Institute for Computer Science VII
Robotics and Telematics

Master's thesis

An optimized polygon based SLAM algorithm for mobile mapping using efficient planar clustering and enhanced 6D gradient descent

Fabian Arzberger

September 2021

First reviewer:	Prof. Dr. Andreas Nüchter
Second reviewer:	Prof. Dr. Klaus Schilling
Advisor:	Dr. Dorit Borrmann

Abstract

In todays world, mobile robots are used in a variety of ways for the application of simultaneous localization and mapping (SLAM). To perform SLAM, an algorithm must register subsequent scans, taken from different locations, into one coordinate system. Usually, an estimation of the current pose of such a robot is done by a global navigation satellite system (GNSS), odometry, and/or an inertial measurement unit (IMU). The better the pose estimate is, the easier it becomes to perform SLAM. However, when pose estimation with GNSS or odometry is not available, one has to rely only on IMU-based methods, which is sensitive to errors. Additionally, many mobile robots can not guarantuee a slow, controlled motion, which obscures the pose estimation further, making SLAM more difficult. Recent publications from the European Space Agency (ESA) presage the rise of spherical robots for mobile mapping, due to advantages regarding locomotion and sensor protection. Despite these advantages, the spherical format suffers from the abovementioned difficulties regarding SLAM, introducing the need for a robust registration algorithm. This thesis proposes a novel 3D point cloud registration method for robotic mobile mapping systems using SLAM, based on planar polygon matching in six degrees of freedom (DoF). The proposed algorithm needs only geometric information, i.e., a point cloud and coarse pose estimations from an error-prone IMU. With these inputs, the algorithm corrects the 6 DoF path of the mobile system, improves map quality, and creates a global planar model of the scene. Furthermore, this thesis proposes a novel segmentation method for 3D point clouds, that utilizes an efficient adaption of k-d trees, called Bkd-trees, for efficient point to cluster distance calculation. The Bkd-tree adds efficient insertion and deletion of points while preserving excellent query performance at all times. An extensive evaluation reveals challenges and open problems for future research, e.g., the lack of a loop closing technique. However, the results also show the relevance of the presented algorithm for mobile mapping systems with different motion profiles and scanning patterns.

Zusammenfassung

In der heutigen Welt werden mobile Roboter auf vielfältige Weise für die Anwendung der simultanen Lokalisierung und Kartierung (SLAM) eingesetzt. Um SLAM durchzuführen, muss ein Algorithmus aufeinanderfolgende Scans, die von verschiedenen Standorten aus aufgenommen wurden, in einem Koordinatensystem registrieren. In der Regel erfolgt eine Schätzung der aktuellen Position eines solchen Roboters durch ein globales Satellitennavigationssystem (GNSS), Odometrie und/oder eine Inertialmesseinheit (IMU). Je besser die Schätzung der Lage ist, desto einfacher ist die Durchführung von SLAM. Wenn jedoch keine Positionsbestimmung mit GNSS oder Odometrie zur Verfügung steht, muss man sich auf IMU-basierte Methoden verlassen, die sehr fehleranfällig sind. Darüber hinaus können viele mobile Roboter keine langsame, kontrollierte Bewegung garantieren, was die Posenschätzung weiter erschwert und SLAM noch schwieriger macht. Jüngste Veröffentlichungen der Europäischen Weltraumorganisation (ESA) lassen den Aufschwung von kugelförmigen Robotern für die mobile Kartierung erwarten, da sie Vorteile bei der Fortbewegung und beim Schutz der Sensoren bieten. Trotz dieser Vorteile leidet das sphärische Format unter den oben erwähnten Schwierigkeiten in Bezug auf SLAM, was die Notwendigkeit eines robusten Registrierungsalgorithmus mit sich bringt. Die vorliegende Arbeit zeigt eine neue Registrierungsmethode in sechs Freiheitsgraden für 3D Punktwolken, gedacht für den Einsatz mit mobilen Robotersystemen. Der vorgestellte Algorithmus basiert auf flachen Polygonen, die mit einer Bewertungsfunktion einander zugeordnet werden. Außerdem benötigt der Algorithmus nur die Punktwolke und eine grobe Poseschätzung für eine Pfadkorrektur in sechs Freiheitsgraden, Verbesserung der Karte, und ein globales Ebenenmodel. Dafür benutzt der Algorithmus eine ebenfalls neue Methode zur Segmentierung von Punktwolken, basierend auf einer Adaption des kd-Baumes, welche das Einfügen und Entfernen von Punkten ermöglicht ohne die exzellente Abfragekomplexität zu verschlechtern. Eine ausführliche Auswertung offenbart Herrausforderungen und offene Probleme für zukünftige Arbeiten, beispielsweise wurde keine Methode zum Schliessen von Schleifen implementiert. Allerdings zeigen die Ergebnisse auch die Relevanz des vorgestellten Algorithmus für alle möglichen Arten von mobilen Robotersystemen, mit verschiedenen Bewegungs- und Scanmustern.

Danksagung

An dieser Stelle möchte ich all den Personen danken, die in irgendeiner Weise an dieser Arbeit mitgewirkt haben. Zuerst ein großes Dankeschön an Dorit Borrmann, die auch außerhalb ihrer Aufgabe als Betreuerin immer für ertragreiche Diskussionen zu haben war und mir bei einigen technischen Details den Weg weisen konnte. Außerdem spreche ich ein ebenso großes Dankeschön an einen Komillitonen und Freund aus, Jasper Zevering. Deine Arbeit für die benutzten Prototypen war ebenso unersetzlich wie die zahlreichen Diskussionen über viele Themen. Ebenso möchte ich einem anderen Arbeitskollegen und Freund danken, Anton Bredenbeck, dessen Simulator für die Ergebnisse der Arbeit benutzt wurde. Ein spezieller Dank gilt außerdem meiner Familie. Vielen Dank an meine Eltern und Großeltern. Ihr habt mein gesamtes Studium durch bedingungslose Unterstützung in jeder Hinsicht bereichert. Deshalb möchte ich euch gern diese Arbeit widmen.

Contents

List of Figures

List of Tables

List of Symbols

$\boldsymbol{p_k} = [x_k, y_k, z_k]$	A point with index k, consisting of the coordinates x_k, y_k, and z_k.
φ	Roll angle, i.e., rotation around x axis
θ	Pitch angle, i.e., rotation around y axis
ψ	Yaw angle, i.e., rotation around z axis
t_x	Translation on the x axis
t_y	Translation on the y axis
t_z	Translation on the z axis
$T(\boldsymbol{p_k})$	Homogeneous transformation defined by t_x, t_y, t_z, φ, θ, and ψ of the point $\boldsymbol{p_k}$
$\mathcal{P}_i$	An infinetly extending plane with index i.
P_i	A polygon with index i.
$D_h^{i,k}$	Hesse distance between point $\boldsymbol{p_k}$ and plane $\mathcal{P}_i$.
$D_{\text{PPD}}^{i,k}$	Projected polygon distance between point $\boldsymbol{p_k}$ and polygon P_i.
$\hat{\alpha}(\boldsymbol{n}_1, \boldsymbol{n}_2)$	Smallest angle between two normal vectors $\boldsymbol{n}_1$ and $\boldsymbol{n}_2$.
K	Number of nearest neighbors for normal calculation and clustering.
S	Number of subsequent scans put together in one linescan.
I	Number of gradient descent iterations before updating the correspondence model.
J	Number of times how hoften the correspondence model is updated.
ε_{H}	Threshold for Hesse distance $D_h^{i,k}$.
ε_{PPD}	Threshold for projection poylgon distance $D_{\text{PPD}}^{i,k}$.
ε_α	Threshold for the smallest angle between normals $\hat{\alpha}(\boldsymbol{n}_1, \boldsymbol{n}_2)$.
n_{min}^c	Minimum number of points allowed in a cluster.
d_{growth}	Growth radius for clustering.
$\boldsymbol{\alpha_0}$	A 6D vector holding the optimization weights for each dimension.

Chapter 1

Introduction

Today's robots for mobile mapping come in all shapes and sizes. State of the art for urban environments are laser scanners mounted to cars. Smaller robotic systems are particularly used when cars no longer have access. Examples for this are human-operated systems such as Zebedee [9], a small Hokuyo 2D scanner on a spring, that is carried through the environment, VILMA [30], a rolling FARO scanner operating in profiler mode, RADLER [8], a SICK 2D laser scanner mounted to a unicycle, or a backpack-mounted "personal laser scanning system" as in [29] or [31]. Recently more and more autonomous systems gained maturity. A stunning example is Boston Dynamics' quadruped "Spot" that autonomously navigates and maps human environments [10]. Also, the mobile mapping approaches implemented on the ANYmal platform such as [16] were very successful.

1.1 Mobile 3D Mapping with Spherical Robots

Of all these formats, one has not been explored thoroughly in the scientific community: The spherical mobile mapping robot. In fact, the spherical format imposes many challenges on current state-of-the-art techniques used for mapping. Section 1.2 names some of these challenges in detail. Nevertheless, let us first dive into some very promising advantages of spherical robots over other formats. For one, the locomotion of a spherical robot inherently results in rotation. That way, a sensor fixed inside the spherical structure will cover the entire environment, given the required locomotion without the need for additional actuators for the sensors. This requires a solution for the spherical simultaneous localization and mapping (SLAM) problem, given the six degrees of freedom of the robot. Secondly, a spherical shell that encloses all sensors protects these from possible hazardous environments. For example, the shell stops any dust that deteriorates sensors or actuators when settling at sensitive locations. In contrast to a usual enclosing the shell can separate the sensors from the environment without the need for several points of connection. A strict requirement then is that the shell is very durable. This is particularly useful for unknown or dangerous environments. For example, old buildings that are in danger of collapsing, narrow underground tunnels, construction sites, or mining shafts. The spherical format is, in fact, also suited for space applications. In the DAEDALUS study [45], such a robot is proposed that is to be lowered into a lunar cave and create a 3D map of the environment. The authors choose this

format as the moon regolith is known to damage instruments and other components. They also present an approach to protect the shell from accumulating dust and dirt.

1.2 Problem Definition

To perform SLAM, an algorithm must register subsequent scans, taken from different locations at different times, into one coordinate system. Usually, an estimation of the current pose of such a robot is done by GNSS, odometry, and/or IMUs. The better the pose estimate is, the easier it becomes to perform SLAM. However, when pose estimation with GNSS or odometry is not available, one has to rely only on IMU-based methods. Relying on IMU-based position measurements as a localization technique alone yields inaccurate and noisy pose measurements. Additionally, many mobile robots can not guarantuee a slow, controlled motion, which obscures the pose estimation further, making SLAM more difficult. Lidar sensors often have a minimum scan distance, resulting in a less dense (sometimes even empty) point cloud when the scanner looks on the ground, whereas density is higher when looking in other directions. The ground itself is likely to be less populated with points, due to weak angles of incidence while mapping it with spherical robots. Therefore, a robust registration procedure is needed for spherical robots, that can cope with vast differences in point cloud density and high noise regarding pose measurements. Although this thesis is heavily motivated by the spherical format, the aforementioned problems also arise in many other mobile mapping systems. Hence the presented results apply to many different motion profiles, too. Yet spherical robots suffer from these difficulties the most, which is why the majority of the presented results have been generated with such.

1.3 Scientific Contribution

The thesis proposes a SLAM algorithm, which identifies planar polygons in each scan. Such planar polygons form by clustering the points in the scan and then calculating the convex hull of these clusters. Point-to-plane correspondences then arise by finding similarities between polygons in each subsequent scan. Then, an optimization of the 6 DoF pose of the mobile robot leads to minimization of point-to-plane distances. This thesis has two major scientific contributions:

- A novel point cloud registration method that uses a score function to find similarities between planar convex polygons for globally consistent alignment. It needs only geometric information, i.e., the plain 3D point cloud, and a coarse pose estimate for each scan and outputs an improved map, improved 6 DoF path, and global plane model. The presented method copes with multiple scanning patterns and motion profiles, as well as high noise levels regarding 6 DoF pose measurements, vast differences in point cloud density in each scan, and huge scan size differences between subsequent scans. The algorithm utilizes multiple distance models to establish point-to-plane correspondences. Then, it uses AdaDelta for 6 DoF optimization, minimizing point-to-plane distances.

- A novel region growing based point cloud segmentation method which utilizes a dynamically scalable, fully space utilized modification to the k-d tree (the Bkd-tree) for calculating

point-to-cluster distances. Note that the Bkd-tree maintains its balance with minimum effort, regardless of the number of point insertions or deletions. However, the dynamic updates on the data structure do not deteriorate the runtime performance of the segmentation due to the improvement of query performance.

1.4 Thesis Outline

The following chapters introduce you to both aforementioned contributions. Mostly, the sections cover those two independently from the other. Chapter 2 presents the state-of-the-art for point cloud segmentation in Section 2.1, while Section 2.2 covers only state of the art SLAM algorithms. Chapter 5 presents the results in the same way: Section 5.1 shows segmentation results, whereas Section 5.2 shows only registration results. However, the two methods are designed to work together, which is why a strict separation does not always make sense. Therefore, Chapter 3 presents the mathematical principles used for both the segmentation and registration procedures. Chapter 4 presents the registration procedure as a whole and, therefore, also includes the segmentation part in Section 4.4. This emphasizes the guiding thread of the chapter and thus it improves readability and understandability. Finally, Chapter 6 draws a conclusion by outlining the strengths and major drawbacks of the algorithms.

Chapter 2

State of the Art

The following sections introduce you to the state-of-the-art (SOTA) methods for point cloud segmentation and registration. As the presented algorithm works by finding planar polygons, section 2.1 presents different approaches for point cloud segmentation. Then, Section 2.2 introduces other SOTA methods for SLAM in 6 DoF for mobile mapping. As this thesis uses point-to-plane based correspondences for point cloud registration, Section 2.3 mentions other algorithms that use this approach.

2.1 Planar Segmentation Algorithms

The subject of clustering 3D point clouds based on certain geometrical properties is a thoroughly studied field of research. Nguyen et al. [37] give a broad overview on the topic. They subdivide the 3D point cloud segmentation problem into five categories, as shown in Figure 2.1: edge-based, region-based, attribute-based, model-based, and graph-based methods.

Edge-based methods segment points utilizing local regions of high contrast, i.e., sharp features. This is often done with gradient-based methods [5] or binary edge maps [48]. Since these methods rely on local qualities like high local intensity differences, they are all very sensitive to

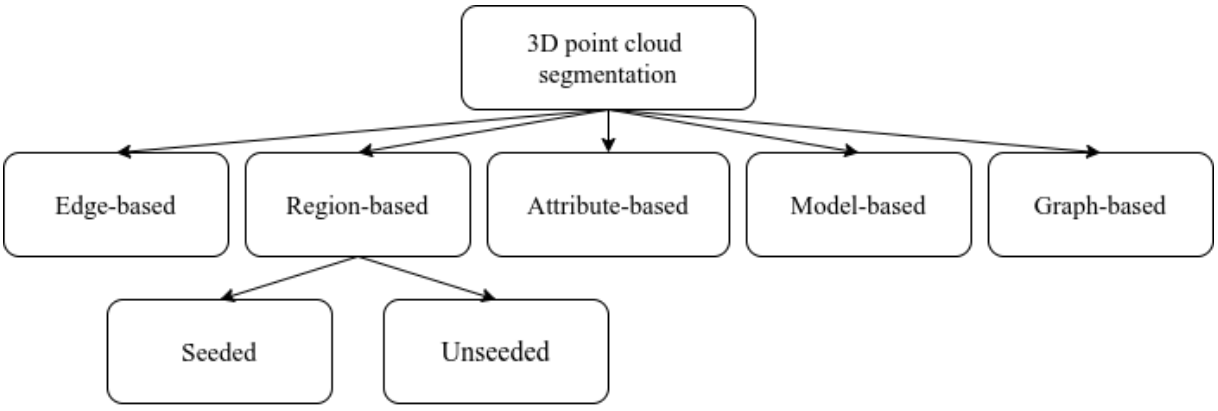


Figure 2.1: Taxonomy of 3D point cloud segmentation methods. Source: Nguyen et al. [37]

noise and in uneven density point clouds.

Region-based methods work by grouping neighboring points together if they have similar properties. There are, in general, two ways of approaching this problem:

- Bottom-up (Seeded-region) methods, where an initial seed of points is used to grow similar regions. An algorithm that implements a seeded region growing was first introduced by Besl et al. [4], where the selection of initial seed points is based on the curvature of each point. Ning et al. [38] adapt the algorithm by proposing a procedure with two stages: rough and detail segmentation, where the information of the normal vector of each point is utilized. Dorninger et al. [13] improve the time complexity of the algorithm by sequentializing the algorithm and utilizing course contour information. However, seeded-region approaches are highly sensitive to the choice of the initial seed points. Starting with an inaccurate seed leads to under- or over-segmentation of the point cloud.

- Top-down (Unseeded-region) methods, where the points are initially grouped in one region, which is then subdivided further into smaller, dissimilar regions. Chen et al. [11] use this approach to perform architectural modeling, introducing a function that expresses confidence in the likelihood that the local neighborhood is planar. Typically, unseeded approaches require prior knowledge about the scene, e.g. an object model, region size, or the number of desired regions.

Attribute-based methods assign properties to each point, or even regions of points, before clustering them together. Biosca et al. [6] introduce a methodology for combining multiple attributes based on unsupervised fuzzy cluster merging, however have difficulties with parameter tuning and time consumption. Filin et al. [20] use normal and height differences between neighboring points as attributes for segmenting airborne laser data. They define the local neighborhood adaptively using inherent properties of the data itself, i.e. distance, point density, and point distribution. It is a robust clustering technique that deals with outliers, noise, and uneven point density. The drawback with most attribute-based methods comes with high amounts of input points since computing multiple attributes is computationally burdensome especially when they are multidimensional.

Model-based methods assume perfect mathematical models to be underlain in the point data, such as cylinders, spheres, or, for the purposes of this thesis: planes. Regions of points with a matching mathematical description are therefore clustered together. The 3D Hough Transform, as well as RANSAC (RANdom SAmple Consensus), are widely known, frequently applied techniques that many algorithms use as a foundation for matching such models to the point data. RANSAC works by considering a set of random samples of n data points, where n is the number of points needed to calculate model parameters. Then, other data points are considered and joined the set if their distance to the model is sufficiently small. If the set has enough points, it is considered as a possible solution. After a predefined number of iterations, the set with the most points is chosen as a solution. The 3D Hough transform maps the input data to a parameter space, the so called Hough space. In this parameter space, the parameters correspond to an arbitrary model representation, e.g., a plane. Therefore, for each point and its local neighborhood, quantized model parameters are calculated. Then, the bin corresponding to these parameters is incremented. Finding the local maxima in parameter

space yields the best model representation of the underlying data. Tarsha-Kurdi et al. [53] compared the 3D Hough transform with RANSAC, segmenting rooftop planes from airborne laser-scan data. In their work, they found that RANSAC yields better segmentation results and is faster in runtime. However, this is hard to generalize, as the time complexity of both RANSAC and the Hough transform heavily depend on many parameters. With RANSAC, the bottleneck is the examination of all points, yet there are also variants which test only a subset. With the Hough transform, the discretization of the Hough space plays a decisive role, as making the bins too small leads to a higher runtime. There are numerous variants for the Hough transform. In particular, the Randomized Hough Transform (RHT) increases the efficiency tremendously. Gelfand et al. [23] use mathematical models of rotationally and translationally symmetric shapes, called "slippable" shapes. Those include sphere, helix, plane, cylinder, linear extrusions, and surfaces of revolution. The algorithm works by merging initial patches of slippable surfaces to create more complicated structures, and thus the quality of the segmentation relies on the size of the initial patches. Li et al. [32] introduce a global coupling algorithm, which is able to correct a set of initial local RANSAC primitives and align them globally. They do so by establishing relations between primitives such as orientation, placement, and equality, which are then iteratively amplified by constrained optimization. Model-based segmentation methods yield fast and accurate results while being able to deal with a large amount of noise and outliers. However, they are inaccurate when the point data does not fit a perfect mathematical model, or the model is unknown.

Graph-based methods work by representing the 3D point data with a graph, then apply algorithms from graph theory. Golovinskiy et al. [25] utilize the k-nearest neighbors (KNNs) of points to represent the point cloud as a graph. Their algorithm infers background and foreground constraints and then finds the min-cut to compute a foreground-background segmentation. Often, graph-based methods make use of random fields, such as Conditional Random Fields (CRF) and Markov Random Fields (MRF) [28], as they generate flexible stochastical image models. Rusu et al. [46] create fast point feature histograms (FPFH) [47] which encode geometrical surface information locally. Then they use CRF for contextual information to put the FPFHs together by defining classes of 3D geometric shapes. Tatavarti et al. [54] introduce a stochastical graph-based method consisting of three steps: The first two steps consist of detecting a candidate set of planar surfaces and merging the initial candidate set. The third step of their algorithm uses MRF on the point data and the planar model to compute the maximum a posteriori probability (MAP) of the segmentation labels using bayesian belief propagation (BBP). Their algorithm shows novelty since unlike other graph-based methods color information is not needed. Furthermore, the algorithm provides a tradeoff between segmentation performance and detail. Other graph based methods exists, though many of them use camera systems, e.g. [51], [49], and [19]. In general, graph-based methods perform well with noise, outliers, and uneven point density. However, they are usually not suited for real-time applications.

2.2 6D SLAM Algorithms For Mobile Mapping

Laser-based SLAM algorithms for motion in six degrees of freedom (DoF) have been thoroughly studied. For outdoor environments [39] provides a first baseline. Adding a heuristic for closed-

loop detection and a global relaxation Borrmann et al. yield highly precise maps of the scanned environment [7]. Zhang et al. propose a real-time solution to the SLAM problem in [58]. They achieve the performance at a lower computational load by dividing the SLAM algorithm into two different algorithms: one performing odometry at a high frequency but low fidelity and another running at a lower frequency performing fine matching and registration of the point clouds. More recently Dröschel et al. also propose an online method using a novel combination of a hierarchical graph structure with local multi-resolution maps to overcome problems due to sparse scans [14]. Another intriguing example is the NavVis VLX system [36]. They perform real-time SLAM on a mobile, wearable platform, producing colored point clouds with high accuracy by combining it with a camera system. However, this platform has to be moved by a human operator. They also employ artificial markers that can be placed in the environment, which get recognized by the camera system to further improve registration accuracy. The de-facto standard for many SLAM algorithms is the Iterative-Closest-Point (ICP) algorithm [3] that employs point-to-point correspondences using closest points, as the name suggests. Since these approaches are based on point-to-point correspondences, they require a rather high point density to achieve precise registration. For low-cost LiDARs, this implies slow motion and a long integration time. To overcome the requirements on point-density imposed by the point-to-point correspondences, instead other correspondences are used.

2.3 Point Cloud Registration With Plane Based Correspondences

In many environments, planes are abundantly available and hence provide an attractive base for correspondences. Pathak et al. [42] propose to reduce the complexity of the registration by using correspondences between planar patches instead of points. They demonstrate the effectiveness of their approach even with noisy data in cluttered environments. However, their approach is designed for data acquired in stop-scan-go fashion and not for mobile mapping applications. As they use a region growing procedure with randomized initialization for detecting planar patches the distortions in the data introduced by pose uncertainties are likely to affect the shape of the planar patches and lead to faulty correspondences. Förster et al. use the planarity of human-made environment successfully in [22]. They register point clouds using plane-to-plane correspondences and include uncertainty measures for the detected planes and the estimated motion. Thereby, they propose a costly exact algorithm and cheaper approximations that yield high-quality maps. Favre et al. [17] use point-to-plane correspondences after preprocessing the point clouds using plane-to-plane correspondences to register two scans with each other successfully. Both approaches use plane-to-plane correspondences to pre-register the scans. However, for pre-registration, the classical point-to-point registration is also very effective. One advantage that point-to-point correspondences have over plane-to-plane correspondences is that they do not require a long stop to obtain enough points to detect planes in each pose. For plane-to-plane correspondences, this is necessary to gather enough data to measure planes in each scan robustly. In particular, this pause is needed when the field-of-view (FOV) is limited, as the detected planes are thin slices of the true planes which are difficult to find correct correspondences for. The resulting scan procedure is stop-scan-and-go. However, a standstill in each pose cannot be guaranteed or even approximated for many mobile mapping systems, making continuous-time

approaches using point-to-plane correspondences the method of choice. LOAM [58] is the baseline algorithm that provides a real-time and low-drift solution based on two parallel registration algorithms using planes and lines. Unfortunately, it is not open-source anymore. LOAM livox extends the LOAM framework to the rotating prism scanner with small FoV [33]. Zhou, Wang, and Kaess write that it also adopts parallel computing to achieve real-time global registration. Parallel computing needs a powerful CPU, it may be not suitable for an embedded system that has limited computational resources [59]. Thus, they extend their smoothing and mapping to the LiDAR and planes case. In their experiments, they used a VLP-16 LiDAR to collect indoor datasets. Further recent planar SLAM approaches include [24, 26, 27, 55, 59]. While Wei et al. use only the ground plane in outdoor experiments. Indoors, Jung et al. used in 2015 several Hokuyo laser scanners for their kinematic scanning system [27], while Grant used a single Velodyne HDL-32E sensor with 32 rotating laser/receiver pairs mounted on a backpack and walked through different environments [26]. The LIPS system [24] employes a so-called Closest Point Plane Representation with an Anchor Plane Factor. RANSAC (RANdom SAmple Consensus) is used to find the planes. Their system couples an eight-channel Quanergy M8 LiDAR operating at 10Hz with a Microstrain 3DM-GX3-25 IMU attached to the bottom of the LiDAR operating at 500Hz. This thesis extends the state-of-the-art by adding a robust, yet flexible algorithm, especially designed for spherical robots with fully IMU-based pose estimation but also other mobile mapping systems with various scanning patterns and motion profiles. Unlike other previously mentioned methods, the motion of the mobile mapping system is not required to be in a stop-and-go fashion, but is also allowed to be continuous. The proposed algorithm is not yet able to perform in real time.

Chapter 3

Mathematical Fundamentals

This chapter introduces you to the math we need to model collections of 3D points, each one having an x, y, and z value, called point clouds. Further, planar polygons are introduced, which in this thesis arise from the convex hulls of similar point collections, called clusters. First, Section 3.1 sets up some basic terminology for 3D point representation and 6D transformations of point clouds. Second, Section 3.2 introduces two distance models that are used for establishing point-to-plane correspondences: the Hesse distance and the Polygon Projection distance (PPD). For the latter, we want to calculate the inclusion of a point in a polygon, which is described in Section 3.2.3 and 3.2.4. The Hesse distance and PPD are not only used for establishing point-to-plane correspondences, but also used in the next chapter to construct similarity criteria for polygons. Additionally, the thesis proposes a computationally efficient algorithm for point segmentation with region growing. The decrease in computational cost is achieved by an adaption of the well known k-d tree, called a Bkd-tree. Therefore, Section 3.3 explains how a Bkd-tree is constructed from multiple k-d trees.

3.1 Points and Transformations

We consider a sub-collection of a 3D point cloud as a scan. Further, we call a collection of scans a linescan. A scan consists of the points $\{\boldsymbol{p}_0, \boldsymbol{p}_1, ..., \boldsymbol{p}_{N-1}\}$, which the laser scanner measures in its own coordinate system (with the scanner located in the origin). To align multiple scans together, 6D pose information is needed to transform the points into a global reference frame. The transformation $T(\boldsymbol{p}_k)$ of any point $\boldsymbol{p}_k$ with respect to a 6 DoF motion is described in homogeneous coordinates using the roll-pitch-yaw ($\varphi - \vartheta - \psi$) Tait-Brian angles as in [12], as well as the translation of the point t_x, t_y, and t_z in x, y, and z direction respectively. Constructing the result from these homogeneous coordinates and using C_k and S_k to denote the cosine and sine of the angle in the subscript yields a left-handed transformation, applied to the point $\boldsymbol{p}_k$:

$$T(\boldsymbol{p}_k) = \begin{bmatrix} xC_\vartheta C_\psi - yC_\vartheta S_\psi + zS_\vartheta + t_x \\ x(C_\varphi S_\psi + C_\psi S_\varphi S_\vartheta) + y(C_\varphi C_\psi - S_\varphi S_\vartheta S_\psi) - zC_\vartheta S_\varphi + t_y \\ x(S_\varphi S_\psi - C_\varphi C_\psi S_\vartheta) + y(C_\psi S_\varphi + C_\varphi S_\vartheta S_\psi) + zC_\varphi C_\vartheta + t_z \end{bmatrix} \tag{3.1}$$

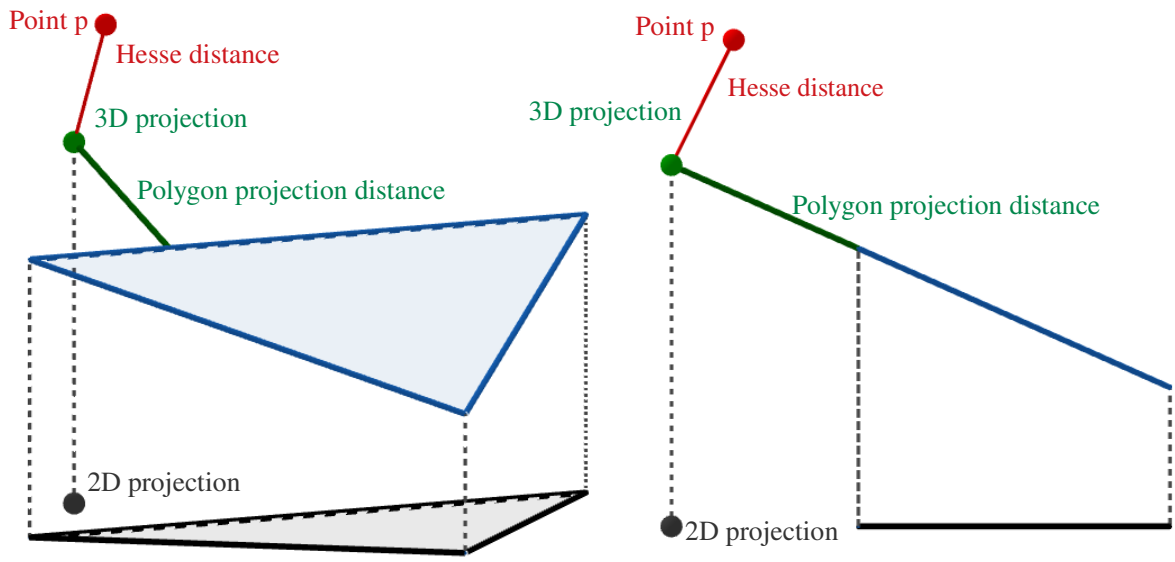


Figure 3.1: Illustration of the Hesse- and polygon projection distance. The point $\boldsymbol{p}$ (red) gets projected onto the infinitley extending global plane. Both the point projection (green) and the global planes convex hull (blue) get projected into 2D space (grey). The Hesse distance, i.e., the shortest distance to the infinitley extending plane (red), is shown as well as the minimum distance from the 3D point projection to the polygon (green).

3.2 Point to Plane Distance Models

The key to improving map quality is to find a model for each linescan, where each point is correctly identified to belong to one of the extracted planes. Then, the sub-scan is optimized in Section 4.6 by finding a 6D pose-transformation that minimizes the distance of each correspondence. The following sections describe two key ideas, which are important later for finding correspondences: the Hesse distance and the polygon projection distance (PPD).

3.2.1 Hesse Distance

The Hesse-normal form describes the distance $D_h^{i,k}$ from the k-th transformed point $T(\boldsymbol{p}_k)$ to an ever extending, infinetly large plane in 3D vector space with normal vector $\boldsymbol{n}_{\mathcal{P}_i}$ and supporting point $\boldsymbol{a}_{\mathcal{P}_i}$ of the i-th plane:

$$D_h^{i,k} = \boldsymbol{n}_{\mathcal{P}_i} \cdot [T(\boldsymbol{p}_k) - \boldsymbol{a}_{\mathcal{P}_i}], \tag{3.2}$$

The distance $D_h^{i,k}$ of the k-th point to the i-th plane reflects the length of the line segment, constructed from the k-th point to its projection on the i-th plane. It corresponds to the distance shown in red in Figure 3.1. Therefore, the line segment is parallel to the normal vector of the plane by definition. Thus, the k-th points projection $\widetilde{T}(\boldsymbol{p}_k)$ onto the i-th plane, which is required later, is easily calculated by shifting the point against normal direction:

$$\widetilde{T}(\boldsymbol{p}_k) = T(\boldsymbol{p}_k) - D_h^{i,k} \cdot \boldsymbol{n}_{\mathcal{P}_i} \tag{3.3}$$

3.2.2 Polygon Projection Distance

The simplicity of the Hesse distance in Section 3.2.1 is in contrast with its inability to represent the extent of the plane. In reality, planes will never be infinelty large, thus it becomes obvious that an additional distance model, based on the polygonic nature of the environments, is required. The convex hull is able of representing the expansion of the plane in all directions and thus is utilized as a distance model. A convex hull is a polygon that fulfills the following criteria:

- It is a 2D plane, which itself is embedded in a 3D space.
- It is constructed with line segments $\boldsymbol{s_{i,j}}$, connecting the edge points $\boldsymbol{p}_i$ and $\boldsymbol{p}_j$.
- It is convex, meaning there are no self-intersections, and no "curving into the polygon".

First, the corresponding point gets projected onto the ever-extending, infinitely large plane representation given by the Hesse form. Keeping track of the projection of the point $\widetilde{T}(\boldsymbol{p}_k)$, i.e., the green point from Figure 3.1, is essential for calculating a point's distance to a polygon in 3D. It represents the shadow of the point onto the same plane as the polygon. The projection of the point is given from Equation (3.3). Then, the 3D polygon, i.e., the points that make up the convex hull, as well as the corresponding 3D point projection, are projected again into a 2D vector space, using the direction of the plane. For example, in Figure 3.1, normal vector of the blue polygon has its major component in the z-direction (upwards), thus the projection polygon distance is calculated in the xy-plane. Using the maximum of the absolute magnitude of the normal vector ensures that the most sensible 2D projection is used for every direction. Considering the 2D projection has the advantage that algorithms exist in 2D space to check whether a point lies in or outside a polygon. Sunday [52] presents two approaches to this: the winding number (WN) algorithm, and the crossing number (CN) algorithm. The next two sections cover both algorithms in detail. In this section we use Sunday's algorithms to check whether a point is inside or outside of a polygon in 2D space.

If the points 2D projection, i.e., the grey point in Figure 3.1, is inside the polygon projection, the so called polygon projection distance (PPD) is set to zero. If, however, the point is outside the polygon in Figure 3.1, the shortest distance of the green point to the blue polygon is calculated. We look at the minimum distance of the query point $\boldsymbol{p_k}$ to each line segment $\boldsymbol{s_{i,j}}$, making up the convex hull of the plane. The line segment consists of the points $\boldsymbol{p_i}$ and $\boldsymbol{p_j}$. Note that for an ordered point set, the iteration process becomes straightforward by choosing j and i to fulfill j - $i = 1$, i.e., iterating the border points of the polygon.

Let $t \in [0, 1]$, then the line segment is parameterized by

$$\boldsymbol{s_{i,j}}(t) = \boldsymbol{p_i} + t \cdot (\boldsymbol{p_j} - \boldsymbol{p_i}) \,. \tag{3.4}$$

We set up a distance function $d_{i,j}(t)$, which measures the distance from the query point $\boldsymbol{p_k}$ to an arbitrary point on the line segment:

$$d_{i,j}(t) = \|\boldsymbol{s_{i,j}}(t) - \boldsymbol{p_k}\|. \tag{3.5}$$

It is now possible to find the shortest distance to the line segment by finding the argument of the function that minimizes this distance. Therefore, we set

$$\frac{\partial \left(d_{i,j}(t)\right)^2}{\partial t} \stackrel{!}{=} 0, \tag{3.6}$$

resulting in the argument t_0 which minimizes Equation (3.5)

$$t_0 = \frac{(\boldsymbol{p_j} - \boldsymbol{p_i}) \cdot (\boldsymbol{p_k} - \boldsymbol{p_i})}{(\boldsymbol{p_j} - \boldsymbol{p_i})^2}. \tag{3.7}$$

Note the possibility of $t_0 \notin [0, 1]$. In that case, the projection of the point $\boldsymbol{p_k}$ onto the line given by Equation (3.4) is not between $\boldsymbol{p_j}$ and $\boldsymbol{p_i}$. Instead, its projection onto the line falls outside of the segment. By constraining t_0 to be between zero and one, we get $\hat{t}$

$$\hat{t} = \min\left(\max\left(t_0, 0\right), 1\right), \tag{3.8}$$

which is the argument that gives the shortest distance to the line segment, when inserted into Equation (3.5). If t_0 was to be inserted into Equation (3.5), the resulting distance corresponds to the shortest distance to the ever extending line, given by $t \in]-\infty, \infty[$. By definition of the line in Equation (3.4), the line segement corresponds to $t \in [0, 1]$, which is why the constraint in Equation (3.8) makes sense. Algorithm 1 computes the shortest distance $D^{i,k}_{\text{PPD}}$ from a point $\boldsymbol{p_k}$ to a polygon P_i, called PPD.

Algorithm 1: Calculates the shortest distance from a point to a polygon in 3D (PPD)

Data: Query point $\boldsymbol{p_k}$, polygon P_i as an ordered set of line segments
Result: Shortest distance $D^{i,k}_{\text{PPD}}$ between $\boldsymbol{p_k}$ and P_i, the points p_1 and p_2

$D^{i,k}_{\text{PPD}} = \infty$;
for $\boldsymbol{s_{i,j}}(t) \in P$ where $j - i = 1$ **do**
  $t_0 = \frac{(\boldsymbol{p_j} - \boldsymbol{p_i}) \cdot (\boldsymbol{p_k} - \boldsymbol{p_i})}{(\boldsymbol{p_j} - \boldsymbol{p_i})^2}$ ▷ Find argument that minimizes squared distance function.;
  $\hat{t} = \min\left(\max\left(t_0, 0\right), 1\right)$ ▷ Constrain it to be between 0 and 1.;
  $\hat{d} = \|\boldsymbol{s_{i,j}}(\hat{t}) - \boldsymbol{p_k}\|$ ▷ Get minimum distance using distance function.;
  if $\hat{d} < d_{min}$ **then**
    $D^{i,k}_{\text{PPD}} = \hat{d}$ ▷ Store minimum distance along all line segments.;
    $\boldsymbol{p_1} = \boldsymbol{p_i}$ ▷ Save the points $\boldsymbol{p_i}$ and $\boldsymbol{p_j}$...;
    $\boldsymbol{p_2} = \boldsymbol{p_j}$ ▷ ...that make up the line segment.;
  end
end
return $D^{i,k}_{\text{PPD}}$, $\boldsymbol{p_1}$, $\boldsymbol{p_2}$

3.2.3 Crossing Number Algorithm

The crossing number (CN) algorithm checks if a point $\boldsymbol{p_k}$ is inside a polygon P_i by counting the intersections from a ray, starting from point $\boldsymbol{p_k}$, with the line segments of the polygon. The

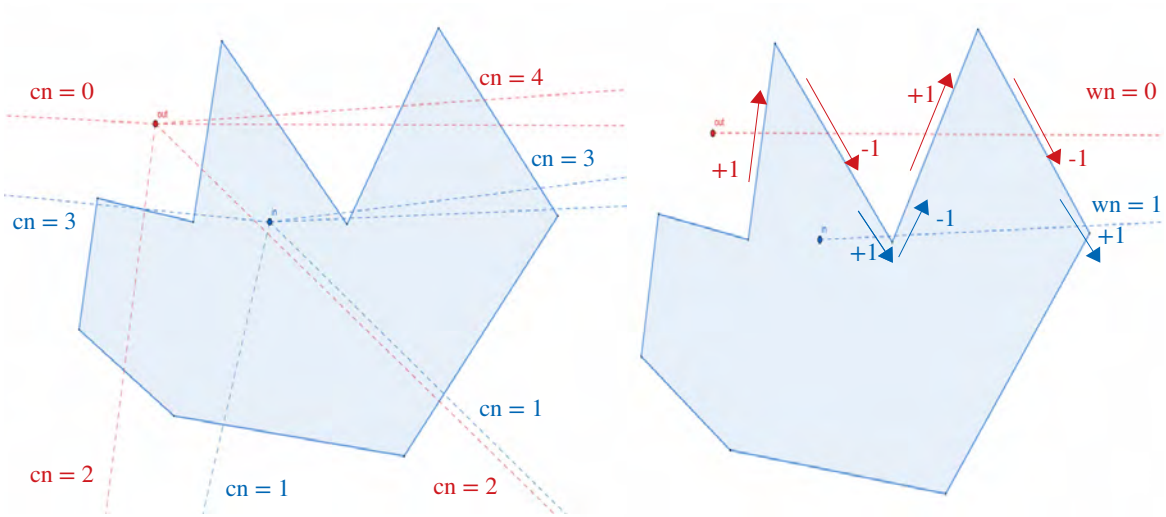


Figure 3.2: (Left) Illustration of the idea behind the CN algorithm. The CN value counts the intersections of the ray with the line segments that make up the polygon. (Right) Illustration of how the WN algorithm computes the winding of the polygon around the point in the most efficient way.

algorithm is to straightforward loop all the line segments that P_i is made of and check if the ray hits the line segment, and thus increment the corresponding CN value. At the end of the process, if CN is even, the point is outside of P_i. Otherwise it is inside P_i. Any infinitely long ray that is cast from $\boldsymbol{p_k}$ must eventually end up outside the polygon. Therefore, if $\boldsymbol{p_k}$ is inside P_i, the sequence of crossings must start with ”inside to outside” and end with ”inside to outside”. This occurs exactly when there is an odd number of crossings. On the other hand, if the point is outside P_i, the sequence of crossings must start with “outside to inside” and, again, end with “inside to outside”. This occurs exactly when there is an even number of crossings. Figure 3.2 illustrates this. Sunday [52] notes that this procedure only works with simple polygons, where self-overlap and holes are not allowed. Since we only use simple polygons, as ensured by the convex hull, the CN algorithm is always sufficient for our purpose. However, the presented algorithm works with a different approach, since its output gives a better intuition, even when applied to non-simple polygons.

3.2.4 Winding Number Algorithm

The winding number (WN) algorithm computes if a point $\boldsymbol{p_k}$ is inside the polygon P_i by calculating how many times P_i winds around the point. Let $t \in [0, 1]$, then $x(t)$ and $y(t)$ describe the x and y coordinates of a continuous, closed curve C, where $C(0) = C(1)$, describing the full polygon.

$$C(t) = C(x(t), y(t)) \ . \tag{3.9}$$

Keeping track of the normalized vector $\boldsymbol{w}(q,t)$, pointing from the query point $\boldsymbol{p_k}$ to any point $C(t)$ on the polygon, we get

$$\boldsymbol{w}(q,t) = \frac{C(t) - \boldsymbol{p_k}}{\|C(t) - \boldsymbol{p_k}\|} \, . \tag{3.10}$$

which performs a continuous, circular motion along the unit circle $U(t) = (cos(\Theta(t)), sin(\Theta(t)))$, where Θ is the current phase on the unit circle. We calculate the resulting rotation $n_w(\boldsymbol{p_k}, C)$ of the vector $\boldsymbol{w}(q,t)$, by integrating over the unit circle:

$$n_w(\boldsymbol{p_k}, C) = \frac{1}{2\pi} \oint_{U(t)} d\Theta = \frac{1}{2\pi} \int_0^1 \Theta(t) dt \, , \tag{3.11}$$

which is directly expressed as

$$n_w(\boldsymbol{p_k}, C) = \frac{1}{2\pi} \sum_{j=0}^{n-1} \Theta_j \, , \tag{3.12}$$

where Θ_j corresponds to the amount of winding that has happened in the j-th line segment. However, the calculation of that angle yields a computationally expensive arccos function and is therefore simplified. To do that, we abuse the fact that the polygon is an ordered set of points, where each sub-sequent point constructs a line segment of the polygon with the point before. Imagine casting a ray from the point $\boldsymbol{p_k}$, as in Figure 3.2. If a given line segment crosses the ray, you observe its direction corresponding to the ray, since there is a start and an endpoint for each line segment. Because of this, the direction is observed to be upwards or downwards with respect to the ray cast from $\boldsymbol{p_k}$. Then, it is sufficient to replace Θ_j with 1 if the direction is upwards, and -1 if the direction is downwards. Figure 3.2 gives an intuition on the working principle. Even if the substitution does not correspond to the actual value of Equation (3.11), it preserves correctness when we only seek to find if the point is inside the polygon or not: The point $\boldsymbol{p_k}$ is outside P_i if and only if the winding number is zero. The time complexity is $\mathrm{O}(n)$, where n is the number of points in the polygon. Note that the given time complexity is valid for all polygons, including non-simple ones. Sunday [52] points out that for convex-shaped polygons, the time complexity can be reduced further to $\mathrm{O}(\log(n))$. Note that Sunday [52] further points out that "the WN algorithm should always be preferred for determining the inclusion of a point in a polygon". The reason he gives for this is that the WN algorithm gives the correct result for non-simple polygons, i.e., polygons that overlap with each other, or polygons with holes in them. The CN algorithm sometimes gives a false result on these types of polygons. Since the polygon we work with is always convex shaped both methods will be guaranteed to produce the correct result. However, we leave further room for extension to concave polygons by using the WN algorithm.

3.3 k-d Trees and the Bkd-Tree

In the next chapter, a novel point cloud segmentation method is introduced, which benefits from a datastructure called Bkd-tree. To understand exactly how the proposed algorithm uses it, and why this leads to a benefit regarding runtime, we must first discuss what properties Bkd-trees have. A Bkd-tree, proposed by Procopiuc et al. [43], is an adaption of the well known k-d

tree, which is able to unfold a k-dimensional space in a binary tree. Let us first discuss the fundamentals of a k-d tree and show why an adaption is needed.

For the algorithms described in the next chapter, it is very beneficial to store a point cloud in such a k-d tree. Starting at the root node, which represents the bounding box of the whole point cloud, the tree subdivides the 3D space in one of the principal axis, i.e., x, y, or z-axis, thus creating a hyperplane that is perpendicular to that axis. Each node has two children, the first corresponding to one side, the second corresponding to the other side of the hyperplane. The median method yields a consistent strategy for finding an optimal division point, i.e., a division point that balances the tree. The 3D space is then subdivided further until each leaf node has only B points in it, which is called the bucketsize of the tree. Building the tree this way for N points costs $\mathrm{O}(N \log(N))$.

This tree structure is later exploited to search for the K-nearest neighbors (KNNs) of a point, as it is perfectly suited for eliminating large portions of the search space fast. To find the KNNs of a point, a recursive approach traverses the tree, where at each node the algorithm has to make one comparison for the corresponding axis. If a leaf node is reached, the recursion starts to unwind and explores the other branches of the tree. When KNNs have been selected during the recursion, branches are eliminated if they can not have points closer to these K-nearest. Therefore, the complexity of the KNN algorithm is $\mathrm{O}(K \log(N))$. In the next section, we discuss a problem that k-d trees have with dynamic updates, and show how the Bkd-tree solves this problem by maintaining multiple k-d trees, i.e., a k-d forest.

3.3.1 A Forest of Trees

Many approaches exist for extending the classical k-d tree structure in such a way that insertion and deletion of points do not cause performance degradation. The classic k-d tree was designed to be static, i.e., it is created once from a collection of points, but never updated afterwards. Performance degradation happens because insertions of point leads to an unbalanced k-d tree (worst case), thus a query on the data structure is processed with worse performance.

To illustrate performance degradation, imagine that a large number of points is sequentially inserted into a k-d tree. For each point, we must split a leaf node, which adds another layer to the tree. Now, imagine that for each point, the split always happens at the most left leaf. This is the worst case, since the k-d tree now has a relativley long branch. Especially when inserting many points, long branches occur more frequently, which means that many operations performed on the k-d tree likely need to visit more nodes. If, on the other hand, the tree was balanced, a maximium number of node traversals is given by the depth of the k-d tree.

Established adaptions of the k-d tree include the hB-tree [34] and the kdb-tree [44]. Procopiuc et al. [43] introduce the Bkd-tree as another extension of the popular k-d tree. While previous methods keep a single k-d tree balanced, a Bkd-tree uses the logarithmic method to maintain multiple k-d trees and rebuild them at regular intervals as more points are inserted into the data structure. This leads to "[...] high space utilization and excellent query and update performance regardless of the number of updates performed on it." [43] For the purposes of this thesis, a Bkd-tree will always start with no points in it. Instead of loading a collection of points directly as with k-d trees, points are then added step-by-step. Nevertheless, query performance degredation does not happen due to the internal structure of the k-d forest.

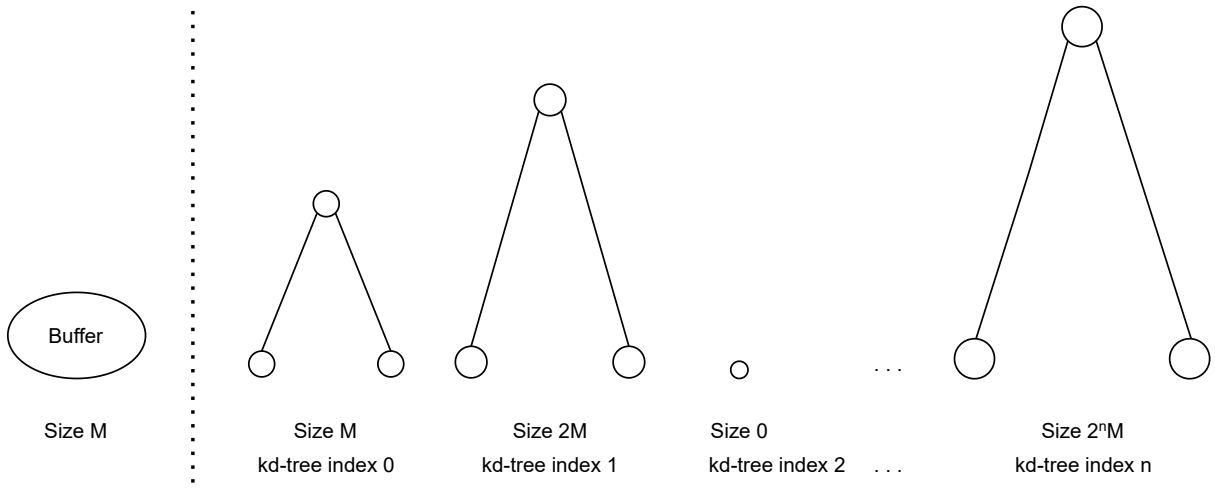


Figure 3.3: Internal structure of a Bkd-tree. Overall, $n+1$ k-d trees and an additional buffer are stored. When the buffer is full, the trees get reorganized using the logarithmic method. Inspired by Procopiuc et al. [43].

Figure 3.3 shows the internal structure of a Bkd-tree, i.e., its internal k-d forest. It consists of multiple k-d trees, each one having a size which is a multiple of a power of two, multiplied by M, where M denotes a predefined memory buffer size. In other words, the k-d forest starts with the smallest tree of size M. Subsequent trees then double in size. Therefore, there are at most $\log_2(\frac{N}{M})$ k-d trees, where N denotes the total number of points inside the data structure. In this thesis, a Bkd-tree is initially empty. Procopiuc et al. [43] introduce two methods to bulk load a Bkd-tree from a given set of points. Note that in this particular implementation, a Bkd-tree can be constructed from N points by creating a forest of $\lceil \log_2(\frac{N}{B}) \rceil$ empty k-d trees, which costs $O(1)$, and then put all N points in an additional k-d tree, which costs $O(N \log(N))$. However, for the purposes of the proposed algorithm, points are only added sequentially from an initially empty Bkd-tree.

3.3.2 Dynamic Updates

At first, it might seem unintuitive how the previously mentioned k-d forest prevents performance degradation. To exemplify this, consider the insertion of a point into the k-d forest, which works as follows: First, insert the point into the buffer. If the buffer is full, iterate the k-d trees (beginning with the smallest one) until you find an empty k-d tree at index i. Then, extract all the points from the previous, non-empty k-d trees and also the points from the buffer, and construct a new k-d tree with all collected points at index i. Therefore, all previous k-d trees which have an index smaller than i are now empty, so the next time the buffer becomes full, the reorganization of the k-d trees naturally is less expensive. The amortized cost of an insertion into the Bkd-tree is $O(B^{-1} \log_{\frac{M}{B}}(\frac{N}{B}) \log_2(\frac{N}{M}))$ (see [43] for details).

To delete a point from the Bkd-tree, simply do the following: Query every internal k-d tree with the point $\boldsymbol{p_k}$, which is to be removed. If $\boldsymbol{p_k}$ is found somewhere in the k-d forest, remove it.

The implementation in this thesis does that by swapping the element, which is to be removed, with the last element in the current leaf. Then, leaf size is decremented by one, or, the leaf is removed completly if no points remain. Since $\log_2(\frac{N}{M})$ is the maximum number of non-empty trees, and $\boldsymbol{p_k}$ is found after $\log_B(\frac{N}{B})$ traversals, the cost of removing a point from a Bkd-tree is $O(\log_B(\frac{N}{B})\log_2(\frac{N}{M}))$.

3.3.3 Queries

A Bkd-tree performs any other query that is likely to be executed on a classical k-d tree, e.g., a K nearest neighbor (KNN) search, by querying all internal non-empty trees, collecting the resulting points, and filtering them if needed. Therefore, the time complexity is nearly the same as with classical k-d trees but with more computational overhead, since a query searches multiple trees and the buffer, instead of just one tree. For example, the KNN query for a point $\boldsymbol{p_k}$ is an $O(K\log(N))$ operation for a k-d tree on average [21]. Querying a maximum of $\log_2(\frac{N}{M})$ non-empty k-d trees with $\boldsymbol{p_k}$ leads to an overall cost for the KNN query of $O(K\log(N)\log_2(\frac{N}{M}))$.

Chapter 4

Registration Procedure

The presented algorithm is especially designed to work with mobile scanning platforms in human-made environments with different motion- and scanning- profiles, where only error-prone IMU data is available. Thus, we make the following assumptions:

- Pose data is heavily influenced by noise, drift, and accumulation of errors
- Point cloud data comes in packages of chronologically ordered scans
- Time differences between scans stay sufficiently constant
- In a sufficiently small number of subsequent scans IMU related errors are insignificant
- Point density and scan size might vary in subsequent scans
- Planar areas, e.g., walls, ceiling, floor, etc. are abundantly available in the environment

Figure 4.1 gives a rough outline of the presented algorithm.

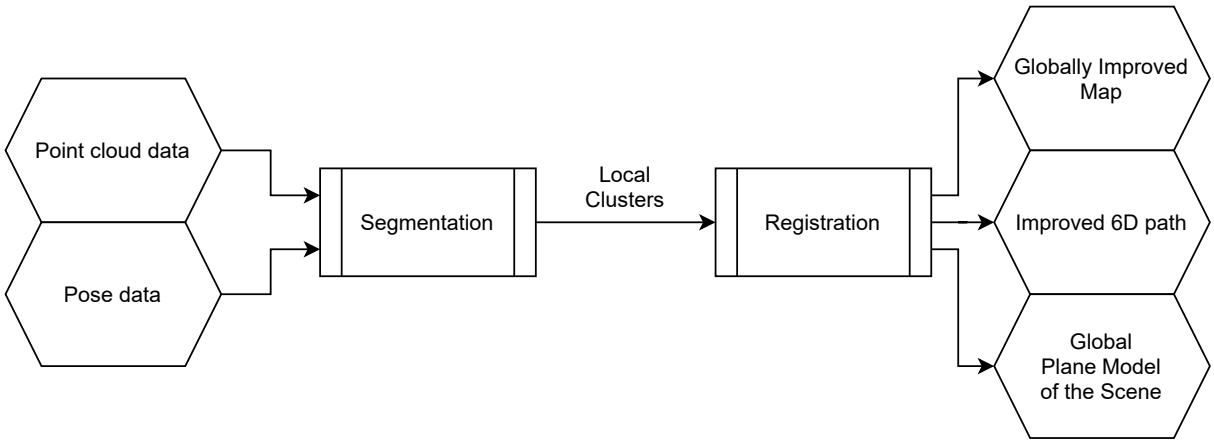


Figure 4.1: Outline of the presented algorithm. Point cloud and pose data are needed for segmentation. The resulting local planar clusters are used for registration.

4.1 Working Pipeline

We subdivide the presented algorithm into three main parts: preprocessing steps, local planar segmentation (clustering) of the point cloud, and global optimization. Figure 4.2 outlines preprocessing and clustering. Note that preprocessing is an optional step that works as follows: First, the point cloud data, which consists of multiple scans, condense, i.e., a predefined number of single scans are put together into one large scan, called a linescan. The single scans, therefore, transform into the coordinate system of a reference scan, which is the median of the single scans. Voxel-based reduction of the linescans ensures that point density is evened out when necessary and decreases computational cost. After preprocessing there are two options when calculating normal vectors for each point in the linescans: SRI based, which is faster, and KNN based, which is more accurate. Section 4.3 describe these methods in detail. Finally, the clustering algorithm segments each linescan using the previously calculated normals, into local planar regions.

Figure 4.3 outlines the third part, i.e., global optimization. The global plane model initializes with the local plane model of the first linescan, i.e., the linescan with index $j = 0$. Then, each subsequent linescan applies the previous pose optimization to itself before finding point-to-plane correspondences. A matching algorithm finds correspondences between points from the local model of the current linescan index j and the global model. Using these correspondences, an optimizer minimizes point-to-plane distances, resulting in an improved pose estimate for index j. The matching algorithm finds correspondences again, using the optimized pose. These correspondences are now used to update the global plane model as follows: Merge the local model with the global model where correspondences arise and extend the global model where correspondences did not arise from the local model.

In the next sections, we introduce each process of Figure 4.2 and 4.3 in detail. Further, these sections clarify the direct interactions with foregoing and following processes, which eventually explains how the proposed algorithm works as a whole.

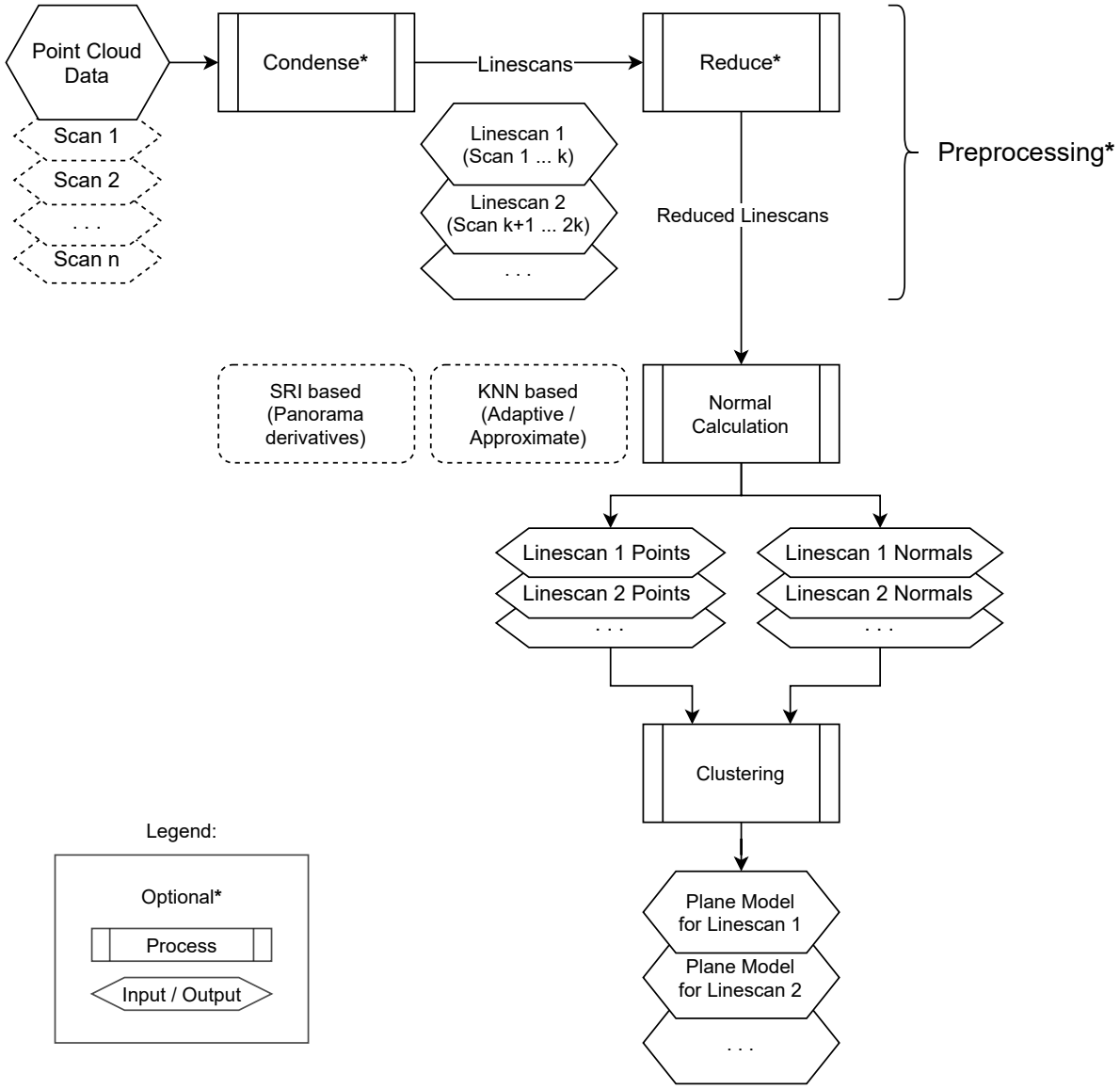


Figure 4.2: Outline of the preprocessing and local planar segmentation (clustering) pipeline of the presented algorithm. Inputs/Ouputs are stacked on top of each other, indicating that the pipeline is parallelizable.

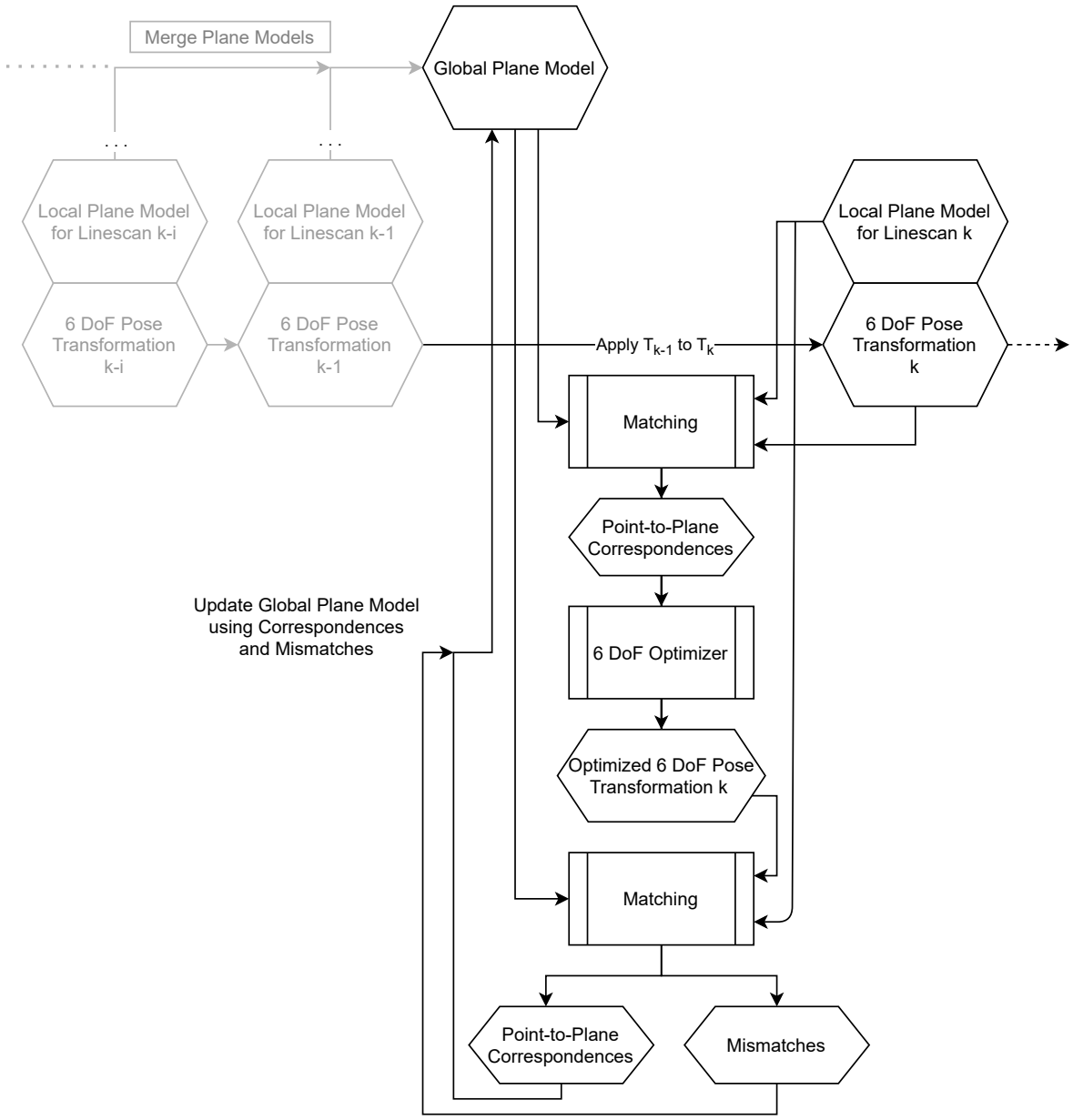


Figure 4.3: Outline of the global optimization pipeline of the presented algorithm. The greyed out part in the top left corner represents the same pipeline, but simplified. This indicates that this pipeline is sequentially applied to all subsequent linescans, and thus is not parallelizable.

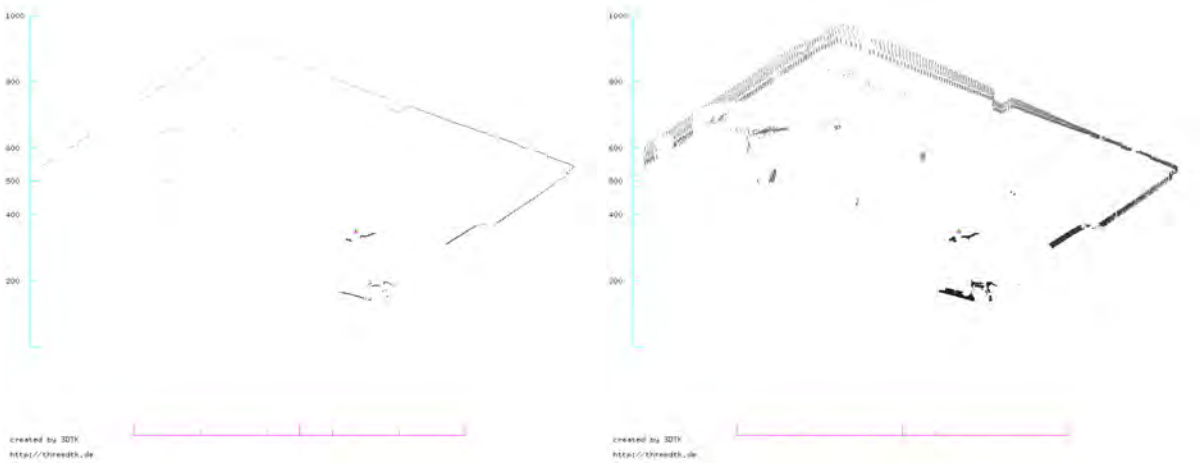


Figure 4.4: Example of condensing a point cloud. (Left) A scan from a 2D line scanner. (Right) Twenty scans from a 2D line scanner captured in 1 s, making up a 3D scan.

4.2 Preprocessing

Note that condensing and reduction is optional, as these steps are not always necessary. However, in many cases, they improve the quality of resulting normals. The next section introduces the preprocessing steps in detail. Section 4.2.1 introduces condensing of point clouds, which is used later to increase both point density and the number of locally available polygons in the environment. Then, Section 4.2.2 explains reduction of point clouds. It is used later as a powerfull tool for noise supression, but also for data compression.

4.2.1 Condensing

Condensing a point cloud refers to the operation where S scans are put together in one reference coordinate system, without actually altering their relative pose to the other scans. In this thesis, the reference frame is always the coordinate system of the median scan. Condensing a point cloud is especially useful for certain scanning patterns, e.g., a 2D line scanner (which creates 3D point clouds when rotated). Figure 4.4 illustrates this with the aforementioned example of a 2D line scanner with $S = 20$. With the assumptions stated in Chapter 4, it is obvious that S must not be too large, because IMU-based errors then distort the linescan. As condensing increases local point density, the calculation of point normals becomes more accurate and less sensitive with outliers if S is chosen well. Currently, S must be set manually. However, the parameter is easy to deduce with the naked eye. When a sensible S is chosen, condensing makes normal calculation (cf. Section 4.3) easier due to increased point density and a potentially higher field of view (FoV).

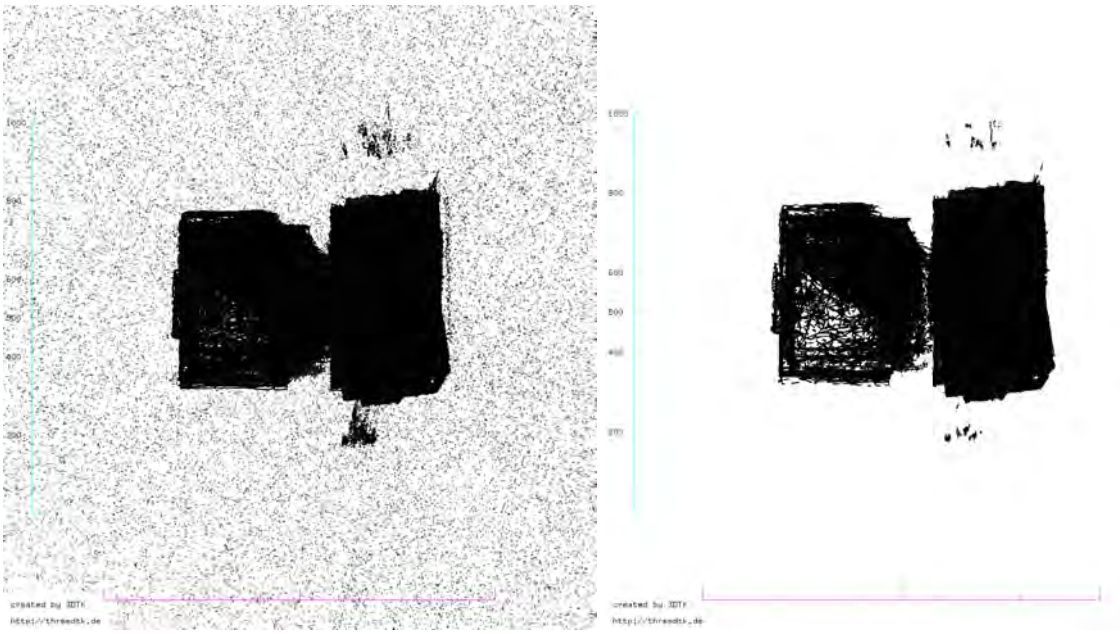


Figure 4.5: Example of reducing a point cloud. (Left) Point cloud data from a 3D scanner, rolled around in a spherical shell. Many reflections off the shell are visible, resulting in outliers. (Right) Reduced point cloud with voxel based octree reduction. Any 10 cm voxel with less than 50 points is deleted.

4.2.2 Reduction

When reducing a scan, points are removed from the point cloud in such a way that the remaining points still correspond to the overall structure of the environment. In this thesis, the reduction is done by distributing the points into voxels of predefined size V using an octree, as in [40]. There are now multiple options for defining constraints for the voxels, e.g., a maximum number of points that shall be inside that voxel. Or, delete any voxel that has more points than another number of points. Many other options exist, making reduction a powerful tool for outlier removal and noise suppression. Figure 4.5 shows an example where any voxel of size $V = 10\,\text{cm}$ that includes less than 50 points is deleted. As reducing the point cloud decreases point density, all succeeding components of the algorithm, i.e., normal calculation, clustering, matching, and optimization, benefit from the decrease in computational cost.

4.3 Normals Calculation

The presented clustering algorithm in Section 4.4 uses normal vectors (normals), as they represent the local planarity of a point cloud well. This means that points that are on the same plane have normals pointing in the same direction. Normals are calculated in a relatively small neighborhood of a point. To calculate a normal vector for a point in the scan, the points K-nearest neighbors (KNN) are required. We find the KNNs efficiently using a k-d tree, described in Section 3.3. Then, a normal vector is calculated by fitting a plane through the KNNs, us-

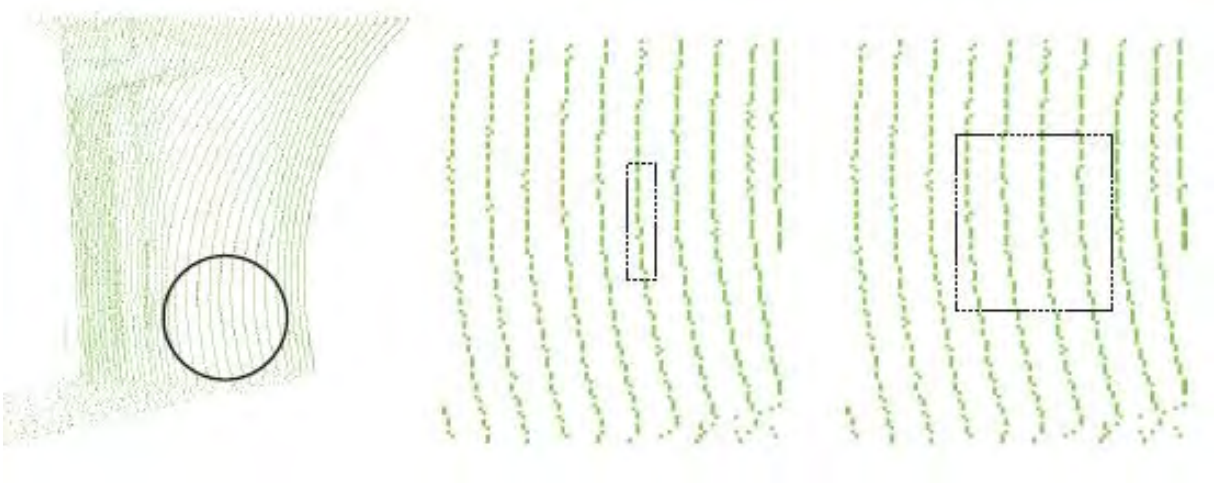


Figure 4.6: Illustration of KNNs for a critical case, where point density is low. (Left) A scene from a 3D point cloud. (Center) Bounding box of the resulting KNNs with low value of K. (Right) Bounding box of resulting KNNs with high value of K. Source: [41]

ing principal component analysis (PCA) or other plane fitting methods like least squares. The normal vector of the point corresponds to the normal vector of the plane fit. We apply this method for every point to obtain the normals for the whole point cloud. Especially for mobile mapping systems, datasets become noisier and less dense with increasing scanning range. Figure 4.6 illustrates the problem with low point density: The resulting KNNs do not represent the planarity of the scene. This indicates that, for those areas of a scan, a higher value of K must be chosen. Typically, this is done by adaptively increasing K until the bounding box around the KNNs fulfills a certain criterion, which is that the smallest eigenvalue of the PCA must be very small compared to the other two, and the other two eigenvalues must be of the same order. Figure 4.7 shows that the resulting normal vectors better resemble the corresponding plane when using K-adaption. The adaptive KNN method is by far the most accurate normal calculation method. However, it is also the most computationally expensive one. The next section introduces approximate methods, which make KNN calculation less expensive.

4.3.1 Approximate Methods

Often, 3D point clouds are large and densely populated with points. In those cases it is a good strategy to choose a large value of K, because it results in averaging a normal vector around the neighborhood of the scan, thus representing a planar feature. Large values for K, however, result in a longer runtime. This runtime is sought to be decreased by making a tradeoff with the accuracy of the algorithm. To do this, the traversal of the k-d tree for obtaining KNNs is modified. The most straightforward way to speed this up is to just stop the recursive backtracking process once K neighbors have been found, even if those are not necessarily the nearest ones. Or, you set a threshold for how many points you consider before abort. However, a disadvantage comes with that, which is the inefficiency of backtracking. The order of traversal is according to the tree structure, thus does not consider the distance to the query point itself.

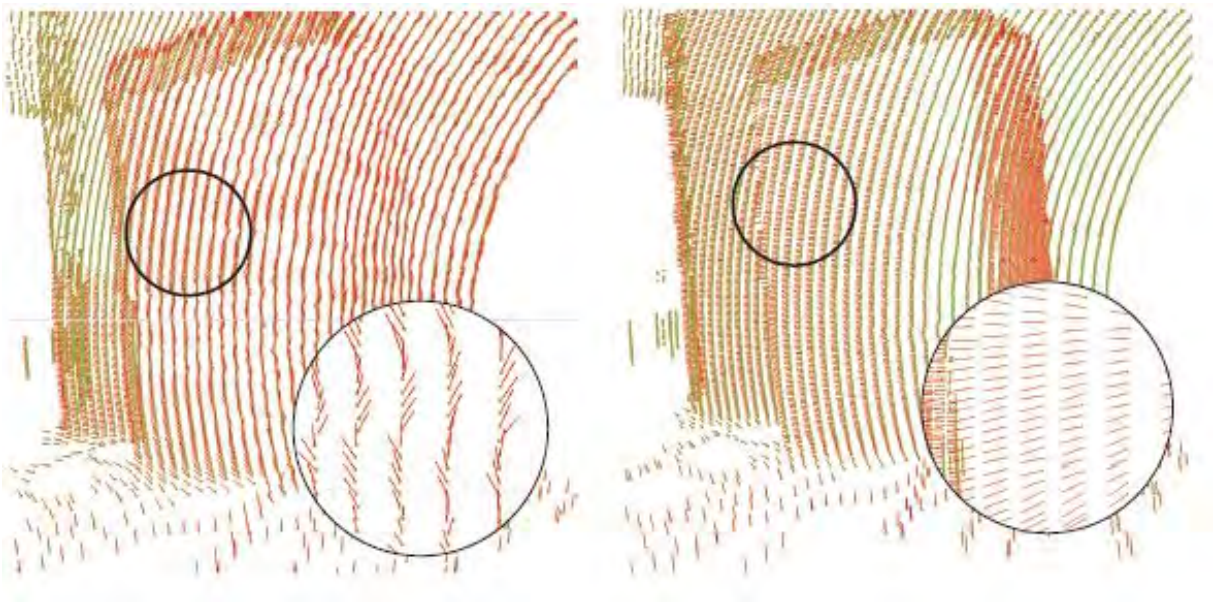


Figure 4.7: Results of K-adaption for normal calculation. (Left:) Resulting normal vectors in a low density scene without K-adaption. Normals point in many different directions, despite beeing on the same plane. (Right:) Resulting normal vectors in the same scene with K-adaption. All normals point in the same, correct direction. Source: [41]

Beis & Lowe [2] propose the best bin first (BBF) approach, where points are searched in bins, ordered increasingly according to the distance from the query point. When searching these ordered bins, the first K points that the algorithm examines are the K-nearest neighbors with high probability. The BBF search, therefore, stops after finding K points. The distance to the bin is defined as the minimum distance between the query point and any point on the bin boundary. Furthermore, they employ a priority queue, where an entry is stored for any branch option not taken. After reaching a leaf node, the top of the priority queue is removed. The removed element contains the branch where the search for the next closest bin is continued. Approximate KNN methods might be a fast way for normal calculation, but there is an even faster way, which is described in the next section.

4.3.2 Panorama Images

The previous two sections introduced normal calculation methods that utilize k-d trees for KNN estimation. However, the usage of panorama images yields another fast method for normal calculation. First, the 3D point cloud obtained by the laser scanner is converted into a spherical range image (SRI), i.e., a panorama image. The equirectangular projection, or any other 3D to 2D projection of your choice, maps the point cloud to a panorama image. Figure 4.8 shows such a SRI. In these images, the local neighborhood of a point corresponds to the neighboring pixels of the panorama. Since the SRI has to be created only once for a scan, this method estimates the nearest neighbors of a point in constant O(1) time. From here, the KNN based methods described above calculate normals for every point using least-squares methods. Note that the

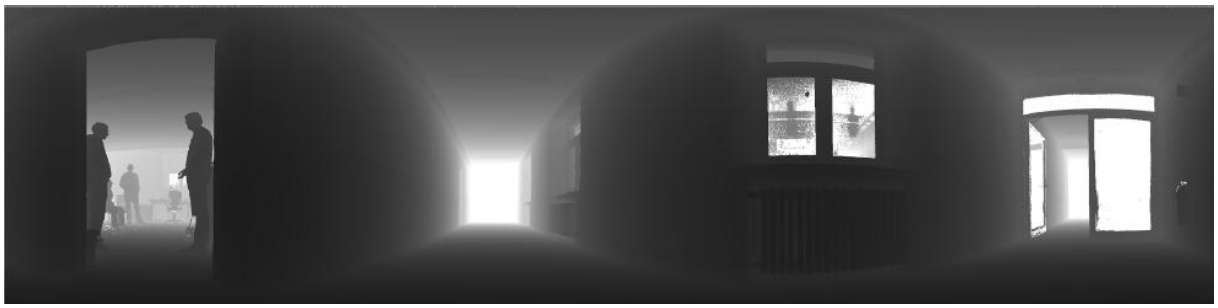


Figure 4.8: Spherical range image (SRI) obtained from a 3D point cloud. Source: [41]

number of neighbors varies for each point, depending on point density (which, again, depends on range). Badino et al. [1] introduce another method, based on SRI derivatives, to eliminate the computationally rather expensive least-squares-part, which includes solving a third-degree polynomial for every point. Figure 4.9 compares the resulting normals generated with the least-squares method with the resulting normals generated with the SRI derivative method. The SRI derivative method is, without doubt, the fastest of normal calculation methods. However, on real-world datasets, the estimated normal vectors are less accurate on the edges, when compared to least-squares methods. For the clustering algorithm, which is described in the next section, it does not matter whether the normals originated from a panorama-based or k-d tree based method.

4.4 Clustering Algorithm

The previous sections described how to obtain normal vectors for each point in the point cloud. The purpose of the clustering algorithm is to encapsulate points in a local neighborhood that have similar normals, for each linescan. Note that a points normal vector $\boldsymbol{n}$ corresponds to a plane representation, thus $\boldsymbol{n}$ and $-\boldsymbol{n}$, which make up an angle of π, actually correspond to the same plane, i.e., the angle between them should be zero. The angle between two normalized vectors is

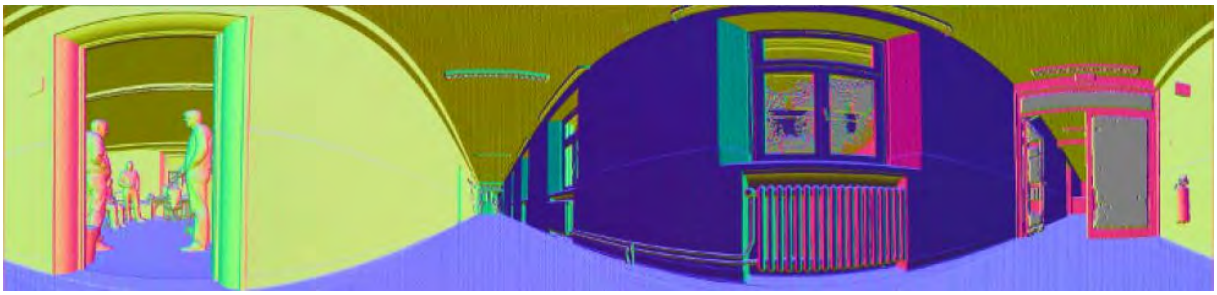
$$\alpha(\boldsymbol{n}_1, \boldsymbol{n}_2) = \arccos(\boldsymbol{n}_1 \cdot \boldsymbol{n}_2) \; . \tag{4.1}$$

However, the *smallest* angle between two normalized normal vectors is

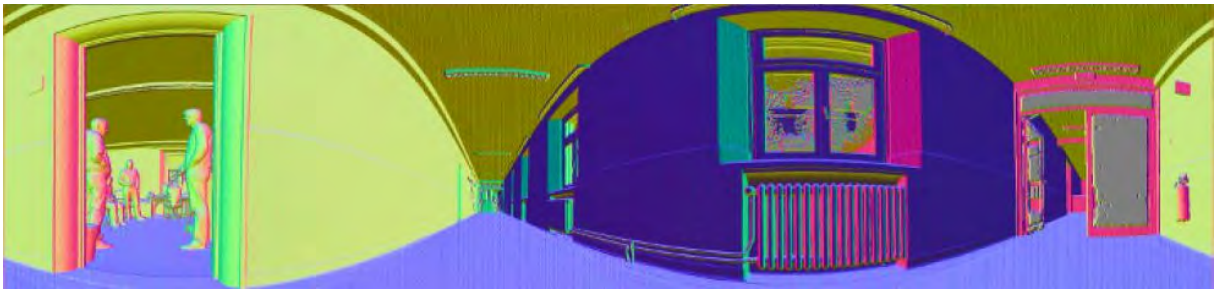
$$\hat{\alpha}(\boldsymbol{n}_1, \boldsymbol{n}_2) = \begin{cases} 2\pi - \alpha(\boldsymbol{n}_1, \boldsymbol{n}_2) & \text{if } \alpha(\boldsymbol{n}_1, \boldsymbol{n}_2) > \frac{3}{2}\pi \\ \alpha(\boldsymbol{n}_1, \boldsymbol{n}_2) - \pi & \text{if } \alpha(\boldsymbol{n}_1, \boldsymbol{n}_2) > \pi \\ \pi - \alpha(\boldsymbol{n}_1, \boldsymbol{n}_2) & \text{if } \alpha(\boldsymbol{n}_1, \boldsymbol{n}_2) > \frac{\pi}{2} \end{cases} , \tag{4.2}$$

because the opposing normals $-\boldsymbol{n}_1$ and $-\boldsymbol{n}_2$ have to be considered since they correspond to the same plane. We say that two normal vectors are similar if the angle $\hat{\alpha}$ from Equation (4.2) is smaller than a threshold ε_α.

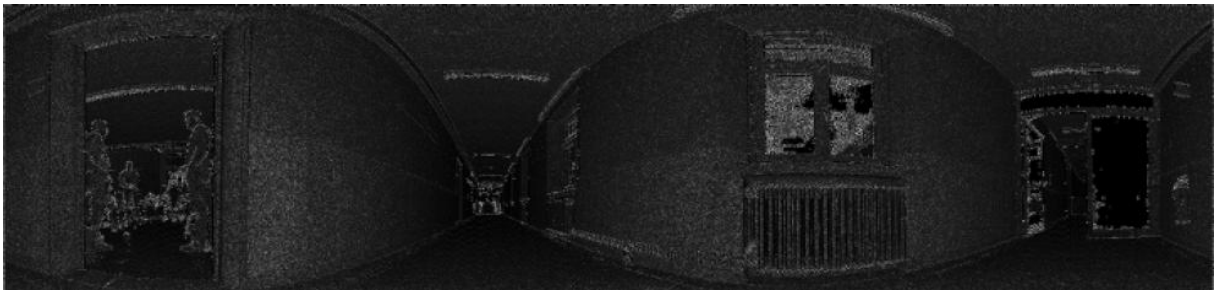
Using this similarity criterion, the algorithm is able to segment the point cloud with a region growing approach. The full algorithm is outlined in Figure 4.10. It consists of two parts: region growing and filter. The next subsection deals in detail with the implementation of the region-growing, which is one of the major scientific contributions of this work.



(a) Resulting normals using the SRI derivative method.



(b) Resulting normals using the least-squares method.



(c) Difference image between (a) and (b).

Figure 4.9: Comparison between least-squares and derivative method for normal calculation on spherical range images (SRIs). They differ especially at the edges of the scene. Source for all three images: [41]

4.4.1 Region Growing with Trees

The region growing implemented in the presented algorithm uses a novel technique that utilizes a dynamically scalable tree structure, i.e., the Bkd-tree (see Section 3.3), as well as a modified version of a static k-d tree, for optimal segmentation runtime. The strategy of the algorithm is to grow only one cluster at a time until it is finished, i.e., the cluster does no longer grow. Then, another random point from the point cloud is selected, and region growing starts again until each point belongs to a cluster.

The modified version of the static k-d tree supports deletion of points, but not insertion. It performs a K-nearest neighbor (KNN) search around a given point $\boldsymbol{p_k}$ (which is the point that currently grows). Once a point has been grown this way, the algorithm removes it from the modified k-d tree. All points that have been found with the KNN search operation are

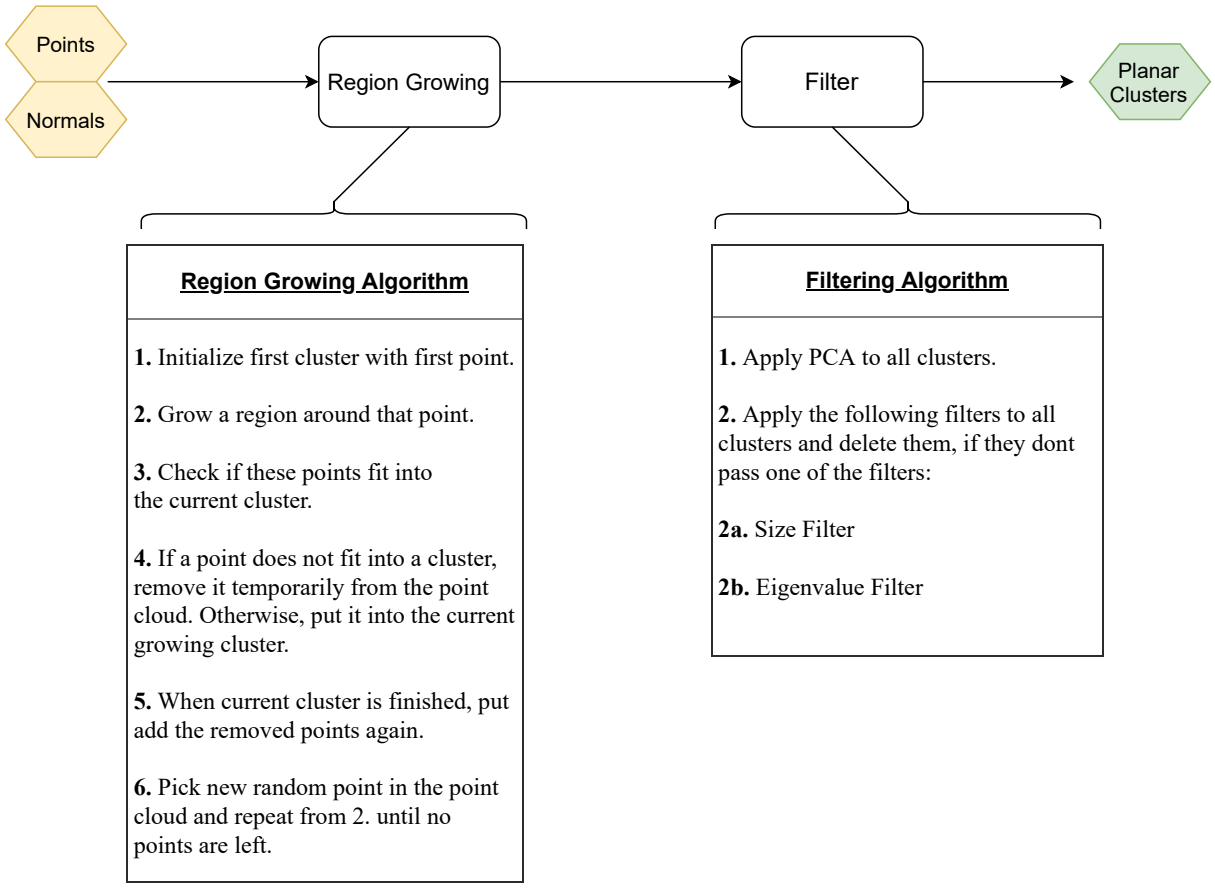


Figure 4.10: Outline of the local clustering algorithm. Inputs are points and corresponding point normals. When the algorithm is finished, each point has a label assigned to itself, representing the local planar cluster it belongs to.

then put into a queue. The queue represents the iteration order of points that the algorithm considers next for region growing. Points that are already in the queue get temporarily marked as long as they are inside the queue, hence the algorithm does not visit a point twice. Now, the algorithm decides if $\boldsymbol{p_k}$ should belong to the cluster C, which is currently growing. If $\boldsymbol{p_k}$ does not belong to C, it is marked temporarily as a visited point that has not yet been assigned to any cluster. Once C is not able to grow anymore, these marks are reset. The point $\boldsymbol{p_k}$ belongs to C if they have similar normals, and $\boldsymbol{p_k}$ has a distance to C which is smaller than a predefined growth radius threshold d_{growth}. The growth radius d_{growth} is adapted quadratically depending on the distance from $\boldsymbol{p_k}$ to the origin of the local coordinate system, i.e., its distance to the laser scanner. This means that, for points with a small distance to the sensor (where point density usually is high), the growth radius is also small. However, when points have a larger distance to the sensor (where point density usually is low), the growth radius must naturally be larger, too. Figure 4.11 shows the result of growth radius adaption on an indoor, real-world dataset.

Any region growing based segmentation method must inevitably face the following problem:

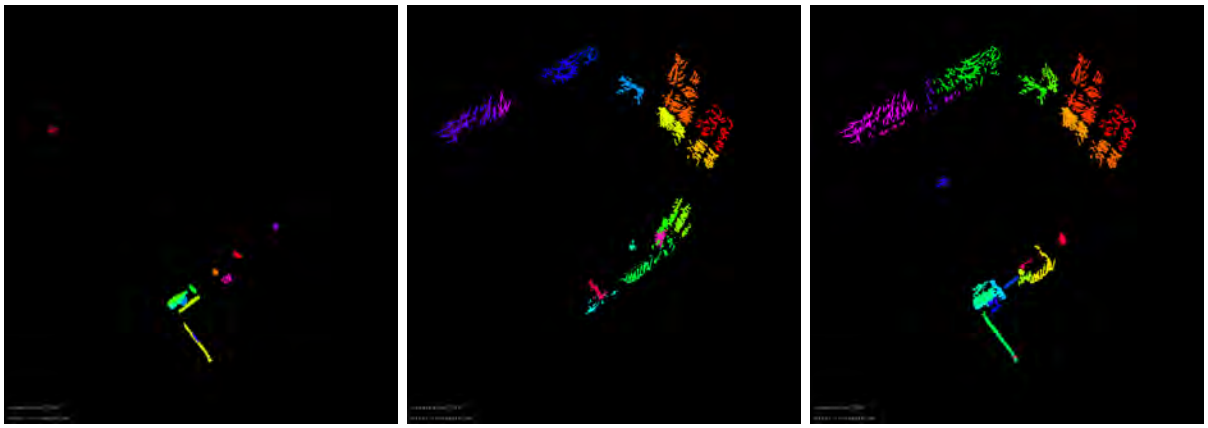


Figure 4.11: Output of the clustering algorithm, using different grow radii d_{growth}. A scan from a real world dataset (see Section 5.2.2) gets clustered. (Left) Using $d_{growth} = 5\,\text{cm}$. Only point with a small distance to the sensor get clustered. (Center) Using $d_{growth} = 50\,\text{cm}$. Only point with a high distance to the sensor get clustered. (Right) Using linear growth adaptation with $5\,\text{cm} \leq d_{growth} \leq 50\,\text{cm}$. Points with any distance to the sensor get clustered.

For a given point $\boldsymbol{p_k}$ from the point cloud, what is its distance to the cluster C? In other words, which point from C is $\boldsymbol{p_k}$'s the nearest neighbor (NN)? When we don't order the points inside C in a tree structure, a naive solution is to calculate the distance from $\boldsymbol{p_k}$ to any point $\boldsymbol{q} \in C$ with brute force (BF). However, this leads to unacceptable segmentation runtime for large point clouds, as the distance calculation is an $O(N)$ operation, where N is the number of points in the cluster. Therefore, the presented algorithm uses a novel strategy for this problem: We store the points inside C using a Bkd-tree, as it supports insertion of points, as well as NN searching with logarithmic amortized cost (see Section 3.3). This admittedly raises the time complexity of insertion from constant $O(1)$ to $O(B^{-1} \log_{\frac{M}{B}}(\frac{N}{B}) \log_2(\frac{N}{M}))$, where B is the bucketsize of the trees, and M is a memory buffer size. However, it decreases the time complexity of the NN query from linear $O(N)$ to $O(\log(N) \log_2(\frac{N}{M}))$. Overall, this yields a faster runtime when compared to the BF approach. Figure 5.1 also shows this empirically. We filter the result of the region growing in the next section, as we seek only planar clusters.

4.4.2 Filter

Applying the region growing algorithm from the previous section to a point cloud with n points consequently results in n points being labeled. However, this unavoidably means that clusters get constructed also in non-planar or sharp parts of the point cloud, where local normal direction differences are large. Planar areas like walls will therefore always be connected in one large cluster, while non-feature parts, i.e., non-planar or detached parts of the scan will have their own, smaller cluster. The histogram in Figure 4.12 illustrates this. It shows the number of points in the clusters, sorted in descending order, found for an artificially generated, fully noise-free dataset, also shown in Figure 4.12. Note that even if the dataset is free from pose- or range-noise, the algorithm finds many small clusters at the edges of the room, simply due to the nature of point normals in that area. Hence putting a threshold n^c_{min} on the minimum cluster

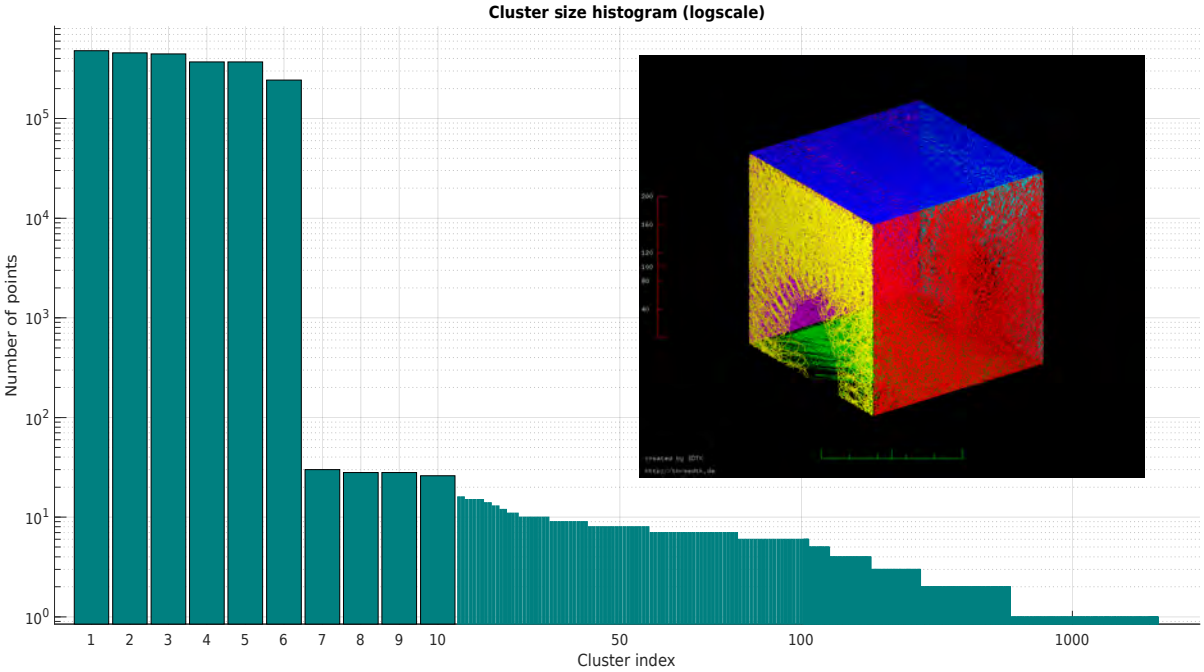


Figure 4.12: Histogram showing cluster size after region growing (before filtering) in an artificially generated dataset with six perfect walls. The colors in the dataset correspond to the biggest six clusters. The other clusters correspond to the sharp edge regions.

size successfully filters edge regions. The filtering operation consists not only of a minimum size threshold but also an additional eigenvalue ratio filter. It puts a maximum threshold ε_e on the ratio between the smallest eigenvalue and the sum of all eigenvalues, where the eigenvalues are a result of a principal component analysis (PCA) that fits a plane to the points in the cluster. The geometric interpretation of this is that the extent in normal direction must be relatively small compared to the extent in the other directions. Figure 4.12 also shows the result after applying both thresholds. The filter successfully removes all the small, non-planar clusters. In the next section, we want to find matches between those filtered clusters in subsequent scans.

4.5 Finding Matches

The previous two sections addressed the creation of local planar clusters (LPCs), using region growing and filtering. Now, the goal of the matching algorithm is to find a corresponding plane in the global model for each point in a local planar cluster (LPC). Hence, we compare each LPC with each other global cluster to find similarities. Five properties have been selected to check if two clusters are similar, which are described in detail below. Note that the global model is sequentially updated for each scan, using the LPCs as indicated in Figure 4.3. Therefore, the global planes share all properties that the LPCs have, like eigenvalues or convex hulls. Figure

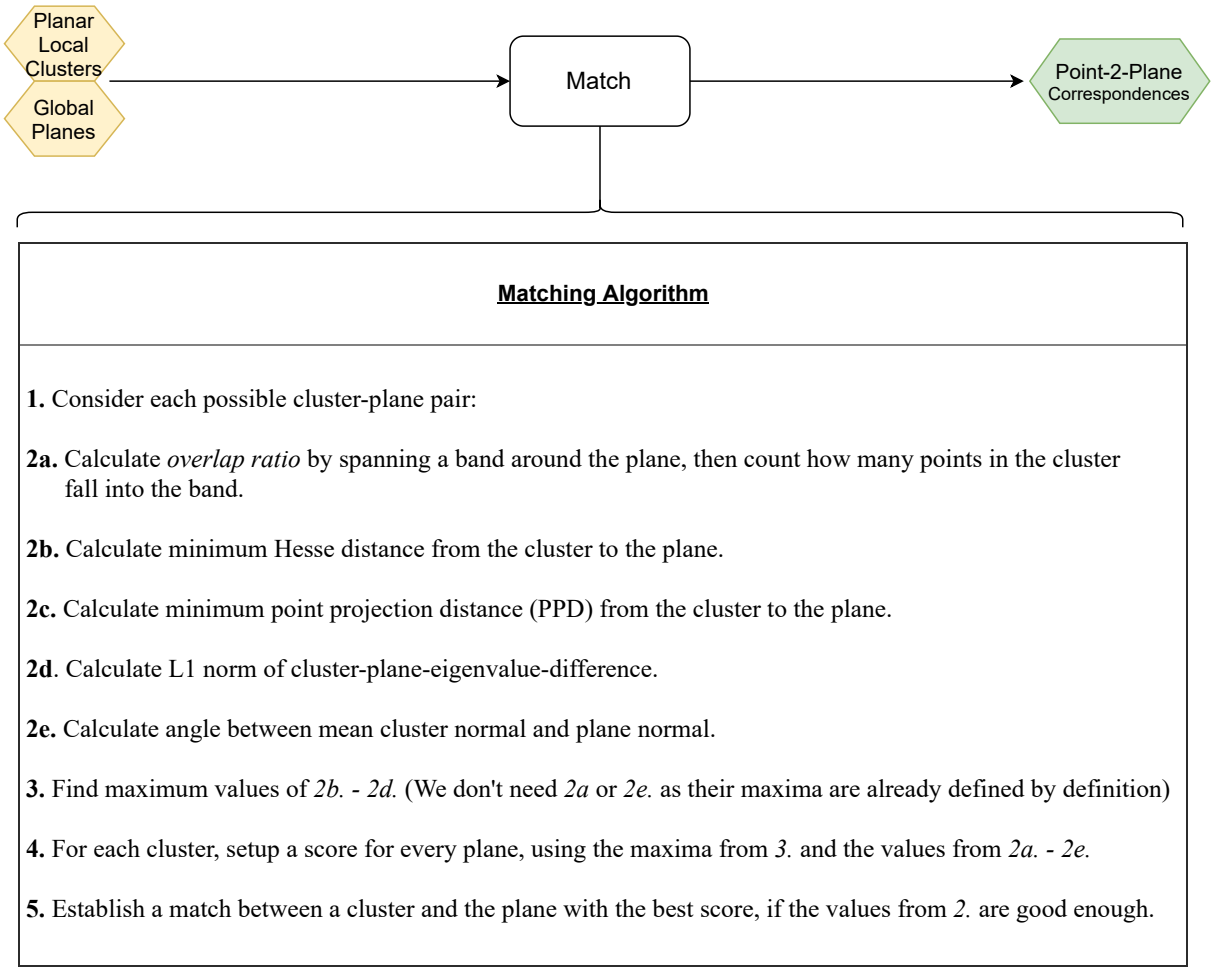


Figure 4.13: Outline of the matching algorithm. Inputs are the local planar clusters (LPC) and the planes from the global model. The output is point-to-plane correspondences.

4.13 outlines the matching algorithm. First, we set up a score function to calculate the likeliness of a cluster to belong to a certain plane. However, such a cluster-plane score (CPS) must not be interpreted as a probability, as its value only matters when being compared to other CPS's. What is true, though, is that a higher score corresponds to a higher similarity, which is sufficient for our purpose.

To calculate the CPS of a local cluster C and global plane P, the matching algorithm first calculates and stores the following properties for any possible pair of C and P:

- The angle between normals $\alpha^{C,P}$ according to Equation (4.2) between C and P.
- The ratio $r^{C,P}$ of all $\boldsymbol{p_k} \in C$ that fall inside a 3D bounding box around P.
- The minimum hesse distance $D^{C,P}_{\mathrm{H,min}}$ from any point $\boldsymbol{p_k} \in C$ to P.
- The minimum PPD $D^{C,P}_{\mathrm{PPD,min}}$ from any point $\boldsymbol{p_k} \in C$ to P.

- The L1 norm of eigenvalue differences $\Delta e^{C,P}$ between the eigenvalues of C and P.

Secondly, the matching algorithm finds the maximum possible values for these quantities. Note that this is not necessary for the angle between C and P, as the maximum angle two planes potentially span is $\frac{\pi}{2}$. It is also not necessary for the ratio of overlap, as the maximum overlapping portion assumably is 100%. We denote the maximum of the remaining quantities as follows:

$$D_{\mathrm{H,max}} = \max\left(D^{C,P}_{\mathrm{H,min}}\right) \quad \forall\ (C,P)\ , \tag{4.3}$$

$$D_{\mathrm{PPD,max}} = \max\left(D^{C,P}_{\mathrm{PPD,min}}\right) \quad \forall\ (C,P)\ , \tag{4.4}$$

$$\Delta e_{\max} = \max\left(\Delta e^{C,P}\right) \quad \forall\ (C,P)\ . \tag{4.5}$$

The next step of the matching algorithm is to calculate a score s for each attribute of a cluster-plane pair (C,P) individually, using the following score functions:

$$s_\alpha(C,P) = 1 - \frac{2\alpha^{C,P}}{\pi}\ , \tag{4.6}$$

$$s_r(C,P) = r^{C,P}\ , \tag{4.7}$$

$$s_{\mathrm{H}}(C,P) = 1 - \frac{D^{C,P}_{\mathrm{H,min}}}{D_{\mathrm{H,max}}}\ , \tag{4.8}$$

$$s_{\mathrm{PPD}}(C,P) = 1 - \frac{D^{C,P}_{\mathrm{PPD,min}}}{D_{\mathrm{PPD,max}}}\ , \tag{4.9}$$

$$s_{\Delta e}(C,P) = 1 - \frac{\Delta e^{C,P}}{\Delta e_{\max}}\ . \tag{4.10}$$

Finally, the CPS incorporates these score functions by computing their weighted sum:

$$\begin{aligned} s(C,P) = {} & w_\alpha \cdot s_\alpha(C,P) \\ & + w_r \cdot s_r(C,P) \\ & + w_{\mathrm{H}} \cdot s_{\mathrm{H}}(C,P) \\ & + w_{\mathrm{PPD}} \cdot s_{\mathrm{PPD}}(C,P) \\ & + w_{\Delta e} \cdot s_{\Delta e}(C,P)\ . \end{aligned} \tag{4.11}$$

The weights correspond to the importance of an attribute for matching. For example, the angle between two planes should be more important than their L1 norm eigenvalue difference. The latter should be the least important factor, as it is only meaningful in a few special edge cases. The authors recommendation on all weights are $w_\alpha = 1.0$, $w_r = 1.0$, $w_{\mathrm{H}} = 0.5$, $w_{\mathrm{PPD}} = 0.1$, and $w_{\Delta e} = 0.1$, indicating that the angle between a pair, as well as their overlap ratio, are most important for finding matches. They were found empirically and used for creating all presented results that this thesis shows.

To establish point-to-plane correspondences, the matching algorithm looks at each cluster C and computes the score from Equation (4.11) for each global plane P. The (C,P)-pair having the highest score is considered a match. Then, point to plane correspondences arise for each point

in C to P, if the following criteria are fulfilled: The minimum Hesse distance for a pair $D^{C,P}_{\mathrm{H,min}}$ is smaller than a threshold ε_{H}. And, the minimum PPD for a pair $D^{C,P}_{\mathrm{PPD,min}}$ is smaller than a threshold $\varepsilon_{\mathrm{PPD}}$. And, the angle between a pair $\alpha^{C,P}$ is smaller than a threshold ε_{α}. If any of these criteria fail, the algorithm discards the (C, P)-match, i.e., no point-to-plane correspondences are established. Once the matching algorithm established point-to-plane correspondences, we seek to minimize the total distance these points have to their respective planes, which we do in the next section.

4.6 Optimization

In the previous section, point-to-plane correspondences arised using a matching algorithm. Now, we want to minimize the distance that each point has to its corresponding plane by optimizing the pose of the laserscanner. Therefore, the optimization is the process of finding a 6 DoF pose transformation that minimizes the distance of all points to their respective planes. Section 3.2.1 and 3.2.2 provide two distance models: the Hesse distance and the polygon projection distance (PPD). The PPD is useful to establish point-to-plane correspondences, indeed. However, it is excluded from the optimization, as minimizing the Hesse distance is sufficient as long as there are three or more in the local plane model, i.e., three or more planes are present in an individual scan (which is ensured by condensing, see Section 4.2.1). Furthermore, minimizing the PPD is not always reasonable, since it constrains the growth of a plane in a senseless way.

On the other hand, imagine the Hesse distance minimization as attaching springs for all corresponding points to their respective planes orthogonally. Imagine the optimization as the springs retracting, and therefore pulling the points onto their corresponding planes. When there are springs attached for each point of only one plane, the optimization potentially fails to align the edges of a scene. When there are springs attached for two planes, the optimization potentially fails to align the corners of a scene. However, attaching those springs to each point of three planes is sufficient for aligning edges and corners. In the next section, we model the point-to-plane distances using an error function. Then, we use the derivatives of that function for gradient descent, which is described in Section 4.6.2.

4.6.1 Error function

We set up the error function as a sum of the squared Hesse distances over all point-to-plane correspondences

$$E = \sum_{\forall \mathcal{P}_i} \sum_{\boldsymbol{p}_k \in \mathcal{P}_i} \| \boldsymbol{n}_{\mathcal{P}_i} \cdot [T(\boldsymbol{p}_k) - \boldsymbol{a}_{\mathcal{P}_i}] \|^2 = \sum_{\forall \mathcal{P}_i} \sum_{\boldsymbol{p}_k \in \mathcal{P}_i} \| D_h^{i,k}(\boldsymbol{\Pi}, \boldsymbol{p}_k) \|^2 , \tag{4.12}$$

where $\boldsymbol{\Pi} = [\varphi, \vartheta, \psi, t_x, t_y, t_z]^\tau$. Its gradient follows then immediately through differentiation:

$$\nabla E = \sum_{\forall \mathcal{P}_i} \sum_{\boldsymbol{p}_k \in \mathcal{P}_i} \begin{bmatrix} \frac{\partial}{\partial \varphi} E & \frac{\partial}{\partial \vartheta} E & \frac{\partial}{\partial \psi} E & \frac{\partial}{\partial t_x} E & \frac{\partial}{\partial t_y} E & \frac{\partial}{\partial t_z} E \end{bmatrix}^\tau \tag{4.13}$$

$$\Rightarrow \nabla E = \sum_{\forall \mathcal{P}_i} \sum_{\boldsymbol{p}_k \in \mathcal{P}_i} 2 \cdot D_h^{i,k} \cdot \begin{bmatrix} \nabla E_\varphi & \nabla E_\vartheta & \nabla E_\psi & n^x_{\mathcal{P}_i} & n^y_{\mathcal{P}_i} & n^z_{\mathcal{P}_i} \end{bmatrix}^\tau \tag{4.14}$$

The derivates in Equation (4.14) are

$$\begin{aligned}\nabla E_\varphi = \; & n^y_{\mathcal{P}_i}(x_k[-S_\varphi S_\psi + C_\varphi C_\psi S_\vartheta] + y_k[-S_\varphi C_\psi - C_\varphi S_\vartheta S\psi] - z_k C_\varphi C_\vartheta) \\ & + n^z_{\mathcal{P}_i}(x_k[C_\varphi S_\psi + C_\varphi C_\psi S_\vartheta] + y_k[C_\varphi C_\psi - S_\varphi S_\vartheta S\psi] - z_k S_\varphi C_\vartheta)\end{aligned} \;, \tag{4.15}$$

$$\begin{aligned}\nabla E_\vartheta = \; & n^x_{\mathcal{P}_i}(-x_k S_\vartheta C_\psi + y_k S_\vartheta S_\psi + z_k C_\vartheta) \\ & + n^y_{\mathcal{P}_i}(x_k C_\psi S_\varphi C_\vartheta - y_k S_\varphi C_\vartheta S_\psi + z_k S_\vartheta S_\varphi) \\ & + n^z_{\mathcal{P}_i}(-x_k C_\varphi C_\psi C_\vartheta + y_k C_\varphi C_\vartheta S_\psi - z_k C_\varphi\end{aligned} \;, \tag{4.16}$$

and

$$\begin{aligned}\nabla E_\psi = \; & n^x_{\mathcal{P}_i}(-x_k C_\vartheta S_\psi - y_k C_\vartheta C_\psi) \\ & + n^y_{\mathcal{P}_i}(x_k[C_\varphi C_\psi - S_\varphi S_\vartheta S_\psi] + y_k[-C_\varphi S_\psi - S_\varphi S_\theta C_\psi]) \\ & + n^z_{\mathcal{P}_i}(x_k[S_\varphi C_\psi + C_\varphi S_\psi S_\theta] + y_k[-S_\psi S_\varphi + C_\varphi S_\vartheta C_\psi])\;.\end{aligned} \tag{4.17}$$

As the error gradient is well-defined we minimize the error function with a gradient descent based method.

4.6.2 Gradient Descent with AdaDelta

This section explains how the error function, which was introduced in the previous section, is minimized with gradient descent. The commonly used, well-known stochastic gradient descent (SDG) algorithm computes

$$\mathbf{\Pi}_{j+1} = \mathbf{\Pi}_j - \alpha \nabla E \tag{4.18}$$

where α is the learning rate. To accelerate convergence and to improve the found solution further modifications are made.

Since we have vastly different effects on the error function by each dimension, the first consideration for improving the SDG is the following: Typically, orientation changes, i.e., the first three elements of the gradient vector $\frac{\partial}{\partial\varphi}E$, $\frac{\partial}{\partial\vartheta}\boldsymbol{E}$, and $\frac{\partial}{\partial\psi}E$, have much more impact on the error function than a change in position. This is intuitively explained since translating the scan makes the error grow linearly for all points. However, when rotating the scan, points with a larger distance to the robot are moved drastically, leading to a higher sensibility on the error function. For this reason, the α applied on orientation has to be much smaller than the α applied on the position. It becomes evident that α needs to be extended into vector form, $\boldsymbol{\alpha}$, therefore weighting each dimension differently.

Another consideration to speed up SDG is to adaptively recalculate $\boldsymbol{\alpha}$ for each iteration. We employ and modify AdaDelta as a technique to do so, which is described in detail in [56]. The main idea is the following: It extends the SDG algorithm by two terms. First, an exponentially decaying average of past gradients $\boldsymbol{G}_j$ which is recursively defined as

$$\boldsymbol{G}_{j+1} = \zeta \boldsymbol{G}_j + (1-\zeta)\nabla E^2 \tag{4.19}$$

and second, an exponentially decaying average of past changes $\boldsymbol{X}_k$ which is defined as

$$\boldsymbol{X}_{j+1} = \zeta \boldsymbol{X}_j + (1-\zeta)\boldsymbol{\alpha}\nabla E^2 \tag{4.20}$$

where $\zeta \leq 1$ is a decay constant, typically close to 1. The root mean squared (RMS) of these quantities are

$$RMS[\boldsymbol{G}]_j = \sqrt{\boldsymbol{G}_j + \varepsilon} \tag{4.21}$$

and

$$RMS[\boldsymbol{X}]_j = \sqrt{\boldsymbol{X}_j + \varepsilon} \tag{4.22}$$

where $\varepsilon > 0$ is a very small constant, typically close to 0. It will prevent dividing by zero in the recalculation of $\boldsymbol{\alpha}$ which is as follows:

$$\boldsymbol{\alpha_j} = \frac{RMS[\boldsymbol{X}]_{j-1}}{RMS[G]_j} \tag{4.23}$$

For our particular application, AdaDelta behaves a little too aggressively. Despite giving a good measure on how to adapt $\boldsymbol{\alpha}$, the algorithm sometimes overshoots and does not converge. Therefore, we employ another scaling factor, typically not found in AdaDelta, extending Equation (4.23) to:

$$\boldsymbol{\alpha_j} = \boldsymbol{\alpha_0} \cdot \frac{RMS[\boldsymbol{X}]_{j-1}}{RMS[\boldsymbol{G}]_j} \tag{4.24}$$

where $\boldsymbol{\alpha_0}$ holds the scaling factors for each dimension. Finally, the SDG model is improved using Equation (4.24) and extends to

$$\boldsymbol{\Pi}_{j+1} = \boldsymbol{\Pi}_j - \boldsymbol{\alpha_0} \frac{RMS[\boldsymbol{X}]_{j-1}}{RMS[\boldsymbol{G}]_j} \cdot \nabla E \, . \tag{4.25}$$

Note that $\boldsymbol{\alpha_0}$ is set manually, depending on the motion profile of the mobile scanning platform. This introduces the possibility to lock dimensions from being optimized or even weighting dimensions based on their expected noise level. Although 6 DoF optimization generally works, reducing the optimization space is particularly useful if the source of error in the system is known and a model exists. That way, as we expect a spherical robot to move on a plane, the position along the axis perpendicular to the plane is constant and should not be used for optimization. Using this algorithm once, after finding correspondences from points to planes, leads to convergence to a local minimum which is often not an optimal solution. Even if we increase the number of iterations dramatically, no better solution than the local minimum is found. That is unless you consider updating the correspondence model after I iterations of gradient descent. Re-assigning point-to-plane correspondences this way J times, with a large enough J, leads to an optimal solution after maximum $I \cdot J$ iterations of gradient descent. The intuition behind this is to force the algorithm after I iterations to see if new correspondences arise, based on the current transformation. This way, we allow the algorithm to converge into a final set of matches before updating the global model, i.e., merging the planes.

Chapter 5

Results

This chapter evaluates the two major scientific contributions mentioned in Section 1.3, namely a novel point cloud segmentation method based on region growing with Bkd-trees, and a novel point cloud registration procedure based on scoring pairs of planar polygons. Therefore, the following presents the results in two distinct sections: The first one shows the results of the point cloud segmentation. It presents the resulting point cloud regions qualitatively, as well as an extensive runtime evaluation for both an artificial and a real-world dataset. The second shows the results of the registration procedure, i.e., improved map, improved 6 DoF path, and global plane model, on an artificial dataset, as well as on three real-world datasets with different motion profiles. A quantitative evaluation compares ground truth point clouds with the artificial dataset, as well as two of the real-world datasets. Furthermore, we make a comparison with a state-of-the-art semi-rigid registration (SRR) in terms of their accuracy and runtime, for one real-world dataset. Any results presented in this chapter have been gathered on a single thread by an Intel Core i7-10750H processor with a frequency of 2.6 GHz, with no running background processes except for the operating system (Ubuntu 18.04 LTS) ones. For the artificial dataset creation, we use a simulator developed at the University of Wuerzburg by Anton Bredenbeck (see source files at `https://github.com/fallow24/SphereTDP/tree/master/tdp/DataSetSim`). It provides the interfaces to define a robot consisting of an arbitrary sensor and mover. The sensor determines the elevation and azimuth of all rays of measurement, and the mover defines which movement the robot executes in a given timestep. Both components are equipped with an error model that models errors of a real measuring system. The robot is then placed in an artificial environment where the robot is simulated for a given amount of time.

5.1 Segmentation

5.1.1 Artificial Dataset

The previously mentioned simulator created this dataset by rolling a 3D laser scanner for 30 s inside a cube-shaped room with a side length of 3 m, free from any pose- or range noise. The simulated sensor is a Livox Mid-100 3D laser scanner. It has a non-repeating, flower-shaped scanning pattern, thus point coverage increases with time. Its horizontal field of view (FOV) is 98.4°, whereas its vertical FOV is 38.4°. The sensor produces about 300,000 pts/s and has a

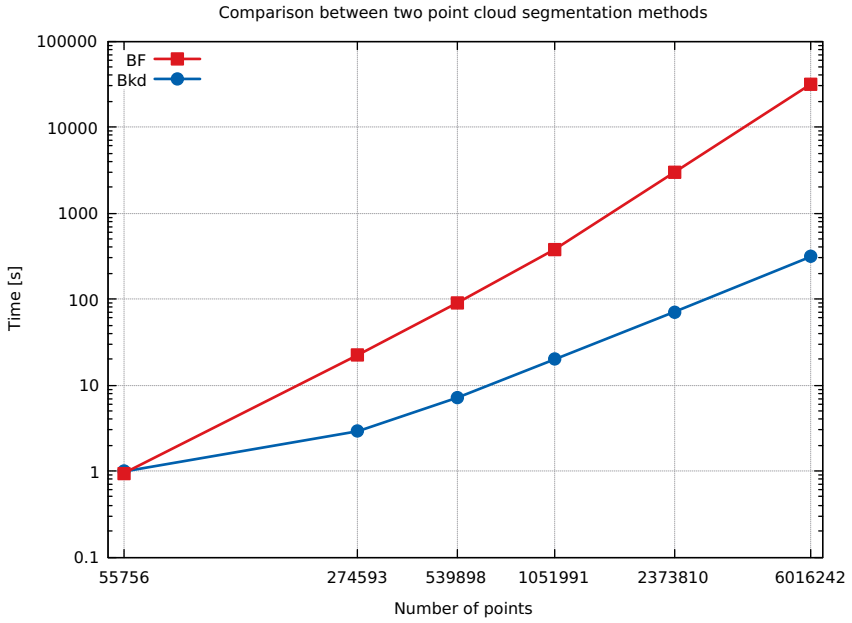


Figure 5.1: Runtime comparison for region growing between bruteforce method and Bkd-tree method, on reduced versions of the artificial dataset. The Bkd-tree method is able to reduce the time complexity for point-to-cluster distance calculation by orders of magnitude, resulting in overall less runtime.

minimum scan range of 1 m. The region growing parameters used were $d_{growth} = 5\,\text{cm}$, $\varepsilon_\alpha = 5°$, $n^c_{\text{min}} = 2000$, and $K = 20$. Figure 5.1 shows a runtime comparison between the Bkd-tree-based method and the brute force (BF) method for segmentation. The red line corresponds to the runtimes recorded for the BF method, while the blue line corresponds to the runtimes of the Bkd-tree method. Note that for a small number of points, in particular 55756 points for this dataset, the computational overhead created by the Bkd-tree makes the runtime worse than with the BF method. However, when increasing the number of points, the new method outperforms the BF approach by orders of magnitude due to its improved time complexity. Figure 5.2 shows the result of the segmentation. There seems to be a hole at one side of the cube. Yet this results only from the restricted FOV of the sensor. All six sides of the cube get segmented in the correct way, whereas potetial problematic regions at the edges and corners of the cube are correctly identified as non-planar areas.

5.1.2 Real World Dataset

In this section, a highly precise terrestrial Riegl VZ-400 laser scanner collected the point cloud on the market square in Wuerzburg, Germany. It is a rotating line scanner with a vertical (line) FOV of 100°, horizontal FOV of 360°, and angular resolution of 0.0005°, that produces highly precise point clouds. The dataset is available for public use on the homepage of 3DTK.[1] The region growing parameters used are $d_{growth} = 50\,\text{cm}$, $\varepsilon_\alpha = 10°$, $n^c_{\text{min}} = 500$, and $K = 40$. Figure

[1] `http://kos.informatik.uni-osnabrueck.de/3Dscans/wue_city.tar.xz`

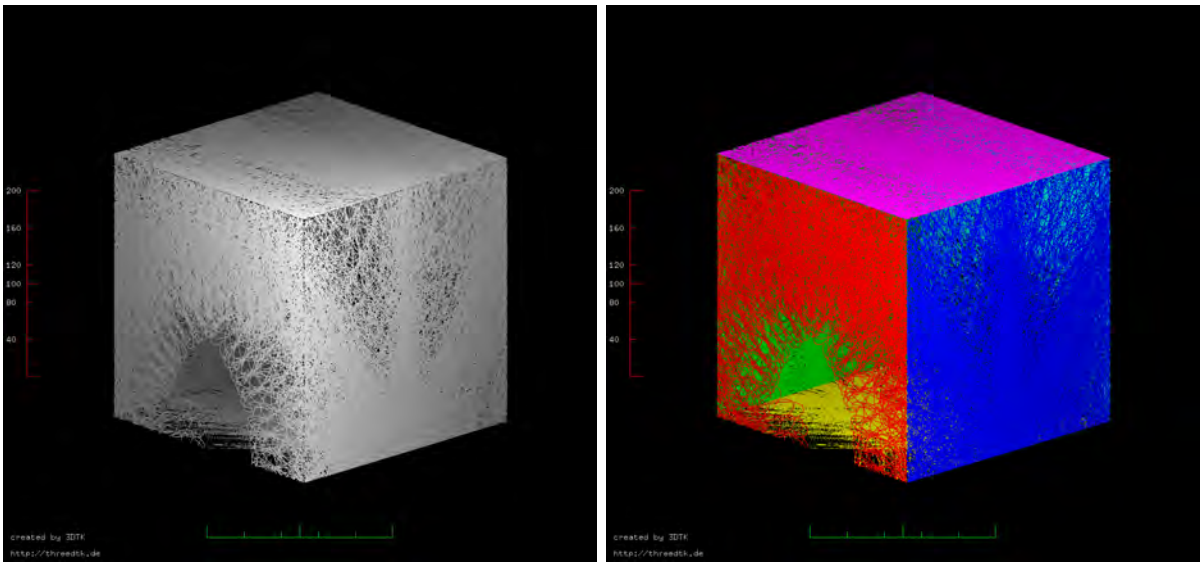


Figure 5.2: Result of the clustering algorihtm on an artificial dataset. (Left) Artificial, noise free, cube shaped point cloud. (Right) Resulting point cloud segmentation using $d_{growth} = 5\,\text{cm}$, $\varepsilon_\alpha = 5°$, $n^c_{\text{min}} = 2000$, and $K = 20$. In the right image, points with the same color belong to the same cluster.

5.3 shows the runtime comparison between the Bkd-tree-based method and the BF method for calculating point-to-cluster distances during point cloud segmentation. For this dataset, the time differences between the former (red line) and the latter method (blue line) are smaller than in the artificial dataset from the previous section (cf. Figure 5.1). This is because in the real dataset there are planes in abundance, while in the artificial dataset there are only six of them. Therefore many individual clusters in the real world dataset contain fewer points, making the computational overhead by the Bkd-tree method more impactful. Nonetheless, the difference between the two methods increases perceptibly as the number of points increases. For example, clustering 7821724 points in this dataset takes approx. 150 min using the BF method. Doing the same with the Bkd-tree method takes only approx. 40 min, which is less than half the time that BF needs. Figure 5.4 shows the result of the segmentation. The clustering algorithm correctly omits sharp, non-planar features like the church spires and identifies planar regions like the floor, exterior walls, or rooftops.

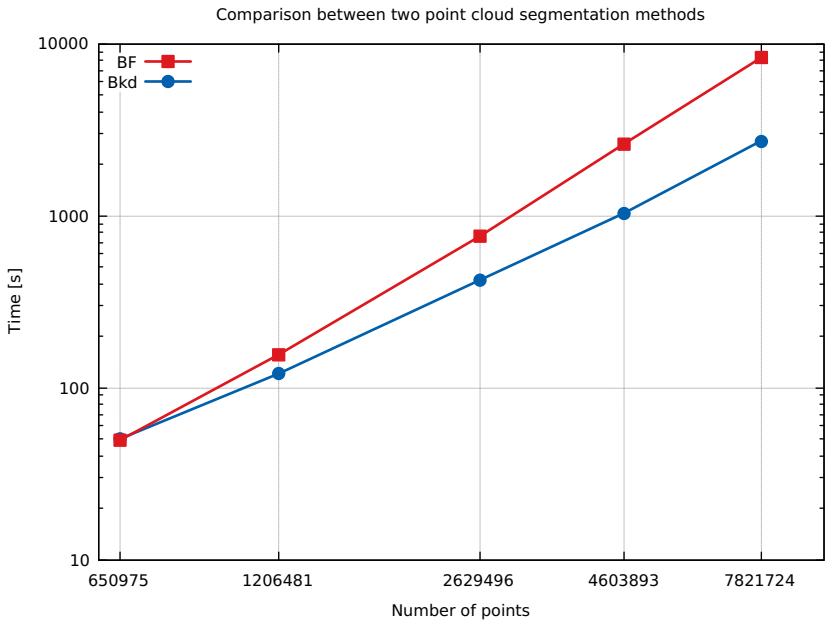


Figure 5.3: Runtime comparison for region growing between bruteforce method and Bkd-tree method, on reduced verions of a real world, market square dataset. The Bkd-tree method is able to reduce the time complexity for point-to-cluster distance calculation by orders of magnitude, resulting in overall less runtime.

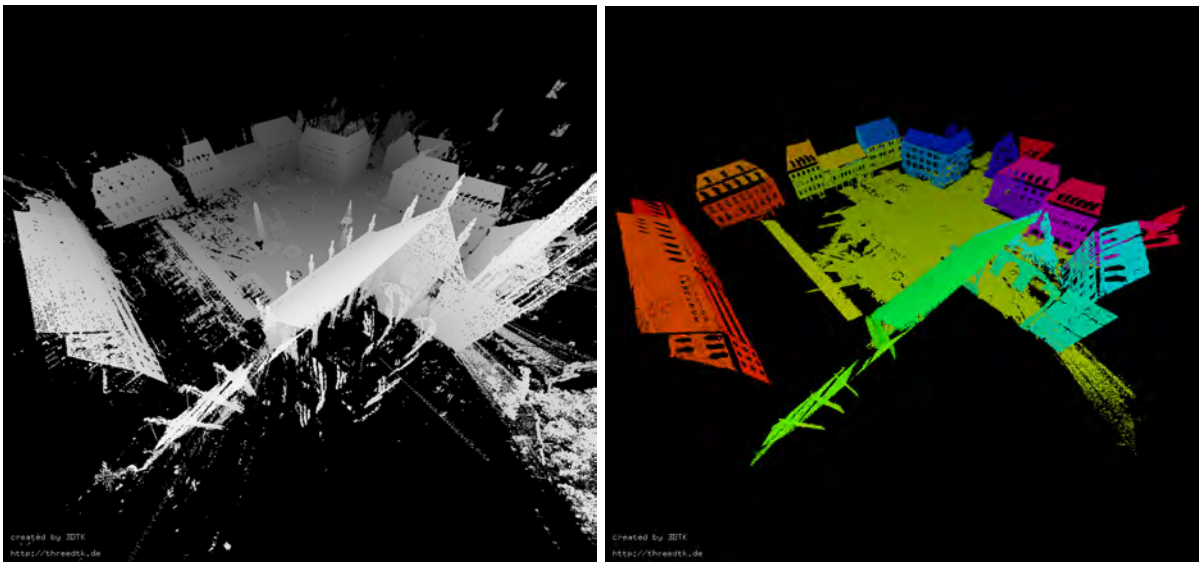


Figure 5.4: Result of the clustering algorithm on a real world, market square dataset from Wuerzburg. (Left) Point cloud before applying clustering algorithm. (Right) Resulting point cloud segmentation using $d_{growth} = 50\,\text{cm}$, $\varepsilon_\alpha = 10°$, $n^c_{\text{min}} = 500$, and $K = 40$. In the right image, points with the same color belong to the same cluster.

	P90	P95	P98
Uncorrected	372.1 cm	553.4 cm	827.9 cm
Corrected	35.9 cm	64.1 cm	122.8 cm

Table 5.1: Comparison of point-distances in the uncorrected and corrected simulated dataset. To be read as follows: "For 90% of all points in the uncorrected dataset, the corresponding distance to their ground truth match is less than or equal to 372.1 cm."

5.2 Registration

5.2.1 Artificial Dataset

The artificial dataset used in this section was created by simulating the previously mentioned rolling Livox Mid 100 sensor (see Section 5.1.1) in a rectangular corridor of length 100 m, with 3 m height and 4 m width. Both range- as well as pose-measurements, underly the influence of simulated noise. The presented registration algorithm is applied to the simulated dataset without further preprocessing. Assuming this represents a coarsely pre-registered 3D point cloud, the distances to the ground truth are evaluated before and after the registration. Figure 5.5 shows the different point-to-point distances. Before the registration, the corridor is only represented acceptably in the front part. The further into the corridor, i.e., the longer the robot accumulates errors, the more imprecise the data becomes. Finally, we see that many points exceed the threshold of 1 m and thus being mapped to the same color value. After registration, we see that, qualitatively, the ideal corridor was nearly restored from the noisy data. In particular, a very large part of the points (90%) has a distance of less than 35.9 cm. Table 5.1 shows the comparison of further percentiles of both datasets. Further, the square and straight shape of the corridor is restored well, and especially a large amount of points with an error of greater than 1 m is removed. Any such errors tend to occur at the back and the front of the corridor where the measured range is the largest hence has the largest contribution of the range error.

5.2.2 Real World Datasets

There are three real-world datasets with different motion profiles and/or scanning patterns. All of the following experiments have been executed with spherical robots. This is due to recent developments within the context of "Descent And Exploration in Deep Autonomy of Lava Underground Structures" (DAEDALUS) [45], which also was a Concurrent Design Facility (CDF) study by the European Space Agency (ESA). In the study, the authors proposed a spherical robot for mobile mapping due to many advantages regarding locomotion and sensor protection.

- A dataset collected with a 2D line scanner inside a floating sphere, allowing only for 3D rotation, but not translation. The line scanner is a Sick LMS141 laser scanner, which has a vertical FOV of 270°, scanning range from 0.5 m up to 40 m, a minimum angular resolution of 0.25°, and a maximum scanning frequency of 50 Hz. The hardware setup was designed and built by the Cylon team (Ignacio Dorado Llamas, Timothy Randolph, and

Camilo Andres Reyes Mantilla) at the University of Wuerzburg. The dataset was acquired by Timo Burger on the floating desk table, as shown in Figure 5.6 in a sequence of images.

- A dataset collected with a 3D laser scanner inside a rolling sphere, where a 2D translation on the ground plane is also possible. The 3D laser scanner in this dataset is a Livox Mid 100, as previously described in Section 5.1.1. The robot estimates the distance traversed with a recent IMU-only based approach for spherical robots [57], using the spheres radius and its angular velocity to obtain the covered distance on the ground plane. The dataset was acquired in an indoor living environment (a bedroom) by rolling the sphere manually with a controlled, constant rotation. The prototype, which is shown in Figure 5.7, was built by students at the University of Wuerzburg (Jasper Zevering, Anton Bredenbeck, and Fabian Arzberger) in the context of DAEDALUS [45].

- A dataset collected with a 3D laser scanner mounted to a crane. The 3D laser scanner in this dataset is, again, a Livox Mid 100, as previously described in Section 5.1.1. The sensor unrestrictedly rotates around the descending axis, corresponding to the cable direction. Let's call this direction the z-direction. The descending laser scanner, therefore, traverses only in the z-direction, while the cable introduced non-linear rotations around the z-axis. We connected the laser scanner to an outsourced processing machine via a 50 m tear-resistant tether cable (Fathom ROV Tether by BlueRobotics) which was rolled around a coil in order to perform the descending and ascending movement of the sphere, as shown in Figure 5.8. A spin encoder measures the rotation of the coil which directly corresponds to the height of the sphere according to the helix arc length formula. That way, we estimate the position (only in the z-direction) of the laser scanner. The descent of the sphere covered a distance of 22 m (see Figure 5.8) and was performed within a duration of 402 s.

For the estimation of orientation in all three setups, we use at least one PhidgetSpatial Precision 3/3/3 1044_0 inertial measurement unit (IMU). The following sections present the raw, unprocessed datasets, as well as the resulting point cloud, path, and global plane model after applying the presented algorithm. The accuracy of the results is evaluated by comparing them with ground truth point clouds, obtained with the previously described terrestrial Riegl VZ 400 laser scanner (see Section 5.1.2). Unfortunately, a ground truth dataset is not available for the first setup (floating sphere), as the room has been refurbished since the first data acquisition.

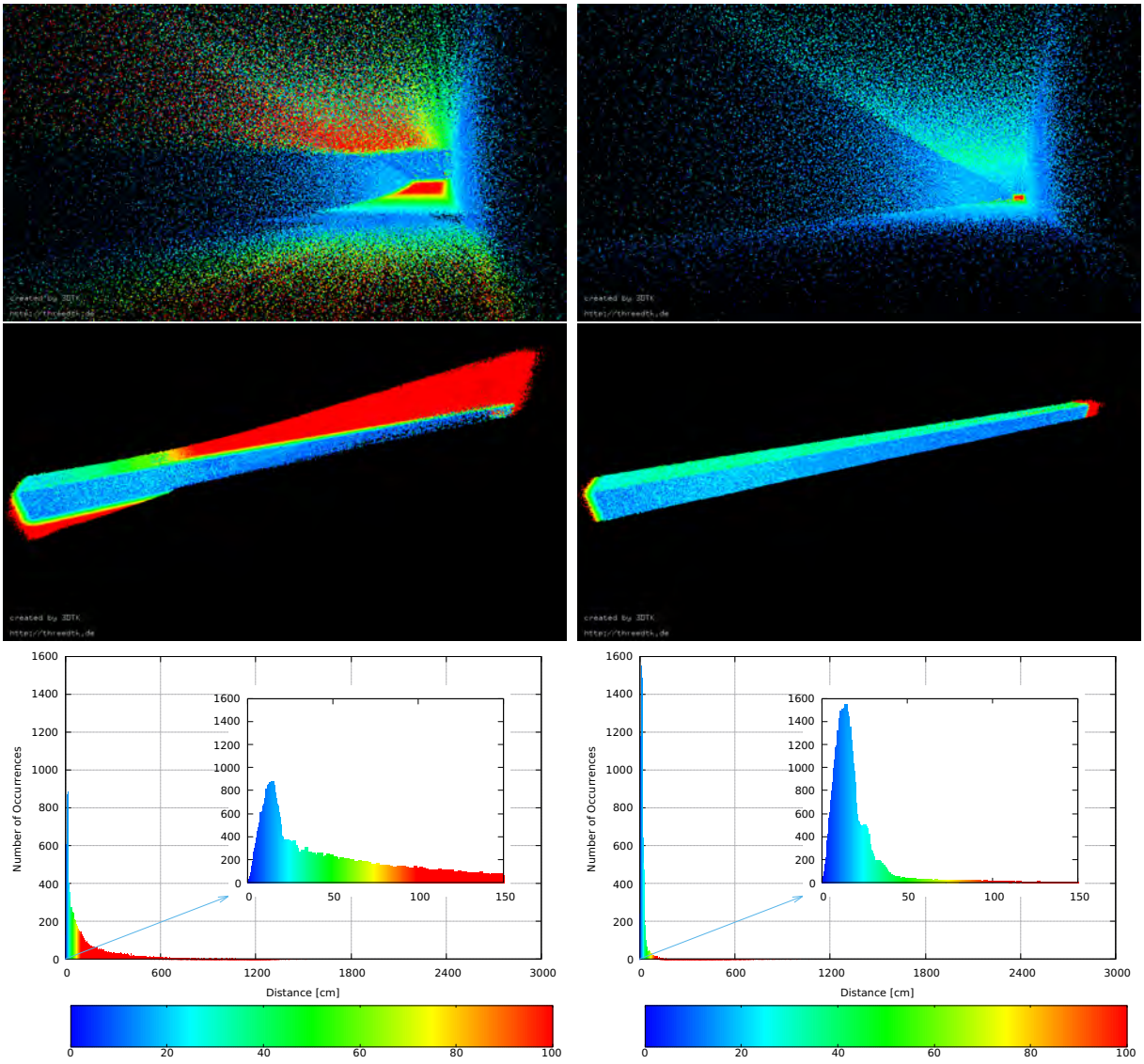


Figure 5.5: Evaluation of point distances before (left) and after (right) the plane-based registration on a simulated dataset. Lateral images always have the same orientation. A maximal distance of 30 m is set, such that all points that display a higher distance value are excluded from the analysis. Both point-clouds were reduced before evaluating point-to-point distances to the ground-truth. Further, the color-space maps all values with a distance greater than 1 m to the same color. The top two columns show a heat map of distances, while the bottom shows the corresponding histogram. The color mapping is equivalent in both. An animation of the matching process is given at `https://youtu.be/hFx2uGkUdXw`.

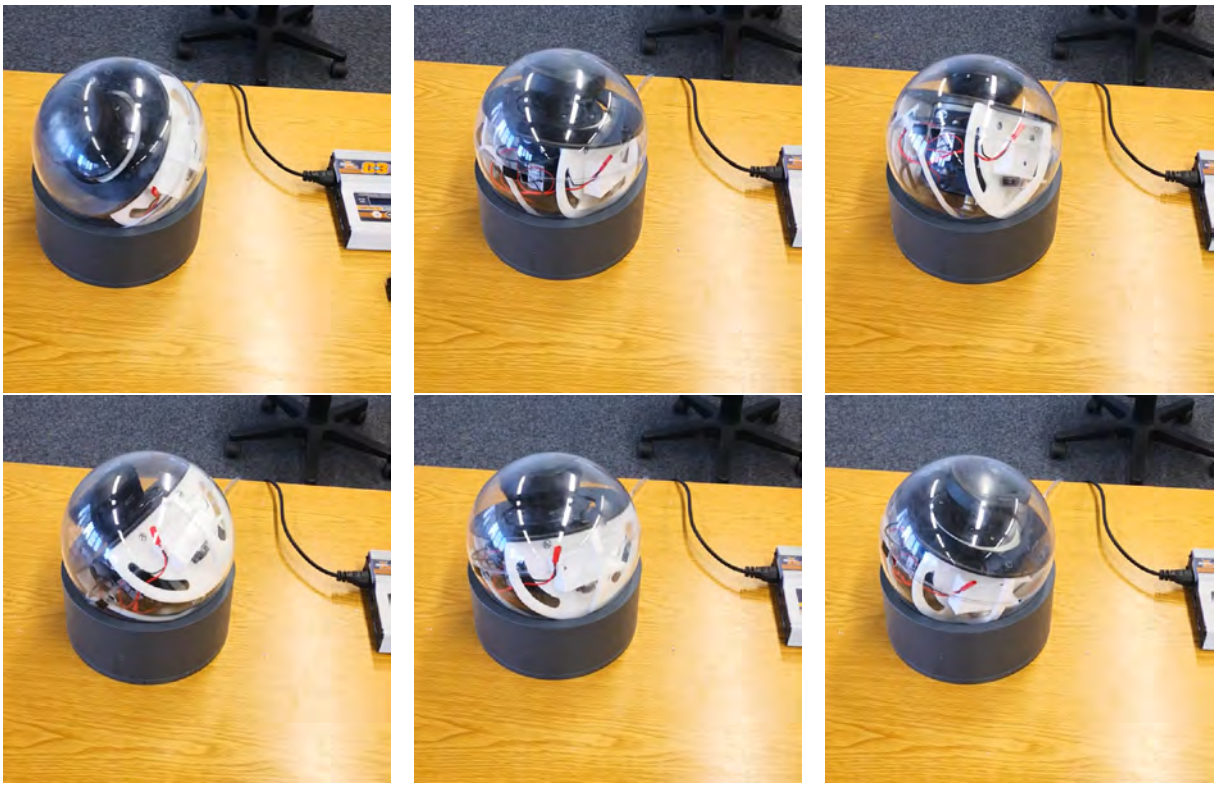


Figure 5.6: Sphere rotating on a floating table. Inside the sphere is a 2D line scanner (Sick LMS141) and IMU, creating a 3D point cloud while rotating.

Figure 5.7: Prototype used for data acquisition with the rolling sphere. The main payload is the Livox Mid-100 laser scanner. For pose-estimation, three IMUs of the manufacturer Phidget are placed inside and a Raspberry Pi 4 for the calculations. On the top are two batteries, and on the bottom one voltage stabilizer and the breakout box of the laser scanner.

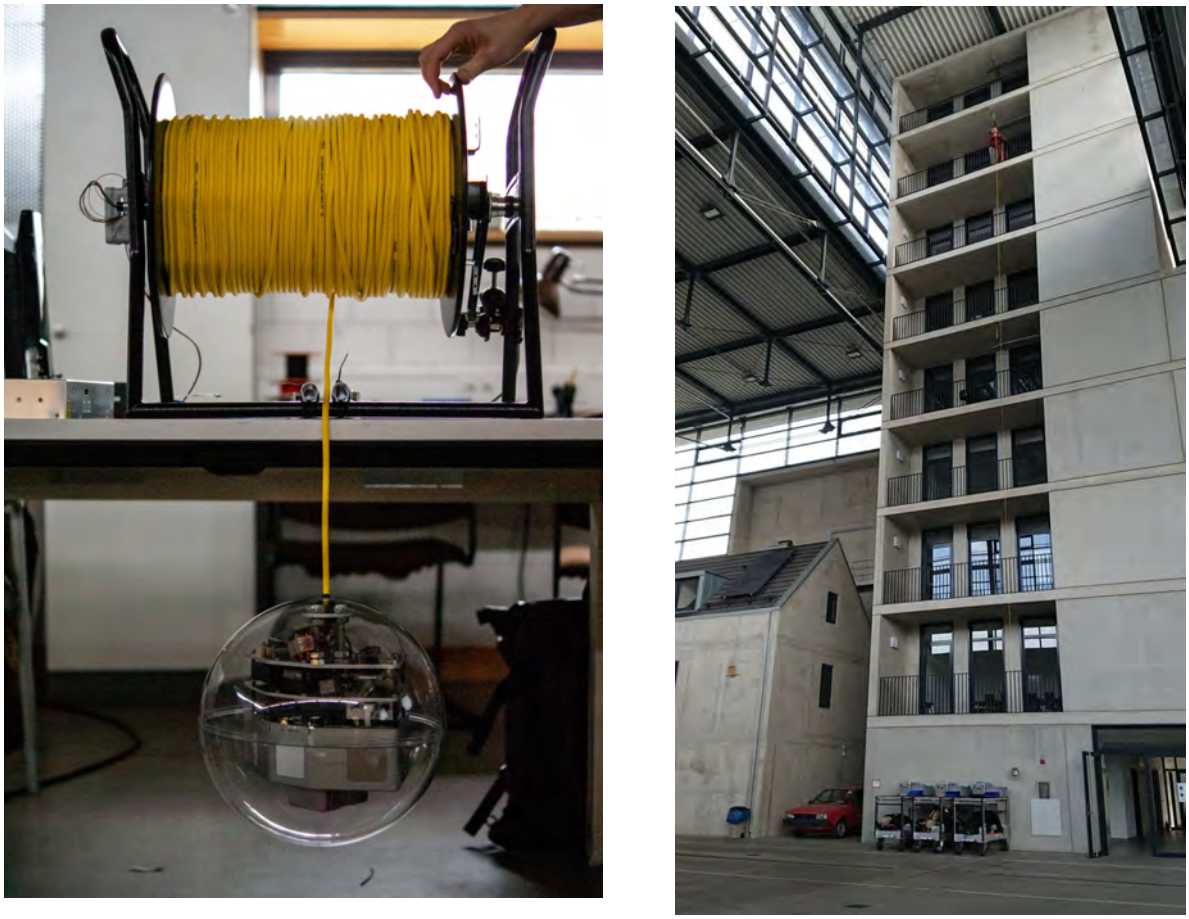


Figure 5.8: Hardware and setup of the crane descent experiment. (Left:) Experimental setup showing the connection of the Livox Mid 100 laser scanner with the tether cable that is rolled around a cable reel. (Right:) Image showing the test sphere as it gets descended 7 floors of a firestation building. Inside the sphere, three IMUs of the manufacturer Phidget are placed. The height of the sphere is measured by a spin encoder at the cable reel.

Floating Sphere

This section shows the results of the first dataset, which has been acquired as described in the previous section. The pre-registration is obtained by determining the orientation of the sphere via Madgwick filtered [35] IMU measurements. Note that for this dataset, a 2D laser scanner is used instead of a 3D laser scanner. To increase the number of possible point-to-plane correspondences we condense twenty temporally successive 2D scans into one 3D linescan, which is then globally registered. We always use the scan at the median index as a reference coordinate system. This does not only speed up convergence due to the proportional effect on the error function but also decreases the risk of transforming a single line scan incorrectly. Transformations like these happen in particular for a small collection of points as outliers have more influence. The following parameters were used for optimization: $S = 20$, $\varepsilon_H = 50$, $\varepsilon_{\text{PPD}} = 200$, $\varepsilon_\alpha = \frac{\pi}{4}$, $K = 20$, $d_{growth} = 50$, $n^c_{\min} = 200$, $\boldsymbol{\alpha_0} = [0.01, 0.01, 0.01, 0, 0, 0]^\tau$, $I = 8$, and $J = 100$. Figure 5.9 shows the results obtained before and after employing the plane-based registration on the dataset acquired by the floating sphere. After the registration, the walls of the room are significantly more prominent in the point cloud. Further, the deviation of points around the walls is notably smaller as the points are moved on their respective plane. Some outliers remain after the registration, especially with increasing distance from the laser scanner. This is because point density is lower there, resulting in fewer points. Hence their influence on the optimization gradient is also small. The proposed solution to this problem is to consider the point density in a local neighborhood of points and weigh their effect on the gradient accordingly. Points that have less local density should therefore be weighted more. Figure 5.10 shows the convex hulls of the planes that have been found by the clustering algorithm during segmentation. Multiple convex hulls represent the same plane. This is due to the faulty behavior of the matching algorithm (see Section 4.5), as it is not able to properly update the global plane model with the individual linescans. One conceivable solution to this problem is to implement a loop closing technique. Without such, the global planes do not merge with already existing planes after they have been registered. However, two initially unconnected planes potentially connect when a third plane is introduced, which overlaps with both. Despite that, the presented algorithm was able to improve the map quality of the environment, even if some outliers remain.

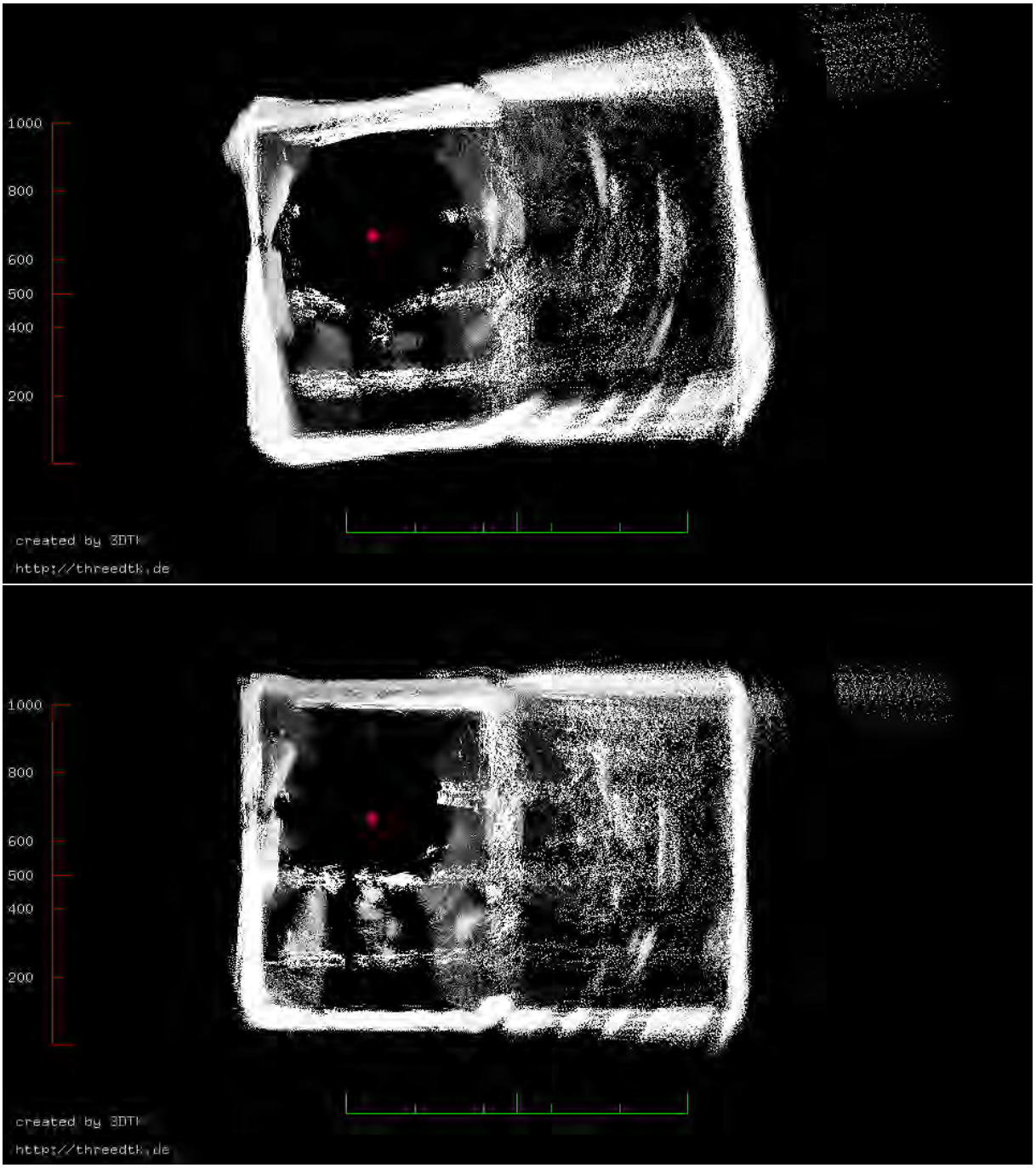


Figure 5.9: Comparison between the unprocessed and resulting point cloud for the floating sphere dataset. (Above) Birds-eye view of the unprocessed point cloud from the floating sphere. (Below) Birds-eye view of the resulting point cloud after applying the presented algorithm. An animation of the registration process is given at `https://youtu.be//ov5Xjgyl9wA`

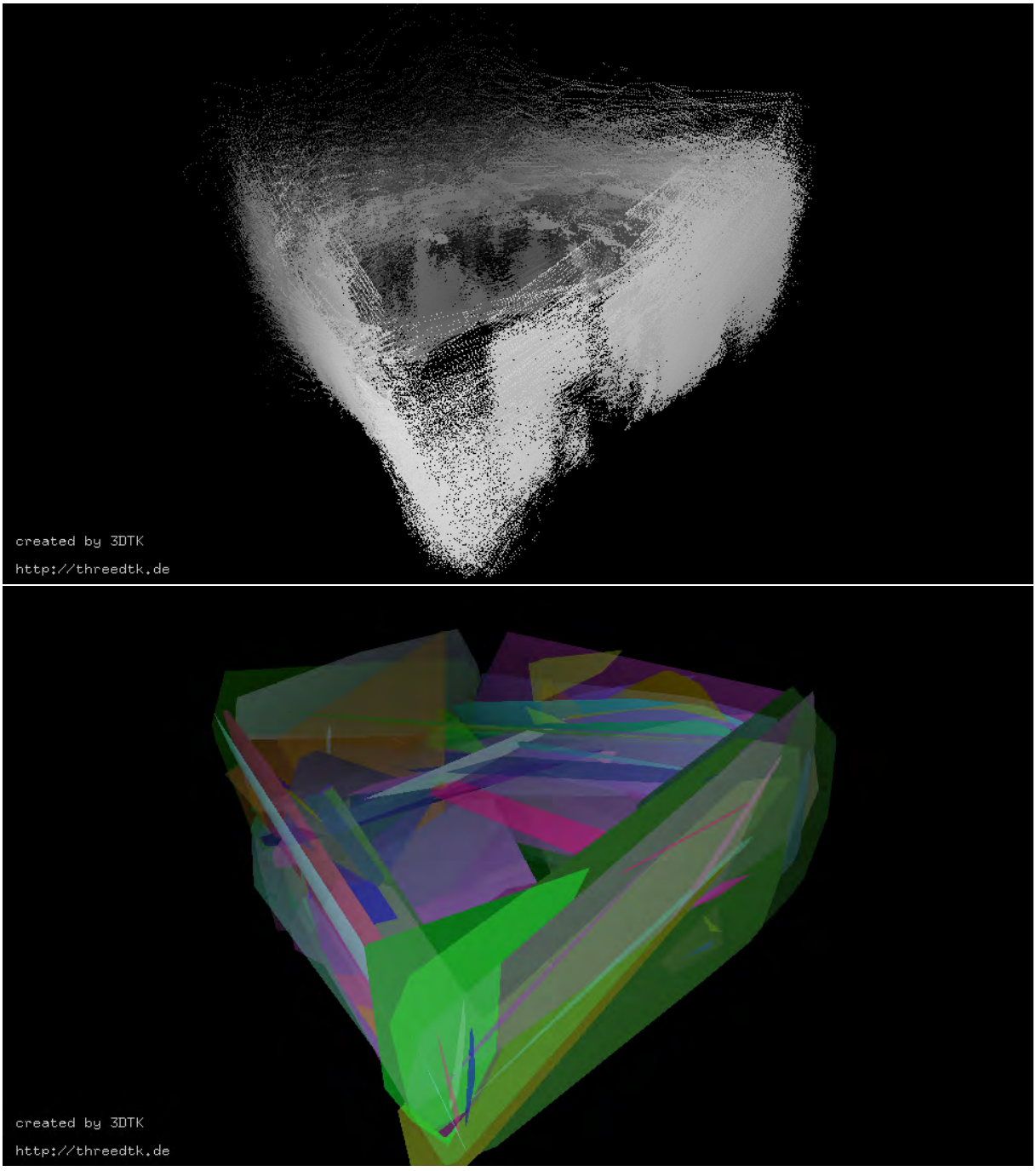


Figure 5.10: Resulting global plane model for the floating sphere dataset. (Above:) The resulting point cloud after applying the presented registration algorithm. (Below:) Resulting convex hulls from the segmentation, which happend during registration. Many clusters corresponding to the same plane are visible.

	P90	P95	P98
Uncorrected	61.3 cm	74.3 cm	88.9 cm
Corrected	17.4 cm	22.0 cm	26.0 cm

Table 5.2: Comparison of point-distances in the uncorrected and corrected indoor real world dataset for the rolling sphere. To be read as follows: "For 90% of all points in the uncorrected dataset, the corresponding distance to their ground truth match is less than or equal to 61.3 cm."

Rolling Sphere

This section shows the results of the second dataset, which has been acquired as described in Section 5.2.2. Figure 5.11 shows a comparison between the unprocessed point cloud, pre-aligned only with the IMU measurements [57], and the resulting point cloud after applying the presented algorithm. For a better view, the ceiling has been cropped in the following images. The presented algorithm used the following parameters for registration: $S = 25$, $\varepsilon_H = 200$, $\varepsilon_{\text{PPD}} = 300$, $\varepsilon_\alpha = \frac{\pi}{4}$, $K = 200$, $50\,\text{cm} \leq d_{growth} \leq 100\,\text{cm}$, $n^c_{\text{min}} = 800$, $\boldsymbol{\alpha_0} = [0.001, 0.001, 0.001, 1, 0, 1]^\tau$, $I = 1500$, and $J = 1$. Before registration, the walls of the environment are present multiple times in the point cloud but displaced with an offset of more than 120 cm. After we apply the presented registration algorithm, the displaced walls are globally aligned. However, the alignment is still not perfect. In particular, a wall's thickness increases when comparing it with individual scans of the same wall. The same Figure also shows the path which results from the IMU measurements, with the optimized path. Looking at the above image, it appears that the robot crashes through the wall, which, of course, did not happen during the experiment. With the optimized path (below image), this does not happen anymore, as it is globally consistent with the map. However, the resulting path looks more distorted than before. This is because, for some individual linescans, only a senseless transformation is found due to the faulty behavior of the matching algorithm, which happens especially for linescans that have only a few points. Ideas exist for solving this problem, e.g., excluding small linescans from the optimization and use path interpolation for them instead. Figure 5.12 directly compares the resulting point cloud after registration with the ground truth point cloud. Note that the dataset acquired with the rolling robot does not have points on the floor due to the minimum scanning distance of the sensor, which is 1 m. Figure 5.13 shows an evaluation of point-to-point distances to ground truth. The coloring of the images represents the distance each point in the cloud has to its corresponding point in the ground truth point cloud. Any distance that is larger than 30 cm gets mapped to the same color (red). The left column corresponds to the unprocessed point cloud, whereas the right column corresponds to the point cloud after applying the presented algorithm. The histograms show that before application of the registration algorithm, the majority of points (98%) have distances to ground truth of 88.9 cm or less. After applying the presented algorithm, the same percentage of points have distances of only 26.0 cm. Table 5.2 shows the comparison for further percentiles between the uncorrected and corrected dataset. Furthermore, in the histograms, the peak for points that have distances less than or equal to 0.3 cm has more than doubled in size. Figure 5.14 shows the extracted convex hulls of the clusters which the segmentation algorithm found during registration. Unlike with the dataset in the previous section, where a 2D scanner

was rotated, the vertical FOV is now higher for each linescan. Therefore, clusters are more likely to overlap, which is why the matching algorithm merges more clusters. The bottom right image shows an exterior view of the resulting plane model. The walls and ceiling each belong to distinct clusters. Since the ground is barely populated with points, there are multiple smaller clusters, as shown in the upper right image. Note that there are even clusters for the window- and door frames. In general, the improved map and exported plane model resemble the environment well.

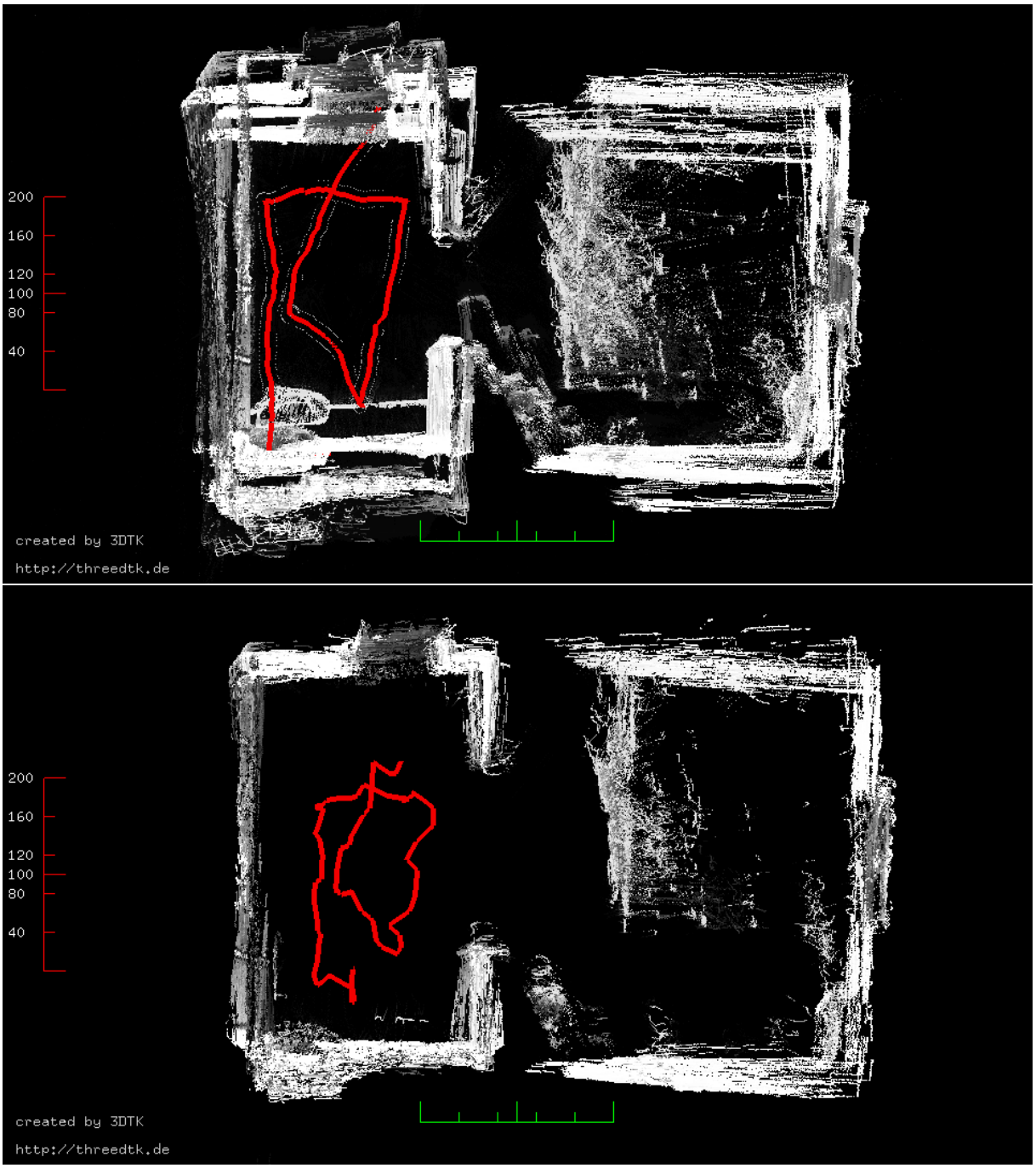


Figure 5.11: Comparison between the unprocessed point cloud obtained by the rolling sphere, pre-registered only using IMU measurements (above) and the resulting point cloud after applying the presented registration algorithm (below). For a better view, the ceiling has been cropped. Both images were captured from the same pose. The contrast between black and white corresponds to reflectivity of the point cloud. The traversed path is shown with the red line in both images.

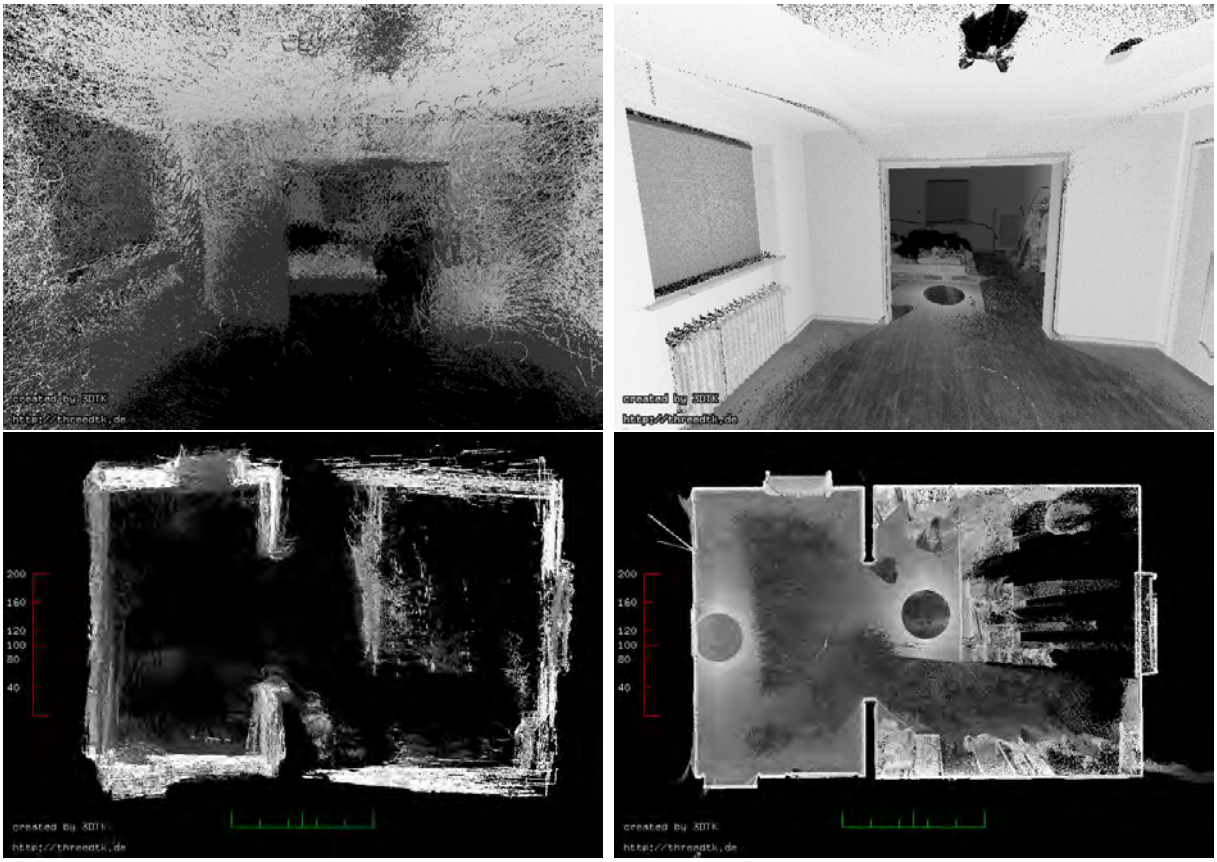


Figure 5.12: Comparison between the resulting point cloud after applying the presented registration algorithm (left column) and the ground truth point cloud (right column) for the rolling sphere dataset. The above images show an indoor view of the scene, captured from the same pose. The below images show a birds-eye view of the scene, also captured from the same pose. For a better view, the ceiling has been cropped. The contrast between black and white corresponds to reflectivity of the point cloud.

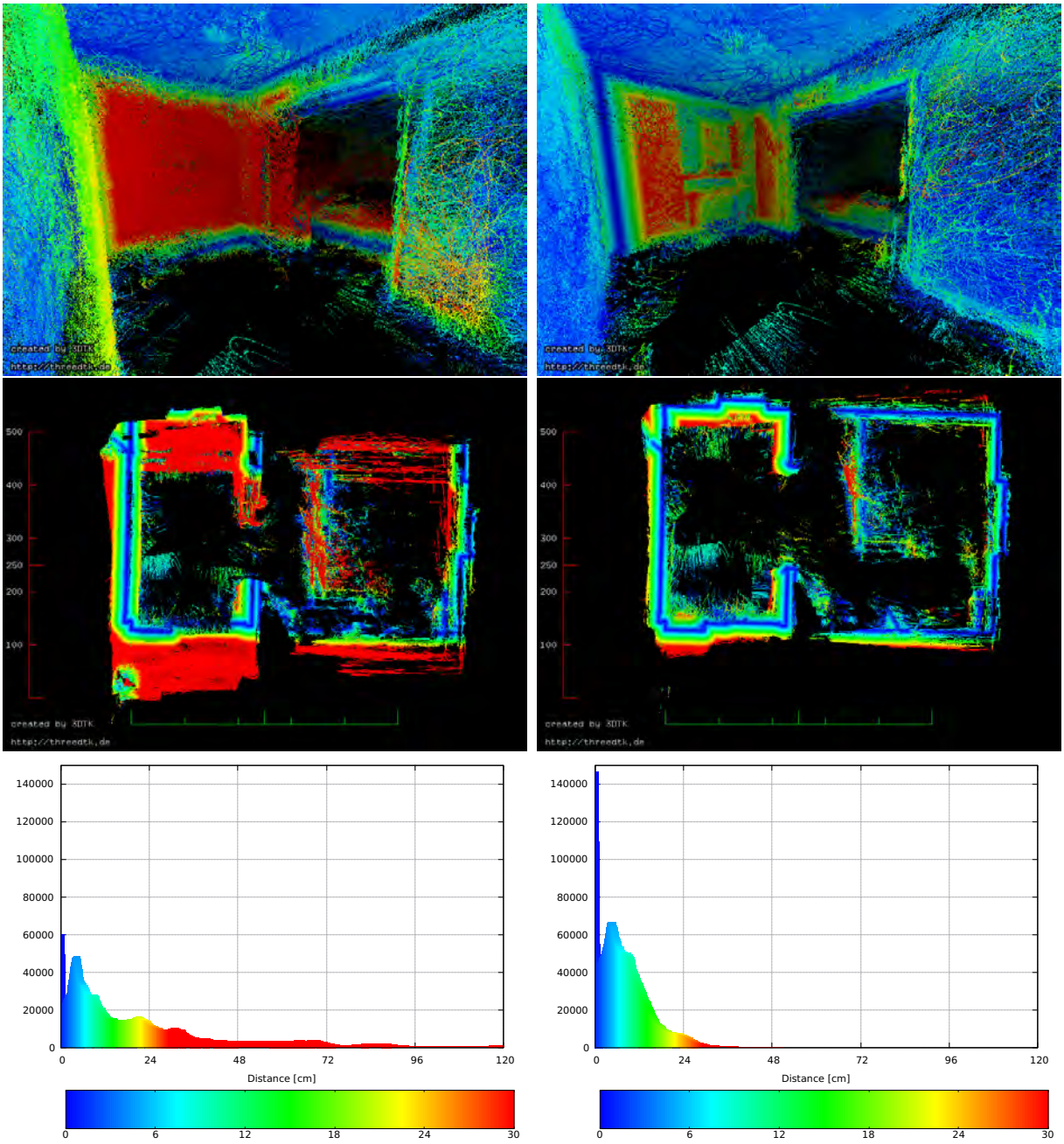


Figure 5.13: Evaluation of point distances before (left) and after (right) the plane-based registration on the rolling sphere dataset. Lateral images always have the same orientation. A maximal distance of 120 cm is set, such that all points that display a higher distance value are excluded from the analysis. Further, the color-space maps all values with a distance greater than 50 cm to the same color. The top two columns show a heat map of distances, while the bottom shows the corresponding histogram. The color mapping is equivalent in both. For the birds-eye view, the ceiling has been cropped to get a better view. An animation of the matching process is given at `https://youtu.be/Mkil0vLk8f8`.

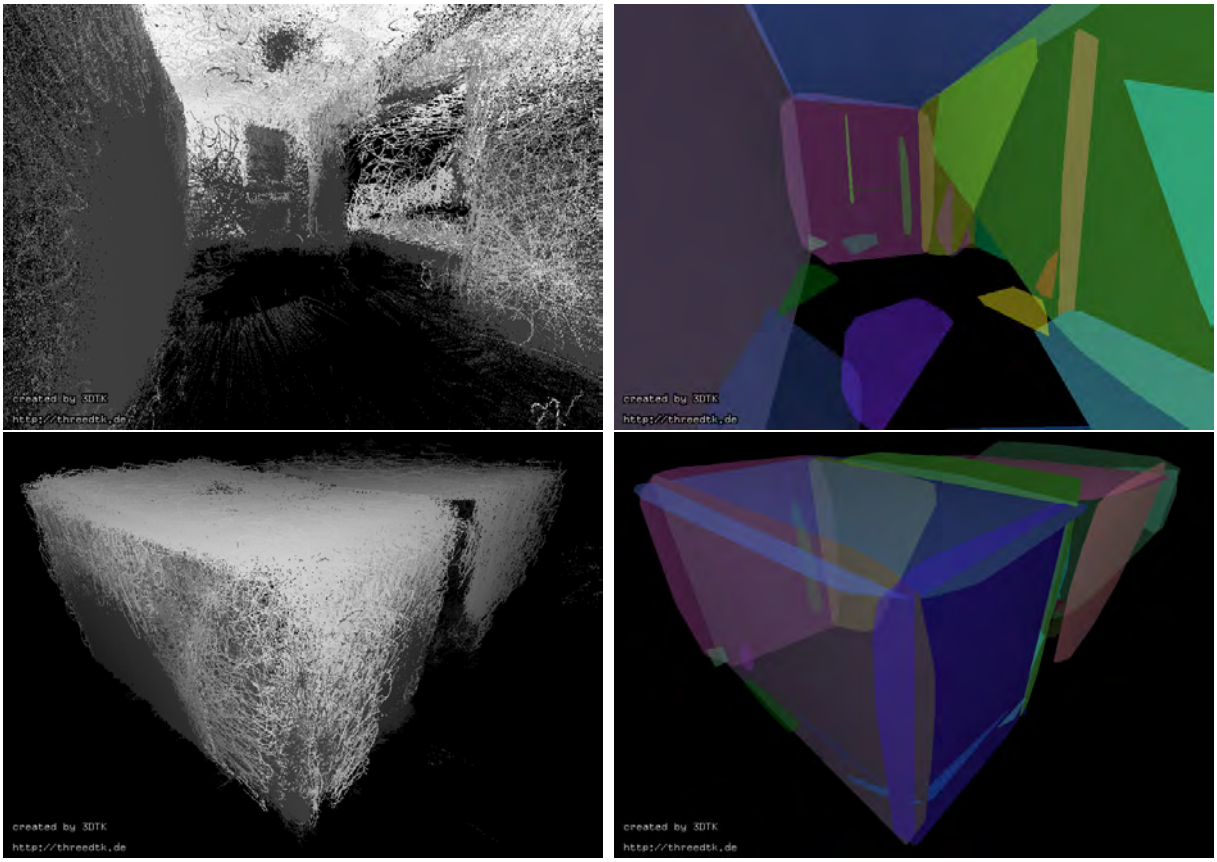


Figure 5.14: Resulting global plane model for the rolling sphere dataset. (Left column) The resulting point cloud after applying the presented algorithm. The above image shows an indoor view of the scene. The below image shows an exterior view of the scene. Black / White contrast corresponds to reflectivity of the point cloud. (Right column) Extracted convex hulls of the global point clusters, which are built up during segmentation and merged during registration. Lateral images always have the same orientation.

	P90	P95	P98
Uncorrected	621 cm	806.3 cm	1369.9 cm
Corrected	321.9 cm	433.8 cm	617.2 cm

Table 5.3: Comparison of point-distances in the uncorrected and corrected indoor real world dataset for the descending sphere. To be read as follows: "For 90% of all points in uncorrected dataset, the corresponding distance to their ground truth match is less than or equal to 621 cm."

Crane Descent

This section shows the results of the third dataset, which has been acquired as described in Section 5.2.2. The following parameters were used for optimization: $S = 20$, $\varepsilon_H = 50$, $\varepsilon_{\text{PPD}} = 50$, $\varepsilon_\alpha = 5°$, $K = 200$, $50\,\text{cm} \leq d_{growth} \leq 200\,\text{cm}$, $n^c_{\text{min}} = 200$, $\boldsymbol{\alpha_0} = [0, 0.01, 0, 0, 1, 0]^\tau$, $I = 5000$, and $J = 1$. The above image in Figure 5.15 shows the 6 DoF pose estimates, as collected directly from the IMU and the spin encoder. Further, you see the resulting point cloud when applying these estimates to their corresponding scans. The image below shows the 6 DoF pose estimates after the presented registration algorithm has been applied. Figure 5.16 compares the resulting point cloud after applying the optimized pose estimates (above image) with a ground truth point cloud, which was obtained by the previously mentioned Riegl VZ 400 terrestrial laser scanner (below image). Note that in the above image, even non-planar details like the lights on the wall are recreated well. This is because, in this environment, planes are excessively available in each linescan thus optimization leaves no room for ambiguities. Figure 5.17 evaluates the distances of the points in the cloud to ground truth before and after applying the presented registration algorithm. The left column corresponds to the point cloud before registration, whereas the right column corresponds to the point cloud after the registration has been applied. Note that in the left column, the images look oversaturated, since the initial orientation estimates underly large errors. The histogram in the left column shows that before registration, the majority of points (98%) have distances smaller than or equal to 1369 cm to their ground truth match. After registration, the same amount of points have distances smaller than or equal to 617 cm to their ground truth match. Table 5.3 shows the comparison for further percentiles between the uncorrected and corrected point cloud. Figure 5.18 shows the extracted convex hulls which were found during the segmentation and merged during registration. Some clusters still correspond to the same plane due to the lack of a loop closing technique. Section 5.2.2 describes this problem in more detail. However, the extracted global planes, as well as the improved point cloud, reproduce the overall structure of the environment well.

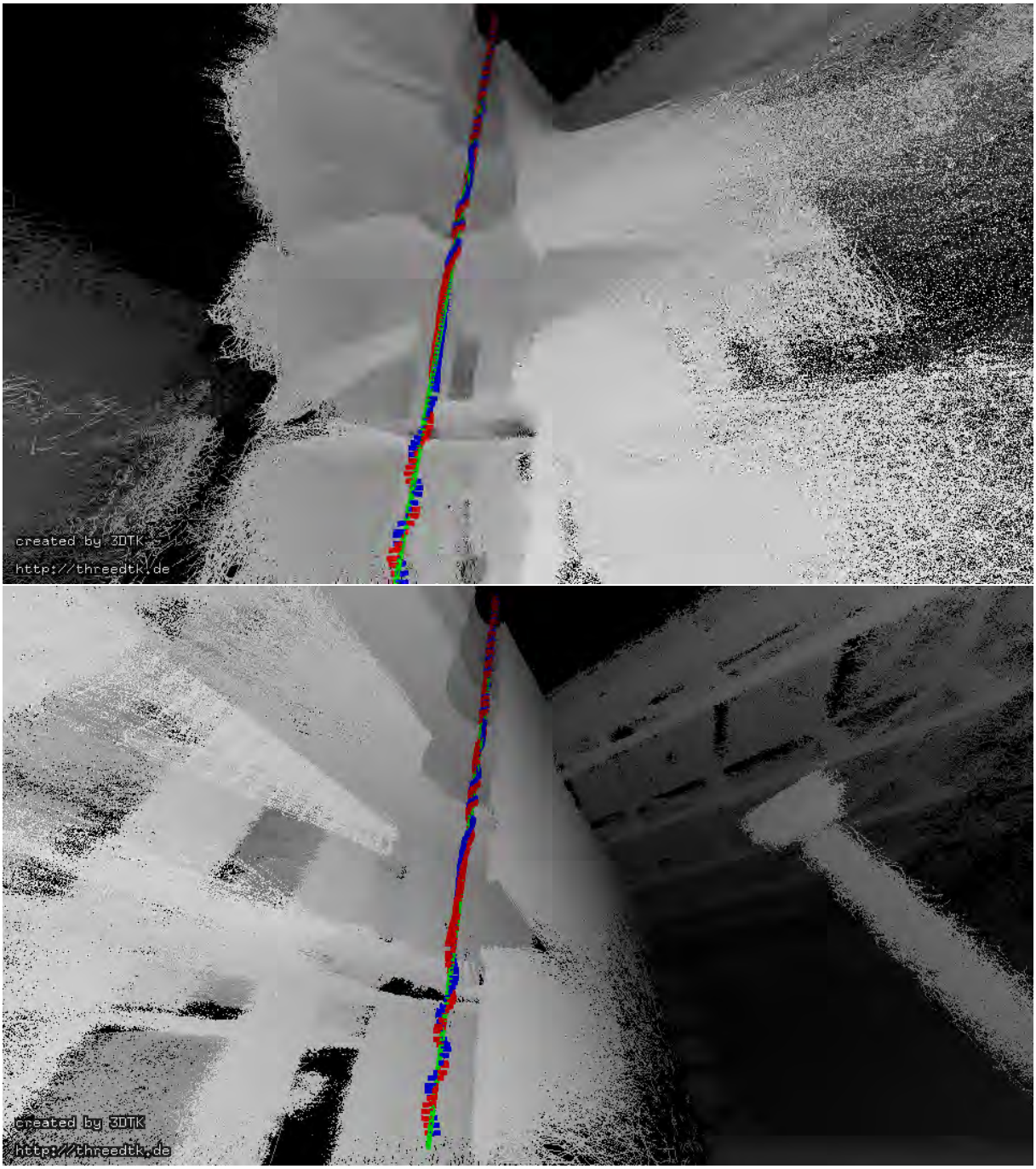


Figure 5.15: Comparison between the unprocessed and resulting point cloud and path for the crane descent dataset. (Above:) 6 DoF pose estimations of the descending robot, before application of the presented algorithm. (Below:) 6 DoF pose estimations of the descending robot after application of the presented algorihtm.

Figure 5.16: Comparison between the resulting point cloud after applying the presented registration algorithm (above) and the ground truth point cloud (below) for the crane descent dataset. In both images, an indoor view of the scene, captured from the same pose, is shown. The contrast between black and white corresponds to reflectivity of the point cloud.

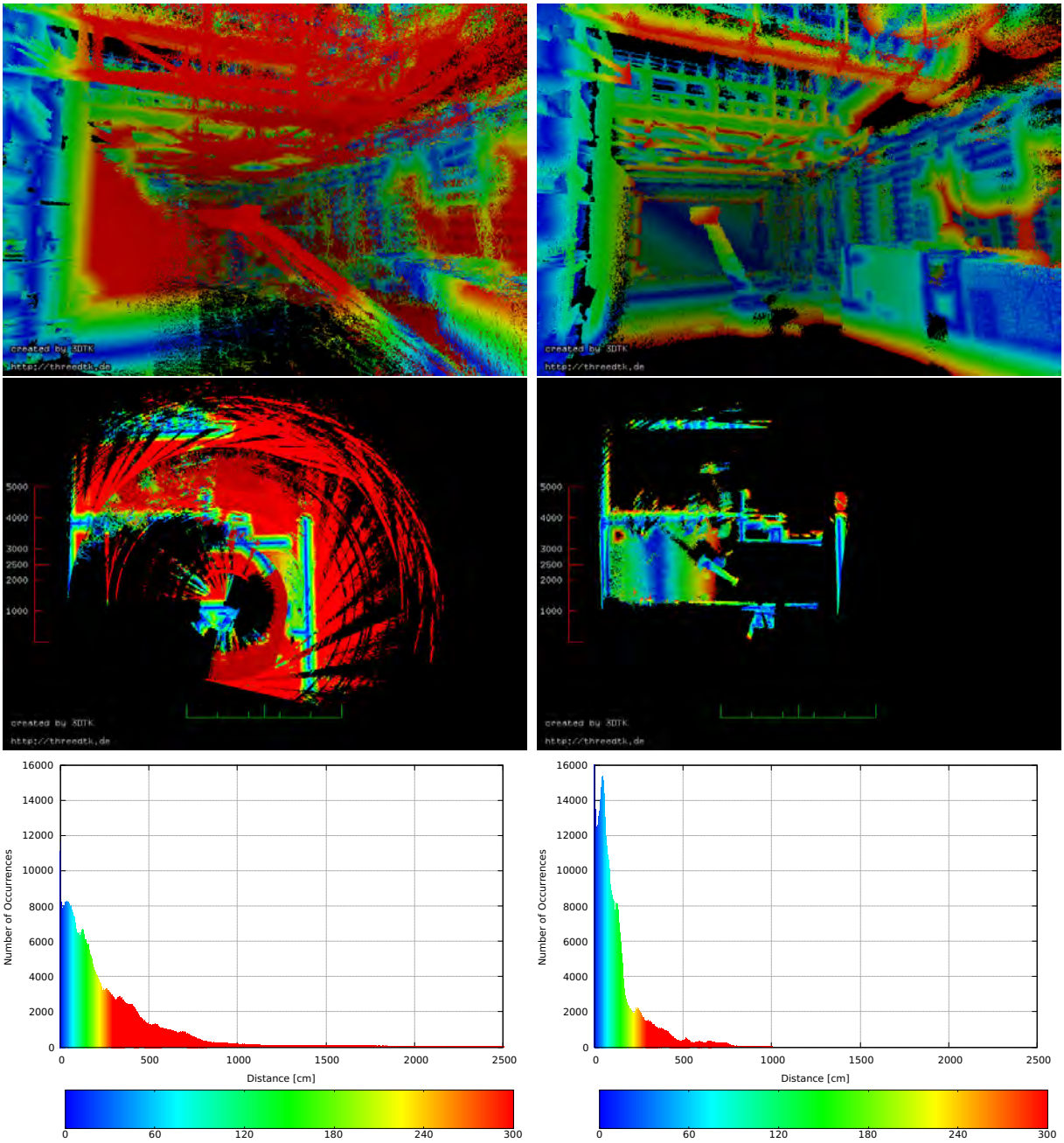


Figure 5.17: Evaluation of point distances before (left) and after (right) the presented registration on the crane descent dataset. Lateral images always have the same orientation. The color-space maps all values with a distance greater than 300 cm to the same color. The top two columns show a heat map of distances, while the bottom shows the corresponding histogram. The color mapping is equivalent in both. For the birds-eye view, the ceiling has been cropped to get a better view.

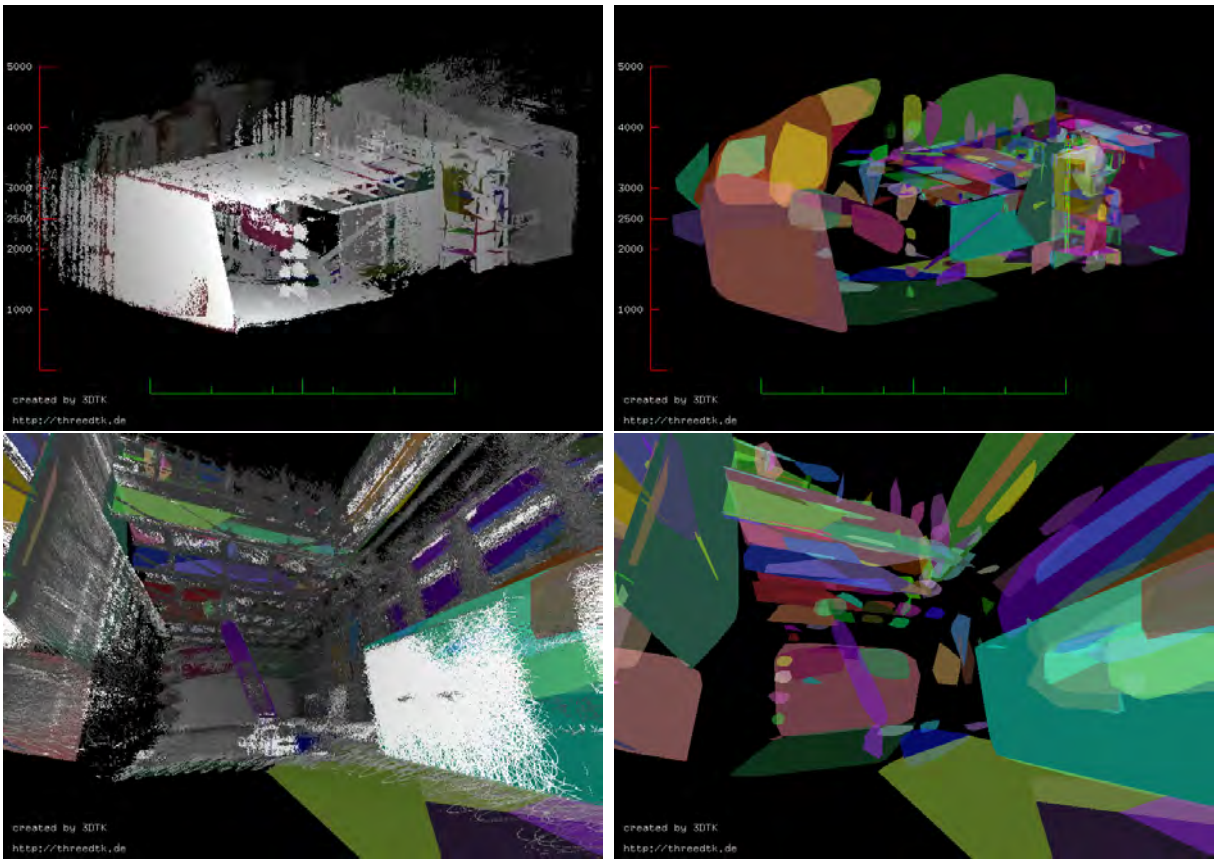


Figure 5.18: Resulting global plane model for the crane descent dataset. (Left column:) The resulting point cloud plus extracted planes after applying the presented algorithm. Black / White contrast corresponds to reflectivity of the point cloud. (Right column:) Extracted convex hulls of the global point clusters, which are built up during segmentation and merged during registration. The above images shows a side view of the scene, which was sliced to gain a better view. The below images shows an indoor view of the scene. Lateral images always have the same orientation.

5.2.3 Comparison with Semi-Rigid Registration

This section uses the dataset from the previous section to compare the novel, polygon-based registration method with an existing state-of-the-art, semi-rigid registration (SRR) [15] method. SRR uses iterative closest point (ICP) based registration as a preregistration method. The algorithm then establishes a correspondence graph between points in all linescans to perform semi-rigid SLAM. Therefore, SRR simultaneously corrects the 6 DoF path and improves map quality for a mobile system globally. Let us first discuss the accuracy that both algorithms achieved. Unlike the previous section, we directly compare their distances to ground truth. Note that the evaluation on this section happens on a reduced version of the previous dataset, thus the histogram in Figure 5.19 looks different than in Figure 5.17 from Section 5.2.2, despite corresponding to the same optimization. Figure 5.19 shows the evaluation of the point distances to ground truth after SRR and after the presented method has been applied to the same dataset. The results look very similar, i.e., both methods achieve a good approximation to ground truth. In particular, the majority of points (98%) have distances less than or equal to 240.6 cm after the application of SRR, and 250.7 cm after the application of the presented method. However, Table 5.4 shows for lower percentiles, the presented method achieves even better results than SRR. The histograms in Figure 5.19 show the highest peak at approximately the same location. However, the peak for the presented method is higher and located more towards the left, indicating that there are more points with less distance to ground truth, hence a higher accuracy for the presented method. An exact interpretation of this is difficult, though, when considering the above two images in Figure 5.19. Since SRR uses ICP based correspondences, it performs better in non-planar areas, e.g., at the turntable ladder and basket, or the details on the ground. The presented method is not designed for an explicit alignment of these parts of the point cloud. Rather, their alignment inherently results from the plane-based registration. Furthermore, both histograms in Figure 5.19 show a second peak, located to the right of the first peak. The second peak makes an interpretation even more difficult since the presented method has its peak located more to the right this time when compared to SRR. This means that there are more points with higher distances to ground truth, indicating less accuracy for the presented method. While a trained eye would likely suggest that SRR has higher accuracy, the presented results support this suggestion only partly.

The runtime evaluation yields a more distinctive result, though. Both methods combine 25 successive scans into one linescan, which is then globally registered. Further, both methods process only a reduced version of the point cloud, where each voxel of size 10 cm is allowed to have only one point. The runtime for SRR is significantly higher. Overall, SRR needs 6069.04 min (approx. 4.2 days) to achive the result shown in Figure 5.19 (left column). The presented method achieves comparable accuracy in only 60.63 min. This includes the runtimes needed for all steps described in Chapter 4, except for preprocessing steps.

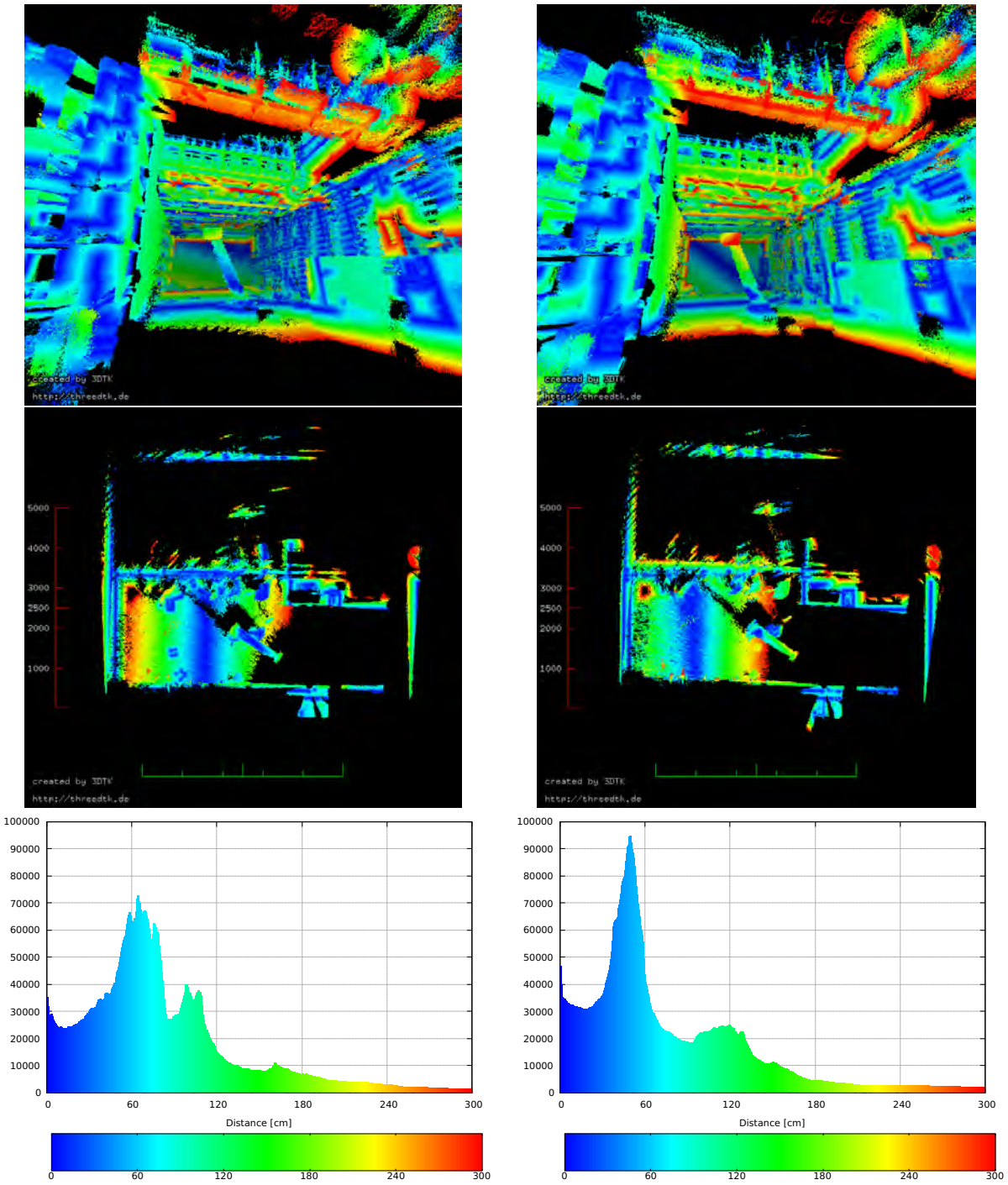


Figure 5.19: Evaluation of point distances after SRR (left) and after the presented method (right) on the crane descent dataset. Lateral images always have the same orientation. The color-space maps all values with a distance greater than 300 cm to the same color. Further, unlike in the previous section, points that have distances larger than 300 cm are excluded from the analysis. The top two columns show a heat map of distances, while the bottom shows the corresponding histogram. The color mapping is equivalent in both. For the birds-eye view, the ceiling has been cropped to get a better view.

	P90	P95	P98	Runtime
Uncorrected	621 cm	806.3 cm	1369.9 cm	0 min
SRR	159.7 cm	196.6 cm	240.6 cm	6069.04 min
presented method	150.0 cm	192.2 cm	250.7 cm	60.63 min

Table 5.4: Comparison of point-distances and runtimes after SRR and after the presented method, in an indoor real world dataset for the descending sphere. SRR used voxel based point cloud reduction, where any voxel of size 10 cm is allowed to only have one point. The presented method does not use reduction. The percentiles are to be read as follows: "For 90% of all points in SRR corrected dataset, the corresponding distance to their ground truth match is less than or equal to 159.7 cm."

Chapter 6

Conclusions

This thesis proposes a novel registration algorithm based on polygon matching and point-to-plane-based optimization, which corrects a 6 DoF path of a mobile mapping system to improve map quality. Further, the algorithm outputs a global planar model of the environment. The algorithm deals with a high amount of outliers, high noise levels, and coarse, IMU-based pose measurements. It is therefore perfectly suited for the application of mobile mapping of manmade environments with 6 DoF, in particular, spherical robot systems.

Another important contribution is the region growing based point cloud segmentation algorithm, which is also used by the registration procedure. The region growing algorithm also shows novelty utilizing the Bkd-tree [43] data structure - an adaption to the k-d tree which allows for dynamic updates without query performance degradation. The proposed algorithm recreates simulated as well as real world datasets well, using only IMU-based pose estimations.

In all examples, the corrected point clouds better resemble the structure of the environment. As the results show, the algorithm utilizes a robust clustering method and a reliable gradient descent procedure. The clustering algorithm is robust, since its not sensitive to outliers, noise, or uneven point density. Further, the gradient descent reliable, as it is fast and converges to a global minimum. Yet the results on two real world datasets also show that the polygon-to-polygon matching, using a score function, leaves room for improvement, as false correspondences arise in some situations. Other SOTA algorithms solve this problem in a more resiliant way, like [18] with random forest (RF) classifiers. However, their method needs some pre-labled training data and, once trained, behaves like an untunable black box. Furthermore, the presented matching procedure has to consider each point that the polygons are originally construced with. Currently, this is the main computational bottleneck of the proposed algorithm. Despite that bottleneck, its runtime is a huge advantage compared to other SOTA methods.

An extensive evaluation shows the huge potential of the presented algorithm, as it achieves nearly the same accuracy as a SOTA semi-rigid registration (SRR) [15] method in only a fraction of its runtime. For one dataset specifically, the new method needs only about an hour, whereas SRR needs more than four days, yet the precision that both methods achieve is almost indistinguishable. The presented method needs at least a coarse pose estimate, whereas SRR could do it without such. On the other hand, SRRs performance depends on the overlap between subsequent scans, which is not guaranteed for many mobile systems, especially spherical ones.

In other examples, the exported global clusters reveal difficulties with plane matching (cf. Section 5.2.2), especially for 2D line scanners, where overlap between subsequent scans is not always ensured. A potential proposed solution to the problem is the implementation of a loop closing technique, e.g., similar to [50]. To increase the autonomy of the system, further research shall reduce the number of input parameters by reworking parts of the algorithm, and automating a tuning method for optimal parameter estimates. Furthermore, an extensive comparison between the presented planar segmentation method must be done with other SOTA methods. A goal for future studies is to see if other SOTA algorithms benefit from the Bkd-tree method. Needless to say, a lot of work remains to be done. However, the presented algorithm brings 6 DoF SLAM to spherical robots, and the results imply relevance for many other types of robotic mobile mapping systems.

Bibliography

[1] H. Badino, D. Huber, Y. Park, and T. Kanade. Fast and accurate computation of surface normals from range images. In *2011 IEEE International Conference on Robotics and Automation*, pages 3084–3091, 2011.

[2] Jeffrey S. Beis and David G. Lowe. Shape indexing using approximate nearest-neighbour search in high-dimensional spaces. CVPR '97. IEEE Computer Society, 1997.

[3] P. J. Besl and N. D. McKay. A method for registration of 3-d shapes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14(2):239–256, 1992.

[4] P.J. Besl and R.C. Jain. Segmentation through variable-order surface fitting. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 10(2):167–192, 1988.

[5] B. Bhanu, S.K. Lee, C.C. Ho, and T.C. Henderson. Range data processing: Representation of surfaces by edges. Department of Computer Science, Univerity of Utah, 1985.

[6] Josep Biosca and José Lerma. Unsupervised robust planar segmentation of terrestrial laser scanner point clouds based on fuzzy clustering methods. *ISPRS Journal of Photogrammetry and Remote Sensing*, 63:84–98, 04 2008.

[7] Dorit Borrmann, Jan Elseberg, Kai Lingemann, Andreas Nüchter, and Joachim Hertzberg. Globally consistent 3D mapping with scan matching. *Robotics and Autonomous Systems*, 56:130–142, 02 2008.

[8] Dorit Borrmann, Sven Jörissen, and Andreas Nuchter. RADLER – A RADial LasER scanning device. In *Proceedings of the International Symposium on Experimental Research*, pages 655–664, Buenos Aires, Argentina, 01 2020.

[9] Michael Bosse, Robert Zlot, and Paul Flick. Zebedee: Design of a Spring-Mounted 3-D Range Sensor with Application to Mobile Mapping. *IEEE Transactions on Robotics*, 28(5):1104–1119, 2012.

[10] Boston Dynamics. Spot robot. `https://www.bostondynamics.com/spot`, 2021.

[11] Jie Chen and Baoquan Chen. Architectural modeling from sparsely scanned range data. *International Journal of Computer Vision*, 78:223–236, 07 2008.

[12] James Diebel. Representing attitude: Euler angles, unit quaternions, and rotation vectors. *Matrix*, 58(15-16):1–35, 2006.

[13] P. Dorninger and Clemens Nothegger. 3d segmentation of unstructured point clouds for building modelling. *International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, 36, 01 2007.

[14] David Droeschel and Sven Behnke. Efficient continuous-time slam for 3d lidar-based online mapping. *2018 IEEE International Conference on Robotics and Automation (ICRA)*, May 2018.

[15] Jan Elseberg, Dorit Borrmann, and Andreas Nüchter. 6DOF semi-rigid SLAM for mobile scanning. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1865–1870, 2012.

[16] Péter Fankhauser, Michael Bloesch, and Marco Hutter. Probabilistic Terrain Mapping for Mobile Robots With Uncertain Localization. *IEEE Robotics and Automation Letters*, 3(4):3019–3026, 05 2018.

[17] Ketty Favre, Muriel Pressigout, Eric Marchand, and Luce Morin. A Plane-based Approach for Indoor Point Clouds Registration. In *ICPR 2020 - 25th International Conference on Pattern Recognition*, Milan (Virtual), Italy, January 2021.

[18] Ketty Favre, Muriel Pressigout, Eric Marchand, and Luce Morin. Plane-based Accurate Registration of Real-world Point Clouds. In *SMC 2021 - IEEE International Conference on Systems, Man, and Cybernetics*, Melbourne / Virtual, Australia, October 2021.

[19] Pedro Felzenszwalb and Daniel Huttenlocher. Efficient graph-based image segmentation. *International Journal of Computer Vision*, 59:167–181, 09 2004.

[20] Sagi Filin and Norbert Pfeifer. Segmentation of airborne laser scanning data using a slope adaptive neighborhood. isprs journal of photogrammetry and remote sensing 60 (2). *ISPRS Journal of Photogrammetry and Remote Sensing*, 60:71–80, 04 2006.

[21] Jerome Friedman, Jon Bentley, and Raphael Finkel. An Algorithm for Finding Best Matches in Logarithmic Expected Time. *ACM Trans. Math. Softw.*, 3:209–226, 09 1977.

[22] W. Förstner and K. Khoshelham. Efficient and accurate registration of point clouds with plane to plane correspondences. In *2017 IEEE International Conference on Computer Vision Workshops (ICCVW)*, pages 2165–2173, 2017.

[23] Natasha Gelfand and Leonidas J. Guibas. Shape segmentation using local slippage analysis. In *Proceedings of the 2004 Eurographics, ACM SIGGRAPH Symposium on Geometry Processing*, SGP 04, page 214–223, New York, NY, USA, 2004. Association for Computing Machinery.

[24] Patrick Geneva, Kevin Eckenhoff, Yulin Yang, and Guoquan Huang. LIPS: LiDAR-Inertial 3D Plane SLAM. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS '18)*, pages 123–130, 2018.

[25] A. Golovinskiy and T. Funkhouser. Min-cut based segmentation of point clouds. In *2009 IEEE 12th International Conference on Computer Vision Workshops, ICCV Workshops*, pages 39–46, Los Alamitos, CA, USA, oct 2009. IEEE Computer Society.

[26] W. Shane Grant, Randolph C. Voorhies, and Laurent Itti. Efficient Velodyne SLAM with point and plane features. *Autonomous Robots*, 43:1207–1224, 2019.

[27] Jaehoon Jung, Sanghyun Yoon, Sungha Ju, and Joon Heo. Development of kinematic 3D laser scanning system for indoor mapping and as-built BIM using constrained SLAM. *Sensors*, 15(10):26430–26456, October 2015.

[28] R. Kindermann and J.L. Snell. *Markov Random Fields and Their Applications*. American Mathematical Society, 06 1980.

[29] Helge A. Lauterbach, Dorit Borrmann, Robin Heß, Daniel Eck, Klaus Schilling, and Andreas Nüchter. Evaluation of a Backpack-Mounted 3D Mobile Scanning System. *Remote Sensing*, 7(10):13753–13781, 2015.

[30] V. V. Lehtola, J.-P. Virtanen, M. T. Vaaja, H. Hyyppä, and A. Nüchter. Localization of a Mobile Laser Scanner via Dimensional Reduction. *ISPRS Journal of Photogrammetry and Remote Sensing (JPRS)*, 121:48–59, November 2016.

[31] Leica. Leica pegasus:backpack. `www.leica-geosystems.com/de/Leica-PegasusBackpack_106730.htm`, May 2015.

[32] Yangyan Li, Xiaokun Wu, Yiorgos Chrysanthou, Andrei Sharf, Daniel Cohen-Or, and Niloy J. Mitra. Globfit: Consistently fitting primitives by discovering global relations. *ACM Transactions on Graphics*, 30(4):52:1–52:12, 2011.

[33] Jiarong Lin and Fu Zhang. Loam livox: A fast, robust, high-precision LiDAR odometry and mapping package for LiDARs of small FoV. In *proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 3126–3131, 2020.

[34] David B. Lomet and Betty Salzberg. The hB-Tree: A Multiattribute Indexing Method with Good Guaranteed Performance. *ACM Trans. Database Syst.*, 15(4):625–658, December 1990.

[35] Sebastian Madgwick. An efficient orientation filter for inertial and inertial/magnetic sensor arrays. *Report x-io and University of Bristol (UK)*, 25:113–118, 2010.

[36] NavVis. The NavVis VLX mobile mapping system. `https://www.navvis.com`. Accessed on 05.07.2021.

[37] Anh Nguyen and Bac Le. 3d point cloud segmentation: A survey. In *2013 6th IEEE Conference on Robotics, Automation and Mechatronics (RAM)*, pages 225–230, 2013.

[38] Xiaojuan Ning, Xiaopeng Zhang, Yinghui Wang, and Marc Jaeger. Segmentation of architecture shape information from 3d point cloud. 12 2009.

[39] Andreas Nuchter, Kai Lingemann, Joachim Hertzberg, and Hartmut Surmann. 6D SLAM - 3D mapping outdoor environments. *Fraunhofer IAIS*, 24, 11 2006.

[40] Andreas Nüchter, Johannes Schauer, and Dorit Borrmann. Technical report: Reduction and compression using octrees — 3DTK's entry to the ICIP 2019 challenge on point cloud coding. 2019.

[41] Nüchter, Andreas. Lecture notes in 3D Point Cloud Processing, July 2014.

[42] Kaustubh Pathak, Andreas Birk, Narunas Vaskevicius, and Jann Poppinga. Fast Registration Based on Noisy Planeswith Unknown Correspondences for 3D Mapping. *IEEE Transactions on Robotics*, 26(3):424–441, 2010.

[43] Octavian Procopiuc, Pankaj K. Agarwal, Lars Arge, and Jeffrey Scott Vitter. Bkd-tree: A dynamic scalable kd-tree. In Thanasis Hadzilacos, Yannis Manolopoulos, John Roddick, and Yannis Theodoridis, editors, *Advances in Spatial and Temporal Databases*, pages 46–65, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.

[44] John T. Robinson. The K-D-B-Tree: A Search Structure for Large Multidimensional Dynamic Indexes. In *Proceedings of the 1981 ACM SIGMOD International Conference on Management of Data*, SIGMOD '81, page 10–18, New York, NY, USA, 1981. Association for Computing Machinery.

[45] Angelo Pio Rossi, Francesco Maurelli, Vikram Unnithan, Hendrik Dreger, Kedus Mathewos, Nayan Pradhan, Dan-Andrei Corbeanu, Riccardo Pozzobon, Matteo Massironi, Sabrina Ferrari, Claudia Pernechele, Lorenzo Paoletti, Emanuele Simioni, Pajola Maurizio, Tommaso Santagata, Dorit Borrmann, Andreas Nüchter, Anton Bredenbeck, Jasper Zevering, Fabian Arzberger, and Camilo Andres Reyes Mantilla. DAEDALUS - Descent And Exploration in Deep Autonomy of Lava Underground Structures. Technical Report 21, Institut für Informatik, 2021.

[46] Radu Rusu, Andreas Holzbach, Nico Blodow, and Michael Beetz. Fast geometric point labeling using conditional random fields. pages 7–12, 12 2009.

[47] Radu Bogdan Rusu, Nico Blodow, and Michael Beetz. Fast point feature histograms (fpfh) for 3d registration. In *2009 IEEE International Conference on Robotics and Automation*, pages 3212–3217, 2009.

[48] A.D. Sappa and M. Devy. Fast range image segmentation by an edge detection strategy. In *Proceedings Third International Conference on 3-D Digital Imaging and Modeling*, pages 292–299, 2001.

[49] Jonathan R. Schoenberg, Aaron Nathan, and M. Campbell. Segmentation of dense range information in complex urban scenes. *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2033–2038, 2010.

[50] Jochen Sprickerhof, Andreas Nuchter, Kai Lingemann, and Joachim Hertzberg. An explicit loop closing technique for 6d SLAM. pages 229–234, 01 2009.

[51] Johannes Strom, Andrew Richardson, and Edwin Olson. Graph-based segmentation for colored 3d laser point clouds. In *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2131–2136, 2010.

[52] Daniel Sunday. *Practical Geometry Algorithms - With C++ Code.* Amazon Digital Services LLC - KDP Print US, Fishers, Indiana 46038, USA, 2021.

[53] Fayez Tarsha-Kurdi. Hough-transform and extended ransac algorithms for automatic detection of 3d building roof planes from lidar data. 09 2007.

[54] Aparna Tatavarti, John Papadakis, and Andrew Willis. Towards real-time segmentation of 3d point cloud data into local planar regions. pages 1–6, 03 2017.

[55] Xin Wei, Jixin Lv, Jie Sun, and Shiliang Pu. Ground-SLAM: Ground Constrained LiDAR SLAM for Structured Multi-Floor Environments, 2021.

[56] Matthew D Zeiler. Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*, 2012.

[57] Jasper Zevering, Anton Bredenbeck, Fabian Arzberger, Dorit Borrmann, and Andreas Nüchter. IMU-based pose-estimation for spherical robots with limited resources. In *2021 IEEE International Conference on Multisensor Fusion and Integration for Intelligent Systems (MFI)*, 2021. [Manuscript accepted for publication].

[58] Ji Zhang and Sanjiv Singh. Loam: Lidar odometry and mapping in real-time. In *Robotics: Science and Systems Conference (RSS)*, 07 2014.

[59] L. Zhou, S. Wang, and M. Kaess. π-LSAM: LiDAR smoothing and mapping with planes. In *Proceedings of IEEE International Conference on Robotics and Automation (ICRA '21)*, Xi'an, China, May 2021. To appear.

Proclamation

Hereby I confirm that I wrote this thesis independently and that I have not made use of any other resources or means than those indicated.

Würzburg, September 2021