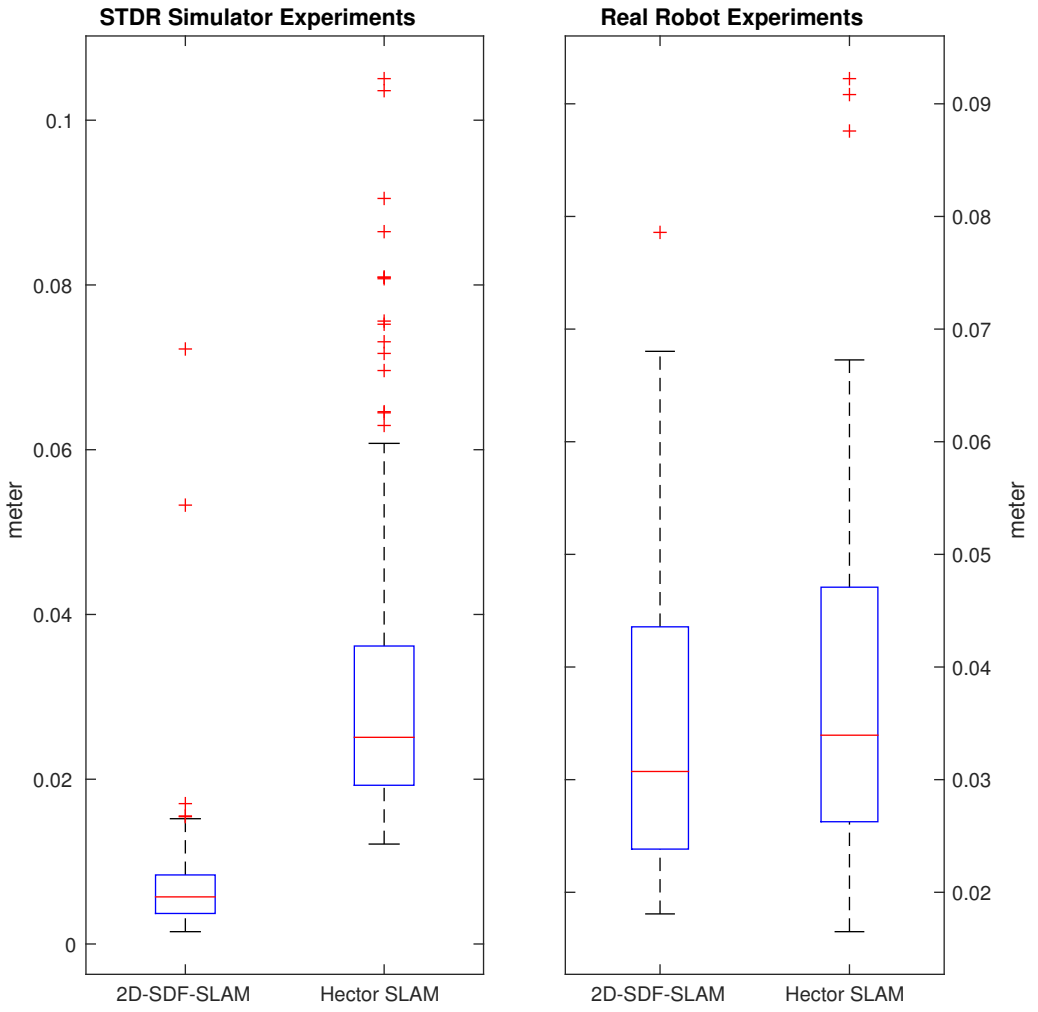
FIGURE 4.7: Visualization of the ground truth, Hector and 2D-SDF-SLAM trajectories for one of the 42 real robot episodes. In this episode 2D-SDF-SLAM outperforms Hector by ~10%.



(A) Hector SLAM map.

(B) 2D-SDF-SLAM map.

FIGURE 4.8: Maps generated by (a) Hector SLAM and (b) 2D-SDF-SLAM for one of the 42 real robot episodes.



(A) 190 simulated episodes. (B) 42 real robot episodes.

FIGURE 4.9: Boxplots of the average error for (a) 190 simulated episodes and (b) 42 real robot episodes. In simulation 2D-SDF-SLAM outperformed Hector SLAM by a ~338.6% lower median and a ~62.9% lower standard deviation. On the real robot 2D-SDF-SLAM outperformed Hector SLAM by a ~12.2% lower median and a 24.6% lower variance. We believe the reduced performance of 2D-SDF-SLAM in (b) is caused by the (lack of) accuracy of the LiDAR sensor used, which is rated ± 0.03m for measurements below 1m, and $\pm 3\%$ otherwise.

of the open radiator ribs on the left side of the map are hard to capture for both approaches. Similarly to the simulation experiments, 2D-SDF-SLAM is able to create a finer grained map of the environment.

Figure 4.9b shows a boxplot for the real robot experiments. Again, 2D-SDF-SLAM achieves a lower median and a more consistent performance than Hector SLAM. More results are given in Tab. 4.1: 2D-SDF-SLAM outperforms Hector SLAM by ~9.5% in terms of mean error with a ~24.6% lower standard deviation. In absolute numbers, 2D-SDF-SLAM achieves a mean error of 0.0307m, and 0.0339m for Hector SLAM. We assume that the lower performance advantage of 2D-SDF-SLAM over Hector SLAM in the real robot experiments results from the inaccuracy of the Hokuyo URG-04LX-UG01.

4.4 Conclusions

We have presented 2D-SDF-SLAM, a novel approach for 2D LiDAR sensor based SLAM with signed distance function based maps. Our approach has been inspired by recent techniques proposed for depth cameras and 3D reconstruction, but includes major modifications to adapt it to laser scan data. We adopted these algorithms for robots equipped with 2D LiDAR sensors and proposed a corresponding SLAM frontend. Using our implementation, we showed in a series of experiments on real and simulated robots that our proposed 2D-SDF-SLAM frontend outperforms a comparable SLAM system based on occupancy grid maps. In particular, we achieve a ~63% lower standard deviation combined with a ~343% better mean error in simulation, and ~25% lower standard deviation with a ~10% better mean error on the real robot. In absolute numbers, 2D-SDF-SLAM averages a 0.024m smaller error in simulation, and a 0.0032m smaller error in the real robot experiments. Regarding the significance of these improvements: a 2.4cm smaller error can make the difference between whether a choke point is traversable or not, especially on smaller robots. However, it can be argued that the 0.32cm smaller error achieved in the real robot experiments is insignificant. Nevertheless, 2D-SDF-SLAM also does not require significantly more resources, and more precise LiDAR sensors may push the real world results towards the simulated ones.

Chapter 5

Octree based 3D map 6 DoF registration SLAM

This chapter is based on work published in [77]. We present a SLAM front end for simultaneous 6 DoF localization and 3D mapping using 3D LiDAR sensors mounted on airborne robots, called *NOctoSLAM*. The approach adopts an octree based map representation that implicitly enables source and reference data association for point-to-plane ICP registration. Additionally, the data structure is used to group map points to approximate surface normals. The multi-resolution capability of octrees, achieved by aggregating information in parent nodes, enables us to compensate for spatially unbalanced sensor data typically provided by multi-line LiDAR sensors. The octree based data association is only approximate, but our empirical evaluation shows that NOctoSLAM achieves the same pose estimation accuracy as a comparable, point cloud based approach. However, NOctoSLAM can perform twice as many registration iterations per time unit. In contrast to point cloud based surface normal maps, where the map update duration depends on the current map size, we achieve a constant map update duration including surface normal recalculation. Therefore, NOctoSLAM does not require elaborate and environment dependent data filters. The results of our experiments show a mean positional error of 0.029m and 0.019 rad, with a low standard deviation of 0.005m and 0.006 rad, outperforming the state-of-the-art by remaining accurate while running online in near real-time.

5.1 Introduction

Mobile robots are commonly equipped with 2D LiDAR sensors motivated by high precision and affordable price. Recently, 3D LiDAR sensors have been getting smaller and more affordable. This trend might amplify in the near future, due to increased demand from the automotive industry and the introduction of solid state 3D LiDAR sensors.

Currently, most 3D LiDAR sensors provide multiple 2D scans at different inclinations. Unlike 2D LiDARs, 3D LiDAR sensors allow for 6D pose estimation and 3D mapping without requiring additional sensors, e.g. inertial measurement units or altitude sensors. However, handling the large amount of data typically provided by such sensors can prove challenging. In

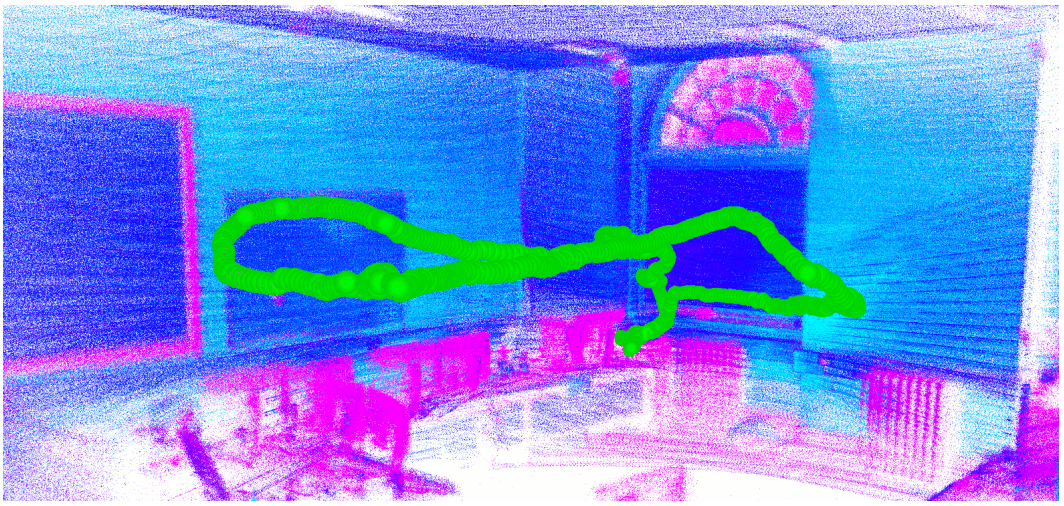


FIGURE 5.1: This figure shows a NOctoSLAM generated 3D trajectory. The pose estimates are used to transform the sensor data, where color represents scan intensity. Recognizable details such as the whiteboard, radiators and the beams in the arched window indicate a good pose estimation accuracy.

order to register 3D scans online and in near real-time, efficient scan matching algorithms and map representations are required. Figure 5.1 provides an example for 3D LiDAR based SLAM. In the example, the sensor was only moved along a relatively short trajectory. Nevertheless, the amount of sensor readings collected is sufficient to generate a detailed representation of the environment.

The main contribution of this chapter is the introduction of NOctoSLAM, a 6 DoF registration SLAM front-end which uses an octree based 3D map representation. The octree data structure is used for fast nearest neighbor approximation, enabling efficient surface normal generation, as well as correspondence search. For registration, NOctoSLAM uses the Iterative Closest Point [4] algorithm with a point-to-plane error metric [42]. With NOctoSLAM, we present a robust and computationally slim alternative to established scan matching algorithms that operate on point cloud based maps. Due to its computational performance, it does not require explicit data filtering, and thus also voids the need for manual fine-tuning of filter parameters.

Our empirical evaluation illustrates that NOctoSLAM computationally outperforms libpointmatcher [22], a highly optimized SLAM approach for multi-line LiDAR sensors, while matching its accuracy. The computational performance of SLAM front ends is important as it has an impact on the latency between acquiring sensor data and updating the pose estimate. Furthermore, better computational performance can increase the update rate, and also determines how much of the sensor data can be processed instead of filtered. Adjusting filter parameters of libpointmatcher in order to enable near real-time processing, unavoidably reduces accuracy. Thus, in an attempt to achieve the same runtime as NOctoSLAM, applying source and reference point filters to speed up libpointmatcher resulted in a positional median error of ~0.09 m, compared to ~0.03 m for NOctoSLAM.

The remainder of this chapter is organized as follows. Section 5.2 introduces NOctoSLAM and specifies the functionality in detail. Section 5.3 describes the experiments performed in

order to compare NOctoSLAM to other approaches. Results are discussed and a conclusion is presented in Section 5.4.

5.2 NOctoSLAM

We propose NOctoSLAM, a 6 DoF 3D map SLAM front-end that uses an octree based map representation which enables efficient point-to-plane scan registration. NOctoSLAM is implemented using the Robot Operating System [67].

Iterative Closest Point (ICP) in the point-to-plane variant requires both source/input to reference/map point association and a surface normal estimate for the reference points. The proposed map structure inherently provides surface normal approximations and allows for direct association of source point cloud to reference map. Thus, we avoid having to maintain additional data structures that perform nearest neighbor search for said tasks. In comparison, point-to-plane ICP [42] and generalized ICP [44] based algorithms, e.g. [22, 97], use kd-trees for data association and surface normal estimation, whilst storing the map in an unorganized point cloud format. While there exist highly optimized approaches to tackle the k-NN problem, such as libnabo [52], kd-tree based nearest neighbor search in large point cloud based maps is not feasible online (see Section 5.3). To achieve registration and map updates that are faster than the sensor update rate, typically requires reducing the number of points by filtering out large amounts of the provided scan endpoints. The nearest neighbor approximation method proposed in this chapter is very fast and allows us to map and register 300,000 points per second, whilst being sufficiently accurate to generate valid surface normals. Hence, NOctoSLAM does not require explicit filtering of sensor data as other point-to-plane ICP based methods do. Tuning such filters can be challenging, environment dependent, and also costly in terms of runtime.

In the following subsections we will explain how (i) updating the map, i.e. scan end point insertion and surface normal approximation, and (ii) registration, i.e. data association and pose updates, are performed in NOctoSLAM.

5.2.1 Mapping

To represent the environment, we extend the octomap approach introduced in [40]. In the octomap approach, every octree node represents a voxel, where its resolution depends on the node's level in the tree. Commonly, every octree node stores the probability of representing occupied space, while the position it represents in 3D space depends on the nodes position in the tree. In NOctoSLAM we additionally store two 3D vectors per node, i.e., a surface normal and a position that can deviate from the center of the voxel. The former is necessary for point-to-plane scan registration. The latter allows for a more precise map representation at coarse map resolutions. Since NOctoSLAM uses both data stored in leaf and non-leaf nodes, such a position anchor is required.

Updating the octree map $M = \{m_{j,k}\}$, where $m_{j,k}$ is octree node number j at level k, consists of two steps: insertion and propagation. Figure 5.2 gives an example in which four points are added to the map. To insert measurements *p1*, *p2*, *p3* and *p4*, the octree is traversed from root

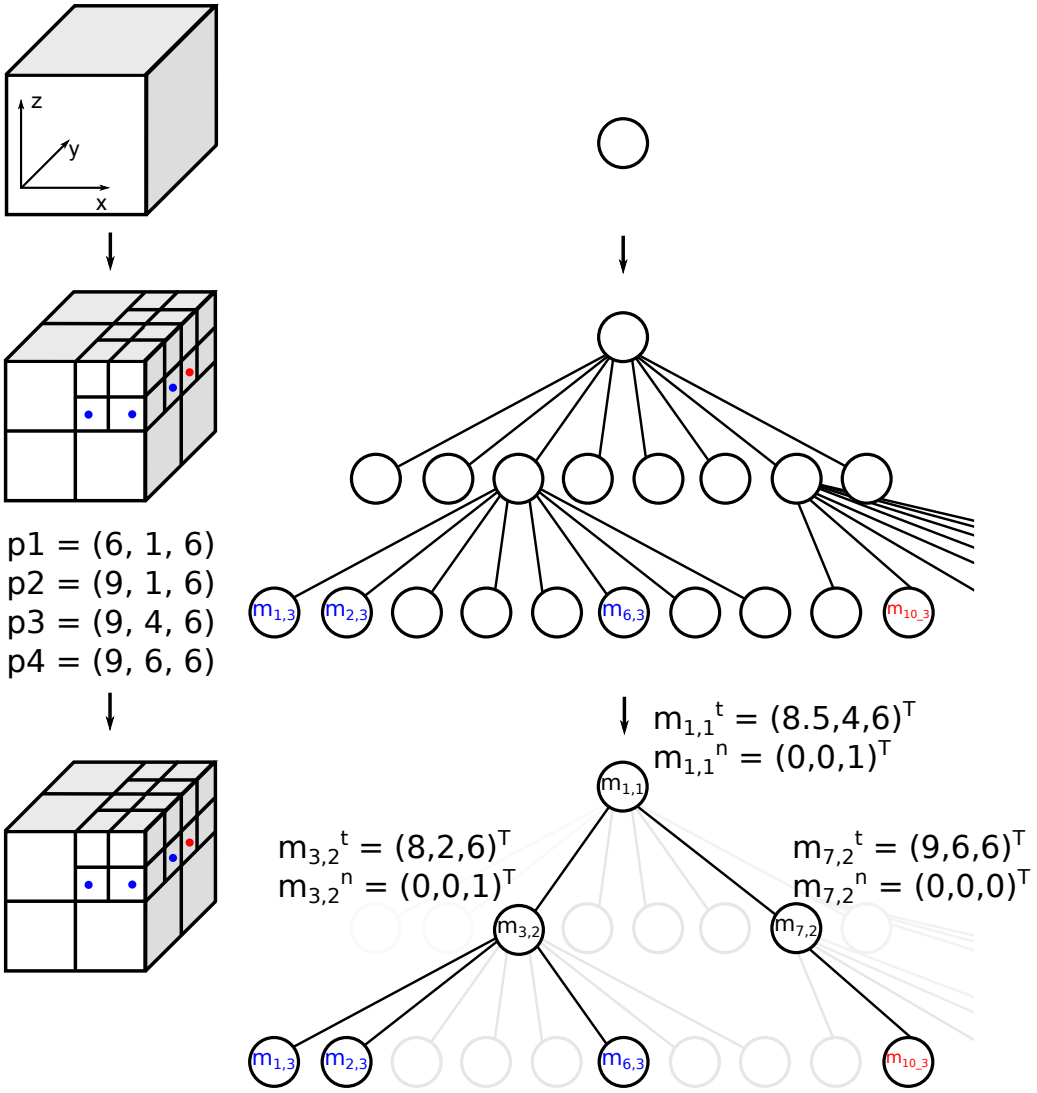


FIGURE 5.2: This figure shows a simplified map update. First, the initially empty octree is extended to accommodate new data points. After inserting the data points into the tree, the changes are propagated through the tree from bottom to top. During propagation surface unit normals $\mathtt{m}^n_{j,k}$ are calculated if feasible ($m^n_{3,2}$, $m^n_{1,1}$). If not, they are set to zero ($m^n_{7,2}$). At the same time, the translation of non leaf nodes ($m_{1,1}4$, $m_{3,2}4$, $m_{7,2}4$) is updated by the centroid of their descendant nodes.

node towards the corresponding leaf nodes. During traversal to the leaf nodes, which are located at maximum tree depth $|M|$, the tree is extended if necessary. Finally, measurements *p1*, *p2*, *p3* and *p4* are inserted into the corresponding nodes $m_{1,3}$, $m_{2,3}$, $m_{6,3}$ and $m_{10,3}$, respectively. Each node $m_{j,k} = (m^p_{j,k}, \mathtt{m}^t_{j,k}, \mathtt{m}^n_{j,k})$ stores an occupancy value $m^p_{j,k}$, a translation $\mathtt{m}^t_{j,k} = (t^x, t^y, t^z)^T$, and a unit surface normal $\mathtt{m}^n_{j,k} = (s^x, s^y, s^z)^T$. Thus, $\mathtt{m}^t_{1,3} = p1$ and $\mathtt{m}^t_{2,3} = p2$ etc. pp. If a leaf node $m_i = m_{i,|M|}$ already exists, the stored position is updated with the new measurement by using the weighted average according to a sensor model Z and node occupancy probability m^p_i:

$$\mathtt{m}^t_i = \frac{\mathtt{m}^t_i * m^p_i + Z * \mathtt{d}_k}{Z + m^p_i}, \tag{5.1}$$

where vector $\mathtt{m}_i^t$ is the position previously stored in node m_i and vector $\mathtt{d}_k = (d_k^x, d_k^y, d_k^z)^T$ is a 3D LiDAR sensor measurement.

After having inserted all scan endpoints in this fashion, the updates are propagated through the tree: all non-leaf nodes traversed during the first step are updated from bottom to top, based on the information stored in their descendants. In particular, parent nodes store the average position of their descendants, and use these positions to approximate a surface normal if feasible (node $m_{3,2}$ in Figure 5.2). We approximate the surface normal by estimating the normal of a plane tangent to the surface, which can be formulated as a least-square problem. As shown in [98] the solution can be reduced to a principal component analysis of the covariance matrix $\mathbf{C}$ generated from the direct descendants $m_{k,j+1}$ of a node $m_{i,j}$:

$$\mathbf{C}(m_{i,j}) = \frac{1}{K} \cdot \sum_{k=l}^{l+8} w_{k,j+1} \cdot (\mathtt{m}_{k,j+1}^t - \mathtt{p}) \cdot (\mathtt{m}_{k,j+1}^t - \mathtt{p})^T, \tag{5.2}$$

$$\mathtt{p} = \frac{1}{K} \sum_{k=l}^{l+8} \mathtt{n}_{k,j+1}^t, \tag{5.3}$$

$$\mathbf{C} \cdot \mathtt{v}_o = \lambda_o \cdot \mathtt{v}_o,\ o \in \{0,1,2\}, \tag{5.4}$$

$$l = 8 \cdot (i-1) + 1, \tag{5.5}$$

$$K = \sum_{k=l}^{l+8} w_{k,j+1}, \tag{5.6}$$

where weight $w_{k,j}$ is 1 if node $m_{k,j}$ has been already mapped. Accordingly, K is the number of mapped descendants of node $m_{i,j}$. λ_o is the o-th eigenvalue of the covariance matrix, and $\mathtt{v}_o$ the o-th eigenvector, which can be computed analytically. If exactly two of three eigenvalues are similar, the corresponding eigenvectors determine the plane through nodes $m_{k,j+1}$, and hence the surface normal for node $m_{i,j}$. If no good surface normal can be estimated, the surface normal vector centroid of the descendants is used instead, as is the case for node $m_{1,1}$ in Figure 5.2. If neither two similar eigenvalues nor a descendant node with a surface normal is available, as for node $m_{7,2}$, no surface normal is set. Figure 5.2 also shows the downside of this approach compared to using traditional nearest neighbor search. In our approach $m_{10,3}$ is not considered a neighbor to $m_{6,3}$, even though the corresponding voxels are adjacent in cartesian space. This relation is ignored because they are in different branches of the octree. Nevertheless, experiments (Section 5.3) show that in practice being able to process more points compensates for not being able to approximate surface normals for some points. Note that the data for such points is not discarded, and it is likely that approximating surface normals in such cases will become feasible at a future time due to the high map update rates achievable with NOctoSLAM.

Unlike RGBD cameras, multi-line LiDARs provide only sparse data along the vertical axis. This can lead to degenerate surface normal approximations, as illustrated in Figure 5.3. In the upper part we can see how a noisy reading (red dot) would lead to a degenerate surface normal estimate. The red striped node would have a surface normal perpendicular to the actual one. To tackle this, we use dynamic map resolutions by inserting pseudo leaf nodes at variable tree depths, as shown in the lower part of Figure 5.3. By decreasing the resolution we implicitly increase the number of reading points that are averaged to estimate a surface normal. Thus,

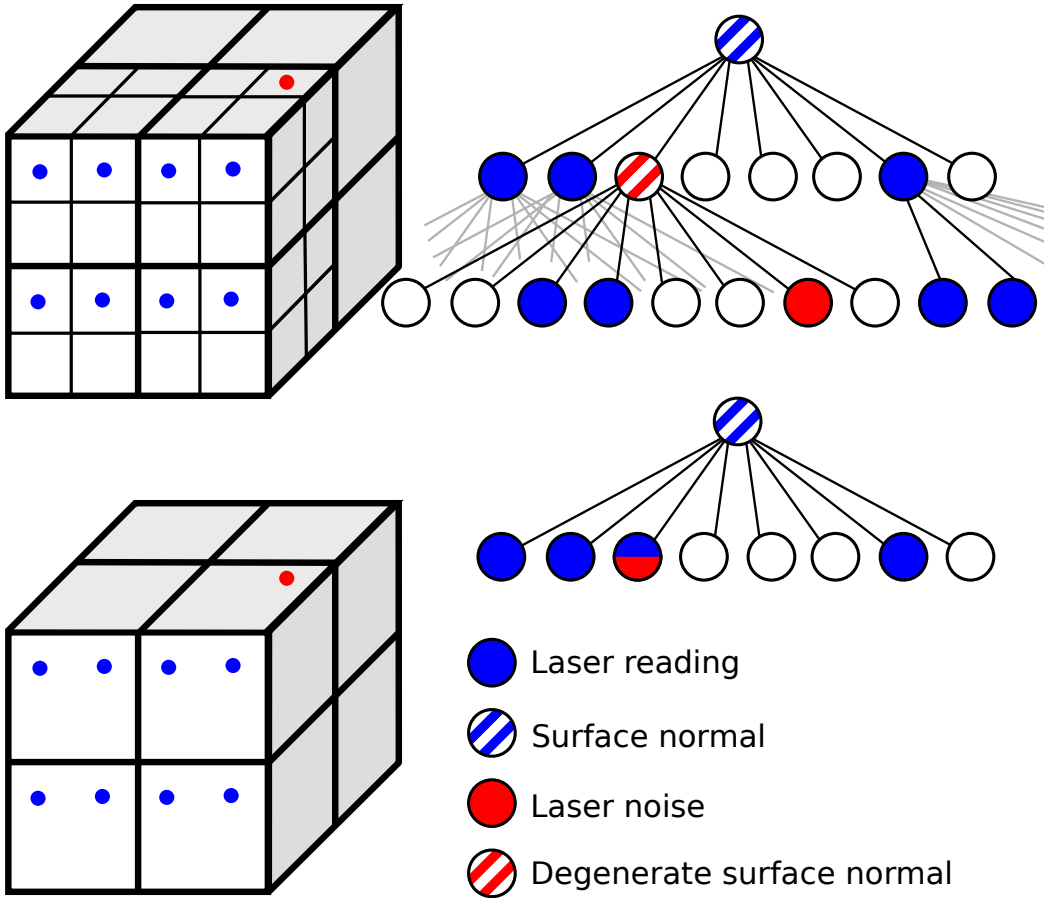


FIGURE 5.3: This figure gives an example for how noisy sensor data can lead to bad surface normal approximations in NOctoSLAM, as shown in the top half of the figure. Here, the noisy data point occupies its own node and the noise propagates to the parent node. The issue is tackled by using dynamic map resolutions. The coarser the map, the more points are being averaged reducing the impact of sensor noise, as shown on the bottom half of the figure, where multiple points are averaged in the leaf nodes.

the surface normal ends up being only slightly skewed, instead of being completely off. In particular, we choose a leaf node resolution that forces nodes on vertical scan lines to be adjacent to each other, depending on distance to sensor origin, vertical resolution of the sensor and map occupancy.

5.2.2 Pose Updates

To update the pose estimate we employ the point-to-plane iterative closest point algorithm as presented in Subsection 2.1.2. The basic ICP algorithm consists of two steps. First, correspondence between the source and reference data is computed (i.e. scan endpoints are associated with points in the map). Second, a transformation that minimizes the distance between corresponding points from input and reference set is computed.

To associate a 3D LiDAR measurement $\mathtt{d_i}$ with a position $\mathtt{m_{j,k}}^t$ and surface normal $\mathtt{m_{j,k}}^n$ stored in the map $M = \{m_{j,k}\}$, the octree is traversed as far as possible from root node $m_{1,1}$ towards the leaf node corresponding to the coordinates of $\mathtt{d_i}$.

The most recent node $m_{j,k}$ encountered during traversal, that has a parent node $\hat{m}_{j,k}$ with a set surface normal, is associated with $\mathtt{d_i}$, where $\hat{m}_{j,k} = m_{(j-1)/8,k-1}$. Instead of calculating the Euclidean distance for maximum correspondence distance rejection, the tree level k in which node $m_{j,k}$ is located, is used instead. A low k, where $1 < k \leq |M|$, implies a large correspondence distance, and thus inaccurate correspondence. The corresponding triplet $(\mathtt{d}_i, \mathtt{m}^t_{j,k}, \hat{\mathtt{m}}^n_{j,k})$ are used to determine the point-to-plane error. For ease of notation, we define the set of I correspondences as $\{(\mathtt{d}_i, \tilde{\mathtt{m}}^t_i, \tilde{\mathtt{m}}^n_i)\}$. Hence, the pose $\tilde{\mathbf{S}}$ that minimizes the error, when using the linear approximation presented in Subsection 2.1.2, is

$$\tilde{\mathbf{S}} = \arg\min_{\tilde{\mathbf{S}}} \sum_{i}^{I} \left(\left(\tilde{\mathbf{S}} \cdot \begin{bmatrix} \mathtt{d}_i^T & 1 \end{bmatrix}^T - \begin{bmatrix} (\tilde{\mathtt{m}}^t_i)^T & 1 \end{bmatrix}^T \right) \cdot \begin{bmatrix} (\tilde{\mathtt{m}}^n_i)^T & 0 \end{bmatrix}^T \right)^2, \tag{5.7}$$

where $\tilde{\mathbf{S}}$ is a pose matrix assuming $\alpha, \beta, \gamma \approx 0$:

$$\tilde{\mathbf{S}} = \begin{bmatrix} 1 & -\gamma & \beta & t_x \\ \gamma & 1 & -\alpha & t_y \\ -\beta & \alpha & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}. \tag{5.8}$$

To solve Equation 5.7 we refer to Equation 2.25 till 2.32.

5.3 Empirical Evaluation

In this section we perform experiments to evaluate the pose estimation quality as well as the runtime performance of NOctoSLAM.

The experiments are conducted on an Intel® Core™ i7-4770 CPU with 16GB of RAM, and the sensor used is a VLP-16[1] multi-line LiDAR. It can provide about 15000 measurements at 20 Hz, with a maximum range of 100m. The measurements are distributed among 16 horizontal lines over a vertical FOV of 30 deg and over a horizontal FOV of 360 deg. It is rated with an accuracy of $\pm$0.03m. We do not perform simulated experiments because we were not able to find a sufficiently efficient 3D simulator for multi-line 3D LiDAR sensors. Gazebo for example only achieves a fraction of real time speed on our machine when simulating the VLP-16.

To estimate the accuracy of pose estimates we compare against ground truth poses provided by an Optitrack[2] motion capture system.

We furthermore compare the NOctoSLAM results against ETH Zurich's ICP Mapping tool[3], which is a ROS wrapper for libpointmatcher [22] for registration and libnabo [52] for nearest neighbor search. The point-to-plane ICP algorithm, excluding data association, is essentially the same in NOctoSLAM and the ETHZ ICP Mapping tool.

The NOctoSLAM (referred to as *NOcto* in the figures) performance is evaluated against the ETH Mapping tool with two different sets of parameters. For NOctoSLAM, a minimum

[1] `http://velodyneLiDAR.com`
[2] `http://optitrack.com`
[3] `http://wiki.ros.org/ethzasl_icp_mapping`

TABLE 5.1: Pose metrics for 15 indoor episodes

	Position in m			Rotation in radians		
	Median	Mean	σ	Median	Mean	σ
NOcto	**0.030**	**0.029**	**0.005**	0.018	0.019	**0.006**
ETH	0.032	0.032	0.011	**0.015**	**0.017**	0.008
*ETH**	0.094	0.113	0.054	0.066	0.068	0.023

voxel size of 0.01m is used, and the map is updated after every registration. In the following, algorithm *ETH* refers to using no input filters and a $0.1 \times 0.1 \times 0.1\ m^3$ voxel grid map filter, i.e. the same map resolution as NOctoSLAM. Thus, registration is performed with approximately 15000 input points at 20 Hz. In order to increase runtime performance, *ETH** uses various input filters that reduce the number of input points to approximately 2500. Among others, a max density filter configured to 300 points per m^3 is used. Additionally, the map is also limited to a point density of 50 points per m^3. The algorithm denoted with *ETH** updates the map not after every registration, but only if 3D LiDAR measurements and map overlap less than 95%.

5.3.1 Pose Estimation Accuracy

We evaluate the accuracy of pose estimates for 15 recorded episodes, where each episode is between one and two minutes long. In each episode, the sensor is moved manually through a $8 \times 8 \times 3.5m^3$ space. The sensor is equipped with motion capture markers, which are externally tracked. A rigid transformation resulting from aligning the ground truth and estimated trajectories via ICP is used to calibrate the sensor's optical axis to the markers. For each episode we calculate the mean error (ME) of the position and rotation in respect to the ground truth data.

Figure 5.4 shows the results for 15 episodes, where the center line marks the mean, the inner box represents the 95% confidence interval, and the outer box illustrates the standard deviation. The figure shows that there is no significant difference of either positional or rotational accuracy between *NOcto* and *ETH*. Strongly reducing the number of source and reference points leads to a significantly worse performance of *ETH**. In fact, with an average positional ME of 0.113m (see Table 5.1), *ETH** approximately quadruples that of *NOcto*. Additionally, the standard deviation of *ETH** is also one order of magnitude higher as the one of *NOcto*. A similar trend can be observed for the rotation MEs.

To visualize the impact of a 0.094m positional error versus a 0.029m error, we plot the trajectories for the median performance episode of *ETH** in Figure 5.5. In order to increase clarity only every fourth trajectory point is plotted, and we omit the *ETH* trajectory as it is very similar to the *NOcto* trajectory. It can be seen that the *ETH** trajectory aligns significantly worse with the motion capture trajectory than the *NOcto* trajectory does.

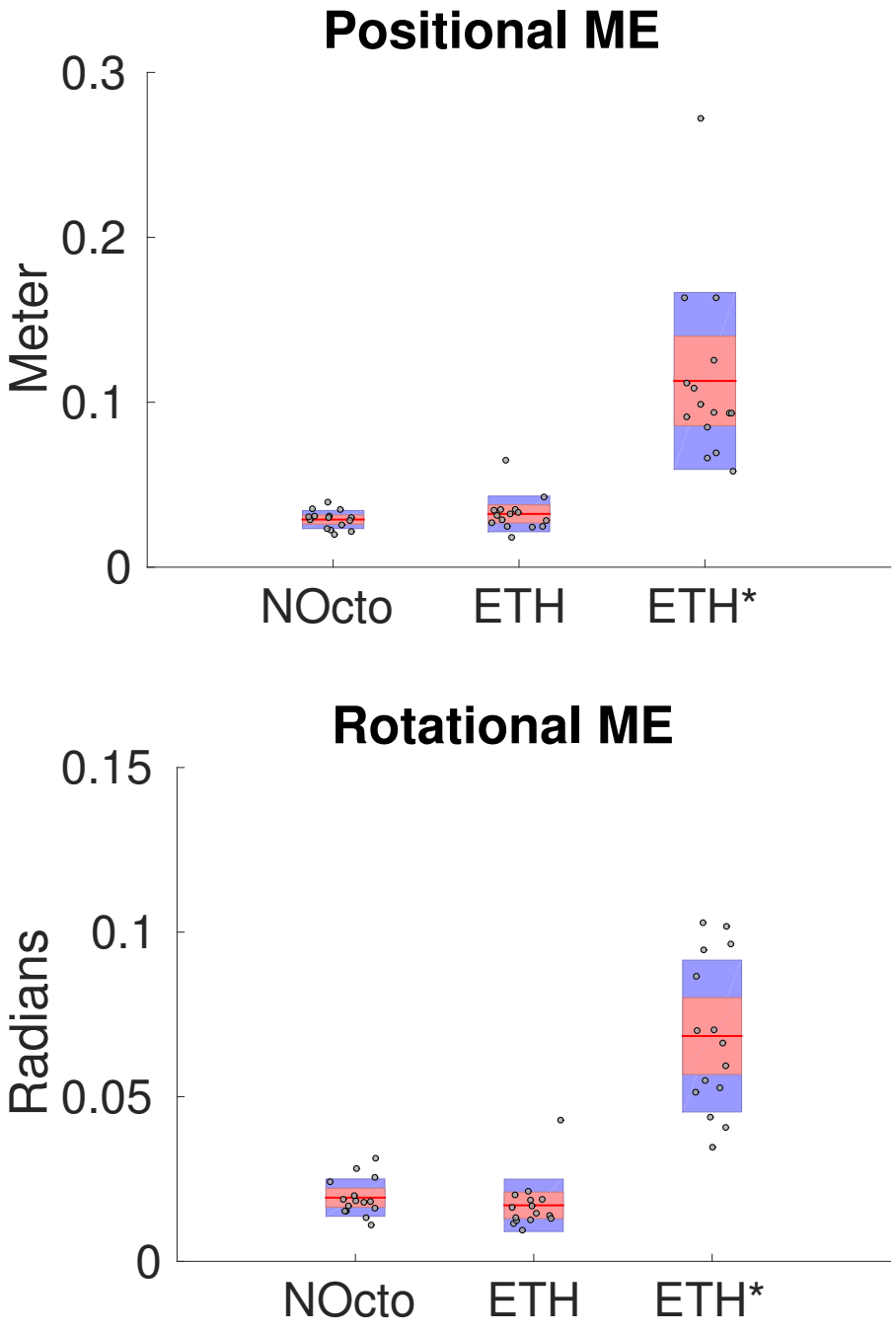


FIGURE 5.4: This figure shows the positional MEs on the top, and the rotational MEs on the bottom. *ETH** performs significantly worse than the other two algorithms, which perform similarly well.

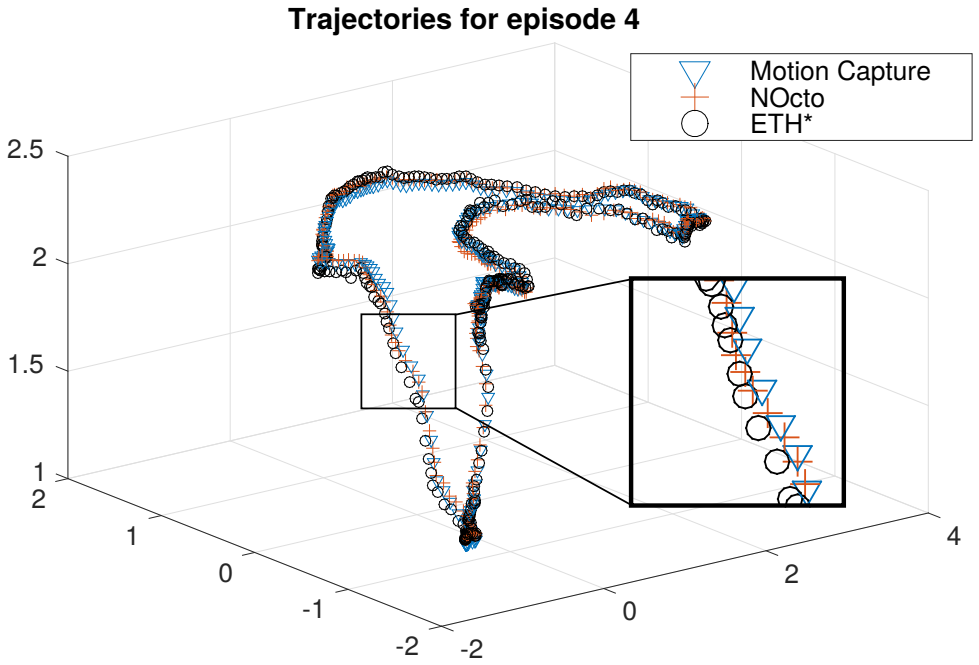


FIGURE 5.5: This figure shows the pose estimates for the median *ETH** episode. The performance difference between *NOcto* and *ETH** is indicated by the superior alignment of *NOcto* to the motion capture trajectory.

5.3.2 Runtime Performance

When performing localization and mapping, the main factors in terms of time consumption are updating the surface normals and associating source to reference data. In the following analysis the same data as for the experiments in the previous subsection are used.

We investigate the map update duration in Figure 5.6. The figure shows that *ETH* and *ETH** map update durations correlate with the number of points in the map. This stems from the unorganized point cloud representation used that requires reprocessing at least large parts of the map during each update. *ETH** shows that the update duration can be reduced by only storing sparse point clouds and using smaller update sizes. Additionally, updating the map in parallel to scan registration, and at a low frequency, makes using a point cloud format computationally feasible. On the other hand, the tree based map representation used in *NOcto* allows for constant time map updates, regardless of map size. Thus, it is not necessary to explicitly filter map data with our approach. The mean map update duration for *NOcto* is 0.0062s with a standard deviation of $6 * 10^{-4}$s, i.e. ~1/8th of the available time at a 20 Hz update rate.

Figure 5.7 shows a box plot of the scan registration duration per iteration for the three algorithms. The figure was generated from ~$4 * 10^5$ iterations. The median of *NOcto* is 0.0018s, halving that of *ETH* (0.0036s), while processing approximately the same number of points. With a median of 0.0009 s, *ETH** outperforms the other two, albeit processing only ~2500 points instead of ~15000 points per iteration. However, reducing the number of input points also causes additional computational cost, where the severity depends on the filters used. In our experiments, input filtering for *ETH** took on average 0.0167s per registration. Note that input filtering is performed only once for all registration iterations performed per sensor return.

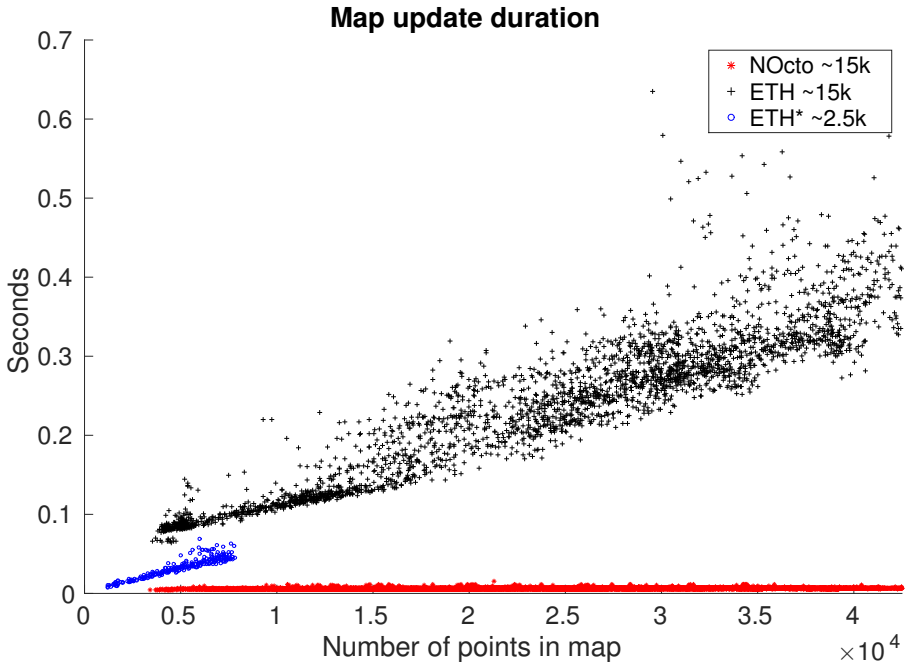


FIGURE 5.6: This figure shows the map update durations in relation to number of points stored in the map. The points describing the lower quasi constant belong to *NOcto*, and the points describing the shorter linear line below 0.08s duration belong to *ETH**. A linear dependency between map size and required time for both *ETH* and *ETH** is visible, while *NOcto* performs map updates in constant time.

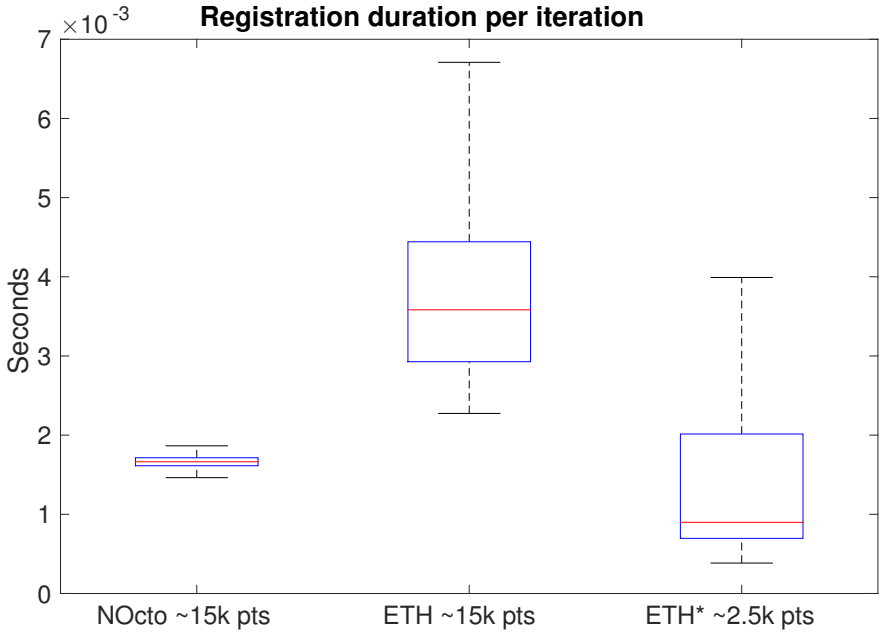


FIGURE 5.7: This figure shows the registration durations per iteration for the three algorithms. *NOcto* significantly outperforms *ETH* when both process 15000 points per iteration. With the number of points reduced to 2500, *ETH** is faster than *NOcto*, but suffers from a significantly larger standard deviation.

5.3.3 Visual Inspection

In this subsection we present point cloud based maps generated by transforming all sensor data with the SLAM pose estimates, where the color represents measurement intensity. We use this representation for visual inspection because it provides more information to determine if features are captured correctly.

Figure 5.9 shows the map and trajectory generated by a short flight with a 3D LiDAR equipped multirotor UAV[4]. We again compare the results of using *NOcto*, *ETH** and *ETH*. While the differences in map quality between the approaches are very hard to make out, small details such as the less sharply defined tree stumps indicate that *ETH** performs less well in terms of accuracy. The lack of points on the *ETH(*)* trajectories shows that even with filtering enabled, both variants struggle to keep up with the sensor rate.

Figure 5.8 shows a map of the DFKI building in Bremen. This episode demonstrates that NOctoSLAM works indoors as well as outdoors. The map was created by carrying the LiDAR sensor from the 3rd floor of the building in the bottom left to the ground floor, through the driveway towards the building in the background[5]. Fine details such as e.g. the books in the bookshelf or the tires of the car indicate a good SLAM accuracy.

The capability of SLAM over multiple levels is shown in Figure 5.10. Here, the four story staircase of the University of Liverpool Ashton building is mapped. A good indicator for SLAM consistency over multiple levels is that the floors are mapped in parallel to each other. The same map, but shown from multiple angles can be found in Appendix B.1 (Figure B.2, B.4, B.7),

[4] `https://www.youtube.com/watch?v=eA03k7rMQY8`
[5] `https://www.youtube.com/watch?v=zdIcQXOEpaw`

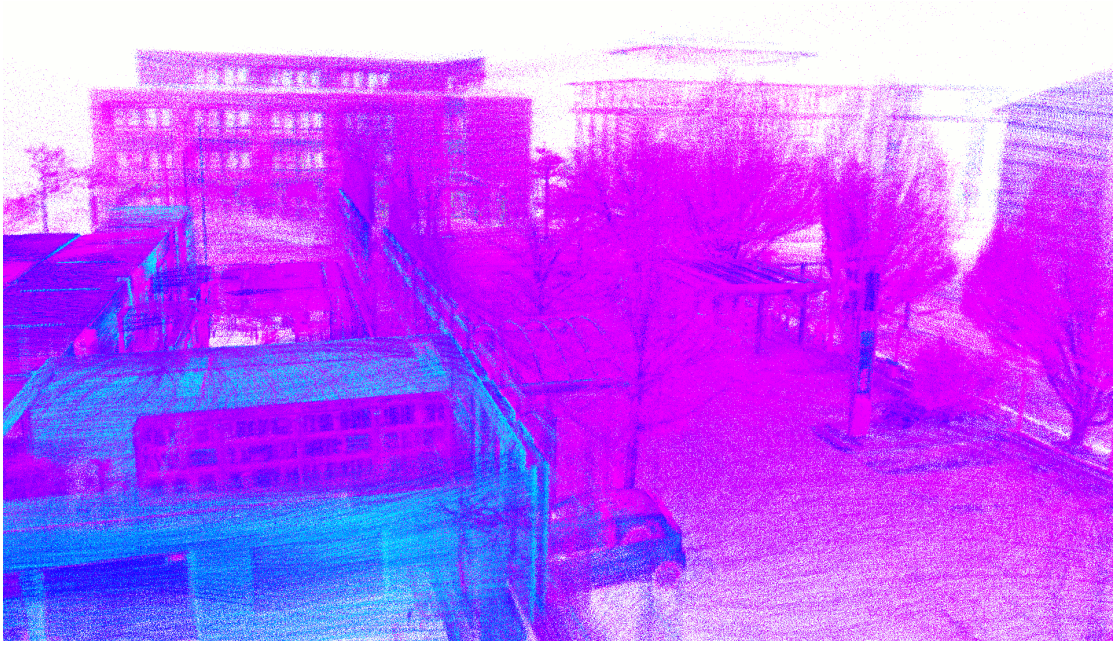


FIGURE 5.8: This figure shows intensity sensor data transformed with NOctoSLAM pose estimates from a combined indoor and outdoor episode. The sensor data was acquired by carrying the sensor through the DFKI Bremen building to the outside.

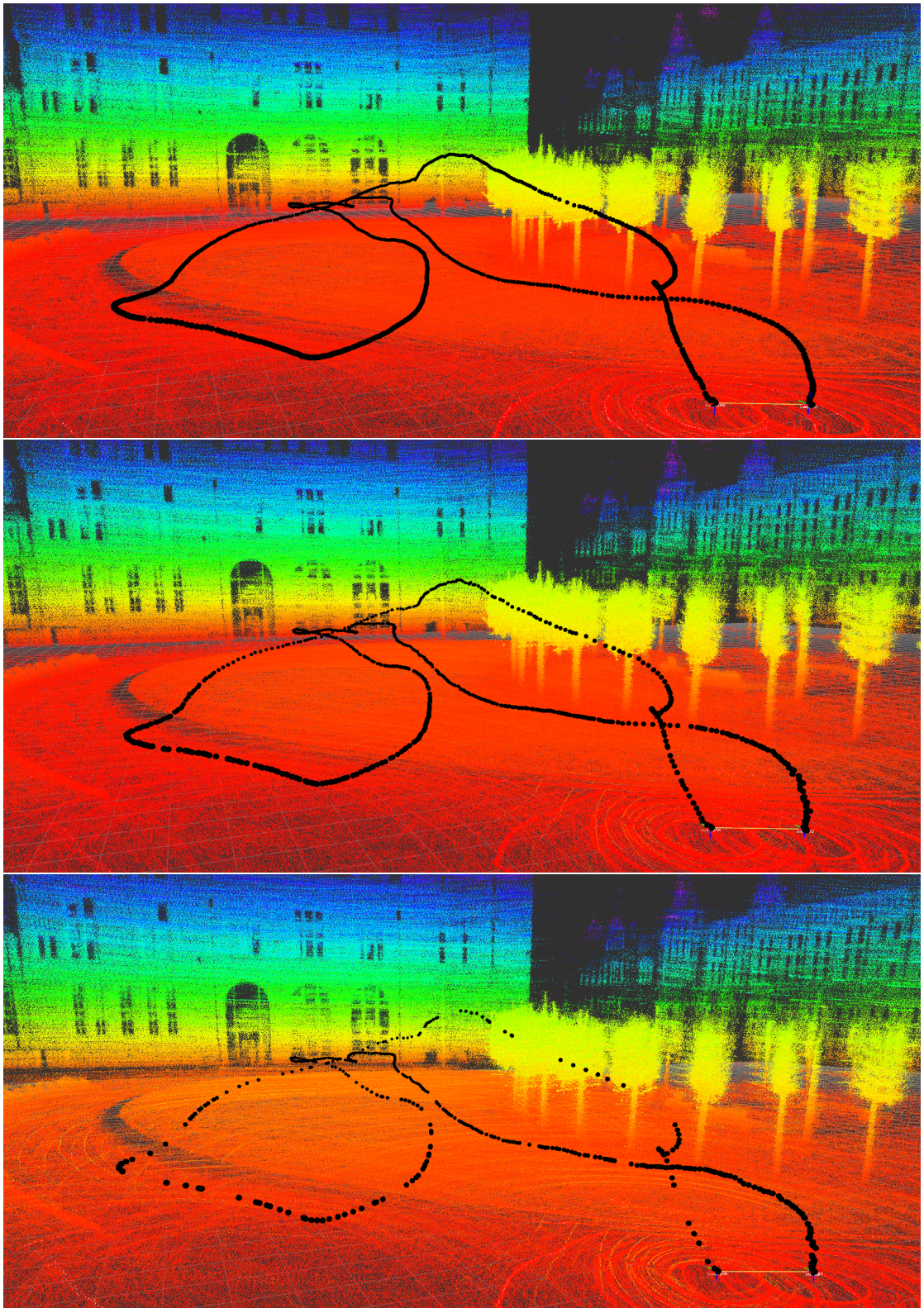


FIGURE 5.9: This figure shows trajectories and intensity sensor data transformed with *NOcto* (top), *ETH** (middle), and *ETH*. The sensor data was generated by a multirotor UAV mounted 3D LiDAR sensor.

FIGURE 5.10: This figure shows intensity sensor data transformed with NOctoSLAM pose estimates. The sensor data was generated by a carrying the 3D LiDAR sensor through the four story staircase of the UoL Ashton building.

where we also provide the internally used 0.1m resolution map (Figure B.8), and plot the map surface normals (Figure B.3, B.5, B.6).

Appendix B.1 furthermore shows the NOctoSLAM results for mapping an office in Figure B.1, and for using sensor data generated by mounting a 3D LiDAR on a car [99] in Figure B.9, B.9[6].

5.3.4 Summary

In this section we have shown that NOctoSLAM performs as well as the ETH ICP Mapping Tool in terms of pose estimation accuracy. Since the VLP-16 has a typical accuracy of ± 0.03 m, both algorithms perform well in this regard, achieving a ME of approximately 0.03m. However, map updates as well as scan registration are significantly faster in NOctoSLAM, enabling near-real

[6] `https://www.youtube.com/watch?v=oKJ3mB-KuZk`

time usage. Especially relevant is the constant map update time of NOctoSLAM, independent of map size. In contrast, we have shown that in point cloud based approaches, map updates quickly become infeasible in near real-time for large maps and high update rates. While the use of source and reference point filters can increase the performance, it is a cumbersome task to find the correct filter parameters that achieve a good trade off between accuracy and computational performance. Also, some parameters depend on the environment, thus requiring prior expert knowledge that may be expensive to get or may not be available. Furthermore, filters might become infeasible if the environment changes, e.g. when changing from indoor to outdoor. For instance, indoor maps typically require more points per m^3 for good pose estimates than outdoor maps. While we do not have quantitative data on the pose estimation accuracy in larger environments, the 3D maps we have presented indicate a good performance.

5.4 Conclusions

In this chapter we have presented NOctoSLAM, a novel approach to surface normal based mapping in conjunction with point-to-plane ICP scan registration. We empirically demonstrated that the proposed algorithm is as accurate, but significantly faster than a traditional point cloud based approach. In terms of precision we achieve a mean error well within the rated accuracy ($\pm$0.03 m) of the sensor used: 0.029m positional, and 0.019 rad rotational, with a low standard deviation of 0.005m and 0.006 rad, respectively. Unlike the case for point cloud based approaches, NOctoSLAM can maintain high map update rates, regardless of the map size. Because of the superior computational performance configuring data filters is not necessary.

A downside of octree based map representations is that the maximum volume the map can cover is limited by the minimum voxel size and maximum tree depth. Using a minimum voxel size of 0.1m and a tree depth of 16, allows for maps with a maximum volume of approximately $3000 \times 3000 \times 3000$ m^3. As just increasing the tree depth leads to higher computation time, a better approach would be using nested [86] or overlapping octrees, if larger volumes are required.

Chapter 6

Recap and Conclusions

In this thesis we have presented three novel approaches for simultaneous localization and mapping (SLAM), suitable for different applications, i.e. ground based 2D LiDAR SLAM and airborne 2D & 3D LiDAR SLAM.

The first application we investigated is SLAM on a multirotor UAV equipped with a 2D LiDAR sensor. The problem with traditional SLAM approaches for this scenario is that they use 2D maps. 2D maps are not well suited because they are unable to capture possibly critical environmental information, e.g. traversable obstacles of low height. Furthermore, reducing 3D environments to 2D can negatively affect localization accuracy, up to complete failure. Therefore we asked the following research questions:

- How is 3D map SLAM achievable on a multirotor UAV equipped with a rigidly mounted 2D LiDAR sensor?
- To what extent does 2D LiDAR sensor based 3D map SLAM outperform its 2D map counterpart in terms of accuracy?

We answer those questions with OctoSLAM, which combines measurements from an altitude, an attitude, and a 2D LiDAR sensor to achieve 3D map SLAM. OctoSLAM employs an octree based 3D map representation that uses the tree structure to efficiently store the map in different resolutions. We use these multiple resolutions to interpolate map gradients, which are then used for scan registration via Gauss-Newton minimization.
We empirically evaluated 2D vs. 3D map SLAM in a series of simulated experiments. We found that using a 3D map yields significant advantages if the environment in view of the LiDAR sensor loses too many features when projected to 2D. In environments where the latter is likely to happen, using a 3D map resulted in approximately half the localization error compared to using 2D map SLAM. The results obtained from the simulated experiments were reinforced by using a 2D LiDAR sensor mounted on a multirotor UAV for SLAM in a warehouse environment. While not quantitatively evaluated, visual inspection of the generated maps showed that 3D map SLAM again outperformed 2D map SLAM if the environment is inhomogeneous along height, as the maps produced by 2D map SLAM were distinctively less accurate. For environments that do not lose many features when projected onto 2D we found that neither approach significantly outperforms the other.

The second scenario we tackled is ground based SLAM with 2D LiDAR sensors. The problem here is that traditional 2D map SLAM systems typically use occupancy grid cell maps, which can not encode environmental details smaller than grid cell size, which impair registration accuracy. Furthermore, using efficient Gauss-Newton based registration requires map gradients, which are cumbersome to interpolate from occupancy grid cell maps as it requires maintaining the same map in different resolutions. Accordingly, we posed the following research questions:

- How can we adopt SDF based 2D mapping for 2D LiDAR based SLAM, that uses Gauss-Newton minimization via map gradients for registration?
- To what extent can we improve 2D LiDAR sensor based 2D map SLAM accuracy by using a SDF instead of an occupancy grid cell based map?

To answer these questions, we developed 2D-SDF-SLAM. 2D-SDF-SLAM approximates the SDF of the environment using a grid cell based map, where each cell stores the lowest distance to the next obstacle. We found that wide angle 2D LiDAR sensors require a different SDF update scheme than visual sensors. Therefore, we proposed an orthogonal Deming regression based update method, that allows consistent mapping of 2D LiDAR sensor measurements. 2D-SDF-SLAM uses the distances provided by the SDF based map to interpolate relatively accurate gradients and map values, which in turn are used for Gauss-Newton minimization based registration.
We compared 2D-SDF-SLAM with Hector SLAM, a system that uses a similar registration technique, but in combination with multiple occupancy grid cell maps. We found that in simulation 2D-SDF-SLAM on average outperformed Hector SLAM by a staggering ~343% in terms of mean localization error. Using a SDF map grid resolution of a 0.05m, 2D-SDF-SLAM achieved a low average error of 0.0070m with a standard deviation of 0.0067m, i.e. well below grid cell size. Outside of simulation though, the performance advantage over Hector SLAM decreased to ~10% in terms of mean error and ~25% in terms of standard deviation, as the mean error increased to 0.0307m, which is in the magnitude of the sensors rated accuracy of $\pm$0.03m.

The third application we investigated is airborne SLAM using 3D LiDAR sensors. Typically point cloud based point-to-plane ICP is used for 6 DoF registration of scans provided by 3D LiDAR sensors. The problem however is that this approach does not scale well with map size, as point clouds possess no geometric information that can be used to restrict surface normal recalculation to relevant parts of the map. Thus, the whole map has to be reprocessed every map update, which quickly becomes infeasible under the near real-time constraint required for online SLAM front-ends. To cope, point cloud based approaches employ data filters, which reduce the number of points that have to be processed. However, configuring such filters can be cumbersome, and most likely requires knowledge about the environment, as different filter parameters are required for e.g. indoor vs. outdoor environments. Furthermore, even with filters, the runtime performance is still correlated with the map size, i.e. filtering only shifts the problem and does not solve it. Accordingly, we posed the following research question.

- How can we develop a 3D map representation suitable to point-to-plane scan registration (i.e. supporting SFN approximation and NN search), increasing the computational performance whilst maintaining accuracy?

With NOctoSLAM, we proposed an octree based 3D map representation that uses the tree structure for surface normal approximation and correspondence search. In contrast to octomap mapping, we do not discretize the environment, but do store positional information with sub voxel accuracy.
We empirically evaluated NOctoSLAM and found that it is as accurate as point cloud based SLAM for which the near real-time constrained is lifted. However, if it is enforced, NOctoSLAM significantly outperformed point cloud based SLAM suffering from data filtering. Additionally, we showed that in our approach the map update duration does not correlate with the number of points in the map, enabling mapping of arbitrary environments online and in near real-time.

To conclude the thesis: we have successfully shown the advantages of interweaving advanced map representations with scan registration, which resulted in three novel algorithms for various scenarios, as discussed in the previous paragraphs. Each algorithm did perform better than a comparable, established SLAM approach.

While we have focused on SLAM using relatively expensive LiDAR sensors, we anticipate that the increased demand by the automotive industry will make such sensors more affordable and thus more commonly used. In the same breath, technological boundaries will be expanded, e.g. with the introduction of solid state LiDAR sensors in the near future.

For future work, we think that research into supplementing the proposed approaches with back-ends, i.e. pose graph optimization, is warranted. However, while possibly achievable in a straight forward manner for 2D-SDF-SLAM, where few map updates produce a consistent map of the local environment, a back-end for LiDAR based 3D map SLAM is more challenging. Here, many more sensor sweeps are required to map the environment, where each only provides sparse data over the z axis. As we aim for online pose graph optimization during runtime, the sheer amount of data prevents using each LiDAR sweep as a pose graph node. One option is to perform map updates less frequently, resulting in a manageable amount of nodes in the pose graph, as performed in the *Berkeley Localization And Mapping*[1] (BLAM) approach. The downside of that approach is that locally within each node, localization accuracy suffers because of the lack of points in the map, thus trading in local accuracy for global consistency. Also, performing only low frequency map updates would be in opposition to the strength of NOctoSLAM, i.e. efficient map handling. Another option is using a frequently updated map for the front-end, and a less frequently updated one for the back-end. We assume that this would be beneficial to global consistency without decreasing local accuracy, but it would also mean that after loop closure the local map is reset with the global one. A more elegant solution would be to use NOctoSLAM to gain locally consistent high fidelity maps, and somehow detect when to spawn a new node in the pose graph, e.g. triggered by registration error. Another good addition

[1] `https://www.youtube.com/watch?v=08GTGfNneCI`

to NOctoSLAM would be the ability to handle quasi unlimited map sizes, without sacrificing resolution or computational efficiency, e.g. via overlapping octrees.

6.1 Critical Notes

We have shown that the proposed algorithms outperform similar approaches that use naive or unorganized map representations. Still, there are limitations one should be aware of.

2D map SLAM for airborne robots assumes that the environment looks the same over the complete z axis. We loosened that restriction with OctoSLAM to the extent that we only assume limited similarity over height, which allows OctoSLAM to perform better than 2D map SLAM in challenging environments. In particular, we assume that there is similarity over height of unmapped parts of the environment that are measured by consecutive scans. If this is not the case, e.g. if the robot performs fast attitude changes resulting in a large angular offset between consecutive scans, performance decreases drastically.

With 2D-SDF-SLAM we showed that accuracy can be increased significantly by using a memory efficient map representation that does not discretize the environment, and yet provides geometrical information. Nevertheless, we found that in practice, the accuracy was instead limited by current sensor technology.

NOctoSLAM leverages an octree to significantly increase computational efficiency, resulting in similar performance as point cloud based SLAM if no time constraints are applied, and better performance, otherwise. If there is no time constraint, e.g. because the data is processed offline, NOctoSLAM yields no advantage. However, it does indirectly apply a voxel grid filter to the map, so it does limit the map accuracy to a small degree. Furthermore, as the surface normals are calculated using only information from sibling or descendant nodes, classical kd-tree based nearest neighbor search theoretically provides more accurate surface normals. Though, in practice we found the NOctoSLAM surface normals to be sufficiently accurate.

6.2 The Future of LiDAR based SLAM

We conclude this thesis with a personal statement about the future of LiDAR based SLAM in comparison to visual SLAM.

Over the past years visual SLAM has become a strong contender to LiDAR based SLAM, especially for airborne robots with restricted payload capabilities. If using a 3D LiDAR is not feasible, e.g. due to monetary or payload constraints, we feel that using visual SLAM produces superior results to 2D LiDAR based 6 DoF SLAM due to the dimensionality of the sensors. Currently, using visual SLAM on airborne robots is probably more popular than LiDAR based SLAM in the research community.

However, 3D LiDAR based SLAM outperforms vision based SLAM in certain regards, such as providing accurate maps over large distances, being more stable to changing lighting conditions, being mostly unrestricted by object textures, as well as providing multiple returns. The

latter means that multiple reflections can be picked up by the sensor, e.g. when the light travels through glass before being reflected by a solid object.

Unfortunately, 3D LiDAR sensors are currently at least a magnitude more expensive than RGB(D) cameras. Nevertheless, we hope that increased demand from the automotive industry will even out the difference in price - mass production and new technologies that simplify 3D LiDAR sensors might lead to cheaper and smaller 3D LiDAR sensors. This trend may have already started, indicated by Velodyne[2] reducing the price of their cheapest 3D LiDAR sensor, the VLP-16 used in Chapter 5, from 8000 USD to 4000 USD in the beginning of this year. An example for new technologies are solid state LiDAR sensors, which do not rely on moving parts, but use silicon to "steer" the emitted light instead, thus drastically reducing complexity. Quanergy[3] claims that they will be able to sell solid state 3D LiDAR sensors for as low as 100 USD in the near future. We predict that at this point 3D LiDAR based 6 DoF SLAM will become more popular again, as the entry barrier in terms of price becomes negligible, while the results obtained with e.g. NOctoSLAM clearly show the validity of 3D LiDAR based SLAM.

[2] http://velodynelidar.com/
[3] http://quanergy.com/

Appendix A

OctoSLAM Results contd.

A.1 Simulated Localization on a given Map

A.1.1 Tables

TABLE A.1: OctoSLAM results for localization on a 2D and 3D map for the "Willow" environment. 2D map refers to using the 3D map projected onto the ground plane for registration.

Map type	LIDAR noise	Mean	Deviation
3D	noise	0.0524	0.0134
	no noise	0.0504	0.0147
2D	noise	0.0587	0.0119
	no noise	0.0600	0.0116

TABLE A.2: OctoSLAM results for localization on a 2D and 3D map for the "corridor" environment. 2D map refers to using the 3D map projected onto the ground plane for registration.

Map type	LIDAR noise	Mean	Standard deviation
3D	noise	0.0516	0.0274
	no noise	0.0467	0.0279
2D	noise	0.1058	0.0421
	no noise	0.1052	0.0373

TABLE A.3: OctoSLAM results for localization on a 2D and 3D map for the "sphere" environment. 2D map refers to using the 3D map projected onto the ground plane for registration.

Map type	LIDAR noise	Mean	Standard deviation
3D	noise	0.0655	0.0770
	no noise	0.0444	0.0180
2D	noise	0.0867	0.1678
	no noise	0.0861	0.1696

TABLE A.4: OctoSLAM results for localization on a 2D and 3D map for the "tilted wall" environment. 2D map refers to using the 3D map projected onto the ground plane for registration.

Map type	LIDAR noise	Mean	Deviation
3D	noise	0.0431	0.0186
	no noise	0.0389	0.0172
2D	noise	0.0874	0.0386
	no noise	0.0850	0.0402

A.1.2 Bar Graphs no Sensor Noise

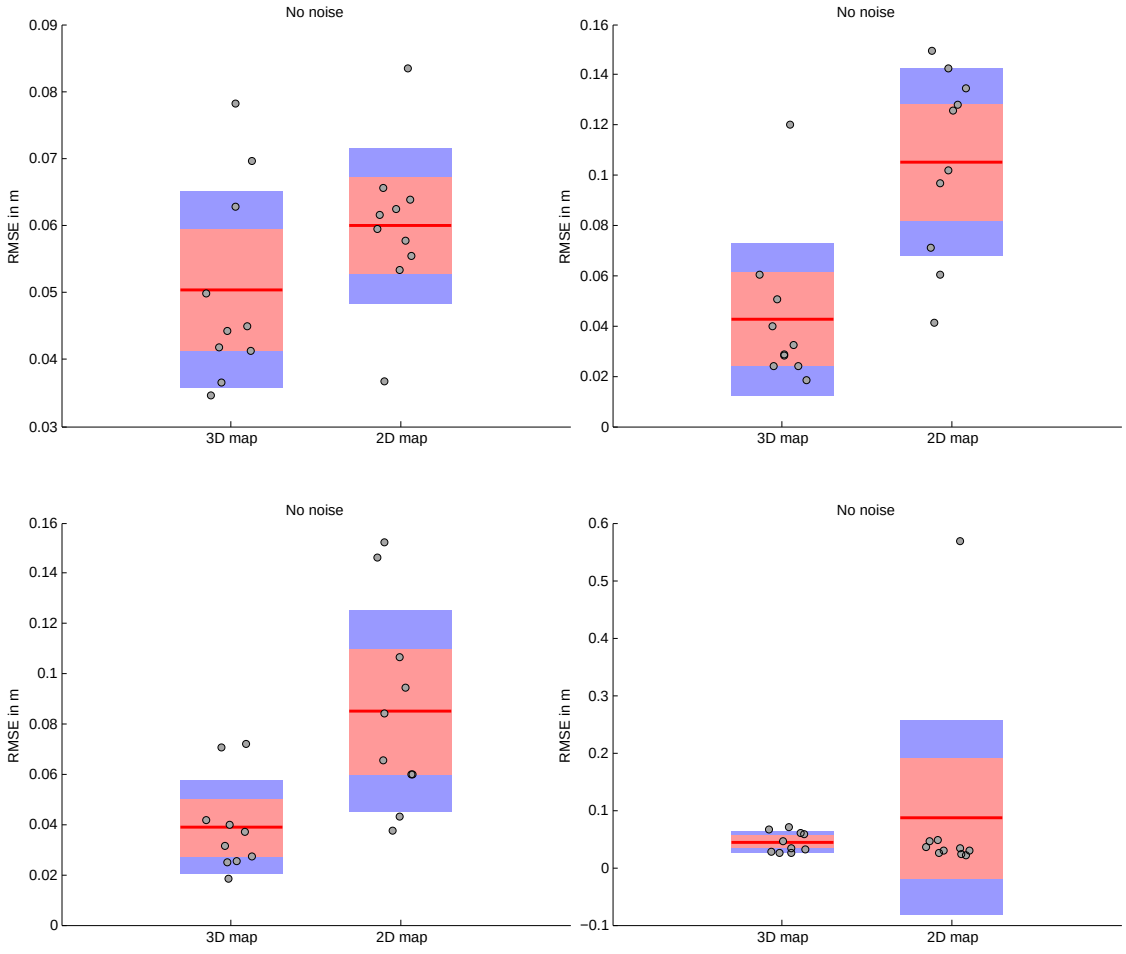


FIGURE A.1: This figure shows OctoSLAM results for localization on a given map in the simulated environments without sensor noise. From top left in clockwise direction: "Willow", "corridor", "sphere" and "tilted wall" environment. 2D map refers to using the 3D map projected onto the ground plane for registration. Note that the scale for the RMSE is not the same for all graphs.

A.2 Simulated SLAM

A.2.1 Tables

TABLE A.5: Results for 2D and 3D map SLAM for the "Willow" environment.

SLAM system	LIDAR noise	Mean	Standard deviation
Octo 3D	noise	0.0904	0.0514
	no noise	0.0896	0.0506
Octo 2D A	noise	0.0566	0.0261
	no noise	0.0550	0.0246
Octo 2D B	noise	0.0454	0.0295
	no noise	0.0434	0.0265
Hector	noise	0.0874	0.1107
	no noise	0.0855	0.1106

TABLE A.6: Results for 2D and 3D map SLAM for the "corridor" environment.

SLAM system	LIDAR noise	Mean	Standard deviation
Octo 3D	noise	0.1062	0.0375
	no noise	0.1034	0.0410
Octo 2D A	noise	0.2257	0.1181
	no noise	0.2249	0.1208
Octo 2D B	noise	0.2948	0.1371
	no noise	0.3324	0.2412
Hector	noise	0.7293	0.9114
	no noise	0.6411	0.9240

TABLE A.7: Results for 2D and 3D map SLAM for the "sphere" environment.

SLAM system	LIDAR noise	Mean	Standard deviation
Octo 3D	noise	0.3452	0.3382
	no noise	0.2540	0.2532
Octo 2D A	noise	0.3961	0.5260
	no noise	0.3308	0.4976
Octo 2D B	noise	0.4192	0.3148
	no noise	0.3133	0.2157
Hector	noise	0.7619	1.3701
	no noise	0.9012	0.5302

TABLE A.8: Results for 2D and 3D map SLAM for the "tilted wall" environment.

SLAM system	LIDAR noise	Mean	Standard deviation
Octo 3D	noise	0.1330	0.0674
	no noise	0.1381	0.0621
Octo 2D A	noise	0.2595	0.1505
	no noise	0.2956	0.1486
Octo 2D B	noise	0.3006	0.1504
	no noise	0.3342	0.1484
Hector	noise	0.8442	0.7881
	no noise	0.8888	0.7608

A.2.2 Bar Graphs no Sensor Noise

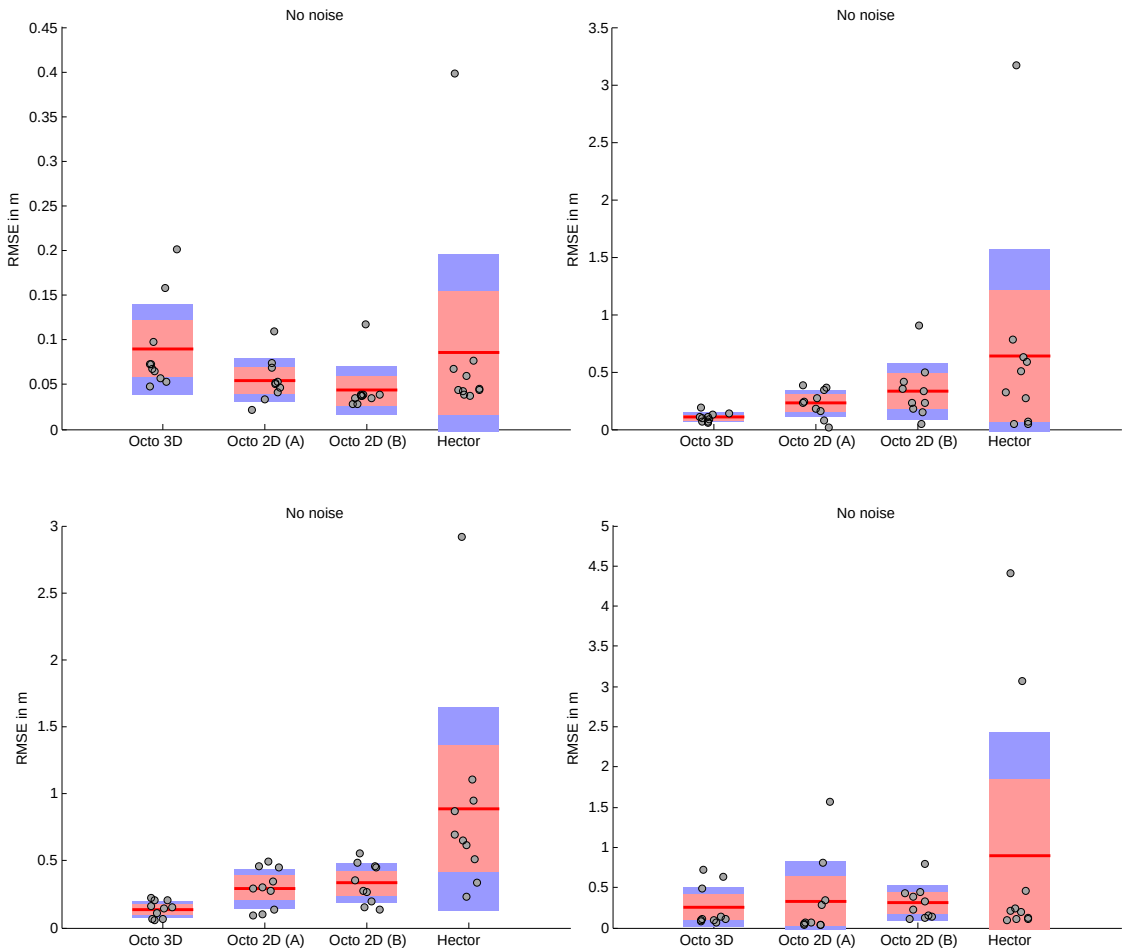


FIGURE A.2: This figure shows the results for SLAM in the simulated environments without sensor noise. From top left in clockwise direction: "Willow", "corridor", "sphere" and "tilted wall" environment. Note that the scale for RMSE differs between the graphs.

A.3 Real Robot SLAM

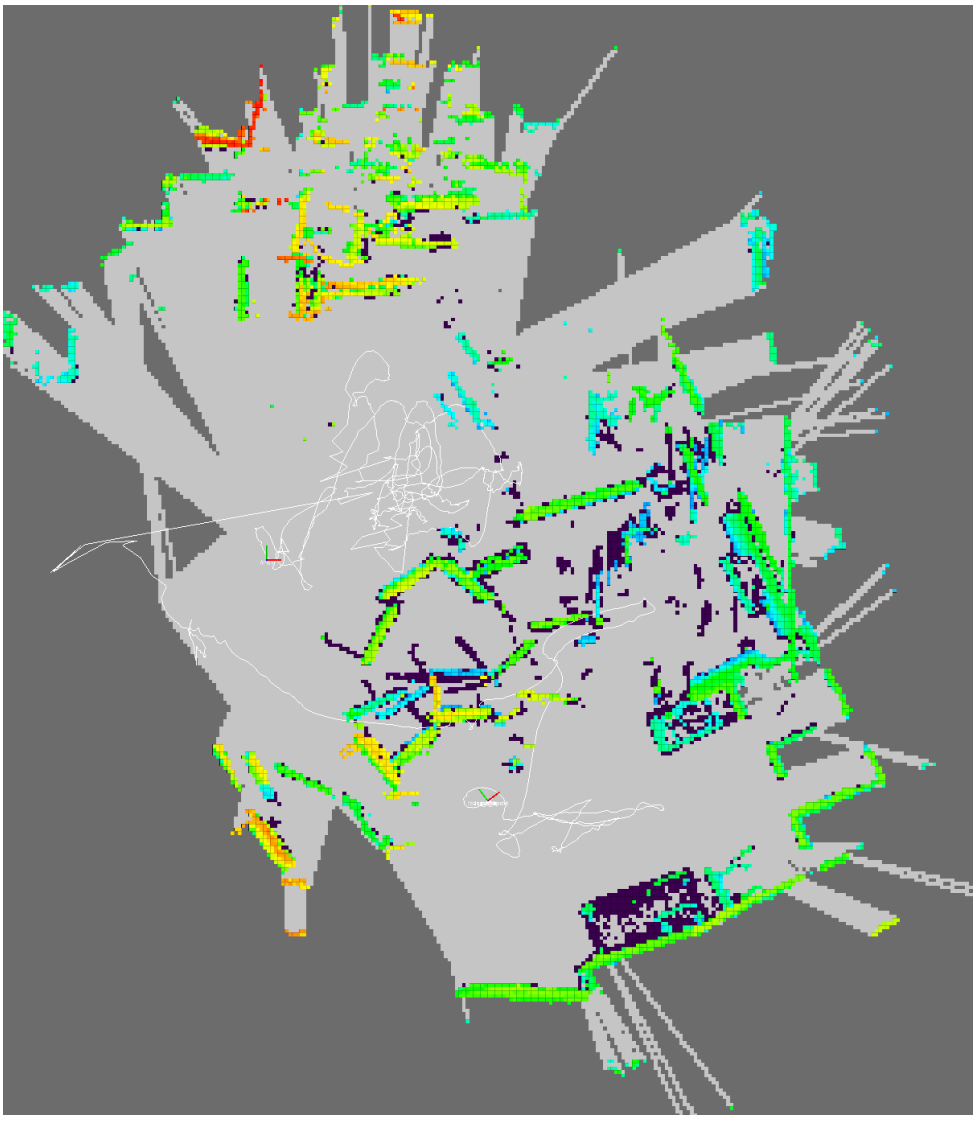


FIGURE A.3: This figure shows the results for OctoSLAM in 2D mode for the challenging environment. The trajectory (white line) shows multiple "jumps", indicated by long straight lines. Subsequently, the quality of the map is low. According to map and trajectory, walls are traversed multiple time, which is not the case.

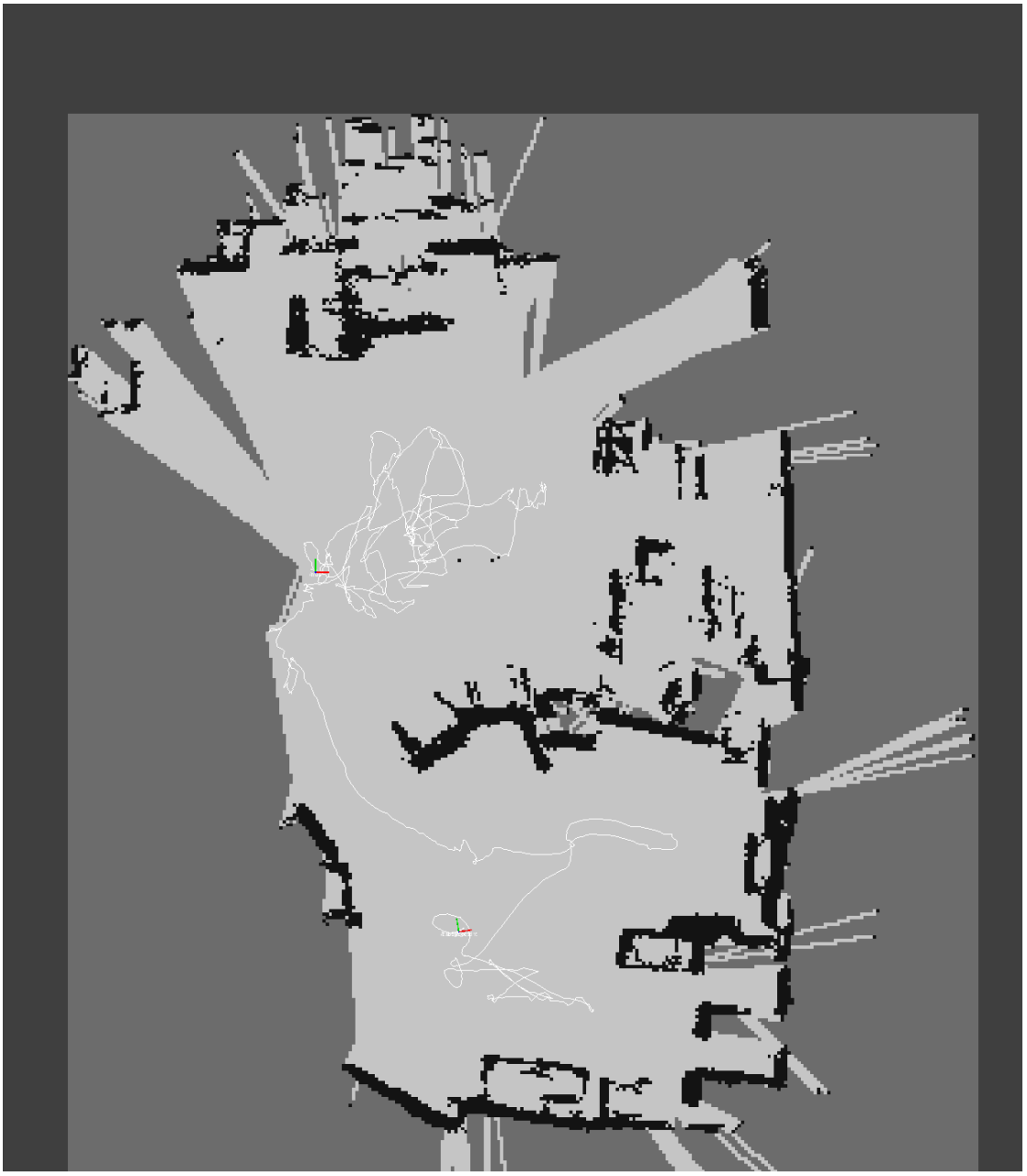


FIGURE A.4: This figure shows the results for 3D map OctoSLAM for the challenging environment. The trajectory appears to be smooth, and the environment is accurately represented by the map, indicating a good SLAM performance.

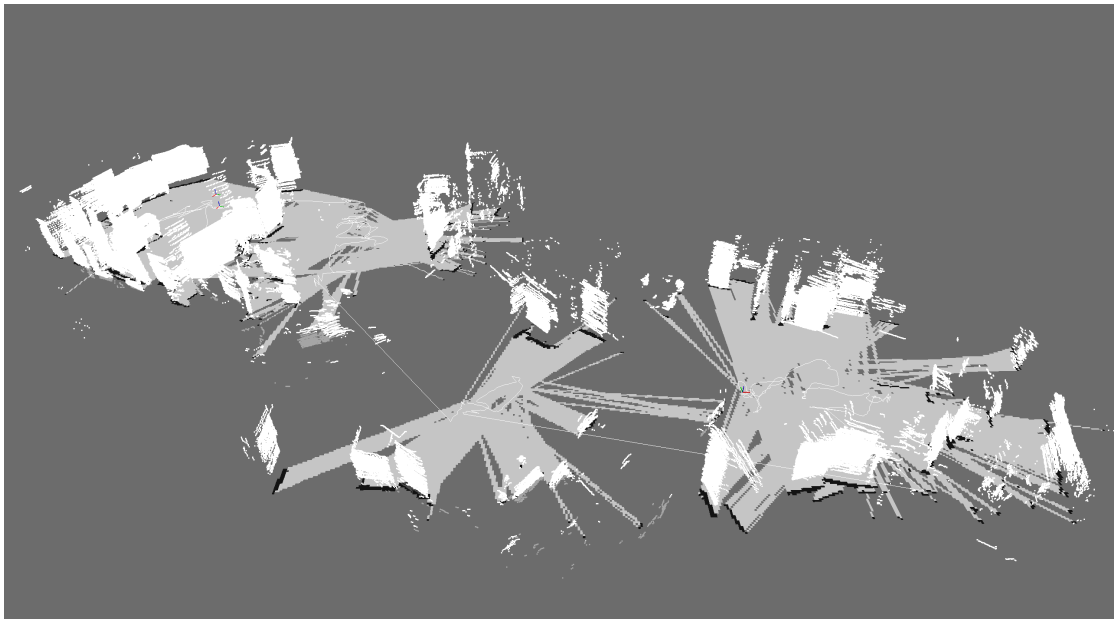


FIGURE A.5: This figure shows the map resulting from using Hector SLAM in the challenging environment. Similar to OctoSLAM in 2D mode (Figure A.3), Hector SLAM performs poorly in this environment.

Appendix B

NOctoSLAM Results contd.

B.1 Visual Inspection

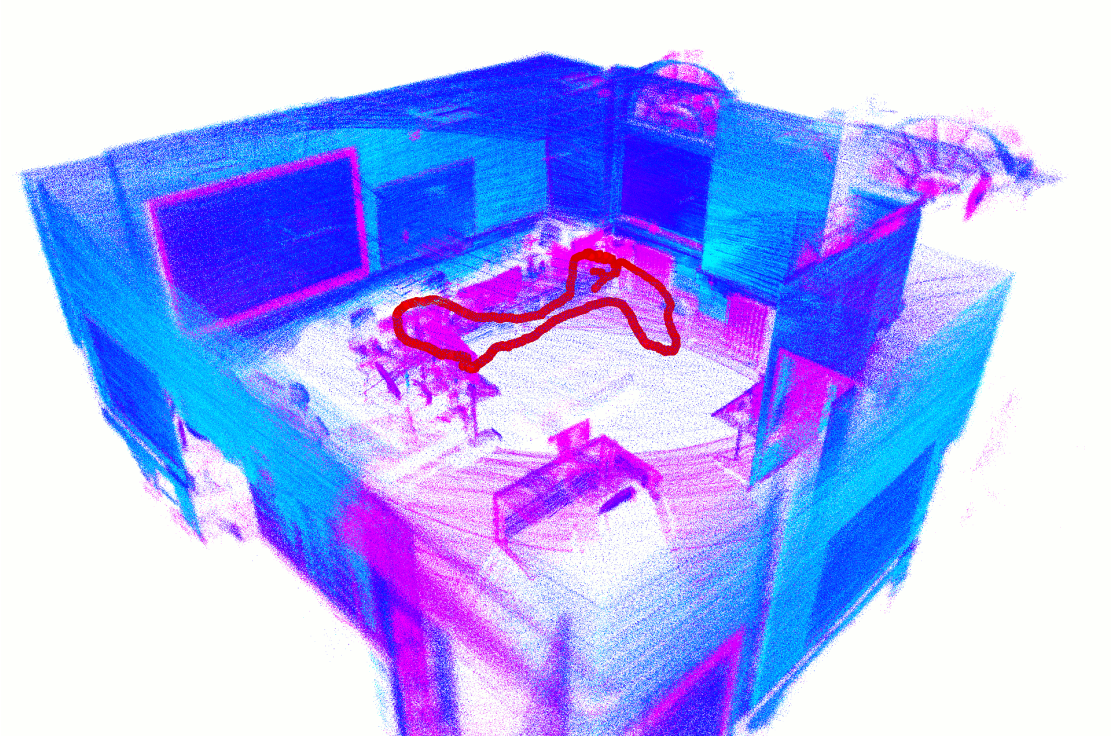


FIGURE B.1: This figure shows intensity sensor data transformed with NOctoSLAM pose estimates, mapping an office.

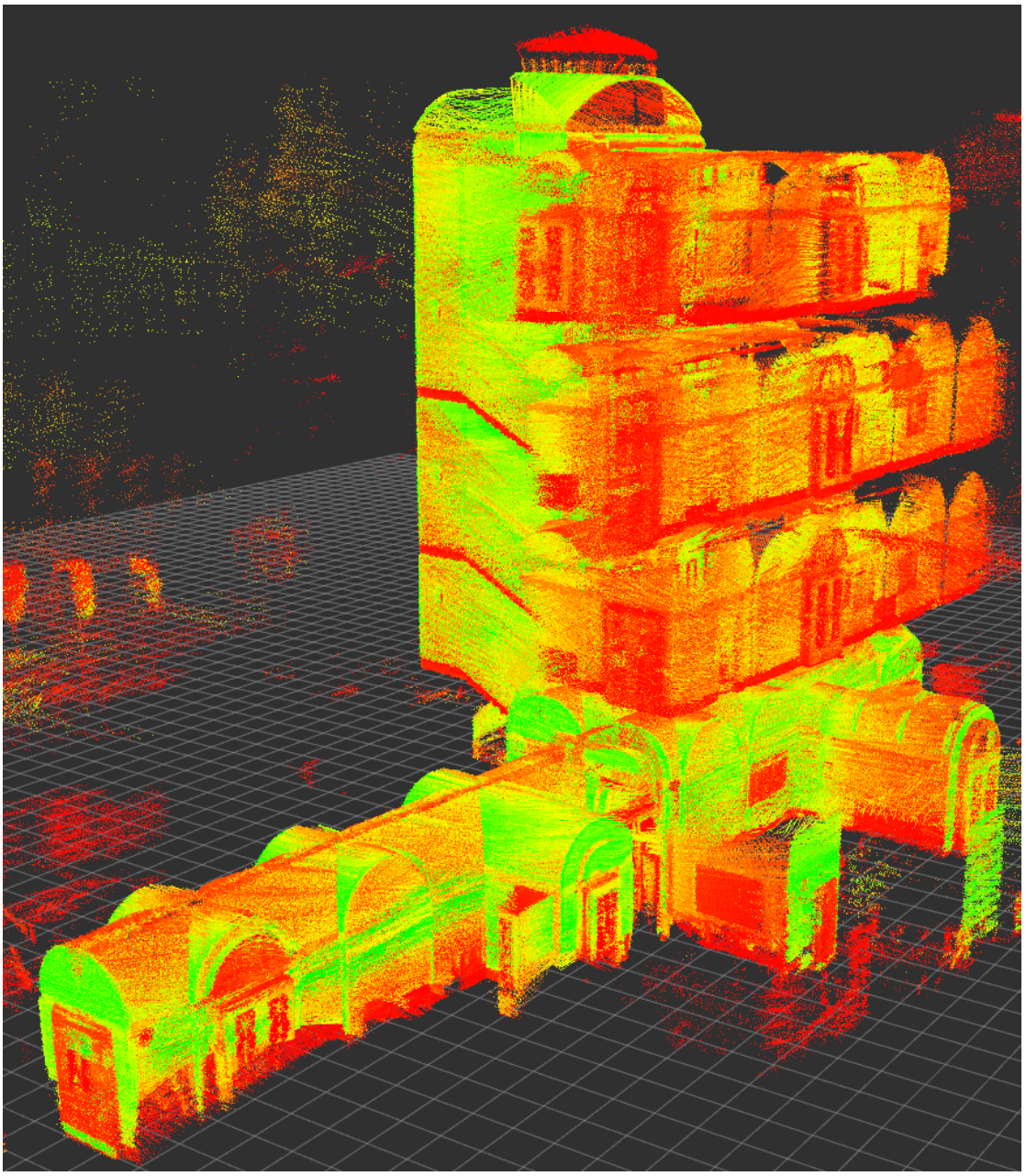


FIGURE B.2: This figure shows intensity sensor data transformed with NOctoSLAM pose estimates, mapping the four story staircase of the UoL Ashton building.

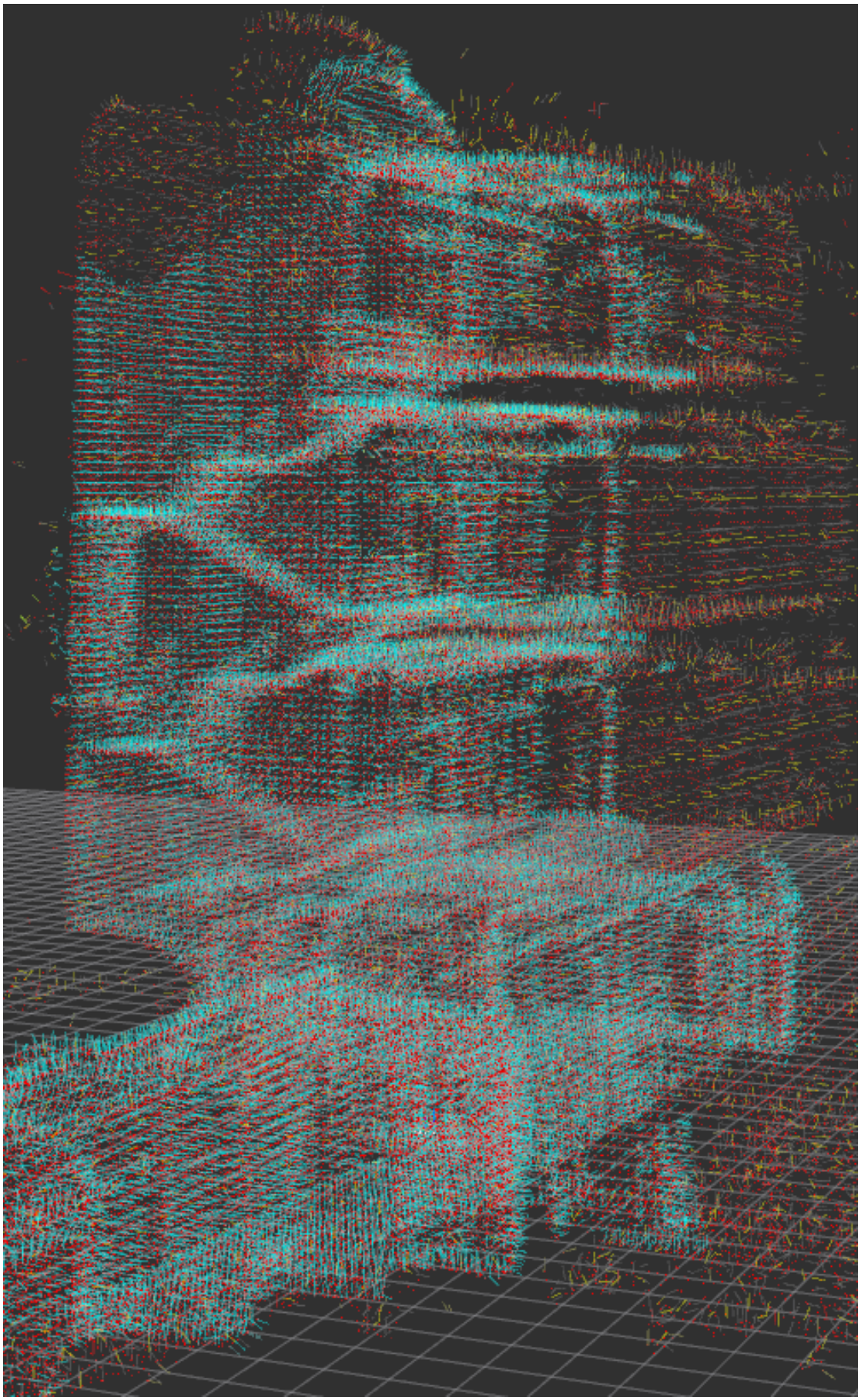


FIGURE B.3: This figure shows the internal NOctoSLAM map for the Ashton building, including surface normals. Color encodes the tree depth of the nodes storing the surface normals.

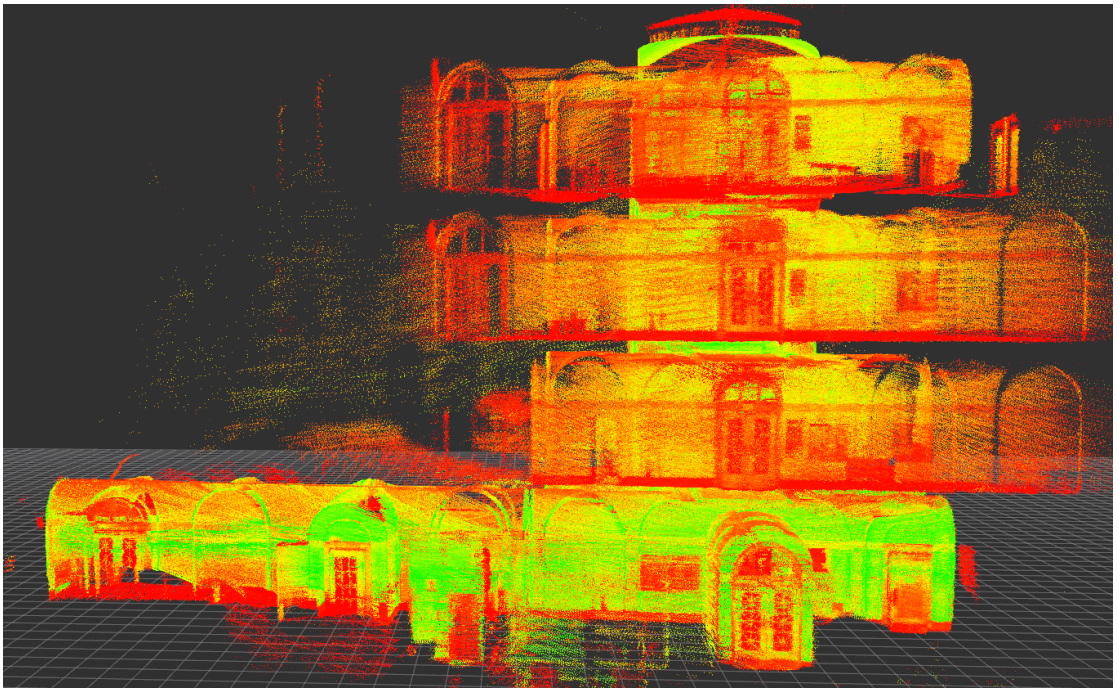


FIGURE B.4: This figure shows intensity sensor data transformed with NOctoSLAM pose estimates, mapping the four story staircase of the UoL Ashton building.

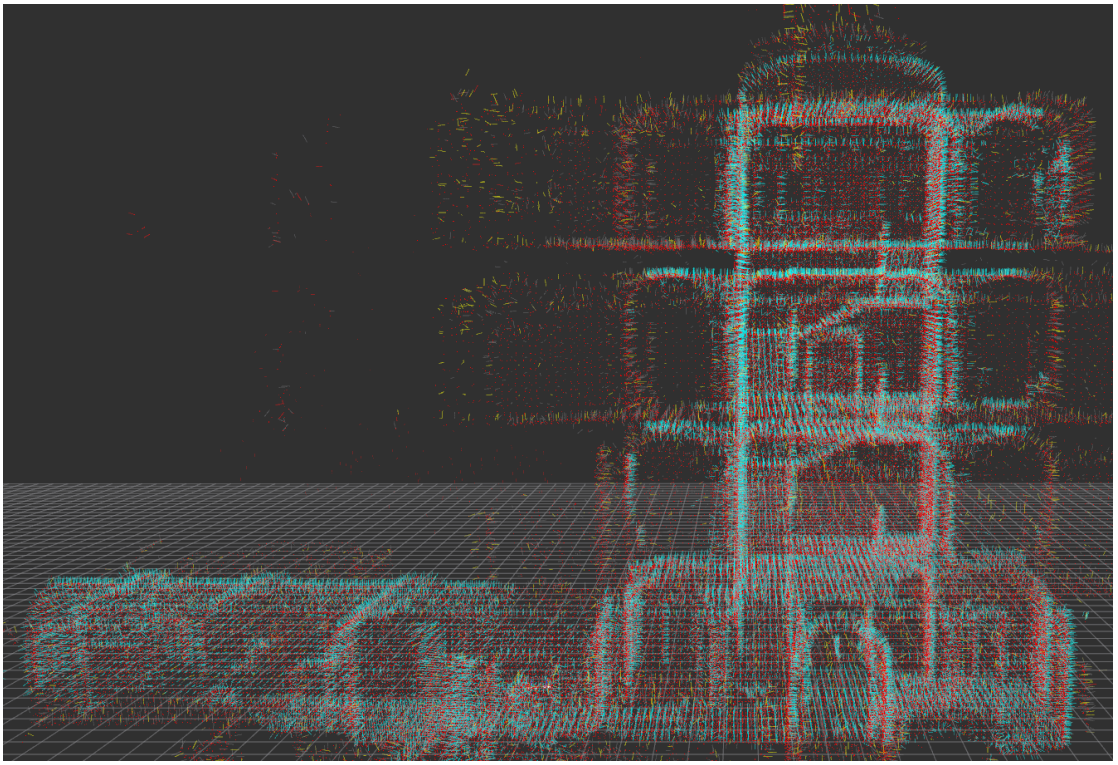


FIGURE B.5: This figure shows the internal NOctoSLAM map for the Ashton building, including surface normals. Color encodes the tree depth of the nodes storing the surface normals.

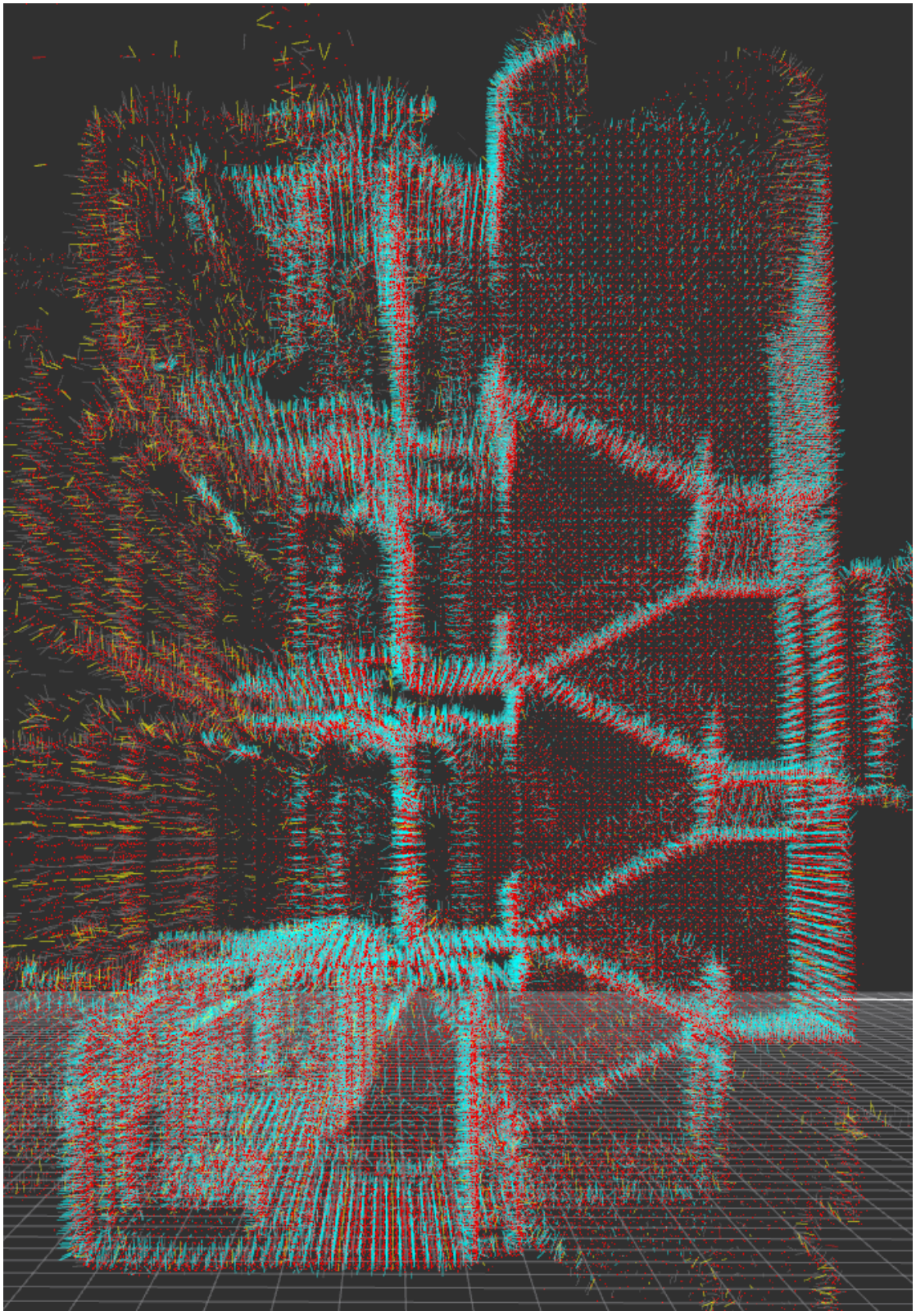


FIGURE B.6: This figure shows the internal NOctoSLAM map for the Ashton building, including surface normals. Color encodes the tree depth of the nodes storing the surface normals.

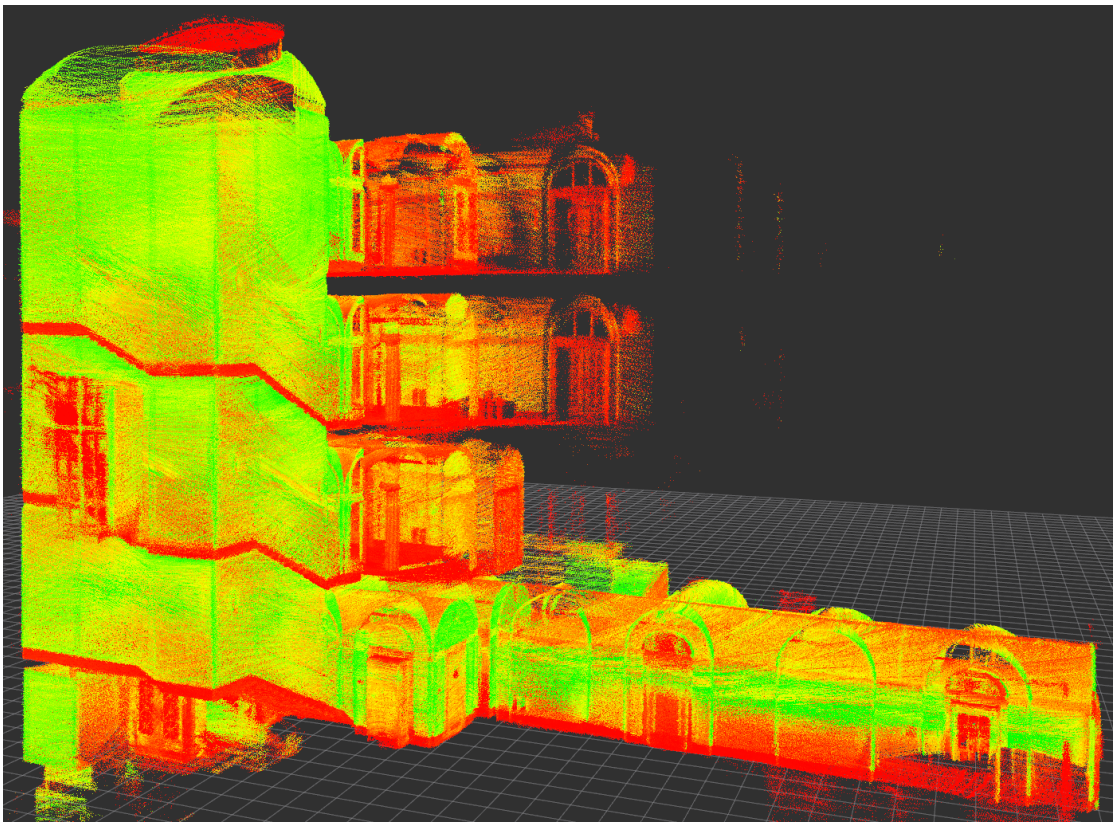


FIGURE B.7: This figure shows intensity sensor data transformed with NOctoSLAM pose estimates, mapping the four story staircase of the UoL Ashton building.

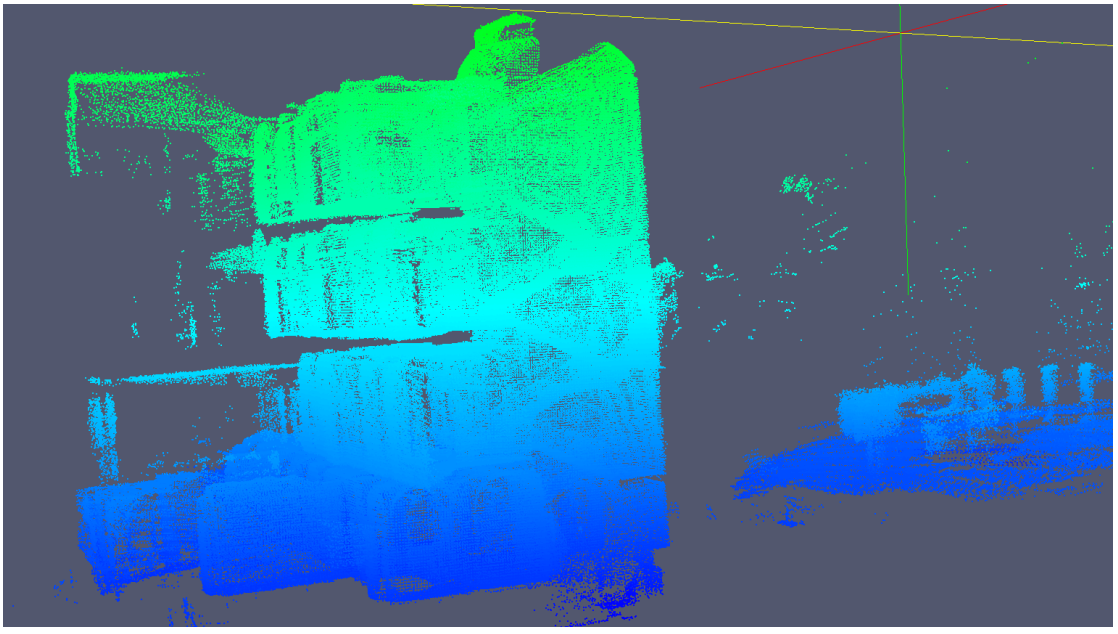


FIGURE B.8: This figure shows the internal NOctoSLAM map of the Ashton building. As we do not store intensity data in the internal map, altitude is used to color the map.

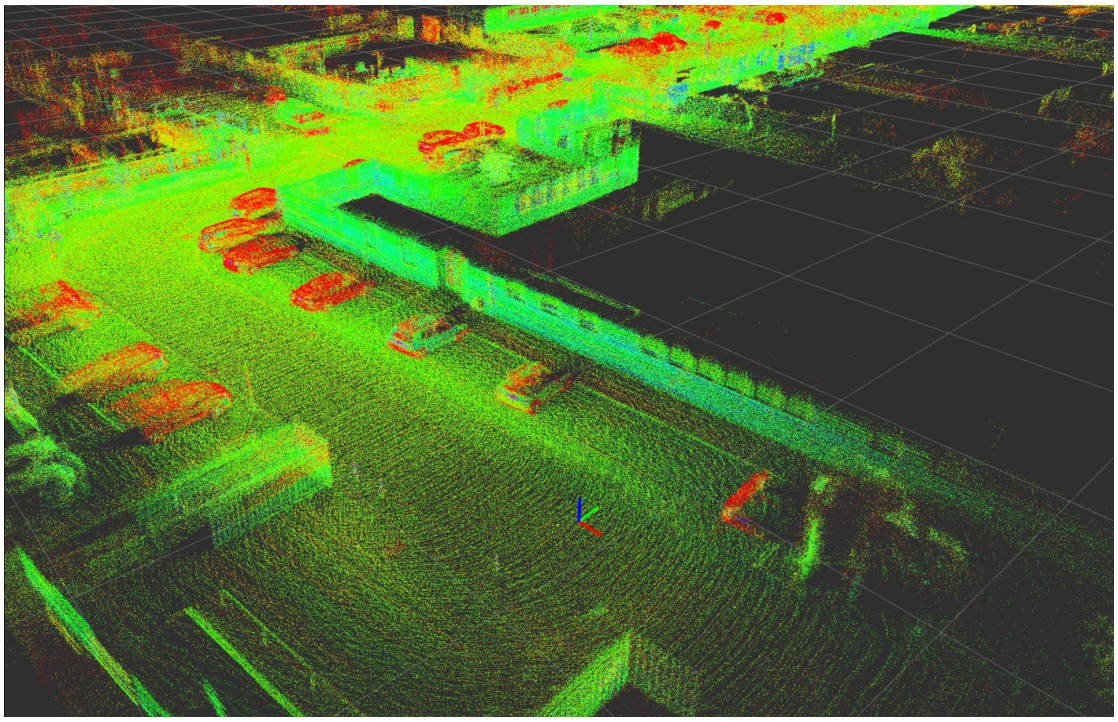


FIGURE B.9: This figure shows intensity sensor data transformed with NOctoSLAM pose estimates, mapping an urban environment with a car mounted 3D LiDAR sensor [99].

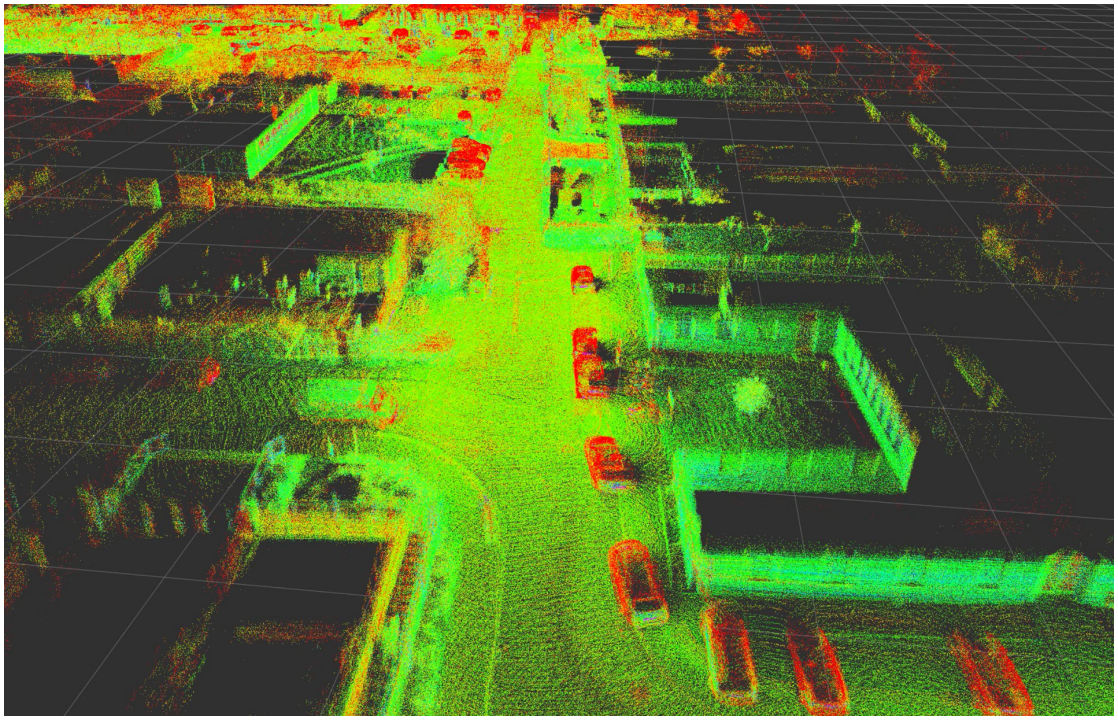


FIGURE B.10: This figure shows intensity sensor data transformed with NOctoSLAM pose estimates, mapping an urban environment with a car mounted 3D LiDAR sensor [99].

Bibliography

[1] S. Thrun and J. J. Leonard, "Simultaneous Localization and Mapping," in *Springer Handbook of Robotics*. Springer, 2008, pp. 871–889.

[2] S. Kohlbrecher, J. Meyer, O. von Stryk, and U. Klingauf, "A Flexible and Scalable SLAM System with Full 3D Motion Estimation," in *Proceedings of the International Symposium on Safety, Security, and Rescue Robotics*. IEEE, 2011, pp. 155–160.

[3] D. Holz and S. Behnke, "Sancta Simplicitas - On the efficiency and achievable results of SLAM using ICP-Based Incremental Registration," in *Proceedings of the International Conference on Robotics and Automation*. IEEE, 2010, pp. 1380 –1387.

[4] P. J. Besl and N. D. McKay, "A Method for Registration of 3-D Shapes," *Transactions on Pattern Analysis and Machine Intelligence*, vol. 14, no. 2, pp. 239–256, 1992.

[5] A. Diosi and L. Kleeman, "Laser Scan Matching in Polar Coordinates with Application to SLAM," in *Proceedings of the International Conference on Intelligent Robots and Systems*. IEEE, 2005, pp. 3317 – 3322.

[6] A. Censi, "An ICP variant using a point-to-line metric," in *Proceedings of the International Conference on Robotics and Automation*. IEEE, 2008, pp. 19–25.

[7] G. Grisetti, R. Kummerle, C. Stachniss, and W. Burgard, "A Tutorial on Graph-Based SLAM," *Intelligent Transportation Systems Magazine*, vol. 2, no. 4, pp. 31–43, 2010.

[8] T. O. Chan and D. Lichti, "Automatic In Situ Calibration of a Spinning Beam LiDAR System in Static and Kinematic Modes," *Remote Sensing*, vol. 7, no. 8, pp. 10 480–10 500, 2015.

[9] M. Bosse, R. Zlot, and P. Flick, "Zebedee: Design of a Spring-Mounted 3-D Range Sensor with Application to Mobile Mapping," *Transactions on Robotics*, vol. 28, no. 5, pp. 1104–1119, 2012.

[10] F. Pomerleau, S. Magnenat, F. Colas, M. Liu, and R. Siegwart, "Tracking a Depth Camera: Parameter Exploration for Fast ICP," in *Proceedings of the International Conference on Intelligent Robots and Systems*. IEEE, 2011, pp. 3824–3829.

[11] G. Grisetti, C. Stachniss, and W. Burgard, "Improving Grid-based SLAM with Rao-Blackwellized Particle Filters by Adaptive Proposals and Selective Resampling," in *Proceedings of the International Conference on Robotics and Automation*. IEEE, 2005, pp. 2432–2437.

[12] M. Montemerlo, S. Thrun, D. Koller, and B. Wegbreit, "FastSLAM: A Factored Solution to the Simultaneous Localization and Mapping Problem," in *Proceedings of the National Conference on Artificial Intelligence*. AAAI, 2002, pp. 593–598.

[13] M. Bosse and R. Zlot, "Continuous 3D Scan-Matching with a Spinning 2D Laser," in *Proceedings of the International Conference on Robotics and Automation*. IEEE, 2009, pp. 4312–4319.

[14] I. D. William Morris and J. Xiao, "3D Indoor Mapping for Micro-UAVs Using Hybrid Range Finders and Multi-Volume Occupancy Grids," in *Proceedings of the RSS Workshop on RGB-D: Advanced Reasoning with Depth Cameras*, 2010.

[15] S. Shen, N. Michael, and V. Kumar, "Autonomous Multi-Floor Indoor Navigation with a Computationally Constrained MAV," in *Proceedings of the International Conference on Robotics and Automation*. IEEE, 2011, pp. 20–25.

[16] A. Elfes, "Using Occupancy Grids for Mobile Robot Perception and Navigation," *Computer*, vol. 22, no. 6, pp. 46–57, 1989.

[17] B. Curless and M. Levoy, "A Volumetric Method for Building Complex Models from Range Images," in *Proceedings of the Annual Conference on Computer Graphics and Interactive Techniques*, ser. SIGGRAPH. ACM, 1996, pp. 303–312.

[18] G. Grisetti, C. Stachniss, and W. Burgard, "Improved Techniques for Grid Mapping With Rao-Blackwellized Particle Filters," *Transactions on Robotics*, vol. 23, pp. 34–46, 2007.

[19] M. Montemerlo and S. Thrun, "FastSLAM 2.0," *FastSLAM: A scalable method for the simultaneous localization and mapping problem in robotics*, pp. 63–90, 2007.

[20] S. Thrun and M. Montemerlo, "The GraphSLAM Algorithm With Applications to Large-Scale Mapping of Urban Structures," *International Journal on Robotics Research*, vol. 25, no. 5/6, pp. 403–430, 2005.

[21] A. Nüchter, K. Lingemann, J. Hertzberg, and H. Surmann, "6D SLAM 3D Mapping Outdoor Environments," *Journal of Field Robotics*, vol. 24, no. 8-9, pp. 699–722, 2007.

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

[23] J. Zhang and S. Singh, "LOAM: Lidar Odometry and Mapping in Real-time," in *Proceedings of Robotics: Science and Systems*, 2014, pp. 109–111.

[24] R. C. Smith and P. Cheeseman, "On the Representation and Estimation of Spatial Uncertainty," *International Journal of Robotics Research*, vol. 5, no. 4, pp. 56–68, 1986.

[25] S. Williams and I. Mahon, "Simultaneous Localisation and Mapping on the Great Barrier Reef," in *Proceedings of the International Conference on Robotics and Automation*, vol. 2. IEEE, 2004, pp. 1771–1776.

[26] D. Ribas, P. Ridao, J. D. Tardós, and J. Neira, "Underwater SLAM in Man-Made Structured Environments," *Journal of Field Robotics*, vol. 25, no. 11-12, pp. 898–921, 2008.

[27] J. Aulinas, Y. R. Petillot, X. Llad, J. Salvi, and R. Garcia, "Vision-based underwater SLAM for the SPARUS AUV," in *Proceedings of the International Conference on Computer and IT Applications in the Maritime Industries*, 2011, pp. 171–179.

[28] F. Dellaert and M. Kaess, "Square Root SAM: Simultaneous Localization and Mapping via square root information smoothing," *International Journal of Robotics Research*, vol. 25, no. 12, pp. 1181–1203, 2006.

[29] S. Thrun, Y. Liu, D. Koller, A. Y. Ng, Z. Ghahramani, and H. Durrant-Whyte, "Simultaneous Localization and Mapping with sparse extended information filters," *International Journal of Robotics Research*, vol. 23, no. 7-8, pp. 693–716, 2004.

[30] J.-H. Kim and S. Sukkarieh, "Airborne Simultaneous Localisation and Map Building," in *Proceedings of the International Conference on Robotics and Automation*, vol. 1. IEEE, 2003, pp. 406–411.

[31] J. Engel, T. Schöps, and D. Cremers, "LSD-SLAM: Large-Scale Direct Monocular SLAM," in *Proceedings of the European Conference on Computer Vision*. Springer, 2014, pp. 834–849.

[32] H. Moravec and A. Elfes, "High Resolution Maps from Wide Angle Sonar," in *Proceedings of the International Conference on Robotics and Automation*, vol. 2. IEEE, 1985, pp. 116–121.

[33] H. Moravec, "Robot Spatial Perception by Stereoscopic Vision and 3D Evidence Grids," *Perception*, 1996.

[34] M. Herbert, C. Caillas, E. Krotkov, I.-S. Kweon, and T. Kanade, "Terrain Mapping for a Roving Planetary Explorer," in *Proceedings of the International Conference on Robotics and Automation*. IEEE, 1989, pp. 997–1002.

[35] P. Pfaff and W. Burgard, "An Efficient Extension to Elevation Maps for Outdoor Terrain Mapping and Loop Closing," in *Proceedings of the International Conference on Field and Service Robotics*, 2005, pp. 165–176.

[36] P. Pfaff, R. Triebel, and W. Burgard, "An Efficient Extension to Elevation Maps for Outdoor Terrain Mapping and Loop Closing," *International Journal of Robotics Research*, vol. 26, no. 2, pp. 217–230, 2007.

[37] R. Triebel, P. Pfaff, and W. Burgard, "Multi-Level Surface Maps for Outdoor Terrain Mapping and Loop Closing," in *Proceedings of the International Conference on Intelligent Robots and Systems*. IEEE, 2006, pp. 2276 –2282.

[38] I. Dryanovski, W. Morris, and J. Xiao, "Multi-volume occupancy grids: An efficient probabilistic 3D mapping model for micro aerial vehicles," in *Proceedings of the International Conference on Intelligent Robots and Systems*. IEEE, 2010, pp. 1553–1559.

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

[40] K. M. Wurm, A. Hornung, M. Bennewitz, C. Stachniss, and W. Burgard, "OctoMap: A Probabilistic, Flexible, and Compact 3D Map Representation for Robotic Systems," in *Proceedings of the ICRA Workshop on Best Practice in 3D Perception and Modeling for Mobile Manipulation*, vol. 2. IEEE, 2010.

[41] R. A. Newcombe, S. Izadi, O. Hilliges, D. Molyneaux, D. Kim, A. J. Davison, P. Kohli, J. Shotton, S. Hodges, and A. W. Fitzgibbon, "KinectFusion: Real-Time Dense Surface Mapping and Tracking," in *Proceedings of the International Symposium on Mixed and Augmented Reality*. IEEE, 2011, pp. 127–136.

[42] Y. Chen and G. Medioni, "Object Modelling by Registration of Multiple Range Images," *Image Vision Computing*, vol. 10, no. 3, pp. 145–155, 1992.

[43] K.-L. Low, "Linear Least-Squares Optimization for Point-to-Plane ICP Surface Registration," *Chapel Hill, University of North Carolina*, vol. 4, 2004.

[44] A. Segal, D. Haehnel, and S. Thrun, "Generalized-ICP," in *Proceedings of Robotics: Science and Systems*, vol. 2, no. 4, 2009, p. 435.

[45] B. D. Lucas and T. Kanade, "An Iterative Image Registration Technique with an Application to Stereo Vision," in *Proceedings of the International Joint Conference on Artificial Intelligence*, 1981, pp. 674–679.

[46] S. Thrun, W. Burgard, and D. Fox, "A Real-Time Algorithm for Mobile Robot Mapping With Applications to Multi-Robot and 3D Mapping," in *Proceedings of the International Conference on Robotics and Automation*, vol. 1. IEEE, 2000, pp. 321–328.

[47] H. Zhao and R. Shibasaki, "Reconstructing Urban 3D Model using Vehicle-borne Laser Range Scanners," in *Proceedings of the International Conference on 3D Digital Imaging and Modeling*, 2001, pp. 349–356.

[48] C. Früh and A. Zakhor, "An Automated Method for Large-Scale, Ground-Based City Model Acquisition," *International Journal of Computer Vision*, vol. 60, no. 1, pp. 5–24, 2004.

[49] P. Kohlhepp, P. Pozzo, M. Walther, and R. Dillmann, "Sequential 3D-SLAM for mobile action planning," in *Proceedings of the International Conference on Intelligent Robots and Systems*, vol. 1. IEEE, 2004, pp. 722–729.

[50] D. Cole and P. Newman, "Using Laser Range Data for 3D SLAM in Outdoor Environments," in *Proceedings of the International Conference on Robotics and Automation*. IEEE, 2006, pp. 1556 –1563.

[51] D. Borrmann, J. Elseberg, K. Lingemann, A. Nüchter, and J. Hertzberg, "Globally consistent 3D mapping with scan matching," *Robotics and Autonomous Systems*, vol. 56, no. 2, pp. 130–142, 2008.

[52] J. Elseberg, S. Magnenat, R. Siegwart, and A. Nüchter, "Comparison of nearest-neighbor-search strategies and implementations for efficient shape registration," *Journal of Software Engineering for Robotics*, vol. 3, no. 1, pp. 2–12, 2012.

[53] S. May, P. Koch, R. Koch, C. Merkl, C. Pfitzner, and A. Nuechter, "A Generalized 2D and 3D Multi-Sensor Data Integration Approach based on Signed Distance Functions for Multi-Modal Robotic Mapping," in *Proceedings of the International Symposium on Vision, Modeling, and Visualization*, 2014, pp. 95–102.

[54] E. Bylow, J. Sturm, C. Kerl, F. Kahl, and D. Cremers, "Real-Time Camera Tracking and 3D Reconstruction Using Signed Distance Functions," in *Proceedings of Robotics: Science and Systems*, vol. 2, 2013, p. 35.

[55] A. J. L. Daniel R. Canelhas, Todor Stoyanov, "SDF Tracker: A Parallel Algorithm for Online Pose Estimation and Scene Reconstruction From Depth Images," in *Proceedings of the International Conference on Intelligent Robots and Systems*. IEEE, 2013, pp. 3671–3676.

[56] A. Doucet, N. de Freitas, K. Murphy, and S. Russell, "Rao-Blackwellised Particle Filtering for Dynamic Bayesian Networks," in *Proceedings of the Annual Conference on Uncertainty in Artificial Intelligence*. Morgan Kaufmann, 2000, pp. 176–183.

[57] R. Sim, P. Elinas, M. Griffin, J. J. Little *et al.*, "Vision-based SLAM using the Rao-Blackwellised Particle Filter," in *Proceedings of the IJCAI Workshop on Reasoning with Uncertainty in Robotics*, vol. 14, no. 1, 2005, pp. 9–16.

[58] J. Welle, D. Schulz, T. Bachran, and A. B. Cremers, "Optimization Techniques for Laser-Based 3D Particle Filter SLAM," in *Proceedings of the International Conference on Robotics and Automation*. IEEE, 2010, pp. 3525–3530.

[59] D. Törnqvist, T. B. Schön, R. Karlsson, and F. Gustafsson, "Particle Filter SLAM with High Dimensional Vehicle Model," *Journal of Intelligent and Robotic Systems*, vol. 55, no. 4-5, pp. 249–266, 2009.

[60] F. Lu and E. Milios, "Globally Consistent Range Scan Alignment for Environment Mapping," *Autonomous robots*, vol. 4, no. 4, pp. 333–349, 1997.

[61] R. Mur-Artal, J. M. M. Montiel, and J. D. Tardos, "ORB-SLAM: a Versatile and Accurate Monocular SLAM System," *Transactions on Robotics*, vol. 31, no. 5, pp. 1147–1163, 2015.

[62] F. Endres, J. Hess, N. Engelhard, J. Sturm, D. Cremers, and W. Burgard, "An Evaluation of the RGB-D SLAM System," in *Proceedings of the International Conference on Robotics and Automation*. IEEE, 2012, pp. 1691–1696.

[63] C. Kerl, J. Sturm, and D. Cremers, "Dense Visual SLAM for RGB-D Cameras," in *Proceedings of the International Conference on Intelligent Robots and Systems*. IEEE, 2013, pp. 2100–2106.

[64] L. Carlone, R. Aragues, J. A. Castellanos, and B. Bona, "A Linear Approximation for Graph-Based Simultaneous Localization and Mapping," in *Proceedings of Robotics: Science and Systems*, 2012, pp. 41–48.

[65] S. Thrun and M. Montemerlo, "The GraphSLAM Algorithm with Applications to Large-Scale Mapping of Urban Structures," *International Journal of Robotics Research*, vol. 25, no. 5-6, pp. 403–429, 2006.

[66] R. Kümmerle, G. Grisetti, H. Strasdat, K. Konolige, and W. Burgard, "g^2o: A General Framework for Graph Optimization," in *Proceedings of the International Conference on Robotics and Automation*. IEEE, 2011, pp. 3607–3613.

[67] M. Quigley, K. Conley, B. P. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "ROS: an open-source Robot Operating System," in *Proceedings of the ICRA Workshop on Open Source Software*, 2009.

[68] 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.

[69] S. Alers, D. Bloembergen, D. Claes, J.-D. Fossel, D. Hennes, and K. Tuyls, "Telepresence Robots as a Research Platform for AI." in *Proceedings of the Spring Symposium: Designing Intelligent Robots*. AAAI, 2013.

[70] J. Fossel, D. Hennes, S. Alers, D. Claes, and K. Tuyls, "OctoSLAM: A 3D Mapping Approach to Situational Awareness of Unmanned Aerial Vehicles (Demonstration)," in *Proceedings of the International Conference on Adaptive Agents and Multiagent Systems*, 2013.

[71] D. Claes, J. Fossel, B. Broecker, D. Hennes, and K. Tuyls, "Development of an Autonomous RC-car," in *Proceedings of the International Conference on Intelligent Robotics and Applications*. Springer, 2013, pp. 108–120.

[72] J. Fossel, D. Hennes, D. Claes, S. Alers, and K. Tuyls, "OctoSLAM: A 3D Mapping Approach to Situational Awareness of Unmanned Aerial Vehicles," in *Proceedings of the International Conference on Unmanned Aircraft Systems*. IEEE, 2013, pp. 179–188.

[73] S. Alers, D. Claes, J. Fossel, D. Hennes, K. Tuyls, and G. Weiss, "How to win Robocup@Work?" in *Proceedings of the Robot Soccer World Cup*. Springer, 2013, pp. 147–158.

[74] S. Alers, D. Claes, J. Fossel, D. Hennes, and K. Tuyls, "Applied robotics: precision placement in Robocup@Work," in *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems*, 2014, pp. 1681–1682.

[75] B. Broecker, D. Claes, J. Fossel, and K. Tuyls, "Winning the Robocup@Work 2014 Competition: The smARTLab Approach," in *Proceedings of the RoboCup 2014: Robot World Cup XVIII*. Springer, 2015, pp. 142–154.

[76] J.-D. Fossel, K. Tuyls, and J. Sturm, "2D-SDF-SLAM: A Signed Distance Function based SLAM Frontend for Laser Scanners," in *Proceedings of the International Conference on Intelligent Robots and Systems*. IEEE, 2015, pp. 1949–1955.

[77] J. Fossel, K. Tuyls, B. Schnieders, D. Claes, and D. Hennes, "NOctoSLAM: Fast octree surface normal mapping and registration," in *Proceedings of the International Conference on Intelligent Robots and Systems*. IEEE, 2017, pp. 6764–6769.

[78] B. Horn, "Closed-form solution of absolute orientation using unit quaternions," *Journal of the Optical Society of America*, vol. 4, no. 4, pp. 629–642, 1987.

[79] M. Muja and D. G. Lowe, "Fast approximate nearest neighbors with automatic algorithm configuration," in *Proceedings of the International Conference on Computer Vision Theory and Application*, 2009, pp. 331–340.

[80] S. Rusinkiewicz and M. Levoy, "Efficient variants of the icp algorithm," in *Proceedings of the International Conference on 3-D Digital Imaging and Modeling*. IEEE, 2001, pp. 145–152.

[81] G. H. Golub and C. Reinsch, "Singular Value Decomposition and Least Squares Solutions," *Numerische Mathematik*, vol. 14, no. 5, pp. 403–420, 1970.

[82] M. C. Seiler and F. A. Seiler, "Numerical Recipes in C: The Art of Scientific Computing," *Risk Analysis*, vol. 9, no. 3, pp. 415–416, 1989.

[83] M. Holmes, A. Gray, and C. Isbell, "Fast SVD for large-scale matrices," in *Proceedings of the NIPS Workshop on Efficient Machine Learning*, vol. 58, 2007, pp. 249–252.

[84] S. Thrun, W. Burgard, and D. Fox, *Probabilistic Robotics*. MIT Press, 2005.

[85] 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.

[86] K. M. Wurm, D. Hennes, D. Holz, R. B. Rusu, C. Stachniss, K. Konolige, and W. Burgard, "Hierarchies of Octrees for Efficient 3D Mapping," in *Proceedings of the International Conference on Intelligent Robots and Systems*. IEEE, 2011, pp. 4249–4255.

[87] A. S. Glassner, *An Introduction to Ray Tracing*. Elsevier, 1989.

[88] F. Steinbrücker, J. Sturm, and D. Cremers, "Volumetric 3D Mapping in Real-Time on a CPU," in *Proceedings of the International Conference on Robotics and Automation*. IEEE, 2014, pp. 2021–2028.

[89] S. Yeong, L. King, and S. Dol, "A Review on Marine Search and Rescue Operations Using Unmanned Aerial Vehicles," *International Journal of Mechanical, Aerospace, Industrial, Mechatronic and Manufacturing Engineering*, vol. 9, no. 2, pp. 396–399, 2015.

[90] M. Silvagni, A. Tonoli, E. Zenerino, and M. Chiaberge, "Multipurpose UAV for search and rescue operations in mountain avalanche events," *Geomatics, Natural Hazards and Risk*, vol. 8, no. 1, pp. 18–33, 2017.

[91] A. Nedjati, B. Vizvari, and G. Izbirak, "Post-earthquake response by small UAV helicopters," *Natural Hazards*, vol. 80, no. 3, pp. 1669–1688, 2016.

[92] S. Chen, D. Laefer, and E. Mangina, "State of Technology Review of Civilian UAVs," *Recent Patents on Engineering*, vol. 10, no. 3, pp. 160–174, 2016.

[93] N. Engelhard, F. Endres, J. Hess, J. Sturm, and W. Burgard, "Real-time 3d visual SLAM with a hand-held RGB-D camera," in *Proceedings of the European Robotics Forum RGB-D Workshop on 3D Perception in Robotics*, vol. 180, 2011, pp. 1–15.

[94] I. Dryanovski, W. Morris, and J. Xiao, "An Open-Source Pose Estimation System for Micro-Air Vehicles," in *Proceedings of the International Conference on Robotics and Automation*. IEEE, 2011, pp. 4449–4454.

[95] N. Koenig and A. Howard, "Design and Use Paradigms for Gazebo, An Open-Source Multi-Robot Simulator," in *Proceedings of the International Conderence on Intelligent Robots and Systems*, vol. 3. IEEE, 2004, pp. 2149–2154.

[96] W. E. Deming, "Statistical adjustment of data," 1964, published in the United Kingdom by Constable and Company, London.

[97] J. Servos and S. L. Waslander, "Multi Channel Generalized-ICP," in *Proceedings of the International Conference on Robotics and Automation*, 2014, pp. 3644–3649.

[98] R. B. Rusu, "Semantic 3D Object Maps for Everyday Manipulation in Human Living Environments," Ph.D. dissertation, Computer Science department, Technische Universität München, Germany, 2009.

[99] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun, "Vision meets robotics: The KITTI dataset," *International Journal of Robotics Research*, vol. 32, no. 11, pp. 1231–1237, 2013.