Institute for Computer Science XVII
Robotics

Master's thesis

Real-Time Calculation of the Ground Plane's Normal Vector from the LiDAR Point Clouds of a Spherical Mobile Mapping System

Carolin Bösch

September 2024

First reviewer:	Prof. Dr. Andreas Nüchter
Second reviewer:	Prof. Dr.-Ing. Sergio Montenegro
Advisor:	M.Sc. Fabian Arzberger

Zusammenfassung

In den letzten Jahren sind autonome Systeme für die Durchführung gefährlicher, sich wiederholender oder zeitaufwändiger Aufgaben wie Erkundung, Überwachung und Präzisionslandwirtschaft unverzichtbar geworden. Diese Systeme sind in hohem Maße von einer genauen Umgebungswahrnehmung und Echtzeitdateninterpretation abhängig, insbesondere von Light Detection and Ranging (LiDAR)-Sensoren. Die Segmentierung der Bodenoberfläche aus 3D-Punktwolken ist für Anwendungen wie autonomes Fahren, Robotik und Stadtplanung von entscheidender Bedeutung. Dabei geht es darum, die Bodenoberfläche von unübersichtlichen Punktwolken zu unterscheiden, was für die Navigation, die Hinderniserkennung und die Umgebungsmodellierung unerlässlich ist. Bei kugelförmigen Robotern ist die Erkennung der Bodenfläche aufgrund der schwachen Lasereinfallswinkel, die durch die sich ständig ändernde Ausrichtung des LiDAR verursacht werden, besonders schwierig. Aggressive Bewegungsdynamik und hohe Rotationsgeschwindigkeiten führen zusätzlich zu Rauschen und Verzerrungen, was eine genaue Segmentierung der Bodenebene erschwert. In dieser Arbeit implementieren und bewerten wir mehrere Ansätze für die Echtzeitberechnung des Normalenvektors der Bodenebene aus den Light Detection and Ranging (LiDAR)-Punktwolken eines sphärischen mobilen Kartierungssystems. Insbesondere vergleichen wir eine k-dimensional tree (kd-tree)-basierte Methode mit einem geometrischen Ansatz für die Vorverarbeitung und evaluieren vier hochmoderne Algorithmen zur Segmentierung von Ebenen unter Verwendung der Point Cloud Library (pcl). Der Code ist als Open-Source auf GitLab [1] verfügbar. Unser Ziel ist es, geeignete Ansätze für kugelförmige mobile LiDAR-inertiale Kartierungssysteme zu identifizieren. Wir führen acht Offline-Experimente in verschiedenen Szenarien durch, z. B. bei unterschiedlichen Neigungen und Geschwindigkeiten, sowie Echtzeittests an Bord. Die Ergebnisse zeigen, dass der geometrische Teilwolkenansatz die kd-tree-Methode durchweg übertrifft, da er eine schnellere Verarbeitung und verbesserte Robustheit bietet. Darüber hinaus erweisen sich Ansätze ohne Filterung als ungeeignet, da sie im Vergleich zu Ansätzen mit Filterung hohe Fehlerquoten und weniger genaue Ergebnisse aufweisen. Unter den Algorithmen zur Segmentierung von Ebenen bieten die Methoden zur Ebenenanpassung (Least Squares Fitting (LSF) und Principal Component Analysis (PCA)) im Allgemeinen schnellere Verarbeitungszeiten als die Algorithmen zur Erkennung von Ebenen (Random Sample Consensus (RANSAC) und Randomized Hough Transform (RHT)), weisen jedoch häufig höhere Fehlerquoten bei der Genauigkeit auf. RHT2, das RHT mit der Ebenenanpassung kombiniert, ist deutlich ungenauer als RANSAC und RHT, mit Genauigkeitseinbußen von 45 % bzw. 27 % für kd-tree und geometrische Teilwolken. RANSAC ist für geometrische Teilwolken am genauesten, während RHT für kd-tree-Teilwolken mit nur einem geringen Unterschied im mittleren absoluten Fehler von 0,3 ° übertrifft. RANSAC erweist sich bei Verwendung der geometrischen Filterung und des Teilwolkenansatzes als die effektivste Methode, die ein Gleichgewicht zwischen Geschwindigkeit, Genauigkeit und Robustheit bietet. Trotz ihrer Vorteile stößt diese Methode unter extremen Bedingungen, wie etwa in einem Tunnel, an ihre Grenzen. Die beobachtete Leistungsverschlechterung von RANSAC in Tunnelumgebungen ist ein erwartetes Ergebnis, das in erster Linie auf die Annahme flacher Ebenen im Modell von RANSAC zurückzuführen ist.

Abstract

In recent years, autonomous systems have become essential for performing tasks that are hazardous, repetitive, or time-consuming, such as exploration, surveillance, and precision agriculture. These systems depend heavily on accurate environmental perception and real-time data interpretation, particularly from Light Detection and Ranging (LiDAR) sensors. Ground plane segmentation from 3D LiDAR point clouds is crucial for applications including autonomous driving, robotics, and urban planning. It involves distinguishing the ground surface from cluttered point clouds, which is vital for navigation, obstacle detection, and environment modeling. For spherical robots, ground plane detection is particularly challenging due to weak laser incidence angles caused by the LiDAR's constantly changing orientation. Aggressive locomotion dynamics and high rotation speeds further introduce noise and distortions, complicating accurate ground plane segmentation. In this thesis, we implement and evaluate multiple approaches for the on-board real-time calculation of the ground plane's normal vector from the LiDAR point clouds of a spherical mobile mapping system. Specifically, we compare a k-dimensional tree (kd-tree)-based method with a geometrical approach for pre-processing and evaluate four state-of-the-art plane segmentation algorithms using the Point Cloud Library (pcl). The code is available open-source on GitLab [1]. Our goal is to identify suitable approaches for spherical mobile mapping LiDAR-inertial systems.We perform eight offline experiments across diverse scenarios, such as different inclinations and speeds, and real-time on-board tests. From the results, we find that the geometrical sub-cloud approach consistently outperforms the kd-tree method, providing faster processing and improved robustness. Additionally, approaches without filtering prove unsuitable due to their high fail rates and less accurate results compared to approaches with filtering. Among the plane segmentation algorithms, the plane fitting methods (Least Squares Fitting (LSF) and Principal Component Analysis (PCA)) generally offer faster processing times compared to the plane detection algorithms (Random Sample Consensus (RANSAC) and Randomized Hough Transform (RHT)), but they often show higher error rates in accuracy. RHT2, which combines RHT with plane fitting, is significantly less accurate than RANSAC and RHT, with accuracy reductions of 45 % and 27 % for geometrical and kd-tree sub-clouds, respectively. RANSAC is the most accurate for geometrical sub-clouds, while RHT excels for kd-tree sub-clouds, with only a minor difference in median absolute error of 0.3 °. RANSAC, when used with geometrical filtering and sub-cloud approach, proves to be the most effective method, offering a balance of speed, accuracy, and robustness. Despite its advantages, this method shows limitations in extreme conditions, such as within a tunnel. The observed performance degradation of RANSAC in tunnel environments is an expected result, which arises primarily due to the assumption of flat planes inherent in the RANSAC model.

Danksagung

An dieser Stelle möchte ich all den Personen danken, die durch fachliche und emotionale Unterstützung an dieser Arbeit mitgewirkt haben. Ein großes Dankeschön gilt zunächst meinem Betreuer Fabian Arzberger, der mir während der gesamten Arbeit mit wertvollen Ratschlägen und konstruktivem Feedback zur Seite stand. Ich möchte mich auch dafür bedanken, dass er stets Zeit für meine Fragen gefunden hat und mir auch das ein oder andere Mal geholfen hat, das Problem meines fehlenden LAN-Anschlusses zu umgehen. Ebenso gilt Prof. Andreas Nüchter ein Dankeschön, ohne den es gar nicht möglich gewesen wäre, meinen straffen Zeitplan einzuhalten und für die motivierenden Puddings, die er zur Verfügung gestellt hat. Nicht zu vergessen sind natürlich meine Familie und Freunde, die mich während der gesamten Arbeit emotional unterstützt und motiviert haben. Ein besonderes Dankeschön gilt an dieser Stelle Pascal, der mich in so vielen Aspekten tatkräftig unterstützt hat, angefangen bei der emotionalen Versorgung bis hin zum Korrekturlesen. Zuletzt möchte ich mich bei Martin bedanken, der mich schon seit dem Bachelor begleitet, immer eine Antwort auf meine Fragen hat und meine Studium in Würzburg besonders geprägt hat.

Contents

Acronyms

3DTK 3D Toolkit.

DAEDALUS Descent And Exploration in Deep Autonomy of Lava Underground Structures.

FLANN Fast Library for Approximate Nearest Neighbors.

FOV field of view.

HT Hough Transform.

ikd-tree incremental k-d tree.

kd-tree k-dimensional tree.

kNN k-Nearest Neighbors.

LiDAR Light Detection and Ranging.

LSF Least Squares Fitting.

MAE Mean Absolute Error.

MedAE Median Absolute Error.

NNS Nearest Neighbor Search.

PCA Principal Component Analysis.

pcl Point Cloud Library.

RANSAC Random Sample Consensus.

RHT Randomized Hough Transform.

ROS Robot Operating System.

SHT Standard Hough Transform.

SLAM Simultaneous Localization and Mapping.

List of Figures

List of Tables

Chapter 1

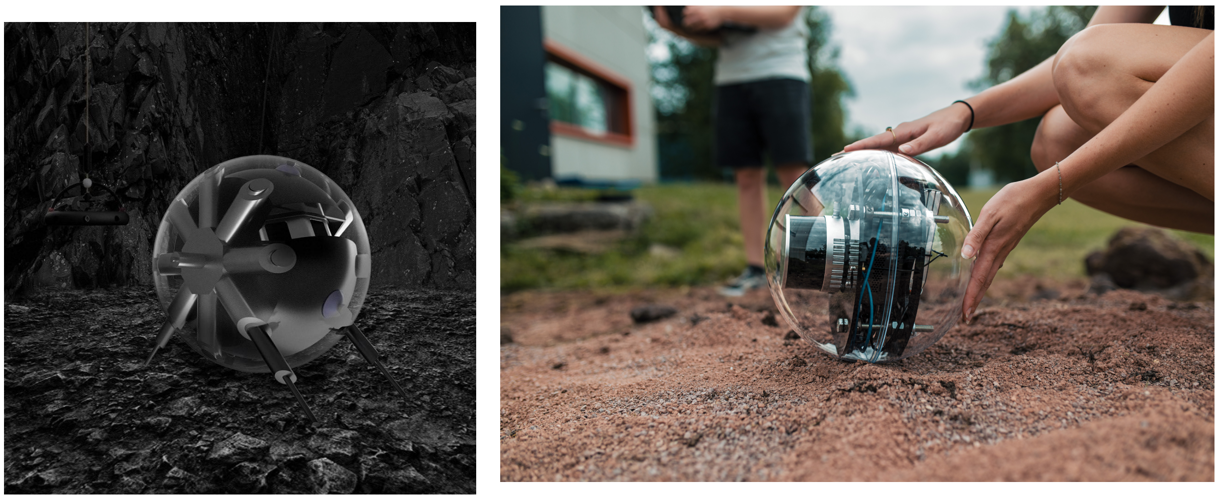
Introduction

In recent years, the development of machines capable of performing tasks without human intervention, also known as autonomous systems, has increased. These systems are crucial for tasks that are either too dangerous, too repetitive or too time-consuming for humans, such as exploration of remote or dangerous areas, e.g., space, surveillance or precision agriculture, to name a few examples. Autonomous systems, including mobile robots, rely on a combination of sensors and algorithms to operate, navigate and make decisions in real time. Their successful deployment relies on accurate perception and understanding of the environment, which increases the need for precise environmental knowledge and the ability to accurately interpret complex data from sensors, such as LiDAR sensors. This thesis explores the task of real-time ground plane segmentation, specifically the calculation of the ground plane's normal vector, from 3D LiDAR point clouds captured by a spherical mobile mapping system. Ground plane segmentation involves identifying and isolating the ground surface from the point cloud data. Accurately calculating the normal vector of the ground plane plays an important role in the pose estimation of the Descent And Exploration in Deep Autonomy of Lava Underground Structures (DAEDALUS) [2] spherical robot.

1.1 Background

The DAEDALUS project [2], developed by the Julius-Maximilians-University of Würzburg, serves as the foundation for this work. In 2019, the European Space Agency (ESA) launched the Open Space Innovation Platform (OSIP) to solicit innovative ideas for lunar cave exploration [3]. Among the numerous proposals, DAEDALUS was selected as one of the five standout concepts chosen for detailed study. This study focuses on the exploration and characterization of lunar lava tubes, subterranean structures formed by ancient volcanic activity on the Moon. These lava tubes, accessible through surface openings known as skylights, offer a unique opportunity to advance our understanding of lunar geology and assess their potential as sites for future lunar bases.

The DAEDALUS mission proposes the deployment of a compact, spherical robot designed to explore these underground environments. The robot, enclosed within a durable spherical shell, is equipped with an array of sensors, including LiDAR and panoramic cameras, to create



(a) Animation of the spherical robot operating within a lunar cave after its descent [2].

(b) Image of the current prototype (spherical mobile mapping system) on the robotics test site (Martian environment) at the University of Würzburg.

Figure 1.1: Daedalus spherical robot.

detailed 3D maps of the lava tubes. The mission is structured in multiple phases: initially, the robot is attached to a crane that lowers it into the cave, providing power and communication via a tether. During the descent, the robot rotates to scan its surroundings, generating a comprehensive point cloud of the cave's entrance and initial interior. Upon reaching the cave floor, the robot detaches from the crane. It then operates autonomously to explore the lunar caves, navigating the challenging terrain. The spherical design not only facilitates movement but also protects the sensitive internal components from the harsh lunar environment, including abrasive lunar dust that could otherwise damage sensors and actuators. [2]

1.2 Problem Definition and Scientific Contribution

In general, ground extraction from 3D LiDAR point clouds is a crucial step in various applications such as autonomous driving, robotics, and urban planning. This process involves identifying and segmenting the ground surface from the point cloud data, which is often cluttered with objects and terrain features. Ground extraction plays a fundamental role in scene understanding and pose estimation tasks, providing essential information for navigation and path planning, obstacle detection, and environment modeling. Through advanced algorithms and techniques, ground segmentation enables the creation of accurate and reliable maps of the environment.

In the context of the DAEDALUS mission, the spherical mobile robot relies on precise real-time ground plane segmentation for pose estimation. It requires accurate information about the ground plane's normal vector to feed into its Kalman filter [4], which is used to enhance its pose estimation. However, the design of the spherical robot introduces specific challenges for ground plane segmentation. The LiDAR sensor rotates with the sphere, which leads to variations in the detection and perceived size of the ground. Some scans provide an expansive view of the ground, while others provide more limited views of the ground, especially when the LiDAR sensor is in a vertical position. Additionally, the LiDAR's field of view (FOV) is affected not only by its positioning and rotation, but also by environmental factors. Particularly tight or complex environments may restrict visibility and introduce challenges to capturing accurate ground data. In addition, noise from the robot's shell may interfere with the LiDAR data, making the segmentation process more difficult. Additionally, motion dynamics, such as sudden accelerations, introduce motion blur in the LiDAR data, which distorts the point cloud and complicates ground plane extraction.

Therefore, the implementation and verification of an algorithm that calculates the normal vector of the ground plane in real-time is essential. In this work, we implement and evaluate multiple approaches for this task. Specifically, we compare a k-dimensional tree (kd-tree)-based method with a geometric approach for pre-processing and four state-of-the-art plane segmentation algorithms, across various scenarios that highlight key aspects such as limited area, varying inclinations, and different movement speeds. The goal is to identify suitable approaches for spherical mobile mapping LiDAR-inertial systems, ensuring accurate and robust real-time ground plane segmentation.

The scientific contribution of this work lies in the comprehensive evaluation of ground surface segmentation approaches under the specific conditions of the DAEDALUS mission. By systematically evaluating various state-of-the-art plane segmentation algorithms under different pre-processing approaches, this research provides valuable insights into the performance of the different methods in practical scenarios. This evaluation identifies the most effective techniques for the DAEDALUS prototype and sets a benchmark for future developments in LiDAR-based ground plane segmentation. The tested approaches, designed for the unique context of a spherical robot, offer a broader relevance and can be applied to other mobile robots with fewer constraints, such as mobile robots with a consistent LiDAR FOV. Additionally, the focus on real-time processing makes these results transferable to other autonomous systems that require fast and accurate ground surface detection.

1.3 Thesis Outline

This work is structured to systematically address the problem of ground plane segmentation from LiDAR point clouds of a spherical mobile mapping system. We begin by reviewing related research in Chapter 2, providing a comprehensive overview of existing approaches and methodologies relevant to ground plane segmentation and point cloud processing. Following this, we introduce the theoretical fundamentals in Chapter 3, where we outline the essential concepts and algorithms that form the foundation of our approach. In the subsequent chapter, Chapter 4, we present our approach in detail, outlining the specific techniques and calculations we employ to tackle the identified problem. We then move on to Chapter 5, where we introduce the experiments conducted and evaluate our approaches, highlighting their effectiveness and performance under various conditions. Finally, in Chapter 6, we conclude with a discussion of our findings, the effectiveness of our solutions for the DAEDALUS prototype, and potential directions for future research.

Chapter 2

Related Work

In this chapter, we first present previous research on Nearest Neighbor Search (NNS) including suitable multidimensional data structures. We use NNS in this work to identify points that likely belong to the ground plane. Subsequently, we review work on plane segmentation from 3D point clouds, covering both general approaches and specific methods for plane matching and recognition. Finally, we introduce existing research on ground detection in LiDAR point clouds and explain how our work integrates the established research in point cloud processing and ground segmentation.

2.1 Nearest Neighbors Search

NNS involves finding the closest points to a given query point based on their spatial distance in the point cloud dataset. According to Elseberg et al. [5] spatial data structures are commonly utilized to organize and index the point cloud data to efficiently perform NNS. Their hierarchical nature enables efficient access to stored elements via positional queries.

Suitable Data Structures Suitable data structures include the kd-tree [6], octree [7], R-tree [8], range-tree [9] and the VP-tree [10]. Nakamura et al. [11] present a new data structure with improved range-search: the MD-tree. Although the MD-tree has improved efficiency of the storage utilization compared to the kd-tree, it takes longer to create an MD-tree. Compared to the R-tree, it has lower storage utilization and is constructed twice as fast. However, kd-trees remain popular in NNS libraries not just for their efficiency, but also for their simplicity and low overhead, which make them ideal for resource-limited, real-time systems where input data is often downsampled [12]. Further, to Elseberg et al.'s knowledge [5], no publicly available NNS libraries exist for the range- and the VP-tree. Vermeulen et al. [13] further states that the kd-tree's performance has the lowest variance of all structures. Adding new points to and removing old points from a kd-tree disrupts the tree's balance, which is only restored by rebuilding the tree [5, 13]. To address the challenges associated with maintaining balance in traditional kd-tree during updates, Cai et al. [14, 15] introduced the incremental k-d tree (ikd-tree) for robotic applications. In addition to point-wise operations, the ikd-tree offers practical features like box-wise operations and down-sampling. It actively monitors its structure during incremental operations such as insertion, reinsertion and deletion and partially rebalances the tree to enable an efficient NNS in later phases. With n points, the ikd-tree achieves a time complexity of $\mathcal{O}(log\, n)$ for insertion and on-tree downsampling operations, and single-thread re-building with a complexity of $\mathcal{O}(n\, log\, n)$. The process of building an ikd-tree tree closely resembles that of building a static kd-tree, albeit with additional operations to handle incremental updates.

Types of Nearest Neighbor Search Queries Beyond the choice of data structure, the different types of NNS queries are of interest. There are three types: k-NNS, fixed radius search, and ranged k-NNS. The k-NNS retrieves the k-Nearest Neighbors (kNN) around a specified query point, while the fixed radius search computes all points within a given radius of the query point. The ranged k-NNS is the combination of both, where it retrieves the kNN within a given distance. According to Elseberg et al. [5] one factor which influences the performance of NNS is the maximal distance parameter. Larger distances impact fixed radius search algorithms more significantly compared to ranged k-NNS algorithms. As the runtime for fixed radius search increases polynomially in $\mathcal{O}(n^{\frac{2}{3}} + k)$, the runtime of ranged searches plateau near the maximum possible distance in the dataset. In opposition, the runtime of the k-NNS is not influenced by that distance.

Comparative Analysis of Nearest Neighbor Search Libraries In Elseberg et al.'s [5] experiments, they compare multiple NNS libraries including 3D Toolkit (3DTK) [16], ANN [17], Fast Library for Approximate Nearest Neighbors (FLANN) [18], libnabo [19], and others. Their results, regarding the runtime on an artificial dataset, show that the libraries employing ranged search (3DTK and libnabo) perform best. However, at a point where the maximal distance is at roughly 1/3 to 1/2 of the point cloud's size, the ANN and FLANN libraries (k-NNS) perform similar to 3DTK and libnabo. For the real dataset, the ranged search also outperforms the other methods. For the smaller ones the k-NNS are able to compete with the ranged search, but for large datasets it performs significantly worse. In contrast, fixed radius search performs similar on larger datasets but less effectively on smaller ones than the ranged search. In addition, the 3DTK library seems to have an advantage with the real datasets, as it does not require any conversion of the query dataset compared to libnabo. Similar research by Vermeulen et al. [13] compares several data structures for finding the kNN in the context of crowd simulation. Based on their results, they conclude that kd-trees are particularly consistent in their performance, as the variance in performance for both implementations (FLANN and nanoflann) is much smaller than for other data structures. Regrading the update and query time, nanoflann has the best performance, followed by the FLANN kd-tree implementation. The slightly superior performance regarding speed and memory usage of nanoflann compared to FLANN is also stated in [20]. The work of Greenspan et al. [21] propose a new solution method for the NNS problem. And although this provides promising results for small point sets, they have not yet been incorporated into any of the libraries mentioned above. In [15] Xu et al. compare the performance of the ikd-tree with the boost geometry library implementation of the R*-tree [22], the Point Cloud Library (pcl) implementation of the octree [23] and the nanoflann [20] implementation of the dynamic kd-tree. They select these three implementations for their high implementation efficiency, dynamic operation support, and range search capability, which are essential for a fair comparison with ikd-tree. The ikd-tree achieves the lowest average k-NNS time for all tested

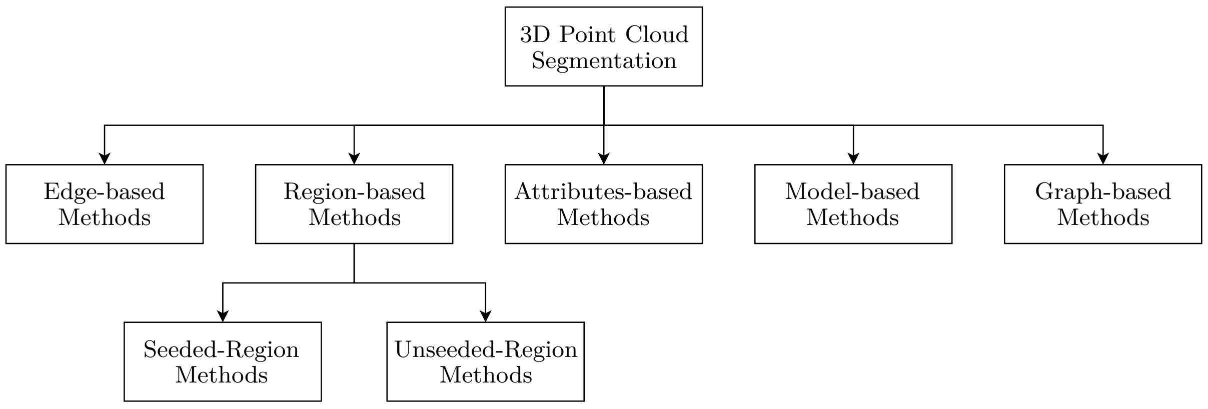


Figure 2.1: Taxonomy of 3D point cloud segmentation methods according to [25].

scenarios. The nanoflann and R*-tree implementations consumes slightly more time, especially when the tree size becomes larger. The octree has by far the highest average k-NNS time. This is also supported by Adamsson and Vorkapic [24], who compare the kd-tree, octree and VP-tree for storing neural morphology data with respect to performance. Their results also indicate that the octree has the highest average search time, as well as the highest memory requirements.

2.2 Plane Segmentation from 3D Point Clouds

Clustering 3D point clouds based on specific geometric characteristics is a widely explored research area. Nguyen and Le [25] summarize different methods proposed for the segmentation of 3D point clouds. They categorize them into five general groups as displayed in Figure 2.1.

Edge-based Methods Edge-based methods detect boundaries between different regions within the point cloud by identifying rapid changes in intensity. These rapid changes in intensity represent the object's edges, which describe the shape of the object. There are multiple techniques for detecting edges, a gradient-based technique [26], a scan line grouping technique [27], and a technique based on binary edge maps [28]. Edge-based methods enable fast segmentation but are susceptible to accuracy problems due to their high sensitivity to noise and uneven point cloud density, which are common in real point clouds.

Region-based Methods Region-based methods use neighborhood information to identify isolated regions and evaluate the dissimilarity between them. They have a higher noise resistance than edge-based methods, but face the challenge of precisely defining region boundaries. There are two types of region-based methods.

A) Seeded-region (or bottom-up/region-growing) methods begin by selecting seed points, from which regions expand by adding neighboring points meeting specific criteria. This method was initially introduced by Besl and Jain [29], who identify seed points based on point curvature and grow them based on proximity and surface planarity. Subsequent improvements include storing relative information between regions (Köster and Spann [30]), adding smoothness constraints (Rusu et al. [31]), and reduction of time complexity (Dorninger and Nothegger [32]). Despite these improvements, seeded-region approaches remain sensitive to noise, time-consuming, and the choice of the initial seed points. Inaccurate starting seed points lead to under- or over-segmentation.

B) Unseeded-region (or top-down) methods first group all points into one region and then divide them into smaller regions until a selected figure of merit for fitting exceeds a threshold. Chen and Chen [33] use this method to cluster planar regions for reconstructing buildings based on the confidence that an area is planar. The major issues of unseeded-region methods are determining where and how to subdivide, and the necessity for a large amount of prior knowledge, which is in general not given for complex scenes.

Attributes-based methods Attributes-based methods cluster the point cloud by first computing the attributes and then grouping the point clouds according to the attributes. Biosca and Lerma [34] introduce a new segmentation strategy using unsupervised clustering and fuzzy algorithms for combining multiple attributes. Although the results are promising, this strategy relies heavily on parameter selection and is time-consuming. Filin and Pfeifer [35] segment airborne LiDAR data by using the normal and height differences between neighboring points. They apply a slope adaptive neighborhood system based on key parameters of the LiDAR data such as point density, measurement accuracy, and point distribution. This results in a robust clustering technique resilient to outliers, noise, and uneven point density. In general, attributes-based approaches are robust and deliver accurate results. However, they are limited by the point density and the need to define point neighborhoods, while also facing time constraints when handling multidimensional attributes for large datasets.

Model-based methods Model-based methods utilize geometric primitive shapes, such as spheres, cones, planes, and cylinders, and their mathematical representation to group points. Two popular approaches are the 3D Hough Transform (HT) and the Random Sample Consensus (RANSAC) algorithm. We provide more detailed research about these algorithms in Section 2.2.2. Schnabel et al. [36] introduced an algorithm utilizing RANSAC for segmenting mesh and point cloud data, capable of automatically detecting basic shapes in unorganized point clouds. To extend the limitations for primitive shapes in point cloud segmentation Gelfand and Guibas [37] propose a method for recognizing slippable shapes such as spheres, helices, planes, cylinders, etc. Model-based methods rely on mathematical principles, offering speed and robustness against outliers, but they are less accurate when handling various sources of point cloud data.

Graph-based methods Graph-based methods represent the point clouds in the form of a graph and then employ algorithms from graph theory for clustering. Golovinskiy and Funkhouser [38] utilize kNN to create a 3D graph of the point cloud to compute a foreground-background segmentation. That approach requires prior knowledge of the location of the objects to be segmented. Strom et al. [39] extend the graph-based method for segmenting colored 3D LiDAR point clouds, proposing a segment union criterion based on color and surface normals. The results demonstrate success in segmenting both indoor and outdoor scenes, running in real-time, and exhibiting robustness compared to segmenting LiDAR or color data alone. Several approaches employ probabilistic inference models, e.g. the work by Rusu et al. [40]

for labeling points with various geometric surface primitives using Conditional Random Fields. Another example is the work by Schoenberg et al. [41] which fuses and segments the data from an optical camera and a LiDAR scanner by applying Markov Random Field to estimate 3D points corresponding to each image pixel. In general, graph-based methods perform better at segmentation than the previously presented method types for point clouds with noise or uneven density. However, they are not real-time compatible.

In a more general view, we differentiate between plane fitting and plane detection. Plane fitting describes the process of finding the best-fitting plane model for a given set of points. Whereas, plane detection focuses on identifying regions within the point cloud that are likely to represent planar surfaces [25]. The most popular techniques for extracting planes from 3D point clouds are Least Squares Fitting (LSF), Principal Component Analysis (PCA), 3D HT and the RANSAC algorithm [25, 42–45]. The following subsections provide a more detailed overview of the research background regarding these specific four algorithms. For an explicit description, refer to Section 3.3. As we aim to extract the ground plane from 3D point clouds in real-time in this work, the focus of this background research lies on the performance and speed of plane segmentation algorithms.

2.2.1 Plane Fitting

LSF and PCA are the two most applied plane fitting methods [42]. LSF [46] is a method that iteratively determines the optimal plane fit while adhering to the least-squares criterion for the distances between scanned points and the plane. Whereas PCA [47] determines the optimal plane parameters by analyzing the covariance matrix of the point cloud. It calculates the eigenvalues and eigenvectors of this matrix, where the eigenvector corresponding to the smallest eigenvalue represents the normal vector of the plane, thereby defining the plane that best fits the points with minimal variance. For a more detailed explanation refer to Section 3.3.1 and 3.3.2. In general, the plane fitting approach is characterized by its fast runtime and achieves good results in simple scenarios [25]. However, LSF and PCA are known to be sensitive to noise and their estimation is impaired by outliers, which possibly lead to inconsistent and misleading estimates [25, 42, 43]. In order to mitigate the effects of outliers or noise on the PCA estimates, robust variants of PCA were developed [45, 48–57]. Vaswani and Narayanamurthy [58] provide a comprehensive overview of the literature on robust PCA. The robust version consistently delivers superior performance but at the cost of runtime [45, 48, 50, 54, 56–58].

In a comparison of surface normal estimation methods by Klasing et al. [59] LSF and PCA outperform the other methods in terms of both quality and runtime for plane fitting and normal estimation on a kNN graph. In a direct comparison, their results show that PCA has a minimal smaller computation time than LSF. This finding is also supported by Huang and Tseng [42], who compare these two plane fitting algorithms in terms of implementation, computation time, and robust estimation. Their results also indicate that the robust versions of LSF and PCA have a very similar standard deviation for the point-to-point error. Klasing et al. [59] further indicate that PCA has the best quality and speed performance for medium-sized neighborhoods when applied to actual range data. Therefore, they declare PCA as the universal method of choice. Further research by Nurunnabi et al. [45] also shows only a little difference between the

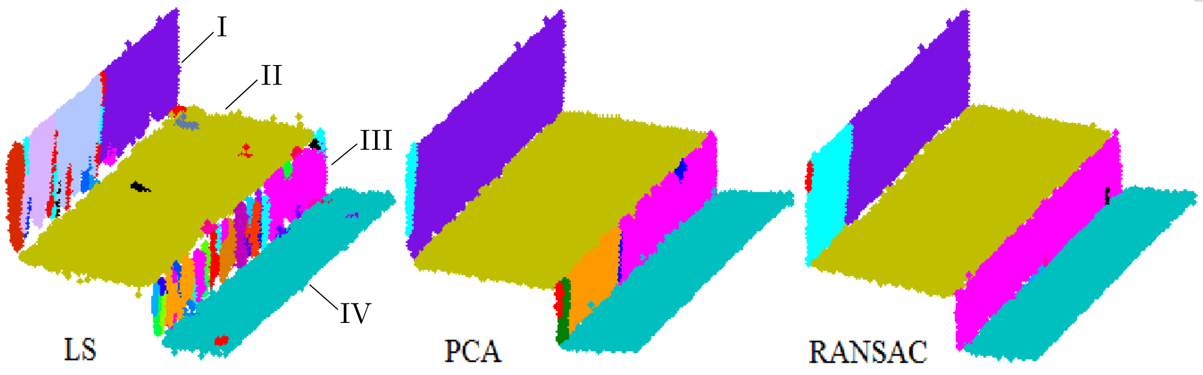


Figure 2.2: Comparison of LSF, PCA and RANSAC performance in plane segmentation from [45].

resulting bias angles from LSF and PCA based on artificial data. Further, their results indicate that PCA is more consistent than LSF and performs better at segmenting planes from real-life data point clouds, as we see from Figure 2.2. However, in contradiction with the runtime results from the other two papers, here LSF achieves minimal smaller computation times than PCA.

2.2.2 Plane Detection

The most applied plane detection algorithms are the 3D HT and RANSAC [60]. The 3D HT [61, 62] converts the point cloud into a parameter space, in which it identifies planes by finding local maxima. These planes are then extracted from the original point cloud data. The RANSAC algorithm [63] iteratively fits models to subsets of data points, reliably estimating parameters by selecting the model with the best fit. Its specific design enables it to cope with large numbers of outliers in the input data. For a more detailed explanation refer to Section 3.3.3 and 3.3.4. In general plane detection yields better results than plane fitting in terms of accuracy but at the cost of runtime [25]. This is proven by e.g. Nurunnabi et al. [45]. They compare their robust PCA against LSF, the classic PCA, and the RANSAC algorithm. Their research shows the large difference in the bias angle evaluation based on an artificial dataset between RANSAC and the two plane fitting algorithms. Further, their results show that RANSAC has considerably higher mean computation times as LSF or PCA by roughly three orders of magnitude. In the direct comparison regarding plane segmentation, we see from Figure 2.2 that LSF and PCA fail to segment planes I and III, and RANSAC generates over-segmentation for planes I and III.

In a direct comparison between the HT and the RANSAC, which one is most suitable depends on the application and circumstances. Regarding feature detection in images, the scale-invariant feature transform (SIFT) algorithm [64] uses the HT based on the statement that RANSAC performs worse if the percentage of outliers falls significantly below 50 %. Other studies compare these two algorithms based on radar applications. Jacobs et al. [65] compare HT to RANSAC to determine which of the two performs better at tracking objects in noisy radar data. Their results show that both detect the lines best in the absence of noise, but still deliver good results in the presence of noise and multiple targets. While RANSAC performs better on most of the artificial datasets, as well as in scenarios with multiple targets, the HT appears to provide more accurate and stable results in the experimental data. The results of another comparative study by Facoiti et al. [66] present both algorithms as robust, efficient, and accurate in terms of detecting road contours in noisy radar data. RANSAC yields slightly lower errors, whereas the HT has minimal lower computing time. However, they only apply the standard version of the 3D HT. Its major drawbacks are the memory demands and especially the high computational times, which make it unsuited for real-time applications. [44, 67].

The most important performance of the two algorithms for this thesis is the extraction of planes from LiDAR point clouds. Tarsha-Kurdi et al. [68] perform an analytic comparison of both algorithms for automatic detection of 3D building roof planes from LiDAR data, in terms of processing time and sensitivity to cloud characteristics. Their results clearly state that the RANSAC algorithm not only delivers faster results, but also results of higher quality with a high percentage of successful outcomes compared to the 3D HT. Further research from Hulik et al. [67] presents an optimized version of the Standard Hough Transform (SHT) that improves plane detection by processing a continuous stream of point cloud data for real-time refinement. To evaluate their version of the HT, they compare it directly to the RANSAC algorithm. Their results show that the detection accuracy under different noise levels in the input data differ less than 0.01 rad in the case of normal vectors and 0.01 m in the case of distances. RANSAC is less stable than their optimized HT in the scenario where one key plane is to be detected, but the fluctuations of the RANSAC results are lower than 0.04 m and 0.006 rad. Further, the RANSAC algorithm outperforms the proposed HT regarding detail (e.g. precise detection into corners of the planes), but is more affected by the noise of the Kinect LiDAR. However, the performance of the optimized HT depends largely on the selected resolution. There are numerous variants for the HT. Borrmann et al. [44] compare different HT methods, including the Standard, the Probabilistic, the Adaptive Probabilistic, the Progressive Probabilistic, and the Randomized method. Their results regarding runtime indicate that the Randomized Hough Transform (RHT) runs significantly faster than all other methods. Also considering its results qualities, where the RHT performed second best after the Progressive Probabilistic HT, makes it the preferred choice for detecting an unknown number of planes in a LiDAR scan map in a reasonable time. Further, they introduce a new accumulator structure, the accumulator ball, as an improvement over traditional designs like the accumulator array and cube. The advantage of the accumulator ball is that it does not unjustifiably favor planes with specific parameters, thanks to the equal size of its patches. On average, this ball design slightly outperforms the cube. Maltezos and Ioannidis [69] also confirm the significantly reduced runtime of RHT compared to the SHT. Subsequently, they propose an extended RHT for automatic plane detection of each building and compare it to RANSAC. In their experiments, the computational time of their proposed extended RHT is much higher than the computational time of RANSAC. However, for their application, the extended RHT performs better. Even though RANSAC detected slightly more correct planes, its correctness is significantly lower due to many over-segmented and spurious planes.

It is difficult to make a general statement about which of the two algorithms (HT and RANSAC) is superior. The performance of both algorithms depend on many parameters. The RANSAC algorithm's runtime is very dependent on the number of available planes in the point cloud and number of maximum iterations [67]. For the HT the discretization of the Hough space is crucial, since too small bins lead to a higher runtime. Additionally, the design of the

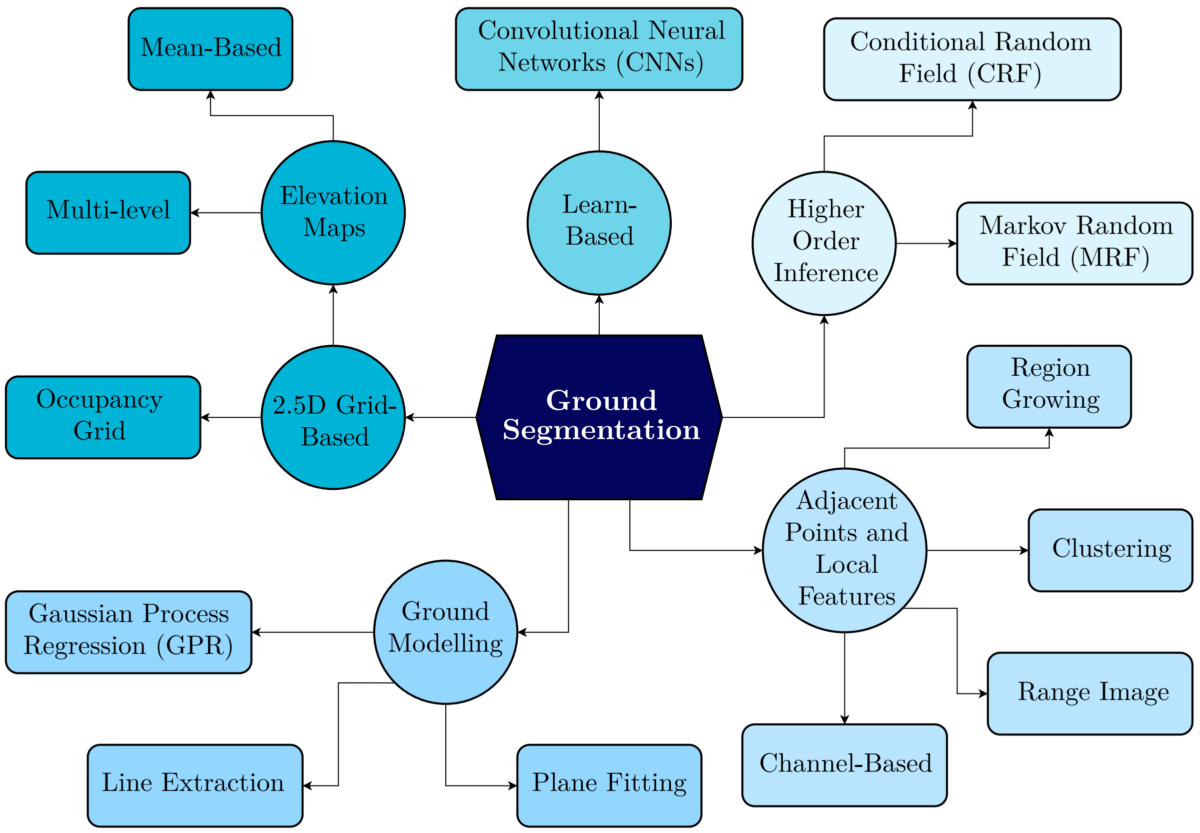


Figure 2.3: Classification and taxonomy of existing ground segmentation methods, according to [70].

accumulator plays a significant role. However, it is clear that RHT performs especially faster than the SHT [44, 69].

2.3 Ground Detection in LiDAR point clouds

Ground segmentation is a pivotal step in autonomous driving systems, particularly in processing LiDAR data to differentiate between ground and non-ground points. In this section, we explore the primary approaches used in ground segmentation, as presented by Gomes et al. [70]. They classify the different approaches into five main categories: 2.5D Grid-Based, Ground Modelling, Adjacent Points and Local Features, Higher Order Inference, and Learn-Based algorithms. Refer to Figure 2.3 for a visual representation of the classification and taxonomy of existing ground segmentation methods.

2.3.1 2.5D Grid-Based Algorithms

2.5D grid-based methods simplify the 3D point cloud data into a 2D grid, significantly reducing computational complexity. These methods are generally categorized into Occupancy Grids and

Elevation Maps.

Elevation Maps This approach involves creating a 2D grid where each cell represents the elevation of the ground surface. Techniques such as the mean-based elevation map proposed by Douillard et al. [71] utilize cell-wise average heights to classify ground points. Asvadi et al. [72] and Li et al. [73] extend this by incorporating height variance, which enhances the detection of flat ground but may misclassify low objects as ground.

Occupancy Grid Maps This technique involves creating a grid where each cell is marked as occupied, free, or unknown. This method is robust and widely used in autonomous driving. Himmelsbach et al. [74] and Luo et al. [75] demonstrate the effectiveness of occupancy grids in ground segmentation and object classification, achieving high accuracy and real-time performance.

2.3.2 Ground Modelling

Ground modelling techniques focus on representing the ground surface using geometric or statistical methods. The primary approaches include Plane Fitting, Line Extraction, and Gaussian Process Regression (GPR).

Plane Fitting Plane fitting involves identifying and estimating the parameters of a ground plane within the point cloud data. Hu et al. [76] use the RANSAC algorithm to fit a plane to the lowest elevation points and classify ground points based on their orthogonal distance from the plane. Lim et al. [77] employ a Concentric Zone Model for point cloud representation and a Region-wise Ground Plane Fitting method using PCA. Zhang et al. [78] use RANSAC for planar modeling, focusing on points with negative vertical angles.

Line Extraction Line extraction methods divide the point cloud into segments and model the ground using local line fits. Himmelsbach et al. [79] use local line fits within a polar grid map to model the ground plane.

Gaussian Process Regression (GPR)-Based GPR-based methods estimate and interpolate ground elevation, handling various data complexities and providing uncertainty measurements. Vasudevan et al. [80] propose a hyper-parameter-based GPR method that divides the problem into one-dimensional regressions, offering real-time performance. They use large-scale data sets from real mining applications, including both sparse mine planning data and dense laser scanner data from a Riegl LMSZ420 laser scanner. Douillard et al. [81] implement a GPR-based algorithm with Gaussian Process Incremental Sample Consensus for terrain modeling and outlier rejection.

2.3.3 Adjacent Points and Local Features

These techniques analyze relationships between neighboring points and local geometric features to classify terrain in point clouds. They leverage patterns, height differences, and terrain gradients. The methods are categorized into Channel-Based, Region-Growing, Clustering, and Range Images.

Channel-Based Channel-based approaches utilize LiDAR sensor patterns to classify points based on vertical and horizontal relationships. Chu et al. [82] analyze vertical lines in multi-layer LiDAR data to detect ground and obstacle points. Rieken et al. [83] enhance channel-based approaches by generating elevation maps for initial ground classification.

Region-Growing Region-growing methods segment the point cloud by expanding regions from seed points based on similarity criteria. Moosmann et al. [84] apply a graph-based region-growing algorithm with local convexity criteria, but the method achieves segmentation times around 250 ms, limiting real-time performance. Na et al. [85] enhance region-growing by merging regions to correct overly segmented areas, improving accuracy.

Clustering Clustering techniques group point cloud data based on local features or statistical properties. Douillard et al. [81] create a voxel grid and cluster voxels by height averages and variances to identify ground points, which are then removed from the point cloud. Yang et al. [86] use shape features of adjacent points to assemble and identify ground segments, showing good results in identifying planar road surfaces.

Range Images Range images project 3D LiDAR data into 2D images, exploiting the trigonometric relationships in spherical coordinates for efficient processing. Bogoslavskyi and Stachniss [87] perform ground filtering in under 4 ms per frame, demonstrating high efficiency with the KITTI dataset. Hasecke et al. [88] introduce FLIC (Fast Lidar Image Clustering), a real-time instance segmentation method that uses Euclidean distance and skip connections to enhance segmentation accuracy and robustness against over-segmentation.

2.3.4 Higher Order Inference

Higher order inference methods address the challenge of sparse point clouds from LiDAR sensors by incorporating advanced probabilistic models. These methods, including Markov Random Fields (MRF) and Conditional Random Fields (CRF), are designed to improve segmentation accuracy in complex and sparse environments.

Markov Random Field MRF models use undirected graphs to represent local dependencies between point cloud data, enhancing segmentation accuracy despite sparsity. Zhang et al. [89] enhance MRF-based methods by introducing probabilistic height measurements and a multi-label Markov network, achieving low false positive rates in complex environments. Huang et al. [90] reduce computational demands by using ring-based elevation maps for coarse segmentation and range images for MRF modeling.

Conditional Random Field CRF extends MRF by incorporating sequence labeling and long-range dependencies, improving ground modeling through spatiotemporal relationships. Rummelhard et al. [91] add spatial and temporal dependencies to CRF, modeling ground surfaces with improved accuracy. Wang et al. [92] apply CRF to a 2D lattice plane for ground modeling, using RANSAC for extraction.

2.3.5 Learn-Based

Learn-based methods have significantly advanced point cloud segmentation for LiDAR data, though they come with challenges related to computational demands and dataset requirements. Pomerleau's ALVINN (1988) [93] was an early neural network for road-following, demonstrating the potential of machine learning in autonomous vehicles despite limitations with high-speed performance due to computational constraints. PointNet, introduced by Qi et al. [94], revolutionized point cloud classification but struggles with large-scale data and fine details due to weak local feature learning. PointNet++ [95] improved performance on large-scale point clouds with a hierarchical framework, though it increased computational complexity. To address computational challenges, Lyu et al. [96] proposed an FPGA-based solution, while Velas et al. [97] achieved ground segmentation in under 7 ms with GPU acceleration, using a Velodyne HDL-64E that processes 1.3 M points per second.

In this work, we build on established methods in the field of point cloud processing and ground segmentation. We utilize the kd-tree data structure for several pre-processing steps, which has been shown to be highly efficient for NNS queries. Specifically, we apply the fixed radius search and k-NNS. For the ground plane segmentation, we implement multiple model-based algorithms, including LSF, PCA, RANSAC, and the RHT. These methods are well-recognized for their effectiveness in identifying planar surfaces within 3D data, making them suitable for tasks that involve segmentation of ground planes or other planar structures in point clouds. By applying these plane segmentation techniques, we aim to accurately detect the ground plane through a mathematical model. Consequently, our approach belongs to ground segmentation, specifically within the domain of ground modeling through plane fitting.

Chapter 3

Theoretical Fundamentals

In this chapter, we provide an overview of essential theoretical background knowledge. First, we present a general background to LiDAR data. Then we explain the data structure chosen for this work to process the data, which occurs in some pre-processing steps, i.e., the kd-tree. Finally, we present the four essential plane segmentation algorithms that are implemented and compared in this work.

3.1 LiDAR Point Clouds

LiDAR, which stands for Light Detection and Ranging, is a remote sensing technology that uses laser pulses to measure distances to objects and surfaces. The result of these measurements is a collection of data points in 3D space ($\mathbb{R}^3$), known as a point cloud. Each point in the point cloud represents a precise location where a laser pulse has been reflected from a surface, capturing the shape and position in relation to the LiDAR sensor of objects or the terrain in the surrounding area. A LiDAR scan involves passing over an area with the laser to capture these data points and map the environment in three dimensions. Multiple scans from different angles can be combined to create a detailed and comprehensive 3D representation of the scanned area. Besides the relative position of the objects, which are represented by either Cartesian or spherical coordinates (typically in 3D, depending on the LiDAR model), most LiDAR sensors also record the strength of the signal reflected from the laser (intensity). This intensity information provides valuable insights into the density or material composition of the surface. In addition, point clouds can store a wide range of other information, including the return number, scan angle, scan direction, point density, and timestamps. This extensive dataset enables advanced analysis and visualization, where points can be classified into different categories or objects based on features such as ground surfaces, vegetation, or man-made structures. Figure 3.1 shows an exemplary scan of the robotics hall, located on the Campus Hubland Süd of the Julius-Maximilians University Würzburg. These point clouds are often used in applications such as topographic mapping, autonomous vehicle navigation, and environmental monitoring.

To convert a local LiDAR point $\mathbf{p}_{\text{local}} = [x, y, z]^T$ into a global frame, we use a rotation matrix. We transform this point into the global frame by applying a rotation matrix $\mathbf{R}$ as

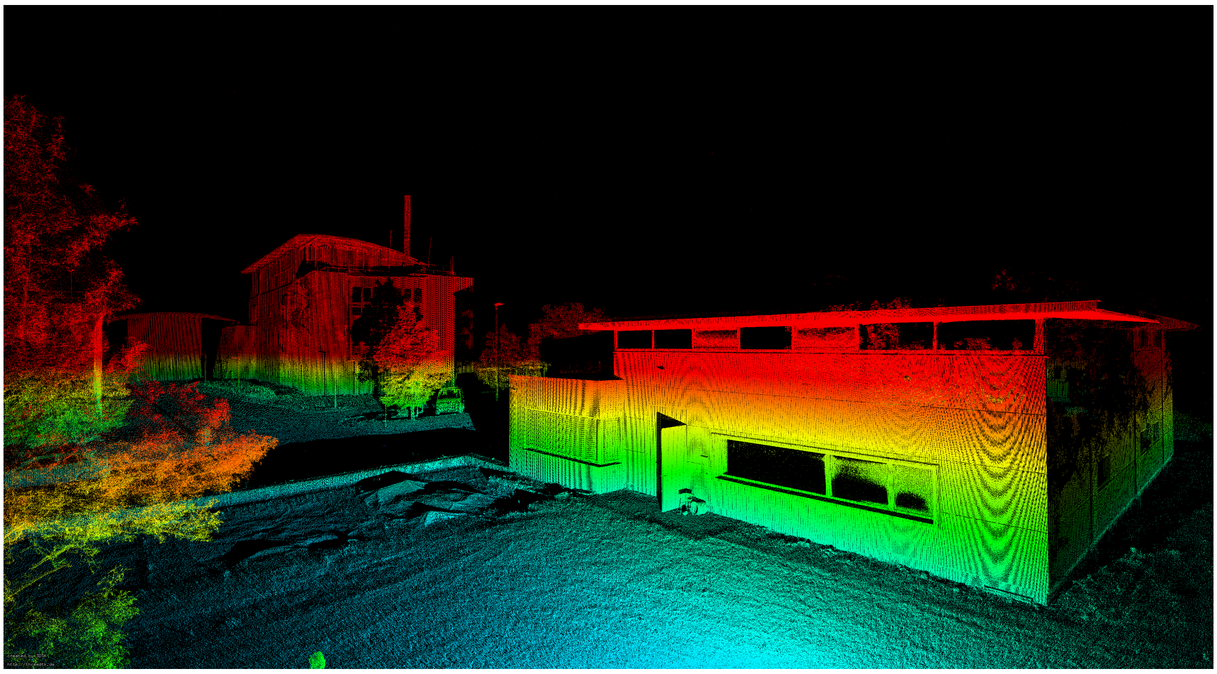


Figure 3.1: Exemplary LiDAR scan of the robotics hall, located on the Campus Hubland Süd of the Julius-Maximilians University Würzburg. Taken with the Riegl VZ-400 laser scanner [98].

follows:

$$\mathbf{p}_{\text{global}} = \mathbf{R} \cdot \mathbf{p}_{\text{local}} \,. \tag{1}$$

The rotation matrix **R** is a 3x3 matrix that rotates the local coordinate system to align with the global coordinate system. Usually, the rotation matrix is based on Euler angles (roll ϕ, pitch θ, yaw ψ). Let C_α and S_α denote the cosine and sine of the angle in the subscript, then R is given by:

$$\mathbf{R} = \begin{bmatrix} C_\psi C_\theta & C_\psi S_\theta S_\phi - S_\psi C_\phi & C_\psi S_\theta C_\phi + S_\psi S_\phi \\ S_\psi C_\theta & S_\psi S_\theta S_\phi + C_\psi C_\phi & S_\psi S_\theta C_\phi - C_\psi S_\phi \\ -S_\theta & C_\theta S_\phi & C_\theta C_\phi \end{bmatrix} . \tag{2}$$

In this work, the rotation matrix **R** comes from the Kalman filter. Using the same rotation matrix, we transform the normal vector n from the local LiDAR frame to the global frame.

3.2 Kd-Tree

The kd-tree, short for k-dimensional tree, is a data structure that was introduced by Louis Bentley in 1975 [6]. It is primarily used for organizing points in a k-dimensional space, allowing for efficient multidimensional search operations such as nearest neighbor searches. This structure is particularly useful in computer graphics, geographic information systems, and machine learning, where handling large, multidimensional datasets is common.

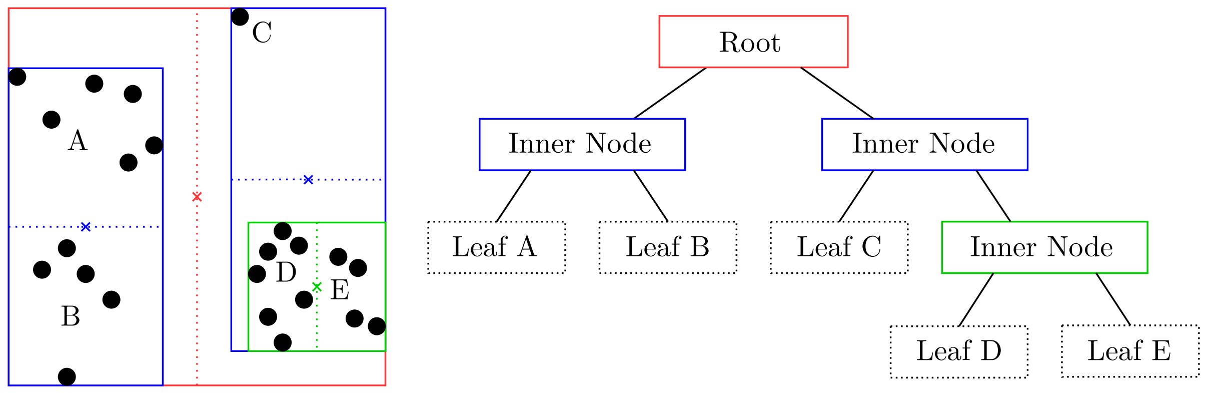


Figure 3.2: Schematic of the structure of a kd-tree, according to [100]. Left: A 2-dimensional kd-tree constructed from 23 points (black circles), showing the bounding boxes (solid lines), their centers (crosses), and the split axes (dotted lines). Each letter indicates the groups of points assigned to each leaf node. Right: The corresponding tree structure for the 2D kd-tree. The color of the solid boxes matches the bounding boxes in the left figure, while boxes with dotted outlines represent leaf nodes. The letters in the leaf nodes correspond to the groups of points shown in the left figure.

3.2.1 Construction

The construction of a kd-tree begins by considering the entire dataset within its bounding box, which represents the entire space that contains all the points. The tree is built by recursively subdividing this space along one of the principal axes (e.g., for a 3D space: x, y, or z). At each level of the tree, a different axis is selected, usually in a cyclical order to determine the direction of the split. The split creates a hyperplane that is perpendicular to the chosen axis, dividing the space into two half-spaces. Each half-space is then enclosed in a new bounding box to define the region for further subdivision. There are multiple splitting methods, such as the center, median, or mean, whose selection depends on the specific requirements. To ensure the tree is balanced, the median method is commonly used to divide the space along the chosen axis. The recursive subdivision continues until each leaf node only contains a specified number of points, known as the bucket size. Figure 3.2 shows an exemplary schematic of the structure of a 2D kd-tree. The construction of the kd-tree typically has a time complexity of $\mathcal{O}(n \log n)$ for n points. However, poor selection of the root node possibly leads to an unbalanced tree and might result in the worst-case construction time of $\mathcal{O}(n^2)$. Choosing the median of the points helps prevent this unbalanced tree structure and mitigate the risk of degraded performance. [6, 99]

3.2.2 Static vs. Dynamic

There are two main types of kd-tree: static and dynamic. Static kd-tree are built once and remain unchanged, making them efficient for repeated search operations on a fixed dataset. However, if the dataset changes (e.g., points are added or removed), the static kd-tree may become unbalanced and inefficient for searches. To address this, libraries like FLANN [18] allow for point insertion and deletion, but the tree must be completely rebuilt periodically to maintain its efficiency, which can be computationally expensive. Dynamic kd-tree, on the other hand, are designed to handle changes in the dataset better. These trees can adjust their structure by rebalancing as points are added or removed without needing a complete rebuild. This makes them suitable for applications where the data is constantly evolving. Examples of a dynamic kd-tree include the ikd-tree [14] and the scapegoat tree [101].

3.2.3 Nearest Neighbor Search

In a kd-tree, the Nearest Neighbor Search (NNS) algorithm efficiently locates the nearest point to a given query point $\boldsymbol{p}_q$ by exploiting the tree's structure. For a detailed breakdown of the algorithm's steps, see Algorithm 1. The algorithm begins at the root node and recursively traverses down the tree, selecting the child node that is closer to the query point along the current splitting axis. Once a leaf node is reached, the algorithm checks the distance between the query point and each point in the leaf, updating the current best candidate for the nearest neighbor if a closer point is found. As the recursion unwinds and it traverses back up the tree, the algorithm considers whether to explore the opposite branch of the tree. This decision is based on whether the distance between the query point and the splitting hyperplane is less than the current best distance. If it is, the opposite branch is explored because it might contain points closer than the current best. Here, it is important to note that there exist other implementations of NNS for kd-trees, where the decision to explore the opposite branch is influenced by the distance to the bounding box of the child node rather than solely the distance to the splitting axis. The process continues until the entire tree is traversed, either confirming or updating the nearest neighbor found so far. The computational complexity of NNS in a kd-tree is $\mathcal{O}(\log n)$, where n is the number of points in the tree. When searching for the k-Nearest Neighbors (kNN), the complexity becomes $\mathcal{O}(k \log n)$, reflecting the additional effort required to track multiple nearest points. [5, 6, 99]

3.2.4 Radius Search

Radius search, also known as range search, in a kd-tree is a process used to find all points within a specific range. Similar to NNS, the radius search exploits the hierarchical structure of the kd-tree to efficiently narrow down the search space and avoid unnecessary computations. The radius search algorithm begins at the root of the kd-tree and recursively traverses the tree. At each node, it checks whether the node's splitting hyperplane intersects the search radius. If it does, the algorithm explores both subtrees, since points within the search radius might exist on either side of the hyperplane. If the splitting hyperplane does not intersect the search radius, the algorithm only explores the subtree that contains points within the radius. During the traversal, when the algorithm reaches a leaf node, it checks each point in the node to determine whether it lies within the specified radius. All points that satisfy the radius condition are added to the result set. The time complexity of radius search depends on the dimensionality of the space and the distribution of points. For a radius search in d-dimensional space, the complexity is $\mathcal{O}(n^{\frac{d-1}{d}} + k)$, where n is the number of points and k is the number of points within the radius. In 3D space, this simplifies to $\mathcal{O}(n^{\frac{2}{3}} + k)$, with a worst-case complexity of $\mathcal{O}(n)$ when the search radius is very large or the point distribution is uniform. [5, 6, 99]

Algorithm 1 Nearest Neighbor Search in a kd-tree

```
1: Input: k-d tree $T$, query point $\boldsymbol{p}_q$
2: Output: Nearest neighbor of $\boldsymbol{p}_q$ in $T$
3: best_distance ← $\infty$
4: nearest_neighbor, best_distance ← search(T.root, $\boldsymbol{p}_q$, best_distance)
5:
6: Function search(node, $\boldsymbol{p}_q$, best_distance)
7:   if is_leaf(node) then
8:     for each point in node do
9:       distance ← $\|\boldsymbol{p}_q -$ point$\|$
10:      if distance < best_distance then
11:        best_distance ← distance
12:        nearest_neighbor ← point
13:      end if
14:    end for
15:  else
16:    axis_distance ← | node.split_coordinate − $\boldsymbol{p}_q$ coordinate along the split axis |
17:    if $\boldsymbol{p}_q$ on left of split axis then
18:      nearest_neighbor, best_distance ← search(node.left, $\boldsymbol{p}_q$, best_distance)
19:      if axis_distance < best_distance then    // Possibly closer points in right subtree
20:        nearest_neighbor, best_distance ← search(node.right, $\boldsymbol{p}_q$, best_distance)
21:      end if
22:    else
23:      nearest_neighbor, best_distance ← search(node.right, $\boldsymbol{p}_q$, best_distance)
24:      if axis_distance < best_distance then    // Possibly closer points in left subtree
25:        nearest_neighbor, best_distance ← search(node.left, $\boldsymbol{p}_q$, best_distance)
26:      end if
27:    end if
28:  end if
29:  return nearest_neighbor, best_distance
30: End Function
```

3.3 Plane Segmentation

Segmentation of 3D point clouds involves classifying the points into distinct homogeneous regions, where each region shares similar properties, as described by Nguyen [25]. In this work, we apply four different algorithms for plane segmentation, which we explain in the following subsections. We first present the plane fitting algorithms, LSF and PCA. Plane fitting involves finding the best-fitting plane model for a given set of points. Following this, we introduce the plane detection algorithms, RANSAC and HT, including the implemented RHT variant. Plane detection focuses on identifying regions within the point cloud that are likely to represent planar surfaces.

3.3.1 Least Square Fitting

The method of Least Squares Fitting (LSF) was first developed by the French mathematician Adrien-Marie Legendre in the late 18th century [102]. It was later independently rediscovered and popularized by the German mathematician Carl Friedrich Gauss in the early 19th century [103]. Gauss's work significantly advanced the application of LSF in various fields, including mathematics, statistics, and scientific research. Both mathematicians originally applied this method to improve the accuracy of astronomical measurements, such as understanding the orbits of comets from inexact positional data. In general, LSF works by minimizing the sum of the squared differences between observed values and the values predicted by a model, thereby finding the model parameters that best approximate the given data.

The application of LSF has significantly evolved and expanded over time. One notable development is the planeSVD technique, introduced for computer vision systems [46]. This method utilizes Singular Value Decomposition (SVD) to robustly fit planes to data, which is crucial for tasks such as 3D reconstruction and object recognition. SVD is a matrix factorization technique, which decomposes a given matrix $\mathbf{A}$ into three components, such that $\mathbf{A} = \mathbf{USV}^T$, to extract important features and structure from data. The matrix $\mathbf{U}$ contains the left singular vectors, which represent directions in the input space of $\mathbf{A}$ where the data varies most. The matrix $\mathbf{S}$ holds the singular values, indicating the strength or importance of these directions, with larger values signifying more significant features. Finally, $\mathbf{V}^T$ contains the right singular vectors, defining directions in the output space that align with the directions in the input space specified by $\mathbf{U}$. This decomposition helps to reveal the structure of $\mathbf{A}$, simplifying data analysis and dimensionality reduction. In addition to advancements in computer vision, LSF has been widely employed in image segmentation and surface classification. For instance, early works demonstrated how LSF could be used to segment range images and classify surfaces by fitting planes to different regions of the image [104, 105]. These techniques facilitate the analysis of complex surfaces and objects in various fields, from robotic perception to environmental modeling. The integration of LSF with modern computational methods continues to enhance its effectiveness and applicability in diverse domains.

Plane Fitting in 3D Data LiDAR point clouds represent data in 3D space, capturing the precise locations of various surfaces and objects. To analyze these point clouds effectively, we employ LSF to fit geometric surfaces, in our case planes, to the data. In 3D Euclidean space $\mathbb{R}^3$, a plane can be expressed by the equation:

$$ax + by + cz + d = 0\,, \tag{3}$$

where $\boldsymbol{n} = [a,\, b,\, c]^T$ is the normal vector of the plane and d is the plane's distance from the origin. To determine the optimal plane that best fits a set of 3D points, LSF minimizes the sum of the squared distances from each point to the plane. The objective is to minimize:

$$\sum_{i=1}^{m} d_i^2\,, \tag{4}$$

where d_i is the perpendicular distance from a point $\boldsymbol{p}_i = [x_i,\, y_i,\, z_i]^T$ to the plane, given by:

$$d_i = \frac{|ax_i + by_i + cz_i + d|}{\sqrt{a^2 + b^2 + c^2}} \,. \tag{5}$$

By minimizing the squared sum of these distances, LSF finds the plane parameters $(a,\, b,\, c,\, d)$ that provide the best fit to the given points, thereby minimizing the overall fitting error. [42]

3.3.2 Principal Component Analysis

Principal Component Analysis (PCA), originally introduced by Pearson in 1901 [47], has been a foundational method in various applications, including plane fitting and normal estimation. One of the earliest uses of PCA for plane fitting was proposed by Hoppe et al. (1992) [106], who utilized PCA to estimate tangent planes from the local neighbors of each sampled point in point clouds [43].

Plane Fitting in 3D Data In the context of LiDAR point clouds, PCA is applied to determine the best-fitting plane by analyzing the covariance matrix of the point cloud. Specifically, PCA calculates the eigenvalue and eigenvector of the covariance matrix C. The eigenvector represents the axes of the point cloud, and the eigenvalue denotes the variance along these axes. The eigenvector corresponding to the smallest eigenvalue is identified as the normal vector of the optimal plane. The covariance matrix of a point cloud in 3D space is computed as follows:

1. **Compute the Centroid:** Calculate the mean of the x, y, and z coordinates to get the centroid $\bar{\mathbf{p}}$ of the point cloud with n points:

$$\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i \,, \quad \bar{y} = \frac{1}{n}\sum_{i=1}^{n} y_i \,, \quad \bar{z} = \frac{1}{n}\sum_{i=1}^{n} z_i \,, \tag{6}$$

$$\bar{\mathbf{p}} = [\bar{x}, \bar{y}, \bar{z}]^T \,, \tag{7}$$

   where x_i, y_i, and z_i are the coordinates of the i-th point, and $\bar{x}$, $\bar{y}$, and $\bar{z}$ are the mean coordinates.

2. **Compute the Covariance Matrix:** The covariance matrix C is a 3×3 matrix that describes the covariance between the coordinates of the points. The covariance matrix C is computed as:

$$C = \frac{1}{n}\sum_{i=1}^{n} (\boldsymbol{p}_i - \bar{\boldsymbol{p}}) \cdot (\boldsymbol{p}_i - \bar{\boldsymbol{p}})^T \,, \tag{8}$$

   where $\boldsymbol{p}_i$ is the i-th point of the point cloud. Specifically, the covariance matrix elements are:

$$C_{xx} = \frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})^2\,, \tag{9}$$

$$C_{xy} = C_{yx} = \frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})\,, \tag{10}$$

$$C_{xz} = C_{zx} = \frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})(z_i - \bar{z})\,, \tag{11}$$

$$C_{yy} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \bar{y})^2\,, \tag{12}$$

$$C_{yz} = C_{zy} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \bar{y})(z_i - \bar{z})\,, \tag{13}$$

$$C_{zz} = \frac{1}{n}\sum_{i=1}^{n}(z_i - \bar{z})^2\,. \tag{14}$$

Combine the computed elements into the covariance matrix C as follows:

$$C = \begin{bmatrix} C_{xx} & C_{xy} & C_{xz} \\ C_{yx} & C_{yy} & C_{yz} \\ C_{zx} & C_{zy} & C_{zz} \end{bmatrix}\,.\ [42] \tag{15}$$

Based on the covariance matrix C, it is possible to get the corresponding eigenvalues and eigenvectors from the following equation.

$$C \cdot \boldsymbol{v}_l = \lambda_l \cdot \boldsymbol{v}_l \tag{16}$$

To find the eigenvalues λ_l, first solve the characteristic polynomial:

$$\det(C - \lambda I) = 0\,. \tag{17}$$

For each eigenvalue $\lambda_{l,i}$, substitute it into the following equation to determine the corresponding eigenvectors $\boldsymbol{v}_{l,i}$:

$$(C - \lambda_{l,i} I) \cdot \boldsymbol{v}_{l,i} = 0\,. \tag{18}$$

The eigenvectors represent the principal directions of variance in the point cloud, and the eigenvalue associated with the smallest value indicates the normal vector $\boldsymbol{n}$ to the best-fitting plane. [42]

3.3.3 Hough Transform

The Hough Transform (HT), originally proposed by Paul V.C. Hough in 1962 [61], is a powerful technique for recognizing complex patterns and detecting parameterized objects in images. Although it was initially used for detecting lines and circles, the method has been extended to detect arbitrary shapes, leading to the development of the Generalized HT by Ballard, which can efficiently find shapes at any orientation or scale using directional edge information [107]. An extensive overview of the HT and its applications is available in [108]. In 3D shape detection, several authors have employed the HT to detect shapes in range data. Henderson and Fai [109] used the Generalized HT for object detection in 3D data obtained from a LiDAR system. Their method involved detecting planar segments in the data and matching them with models. Belaïd and Mohr [110] also applied the HT to 3D shape detection, fitting quadric surfaces and planes to local neighborhoods of data points. They proposed an efficient method for transform calculation based on a divide-and-conquer approach.

Essentially, the HT works by transforming the representation of shapes from the spatial domain into a parameter space, known as Hough space. In this space, the shapes are represented by their defining parameters, e.g., the gradient and the intersection point for a line or the radius and the center coordinates for a circle. Each point in the spatial domain is assigned to a curve or surface in Hough space, which represents all possible shapes containing that point. The intersections of these curves or surfaces indicate the presence of the corresponding shape in the original data. To efficiently identify the shapes, the Hough space is discretized into cells, with each cell representing a possible shape. A data structure called the accumulator stores these cells, with each cell containing a score that is incremented when a point lies on the corresponding shape. After all points have been processed, the cells with the highest scores represent the detected shapes. [44, 99]

Plane Detection in 3D Data The HT is also applied to detect planes in 3D point clouds. In this context, a plane in $\mathbb{R}^3$ is typically represented using the Hesse normal form to avoid issues with undefined slopes that arise in other representations. The Hesse normal form of a plane is given by:

$$\rho = x \cdot n_x + y \cdot n_y + z \cdot n_z = \boldsymbol{n} \cdot \boldsymbol{p}\,, \tag{19}$$

where $\boldsymbol{p} = [x,\, y,\, z]^T$ is a point on the plane, $\boldsymbol{n} = [n_x,\, n_y,\, n_z]^T$ is the normal vector of the plane, and ρ is the perpendicular distance from the origin to the plane. Alternatively, the plane is expressed in terms of spherical coordinates:

$$\rho = x \cdot C\varphi \cdot S\theta + y \cdot S\varphi \cdot S\theta + z \cdot C\theta,\,, \tag{20}$$

where θ is the angle between the normal vector and the xy-plane, and φ is the angle between the xy-plane and the normal vector in the z-direction. These three parameters, ρ, θ, and φ, define the 3D Hough space. There are several variants of the HT that are used to detect planes in this Hough space. [44, 99]

Standard Hough Transform With the Standard Hough Transform (SHT), by Duda et al. (1972) [111] each point in the point cloud is traversed and mapped to the Hough space.

1. **Discretize the Hough Space:** The Hough space is divided into cells with defined extents in ρ, θ, and φ. Each cell represents a potential plane.
2. **Use an Accumulator:** A data structure, called the accumulator, stores votes for each cell. For each point, the cells corresponding to possible planes are incremented. For plane detection in $\mathbb{R}^3$, the accumulator equals a 3-dimensional array.

3. **Voting Process:** Each point contributes to the cells in Hough space. After all points have been processed, cells with high scores indicate the presence of planes.

Refer to Algorithm 2, which outlines the process of the SHT, based on [44].

Algorithm 2 Standard Hough Transform (SHT)

```
1: Input: Point set $P$
2: Output: Detected planes
3: Discretize Hough space into cells with parameters $\rho'$, $\varphi'$, $\theta'$
4: Initialize accumulator with zero scores
5: for each point $\boldsymbol{p}_i \in P$ do
6:    for each cell $(\theta, \varphi, \rho)$ in accumulator do
7:       if point $\boldsymbol{p}_i$ lies on the plane defined by $(\theta, \varphi, \rho)$ then
8:          Increment cell $A(\theta, \varphi, \rho)$
9:       end if
10:   end for
11: end for
12: Search for cells with the highest scores
13: Identify the planes based on the highest scoring cells
```

Due to its high computation time, the Standard HT (SHT) proves impractical for real-time applications. To enhance efficiency and manage larger datasets, several variants of the HT have been developed. The Probabilistic HT improves efficiency by using only a subset of the input points, thus reducing the computational load. The Progressive Probabilistic HT (PPHT) further refines this approach by randomly selecting points and performing the transform. It halts when a cell in the Hough space accumulates votes from a predefined percentage of points. The Adaptive Probabilistic HT (APHT) operates similarly to PPHT but includes an additional step of maintaining a list of potential maxima, terminating when this list becomes stable. Lastly, the Randomized Hough Transform (RHT) adopts a different strategy by randomly selecting three points to define a plane and then accumulates votes in the corresponding cell based on these points. We implement the RHT for our application. [44, 99]

Randomized Hough Transform The Randomized Hough Transform (RHT), as described by Up et al. (1990) [112], reduces the number of computations by focusing on subsets of points. For plane detection, it involves:

1. **Selecting Three Points:** Randomly pick three points from the point cloud and check if they fulfill the distance criterion, which ensures that the distance between any two of the three points is below the maximum distance threshold and above the minimum distance threshold. This criterion helps in reducing errors due to noise and ensuring that the selected points are likely to belong to the same plane.

2. **Calculating Plane Parameters:** Compute the plane parameters (ρ, θ, and φ) based on these three points.

3. **Updating Accumulator:** Increment the cell in the accumulator corresponding to the computed plane.

4. **Checking Threshold:** If a cell's count reaches a threshold, the corresponding plane is considered detected.

5. **Removing Points:** Delete points close to the detected plane and reset the accumulator for the next iteration.

Refer to Algorithm 3, which outlines the process of the RHT, based on [44].

Algorithm 3 Randomized Hough Transform (RHT)

```
1: **Input:** Point set $P$, minimum and maximum distance threshold
2: **Output:** Detected planes
3: Discretize Hough space into cells with parameters $\rho'$, $\varphi'$, $\theta'$
4: Initialize accumulator with zero scores
5: **while** there are still enough points in $P$ **do**
6:     Randomly pick three points $p_1$, $p_2$, $p_3$ from $P$
7:     **if** points $p_1$, $p_2$, $p_3$ fulfill the distance criterion **then**
8:         Calculate plane parameters $(\theta, \varphi, \rho)$ spanned by $p_1$, $p_2$, $p_3$
9:         Increment cell $A(\theta, \varphi, \rho)$ in the accumulator
10:        **if** the counter $|A(\theta, \varphi, \rho)|$ equals threshold $accumulator\_max$ **then**
11:            Delete all points close to $(\theta, \varphi, \rho)$ from $P$
12:            Reset the accumulator
13:        **end if**
14:    **end if**
15: **end while**
```

3.3.4 Random Sample Consensus

The Random Sample Consensus (RANSAC) algorithm, introduced by Fischler and Bolles in 1981 [63], is a fundamental technique for robust parameter estimation in datasets with outliers. It identifies the model that best fits the majority of data points by leveraging the largest consensus of inliers, effectively mitigating the influence of outliers.

Since its inception, RANSAC has become an essential tool in computer vision and image processing. On its 25th anniversary in 2006, a dedicated workshop at the International Conference on Computer Vision and Pattern Recognition highlighted numerous advancements. These improvements focused on enhancing the algorithm's speed, robustness, and accuracy while reducing dependence on user-defined parameters. Despite its effectiveness, RANSAC's performance is sensitive to the choice of the noise threshold. An excessively large threshold may lead to all hypotheses being ranked similarly, while a small threshold may cause parameter estimates to be unstable. To address these issues, modifications such as MSAC (M-estimator SAmple and Consensus) and MLESAC (Maximum Likelihood Estimation SAmple and Consensus) have been proposed by Torr and Zissermann [113], which evaluate consensus set quality based on likelihood rather than cardinality. Further refinements include PROSAC (PROgressive SAmple Consensus) [114], which leverages a priori information about data points, and R-RANSAC [115], which introduces a randomized approach that evaluates model goodness using a reduced dataset before considering the full set to reduce computational demands. Many additional variants and enhancements of RANSAC continue to emerge, addressing various challenges in robust model fitting.

Plane Detection in 3D Data To detect planes within 3D datasets, RANSAC is utilized to robustly estimate the plane parameters despite the presence of outliers. The algorithm iterates through several steps to identify the most fitting plane model from the data.

1. **Random Selection:** Randomly select the smallest number of points needed to fit the model (three for a plane).
2. **Model Estimation:** Fit the plane to the selected subset of points.
3. **Inlier Identification:** Evaluate all other points of the point cloud against the fitted plane. Points that fit the plane within a predefined tolerance (inlier threshold) are considered inliers.
4. **Consensus Set:** Add inliers to the consensus set and assess the quality of the fitted plane based on the size of the consensus set.
5. **Model Refinement:** If the consensus set is large enough, refine the plane using all points in this set.

RANSAC performs this process over a predefined number of iterations n_{max}. Algorithm 4 outlines the process of the RANSAC, based on [116].

To enhance the efficiency of RANSAC and potentially terminate it earlier than the predefined maximum number of iterations, it is crucial to understand how to determine the necessary number of iterations. The algorithm's termination criteria are influenced by the confidence level and the inlier ratio. The confidence level represents the desired probability that the final model is accurate. The inlier ratio is the proportion of inliers to the total number of data points. The required number of iterations n_I is calculated to ensure a high probability of finding a model with the desired confidence level. If the inlier ratio is ratio_i, and the probability of drawing all inlier points in a single iteration is $\text{ratio}_i^{n_P}$, where n_P is the number of points needed to fit the model, then the probability of not achieving this in one iteration is: $1 - \text{ratio}_i^{n_P}$. To ensure that this probability is sufficiently low, the number of iterations n_I should be chosen so that:

$$(1 - \text{ratio}_i^{n_P})^{n_I} < \alpha \,, \tag{21}$$

where α is the acceptable failure probability (e.g., 0.05 for a 95 % confidence level). Solving for n_I gives:

$$n_I = \frac{\log(\alpha)}{\log(1 - \text{ratio}_i^{n_P})} \,. \tag{22}$$

This calculation allows for early termination of the RANSAC algorithm once a sufficiently accurate model is found with high confidence, rather than running the full set of maximum iterations. [116]

Algorithm 4 Random Sample Consensus Algorithm

```
1: Input: Data points $D$, number of iterations $n_{max}$, model fitting function, distance thresh-
   old $t$
2: Output: Best model and inliers
3: best_inliers $\leftarrow \emptyset$
4: best_model $\leftarrow$ NULL
5: for $i = 1$ to $n_{max}$ do
6:     Randomly select $n_p$ points from $D$
7:     Fit a model $M$ to the selected points
8:     inliers $\leftarrow \emptyset$
9:     for each point $\boldsymbol{p}$ in $D$ do
10:        compute distance $d$ from $\boldsymbol{p}$ to model $M$
11:        if $d < t$ then
12:            inliers $\leftarrow$ inliers + point $\boldsymbol{p}$
13:        end if
14:    end for
15:    if size of inliers $>$ size of best_inliers then
16:        best_inliers $\leftarrow$ inliers
17:        best_model $\leftarrow$ model $M$
18:    end if
19: end for
20: return best_model and best_inliers
```

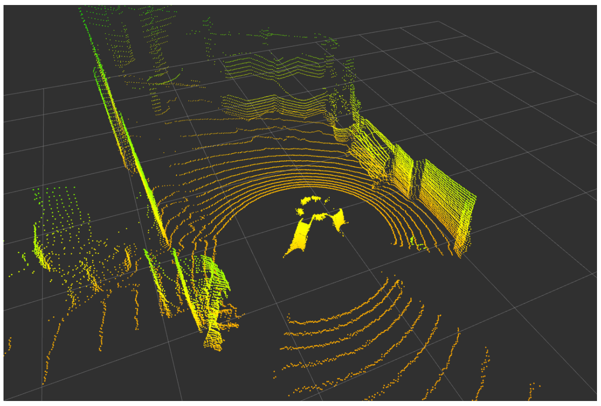
Chapter 4

Approach

In this work, we aim to calculate the normal vector $\boldsymbol{n}$ of the ground plane in real-time based on the data from the LiDAR of the spherical mobile mapping system. The Kalman filter [4], a crucial component of the spherical mobile mapping system's navigation and control system, relies on the orientation of the ground to ensure consistent and reliable state estimation. In this Kalman filter, the normal vector of the ground plane $\boldsymbol{n}$ is defined to be pointing into the ground. This way the cross product with the rotation speed vector results in the linear velocity of the robot. The spherical mobile mapping system is equipped with a Hesai Pandar XT-32 [117] LiDAR scanner. It provides scans in which the ground is always visible due to its minimum scanning distance of 0 cm, refer to Figure 4.1. This visibility enables the extraction of the ground plane's normal vector $\boldsymbol{n}$ in each scan. The Hesai Pandar LiDAR operates at a frequency of 20 Hz, which corresponds to a real-time requirement to calculate $\boldsymbol{n}$ within 50 ms. Each scan contains around 30,000 points and provides a dense point cloud for this calculation. As the prototype currently has no locomotion mechanism, the scans also frequently capture the hands of the person moving the sphere, as depicted in Figure 4.1. The spherical mobile mapping system uses the Robot Operating System (ROS) for its programming. ROS is a comprehensive framework for the development of robot software that provides important tools and libraries to facilitate the integration of different hardware and software components. It works with a publisher-subscriber model, which allows different software modules, referred to as nodes, to communicate by exchanging messages on predefined topics. To achieve the objective of calculating the ground plane's normal vector, we implement a new ROS node that performs the following general steps.

1. Read in the new scan.
2. Identify the ground plane.
3. Calculate the normal vector of the ground plane.
4. Transform the normal vector from the LiDAR scan frame (`pandar_frame`) to the world frame (`map2`) and publish it.

For the segmentation of the ground plane, there are several state-of-the-art plane segmentation algorithms, refer to Section 2.2. These algorithms vary in their approach to filtering and processing point cloud data for accurate plane identification and possibly require different pre-processing


(a) LiDAR in horizontal position.

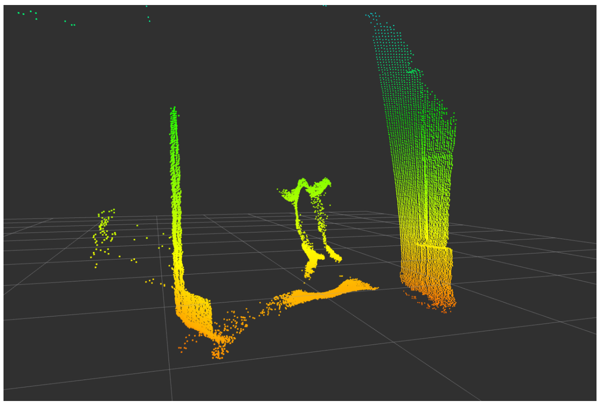

(b) LiDAR in almost vertical position.

Figure 4.1: Exemplary scans of the Hesai Pandar LiDAR during movement, illustrating that a person is moving the spherical mobile mapping system, the ground remains consistently visible, and that reflections from both the sphere and the person's hands are present.

steps. By implementing and comparing multiple plane segmentation algorithms under different pre-processing conditions, we aim to evaluate their performance regarding runtime, accuracy, and robustness in various scenarios. Through this detailed comparison, we intend to identify the most effective and efficient solution for calculating the normal vector of the ground plane for this specific real-time application.

In this chapter, we first introduce our approach for the ground plane extraction including which plane segmentation algorithms we compare, and their different needs of point cloud pre-processing. Then we present the necessary steps to get the normal vector of the extracted ground plane. Finally, we discuss the selection of the library for our application.

4.1 Ground Plane Extraction

Ground plane extraction involves sophisticated algorithms designed to identify and segment the ground surface from the point cloud data captured by LiDAR sensors. These plane segmentation algorithms aim to detect the dominant flat surface representing the ground. For a detailed overview of plane segmentation algorithms, refer to Section 2.2. In general, there are two fundamental approaches to plane extraction from point clouds: plane fitting and plane detection. Plane fitting aims to find the best-fitting plane model for a given set of points, while plane detection identifies regions within the point cloud likely to represent planar surfaces. Typically, the plane fitting method is known for its fast runtime and effectiveness in simple scenarios, while plane detection provides better results than plane fitting in terms of accuracy, but at the expense of runtime [25]. Moreover, plane fitting is susceptible to noise, making it vulnerable to outliers that may disrupt their estimations, resulting in inconsistent and potentially misleading outcomes [25, 42, 43]. Refer to Table 4.1 for an overview of the plane segmentation approaches' advantages and disadvantages, as well as their most popular algortihms.

In the following sections, we first present the plane segmentation algorithms used for this

Table 4.1: Advantages and disadvantages of the two fundamental plane segmentation approaches.

Approach	Plane Fitting	Plane Detection
Advantages	• fast (low computational effort) • effective in simple scenarios	• robust to outliers • high accuracy in any scenario
Disadvantages	• sensitive to noise and outliers	• slow (high computational effort)
Algorithms	LSF & PCA	RANSAC & RHT

work. Subsequently, we introduce the necessary point cloud processing to ensure the detection of the ground plane.

4.1.1 Plane Segmentation Algorithms

The most popular algorithms are: LSF and PCA as representatives of plane fitting methods, and RANSAC and 3D HT as examples of plane detection techniques. In Section 2.2.1 we reveal that there is no clear superiority between LSF and PCA. The same applies to the direct comparison of HT and RANSAC, refer to Section 2.2.2. Moreover, it is difficult to make a general statement, as both plane detection algorithms depend on several parameters. The runtime of the RANSAC algorithm is strongly dependent on the number of available planes in the point cloud and the number of maximum iterations [67]. For the HT, the discretization of the Hough space is decisive, as smaller bins lead to a higher runtime [44].

For this work, we implement and compare the performance of all four previously named algorithms. We select the RHT instead of the standard 3D HT due to its significant faster runtime [44, 69]. Further, the choice of accumulator also has an effect on the performance of the RHT algorithm. The comparative study from Borrmann et al. [44] determined, that the ball accumulator design, as opposed to the cube accumulator, yields slightly better results. Therefore, we chose for the ball accumulator for the implementation of the RHT in this work. For a detailed explanation of the individual algorithms, refer to Section 3.3. More importantly, for the usage of plane fitting methods, we require some sort of pre-processing of the point cloud. Taking the entire scan for plane fitting will lead to erroneous results, due to the fact that they are susceptible to noise and unable to handle outliers effectively. Therefore, we first need to identify a subset of points, which most likely is part of the ground plane. In contrast to the plane fitting methods, the plane detection methods are capable of processing the entire scan. However, RANSAC and RHT possibly identify multiple planes depending on the implementation, which poses the challenge of selecting the ground plane. Even in cases where they detect only one plane, we need to verify that the identified plane corresponds to the ground plane. We present strategies for addressing this in the subsequent section.

4.1.2 Point Cloud Pre-Processing

As mentioned in the previous Section 4.1.1, at least two of the four selected plane segmentation algorithms require pre-processing of the point cloud to ensure the segmentation of the ground plane. The scan we receive from the Hesai Panadar LiDAR consists of roughly 30,000 points per

Table 4.2: Possible point cloud pre-processing steps for the corresponding plane segmentation algorithms.

Algorithm	**Point Cloud Pre-Processing**
LSF & PCA	• sub-cloud - removal of outside points • filtered sub-cloud - downsampling and removal of reflections & hands
RHT & RANSAC	• none - entire scan/point cloud • sub-cloud - removal of outside points • filtered sub-cloud - downsampling and removal of reflections & hands

scan. This includes not only the scanned environment, but also the reflections of the sphere's shell and the hands of the person moving the sphere, due to its lack of a locomotion mechanism, refer to Figure 4.1. Based on these facts, Table 4.2 lists the possible point cloud pre-processing steps for each selected plane segmentation algorithm. For the scope of this work, we define a point cloud after the removal of the points, which are theoretically located outside the ground plane, as a **sub-cloud**. Furthermore, we indicate the removal of the reflections from the sphere and the hands, as well as the downsampling of the point cloud with the keyword **filtered**. Downsampling the point cloud reduces the number of points to smooth out the ground plane, remove isolated noise points, and achieve a more uniform distribution. This simplification of the 3D data further leads to improved efficiency in subsequent processing steps due to the reduced number of points.

When using the direct sub-cloud for LSF and PCA, the pre-processing is most likely faster than using the filtered sub-cloud as no additional filtering is required. But this approach might lead to lower accuracy due to noise and outliers. Applying LSF and PCA to the filtered sub-cloud may enhance accuracy but will increase the runtime due to the extra pre-processing. Both scenarios have potential advantages for our real-time detection of the ground plane. Therefore, we need to determine the extent of the impact on speed and accuracy and establish a trade-off between these factors. For RANSAC and RHT, taking the entire scan results in multiple detected planes, which then requires additional steps to identify the ground plane. This creates additional overhead, which increases the runtime, and the algorithms themselves also take longer to identify multiple planes. Therefore, we do not consider this as a suitable option for our application. Consequently, as with LSF and PCA, we only consider the sub-cloud, where we assume that the dominant plane represents the ground plane. As a result, RANSAC and RHT only need to identify the dominant plane, leading to improved efficiency and speed in their detection process. The impact of filtering on the results may be minimal, as both plane detection methods are known to be robust to outliers. Even though filtering involves additional processing, it possibly enhances the speed of RANSAC and RHT and potentially result in more accurate outcomes. It is essential to evaluate this aspect through experimentation to determine the most effective approach for real-time detection of the ground plane.

When pre-processing point clouds, we distinguish between two main tasks: reducing the point cloud to the points that are most likely to represent the ground plane (creating the sub-cloud) and filtering the point cloud to remove unwanted elements, i.e. spherical reflections and the hands of the person moving the sphere. In order to perform the steps including radius search and k-NNS listed below efficiently, the selection of a suitable multidimensional data structure beforehand is crucial. Based on previous research described in Section 2.1, we chose the kd-tree as the suitable data structure. It is not only the most commonly used tree, but also offers the best performance compared to other types, especially regarding the speed of NNS.

Strategies for Efficient Point Cloud Filtering To filter the point cloud, it is important to first examine the reflections and the hands in a scan. As can be seen from Figure 4.1, the reflections of the shell represent the spherical shape of the robot. As a result, they are all located within a certain radius around the origin of the LiDAR scan frame. By systematically removing all points falling within a specified radius r_{fil}, which contains the reflections and the hands of the person moving the sphere, around the origin of the LiDAR scan frame, we effectively filter the point cloud. To accomplish this, there are the following two potential options.

1. **Geometrical Approach:** Iterating through the entire point cloud and determine whether the point falls within this radius based on its position ($\boldsymbol{p}_i = [x_i,\, y_i,\, z_i]^T$). We calculate the squared distance d_i^2 of a point $\boldsymbol{p}_i$ from the origin using equation 23, and if $d_i^2 \leq r_{fil}^2$ we need to remove this point. To optimize computation time, we always use the squared Euclidean length of a vector in our calculations to avoid time-consuming square root operations.

$$d_i^2 = ||\boldsymbol{p}_i||^2 = x_i^2 + y_i^2 + z_i^2 \tag{23}$$

2. **kd-tree Approach:** Building a kd-tree from the point cloud and then use the fixed radius search function. As arguments, we pass the predefined radius r_{fil} to the function and for the query point we simply use the origin $\boldsymbol{o} = [0,\, 0,\, 0]^T$ of the LiDAR scan frame.

For the downsampling, which is also a part of the filter pre-processing step, we rely on an appropriate function from the selected library to complete the task efficiently. Here, we only set the parameter of the leaf size d_{ds}. Refer to Section 4.3.1 for further information.

Strategies for Efficient Sub-Cloud Creation We create a sub-cloud of a point cloud by reducing it to the points, which most likely represent the ground plane. These points lie within a region around the point in which the spherical mobile mapping system touches the ground. We define this point as the query point $\boldsymbol{p}_q$. Let $\boldsymbol{n}$ be the previously calculated normal vector of the ground plane and r_s be the radius of the robot, then we calculate $\boldsymbol{p}_q$ as:

$$\boldsymbol{p}_q = r_s\, \boldsymbol{n}\,. \tag{24}$$

To determine which points now most likely represent the ground based on $\boldsymbol{p}_q$, there are the following two potential options:

1. **Geometrical Approach:** Iterating through the (filtered) point cloud and determine whether the point falls outside our defined region based on its position ($\boldsymbol{p}_i = [x_i,\, y_i,\, z_i]^T$). One possible definition of a suitable region is a simple cuboid, depicted in Figure 4.2a. There are two possible options to determine whether a point lies within this cuboid. The

first one utilizes the four edge points ($\boldsymbol{p}_k$ with $k \in \{1\,,2\,,3\,,4\}$) that span up the cuboid. Let $\boldsymbol{d}_i = \boldsymbol{p}_i - \boldsymbol{p}_1$ be the vector from $\boldsymbol{p}_i$ to $\boldsymbol{p}_1$, and $\boldsymbol{u} = \boldsymbol{p}_2 - \boldsymbol{p}_1$, $\boldsymbol{v} = \boldsymbol{p}_3 - \boldsymbol{p}_1$, and $\boldsymbol{w} = \boldsymbol{p}_4 - \boldsymbol{p}_1$ be the three vectors that span up the cuboid, then a point $\boldsymbol{p}_i$ lies within the cuboid if all the following conditions are met:

$$0 \leq \boldsymbol{d}_i \cdot \boldsymbol{u} \leq \boldsymbol{u} \cdot \boldsymbol{u}\,, \quad 0 \leq \boldsymbol{d}_i \cdot \boldsymbol{v} \leq \boldsymbol{v} \cdot \boldsymbol{v}\,, \quad 0 \leq \boldsymbol{d}_i \cdot \boldsymbol{w} \leq \boldsymbol{w} \cdot \boldsymbol{w}\,. \tag{25}$$

The second option to determine whether a point $\boldsymbol{p}_i$ lies within the cuboid is based on the center point $\boldsymbol{p}_c$, the three normalized vectors defining the cuboid's orientation $\boldsymbol{d}_h$, $\boldsymbol{d}_l$, and $\boldsymbol{d}_w$ and the cuboid's length l, width w and height h. Let $\boldsymbol{d}_i = \boldsymbol{p}_i - \boldsymbol{p}_c$ be the vector from $\boldsymbol{p}_i$ to $\boldsymbol{p}_c$ then $\boldsymbol{p}_i$ lies within the cuboid if all the following conditions are met:

$$|\boldsymbol{d}_i \cdot \boldsymbol{d}_l| \leq \frac{l}{2}\,, \quad |\boldsymbol{d}_i \cdot \boldsymbol{d}_w| \leq \frac{w}{2}\,, \quad |\boldsymbol{d}_i \cdot \boldsymbol{d}_h| \leq \frac{h}{2}\,. \tag{26}$$

Of the two options, the second one with the center point is easier to apply, given that as $\boldsymbol{d}_h$ we take the normal vector of the previously found ground plane and the calculated query point $\boldsymbol{p}_q$ as $\boldsymbol{p}_c$. Then we need to find two orthogonal vectors to $\boldsymbol{d}_h$ as $\boldsymbol{d}_l$ and $\boldsymbol{d}_w$. These calculations require less computational effort than the four other points that span up the box. However, this cuboid shape has one main drawback: the orientation of the area spanned by $\boldsymbol{d}_l$ and $\boldsymbol{d}_w$ is either randomly determined by their calculation based on $\boldsymbol{d}_h$ or must be reasonably determined beforehand.

To avoid this drawback and to reduce the computational effort, a better suited shape is the cheese shape, a height limited sphere, depicted in Figure 4.2b. This shape is also defined by its center point $\boldsymbol{p}_c$, the normalized vector $\boldsymbol{d}_h$ which defines the orientation of the height h_c and its radius r_c. Let $\boldsymbol{d}_i = \boldsymbol{p}_i - \boldsymbol{p}_c$ be the vector from $\boldsymbol{p}_i$ to $\boldsymbol{p}_c$ then $\boldsymbol{p}_i$ lies within the cuboid if the following two conditions are met:

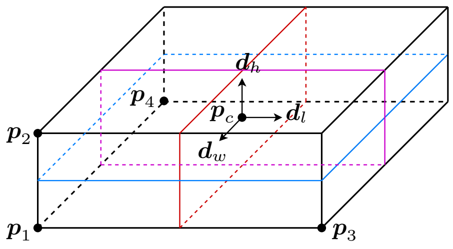
$$||\boldsymbol{d}_i||^2 \leq r_c^2\,, \quad |\boldsymbol{d}_i \cdot \boldsymbol{d}_h| \leq \frac{h_c}{2}\,. \tag{27}$$

Here we take the normal vector of the previously found ground plane as $\boldsymbol{d}_h$ and the calculated query point $\boldsymbol{p}_q$ as $\boldsymbol{p}_c$. This is the shape we select and implement for the geometrical sub-cloud creation.

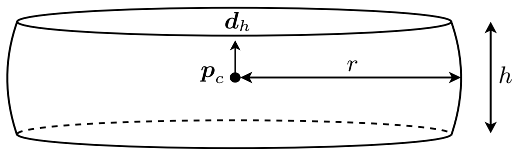
2. **kd-tree Approach:** Building a kd-tree from the (filtered) point cloud and then use the k-NNS function. As arguments, we pass an emperically predefined k and the calculated query point $\boldsymbol{p}_q$. For a ranged k-NNS we further pass the same radius r_c as defined for the cheese shape as the maximal range.

4.2 Normal Vector Calculation

While the two plane fitting algorithms (LSF and PCA) automatically provide the normal vector of the fitted plane, the plane detection algorithms (RANSAC and RHT) do not directly provide the normal vector of the found ground plane. For the RHT we estimate the parameters of the



(a) Cuboid shape. Defined by either the center point $\boldsymbol{p}_c$, the three vectors defining the cuboid's orientation $\boldsymbol{d}_h$, $\boldsymbol{d}_l$, and $\boldsymbol{d}_w$ and the cuboid's length, width and height, or by four edge points (here depicted as $\boldsymbol{p}_i$ with $i \in \{1, 2, 3, 4\}$ that span up the cuboid.



(b) Cheese shape (height limited sphere). Defined by the center point $\boldsymbol{p}_c$, the vector $\boldsymbol{d}_h$, which defines the orientation, the radius r of the sphere and the height h in $\boldsymbol{d}_h$ direction.

Figure 4.2: Possible forms for the sub-cloud geometrical approach.

ground plane by finding the peak in the accumulator space. Once we detect the parameters of the plane, we are able to directly extract the normal vector from these parameters. We calculate the normal vector $\boldsymbol{n} = [n_x, n_y, n_z]^T$ from the plane parameters ρ, θ, and φ as follows:

$$n_x = \cos(\theta)\sin(\varphi) \tag{28}$$
$$n_y = \sin(\theta)\sin(\varphi) \tag{29}$$
$$n_z = \cos(\varphi) \tag{30}$$

For a more explicit derivation of these equations, as well as for an explanation of the HT and the parameters refer to Section 3.3.3. RANSAC typically returns the inliers of the estimated model, that belong to the estimated plane. However, once we obtain the inliers of the ground plane, we are able to compute the normal vector using one of the plane fitting algorithms, e.g. LSF or PCA. Furthermore, it is theoretically possible to use HT for the normal calculation after RANSAC. However, plane detection algorithms are more time-consuming than plane fitting methods. Given the assumption that RANSAC already provides an accurate subset of points for the ground plane, further application of the HT, may not yield significant improvement in accuracy but definitely incurs additional computational overhead. Therefore, we consider the combination of the two plane detection algorithms for the normal vector calculation in real-time as unnecessary overhead compared to simpler methods like LSF and PCA. As a result, we do not further consider this approach for our work. It is possible to use a similar approach for RHT. In that case we apply a RHT with less accuracy to identify the inliers of the ground plane and then use a plane fitting algorithm to calculate the normal vector.

Figure 4.3 presents all suitable combinations of pre-processing steps and possible plane segmentation algorithms for the calculation of the ground plane's normal vector. With a total of six different pre-processing combinations (filtering: none, geometrical or kd-tree approach; sub-cloud: geometrical or kd-tree approach) and seven possible normal vector calculation options, the total number of possible combinations is 42. Based on the research presented in Section 2.2.1, PCA and LSF only show insignificant differences in accuracy and runtime. Therefore, to limit the extent of this work, we only implement PCA for the normal vector calculation after RANSAC and RHT based on [59], which declares PCA as the universal method of choice.

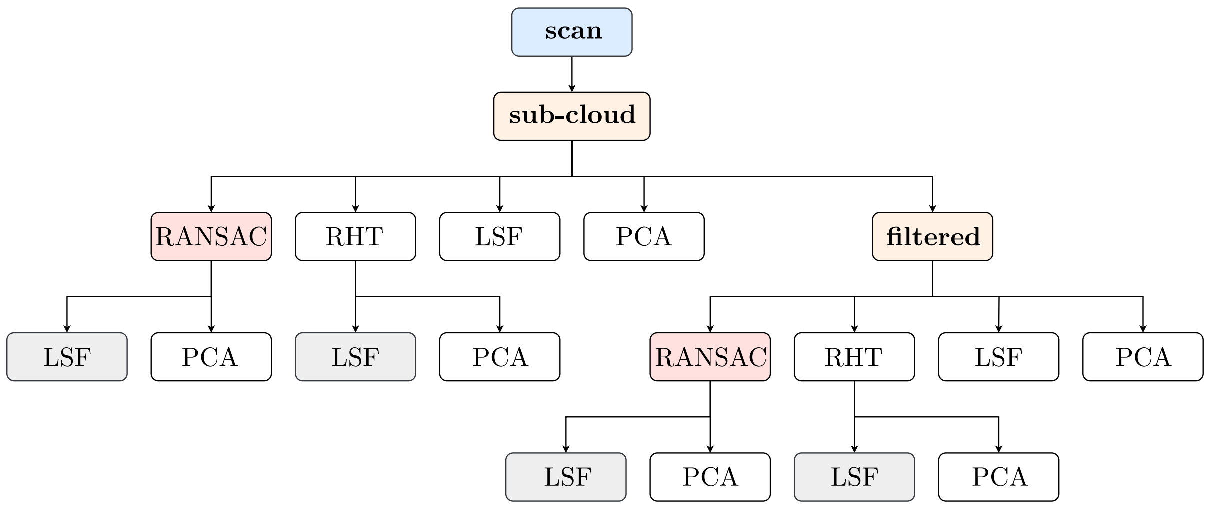


Figure 4.3: Flowchart of all suitable combinations of pre-processing steps and possible options for the calculation of the normal vector of the ground plane. Orange: pre-processing steps with either the kd-tree approach or the geometrical approach. White: Plane segmentation algorithms, which deliver the normal vector. Red: Plane segmentation algorithms, which do not deliver the normal vector and therefore require post-processing. Grey: not to be implemented.

This reduces the possible normal vector calculation options from seven to five options and leads to a total of 30 possible combinations.

4.3 Library Selection

In order to select the appropriate library for our application, we need to consider the following three fundamental aspects: the implementation of a suitable kd-tree and its search capabilities, as well as the support of plane segmentation algorithms and its compatibility with ROS. In the following two sections, we first consider suitable kd-tree libraries and their NNS, as well as downsampling capabilities, and second their support for plane segmentation algorithms.

4.3.1 Nearest Neighbor Search and Downsampling

As stated by [5] there are three different types of NNS queries: k-NNS, fixed radius search, and ranged k-NNS. The k-NNS retrieves the kNN around a specified query point, while the fixed radius search computes all points within a given radius of the query point. The ranged k-NNS is the combination of both, where it retrieves the kNN within a given distance. Table 4.3 lists suitable libraries and their search and downsampling properties for the filtering of the point cloud. We chose these libraries based on their superior performance regarding NNS, as described in Section 2.1.

pcl The pcl [23] is an open-source library designed for processing 2D/3D point cloud data. It offers a wide range of functionalities including point cloud filtering, segmentation, registration,

Table 4.3: Search and downsampling properties for suitable kd-tree libraries.

Library	k-NNS	Radius Search	Ranged k-NNS	Downsampling
pcl [23] (FLANN)	✓	✓	×	✓
3DTK [16]	✓	✓	✓	✓
libnabo [19]	✓	×	✓	×
nanoflann [20]	✓	✓	✓	×
ikd-tree [14]	✓	✓	✓	✓

feature extraction, and visualization. It is a popular choice for researchers and developers in fields such as robotics, computer vision, and augmented reality due to its extensive documentation. The pcl is easy to integrate into ROS applications and is often used in combination with ROS. Further, it has two implementations of a kd-tree: a normal and a FLANN version. Since the FLANN kd-tree is one of the superior versions, refer to Section 2.1, we only consider this implementation as a suitable choice.

3DTK The 3DTK [16], developed by the University of Würzburg, is a powerful open-source library tailored for processing and analyzing 3D point cloud data. It provides an extensive set of tools for point cloud registration, segmentation, feature extraction, and object recognition. It implements an optimized version of a kd-tree, which recursively partitions the points along the dimensions of maximum variance. In contrast to a conventional kd-tree, the search volumes of child nodes are improved by recalculating the bounding boxes around the included points, which is the strategy used in R-trees. This optimization increases search performance by enabling efficient search completion instead of traversing the entire kd-tree to reach leaf nodes for verification.

libnabo Libnabo [19] is a high-performance library for NNS in large datasets, written in ISO/IEC C++ 2003. It offers an efficient kd-tree implementation for spatial indexing, employing a sliding-midpoint rule during tree creation and a recursive descent algorithm during search. Unlike ANN [17], libnabo uses a compact vector of nodes instead of a tree of objects based on pointers, resulting in significant memory savings and enhanced performance, especially in large-scale applications.

nanoflann Nanoflann [20] is a lightweight, header-only open-source library designed for building fast kd-tree for NNS and other spatial indexing tasks. Its implementation focuses on simplicity and efficiency, making it suitable for embedding into various applications. Nanoflann constructs its kd-tree by recursively partitioning the data points along the dimensions of maximum variance, enabling fast nearest neighbor queries and range searches.

ikd-tree The ikd-tree [14] is a dynamic approach to constructing kd-trees without prior knowledge of the dataset size, making it suitable for real-time and streaming applications. As points are inserted, the tree grows incrementally by expanding leaf nodes along the dimension of maximum variance. This ensures balanced tree growth and efficient spatial organization. Deletion of points involves automatic rebalancing of the tree accordingly. Periodic rebalancing maintains

Table 4.4: Plane segmentation properties for suitable kd-tree libraries.

Library	LSF	PCA	RHT	RANSAC
pcl [23]	✓	✓	×	✓
3DTK [16]	✓	×	✓	✓

tree balance, preventing degeneration and preserving search efficiency. Overall, the ikd-tree offers a flexible solution for dynamic spatial indexing, enabling fast NNS in evolving datasets. The implementation of the ikd-tree is template based and is easily compatible with the pcl without time-consuming conversions.

4.3.2 Plane Segmentation

In order to get the normal vector of the ground plane, we need to segment the ground plane from the point cloud. The most popular techniques for extracting planes from 3D point clouds are LSF, PCA, 3D HT and the RANSAC algorithm [25, 42–45]. Of the selected libraries from Section 4.3.1 only the pcl and the 3DTK include plane segmentation algorithms. Table 4.4 lists the available plane segmentation algorithms for these two libraries. Nanoflann and libnabo are both only intended for NNS using a kd-tree. Thus, they require unnecessary data conversions, which are time-consuming and therefore undesirable, and neither supports downsampling the data. In addition, libnabo shows comparable performance to 3DTK for artificial data or even worse for real data due to these time-consuming conversions as stated by [5]. Further, the ikd-tree outperforms nanoflann in its search capabilities, as noted in [15]. Since the ikd-tree is compatible with the pcl, we ensure efficient performance without unwanted data conversions between the filter and the ground plane segmentation step. Consequently, the suitable libraries for our application are pcl, 3DTK and the ikd-tree.

We chose the pcl since it fulfills all our requirements except for the RHT and ranged k-NNS. We implement the RHT based on 3DTK and for the processing of one scan the standard k-NNS is sufficient, since we are dealing with small point clouds, and adding the ranged search is too demanding for the scope of this work due to pcl's complex codebase and its high level of abstraction. Additionally, it allows for the seamless integration of the ikd-tree library and is further the recommended choice for ROS applications, making it the suitable library for our project's needs.

4.3.3 Performance of the Ikd-Tree vs. PCL Kd-Tree

In order to determine the utility of the ikd-tree library, which is newer and less commonly used than the pcl, we conduct a preliminary performance comparison. This comparison is based on two test scripts, one utilizing the ikd-tree library and the other the pcl. Each test script subscribes to the laser scan topic. The callback function then filters the point cloud and creates the sub-cloud. It is important to note that the filtering process does not include downsampling yet. Based on the previous strategies for point cloud pre-processing, we get the following options in which the kd-tree is used:

Table 4.5: Preliminary Performance Evaluation of the **ikd-tree**. Average runtimes for five different scenarios ($000 - 004$). Top: results for filtering using kd-tree and bottom: results for filtering using the geometric approach. Coverage is the number of scan callbacks/total ROS messages published on LiDAR topic.

	Step	000	001	002	003	004	Avg.
kd-tree	**Build Tree [ms]**	91.032	90.041	89.745	93.563	96.123	**92.101**
	Radius Search [ms]	2.328	1.818	2.127	1.997	2.261	**2.106**
	Delete [ms]	138.237	113.309	129.279	127.335	149.311	**131.494**
	k-NNS [ms]	1.699	1.586	1.798	1.848	1.922	**1.771**
	Total [ms]	233.296	206.754	222.949	224.743	249.616	**227.471**
	Coverage [%]	17.442	19.283	17.608	17.610	16.187	**17.626**
Geometric	**Radius Search [ms]**	6.086	5.875	6.027	6.866	6.695	**6.310**
	Delete [ms]	0.979	0.818	0.784	0.829	0.848	**0.852**
	Build Tree [ms]	47.718	59.871	48.730	54.867	48.739	**51.985**
	k-NNS [ms]	1.095	1.128	1.071	1.298	1.150	**1.148**
	Total [ms]	55.878	67.692	56.611	63.860	57.432	**60.295**
	Coverage [%]	56.977	50.224	58.472	52.358	55.830	**54.772**

1. **Filtering:** geometrical, **sub-cloud creation:** using kd-tree and k-NNS;
2. **Filtering:** using kd-tree and radius search, **sub-cloud creation:** geometrical;
3. **Filtering:** using kd-tree and radius search, **sub-cloud creation:** using kd-tree and k-NNS.

For this preliminary performance comparison it is sufficient to examine the first and the last option listed above. This is due to the fact that the second option contains the same information about the build and radius search time as the third option. All test scripts are available on GitLab within the source folder of the `ground_finder` node, refer to [1]. For this comparison, we record five different scenarios with the prototype, all of which were captured in the hallway of the IT building at the University Würzburg. As we only look at the runtime evaluation the movement of the prototype plays no role for this evaluation.

We run the test scripts for each scenario on a Lenovo IdeaPad S540, equipped with a 1.80 GHz Intel Core i7-8565U CPU and 12 GB of RAM. We set the parameters as follows: $r_{fil} = 0.35\,\text{m}$ (considering a sphere radius of 0.145 m plus noise and the hands - empirically determined in rviz), $k = 1000$, $d_{kNN} = 2.0\,\text{m}$ (for ranged k-NNS), and for the ikd-tree we set $\alpha_{bal} = 0.6$, $\alpha_{del} = 0.5$ and the box length $l_{box} = 0.2$ based on [14]. Changes in the parameters α_{bal} and α_{del}, which influence rebalancing, result in insignificant variations regarding the runtime performance of the ikd-tree's delete operation. The results of these tests are summarized in Table 4.5 and Table 4.6.

From the results, we gain the important insight that the ikd-tree is not suitable for our real-time requirements. This conclusion is based on the runtime performance of building the

Table 4.6: Preliminary Performance Evaluation of the **pcl kd-tree**. Average runtimes for five different scenarios (000 – 004). Top: results for filtering using kd-tree and bottom: results for filtering using the geometric approach. Coverage is the number of scan callbacks/total ROS messages published on LiDAR topic.

	Step	**000**	**001**	**002**	**003**	**004**	**Avg.**
kd-tree	**Build Tree 1 [ms]**	10.892	10.084	10.540	10.527	10.282	**10.465**
	Radius Search [ms]	2.353	1.717	2.251	2.152	2.369	**2.168**
	Delete [ms]	1.625	1.323	1.593	1.549	1.607	**1.539**
	Build Tree 2 [ms]	4.441	5.258	4.579	4.666	4.285	**4.646**
	k-NNS [ms]	0.178	0.175	0.207	0.227	0.203	**0.198**
	Total [ms]	19.489	18.556	19.171	19.121	18.745	**19.017**
	Coverage [%]	97.674	100.000	100.000	99.686	100.000	**99.472**
Geometric	**Radius Search [ms]**	9.220	9.468	9.290	9.516	8.465	**9.192**
	Delete [ms]	0.948	0.964	1.011	0.967	0.985	**0.975**
	Build Tree [ms]	5.516	6.714	5.563	5.828	5.754	**5.875**
	k-NNS [ms]	0.204	0.206	0.238	0.270	0.258	**0.235**
	Total [ms]	15.888	17.351	16.102	16.581	15.462	**16.277**
	Coverage [%]	98.837	100.000	100.000	100.000	100.000	**99.767**

ikd-tree and the incremental deletion of points after filtering with the associated rebalancing. Building the ikd-tree takes 51.985 ms after geometric filtering and 92.101 ms before filtering. In contrast, building the pcl kd-tree takes 10.465 ms before filtering, 4.646 ms after geometric filtering, and 5.875 ms after kd-tree filtering, making it more than 8.8 times faster in all cases. Additionally, the incremental deletion of points with the ikd-tree takes an average of 131.494 ms, whereas deleting points with pcl extract takes less than 1.2 % of the time ($<$ 1.539 ms). This inefficiency in the ikd-tree deletion process is due to its point-wise deletion method, which is not index-based and is thus unsuitable for real-time applications. With these runtimes, the ikd-tree library only covers 17.626 % of all incoming laser scans for the kd-tree filtering and 54.772 % for the geometrical filtering, whereas the pcl achieves a coverage of more than 99 % for both options. Further, from the results we observe, that the ranged k-NNS of the ikd-tree library performs with a runtime of more than 1.00 ms worse than the pcl k-NNS, which runs in less than 0.25 ms. This confirms the findings in [14]. In conclusion, we choose the pcl for the real-time calculation of the ground plane normal vector.

Chapter 5

Experiments and Evaluation

In this chapter, we conduct experiments to compare the selected plane segmentation algorithms under different pre-processing combinations. We focus on their performance in terms of runtime, accuracy and robustness. This detailed evaluation aims to determine suitable methods to compute the normal vector of the ground plane in real-time for the spherical mobile mapping system. We measure runtime as the total processing time for each approach using the system's high-resolution clock to determine real-time capability, which we define as a runtime faster than 50 ms. We assess accuracy through three key metrics: Mean Absolute Error (MAE), Median Absolute Error (MedAE), and result variance, which quantify the precision of plane segmentation. To evaluate robustness, we determine the error rate by dividing the number of plane segmentation errors by the total number of calls to callback functions triggered by ROS topic messages. These metrics collectively provide a comprehensive assessment of each approach's performance.

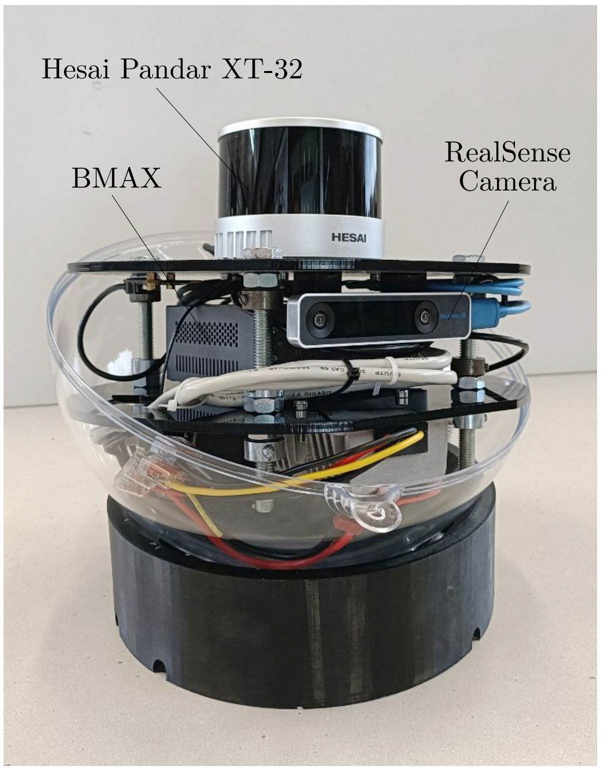
We first provide an overview of the technical setup for the experiments. This is followed by the performance of the remote experiments and, subsequently, their comparative analysis. After that, we present the on-board runtime experiment and its evaluation. Finally, we summarize and discuss the results of the experiments.

5.1 Technical Setup

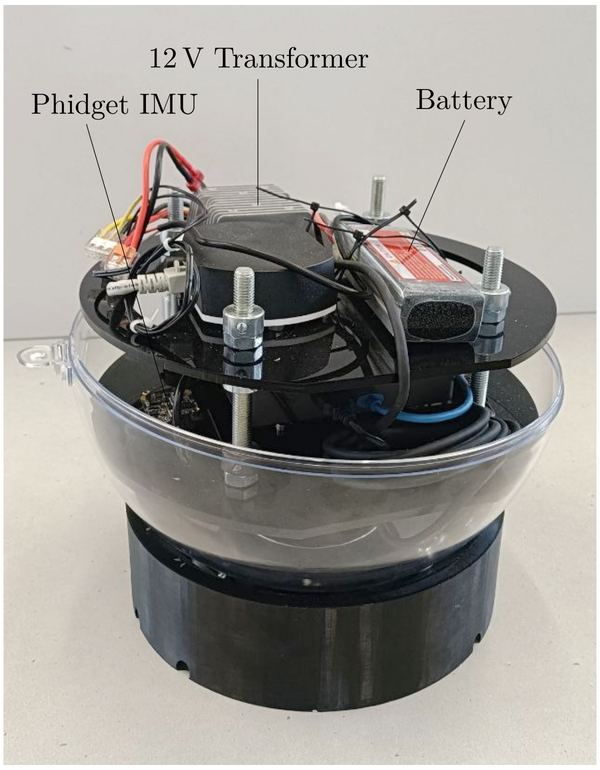
In this section, we introduce the technical setup of our experiments. First, we introduce the spherical mobile mapping system and its components relevant to our work. Following that, we present the new ROS node's implementation, called the `ground_finder_node`. In the final section, we present the determination of the ground truth used to evaluate our results and assess the accuracy of the different approaches.

5.1.1 Spherical Mobile Mapping System

The spherical mobile mapping system, the current DAEDALUS prototype, is equipped with a clear perspex shell, which not only protects its internal components but also allows visibility of its intricate structure. The shell is transparent to ensure that the sensors, particularly the RealSense camera and the LiDAR, are able to effectively penetrate it without interference.



(a) The middle and top layer of the robot, showcasing the LiDAR, on-board computer (BMAX), and the RealSense camera.



(b) The bottom layer of the robot, featuring the power supply system, including the 12 V transformer, battery and one of the three Phidget IMUs.

Figure 5.1: Overview of the spherical mobile mapping system's internal components with the open perspex shell.

Inside, the robot is organized into multiple layers, each dedicated to specific functions. The top layer houses only the Hesai Pandar LiDAR. The middle layer serves as the main platform for essential sensors and processing units, including the on-board computer, inertial measurement units and RealSense camera. The bottom layer contains the power supply system, featuring a 12 V transformer and battery, ensuring a stable energy source for the robot's operations.

LiDAR The spherical mobile mapping system features the Pandar XT-32 LiDAR, developed by Hesai Technology [117]. This sensor features 32 laser channels and operates at a wavelength of 905 nm, classified as Class 1 Eye Safe. The instrument range is between 0.05 and 120 m, with a typical range accuracy of ± 1 cm and a range precision of 0.5 cm. The sensor boasts a data acquisition rate of up to 1,200,000 points/s and supports a frame rate of up to 20 Hz. We configure the Pandar XT-32 to operate at 20 Hz and utilize single return mode, capturing the strongest signal. The horizontal FOV is $360\,^\circ$ with a resolution of $0.36\,^\circ$ at 20 Hz, while the vertical FOV is $31\,^\circ$ with a vertical resolution of $1\,^\circ$. The Pandar XT-32 operates at a voltage range of DC 9 to 36 V, with a typical power consumption of 10 W, and weighs 0.8 kg. The design utilizes mechanical rotation for scanning.

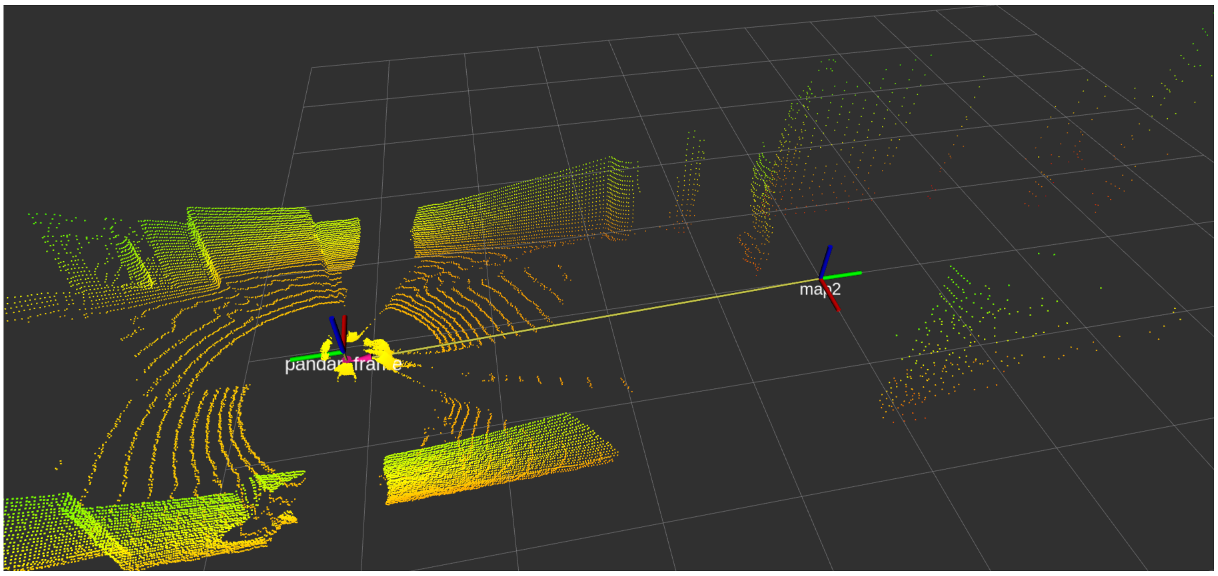


Figure 5.2: Screenshot of a Hesai Pandar LiDAR scan in a hallway including the display of the local and the world coordinate system. Right: `map2` frame (world). Left: `pandar_frame` (local). The transformation between those two TF frames is displayed as a yellow line.

On-Board Computer The on-board computer of the spherical mobile mapping system is the MaxMini B3 Plus from BMAX [118]. It is equipped with an Intel Pentium Gold 5405U processor operating at a base frequency of 2.3 GHz and an integrated Intel 9th Gen UHD Graphics 610. The system has 8 GB RAM and a 256 GB SSD. There are multiple connectivity options, including a 1000 Mbps LAN port, WiFi and Bluetooth 5.0. The computer has multiple USB ports (USB 3.0 $\times$ 2, USB 2.0 $\times$ 2), a DC 12 V/2 A power input, a 3.5 mm headphone jack, HDMI, MiniDP, a Type-C port (full functionality), and an RJ45 port. Weighing approximately 450 g, the MaxMini B3 Plus is a lightweight yet formidable component that is ideal for the dynamic operating requirements of the spherical mobile mapping system.

Software The ROS runs on the spherical mobile mapping system's on-board computer. Several nodes have already been implemented for tasks such as Simultaneous Localization and Mapping (SLAM), reading data from other sensors such as the RealSense camera, or running a Kalman filter [4] for sensor fusion. However, the most important node for our application regarding LiDAR data processing performs inertial LiDAR motion distortion correction to publish undistorted LiDAR data to ensure accurate and reliable perception for navigation and mapping tasks. This node corrects the LiDAR measurements by compensating for the robot's movements, producing spatial data with high accuracy, refer to [119]. In ROS, the framework uses a system of coordinate frames organized in a TF tree to manage spatial relationships between different data sources and reference points. For this work, the following two TF frames are relevant: the local frame `pandar_frame`, in which the LiDAR data is received, and the global frame `map2`, which serves as the world coordinate system and has its origin in the robot's initial position. Refer to Figure 5.2 for a visual representation of the two frames in rviz.

5.1.2 Implementation

The implemented ROS node named `ground_finder_node` analyzes the LiDAR data to calculate the ground plane's normal vector. All processing is performed single-threaded. The code is open-source and available on GitLab [1]. The main function creates an object of the `ground_finder` class, which is initialized with the parameters passed to the launch file. These parameters select the plane segmentation algorithm (LSF, PCA, RANSAC, RHT or RHT in combination with PCA, which we define as RHT2 for the scope of this work), the filter method (none, geometric or kd-tree) and the option for the sub-cloud creation (geometric or kd-tree). In addition, it supports optional parameters for quiet operation and file saving, where the resulting normal vectors and processing times are logged in CSV files for further evaluation. The core functionality of the `ground_finder` object is managed by a callback function, that subscribes to the undistorted LiDAR points from the ROS topic `/lidar/points_undistorted`. See Figure 5.3 for a visual representation of its workflow. It first converts the received sensor messages into pcl format, followed by the pre-processing of the point cloud according to the selected filter and sub-cloud parameters. The implementation of these functions follows the procedures detailed in Section 4.1.2. However, for the geometric filtering and sub-cloud extraction, we use a single combined function. This approach replaces the need for separate functions for each task, allowing us to process the entire point cloud in one loop instead of using two distinct ones. After the pre-processing, the function calculates the normal vector of the sub-cloud using the selected plane segmentation algorithm, as described in Section 4.2, and transforms it from the `pandar_frame` into the `map2` frame. This allows us to verify whether we successfully identified the ground plane. For plane fitting, we make a single attempt to detect the ground plane. If it fails, we immediately throw a plane segmentation error. In contrast, for plane detection algorithms such as RANSAC, RHT, and RHT2, we perform multiple attempts. If we do not identify the ground plane, we delete the points from the falsely identified plane and retry up to the maximum number of iterations (defined by the `max_iterations` parameter in Table 5.1). Only after exhausting these attempts without detecting the ground plane do we throw a plane segmentation error. If we recognize the ground plane successfully, the callback function publishes the normal vector in the `map2` frame.

Verification of the Ground Plane's Detection To determine whether we identified the ground plane, we assume that the inclination of the ground plane does not exceed a specified maximum angle, α_{max}. For this work, we set this maximum inclination angle α_{max} to $45\,°$. After converting the normal vector of the detected plane to the `map2` frame, we calculate the dot product of this normal vector with the downward vector $\boldsymbol{v}_{down} = [0, 0, -1]^T$. If the absolute value of the dot product is less than the specified threshold, which is defined by the cosine of the maximum allowable angle ($\cos(\alpha_{max}) =$ `wall_thresh`), the detected plane is classified as a wall. Otherwise, we confirm that the detected plane is the ground plane.

Parameters Used in the Implementation Table 5.1 provides an overview of the general parameters used for the pre-processing and plane segmentation in our implementation. It includes settings relevant to filtering methods and sub-cloud configurations, as well as relevant to the identification of the ground plane. The second table, Table 5.2, lists the specific parameters

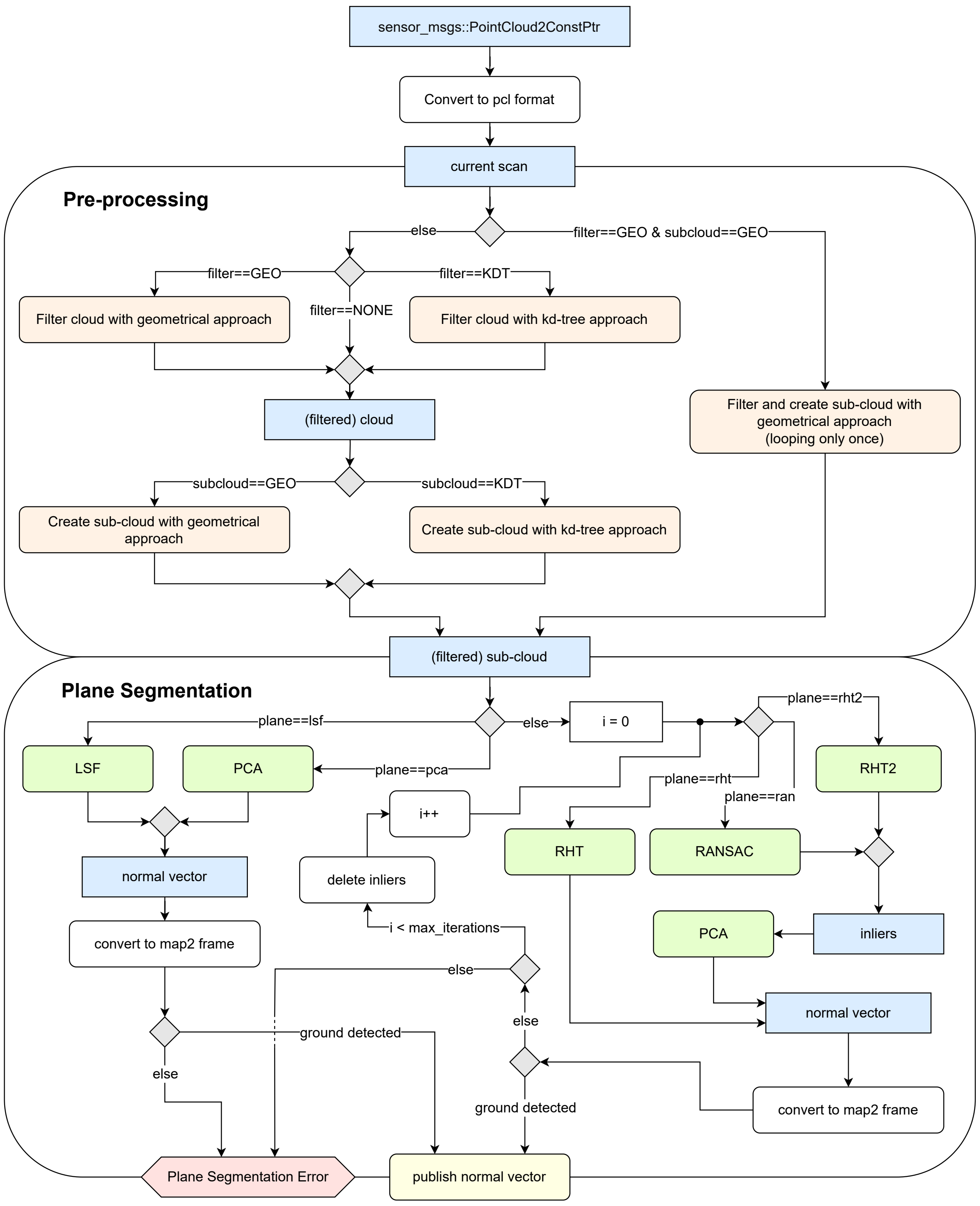


Figure 5.3: Schematic of the `ground_finder`'s LiDAR scan callback function displayed systematically from top to bottom with the initial message at the top. It is divided into two main parts: the pre-processing and the plane segmentation. Each rounded rectangle depicts an action node and each rectangle an object node. Blue: Important objects. Orange: Pre-processing functions. Green: Plane segmentation algorithms. Red: Error. Yellow: Publishing of the final result. Grey: Decision and merge nodes.

Table 5.1: Overview of general parameters for pre-processing and plane segmentation used in the implementation of the `ground_finder` class.

Name	Symbol	Value	Description
Pre-Processing			
`radius_filter`	r_{fil}	0.35	Radius of the spherical mobile mapping system + noise + hands (empirically determined with rviz) used for filtering [m]
`ds_size`	d_{ds}	0.10	Leaf size used for downsampling, such that the leaf is $d_{ds} \times d_{ds} \times d_{ds}$ big [m]
`radius_sphere`	r_s	0.145	radius of the spherical mobile mapping system used for sub-cloud query point calculation [m]
`radius_subcloud`	r_c	2.0	Radius of cheese used for geometrical sub-cloud approach [m]
`height_subcloud`	h_c	0.2	Height of cheese used for geometrical sub-cloud approach [m]
`k`	k	150	Number of points used for k-NNS for kd-tree sub-cloud approach
Plane Segmentation General			
`max_iterations`		3	Maximal number of tries for plane detection algorthims (RANSAC, RHT and RHT2)
`wall_thresh`		0.707	Threshold used to determine when we recognized a wall; $\cos(\alpha_{max})$ with $\alpha_{max} = 45\,^\circ$
`inlier_plane`		0.05	Threshold used to determine when a point counts as an inlier to the detected plane; Used for RHT to delete points and for RHT2 to pass as argument to PCA, as well as deletion of inliers

for the plane detection algorithms (RANSAC, RHT and RHT2).

5.1.3 Ground Truth

To establish the ground truth for our experiments, we start by assuming that all flat surfaces inside buildings are "leveled", meaning their inclination is considered to be $0\,^\circ$ and therefore the normal vector to be $\boldsymbol{n}_{gt} = [0, 0, -1]^T$. We verified this assumption approximately for the selected rooms using a water scale. For more complex scenarios, we utilize the Riegl VZ-400 laser scanner [98]. This high-precision LiDAR is renowned for its accuracy, with a measurement rate of up to 122,000 points/s, an angular measurement resolution of $0.0005\,^\circ$ and a range accuracy of ± 5 mm at 100 m. These characteristics make it ideal for capturing detailed 3D measurements of various surfaces and for the establishment of the ground truth for this work. The process of establishing the ground truth with the Riegl VZ-400 involves several steps. First, we perform an

Table 5.2: Overview of parameters for each plane detection algorithm (RANSAC, RHT and RHT2) used in the implementation of the `ground_finder` class. The parameters of RHT2 are only the changed parameters compared to RHT.

Name	Value	Description
RANSAC		
`dist_thresh`	0.01	Maximum allowable distance from the model to a point for it to be considered an inlier [m]
`max_iterations_`	10,000	Default value from parent class and not used in RANSAC, refer to `probability_`
`probability_`	0.99	Confidence level for certainty of the final model's accuracy and used to calculate the required number of iterations based on the proportion of inliers in the dataset and the desired confidence level (default value)
RHT		
`rhoNum`	7	Number of discrete cells for the distance (ρ) from the origin to the plane along the plane's normal vector - Accumulator size
`phiNum`	90	Number of discrete cells for the angle (φ), which is the angle between the plane's normal vector and the positive z-axis - Accumulator size
`thetaNum`	180	Number of discrete cells for the angle (θ), which is the azimuth angle of the plane's normal vector around the z-axis - Accumulator size
`rhoMax`	5	Maximum distance from plane to origin of frame ρ_{max} [m]
`accumulatorMax`	10	Threshold for determining when a dominant plane has been identified
`minDist`	0	Minimum distance required between the randomly selected points
`maxDist`	1.797×10^{308}	Maximum distance required between the randomly selected points (`std::numeric_limits<double>::max()`)
`stop`	10,000	Maximum number of iterations allowed for RHT
RHT2		
`phiNum`	36	Number of discrete cells for the angle (φ) - Accumulator size
`thetaNum`	72	Number of discrete cells for the angle (θ) - Accumulator size

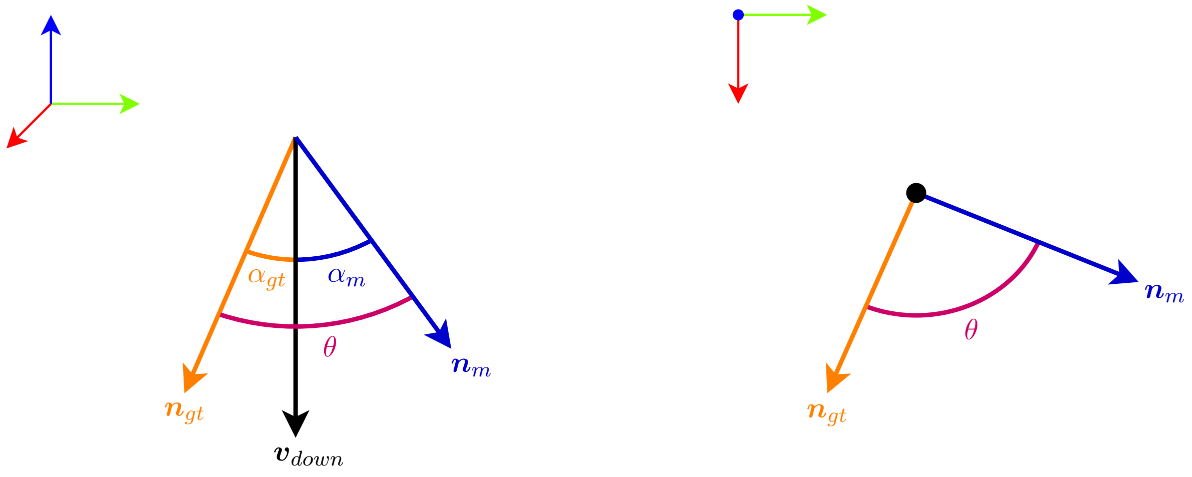


Figure 5.4: Calculation of the absolute angle error $|\alpha_{gt} - \alpha_m|$. The down vector $\boldsymbol{v}_{down} = [0, 0, -1]^T$ denotes the direction to the center of the earth (in gravitational direction), $\boldsymbol{n}_{gt}$ is the measured ground truth vector of the ground plane with its inclination angle α_{gt} (orange) and $\boldsymbol{n}_m$ is the measured vector of the ground plane with its inclination angle α_m (blue). The angle between $\boldsymbol{n}_{gt}$ and $\boldsymbol{n}_m$ is denoted as θ and is affected by the vectors' orientation in the x-y-plane. Left: Cabinet projection. Right: Top view.

initial scan to create a detailed point cloud representation of the environment. We then use the 3DTK software to calculate the normal vectors of the scanned surfaces, using kNN as the normal calculation method with k set to 20. Next, we select seven points from each ground plane of the experiment's scenario with their computed normal vectors. Then we convert the normal vectors from the 3DTK coordinate system to our `map2` coordinate system, where the y-axis in 3DTK, which represents the up direction, becomes the z-axis in `map2`, and the z-axis in 3DTK becomes the y-axis in `map2`. This ensures that the left-handed coordinate system of 3DTK is converted to the right-handed coordinate system of `map2`. We then ensure that all normal vectors point downwards (negative z-component). It is important to note that there is an unknown rotation about the z-axis between the 3DTK coordinate system and the `map2` frame. The determination of this rotation angle is complex and time-consuming. To avoid this, we use the inclination angles α_{gt} as ground truth rather than the normal vectors themselves, as this unknown yaw rotation leads to significant errors in accuracy determination, refer to Figure 5.4. Finally, we calculate the mean inclination of the selected points' normal vectors, which serves as the ground truth for the selected ground planes. We verify the Riegl's ground truth once in Section 5.2.4 by measuring the ramp's inclination using a laser distance meter, which determines the height and length of the ramp, and cross-checking it against the inclination values obtained from the Riegl measurements.

5.2 Offline Experiments

In this section, we perform eight experiments across different scenarios to evaluate 30 different methods for the normal vector calculation, which derive from combining the six pre-processing approaches with the selected five plane segmentation algorithms. We assess their performance with regard to runtime, accuracy, and robustness. Specifically, we evaluate the robustness by the fail rate, defined as the ratio of plane segmentation errors to the total number of callback function calls triggered by ROS topic `/lidar/points_undistorted` messages. We design each experiment to highlight specific key aspects such as limited area, varying inclinations, and different movement speeds. A Lenovo IdeaPad S540, equipped with a 1.80 GHz Intel Core i7-8565U CPU and 12 GB of RAM, remotely processes the data recorded during these experiments. These experiments focus primarily on accuracy and robustness, while providing a comparative analysis of the runtime across the different approaches. We perform all experiments on the Campus Hubland Süd of the Julius-Maximilians University Würzburg.

5.2.1 Experiment A: Hallway

We conduct the first experiment in the hallway of the IT building between the offices on the first floor. This scenario offers a virtually unlimited area in one axis of the ground plane and is limited to the width of the hallway in the other. The hallway's width varies, measuring 2.71 m at the starting position and narrowing to 2.42 m at the turning point. In addition, this location features a flat floor. This allows the evaluation of robustness and accuracy over time, without a change in inclination. When conducting this experiment, we roll the spherical mobile mapping system slowly as far as possible in the middle of the hallway approximately 5.2 m forward and then back to the starting position. We try to ensure that the sphere moves as smoothly as possible.

Ground Truth As described in the Section 5.1.3 on ground truth, we assume that the ground plane has an inclination of 0 °, with the corresponding normal vector of the ground truth set to $\boldsymbol{n}_{gt} = [0, 0, -1]^T$.

Results We present the runtime and plane segmentation error rate (fail rate) results for each approach in Table 5.3, and we summarize the accuracy results in Table 5.4. From the results, we see that the approaches without filtering perform worst in terms of runtime for all plane segmentation algorithms, i.e, none-geo with 20.137 ms and none-kdt with 15.327 ms. With the none-kdt approach, this is mainly due to the construction of the kd-trees and with the none-geo approach due to the plane segmentation. The geo-geo approach performs the fastest with an average runtime of 8.162 ms. Nevertheless, all approaches have a runtime lower than 50 ms and therefore are able to operate in real-time. RHT has the lowest MAE for the kd-tree sub-cloud approaches (geo-kdt and kdt-kdt) with roughly 7.9 °. With the low variance of roughly 301.0 $(°)^2$ and the MedAE of 6.3 °, these two combinations prove to be the most accurate for the "hallway" experiment. Despite the low MedAE for the combinations: LSF, PCA and RHT2 with pre-processing geo-kdt or kdt-kdt, as well as RHT2 for none-kdt, all have a significantly higher MAE and variance compared to RHT. The only comparable accuracy is provided by the geometrical sub-cloud approaches with RANSAC with a MAE of approximately 8.7 °, a MedAE of 6.9 ° and a variance of 373.5 $(°)^2$. Overall, the none-geo approach has the lowest fail rate, with an average below 2.8 %, while RANSAC consistently shows the lowest fail rate among the plane segmentation algorithms, with a maximum fail rate of 25.913 % for none-kdt. Considering the combination of runtime, accuracy, and robustness, the best overall solution is RANSAC with geo-geo pre-processing. Figure 5.5 displays the absolute angle error over the number of calls to the callback function for RANSAC with geo-geo pre-processing.

Table 5.3: Runtime and fail rate results of experiment ”hallway”. Runtimes [ms] of each step of the different pre-processing combinations for each plane segmentation algorithm, as well as the total runtime. The fail rate indicates the percentage of plane segmentation errors.

	Step	**LSF**	**PCA**	**RAN**	**RHT**	**RHT2**	**Avg.**
none-geo	**Cheese Search [ms]**	4.735	2.406	2.703	5.712	2.639	**3.639**
	Delete [ms]	0.830	0.426	0.474	1.005	0.466	**0.640**
	Plane [ms]	8.621	24.426	21.858	3.375	21.008	**15.858**
	Total [ms]	14.186	27.258	25.035	10.092	24.113	**20.137**
	Fail Rate [%]	3.130	3.391	1.478	1.739	4.261	**2.800**
none-kdt	**Build Tree [ms]**	13.358	14.029	13.352	11.482	12.553	**12.955**
	k-NNS [ms]	0.146	0.149	0.132	0.138	0.147	**0.142**
	Delete [ms]	0.724	0.755	0.719	0.648	0.686	**0.706**
	Plane [ms]	0.235	1.144	1.166	2.932	2.141	**1.523**
	Total [ms]	14.463	16.077	15.370	15.200	15.526	**15.327**
	Fail Rate [%]	35.043	35.043	25.913	43.478	48.000	**37.496**
kdt-geo	**Downsample [ms]**	5.731	5.545	5.481	5.093	5.371	**5.444**
	Build Tree [ms]	1.490	1.452	1.417	1.332	1.408	**1.420**
	Radius Search [ms]	0.029	0.027	0.027	0.024	0.026	**0.027**
	Delete [ms]	0.204	0.198	0.193	0.182	0.191	**0.194**
	Cheese Search [ms]	0.448	0.436	0.430	0.404	0.423	**0.428**
	Delete [ms]	0.111	0.109	0.107	0.101	0.105	**0.107**
	Plane [ms]	0.348	1.607	1.878	2.853	1.816	**1.700**
	Total [ms]	8.361	9.374	9.532	9.988	9.341	**9.319**
	Fail Rate [%]	5.304	5.304	1.391	2.522	6.696	**4.243**
geo-geo	**Downsample [ms]**	6.094	5.919	5.847	5.227	5.639	**5.745**
	Cheese Search [ms]	0.538	0.519	0.510	0.467	0.501	**0.507**
	Delete [ms]	0.128	0.123	0.122	0.111	0.119	**0.120**
	Plane [ms]	0.375	1.727	2.016	2.884	1.949	**1.790**
	Total [ms]	7.135	8.288	8.495	8.689	8.208	**8.163**
	Fail Rate [%]	5.304	5.304	1.391	3.304	6.783	**4.417**

Table 5.3: Runtime and fail rate results of experiment "hallway". Runtimes [ms] for each step of the different pre-processing combinations for each plane segmentation algorithm, as well as the total runtime. The fail rate indicates the percentage of plane segmentation errors. (continued)

	Step	**LSF**	**PCA**	**RAN**	**RHT**	**RHT2**	**Avg.**
geo-kdt	**Downsample [ms]**	5.865	5.635	5.314	4.316	5.265	**5.279**
	Radius Search [ms]	0.180	0.175	0.148	0.119	0.150	**0.155**
	Delete [ms]	0.233	0.227	0.187	0.158	0.196	**0.200**
	Build Tree [ms]	1.739	1.685	1.358	1.150	1.441	**1.474**
	k-NNS [ms]	0.058	0.056	0.053	0.044	0.052	**0.052**
	Delete [ms]	0.131	0.128	0.105	0.089	0.110	**0.113**
	Plane [ms]	0.358	1.654	3.121	5.454	2.915	**2.700**
	Total [ms]	8.564	9.558	10.285	11.329	10.130	**9.973**
	Fail Rate [%]	27.913	28.348	1.913	6.870	21.565	**17.322**
kdt-kdt	**Downsample [ms]**	5.654	5.449	5.122	4.131	5.084	**5.088**
	Build Tree [ms]	1.700	1.656	1.333	1.116	1.421	**1.445**
	Radius Search [ms]	0.027	0.027	0.024	0.020	0.025	**0.025**
	Delete [ms]	0.226	0.222	0.182	0.151	0.190	**0.194**
	Build Tree [ms]	1.619	1.584	1.274	1.080	1.363	**1.384**
	k-NNS [ms]	0.050	0.050	0.047	0.039	0.047	**0.046**
	Delete [ms]	0.123	0.121	0.101	0.085	0.105	**0.107**
	Plane [ms]	0.338	1.567	3.016	5.228	2.672	**2.564**
	Total [ms]	9.737	10.676	11.099	11.849	10.906	**10.853**
	Fail Rate [%]	28.000	28.435	1.913	6.870	23.913	**17.826**

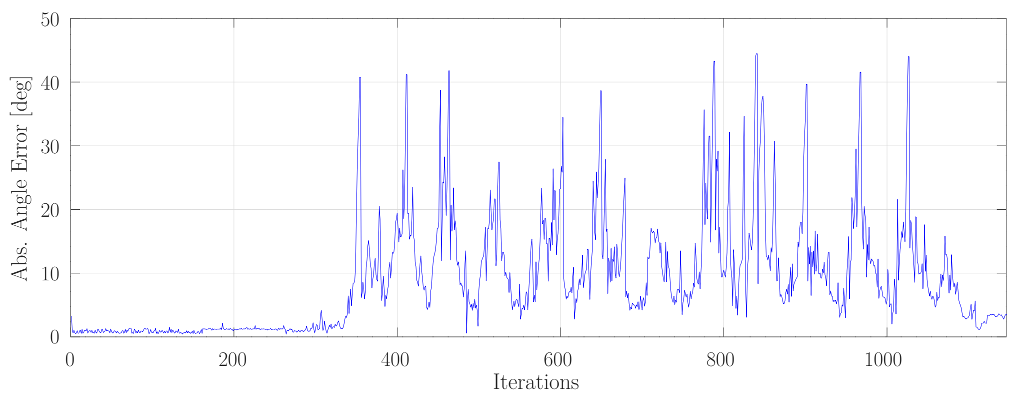


Figure 5.5: Experiment A: Plot of the absolute angle error over iterations, i.e. the number of calls to the callback function, for RANSAC using geo-geo pre-processing. Red lines indicate a change of ground plane and thereby inclination.

Table 5.4: MAE [deg], MedAE [deg] and result variance [deg^2] of experiment ”hallway” for the different plane segmentation algorithms under different pre-processing combinations.

	Result	LSF	PCA	RAN	RHT	RHT2
none-geo	**MAE [deg]**	16.556	17.148	11.676	12.500	17.589
	MedAE [deg]	15.592	16.956	8.615	8.781	16.508
	Variance [deg^2]	1280.265	1368.975	650.148	750.006	1418.717
none-kdt	**MAE [deg]**	15.785	15.901	17.004	13.400	12.355
	MedAE [deg]	10.414	9.229	13.506	7.678	3.907
	Variance [deg^2]	1228.475	1254.008	1341.362	882.512	803.786
kdt-geo	**MAE [deg]**	16.091	16.110	8.728	12.412	16.885
	MedAE [deg]	12.545	12.557	6.946	8.084	14.932
	Variance [deg^2]	1234.723	1236.845	374.021	763.528	1347.197
geo-geo	**MAE [deg]**	16.112	16.182	8.717	12.580	16.402
	MedAE [deg]	12.736	13.228	6.855	8.224	13.886
	Variance [deg^2]	1236.873	1246.175	373.142	790.136	1283.571
geo-kdt	**MAE [deg]**	11.051	10.875	12.664	7.986	11.560
	MedAE [deg]	5.980	5.965	8.205	6.343	6.753
	Variance [deg^2]	602.196	582.897	805.446	300.800	650.288
kdt-kdt	**MAE [deg]**	10.993	10.852	12.749	7.990	11.883
	MedAE [deg]	5.994	5.961	8.216	6.283	6.140
	Variance [deg^2]	595.683	579.601	815.775	301.535	686.684

5.2.2 Experiment B: Hall

We conduct the second experiment in the lower hall of the IT building, where the ground plane is less restricted in width compared to the hallway used in experimentA. With a width of 6.28 m, this area provides more points representing the ground plane in each scan. The floor remains flat, providing a consistent surface to evaluate the robustness and accuracy of each approach over time, just as in the first experiment. During this experiment, we roll the spherical mobile mapping system in a straight line for approximately 14.5 m, moving slowly and steadily to maintain consistency. Refer to Figure 5.6 for a visual representation of the conduction of this experiment.

Ground Truth As with the first experiment, we assume a normal vector of $\boldsymbol{n}_{gt} = [0, 0, -1]^T$ as the ground truth, with the corresponding inclination of $0\,°$.

Results We present the runtime and plane segmentation error rate (fail rate) results for each approach in Table 5.5, and we summarize the accuracy results in Table 5.6. From the results, we observe that the approaches without filtering again perform the worst in terms of runtime for

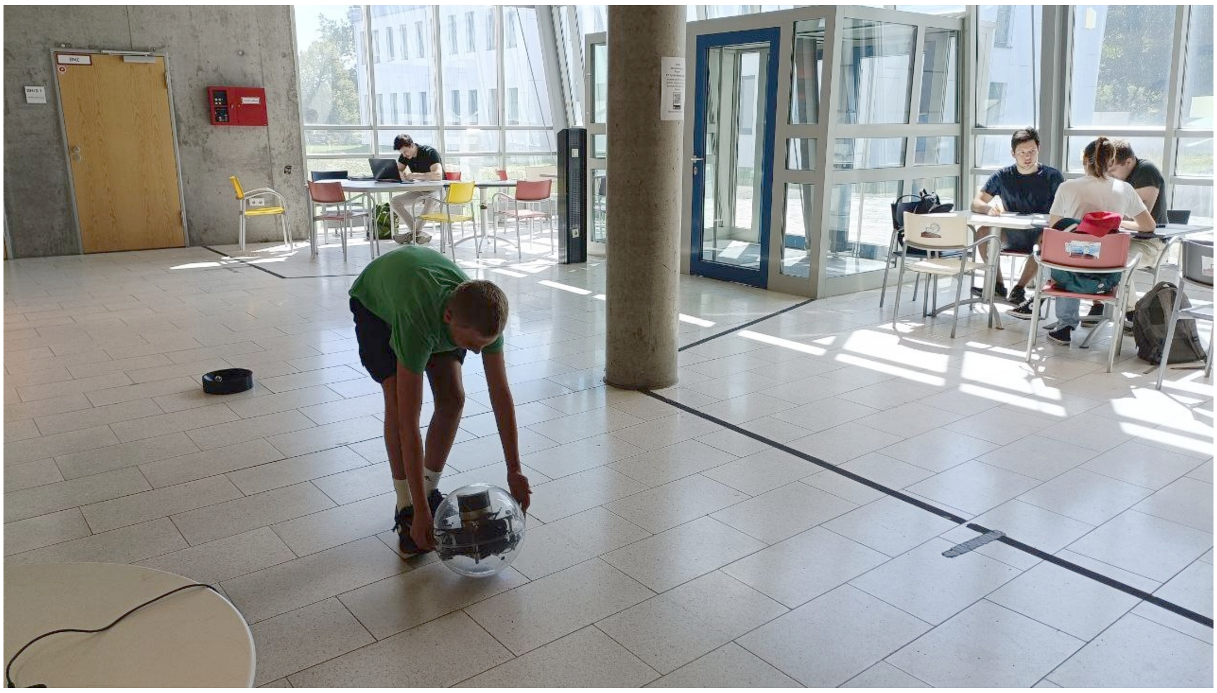


Figure 5.6: Execution of experiment B: hall. Photo taken during the experiment, showing the person moving the spherical mobile mapping system in the lower hall of the IT building.

all plane segmentation algorithms, with none-geo at 21.895 ms and none-kdt at 15.571 ms. The geo-geo approach remains the fastest, achieving an average runtime of 9.158 ms. All approaches still maintain a runtime below 50 ms, ensuring real-time operation. Overall, this experiment exhibits higher errors and variances compared to experiment A. In terms of accuracy, RANSAC and RHT demonstrate the most precise results for the sub-cloud approaches with kd-tree (geo-kdt and kdt-kdt), as well as the geometrical sub-cloud approaches (geo-geo and kdt-geo), with both the MAE and MedAE averaging around 10.9 °. However, RANSAC shows a slightly lower variance across all four cases, indicating more consistent performance. Regarding robustness, the none-geo approach again exhibits the lowest fail rate, averaging 1.816 %, followed by the geometrical sub-cloud approaches, which have a fail rate of approximately 2.0 %. Notably, we observe lower fail rates in this experiment than in experiment A, indicating improved robustness in this experiment. Considering the balance of runtime, accuracy, and robustness, the best overall solution for this experiment is, once again, RANSAC with geo-geo pre-processing. Figure 5.7 displays the absolute angle error over the number of calls to the callback function for RANSAC with geo-geo pre-processing.

Table 5.5: Runtime and fail rate results of experiment "hall". Runtimes [ms] of each step of the different pre-processing combinations for each plane segmentation algorithm, as well as the total runtime. The fail rate indicates the percentage of plane segmentation errors.

	Step	LSF	PCA	RAN	RHT	RHT2	Avg.
none-geo	**Cheese Search [ms]**	4.085	1.827	2.155	5.591	2.170	**3.165**
	Delete [ms]	0.777	0.348	0.408	1.065	0.410	**0.601**
	Plane [ms]	10.499	28.080	23.342	3.234	25.487	**18.128**
	Total [ms]	15.361	30.254	25.904	9.889	28.067	**21.895**
	Fail Rate [%]	2.304	2.304	0.136	0.136	4.201	**1.816**
none-kdt	**Build Tree [ms]**	14.531	14.022	13.448	11.402	12.684	**13.217**
	k-NNS [ms]	0.128	0.128	0.111	0.111	0.118	**0.119**
	Delete [ms]	0.709	0.730	0.698	0.622	0.671	**0.686**
	Plane [ms]	0.235	1.123	1.171	3.167	2.044	**1.548**
	Total [ms]	15.604	16.003	15.427	15.302	15.517	**15.571**
	Fail Rate [%]	41.734	40.921	29.675	46.070	48.103	**41.301**
kdt-geo	**Downsample [ms]**	5.871	5.414	5.455	5.148	5.337	**5.445**
	Build Tree [ms]	2.611	2.458	2.464	2.337	2.401	**2.454**
	Radius Search [ms]	0.030	0.026	0.025	0.024	0.025	**0.026**
	Delete [ms]	0.322	0.303	0.303	0.287	0.295	**0.302**
	Cheese Search [ms]	0.685	0.650	0.648	0.619	0.638	**0.648**
	Delete [ms]	0.176	0.170	0.168	0.161	0.165	**0.168**
	Plane [ms]	0.496	2.433	2.423	1.700	2.104	**1.831**
	Total [ms]	10.191	11.453	11.486	10.276	10.965	**10.874**
	Fail Rate [%]	2.168	2.168	0.000	0.000	5.962	**2.060**
geo-geo	**Downsample [ms]**	6.321	6.417	5.954	5.535	5.916	**6.029**
	Cheese Search [ms]	0.867	0.877	0.809	0.765	0.813	**0.826**
	Delete [ms]	0.207	0.211	0.196	0.184	0.196	**0.199**
	Plane [ms]	0.554	2.917	2.758	1.878	2.413	**2.104**
	Total [ms]	7.948	10.422	9.718	8.363	9.338	**9.158**
	Fail Rate [%]	2.168	2.168	0.000	0.136	5.691	**2.033**

Table 5.5: Runtime and fail rate results of experiment "hall". Runtimes [ms] for each step of the different pre-processing combinations for each plane segmentation algorithm, as well as the total runtime. The fail rate indicates the percentage of plane segmentation errors. (continued)

	Step	LSF	PCA	RAN	RHT	RHT2	Avg.
geo-kdt	**Downsample [ms]**	5.920	5.747	5.927	5.062	5.544	**5.640**
	Radius Search [ms]	0.299	0.308	0.255	0.224	0.251	**0.267**
	Delete [ms]	0.375	0.369	0.320	0.285	0.316	**0.333**
	Build Tree [ms]	3.085	3.014	2.574	2.264	2.570	**2.702**
	k-NNS [ms]	0.057	0.056	0.054	0.048	0.052	**0.053**
	Delete [ms]	0.212	0.205	0.179	0.158	0.178	**0.186**
	Plane [ms]	0.334	1.550	1.393	1.832	1.561	**1.334**
	Total [ms]	10.280	11.250	10.702	9.873	10.472	**10.515**
	Fail Rate [%]	31.165	31.436	0.000	0.000	10.434	**14.607**
kdt-kdt	**Downsample [ms]**	5.500	5.275	5.348	4.729	5.178	**5.206**
	Build Tree [ms]	2.895	2.792	2.408	2.141	2.393	**2.526**
	Radius Search [ms]	0.026	0.025	0.025	0.022	0.025	**0.025**
	Delete [ms]	0.352	0.333	0.297	0.263	0.293	**0.307**
	Build Tree [ms]	2.777	2.663	2.312	2.080	2.309	**2.428**
	k-NNS [ms]	0.048	0.045	0.045	0.041	0.044	**0.045**
	Delete [ms]	0.187	0.180	0.161	0.145	0.161	**0.167**
	Plane [ms]	0.299	1.363	1.266	1.676	1.419	**1.205**
	Total [ms]	12.083	12.676	11.861	11.097	11.822	**11.908**
	Fail Rate [%]	31.436	31.707	0.000	0.000	9.621	**14.553**

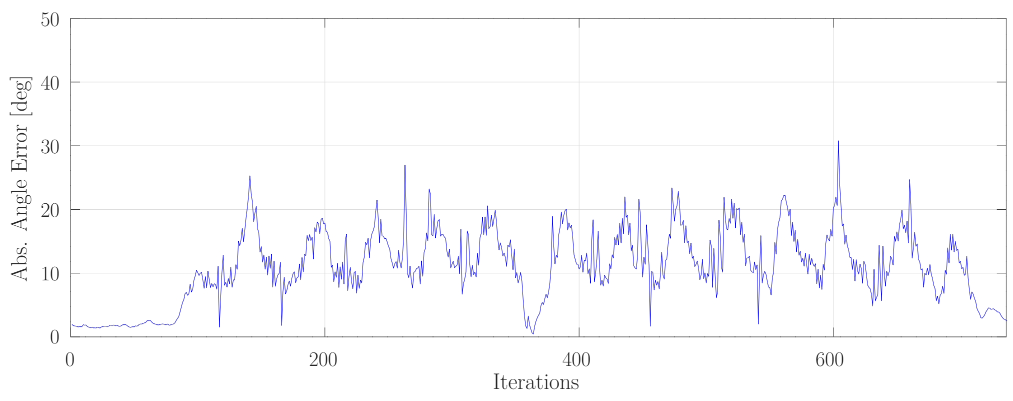


Figure 5.7: Experiment B: Plot of the absolute angle error over iterations, i.e. the number of calls to the callback function, for RANSAC using geo-geo pre-processing. Red lines indicate a change of ground plane and thereby inclination.

Table 5.6: MAE [deg], MedAE [deg] and result variance [deg^2] of experiment "hall" for the different plane segmentation algorithms under different pre-processing combinations.

	Result	LSF	PCA	RAN	RHT	RHT2
none-geo	**MAE [deg]**	15.897	17.465	16.087	13.465	19.401
	MedAE [deg]	14.672	17.143	16.276	13.111	19.423
	Variance [deg^2]	1119.225	1333.281	1127.814	794.615	1629.677
none-kdt	**MAE [deg]**	20.015	20.225	22.919	20.215	18.696
	MedAE [deg]	15.106	15.166	22.245	18.222	14.034
	Variance [deg^2]	1761.558	1797.956	2248.340	1793.820	1540.422
kdt-geo	**MAE [deg]**	14.700	14.769	11.047	11.314	17.390
	MedAE [deg]	14.174	14.334	11.061	11.210	15.271
	Variance [deg^2]	946.610	954.611	520.434	546.426	1340.566
geo-geo	**MAE [deg]**	14.591	14.667	11.020	11.252	17.379
	MedAE [deg]	13.979	14.050	10.966	11.271	15.119
	Variance [deg^2]	933.601	943.827	517.921	541.000	1342.117
geo-kdt	**MAE [deg]**	13.882	13.866	10.872	10.960	13.858
	MedAE [deg]	10.332	10.474	10.867	10.836	11.931
	Variance [deg^2]	883.600	879.815	504.202	512.996	866.319
kdt-kdt	**MAE [deg]**	13.783	13.717	10.879	10.963	13.855
	MedAE [deg]	10.469	10.484	10.987	11.071	12.020
	Variance [deg^2]	870.299	859.650	505.058	513.370	864.140

5.2.3 Experiment C: Wall

We conduct the third experiment in the same hallway used in experimentA. In this scenario, the spherical mobile mapping system starts from the center of the hallway and moves towards one wall, traveling approximately 3.8 m forward before switching to the opposite side of the hallway. Then we roll the robot along the second wall until it returns to the height of the starting point. This experiment introduces an extreme limitation of the ground plane on one side of the sphere, while maintaining a flat surface throughout. We ensure the robot moves at the same slow and steady pace as in the previous experiments for consistency.

Ground Truth As in the previous experiments, we assume a normal vector of $\boldsymbol{n}_{gt} = [0, 0, -1]^T$ as the ground truth, corresponding to an inclination of $0\,°$.

Results We present the runtime and plane segmentation error rate (fail rate) results for each approach in Table 5.7, and we summarize the accuracy results in Table 5.8. From the results, we observe that the runtime for the approaches remains consistent with previous experiments. The none-geo approach performs the slowest with 21.248 ms, followed by none-kdt with 15.258 ms.

The geo-geo approach continues to be the fastest, with an average runtime of 7.557 ms. All approaches maintain a runtime below 50 ms, ensuring real-time operation. In terms of accuracy, RANSAC demonstrates the lowest MAE for the kd-tree sub-cloud approaches (geo-kdt and kdt-kdt) with approximately 8.8 ° and MedAE of 8.3 °. It is closely followed by RANSAC with geometrical sub-cloud approaches and RHT with kd-tree sub-cloud approaches, where MAE is below 9 ° and MedAE is roughly 9.6 °. Regarding robustness, the fail rate trends in experimentC follow the same pattern as in previous experiments. However, the fail rates are higher compared to experiment B. The none-geo approach shows the lowest fail rate. With an average of 4.2 % for geometrical sub-cloud approaches, we observe a similar robustness as in experiment A. For the kd-tree sub-cloud approaches However, the fail rate increases significantly to 26.5 % in experiment C, compared to 17.5 % observed in experiment A. Considering the balance of runtime, accuracy, and robustness, the best overall solution for this experiment remains RANSAC with geo-geo pre-processing. Figure 5.8 displays the absolute angle error over the number of calls to the callback function for RANSAC with geo-geo pre-processing.

Table 5.7: Runtime and fail rate results of experiment "wall". Runtimes [ms] of each step of the different pre-processing combinations for each plane segmentation algorithm, as well as the total runtime. The fail rate indicates the percentage of plane segmentation errors.

	Step	LSF	PCA	RAN	RHT	RHT2	Avg.
none-geo	**Cheese Search [ms]**	4.379	2.071	2.344	5.235	2.321	**3.270**
	Delete [ms]	0.791	0.374	0.424	0.937	0.416	**0.588**
	Plane [ms]	9.849	26.778	22.799	3.925	23.596	**17.389**
	Total [ms]	15.019	29.223	25.567	10.097	26.332	**21.248**
	Fail Rate [%]	3.851	4.091	3.851	1.685	3.129	**3.321**
none-kdt	**Build Tree [ms]**	14.198	13.484	13.363	11.393	12.342	**12.956**
	k-NNS [ms]	0.125	0.129	0.113	0.124	0.128	**0.124**
	Delete [ms]	0.700	0.706	0.705	0.636	0.666	**0.683**
	Plane [ms]	0.228	1.081	1.128	2.950	2.092	**1.496**
	Total [ms]	15.252	15.399	15.309	15.105	15.227	**15.258**
	Fail Rate [%]	53.911	47.172	35.259	49.819	57.040	**48.640**
kdt-geo	**Downsample [ms]**	5.579	5.510	5.489	4.961	5.300	**5.368**
	Build Tree [ms]	1.368	1.333	1.329	1.215	1.306	**1.310**
	Radius Search [ms]	0.027	0.027	0.027	0.025	0.026	**0.026**
	Delete [ms]	0.192	0.187	0.186	0.171	0.183	**0.184**
	Cheese Search [ms]	0.393	0.385	0.385	0.353	0.376	**0.378**
	Delete [ms]	0.104	0.102	0.102	0.093	0.099	**0.100**
	Plane [ms]	0.268	1.112	1.211	2.365	1.546	**1.300**
	Total [ms]	7.931	8.657	8.729	9.182	8.836	**8.667**
	Fail Rate [%]	3.971	3.971	2.286	3.249	8.183	**4.332**

Table 5.7: Runtime and fail rate results of experiment”wall”. Runtimes [ms] for each step of the different pre-processing combinations for each plane segmentation algorithm, as well as the total runtime. The fail rate indicates the percentage of plane segmentation errors. (continued)

	Step	LSF	PCA	RAN	RHT	RHT2	Avg.
geo-geo	**Downsample [ms]**	6.016	5.792	5.798	5.068	5.521	**5.639**
	Cheese Search [ms]	0.482	0.459	0.457	0.412	0.444	**0.451**
	Delete [ms]	0.122	0.115	0.116	0.104	0.111	**0.114**
	Plane [ms]	0.291	1.189	1.299	2.413	1.575	**1.354**
	Total [ms]	6.911	7.554	7.671	7.997	7.651	**7.557**
	Fail Rate [%]	3.971	3.971	2.286	3.249	7.100	**4.116**
geo-kdt	**Downsample [ms]**	5.657	5.487	5.388	4.233	5.158	**5.184**
	Radius Search [ms]	0.174	0.167	0.144	0.116	0.150	**0.150**
	Delete [ms]	0.226	0.222	0.190	0.153	0.198	**0.198**
	Build Tree [ms]	1.658	1.614	1.338	1.099	1.444	**1.431**
	k-NNS [ms]	0.054	0.052	0.051	0.040	0.048	**0.049**
	Delete [ms]	0.128	0.124	0.106	0.086	0.111	**0.111**
	Plane [ms]	0.353	1.639	2.654	5.206	2.400	**2.450**
	Total [ms]	8.248	9.305	9.870	10.933	9.510	**9.573**
	Fail Rate [%]	39.711	39.471	9.146	11.432	30.927	**26.137**
kdt-kdt	**Downsample [ms]**	5.452	5.247	5.115	4.056	4.975	**4.969**
	Build Tree [ms]	1.615	1.548	1.290	1.068	1.419	**1.388**
	Radius Search [ms]	0.027	0.026	0.025	0.020	0.024	**0.025**
	Delete [ms]	0.218	0.211	0.179	0.149	0.193	**0.190**
	Build Tree [ms]	1.552	1.488	1.231	1.026	1.361	**1.332**
	k-NNS [ms]	0.046	0.045	0.044	0.036	0.042	**0.043**
	Delete [ms]	0.120	0.116	0.100	0.083	0.106	**0.105**
	Plane [ms]	0.337	1.552	2.526	4.990	2.487	**2.379**
	Total [ms]	9.366	10.234	10.509	11.428	10.608	**10.429**
	Fail Rate [%]	39.711	39.471	9.146	11.552	35.018	**26.980**

5.2.4 Experiment D: Physics Building Ramp - Slow Motion

In the fourth experiment, we carry out our tests in the hall of the physics building, which features a flat ramp. This ramp is remarkably long in one direction, extends over several meters (see Figure 5.9), and is limited to a width of about 2.00 m. The floor in this scenario consists of two different flat surfaces: the base floor and the ramp with a sharp transition between them. We roll the spherical mobile mapping system a short distance in front of the ramp, then about

Table 5.8: MAE [deg], MedAE [deg] and result variance [deg^2] of experiment ”wall” for the different plane segmentation algorithms under different pre-processing combinations.

	Result	LSF	PCA	RAN	RHT	RHT2
none-geo	**MAE [deg]**	16.109	16.714	14.097	14.454	17.258
	MedAE [deg]	14.516	15.940	13.160	12.765	16.440
	Variance [deg^2]	1166.633	1248.662	896.949	945.731	1311.565
none-kdt	**MAE [deg]**	23.984	22.907	23.593	19.268	19.350
	MedAE [deg]	23.605	21.597	21.868	16.356	16.400
	Variance [deg^2]	2457.116	2262.005	2379.095	1652.010	1667.894
kdt-geo	**MAE [deg]**	13.395	13.514	9.554	10.249	15.596
	MedAE [deg]	11.108	11.171	8.935	9.195	12.126
	Variance [deg^2]	820.830	836.082	399.909	471.169	1107.992
geo-geo	**MAE [deg]**	13.310	13.476	9.507	10.010	15.405
	MedAE [deg]	11.063	11.121	8.933	9.139	12.040
	Variance [deg^2]	811.650	832.319	395.842	446.090	1086.810
geo-kdt	**MAE [deg]**	17.074	17.276	8.815	9.777	14.908
	MedAE [deg]	14.424	14.723	8.280	8.496	12.522
	Variance [deg^2]	1298.477	1330.224	336.189	432.958	1016.160
kdt-kdt	**MAE [deg]**	17.066	17.236	8.809	9.518	14.596
	MedAE [deg]	14.723	14.720	8.290	8.351	11.141
	Variance [deg^2]	1297.396	1324.242	335.880	408.530	987.427

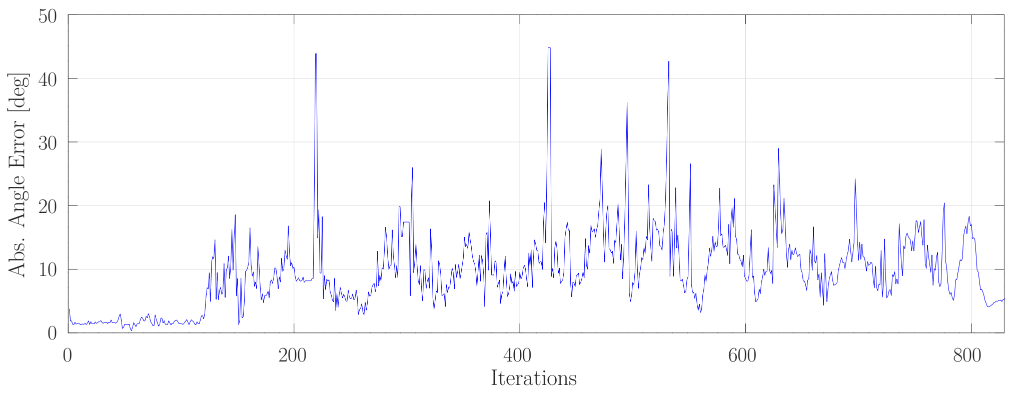
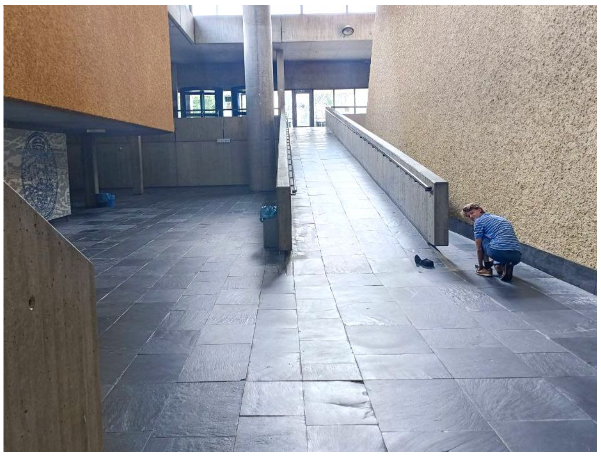
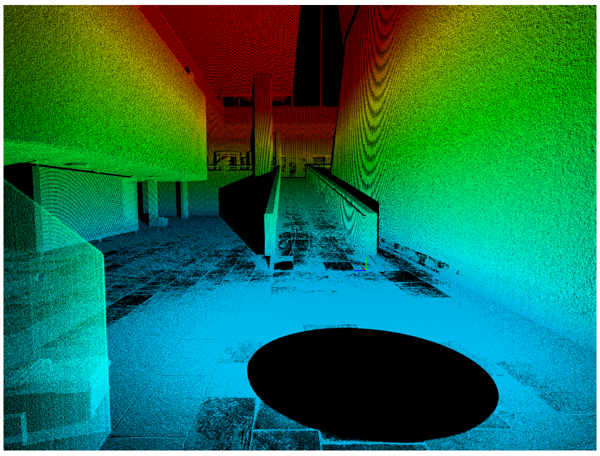


Figure 5.8: Experiment C: Plot of the absolute angle error over iterations, i.e. the number of calls to the callback function, for RANSAC using geo-geo pre-processing. Red lines indicate a change of ground plane and thereby inclination.

7.0 m up, as well as back down the ramp, and finally return to starting position on the base floor. We try to maintain the same slow and steady pace as in all previous experiments to ensure consistency.



(a) Image of the physics building ramp during the laser distance measurement for the ground truth determination.



(b) 3DTK screenshot of the scan taken with the Riegl scanner of the same scenario, illustrating the captured point cloud.

Figure 5.9: Visual comparison of the actual physics building ramp and its corresponding 3D scan data used for ground truth determination.

Ground Truth To determine the ground truth, we use the Riegl VZ-400 laser scanner, as explained in Section 5.1.3, to extract accurate normal vectors of the ground planes and their respective inclination angles from a full scan. We determine that the ramp has an inclination of 6.86 °, while the ground has a minimum inclination of 1.90 °. In addition, we measure the inclination for this ramp with a laser distance meter by determining the height and length of the ramp. Refer to Figure 5.9a for a visual representation of the ground truth's measurement process. This method results in an inclination of 7.08 ° for the ramp, while we assume that the ground is level (0.0 °) according to the previous assumptions. Therefore, we set the combined ground truth to 1.0 ° for the base floor and 7.0 ° for the ramp.

Results The results of this experiment, including runtime and plane segmentation error rates for each approach, are presented in Table 5.9. The accuracy of the results is summarized in Table 5.10. From the results, we observe that the runtime remains consistent across all approaches, with similar values to those seen in previous experiments. In terms of robustness, geometrical sub-cloud approaches exhibit the best performance, with fail rates consistently below 1.8 %. The none-geo approach ranks second, with a fail rate of 2.5 %. The accuracy is notably improved in experiment D, with the lowest average errors and variances observed across all experiments to date. The most accurate results are achieved by RHT for kd-tree sub-cloud approaches, which yields a MAE of 4.8 ° and a MedAE of 5.7 °. RANSAC shows slightly higher values with a MAE of 5.0 ° and a MedAE of 5.6 ° for the kd-tree sub-cloud approaches, and MAE of 5.3 ° and MedAE of 5.9 ° for the geometrical sub-cloud approaches. For the geometrical sub-cloud approaches, we observe a low MedAE of 5.8 ° for the plane fitting algorithms, but three times higher variances compared to RHT and RANSAC, along with higher MAE values. Considering the combination of runtime, accuracy, and robustness, the best overall solution for experiment D is RANSAC with geo-geo pre-processing. Figure 5.10 displays the absolute angle error over the number of calls to the callback function for RANSAC with geo-geo pre-processing.

Table 5.9: Runtime and fail rate results of experiment "physics building ramp with slow motion". Runtimes [ms] for each step of the different pre-processing combinations for each plane segmentation algorithm, as well as the total runtime. The fail rate indicates the percentage of plane segmentation errors.

	Step	LSF	PCA	RAN	RHT	RHT2	Avg.
none-geo	**Cheese Search [ms]**	4.253	1.985	2.094	5.058	2.347	**3.147**
	Delete [ms]	0.761	0.358	0.379	0.926	0.438	**0.573**
	Plane [ms]	10.323	27.607	25.087	5.246	24.156	**18.484**
	Total [ms]	15.337	29.950	27.561	11.230	26.942	**22.204**
	Fail Rate [%]	2.017	2.073	2.465	2.129	3.978	**2.532**
none-kdt	**Build Tree [ms]**	13.373	13.137	12.850	10.847	11.153	**12.272**
	k-NNS [ms]	0.116	0.109	0.101	0.109	0.111	**0.109**
	Delete [ms]	0.673	0.661	0.646	0.566	0.562	**0.621**
	Plane [ms]	0.231	1.049	1.076	3.532	2.025	**1.583**
	Total [ms]	14.393	14.956	14.673	15.055	13.850	**14.585**
	Fail Rate [%]	53.333	50.196	37.479	53.165	60.056	**50.846**
kdt-geo	**Downsample [ms]**	5.432	5.182	5.203	4.539	4.535	**4.978**
	Build Tree [ms]	2.435	2.323	2.318	2.063	2.043	**2.236**
	Radius Search [ms]	0.028	0.026	0.026	0.023	0.023	**0.025**
	Delete [ms]	0.313	0.298	0.299	0.265	0.285	**0.292**
	Cheese Search [ms]	0.660	0.627	0.631	0.556	0.548	**0.604**
	Delete [ms]	0.173	0.166	0.166	0.147	0.152	**0.161**
	Plane [ms]	0.357	1.601	1.671	2.139	1.537	**1.461**
	Total [ms]	9.398	10.224	10.314	9.732	9.122	**9.758**
	Fail Rate [%]	1.849	1.849	0.056	1.064	3.754	**1.714**
geo-geo	**Downsample [ms]**	5.975	5.610	5.570	4.587	5.204	**5.389**
	Cheese Search [ms]	0.818	0.759	0.755	0.657	0.736	**0.745**
	Delete [ms]	0.200	0.189	0.188	0.160	0.186	**0.185**
	Plane [ms]	0.401	1.765	1.836	2.217	1.878	**1.619**
	Total [ms]	7.394	8.323	8.349	7.622	8.005	**7.939**
	Fail Rate [%]	1.849	1.849	0.056	0.784	3.529	**1.613**

Table 5.9: Runtime and fail rate results of experiment "physics building ramp with slow motion". Runtimes [ms] for each step of the different pre-processing combinations for each plane segmentation algorithm, as well as the total runtime. The fail rate indicates the percentage of plane segmentation errors. (continued)

	Step	LSF	PCA	RAN	RHT	RHT2	Avg.
geo-kdt	**Downsample [ms]**	5.425	5.221	5.375	4.314	4.658	**4.999**
	Radius Search [ms]	0.262	0.254	0.248	0.206	0.228	**0.240**
	Delete [ms]	0.332	0.318	0.312	0.260	0.298	**0.304**
	Build Tree [ms]	2.588	2.471	2.399	2.019	2.081	**2.312**
	k-NNS [ms]	0.060	0.057	0.055	0.046	0.057	**0.055**
	Delete [ms]	0.184	0.178	0.176	0.147	0.166	**0.170**
	Plane [ms]	0.343	1.552	2.283	3.849	2.256	**2.057**
	Total [ms]	9.195	10.050	10.847	10.840	9.743	**10.135**
	Fail Rate [%]	36.022	36.415	0.728	8.011	12.325	**18.700**
kdt-kdt	**Downsample [ms]**	5.117	4.889	4.845	4.014	4.270	**4.627**
	Build Tree [ms]	2.463	2.344	2.188	1.891	1.948	**2.167**
	Radius Search [ms]	0.034	0.025	0.025	0.021	0.022	**0.025**
	Delete [ms]	0.311	0.296	0.282	0.241	0.274	**0.281**
	Build Tree [ms]	2.353	2.257	2.108	1.830	1.874	**2.085**
	k-NNS [ms]	0.051	0.050	0.046	0.039	0.044	**0.046**
	Delete [ms]	0.168	0.163	0.155	0.135	0.149	**0.154**
	Plane [ms]	0.314	1.418	2.042	3.538	1.999	**1.862**
	Total [ms]	10.813	11.441	11.692	11.708	10.581	**11.247**
	Fail Rate [%]	36.134	36.359	0.728	7.955	12.213	**18.678**

5.2.5 Experiment E: Physics Building Ramp - Fast Motion

We conduct a similar test in the hallway of the physics building for the fifth experiment, using the same flat ramp as in experiment D (refer to Section 5.2.4). This time, the spherical mobile mapping system rolls up the ramp for approximately 9.5 m but at a faster speed compared to previous experiments. After reaching the top, the robot descends back down and returns to the base floor, following a similar path as before. The increased speed introduces an additional challenge for the LiDAR sensor and the algorithms, as the faster movement may affect the accuracy of the ground plane detection.

Ground Truth The ground truth for this experiment remains the same as described in experiment D, refer to Section 5.2.4. The ramp is set at 7.0 ° inclination, and the base floor at 1.0 °. These values are determined using the RIEGL scanner and laser distance measurements, as detailed previously.

Table 5.10: MAE [deg], MedAE [deg] and result variance [deg^2] of experiment "physics building ramp with slow motion" for the different plane segmentation algorithms under different pre-processing combinations.

	Result	LSF	PCA	RAN	RHT	RHT2
none-geo	**MAE [deg]**	12.067	12.800	12.067	11.688	15.160
	MedAE [deg]	9.233	10.403	8.797	8.158	12.583
	Variance [deg^2]	659.025	734.910	674.969	633.290	1021.330
none-kdt	**MAE [deg]**	18.809	18.440	20.017	19.974	19.602
	MedAE [deg]	18.991	18.517	19.553	20.365	19.855
	Variance [deg^2]	1545.297	1488.115	1733.879	1715.938	1653.665
kdt-geo	**MAE [deg]**	9.055	9.061	5.900	7.286	13.302
	MedAE [deg]	5.820	5.763	5.308	5.447	9.870
	Variance [deg^2]	374.166	374.313	125.354	227.975	801.576
geo-geo	**MAE [deg]**	8.961	9.006	5.843	7.098	12.939
	MedAE [deg]	5.761	5.739	5.250	5.432	9.797
	Variance [deg^2]	365.226	369.418	122.446	212.501	754.001
geo-kdt	**MAE [deg]**	10.445	10.435	5.650	5.741	10.448
	MedAE [deg]	7.529	7.571	5.040	4.803	6.704
	Variance [deg^2]	505.990	506.071	111.201	121.166	502.037
kdt-kdt	**MAE [deg]**	10.396	10.463	5.675	5.759	10.813
	MedAE [deg]	7.634	7.657	5.049	4.815	6.494
	Variance [deg^2]	501.406	508.418	112.995	122.275	539.467

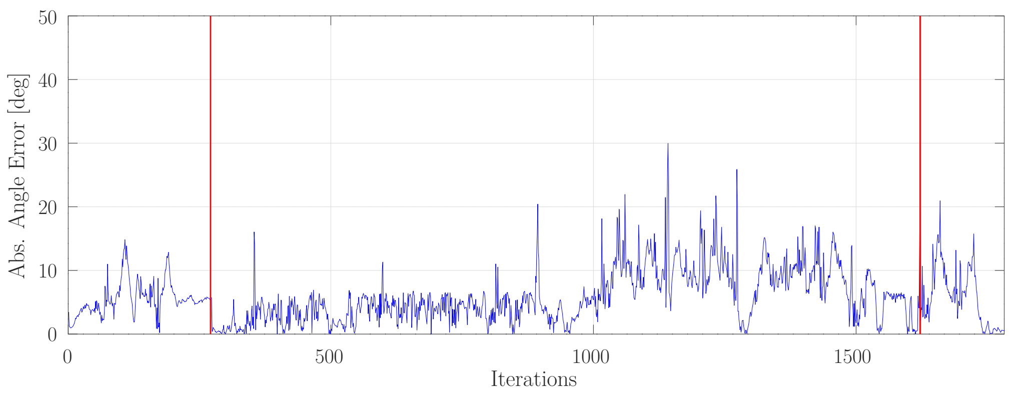


Figure 5.10: Experiment D: Plot of the absolute angle error over iterations, i.e. the number of calls to the callback function, for RANSAC using geo-geo pre-processing. Red lines indicate a change of ground plane and thereby inclination.

Results We present the results of this experiment, regarding the runtime and plane segmentation error rates for each method, in Table 5.11. A summary of the accuracy results is provided in Table 5.12. From the results, we observe that the runtime remains consistent across all approaches, with values similar to those seen in previous experiments. In terms of robustness, the none-geo approach exhibits the best performance, with a fail rate of 9.2 %. This is followed by the geometrical sub-cloud approaches, which show a fail rate of roughly 10.0 %. But these fail rates are significantly higher compared to previous experiments, where they were for the same pre-processing approaches consistently below 5.0 %. In terms of accuracy, we observe in experiment E an overall lower performance compared to experiment D. The most accurate results are achieved by RANSAC for all approaches with filtering, with a MedAE of approximately 6.3 ° and a MAE of 9.9 °. This is closely followed by RHT for the kd-tree sub-cloud approaches, which yields a MedAE of 6.4 ° and a MAE of 10.0 °. The worst performance is observed with the none-kdt approach for all plane segmentation algorithms, reflecting trends seen in previous experiments. Considering the balance of runtime, accuracy, and robustness, the best overall solution for experiment E is RANSAC with the geo-geo pre-processing. Figure 5.11 displays the absolute angle error over the number of calls to the callback function for RANSAC with geo-geo pre-processing.

Table 5.11: Runtime and fail rate results of experiment "physics building ramp with fast motion". Runtimes [ms] of each step of the different pre-processing combinations for each plane segmentation algorithm, as well as the total runtime. The fail rate indicates the percentage of plane segmentation errors.

	Step	LSF	PCA	RAN	RHT	RHT2	Avg.
none-geo	**Cheese Search [ms]**	3.651	1.736	2.062	4.804	2.062	**2.863**
	Delete [ms]	0.664	0.322	0.383	0.885	0.409	**0.532**
	Plane [ms]	9.769	28.329	24.596	6.315	25.081	**18.818**
	Total [ms]	14.084	30.387	27.041	12.004	27.552	**22.214**
	Fail Rate [%]	10.386	12.017	4.807	4.635	14.163	**9.202**
none-kdt	**Build Tree [ms]**	14.433	13.705	12.824	10.509	10.925	**12.479**
	k-NNS [ms]	0.127	0.112	0.101	0.095	0.089	**0.105**
	Delete [ms]	0.718	0.700	0.644	0.539	0.561	**0.632**
	Plane [ms]	0.250	1.147	1.083	4.164	2.732	**1.875**
	Total [ms]	15.528	15.664	14.651	15.307	14.306	**15.091**
	Fail Rate [%]	52.017	48.584	40.172	52.790	78.455	**54.403**

Table 5.11: Runtime and fail rate results of experiment ”physics building ramp with fast motion”. Runtimes [ms] for each step of the different pre-processing combinations for each plane segmentation algorithm, as well as the total runtime. The fail rate indicates the percentage of plane segmentation errors. (continued)

	Step	LSF	PCA	RAN	RHT	RHT2	Avg.
kdt-geo	**Downsample [ms]**	5.380	5.121	5.194	4.511	4.412	**4.924**
	Build Tree [ms]	2.386	2.270	2.283	2.017	1.966	**2.185**
	Radius Search [ms]	0.028	0.026	0.027	0.024	0.034	**0.028**
	Delete [ms]	0.300	0.285	0.288	0.254	0.273	**0.280**
	Cheese Search [ms]	0.640	0.608	0.616	0.543	0.530	**0.587**
	Delete [ms]	0.164	0.156	0.158	0.140	0.148	**0.153**
	Plane [ms]	0.364	1.639	1.523	2.156	1.622	**1.461**
	Total [ms]	9.263	10.106	10.089	9.645	8.985	**9.618**
	Fail Rate [%]	12.275	12.017	3.090	5.408	16.738	**9.906**
geo-geo	**Downsample [ms]**	5.885	5.643	5.652	4.938	4.865	**5.397**
	Cheese Search [ms]	0.791	0.760	0.758	0.666	0.660	**0.727**
	Delete [ms]	0.192	0.186	0.185	0.162	0.168	**0.179**
	Plane [ms]	0.403	1.900	1.689	2.445	1.886	**1.665**
	Total [ms]	7.271	8.489	8.283	8.211	7.580	**7.967**
	Fail Rate [%]	12.189	12.103	3.519	6.009	16.567	**10.077**
geo-kdt	**Downsample [ms]**	5.400	5.220	5.398	4.364	4.537	**4.984**
	Radius Search [ms]	0.237	0.230	0.239	0.202	0.221	**0.226**
	Delete [ms]	0.302	0.293	0.301	0.254	0.291	**0.288**
	Build Tree [ms]	2.361	2.277	2.348	2.004	2.065	**2.211**
	k-NNS [ms]	0.055	0.052	0.051	0.043	0.045	**0.049**
	Delete [ms]	0.167	0.163	0.167	0.143	0.160	**0.160**
	Plane [ms]	0.352	1.620	2.192	4.325	2.155	**2.129**
	Total [ms]	8.873	9.855	10.697	11.334	9.474	**10.047**
	Fail Rate [%]	48.412	48.326	2.918	10.215	28.927	**27.760**

Table 5.11: Runtime and fail rate results of experiment "physics building ramp with fast motion". Runtimes [ms] for each step of the different pre-processing combinations for each plane segmentation algorithm, as well as the total runtime. The fail rate indicates the percentage of plane segmentation errors. (continued)

	Step	LSF	PCA	RAN	RHT	RHT2	Avg.
kdt-kdt	**Downsample [ms]**	5.094	4.813	4.820	3.993	4.207	**4.585**
	Build Tree [ms]	2.225	2.098	2.160	1.837	1.963	**2.057**
	Radius Search [ms]	0.043	0.026	0.026	0.021	0.022	**0.028**
	Delete [ms]	0.282	0.269	0.273	0.232	0.268	**0.265**
	Build Tree [ms]	2.115	2.024	2.059	1.775	1.863	**1.967**
	k-NNS [ms]	0.046	0.044	0.042	0.036	0.039	**0.041**
	Delete [ms]	0.152	0.146	0.147	0.128	0.145	**0.144**
	Plane [ms]	0.322	1.467	1.988	3.950	2.004	**1.946**
	Total [ms]	10.278	10.887	11.514	11.972	10.512	**11.032**
	Fail Rate [%]	48.326	48.240	3.004	9.957	30.386	**27.983**

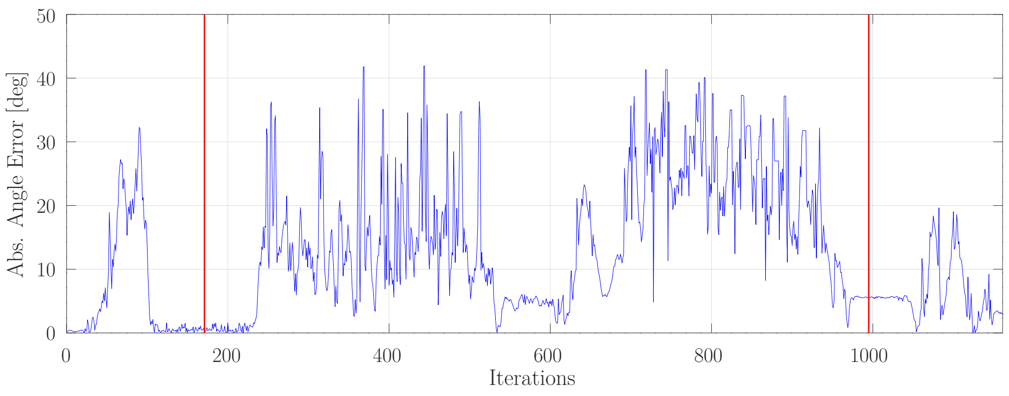


Figure 5.11: Experiment E: Plot of the absolute angle error over iterations, i.e. the number of calls to the callback function, for RANSAC using geo-geo pre-processing. Red lines indicate a change of ground plane and thereby inclination.

5.2.6 Experiment F: Outside Hill Moving Up

In the sixth experiment, we test the spherical mobile mapping system on the mowed hill outside the robotics hall, refer to Figure 5.13. We perform this experiment on the mowed hill to minimize noise in the LiDAR data caused by long grass and flowers, resulting in clearer and more accurate plane measurements. Moreover, unlike the ramp in the physics building or the hallway inside the IT building, the hill is not limited by walls to the outside and instead declines steeply afterwards. It is possible that this open-ended decline reduces the noise in the data, leading

Table 5.12: MAE [deg], MedAE [deg] and result variance [deg^2] of experiment "physics building ramp with fast motion" for the different plane segmentation algorithms under different pre-processing combinations.

	Result	LSF	PCA	RAN	RHT	RHT2
none-geo	**MAE [deg]**	15.607	16.525	16.037	15.117	17.132
	MedAE [deg]	13.784	15.743	14.785	13.944	15.346
	Variance [deg^2]	1086.722	1213.928	1128.504	1014.750	1297.775
none-kdt	**MAE [deg]**	21.242	19.549	20.812	21.861	21.156
	MedAE [deg]	21.639	18.359	21.504	22.044	21.727
	Variance [deg^2]	1916.557	1642.597	1856.516	2046.889	1908.405
kdt-geo	**MAE [deg]**	12.190	12.311	10.064	10.557	15.778
	MedAE [deg]	8.848	8.747	6.587	7.045	13.991
	Variance [deg^2]	685.457	697.351	466.055	512.334	1102.959
geo-geo	**MAE [deg]**	12.298	12.282	9.846	10.464	14.894
	MedAE [deg]	8.603	8.575	6.386	6.888	12.726
	Variance [deg^2]	699.354	698.389	446.552	504.977	994.401
geo-kdt	**MAE [deg]**	19.470	19.613	9.817	10.208	12.313
	MedAE [deg]	19.009	19.118	6.265	6.477	7.060
	Variance [deg^2]	1645.443	1665.713	445.951	488.939	704.716
kdt-kdt	**MAE [deg]**	19.696	19.694	9.933	10.095	14.190
	MedAE [deg]	19.180	19.593	6.366	6.359	10.869
	Variance [deg^2]	1678.698	1680.145	456.538	479.515	916.880

to better accuracy by minimizing the number of points in the scan that do not belong to the ground plane. We start the movement of the spherical mobile mapping system from the base of the hill and move it slowly up the front side facing the robotics hall, maintaining a steady pace similar to most previous experiments, except for experiment E (Section 5.2.5). As the robot ascends, we primarily navigate it along the center of the hill to achieve an even distribution of ground area on both sides of the scan. Refer to Figure 5.12 for a visual representation of the conduction of this experiment.

Ground Truth We establish the ground truth for this experiment by combining two Riegl scans of the hill, one from each side, into a single dataset. Based on this dataset, we divide the terrain into three predominantly flat planes with clear inclination differences. We measure the inclination of the ground in front of the hill as 1.91 °, the steep portion of the front side as 10.29 °, and the top plane as 1.84 °. We base these measurements on a sub-sampled point cloud using an octree with voxels no smaller than 10 cm and containing only one random point per voxel to account for noise from plants. Although we cannot validate with laser distance measurements, we rely on the Riegl scan for ground truth, as it has proven accurate based on the physics building ramp experiment, refer to Section 5.2.4.

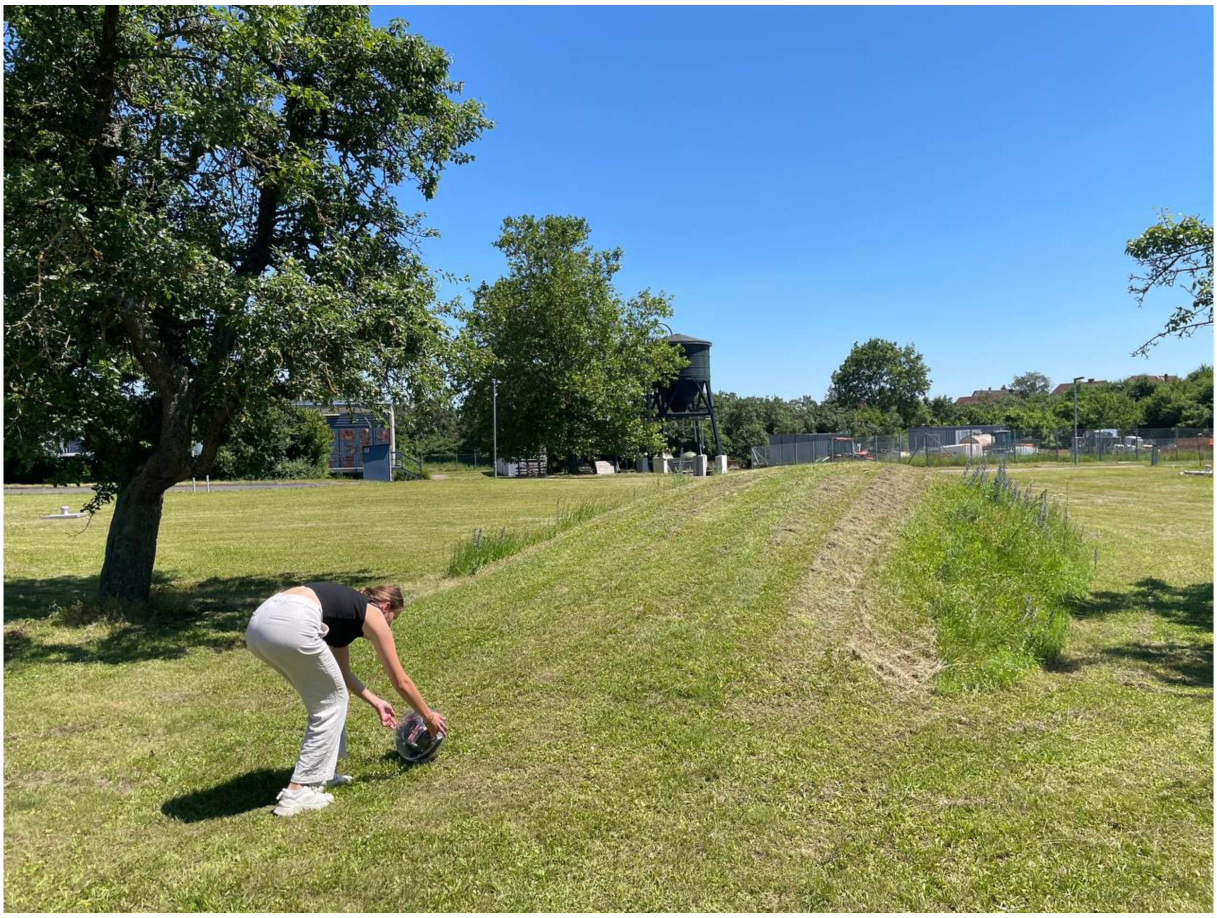
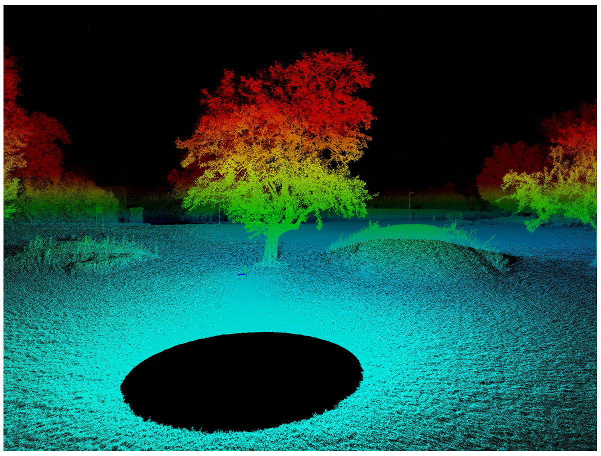


Figure 5.12: Execution of experiment F: outside hill moving up. Photo taken during the experiment, showing the person moving the spherical mobile mapping system up the front side of the hill outside the robotics hall.

Results We present the runtime and plane segmentation error rate (fail rate) results for each approach in Table 5.13, and we summarize the accuracy results in Table 5.14. From the results, we observe that the runtime remains consistent across all approaches. Regarding robustness, we see that the none-geo approach performs best with a fail rate of 0.6 %, followed by the geometrical sub-cloud approaches with a fail rate of approximately 1.1 %. This demonstrates an improvement in robustness compared to experiment D. In terms of accuracy, we find that RANSAC and RHT for filtering approaches yield very similar results, with a MedAE of 5.0 ° and a MAE of approximately 5.2 °, and variances below 70 $(°)^2$. Notably, for the first time, we observe that plane fitting methods (PCA and LSF) achieve comparable accuracy with the geometrical sub-cloud approaches, with a MedAE and MAE of 5.5 ° and a variance of roughly 80.0 $(°)^2$. The none-kdt approach consistently shows the worst performance. We conclude that both RANSAC and RHT with geo-geo or kdt-geo pre-processing provide the best performance. Given their very similar results, it is challenging to distinguish between these two combinations. Figure 5.14 displays the absolute angle error over the number of calls to the callback function for RANSAC with geo-geo pre-processing.

(a) Image of the front side of the hill outside the robotics hall during the measurement of the ground truth using the Riegl scanner.



(b) 3DTK screenshot of the scan taken with the Riegl scanner of the same scenario, illustrating the captured point cloud.

Figure 5.13: Visual comparison of the actual front side of the hill outside the robotics hall and its corresponding 3D scan data used for ground truth determination.

Table 5.13: Runtime and fail rate results of experiment ”outside hill moving up”. Runtimes [ms] of each step of the different pre-processing combinations for each plane segmentation algorithm, as well as the total runtime. The fail rate indicates the percentage of plane segmentation errors.

	Step	**LSF**	**PCA**	**RAN**	**RHT**	**RHT2**	**Avg.**
none-geo	**Cheese Search [ms]**	4.521	2.788	3.016	5.051	2.797	**3.635**
	Delete [ms]	0.806	0.503	0.543	0.908	0.504	**0.653**
	Plane [ms]	7.555	21.282	17.034	3.808	19.564	**13.849**
	Total [ms]	12.883	24.573	20.593	9.767	22.865	**18.136**
	Fail Rate [%]	0.617	0.617	0.493	0.062	1.418	**0.641**
none-kdt	**Build Tree [ms]**	14.387	13.740	13.798	11.928	12.999	**13.370**
	k-NNS [ms]	0.179	0.174	0.165	0.178	0.181	**0.176**
	Delete [ms]	0.654	0.631	0.669	0.605	0.650	**0.642**
	Plane [ms]	0.301	1.147	1.293	2.577	2.256	**1.515**
	Total [ms]	15.521	15.692	15.925	15.288	16.087	**15.703**
	Fail Rate [%]	29.840	31.443	23.859	35.388	52.774	**34.661**
kdt-geo	**Downsample [ms]**	4.739	4.525	4.607	4.179	4.450	**4.500**
	Build Tree [ms]	3.114	2.960	3.015	2.748	2.907	**2.949**
	Radius Search [ms]	0.030	0.029	0.029	0.027	0.028	**0.028**
	Delete [ms]	0.387	0.370	0.374	0.342	0.363	**0.367**
	Cheese Search [ms]	0.727	0.691	0.705	0.644	0.682	**0.690**

Table 5.13: Runtime and fail rate results of experiment ”outside hill moving up”. Runtimes [ms] for each step of the different pre-processing combinations for each plane segmentation algorithm, as well as the total runtime. The fail rate indicates the percentage of plane segmentation errors. (continued)

	Step	**LSF**	**PCA**	**RAN**	**RHT**	**RHT2**	**Avg.**
kdt-geo	**Delete [ms]**	0.211	0.201	0.205	0.186	0.197	**0.200**
	Plane [ms]	0.452	1.799	1.407	1.973	1.783	**1.483**
	Total [ms]	9.659	10.574	10.342	10.098	10.410	**10.217**
	Fail Rate [%]	0.617	0.678	0.493	0.555	2.959	**1.060**
geo-geo	**Downsample [ms]**	5.094	4.923	5.111	4.604	4.913	**4.929**
	Cheese Search [ms]	0.898	0.875	0.921	0.824	0.875	**0.879**
	Delete [ms]	0.238	0.229	0.240	0.214	0.228	**0.230**
	Plane [ms]	0.502	2.017	1.646	2.279	2.058	**1.700**
	Total [ms]	6.731	8.044	7.918	7.921	8.074	**7.738**
	Fail Rate [%]	0.617	0.678	0.493	0.617	3.268	**1.134**
geo-kdt	**Downsample [ms]**	4.847	4.726	4.672	4.185	4.480	**4.582**
	Radius Search [ms]	0.332	0.323	0.297	0.273	0.287	**0.302**
	Delete [ms]	0.423	0.408	0.380	0.348	0.367	**0.385**
	Build Tree [ms]	3.364	3.268	2.998	2.755	2.925	**3.062**
	k-NNS [ms]	0.052	0.051	0.050	0.046	0.048	**0.049**
	Delete [ms]	0.234	0.230	0.213	0.197	0.207	**0.216**
	Plane [ms]	0.409	1.629	1.568	2.640	1.892	**1.628**
	Total [ms]	9.660	10.634	10.179	10.444	10.206	**10.225**
	Fail Rate [%]	26.880	26.942	0.185	1.295	4.316	**11.924**
kdt-kdt	**Downsample [ms]**	4.456	4.253	4.311	3.765	4.143	**4.185**
	Build Tree [ms]	3.130	2.998	2.809	2.496	2.740	**2.834**
	Radius Search [ms]	0.028	0.027	0.027	0.024	0.027	**0.027**
	Delete [ms]	0.388	0.374	0.349	0.311	0.341	**0.353**
	Build Tree [ms]	3.005	2.892	2.689	2.425	2.629	**2.728**
	k-NNS [ms]	0.044	0.041	0.042	0.038	0.042	**0.041**
	Delete [ms]	0.209	0.199	0.190	0.172	0.186	**0.191**
	Plane [ms]	0.369	1.432	1.416	2.278	1.696	**1.438**
	Total [ms]	11.629	12.215	11.834	11.508	11.802	**11.798**
	Fail Rate [%]	26.880	26.880	0.123	1.295	4.316	**11.899**

Table 5.14: MAE [deg], MedAE [deg] and result variance [deg^2] of experiment ”outside hill moving up” for the different plane segmentation algorithms under different pre-processing combinations.

	Result	LSF	PCA	RAN	RHT	RHT2
none-geo	**MAE [deg]**	6.725	7.032	6.839	6.217	8.379
	MedAE [deg]	5.216	5.367	4.681	4.605	6.302
	Variance [deg^2]	177.168	201.085	194.274	147.958	295.611
none-kdt	**MAE [deg]**	12.439	11.604	14.123	10.272	11.271
	MedAE [deg]	7.960	7.272	8.513	5.080	6.307
	Variance [deg^2]	689.916	609.139	912.723	491.282	593.885
kdt-geo	**MAE [deg]**	5.538	5.544	5.252	5.415	6.874
	MedAE [deg]	5.558	5.534	5.010	5.162	5.554
	Variance [deg^2]	81.361	81.962	59.649	71.102	169.457
geo-geo	**MAE [deg]**	5.494	5.508	5.208	5.404	7.096
	MedAE [deg]	5.576	5.549	5.003	5.161	5.762
	Variance [deg^2]	78.880	79.610	57.292	69.990	177.552
geo-kdt	**MAE [deg]**	7.852	7.835	5.032	5.001	5.682
	MedAE [deg]	5.453	5.471	5.130	4.922	4.912
	Variance [deg^2]	270.115	268.484	47.074	51.147	100.420
kdt-kdt	**MAE [deg]**	7.860	7.877	5.045	5.016	5.706
	MedAE [deg]	5.461	5.477	5.085	4.875	5.061
	Variance [deg^2]	270.286	271.479	47.448	51.882	101.624

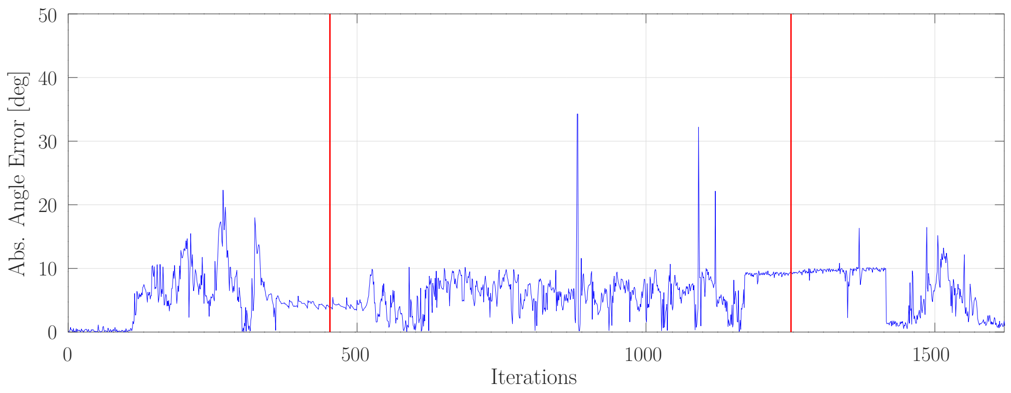
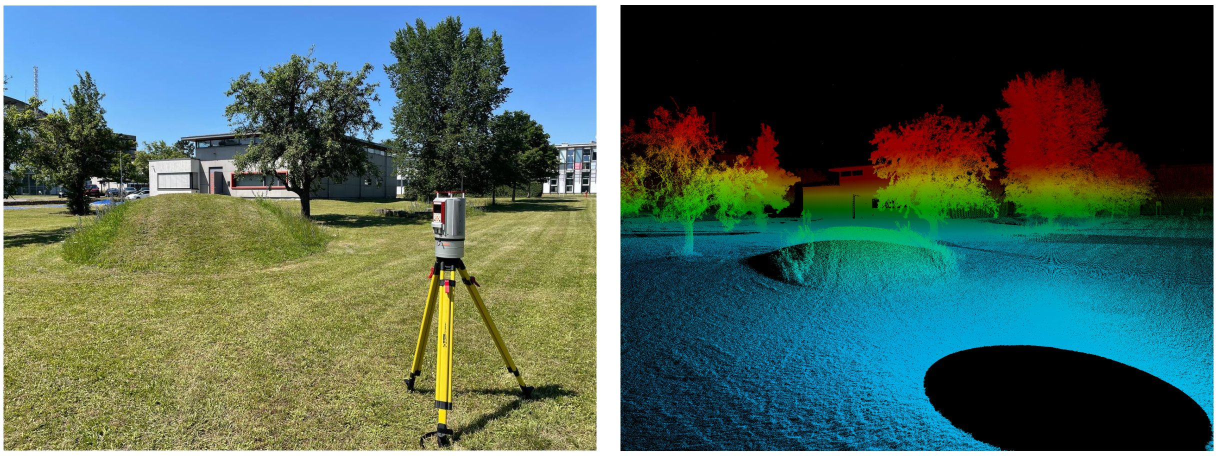


Figure 5.14: Experiment F: Plot of the absolute angle error over iterations, i.e. the number of calls to the callback function, for RANSAC using geo-geo pre-processing. Red lines indicate a change of ground plane and thereby inclination.



(a) Image of the back side of the hill outside the robotics hall during the measurement of the ground truth using the Riegl scanner.

(b) 3DTK screenshot of the scan taken with the Riegl scanner of the same scenario, illustrating the captured point cloud.

Figure 5.15: Visual comparison of the actual back side of the hill outside the robotics hall and its corresponding 3D scan data used for ground truth determination.

5.2.7 Experiment G: Outside Hill Moving Down

In the seventh experiment, we test the spherical mobile mapping system descending the back part of the hill outside the robotics hall (see Figure 5.15). This experiment closely reflects experiment F, refer to Section 5.2.6, with the mowed hill selected to reduce noise in the LiDAR data. We descend the hill at a slow and consistent pace, maintaining uniformity with most previous experiments. Additionally, we guide the robot along the center of the hill to ensure an even distribution of the ground plane across both sides of the scan.

Ground Truth We determine the ground truth for this experiment by analyzing the same combined and reduced Riegl scan of experiment F of the hill. Refer to Figure 5.15 for a visual representation. We divide the back of the hill into four different planes. We measure the inclination of the top of the hill as 1.84 °, the upper inclination of the back of the hill as 11.90 °, the steepest part as 19.31 ° and the ground in front of the back of the hill as 1.75 °.

Results The results of this experiment, including runtime and plane segmentation error rates for each approach, are presented in Table 5.15. The accuracy of the results is summarized in Table 5.16. We observe that the runtime remains consistent with previous experiments. Regarding robustness, we find that the none-geo approach performs the best with a fail rate of 0.9 %, followed by the geometrical sub-cloud approaches with 1.8 % for kdt-geo and 2.7 % for geo-geo. This difference of 0.9 % between the fail rate of kdt-geo and geo-geo is the biggest we have observed so far. The none-kdt approach shows the worst performance. This trend aligns with our observations in earlier experiments. In terms of accuracy, the results are very similar to those in experiment F, with an insignificant lower MedAE for plane detection by approximately 0.3 °. The plane fitting algorithms also show strong performance for geometrical approaches, maintaining the same trend observed in experiment F. We conclude that the best overall performance remains consistent with experiment F, with RANSAC and RHT using geo-geo or kdt-geo pre-processing providing suitable results. Figure 5.16 displays the absolute angle error over the number of calls to the callback function for RANSAC with geo-geo pre-processing.

Table 5.15: Runtime and fail rate results of experiment "outside hill moving down". Runtimes [ms] of each step of the different pre-processing combinations for each plane segmentation algorithm, as well as the total runtime. The fail rate indicates the percentage of plane segmentation errors.

	Step	LSF	PCA	RAN	RHT	RHT2	Avg.
none-geo	**Cheese Search [ms]**	4.244	1.967	2.324	5.128	2.278	**3.188**
	Delete [ms]	0.749	0.349	0.411	0.901	0.400	**0.562**
	Plane [ms]	10.396	25.872	21.162	4.796	23.246	**17.094**
	Total [ms]	15.389	28.188	23.897	10.825	25.924	**20.845**
	Fail Rate [%]	0.797	1.087	0.217	1.014	1.449	**0.913**
none-kdt	**Build Tree [ms]**	13.923	13.260	13.252	11.430	12.432	**12.859**
	k-NNS [ms]	0.144	0.145	0.144	0.149	0.152	**0.147**
	Delete [ms]	0.625	0.607	0.634	0.562	0.604	**0.606**
	Plane [ms]	0.246	1.125	1.247	2.797	2.145	**1.512**
	Total [ms]	14.938	15.137	15.278	14.937	15.333	**15.124**
	Fail Rate [%]	30.507	31.449	24.855	33.551	48.043	**33.681**
kdt-geo	**Downsample [ms]**	4.690	4.549	4.584	4.196	4.512	**4.506**
	Build Tree [ms]	2.604	2.510	2.532	2.360	2.493	**2.500**
	Radius Search [ms]	0.036	0.029	0.029	0.027	0.029	**0.030**
	Delete [ms]	0.311	0.300	0.304	0.283	0.297	**0.299**
	Cheese Search [ms]	0.624	0.604	0.612	0.572	0.602	**0.603**
	Delete [ms]	0.170	0.165	0.168	0.157	0.164	**0.165**
	Plane [ms]	0.418	1.978	1.507	2.159	1.844	**1.581**
	Total [ms]	8.854	10.135	9.736	9.754	9.941	**9.684**
	Fail Rate [%]	1.377	1.377	1.232	2.464	3.043	**1.899**
geo-geo	**Cheese Search [ms]**	0.815	0.767	0.764	0.695	0.763	**0.761**
	Delete [ms]	0.207	0.193	0.191	0.174	0.191	**0.191**
	Plane [ms]	0.487	2.258	1.682	2.347	1.829	**1.720**
	Total [ms]	6.767	8.188	7.571	7.682	7.671	**7.576**
	Fail Rate [%]	1.377	1.377	1.232	1.304	8.696	**2.797**

Table 5.15: Runtime and fail rate results of experiment "outside hill moving down". Runtimes [ms] for each step of the different pre-processing combinations for each plane segmentation algorithm, as well as the total runtime. The fail rate indicates the percentage of plane segmentation errors. (continued)

	Step	LSF	PCA	RAN	RHT	RHT2	Avg.
geo-kdt	**Downsample [ms]**	4.739	4.727	4.617	4.107	4.535	**4.545**
	Radius Search [ms]	0.274	0.272	0.241	0.219	0.244	**0.250**
	Delete [ms]	0.346	0.345	0.307	0.278	0.309	**0.317**
	Build Tree [ms]	2.875	2.871	2.535	2.299	2.570	**2.630**
	k-NNS [ms]	0.051	0.051	0.050	0.045	0.050	**0.049**
	Delete [ms]	0.194	0.194	0.172	0.157	0.174	**0.178**
	Plane [ms]	0.356	1.710	1.864	3.064	2.018	**1.802**
	Total [ms]	8.837	10.169	9.786	10.168	9.901	**9.772**
	Fail Rate [%]	18.696	18.696	1.304	2.391	7.391	**9.696**
kdt-kdt	**Downsample [ms]**	4.524	4.431	4.371	3.835	4.252	**4.283**
	Build Tree [ms]	2.766	2.719	2.421	2.148	2.447	**2.500**
	Radius Search [ms]	0.029	0.029	0.028	0.025	0.027	**0.028**
	Delete [ms]	0.328	0.324	0.291	0.258	0.292	**0.299**
	Build Tree [ms]	2.639	2.595	2.296	2.079	2.335	**2.389**
	k-NNS [ms]	0.044	0.044	0.042	0.038	0.042	**0.042**
	Delete [ms]	0.177	0.174	0.156	0.142	0.159	**0.162**
	Plane [ms]	0.335	1.561	1.748	2.931	1.854	**1.686**
	Total [ms]	10.842	11.877	11.354	11.456	11.408	**11.387**
	Fail Rate [%]	18.696	18.841	1.304	1.957	8.406	**9.841**

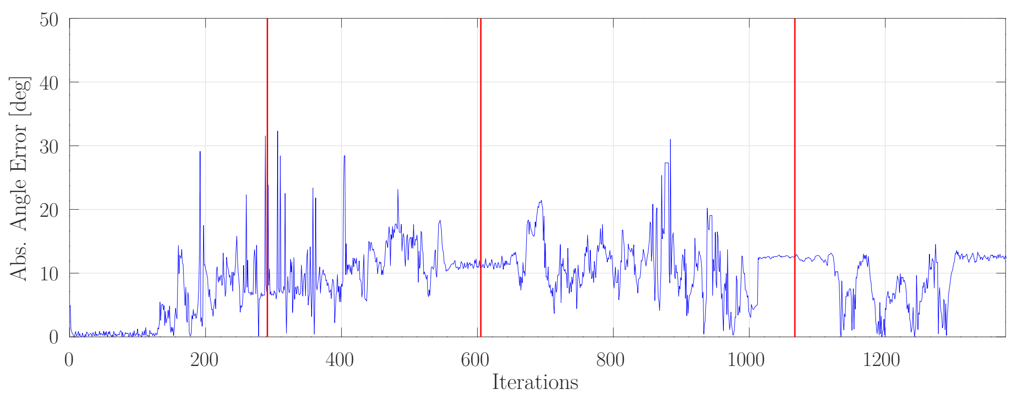


Figure 5.16: Experiment G: Plot of the absolute angle error over iterations, i.e. the number of calls to the callback function, for RANSAC using geo-geo pre-processing. Red lines indicate a change of ground plane and thereby inclination.

Table 5.16: MAE [deg], MedAE [deg] and result variance [deg^2] of experiment ”outside hill moving down” for the different plane segmentation algorithms under different pre-processing combinations.

	Result	LSF	PCA	RAN	RHT	RHT2
none-geo	**MAE [deg]**	9.339	9.789	9.589	9.452	10.745
	MedAE [deg]	7.242	7.675	7.815	7.156	8.545
	Variance [deg^2]	372.754	412.831	397.565	378.145	523.177
none-kdt	**MAE [deg]**	12.348	13.056	13.837	12.042	10.600
	MedAE [deg]	9.290	10.976	12.439	11.210	8.454
	Variance [deg^2]	653.763	750.205	843.210	614.273	471.531
kdt-geo	**MAE [deg]**	6.223	6.253	5.453	5.942	7.746
	MedAE [deg]	5.391	5.409	4.756	4.864	5.894
	Variance [deg^2]	147.120	148.819	112.600	134.848	257.731
geo-geo	**MAE [deg]**	6.152	6.199	5.428	5.812	8.612
	MedAE [deg]	5.350	5.369	4.763	4.897	6.271
	Variance [deg^2]	142.788	145.360	111.478	129.461	336.258
geo-kdt	**MAE [deg]**	9.253	9.246	5.289	6.043	7.050
	MedAE [deg]	7.642	7.663	4.524	4.764	5.761
	Variance [deg^2]	370.709	369.596	93.985	138.705	196.301
kdt-kdt	**MAE [deg]**	9.278	9.210	5.299	6.130	6.992
	MedAE [deg]	7.737	7.705	4.524	4.816	5.762
	Variance [deg^2]	373.012	365.870	95.152	143.768	190.306

5.2.8 Experiment H: Tunnel

In the eighth and final remote experiment, we test the spherical mobile mapping system inside the tunnel located under the hill outside the robotics hall, as shown in Figure 5.17. The tunnel consists of concrete pipes with an inside diameter of 99.5 cm, which are embedded in the hill. We start by positioning the robot approximately 1.0 m in front of the tunnel’s entrance. Then we move it through the tunnel, and continue its movement a short distance beyond the exit to capture the full extent of the tunnel’s interior. Throughout the experiment, we maintain a slow and steady pace, similar to most prior experiments, to ensure consistent data collection.

Ground Truth We determine the ground truth based on the assumption that the tunnel’s ground is level, as it consists of concrete pipes, placed by humans and embedded in the hill. We verify this assumption with a water scale, confirming the ground has a 0.0 ° inclination. Thus, we apply the same ground truth for this experiment as in previous experiments, using a normal vector of $\boldsymbol{n}_{gt} = [0, 0, -1]^T$ and an inclination of 0.0 °.

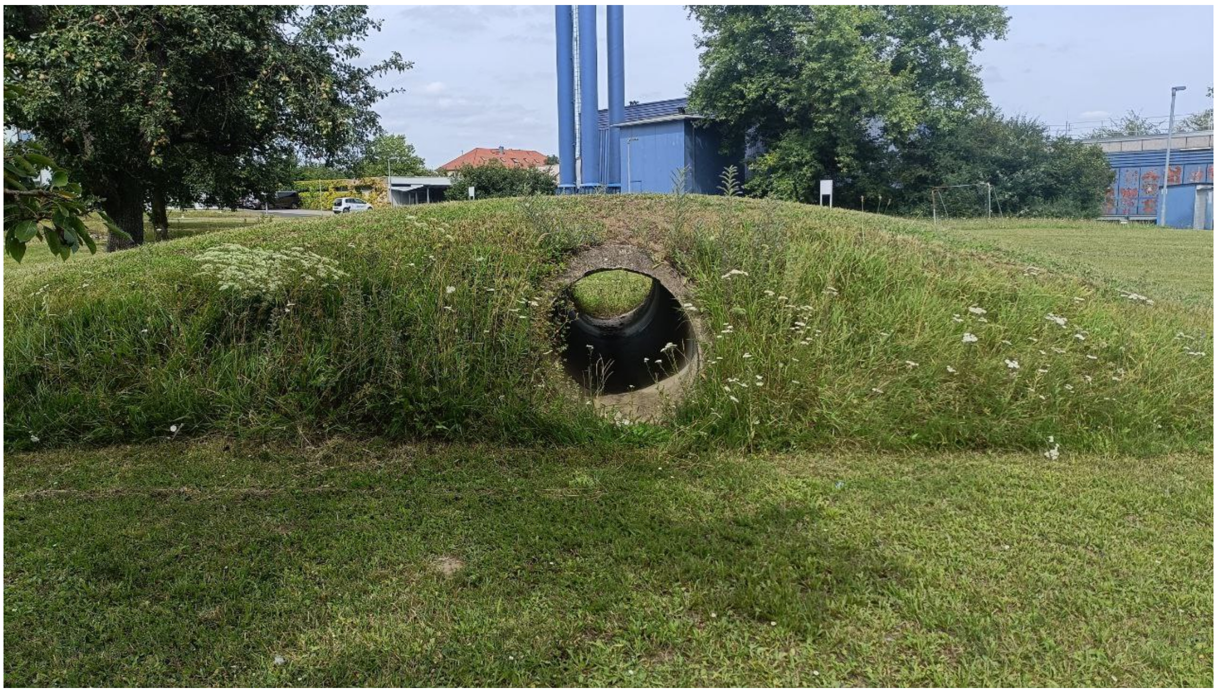


Figure 5.17: Image of the tunnel located under the hill outside the robotics hall.

Results We present the results of this experiment, regarding the runtime and plane segmentation error rates for each method, in Table 5.17. A summary of the accuracy results is provided in Table 5.18. In experiment H, we observe that the runtime remains consistent with previous experiments, showing similar values across all approaches. However, the robustness is notably poor, with the none-geo approach, as the again best performing approach, exhibiting the worst fail rate across all experiments at 6.0 %. All other pre-processing approaches lead to fail rates exceeding 25 %. The accuracy is also severely compromised in this experiment. We observe very similar values for all approaches, with the MAE ranging from 25.0 ° to 19.0 ° and the MedAE varying between 22.8 ° and 16.0 °. Variances are all above 1600 $(°)^2$, marking the worst accuracy observed in any of the experiments. Overall, while none-geo has the highest runtime, it shows the lowest fail rate and outperforms other approaches in terms of robustness. Among plane segmentation methods, the plane fitting algorithms, particularly LSF and PCA, provides better accuracy than plane detection methods, except for RHT, for this pre-processing approach, although all methods yield suboptimal results. Therefore, the best solution for this experiment is RHT for none-geo due to the comparatively small errors and variance, as well as due to its robustness and despite its runtime performance. Figure 5.18 displays the absolute angle error over the number of calls to the callback function for RANSAC with geo-geo pre-processing.

Table 5.17: Runtime and fail rate results of experiment "tunnel". Runtimes [ms] of each step of the different pre-processing combinations for each plane segmentation algorithm, as well as the total runtime. The fail rate indicates the percentage of plane segmentation errors.

	Step	LSF	PCA	RAN	RHT	RHT2	Avg.
none-geo	**Cheese Search [ms]**	4.693	2.194	2.170	5.156	2.240	**3.290**
	Delete [ms]	0.763	0.365	0.363	0.851	0.385	**0.545**
	Plane [ms]	10.017	27.266	29.819	6.477	27.558	**20.227**
	Total [ms]	15.472	29.825	32.351	12.484	30.182	**24.063**
	Fail Rate [%]	7.259	7.692	4.767	4.334	5.959	**6.002**
none-kdt	**Build Tree [ms]**	13.610	13.210	12.982	11.229	12.147	**12.635**
	k-NNS [ms]	0.141	0.135	0.123	0.126	0.133	**0.132**
	Delete [ms]	0.678	0.670	0.663	0.592	0.616	**0.644**
	Plane [ms]	0.227	1.042	1.120	2.981	1.985	**1.471**
	Total [ms]	14.656	15.056	14.888	14.929	14.881	**14.882**
	Fail Rate [%]	45.937	42.362	34.453	54.821	63.705	**48.256**
kdt-geo	**Downsample [ms]**	5.300	5.263	4.892	4.245	4.847	**4.909**
	Build Tree [ms]	0.720	0.699	0.682	0.610	0.810	**0.704**
	Radius Search [ms]	0.033	0.030	0.028	0.024	0.028	**0.029**
	Delete [ms]	0.106	0.104	0.101	0.090	0.114	**0.103**
	Cheese Search [ms]	0.220	0.215	0.210	0.184	0.244	**0.215**
	Delete [ms]	0.055	0.054	0.052	0.047	0.060	**0.054**
	Plane [ms]	0.319	1.414	3.159	4.395	2.848	**2.427**
	Total [ms]	6.754	7.779	9.123	9.595	8.952	**8.441**
	Fail Rate [%]	20.043	20.043	22.319	20.585	43.987	**25.395**
geo-geo	**Downsample [ms]**	4.715	5.192	4.883	4.316	4.940	**4.809**
	Cheese Search [ms]	0.239	0.243	0.235	0.212	0.288	**0.243**
	Delete [ms]	0.061	0.062	0.061	0.054	0.071	**0.062**
	Plane [ms]	0.303	1.462	3.354	4.340	2.830	**2.458**
	Total [ms]	5.317	6.959	8.533	8.922	8.130	**7.572**
	Fail Rate [%]	20.043	19.935	22.210	19.827	46.046	**25.612**

Table 5.17: Runtime and fail rate results of experiment ”tunnel”. Runtimes [ms] for each step of the different pre-processing combinations for each plane segmentation algorithm, as well as the total runtime. The fail rate indicates the percentage of plane segmentation errors. (continued)

	Step	LSF	PCA	RAN	RHT	RHT2	Avg.
geo-kdt	**Downsample [ms]**	5.342	5.272	4.503	3.457	4.806	**4.676**
	Radius Search [ms]	0.075	0.074	0.067	0.047	0.081	**0.069**
	Delete [ms]	0.109	0.108	0.096	0.069	0.113	**0.099**
	Build Tree [ms]	0.732	0.720	0.660	0.463	0.787	**0.672**
	k-NNS [ms]	0.053	0.052	0.045	0.034	0.048	**0.046**
	Delete [ms]	0.060	0.058	0.054	0.037	0.061	**0.054**
	Plane [ms]	0.368	1.726	7.171	8.583	4.102	**4.390**
	Total [ms]	6.739	8.009	12.595	12.691	9.997	**10.006**
	Fail Rate [%]	35.428	35.753	32.178	26.111	50.704	**36.035**
kdt-kdt	**Downsample [ms]**	5.262	5.114	4.274	3.445	4.612	**4.541**
	Build Tree [ms]	0.734	0.712	0.621	0.475	0.773	**0.663**
	Radius Search [ms]	0.030	0.029	0.025	0.020	0.026	**0.026**
	Delete [ms]	0.107	0.105	0.091	0.070	0.108	**0.096**
	Build Tree [ms]	0.674	0.661	0.572	0.438	0.718	**0.613**
	k-NNS [ms]	0.047	0.046	0.039	0.031	0.041	**0.041**
	Delete [ms]	0.057	0.056	0.048	0.037	0.058	**0.051**
	Plane [ms]	0.358	1.676	6.783	8.635	4.014	**4.293**
	Total [ms]	7.268	8.400	12.453	13.150	10.349	**10.324**
	Fail Rate [%]	35.428	35.753	32.286	26.436	50.163	**36.013**

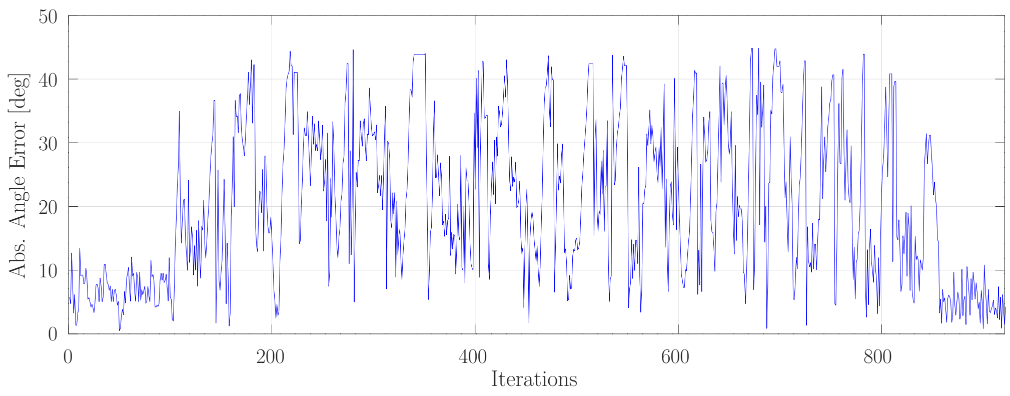


Figure 5.18: Experiment H: Plot of the absolute angle error over iterations, i.e. the number of calls to the callback function, for RHT using none-geo pre-processing. Red lines indicate a change of ground plane and thereby inclination.

Table 5.18: MAE [deg], MedAE [deg] and result variance [deg^2] of experiment "tunnel" for the different plane segmentation algorithms under different pre-processing combinations.

	Result	**LSF**	**PCA**	**RAN**	**RHT**	**RHT2**
none-geo	**MAE [deg]**	19.162	19.456	20.436	19.392	21.296
	MedAE [deg]	16.035	16.590	20.023	17.851	20.928
	Variance [deg^2]	1612.408	1660.973	1797.753	1641.891	1987.075
none-kdt	**MAE [deg]**	22.098	21.917	23.003	21.428	19.987
	MedAE [deg]	22.088	19.780	23.321	21.180	18.957
	Variance [deg^2]	2141.381	2115.978	2294.861	2024.684	1798.037
kdt-geo	**MAE [deg]**	20.692	20.788	20.899	20.951	21.733
	MedAE [deg]	19.402	19.620	19.195	19.748	22.093
	Variance [deg^2]	1860.662	1874.603	1912.043	1904.528	2086.534
geo-geo	**MAE [deg]**	20.614	20.178	20.629	20.837	22.329
	MedAE [deg]	19.470	18.189	18.287	18.806	22.515
	Variance [deg^2]	1846.834	1778.773	1871.920	1886.189	2170.066
geo-kdt	**MAE [deg]**	19.614	19.523	22.042	20.765	24.484
	MedAE [deg]	18.989	18.806	20.305	18.150	22.533
	Variance [deg^2]	1653.261	1638.067	2101.900	1855.286	2523.215
kdt-kdt	**MAE [deg]**	19.600	19.549	22.006	20.999	24.942
	MedAE [deg]	18.941	18.963	20.314	18.701	22.775
	Variance [deg^2]	1650.494	1641.983	2094.925	1897.027	2609.148

5.3 Comparative Analysis of Remote Experiments

In this section, we evaluate the performance of the 30 different approaches, by assessing their performance over all previously performed experiments with regard to runtime, accuracy, and robustness. We specifically measure robustness as the fail rate, calculating it as the ratio of plane segmentation errors to the total number of callback function calls triggered by ROS topic `/lidar/points_undistorted` messages. We define the real-time operation as a runtime faster than 50 ms, corresponding to the frequency of the Hesai Pandar LiDAR at 20 Hz. First, we analyze the runtime of different pre-processing methods, followed by an evaluation of plane segmentation algorithms in combination with the different pre-processing approaches.

5.3.1 Pre-Processing Runtime Evaluation

The pre-processing phase plays an important role in the overall performance of the different approaches in terms of runtime. Table 5.19 lists the average runtime of all pre-processing methods, the complexity notation for each method, as well as the average total runtime of the callback function across all experiments and plan segmentation algorithms.

Based on these results, we see that among the different pre-processing methods tested, the combination of none-geo proves to be the fastest, with an average runtime of 3.862 ms. This is due to the fact, that the none-geo approach only needs to traverse the entire point cloud once, which results in a time complexity of $\mathcal{O}(n)$ with n being the number of points before the downsampling. This single pass has an average runtime under 3.639 ms in all experiments. All other combinations with a filtering step include the downsampling, which also has a time complexity of $\mathcal{O}(n)$ with n being the number of points before the downsampling. Therefore, all subsequent processing steps are additional time-consuming work and lead to a higher runtime compared to none-geo. However, after the downsampling, the number of points gets reduced from $30,000$ points to roughly $4,000$ points, which decreases the runtime of all following steps. The none-kdt approach, which also does not benefit from the downsampling, has the slowest performance with an average runtime of 13.627 ms. The creation of the kd-tree, which has a time complexity of $\mathcal{O}(n \log n)$, remains the most time-consuming step for all approaches using a kd-tree. The high runtime is due to the fact that in this approach we build a kd-tree from all $30,000$ points, which alone results in an average runtime of more than 12.272 ms. In all other approaches using a kd-tree and with a time complexity of $\mathcal{O}(n \log n)$, the tree is constructed after downsampling, using fewer points, and therefore making it faster. A closer look at the other combinations employing a kd-tree shows that kdt-geo is slower than geo-kdt in terms of pure pre-processing time. This difference occurs as a result of building the kd-tree with different point clouds. When we build the kd-tree for filtering, the point cloud has a slightly higher number of points compared to the sub-cloud creation step, which leads to the higher runtime. Consequently, the kdt-kdt approach is the slowest approach with filtering, as we have to build the kd-tree twice, which is reflected in the results.

Although none-geo is the fastest pre-processing approach, without the downsampling and the limitation of the number of points in the sub-cloud, it is the slowest method overall. The high number of points in the sub-cloud leads to the longest runtimes for the plane segmentation algorithms. Conversely, the none-kdt approach benefits from the restriction on the number of points in the sub-cloud, resulting in faster overall runtime compared to none-geo. As the final result, we prove the geo-geo combination to be the fastest overall with 7.959 ms and the second fastest in pre-processing with 6.157 ms. This result follows naturally from the time complexity analysis, which shows that both geo-geo and none-geo are the only pre-processing methods with a runtime complexity of $\mathcal{O}(n)$, while all other methods exhibit a complexity of $\mathcal{O}(n \log n)$. Also, the higher total runtime of the geo-kdt approach compared to kdt-geo indicates that the geometrical approach utilizing the cheese shape is superior to the kd-tree approach as a sub-cloud creation method in terms of runtime.

Table 5.19: Final results of the pre-processing analysis. Average runtime and time complexity of pre-processing methods and total callback function execution time, averaged across all experiments and plane segmentation algorithms.

Avg. Runtime	**none-geo**	**none-kdt**	**kdt-geo**	**geo-geo**	**geo-kdt**	**kdt-kdt**
Pre-Proc. [ms]	**3.862**	13.627	7.917	6.157	7.720	8.951
Complexity	$\mathcal{O}(n)$	$\mathcal{O}(n \log n)$	$\mathcal{O}(n \log n)$	$\mathcal{O}(n)$	$\mathcal{O}(n \log n)$	$\mathcal{O}(n \log n)$
Total [ms]	21.343	15.193	9.572	**7.959**	10.031	11.122

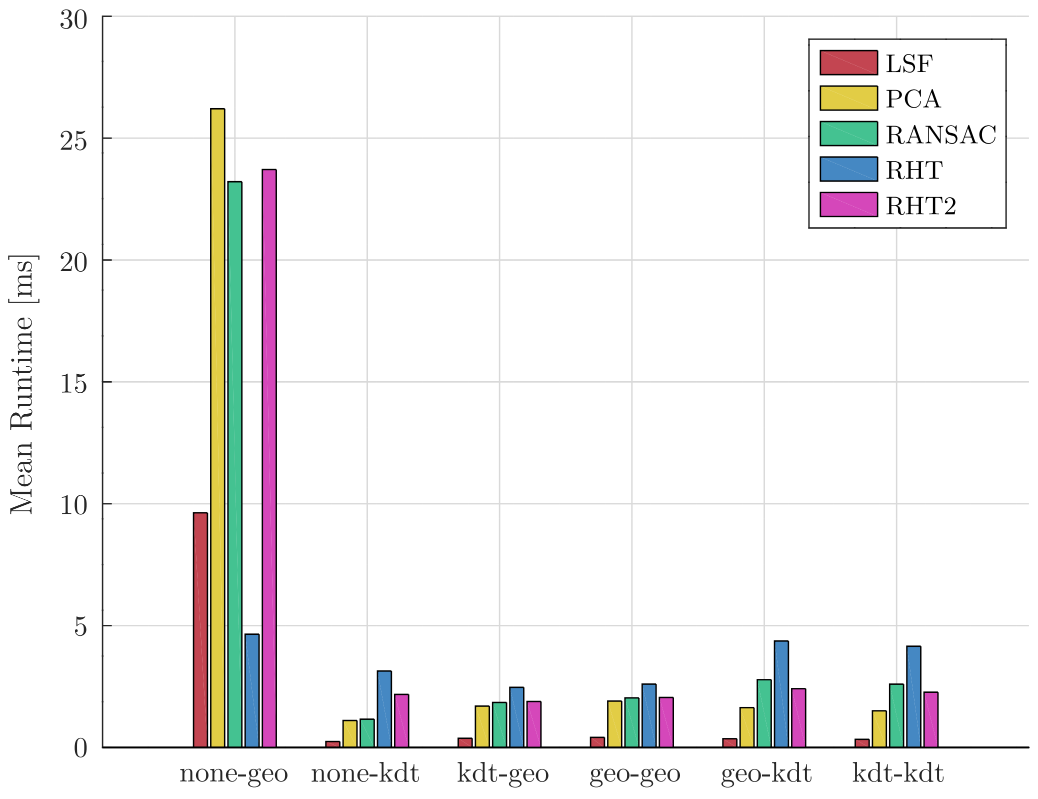


Figure 5.19: Final results of the plane segmentation runtime analysis. Average runtime [ms] for different plane segmentation algorithms under various pre-processing methods across all experiments.

5.3.2 Plane Segmentation Analysis

The evaluation of the plane segmentation algorithms involves analyzing runtime, accuracy and robustness to understand their performance under various pre-processing methods. To illustrate these findings, we provide Figure 5.19 and Table 5.20. Figure 5.19 presents the final results of the plane segmentation runtime analysis, showing the average runtime [ms] across all experiments for the different plane segmentation algorithms for all pre-processing approaches. Table 5.20 offers a more comprehensive summary of the plane segmentation analysis, detailing the total average runtime [ms], MAE [deg], MedAE [deg], and fail rate [%] for each plane segmentation algorithm under different pre-processing combinations across all experiments.

From the results, we conclude that both pre-processing approaches without filtering are unsuitable for our application. Although the none-geo approach results in the lowest fail rates ($< 5.0\,\%$) for all plane segmentation algorithms, it leads to the longest runtimes of all algorithms, as already explained in Section 5.3.1. This is both due to the significantly higher number of points within the sub-cloud compared to the other pre-processing combinations. These higher

Table 5.20: Final results of the plane segmentation analysis. Total average runtime [ms], MAE [deg], MedAE [deg] and result variance [deg^2] for the different plane segmentation algorithms under different pre-processing combinations across all experiments. Red: Worst value for plane segmentation algorithm. Green: Best overall combination. Bold: Highest accuracy.

	Avg. Results	LSF	PCA	RAN	RHT	RHT2
none-geo	**Runtime [ms]**	9.629	26.205	23.212	4.647	23.712
	MAE [deg]	13.933	14.616	13.353	12.786	15.870
	MedAE [deg]	12.036	13.227	11.769	10.796	14.509
	Fail rate [%]	3.795	4.159	2.277	1.967	4.820
none-kdt	**Runtime [ms]**	0.244	1.107	1.161	3.138	2.178
	MAE [deg]	18.340	17.950	19.413	17.308	16.627
	MedAE [deg]	16.137	15.112	17.869	15.267	13.705
	Fail rate [%]	42.790	40.896	31.458	46.135	57.022
kdt-geo	**Runtime [ms]**	0.378	1.698	1.847	2.468	1.888
	MAE [deg]	12.236	12.294	**9.612**	10.516	14.413
	MedAE [deg]	10.356	10.392	**8.475**	8.844	12.466
	Fail rate [%]	5.951	5.926	3.858	4.481	11.415
geo-geo	**Runtime [ms]**	0.415	1.904	2.035	2.600	2.052
	MAE [deg]	12.191	12.187	**9.525**	10.432	14.382
	MedAE [deg]	10.317	10.228	**8.305**	8.727	12.264
	Fail rate [%]	5.940	5.923	3.898	4.320	12.210
geo-kdt	**Runtime [ms]**	0.359	1.635	2.781	4.369	2.412
	MAE [deg]	13.580	13.584	10.023	**9.560**	12.538
	MedAE [deg]	11.170	11.224	8.577	**8.099**	9.772
	Fail rate [%]	33.028	33.173	6.047	8.291	20.824
kdt-kdt	**Runtime [ms]**	0.334	1.505	2.598	4.153	2.268
	MAE [deg]	13.584	13.575	10.049	**9.559**	12.872
	MedAE [deg]	11.267	11.320	8.604	**8.159**	10.033
	Fail rate [%]	33.076	33.211	6.063	8.253	21.755

runtimes for the none-geo approach are clearly evident in Figure 5.19. The none-kdt approach is unsuitable due to its consistently high fail rates for all plane segmentation algorithms. These plane segmentation faults occur because, without the downsampling, the point cloud has a higher point density, which leads to a much smaller covered area with the same k. The smaller area also increases the noise-to-ground ratio in the sub-cloud. In addition, the MAEs and MedAEs reach their highest values for the none-kdt approach, followed by the none-geo approach. For none-geo, the errors are substantially higher compared to approaches that include filtering for RANSAC and RHT. However, the increase in error is only slight for the other plane segmentation algorithms.

In Figure 5.19, we clearly see that the sub-cloud creation method primarily determines the runtime of the plane segmentation algorithms, not the filtering step. We observe similar runtime bars for the geo-geo and kdt-geo combinations, as well as for kdt-kdt and geo-kdt, with only the none-kdt approach showing a slightly different pattern. The none-geo approach stands out as an exception. This trend is also apparent in Table 5.20, particularly for the algorithms RANSAC and RHT. However, the runtimes of PCA, LSF and RHT2 follow this pattern less consistently. When we directly compare the geometrical sub-cloud approach with the kd-tree sub-cloud approach, we find that the geometrical approach results in lower runtimes for all plane detection algorithms (RANSAC, RHT and RHT2), and slightly higher runtimes for the plane fitting algorithms (LSF and PCA). We attribute this to the fact that the geometrical approach more effectively represents the ground plane, making it more likely to identify the dominant plane as the ground plane. This reduces the number of iterations required for plane detection, thereby speeding up the process. In contrast, the plane fitting algorithms perform only a single attempt at fitting a plane. The restricted number of points in the kd-tree sub-cloud approach allows faster plane fitting, but it does not benefit plane detection algorithms, where fewer points result in less comprehensive coverage and longer detection times. In general, the sub-cloud approach affects the runtimes of the plane detection algorithms more than the plane fitting algorithms. In addition to the lower runtime, the geometrical approach results in significantly lower fail rates than the kd-tree sub-cloud creation across all plane segmentation algorithms. This is particularly evident in the results of LSF and PCA, which have very similar fail rates for all four pre-processing methods with filtering. Both have a fail rate of 5.9 % for the geometric approach and a fail rate of approximately 33.1 % for the kd-tree approach. We attribute this significant difference to the higher noise-to-ground ratio associated with the smaller area represented in the kd-tree sub-cloud approach. This observation aligns with previous research, which indicates that plane fitting is more susceptible to noise compared to plane detection, as discussed in Section 2.2. We also find this similarity between LSF and PCA for the MAE and the MedAE. Here, the plane fitting algorithms perform slightly better for the geometrical sub-cloud approach with a MAE of approximately 12.2 ° and a MedAE of 12.3 °. For the kd-tree sub-cloud approach, the MAE increases by 1.3 ° and the MedAE by 0.9 °. This pattern aligns with the fail rate observations, indicating that the reduced accuracy and higher fail rates for the kd-tree approach are linked to the same factors (noise-to-ground ratio) affecting the plane fitting algorithms. Overall, the geometrical sub-cloud approach outperforms the kd-tree method due to its significantly lower fail rates, and reduced total runtime of the callback function, as stated in Section 5.3.1.

Among the plane segmentation algorithms, LSF consistently performs the fastest in terms of runtime, with a maximum runtime of 0.415 ms. Whereas none of the other plane segmentation algorithms run in less than 1.1 ms. The only exception is the none-geo approach, where the high point count affects its performance. Here, the RHT significantly outperforms all other algorithms due to its randomized selection process, which mitigates the impact of the high number of points on runtime. Following LSF in runtime is PCA, which shows that plane fitting algorithms are generally faster than plane detection algorithms. This finding is consistent with previous research, refer to Section 2.2. Next come RANSAC and RHT2, both of which require 1.8 ms for the kdt-geo and 2.0 ms for the geo-geo pre-processing combinations. But RHT2 is slightly faster than RANSAC by 0.3 ms for the kd-tree sub-cloud approach. The slowest algorithm is RHT with runtimes of about 2.5 ms for the geometric and 4.2 ms for the kd-tree sub-cloud approach. When evaluating the accuracy of the plane segmentation algorithms, we find that LSF and PCA only compete effectively with RANSAC and RHT for the approaches without filtering. But overall, the plane fitting algorithms have higher MAEs and MedAEs compared to the plane detection algorithms, with the notable exception of RHT2. When we directly compare RHT2 with the two plane fitting algorithms, it proves to be more accurate for the kd-tree sub-cloud approaches, supporting the conclusion that plane detection methods are better suited for noisy data. Conversely, in the geometrical sub-cloud approaches, where the points represent the ground more accurately (resulting in a lower noise-to-ground ratio), the plane fitting algorithms outperform RHT2 in terms of accuracy by roughly 2.0 ° for the MAEs and MedAEs. Furthermore, the RHT2 algorithm consistently shows higher MAEs compared to both RANSAC and RHT. For the geometrical sub-cloud, RHT2 records a MAE of 14.4 ° and a MedAE of 12.3 °. We observe slightly lower values for the kd-tree sub-cloud, with MAEs of approximately 12.7 ° and MedAEs of 10.0 °. We attribute this to the double process of point reduction before applying PCA: first by creating the sub-cloud, then by selecting only the inliers found by the RHT2 algorithm within a specified plane threshold (refer to Table 5.1). This approach results in fewer points in an already sparse dataset. Due to the parameterization of the RHT2 algorithm, it sometimes leaves fewer than three points for PCA, which further leads to higher fail rates than RHT.

For the two geometrical sub-cloud approaches, RANSAC emerges as the most accurate, with a MAE of approximately 9.6 ° and a MedAE of 8.4 °. Whereas RHT leads in accuracy for the kd-tree sub-cloud approaches, showing very similar MAE and MedAE values (9.6 ° for MAE and 8.1 ° for MedAE) for both kd-tree based sub-cloud approaches. Conversely, RANSAC records higher MAEs of 10.3 ° and MedAE of 8.6 ° for the kd-tree sub-cloud approaches, whereas RHT maintains comparable accuracy for the geometrical sub-cloud approaches. Despite these trends, the difference in MedAE between RHT for the kd-tree sub-cloud and RANSAC for the geometrical sub-cloud is minimal, with RHT having a lower MedAE by just 0.3 °, which we define as statistically insignificant. However, when considering both runtime and fail rate, RANSAC consistently outperforms RHT, regardless of the sub-cloud method used, with the none-geo approach being the only exception. In nearly all cases, RANSAC achieves the lowest fail rate, making it the most reliable option. Given the pre-processing runtime results, the best overall combination is geo-geo with RANSAC.

Comparison of RANSAC-PCA and RANSAC-LSF Based on these results, we disprove the original assumption that PCA and LSF perform equally well for the pcl, refer to Section 2.2.1. The algorithm LSF is significantly faster than PCA in the pcl implementation. To investigate this further, we implement the combination of RANSAC-LSF and compare its performance with that of the already established version RANSAC-PCA. We do not implement the RHT-LSF combination, due to the poor results previously observed with RHT2, which already disqualify it as a candidate for any suitable solution. Table 5.21 lists the results of RANSAC-PCA and the new ones of RANSAC-LSF for the geo-geo pre-processing approach for each experiment.

We see from the results, that the runtime of RANSAC-PCA with 2.0 ms is about twice as long as that of RANSAC-LSF with a runtime of 1.1 ms. Furthermore, RANSAC-LSF also has a slightly better accuracy with a lower average MAE and MedAE of roughly 0.5 ° and a lower

Table 5.21: Final results of the direct comparison of RANSAC-PCA and RANSAC-LSF. Pre-processing time [ms], plane segmentation time [ms], total runtime [ms], fail rate [%], MAE [deg], MedAE [deg], and variance [deg^2] for each remote experiment (A through H) and their respective average value.

	Avg. Results	A	B	C	D	E	F	G	H	Avg.
RAN-PCA	**Prepr. [ms]**	6.479	6.960	6.371	6.513	6.594	6.272	5.889	5.179	6.282
	Plane [ms]	2.016	2.758	1.299	1.836	1.689	1.646	1.682	3.354	**2.035**
	Total [ms]	8.495	9.718	7.671	8.349	8.283	7.918	7.571	8.533	8.317
	Fail Rate [%]	1.391	0.000	2.286	0.056	3.519	0.493	1.232	22.210	**3.898**
	MAE [deg]	8.717	11.020	9.507	5.843	9.846	5.208	5.428	20.629	**9.525**
	MedAE [deg]	6.855	10.966	8.933	5.250	6.386	5.003	4.763	18.287	**8.305**
	Var. [deg^2]	373.1	517.9	395.8	122.4	446.6	57.3	111.5	1871.9	**487.1**
RAN-LSF	**Prepr. [ms]**	6.235	7.640	6.577	7.322	6.918	6.878	6.259	5.157	6.623
	Plane [ms]	1.182	0.842	0.546	0.850	0.643	0.723	0.781	2.951	**1.065**
	Total [ms]	7.418	8.483	7.123	8.173	7.561	7.601	7.040	8.107	7.688
	Fail Rate [%]	7.391	0.000	4.091	1.289	9.528	0.740	1.087	33.044	**7.146**
	MAE [deg]	7.842	10.979	9.285	5.703	8.966	5.185	5.454	18.037	**8.931**
	MedAE [deg]	6.232	10.925	8.785	5.261	5.911	4.986	4.801	15.745	**7.831**
	Var.[deg^2]	302.1	513.9	375.7	114.4	367.0	55.8	111.9	1453.8	**411.8**

variance with 411.8 $(°)^2$ in comparison to RANSAC-PCA, which has a variance of 487.1 $(°)^2$. However, the error rate for RANSAC-LSF is with 7.1 % almost twice as high as the fail rate of RANSAC-PCA with 3.9 %. A possible explanation for this higher error rate is that RANSAC-LSF recognizes a wall earlier on average, which leads to faster errors in plane segmentation. This potentially also contributes to the lower MAE and the observed variance. Despite the faster runtime of RANSAC-LSF, RANSAC-PCA demonstrates better overall performance due to its robustness, with the slightly higher errors deemed insignificant.

5.4 On-Board Runtime Evaluation

Although the remote experiments offer valuable insights, especially with regard to accuracy and robustness, evaluating the on-board runtime remains crucial. Based on the remote results, we further assess the on-board runtime by running selected pre-processing approaches directly on the spherical mobile mapping system, attempting to replicate the same movement pattern manually for each approach to maintain comparability. Although this manual movement yields less consistent results regarding accuracy and fail rate due to slight variations in each run, our primary focus is on runtime. The goal is to determine which approaches operate in real-time on the spherical mobile mapping system. We select the fastest (geo-geo) and the slowest (none-geo) pre-processing approach in combination with each plane segmentation algorithm, which now includes RANSAC-LSF as well, to evaluate their real-time compability. As previously

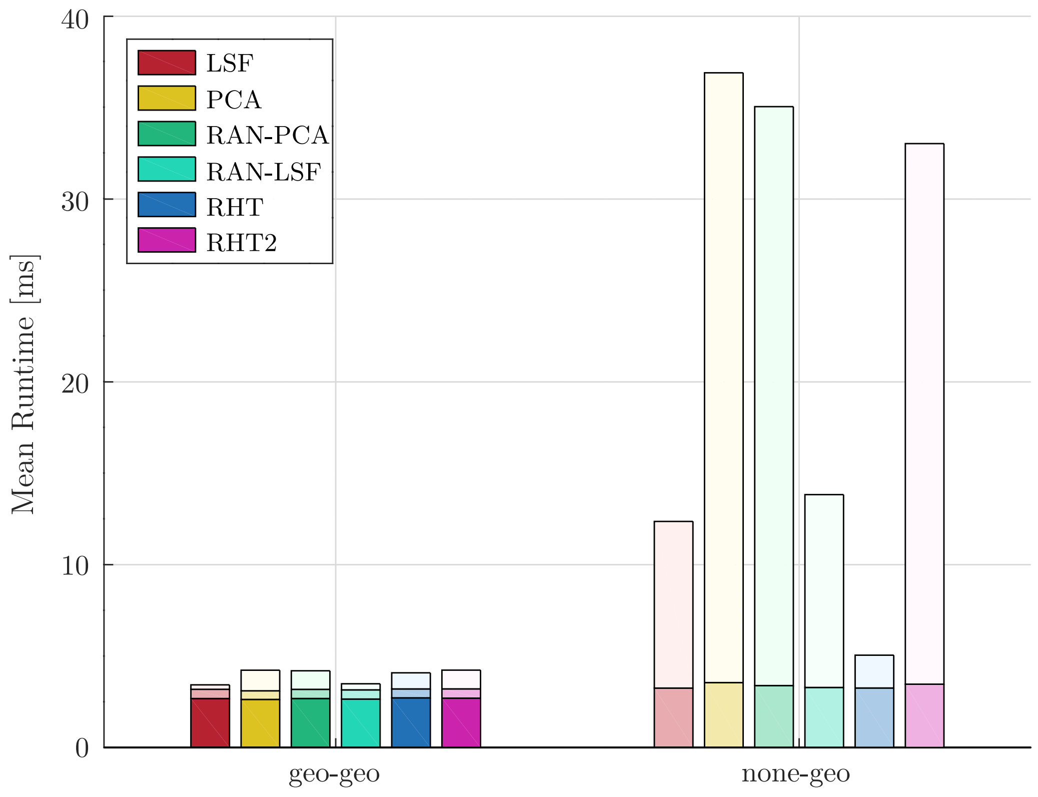


Figure 5.20: Final results of the on-board segmentation runtime analysis. Average runtime [ms] for different plane segmentation algorithms under the fastest and slowest pre-processing methods across all experiments. Intensities representing from dark to bright: filtering, sub-cloud creation and plane segmentation algorithm.

mentioned, we define real-time as a runtime faster than 50 ms, based on the frequency of the Hesai Pandar LiDAR of 20 Hz. For the on-board runtime evaluation, we conduct the tests on the on-board computer using an Intel Pentium Gold 5405U processor, compared to the remote experiments, where we ran the tests on an Intel Core i7-8565U. The latter offers a significant performance advantage, being approximately 1.57 times faster and featuring four additional CPU threads. Additionally, the on-board system operates with multiple other nodes running simultaneously, which further affects the runtime performance, unlike the remote experiments where such conditions were not present.

Table 5.22 lists the resulting runtimes and fail rates on-board the spherical mobile mapping system for each plane segmentation algorithm using the geo-geo and none-geo approach. Whereas Table 5.23 displays the MAE [deg], MedAE [deg], and variancee [deg^2] of each plane segmentation algorithm for the two selected pre-processing approaches. Figure 5.20 presents the final results of the on-board runtime analysis, showing the average runtime [ms] for the different plane segmentation algorithms under both the fastest and slowest pre-processing method. The figure uses intensity gradients, from dark to bright, to represent the stages of filtering, sub-cloud creation, and plane segmentation algorithm. From the results, we conduct that all approaches are able to operate in real-time on the robot. The slowest approach is PCA for the none-geo pre-processing with 36.9 ms runtime, followed by RANSAC-PCA with 35.1 ms and RHT2 with 33.0 ms. Notably, for none-geo, LSF is only 1.5 ms faster than RANSAC-LSF with a total runtime of 13.8 ms. Meanwhile, RHT remains the fastest for none-geo with a total runtime of 5.0 ms, following the trend established in the remote results (refer to Section 5.3.2). Overall, the none-geo approach requires for PCA, RANSAC-PCA and RHT2 roughly one-third longer compared to the remote results, whereas the RHT runs almost three times faster and LSF remains a similar runtime. For geo-geo, the fastest pre-processing approach, the total runtimes are generally much closer to each other. However, LSF and RANSAC-LSF stand out with total runtimes of approximately 3.4 ms, making them faster than the other methods, which all have total runtimes exceeding 4 ms. Most trends from the remote results remain consistent: LSF and RANSAC-LSF are the fastest, and RANSAC-PCA has a very similar runtime to RHT2. However, this time PCA requires slightly longer than both RANSAC-PCA and RHT2. Additionally, RHT runs slightly faster than RANSAC-PCA and RHT2. But in opposition to none-geo, all algorithms run faster than on the remote laptop for geo-geo. Based on these results, we assume that all sub-cloud approaches operate in real-time on the spherical mobile mapping system, since even the slowest approach, none-geo, meets the real-time criteria. Further, it is likely that the other approaches follow the trend of geo-geo and perform on-board even faster than for the remote experiments, due to the point limitation of the sub-cloud through downsampling or the usage of the kd-tree for the sub-cloud creation.

Table 5.22: Runtime and fail rate results of experiment ”onboard”. Runtimes [ms] of each step of the different pre-processing combinations for each plane segmentation algorithm, as well as the total runtime. The fail rate indicates the percentage of plane segmentation errors. RAN I indicates RANSAC in combination with PCA and RAN II indicates RANSAC in combination with LSF. Definition of real-time operation as a total runtime of less than 50 ms.

	Step	**LSF**	**PCA**	**RAN I**	**RAN II**	**RHT**	**RHT2**	**Avg.**
geo-geo	**Downsample [ms]**	2.678	2.629	2.680	2.650	2.715	2.699	**2.675**
	Cheese Search [ms]	0.404	0.379	0.401	0.403	0.392	0.407	**0.398**
	Delete [ms]	0.100	0.093	0.099	0.098	0.097	0.099	**0.098**
	Plane [ms]	0.243	1.127	1.015	0.333	0.881	1.026	**0.771**
	Total [ms]	3.426	4.227	4.196	3.485	4.084	4.232	**3.942**
	Fail Rate [%]	0.234	0.468	0.117	0.117	0.000	2.222	**0.526**
none-geo	**Cheese Search [ms]**	2.715	3.019	2.848	2.762	2.738	2.930	**2.835**
	Delete [ms]	0.535	0.531	0.539	0.522	0.516	0.532	**0.529**
	Plane [ms]	9.117	33.357	31.670	10.548	1.796	29.572	**19.343**
	Total [ms]	12.366	36.907	35.056	13.832	5.049	33.035	**22.708**
	Fail Rate [%]	0.851	13.050	3.404	3.404	0.426	6.809	**4.657**

Table 5.23: MAE [deg], MedAE [deg] and result variance [deg^2] of experiment "onboard" for the different plane segmentation algorithms under different pre-processing combinations. RAN I indicates RANSAC in combination with PCA and RAN II indicates RANSAC in combination with LSF.

	Result	LSF	PCA	RAN I	RAN II	RHT	RHT2
geo-geo	**MAE [deg]**	5.052	5.634	5.002	6.006	5.383	8.810
	MedAE [deg]	2.079	2.858	2.404	3.120	2.788	5.158
	Variance [deg^2]	146.93	176.27	140.02	197.44	156.85	395.65
none-geo	**MAE [deg]**	9.240	17.384	17.570	7.939	9.604	15.238
	MedAE [deg]	6.275	15.201	15.305	5.045	6.230	9.951
	Variance [deg^2]	414.38	1396.02	1432.63	312.95	447.53	1134.09

5.5 Final Evaluation and Discussion

In this final section, we first provide a comprehensive summary of the results from our experiments, including recommendations for suitable solutions across different scenarios. Here, we abbreviate the six pre-processing combinations as "filtering - sub-cloud" (e.g., none-geo refers to no filtering with the geometrical sub-cloud approach, and kdt-kdt indicates the use of kd-tree for both filtering and sub-cloud creation). Finally, we provide a detailed analysis of the effects of the key factors tested in the experiments on the robustness and accuracy.

5.5.1 Summary of Results and Recommendations

From the results of the pre-processing analysis, listed in Table 5.19, we observe that the none-geo method is the fastest with an average pre-processing runtime of 3.862 ms. This is due to its single point cloud traversal with a time complexity of $\mathcal{O}(n)$. The none-kdt method is the slowest with 13.627 ms, resulting from the kd-tree construction with all 30,000 points. All other methods involve downsampling, which alone has a complexity of $\mathcal{O}(n)$, but they require additional steps, which leads to the higher pre-processing runtime compared to none-geo. Although the none-geo method is the fastest regarding the pre-processing runtime, without downsampling and limiting the number of points in the sub-cloud for the plane segmentation, it is the slowest in terms of total runtime with 21.3 ms. This leads to the geo-geo combination emerging as the fastest overall with a total runtime of 7.959 ms and second-fastest regarding pre-processing runtime with 6.157 ms. This is because geo-geo and none-geo are the only pre-processing methods with a runtime complexity of $\mathcal{O}(n)$, while all other methods have a complexity of $\mathcal{O}(n \log n)$ due to the kd-tree construction. Additionally, even though geo-kdt is slightly faster than kdt-geo regarding pre-processing runtime, the higher total runtime of the geo-kdt approach indicates that the geometrical approach is superior to the kd-tree method for sub-cloud creation in terms of overall runtime.

In combination with the results from the plane segmentation analysis, presented in Figure 5.19 and Table 5.20, we conclude that none-geo and none-kdt are unsuitable for our application. The none-kdt method consistently shows the highest fail rates ($> 31.4\,\%$), as well as MAEs and MedAEs ($> 13.5\,°$), because of the smaller covered ground area due to the increased point density. Here, the impact on RHT2 is the smallest, with a difference of 4.0 ° between none-kdt and its best-case (lowest achieved MAE and MedAE) scenario, while RANSAC experiences the largest impact, with a 9.7 ° difference. And although the none-geo method achieves the lowest error rates (< 5.0 %), with the most significant impact on LSF, showing a runtime twenty-seven times higher, and the smallest effect on RHT, with only a 9 % increase compared to kd-tree sub-cloud approaches. Further, we see the second-highest errors (> 10.0 °) for all plane segmentation algorithms.

Furthermore, from Figure 5.19 we infer that the sub-cloud creation method primarily determines the runtime of the plane segmentation algorithms, not the filtering step. We find that the geometrical method results in lower runtimes for all plane detection algorithms (RANSAC, RHT, and RHT2), due to its better representation of the ground plane thus reducing the number of iterations. The largest increase occurs for RHT showing an increase of approximately 72 % in comparison to the kd-tree sub-cloud approaches. In contrast, the kd-tree approach allows for slightly faster plane fitting (LSF and PCA) by roughly 15 %, because of the restricted number of points. This benefit does not extend to plane detection algorithms, since fewer points lead to less comprehensive coverage (higher noise-to-ground ratio) and therefore longer detection times. Overall, the sub-cloud approach has a more significant impact on the runtimes of plane detection algorithms compared to plane fitting algorithms. In addition, we observe significantly lower fail rates for the geometrical sub-cloud creation compared to the kd-tree method, roughly half for most plane segmentation algorithms. The sub-cloud approach affects LSF and PCA even stronger, where the kd-tree approach has a higher fail rate of 33.1 % compared to 5.9 % for the geometrical approach, and further leads to increased MAEs and MedAEs by 10 % and 7 %, respectively. The higher noise-to-ground ratio in the kd-tree sub-cloud accounts for these differences, aligning with previous research [25, 42, 43, 45] that indicates plane fitting is more susceptible to noise. In summary, the geometric sub-cloud approach outperforms the kd-tree method due to the significantly higher robustness and the lower total runtime of the callback function.

Among the plane segmentation algorithms, LSF consistently performs the fastest with a maximum runtime of 0.415 ms, followed by PCA with a maximum runtime of 1.9 ms. We do not consider the non-filtering approaches. RANSAC and RHT2 are similarly fast, although they are about 30 % faster for the geometric sub-cloud approaches as in comparison to their runtime for the kd-tree sub-cloud approaches. The slowest algorithm is RHT, with runtimes of 2.5 ms for the geometrical sub-cloud approaches and 4.2 ms for the kd-tree approaches. This demonstrates that plane fitting algorithms are generally faster than plane detection algorithms, confirming the results of previous research [25, 45]. Regarding accuracy, we see higher MAEs and MedAEs (> 10.0 °) for the plane fitting algorithms compared to the plane detection algorithms for preprocessing approaches with filtering. The only exception is RHT2. It performs less accurate than LSF and PCA for geometrical sub-clouds but proves more accurate for the kd-tree sub-clouds. This supports previous research [25, 42, 43, 45] indicating that plane detection methods are better suited for noisy data. Furthermore, we find that LSF and PCA compete effectively with RANSAC and RHT only for the non-filtering approaches, which we have already identified as unsuitable. Consequently, we determine that the plane fitting algorithms are not part of a suitable solution for this application. Additionally, RHT2 is significantly less accurate than RANSAC by 49 % for the geometrical sub-cloud approaches and 25 % for kd-tree method, and also less accurate than RHT by 41 % and 28 %, respectively. For the geometrical sub-cloud approaches, RANSAC is the most accurate, with a MAE of 9.6 ° and a MedAE of 8.4 °, while RHT excels in the kd-tree sub-cloud approaches with very similar errors, MAE of 9.6 ° and MedAE of 8.1 °. The MedAE difference between RHT for the kd-tree sub-cloud and RANSAC for the geometrical sub-cloud is just 0.3 °, a difference that we define as statistically insignificant.

Given that LSF is significantly faster than PCA, we disproved the original assumption that PCA and LSF perform equally well for the pcl implementation. Therefore, we also implemented the RANSAC-LSF combination to evaluate its performance against the established RANSAC-PCA setup. The results show that RANSAC-LSF is approximately twice as fast as RANSAC-PCA and has slightly better accuracy, with a lower average MAE, MedAE, and variance. However, RANSAC-PCA is more robust, with a lower fail rate (3.9 % versus 7.1 %), making it the better overall performer despite RANSAC-LSF's speed advantage. We therefore only consider RANSAC-PCA for any further evaluation.

To evaluate real-time performance on the spherical mobile mapping system, we tested the fastest (geo-geo) and slowest (none-geo) pre-processing approaches with each plane segmentation algorithm, including RANSAC-LSF. From the on-board results we infer that all approaches meet the real-time criterion of 50 ms. Even for none-geo PCA and RANSAC-PCA, which are the slowest for that approach, operate with total runtimes of 36.9 ms and 35.1 ms respectively, while RHT remains the fastest with 5.0 ms. The geo-geo approach shows more consistent runtimes, with LSF and RANSAC-LSF achieving the fastest runtimes around 3.4 ms, while other methods exceed 4 ms.

Based on our experimental results, we offer the following recommendations. In addition to the plane fitting algorithms (LSF and PCA) and pre-processing approaches without filtering, we advise against using RHT2 for calculating the ground plane's normal vector in real time with the spherical mobile mapping system. This is based on RHT2's consistently lower accuracy and robustness compared to the other two plane detection algorithms. This leaves RANSAC and RHT as suitable solutions, which offer the highest accuracy. While RHT is most accurate for geo-kdt and kdt-kdt, RANSAC shows similar errors of for the geometrical sub-cloud approaches (geo-geo and kdt-geo). Regarding the robustness, we observe that RANSAC has in most experiments lower fail rates for the pre-processing approaches with filtering. With the geometrical sub-cloud approach being superior regarding robustness and runtime and geo-geo being faster than kdt-geo, we recommend the combination of RANSAC-PCA with geo-geo as pre-processing as the suitable solution for our application. This combination proves itself across all experiments performed as the top performer, with the only exception being experiment H (tunnel). However, experiment H is an extreme scenario, which highlights the biggest challenges for this implementation across all approaches. Those challenges are the non-flat ground within the tunnel and the very extreme limitation of the environment in one direction of the ground. Additionally, we find that while none-geo performs best for most plane segmentation algorithms in this experiment, the number of points within the sub-cloud becomes crucial in more challenging scenarios.

5.5.2 Impact Analysis of Key Factors

In this section, we analyze the impact of the key factors on the robustness and accuracy of the plane segmentation algorithms tested. The key factors are: limited visibility of the ground area within the scan, the speed of the spherical mobile mapping system, and varying ground inclinations. First, we examine how reduced ground visibility influences fail rates and accuracy. We then explore the effect of the robot's speed, assessing how changes in speed impact robustness and accuracy across different sub-cloud approaches. Finally, we analyze the role of ground inclinations to determine their effect on the performance of the algorithms. We focus exclusively on accuracy and robustness in this section, as speed is unaffected by the key factors, as stated in the offline experiments Sections 5.2.1 to 5.2.8.

Limitation of Ground Area We observe that the limitation of the ground area within the scan, which arises from the environment and not from the rotation of the LiDAR, significantly influences the robustness and accuracy of plane segmentation algorithms across all experiments. This effect is most pronounced when comparing the experiments with varying ground visibility: from the most limited in experiment C (wall), to intermediate cases in experiments A (hallway) and B (hall), and the least limited in experiments G and F (outside hill experiments). The inclinations of the ground in experiments G and F possibly contribute to the following observations.

From Figure 5.21 we observe a clear trend regarding robustness: higher limitation of the ground area correlates with increased fail rates. This trend is consistent across all pre-processing combinations. Notably, while reviewing the fail rates for these experiments, we notice that LSF, PCA, and RANSAC maintain relatively consistent fail rates between kdt-geo and geo-geo, as well as geo-kdt and kdt-kdt, indicating that the sub-cloud approach is the primary factor influencing robustness. In contrast, RHT and RHT2 exhibit more significant variations in fail rates between these pairs, highlighting their randomized approach to detect the dominant plane. When analyzing the impact on accuracy, we focus on the MedAE error, which is more robust to outliers and shows the same trend as the MAE. The results, depicted in Figure 5.22, clearly indicate that the choice of sub-cloud approach dictates the overall trend. Specifically, both none-geo and the combined geometrical approaches follow a similar trend, while none-kdt mirrors the pattern seen in the combined kd-tree approaches. Additionally, for geometrical sub-clouds, we see that the outdoor experiments yield more accurate results compared to the indoor experiments. We observe a similar pattern for the kd-tree sub-cloud approaches, although in experiment A (hallway), the accuracy is comparable to the outdoor accuracy. However, unlike for robustness, we do not observe a declining trend. Instead, the results for the geometrical sub-cloud approaches even show a slight increase in error for the indoor experiments. The slight increase in error for indoor experiments might be attributed to the lower fail rates seen in these conditions, as previously observed in the comparison between RANSAC-PCA and RANSAC-LSF, refer to Section 5.3.2.

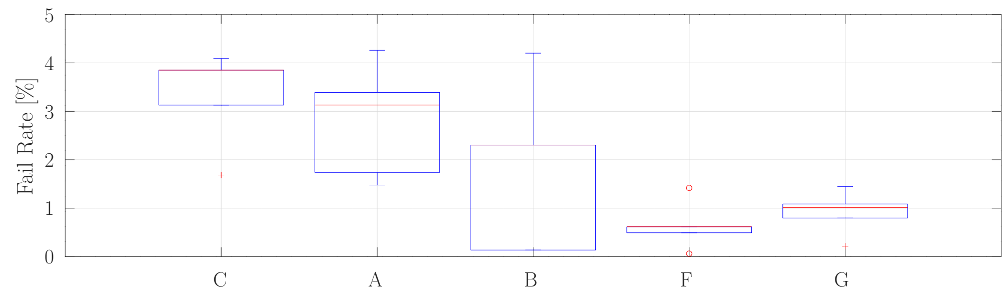
We tested the greatest limitation on the ground area's visibility in experiment H (tunnel), where the ground plane is effectively a rounded floor due to the tunnel's shape, and one axis is constrained by the tunnel's diameter of 99.5 cm. The lack of a flat ground plane is also why we exclude this experiment from the previous trend analysis. From the results, we observe very similar values for all approaches, with MAE ranging from 25.0 ° to 19.0 ° and MedAE varying between 22.8 ° and 16.0 °. Most pre-processing approaches lead to fail rates exceeding 25 %. These results confirm that the extreme limitation of the ground plane, as well as the non-flat ground, in this tunnel scenario results in significantly degraded performance in both robustness and accuracy. In summary, greater visibility of the ground area within the scan improves accuracy and robustness compared to scenarios with restricted visibility.

Speed In comparing experiments D (physics building ramp - slow motion) and E (physics building ramp - fast motion), which we conducted in the same location and thereby identical inclinations, we observe from Figure 5.23b that the slower speed leads to greater accuracy across all pre-processing approaches. Regarding robustness, we see that the speed significantly impacts the geometrical sub-cloud approaches, with higher speeds resulting in increased fail rates. This trend is also observed in the combined kd-tree sub-cloud approaches (geo-kdt and kdt-kdt). However, the none-kdt approach remains the worst in both cases, showing no sensitivity to changes in speed. This lack of sensitivity is due to the consistently poor accuracy of the none-kdt approach, as detailed in the previous Section 5.5.1 and Section 5.3.2.

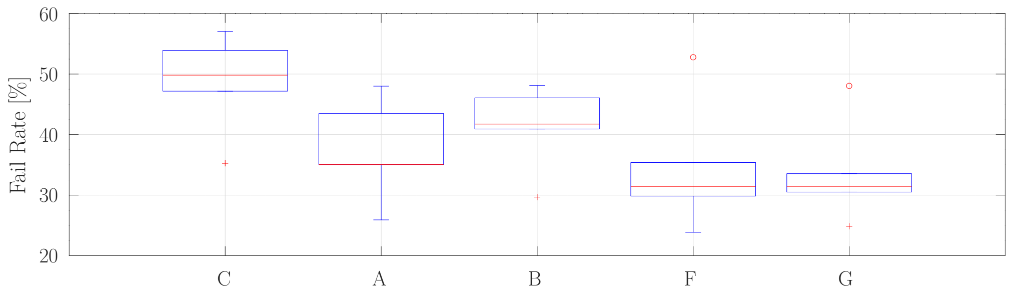
Varying Inclinations In this subsection, we investigate the effects of different inclinations on the accuracy of plane segmentation, excluding robustness as it is unaffected by inclinations. We analyze three experiments: D (physics building ramp with slow motion), F (outside hill moving up), and G (outside hill moving down). We only evaluate the RANSAC algorithm in combination with the geo-geo approach, as it has shown the best overall performance. Figure 5.24 displays the absolute angle error over the number of calls to the callback function, for the experiments D, F and G.

Experiment D introduces an additional constraint due to the ramp's walls, which restrict the ground area in a manner similar to a wall. Initially, the angle error is approximately 5 °, with fluctuations reaching up to 10 °. On the inclined ramp, errors decrease during the ascent but increase significantly during the descent, with errors reaching around 12 °. The final return to the flat ground plane shows a rise in errors compared to the first traversal. The increase in error during the descent is likely due to the restricted scanning area caused by the ramp's geometry. Specifically, we did not conduct the turn on the ramp in the middle, resulting in a more limited visibility of the ground area and higher errors. These higher errors affect the accuracy of subsequent geometrical point clouds because we calculate the geometrical sub-clouds based on previous normal vectors. Therefore, any inaccuracies from earlier scans propagate through the data, causing an offset.

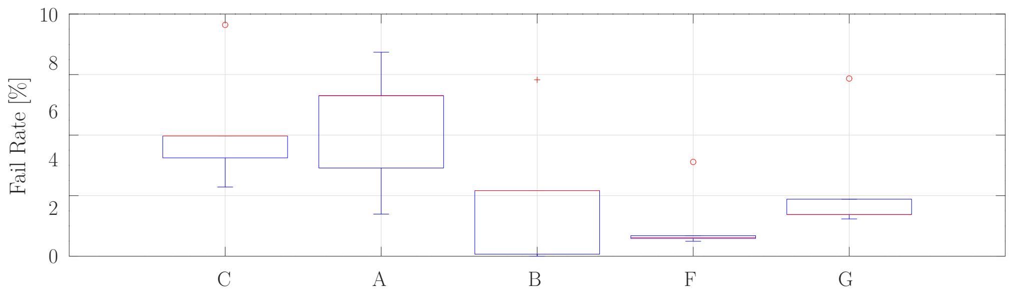
In experiment F, we observe the highest angle errors before the robot moves up the hill. Once on the inclined plane and at the top, the errors stabilize around 6 °. Notably, the consistent errors around the plane transitions result from stopping the robot to increase the plane counter, thereby maintaining accuracy. Overall, we do not observe a clear trend in the angular error. Furthermore, in this and the following experiment G, we apply a limited ground truth as we divided the hill into multiple flat planes with abrupt transitions, which is only an approximation. This strict subdivision possibly leads to minor errors at the transitions, so the actual error might be smaller than shown in the figure. Experiment G exhibits the greatest fluctuations in angle error, varying from approximately 7 ° to 11 ° during the descent, before stabilizing at around 7 ° after leaving the hill. These higher errors are presumably caused by the limited ground truth, which is particularly significant in this experiment due to the numerous plane changes with the largest differences in inclination. These appear to have a considerable influence on the observed errors. In summary, the RANSAC algorithm combined with the geo-geo pre-processing approach effectively handles varying inclinations and ground planes.



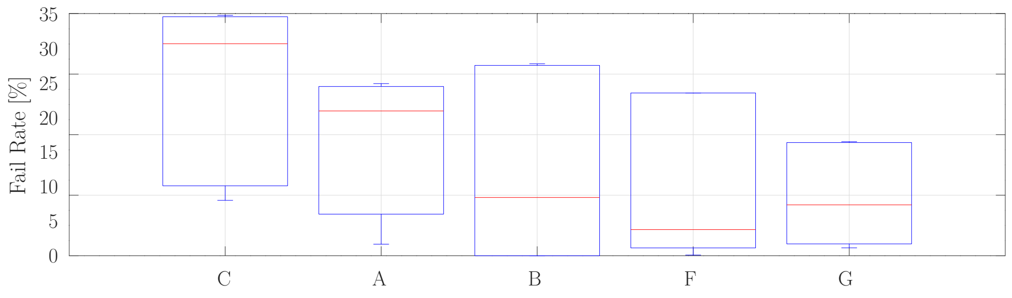
(a) Pre-processing: none-geo.



(b) Pre-processing: none-kdt.

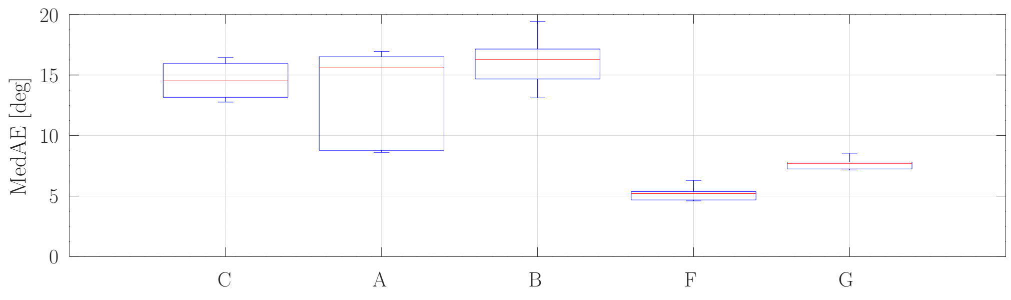


(c) Pre-processing: geometrical sub-cloud approaches combined (kdt-geo and geo-geo).

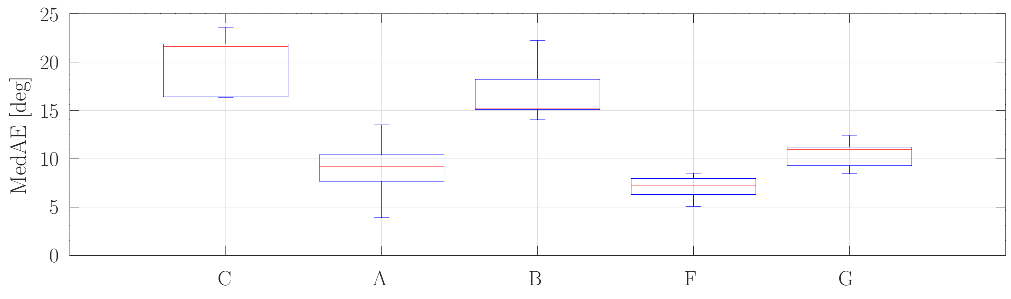


(d) Pre-processing: kd-tree sub-cloud approaches combined (geo-kdt and kdt-kdt).

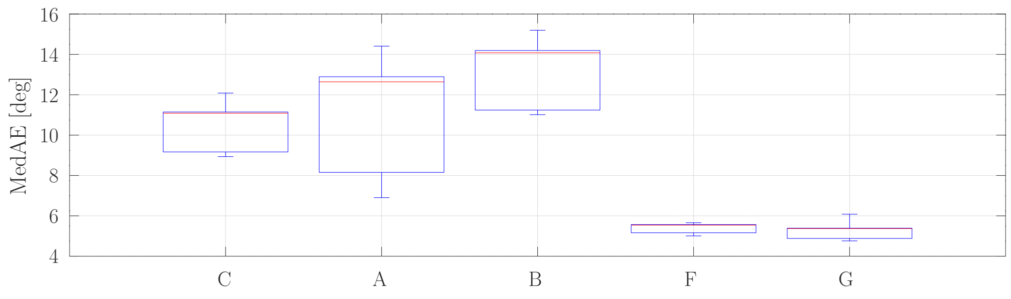
Figure 5.21: Final results of the key aspect ”limited area” analysis regarding robustness. Box plots of fail rates for five selected experiments (A, B, C, F, and G) across the different pre-processing approaches. Each subfigure presents the distribution of fail rates for a specific pre-processing approach, illustrating how limited ground visibility affects failure rates.



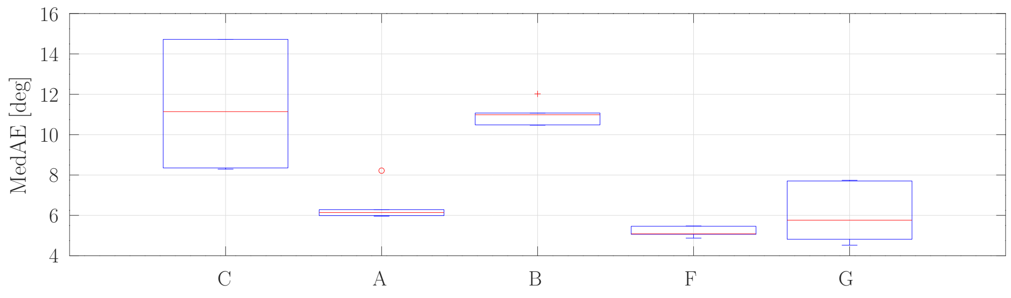
(a) Pre-processing: none-geo.



(b) Pre-processing: none-kdt.

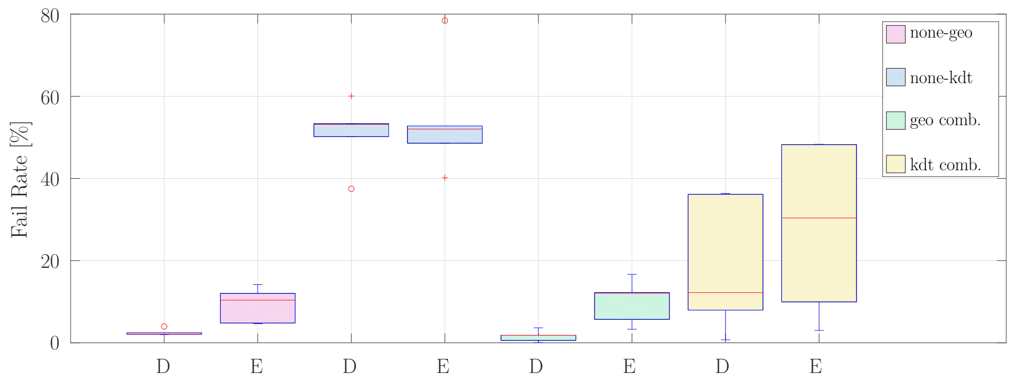


(c) Pre-processing: geometrical sub-cloud approaches combined (kdt-geo and geo-geo).

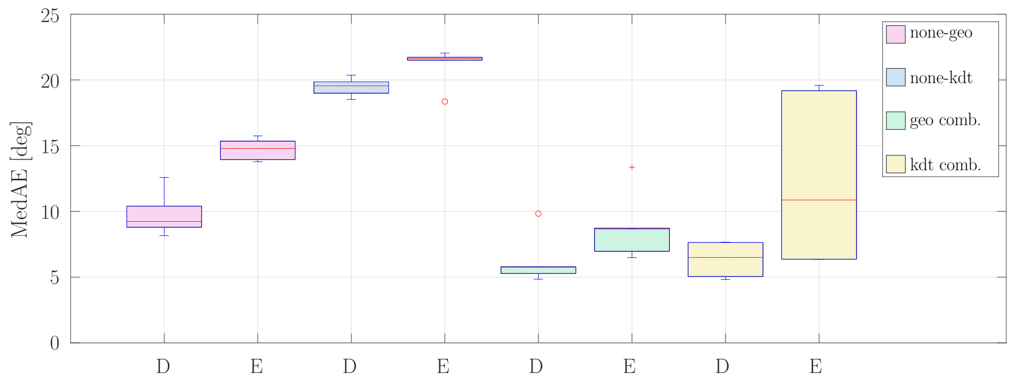


(d) Pre-processing: kd-tree sub-cloud approaches combined (geo-kdt and kdt-kdt).

Figure 5.22: Final results of the key aspect ”limited area” analysis regarding accuracy. Box plots of MedAE for five selected experiments (A, B, C, F, and G) across the different pre-processing approaches. Each subfigure presents the distribution of fail rates for a specific pre-processing approach, illustrating how limited ground visibility affects failure rates.

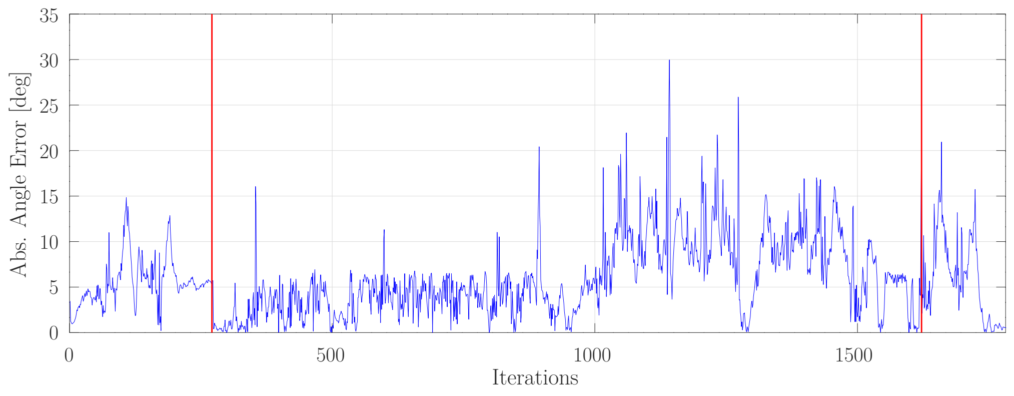


(a) Robustness box plots.

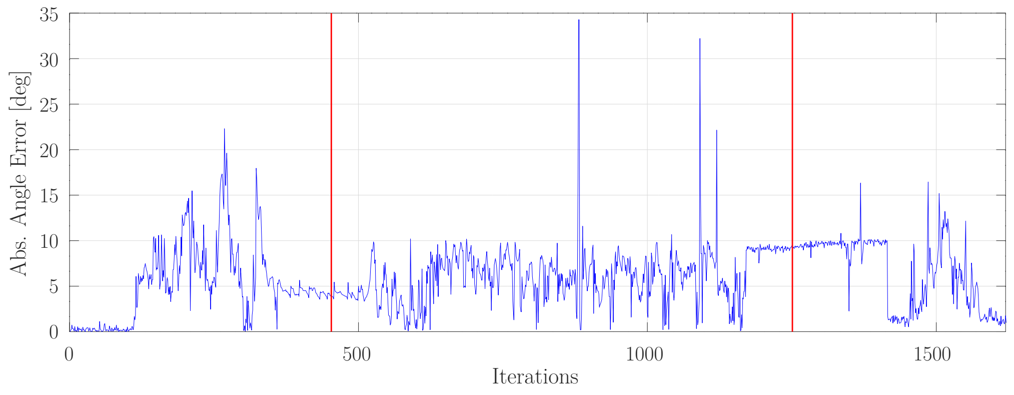


(b) Accuracy box plots.

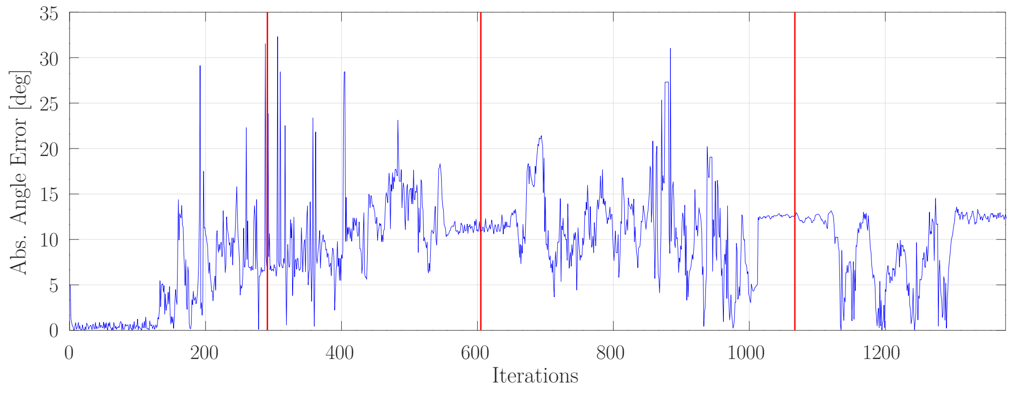
Figure 5.23: Final results of the key aspect "speed" analysis. Box plots of fail rates and MedAE for experiment E and D across the different pre-processing approaches, illustrating how the speed of the spherical mobile mapping system affects robustness and accuracy. Geometrical combined includes kdt-geo and geo-geo, and kd-tree combined inlcudes geo-kdt and kdt-kdt.



(a) Experiment D: physics building ramp with slow motion.



(b) Experiment F: outside hill moving up.



(c) Experiment G: outside hill moving down.

Figure 5.24: Final results of the key aspect "varying inclinations" analysis regarding accuracy. Plots of the absolute angle error over iterations, i.e. the number of calls to the callback function, for the experiments D, F and G. Red lines indicate a change of ground plane and thereby inclination.

Chapter 6

Conclusions

In this thesis, we implement and evaluate multiple approaches for the real-time calculation of the ground plane's normal vector from the LiDAR point clouds of a spherical mobile mapping system. Specifically, we compare a kd-tree-based method with a geometrical approach for pre-processing and evaluate four state-of-the-art plane segmentation algorithms using the pcl [23]. The code is available open-source on GitLab [1]. Our goal is to identify suitable solutions for the DAEDALUS [2] prototype to enhance pose estimation, since it relies on an accurate normal vector of the ground plane for its Kalman filter. Challenges due to the spherical shape of the robot include varying ground plane perceptions due to the robot's rotation, environmental noise from the robot's shell, and motion blur. The results aim to provide effective techniques for the DAEDALUS robot and serve as a benchmark for LiDAR-based segmentation in spherical and other mobile robots.

For the evaluation, we perform eight offline experiments across various scenarios to assess the performance of the 30 established approaches. We design each experiment to address a key aspect such as limited area, varying inclinations, and different movement speeds.Based on these results, we conduct the on-board experiments to evaluate the real-time capability. For that purpose, we run selected pre-processing approaches directly on the spherical mobile mapping system. We evaluate the performance based on runtime, accuracy, and robustness, measuring runtime as the total processing time with real-time capability defined as faster than 50 ms. We assess the accuracy by using MAE, MedAE, and result variance, while robustness is determined by the error rate, the ratio of plane segmentation errors to ROS callback function calls. For the evaluation, we abbreviate the pre-processing combinations as "filtering - sub-cloud" (e.g., none-kdt refers to no filtering with the kd-tree sub-cloud approach, and kdt-geo indicates the use of kd-tree for filtering and the geometrical sub-cloud creation).

Regarding the pre-processing approaches, we find that the none-geo method is the fastest with an average runtime of 3.862 ms, while the none-kdt method is the slowest at 13.627 ms. However, the none-geo method is the slowest overall with a total runtime of 21.3 ms. We conclude that the geo-geo combination, with a total runtime of 7.959 ms, is the fastest overall. This is because geo-geo and none-geo have $\mathcal{O}(n)$ complexity, whereas other methods have $\mathcal{O}(n \log n)$ complexity. Although geo-kdt is slightly faster in pre-processing than kdt-geo, it results in a higher total runtime, showing that the geometrical approach is superior for sub-cloud creation regarding runtime. In combination with accuracy and robustness, we determine the none-geo and none-kdt methods unsuitable. The none-kdt method has the highest fail rates ($> 31.4\,\%$), and the worst accuracy, with MAE and MedAE exceeding $13.5\,°$, due to the increased point density and smaller covered ground area. The none-geo method, while achieving the lowest error rates at under $5.0\,\%$, results in the second-highest errors ($> 10.0\,°$) for all plane segmentation algorithms. It also shows an increase in runtime of up to twenty-seven times higher. Furthermore, we conclude that the geometrical sub-cloud approach significantly outperforms the kd-tree method. It results in lower runtimes for plane detection algorithms due to better ground plane representation, which reduces the number of iterations needed. The largest increase occurs for RHT, with an increase of approximately 72 % of the runtime in comparison to its runtime for the kd-tree sub-cloud approaches. In contrast, the kd-tree method allows for slightly faster plane fitting by roughly 15 %, but leads to longer detection times due to higher noise-to-ground ratios. In addition, we observe significantly lower fail rates for the geometrical sub-cloud creation compared to the kd-tree method, roughly half for most plane segmentation algorithms.

Among the plane segmentation algorithms, LSF is the fastest at 0.415 ms, followed by PCA at 1.9 ms, with non-filtering approaches excluded from consideration. RANSAC and RHT2 are similarly fast, and they are about 30 % faster for the geometric sub-cloud approaches as in comparison to their runtime for the kd-tree sub-cloud approaches. The slowest algorithm is RHT, with runtimes of 2.5 ms for geometrical and 4.2 ms for kd-tree sub-clouds. With these results, we determine that plane fitting algorithms are generally faster than plane detection algorithms.Regarding accuracy, plane fitting algorithms generally show higher MAE and MedAE ($> 10.0\,°$) compared to plane detection algorithms. The exception is RHT2, which, while less accurate than LSF and PCA with geometrical sub-clouds, performs better with kd-tree sub-clouds, confirming that plane detection methods handle noisy data more effectively. LSF and PCA show competitive performance only in the non-filtering approaches, which we determine as unsuitable. Therefore, we conclude that plane fitting algorithms are not suitable for this application. RHT2 is significantly less accurate than RANSAC and RHT by 45 % geometrical and 27 % for kd-tree sub-clouds. Therefore, we advise against using RHT2. RANSAC is the most accurate for geometrical sub-clouds, and RHT excels for kd-tree sub-clouds, with a statistically insignificant difference in MedAE of $0.3\,°$. With the previous results, we disproved the initial assumption that PCA and LSF perform equally wellfor the pcl implementation, as LSF is significantly faster. In the following comparison between RANSAC-LSF and RANSAC-PCA, we find RANSAC-LSF to be about twice as fast and slightly more accurate. Despite this, RANSAC-PCA is more robust with an almost 50 % lower fail rate, making it the better overall performer. We thus recommend RANSAC-PCA.

Based on our findings, we recommend RANSAC-PCA with the geo-geo approach as the suitable solution, offering the best balance of accuracy and robustness. Although RHT is slightly more accurate for kd-tree sub-clouds, RANSAC generally shows lower fail rates. Although RANSAC-geo-geo is the top performer in most cases, it faces challenges in extreme scenarios like tunnels, where none-geo performs better due to its higher number of points. Regarding the real-time performance on the spherical mobile mapping system, we infer that all approaches meet the real-time criterion of 50 ms. limitation of the ground area within the scan When considering the impact of key aspects on the performance of these approaches, we observe that the limitation of ground visibility due to the environment significantly influences both accuracy and robustness, with greater visibility leading to better outcomes. Speed also plays a crucial role, with slower speeds enhancing accuracy, particularly for the geometrical sub-cloud approaches. In scenarios with varying inclinations, RANSAC with the geo-geo approach effectively handles the changes in inclination. However, we do not observe a clear trend in the angular error across different inclinations, suggesting that error fluctuations and offsets occur due to other conditions.

Needless to say, a lot of work remains to be done. First, future work needs to adapt the Kalman filter to utilize the on-board calculated normal vector n instead of the fixed vector $\boldsymbol{v}_{down} = [0,\, 0,\, -1]^T$ and test the new pose estimation capabilities. Additionally, future work should include the development of an online certainty measure to assess the reliability of the normal vector calculation, as well as the pose estimation in real-time. Moreover, several areas are promising for enhancing the robustness and accuracy of the current implementation. The tunnel experiment suggests that for most approaches increasing the number of points within the sub-cloud improves performance. Therefore, implementing a sliding window approach to combine multiple scans possibly enhances accuracy and robustness, especially in complex environments. Instead of combining subsequent scans, combining the current scan with key frames, selective scans that together build a comprehensive representation of the environment, may offer even better results. Another approach is to improve the verification process for detecting the ground plane to avoid lasting error offsets, that emerge from the geometrical sub-cloud approach since it relies on the previous normal vector. Further, including an alternative model than a flat plane for RANSAC might enhance accuracy in the tunnel scenario, thereby advancing the ultimate goal of exploring lunar caves. Also exploring how to optimize the balance between real-time processing and accuracy, particularly in scenarios where speed and environmental complexity are significant factors, presents an important direction. Moreover, with the integration of the locomotion mechanism into the prototype, further testing and adaptions to the filtering become necessary, as the hands of the person moving the sphere no longer need to be filtered, but potential interferences from support rods possibly arise. Finally, future work might refine the accuracy assessment for the spherical mobile mapping system by using an ideal flat (inclined) plane with no ground visibility restrictions (infinite), or by remodeling the outside hill ground truth with smooth angles, as well as determine the accuracy for non-spherical mobile robots.

Bibliography

[1] Carolin Bösch. MA-Thesis GitLab. `https://gitlab2.informatik.uni-wuerzburg.de/s391207/ma-thesis`, 2024. Accessed: 06.08.2024.

[2] Angelo Pio Rossi, Francesco Maurelli, Vikram Unnithan, Hendrik Dreger, Kedus Mathewos, Nayan Pradhan, Dan-Andrei Corbeanu, Riccardo Pozzobon, Matteo Massironi, Sabrina Ferrari, et al. *DAEDALUS-Descent And Exploration in Deep Autonomy of Lava Underground Structures: Open Space Innovation Platform (OSIP) Lunar Caves-System Study*. 2021.

[3] ESA. ESA plans mission to explore lunar caves. `https://www.esa.int/Enabling_Support/Preparing_for_the_Future/Discovery_and_Preparation/ESA_plans_mission_to_explore_lunar_caves`, 2021. Accessed: 22.08.2024.

[4] Rudolph Emil Kalman. A new approach to linear filtering and prediction problems. 1960.

[5] Jan Elseberg, S. Magnenat, Roland Siegwart, and A Nuechter. Comparison on nearest-neigbour-search strategies and implementations for efficient shape registration. *Journal of Software Engineering for Robotics (JOSER)*, 3:2–12, 01 2012.

[6] Jon Louis Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517, 1975.

[7] Donald Meagher. Geometric modeling using octree encoding. *Computer graphics and image processing*, 19(2):129–147, 1982.

[8] Antonin Guttman. R-trees: A dynamic index structure for spatial searching. In *Proceedings of the 1984 ACM SIGMOD international conference on Management of data*, pages 47–57, 1984.

[9] George S Lueker. A data structure for orthogonal range queries. In *19th Annual Symposium on Foundations of Computer Science (sfcs 1978)*, pages 28–34. IEEE, 1978.

[10] Peter N Yianilos. Data structures and algorithms for nearest neighbor search in general metric spaces. In *Soda*, volume 93, pages 311–21, 1993.

[11] Yasuaki Nakamura, Shigeru Abe, Yutaka Ohsawa, and Masao Sakauchi. A balanced hierarchical data structure for multidimensional data with highly efficient dynamic characteristics. *IEEE Transactions on Knowledge and Data Engineering*, 5(4):682–694, 1993.

[12] Andreas Nüchter. *3D robotic mapping: the simultaneous localization and mapping problem with six degrees of freedom*, volume 52. Springer, 2008.

[13] Jordi L Vermeulen, Arne Hillebrand, and Roland Geraerts. A comparative study of k-nearest neighbour techniques in crowd simulation. *Computer Animation and Virtual Worlds*, 28(3-4):e1775, 2017.

[14] Yixi Cai, Wei Xu, and Fu Zhang. ikd-tree: An incremental kd tree for robotic applications. *arXiv preprint arXiv:2102.10808*, 2021.

[15] Wei Xu, Yixi Cai, Dongjiao He, Jiarong Lin, and Fu Zhang. Fast-lio2: Fast direct lidar-inertial odometry. *IEEE Transactions on Robotics*, 38(4):2053–2073, 2022.

[16] Automation Group (Jacobs University Bremen) and Knowledge-Based Systems Group (University of Osnabrück). 3dtk – the 3d toolkit. `http://slam6d.sourceforge.net/`, 2011. Accessed: 10.04.2024.

[17] David M. Mount and Sunil Arya. Ann: A library for approximate nearest neighbor searching. `http://www.cs.umd.edu/~mount/ANN/`, 2010. Accessed: 10.04.2024.

[18] Marius Muja and David G. Lowe. Scalable nearest neighbor algorithms for high dimensional data. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 36(11):2227–2240, 2014.

[19] S. Magnenat. libnabo. `https://github.com/norlab-ulaval/libnabo`, 2011. Accessed: 10.04.2024.

[20] Jose Luis Blanco and Pranjal Kumar Rai. nanoflann: a C++ header-only fork of FLANN, a library for nearest neighbor (NN) with kd-trees. `https://github.com/jlblancoc/nanoflann`, 2014. Accessed: 27.02.2024.

[21] Michael Greenspan, Guy Godin, and Jimmy Talbot. Acceleration of binning nearest neighbor methods. *Proceedings of Vision Interface 2000*, pages 337–344, 2000.

[22] Barend Gehrels, Bruno Lalande, Mateusz Loskot, Adam Wulkiewicz, Menelaos Karavelas, and Vissarion Fisikopoulos. Boost geometry library. `https://www.boost.org/doc/libs/1_65_1/libs/geometry/doc/html/index.html`, 2017. Accessed: 10.04.2024.

[23] Radu Bogdan Rusu and Steve Cousins. 3d is here: Point cloud library (pcl). In *2011 IEEE International Conference on Robotics and Automation*, pages 1–4, 2011.

[24] Marcus Adamsson and Aleksandar Vorkapic. A comparison study of kd-tree, vp-tree and octree for storing neuronal morphology data with respect to performance, 2016.

[25] 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. IEEE, 2013.

[26] Bir Bhanu, Sungkee Lee, Chih-Cheng Ho, and Tom Henderson. Range data processing: Representation of surfaces by edges. In *Proceedings of the eighth international conference on pattern recognition*, volume 236, page 238. IEEE Computer Society Press Piscataway, NJ, USA, 1986.

[27] Xiaoyi Y Jiang, Urs Meier, and Horst Bunke. Fast range image segmentation using high-level segmentation primitives. In *Proceedings Third IEEE Workshop on Applications of Computer Vision. WACV'96*, pages 83–88. IEEE, 1996.

[28] Angel Domingo Sappa and Michel 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. IEEE, 2001.

[29] Paul J. Besl and Ramesh C Jain. Segmentation through variable-order surface fitting. *IEEE Transactions on pattern analysis and machine intelligence*, 10(2):167–192, 1988.

[30] Klaus Köster and Michael Spann. Mir: An approach to robust clustering-application to range image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(5):430–444, 2000.

[31] Radu Bogdan Rusu, Zoltan Csaba Marton, Nico Blodow, Mihai Dolha, and Michael Beetz. Towards 3d point cloud based object maps for household environments. *Robotics and Autonomous Systems*, 56(11):927–941, 2008.

[32] Peter Dorninger and Clemens Nothegger. 3d segmentation of unstructured point clouds for building modelling. *International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, 35(3/W49A):191–196, 2007.

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

[34] Josep Miquel Biosca and José Luis Lerma. Unsupervised robust planar segmentation of terrestrial laser scanner point clouds based on fuzzy clustering methods. *ISPRS Journal of Photogrammetry and Remote Sensing*, 63(1):84–98, 2008.

[35] 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):71–80, 2006.

[36] Ruwen Schnabel, Roland Wahl, and Reinhard Klein. Efficient ransac for point-cloud shape detection. In *Computer graphics forum*, volume 26, pages 214–226. Wiley Online Library, 2007.

[37] Natasha Gelfand and Leonidas J Guibas. Shape segmentation using local slippage analysis. In *Proceedings of the 2004 Eurographics/ACM SIGGRAPH symposium on Geometry processing*, pages 214–223, 2004.

[38] Aleksey Golovinskiy and Thomas Funkhouser. Min-cut based segmentation of point clouds. In *2009 IEEE 12th International Conference on Computer Vision Workshops, ICCV Workshops*, pages 39–46. IEEE, 2009.

[39] 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. IEEE, 2010.

[40] Radu Bogdan Rusu, Andreas Holzbach, Nico Blodow, and Michael Beetz. Fast geometric point labeling using conditional random fields. In *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 7–12. IEEE, 2009.

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

[42] Chien Ming Huang and Yi-Hsing Tseng. Plane fitting methods of lidar point cloud. In *29th Asian Conference on Remote Sensing 2008, ACRS 2008*, pages 1925–1930, 2008.

[43] Abdul Nurunnabi, David Belton, and Geoff West. Robust statistical approaches for local planar surface fitting in 3d laser scanning data. *ISPRS Journal of Photogrammetry and Remote Sensing*, 96:106–122, 2014.

[44] Dorit Borrmann, Jan Elseberg, Kai Lingemann, and Andreas Nüchter. The 3d hough transform for plane detection in point clouds: A review and a new accumulator design. *3D Research*, 2(2):1–13, 2011.

[45] Abdul Nurunnabi, David Belton, and Geoff West. Diagnostics based principal component analysis for robust plane fitting in laser data. In *16th Int'l Conf. Computer and Information Technology*, pages 484–489. IEEE, 2014.

[46] Richard Hoffman and Anil K Jain. Segmentation and classification of range images. *IEEE transactions on pattern analysis and machine intelligence*, pages 608–620, 1987.

[47] Karl Pearson. Liii. on lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 2(11):559–572, 1901.

[48] Mia Hubert, Peter J Rousseeuw, and Karlien Vanden Branden. Robpca: a new approach to robust principal component analysis. *Technometrics*, 47(1):64–79, 2005.

[49] Jiashi Feng, Huan Xu, and Shuicheng Yan. Robust pca in high-dimension: A deterministic approach. *arXiv preprint arXiv:1206.4628*, 2012.

[50] Praneeth Netrapalli, Niranjan UN, Sujay Sanghavi, Animashree Anandkumar, and Prateek Jain. Non-convex robust pca. *Advances in neural information processing systems*, 27, 2014.

[51] Emmanuel J Candès, Xiaodong Li, Yi Ma, and John Wright. Robust principal component analysis? *Journal of the ACM (JACM)*, 58(3):1–37, 2011.

[52] Venkat Chandrasekaran, Sujay Sanghavi, Pablo A Parrilo, and Alan S Willsky. Rank-sparsity incoherence for matrix decomposition. *SIAM Journal on Optimization*, 21(2):572–596, 2011.

[53] Prateek Jain, Praneeth Netrapalli, and Sujay Sanghavi. Low-rank matrix completion using alternating minimization. In *Proceedings of the forty-fifth annual ACM symposium on Theory of computing*, pages 665–674, 2013.

[54] Yuxin Chen and Emmanuel Candes. Solving random quadratic systems of equations is nearly as easy as solving linear systems. *Advances in Neural Information Processing Systems*, 28, 2015.

[55] Chenlu Qiu and Namrata Vaswani. Real-time robust principal components' pursuit. In *2010 48th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, pages 591–598. IEEE, 2010.

[56] Xinyang Yi, Dohyung Park, Yudong Chen, and Constantine Caramanis. Fast algorithms for robust pca via gradient descent. *Advances in neural information processing systems*, 29, 2016.

[57] Huan Xu, Constantine Caramanis, and Sujay Sanghavi. Robust pca via outlier pursuit. *Advances in neural information processing systems*, 23, 2010.

[58] Namrata Vaswani and Praneeth Narayanamurthy. Static and dynamic robust pca and matrix completion: A review. *Proceedings of the IEEE*, 106(8):1359–1379, 2018.

[59] Klaas Klasing, Daniel Althoff, Dirk Wollherr, and Martin Buss. Comparison of surface normal estimation methods for range sensing applications. In *2009 IEEE international conference on robotics and automation*, pages 3206–3211. Ieee, 2009.

[60] Wei Song, Lingfeng Zhang, Yifei Tian, Simon Fong, and Su Sun. 3d hough transform algorithm for ground surface extraction from lidar point clouds. In *2019 International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*, pages 916–921. IEEE, 2019.

[61] Paul VC Hough. Method and means for recognizing complex patterns, December 18 1962. US Patent 3,069,654.

[62] George Vosselman. Building reconstruction using planar faces in very high density height data. *International Archives of Photogrammetry and Remote Sensing*, 32(3; SECT 2W5):87–94, 1999.

[63] Robert C Bolles and Martin A Fischler. A ransac-based approach to model fitting and its application to finding cylinders in range data. In *IJCAI*, volume 1981, pages 637–643, 1981.

[64] David G Lowe. Object recognition from local scale-invariant features. In *Proceedings of the seventh IEEE international conference on computer vision*, volume 2, pages 1150–1157. Ieee, 1999.

[65] Lucas Jacobs, John Weiss, and Dan Dolan. Object tracking in noisy radar data: comparison of hough transform and ransac. In *IEEE International Conference on Electro-Information Technology, EIT 2013*, pages 1–6. IEEE, 2013.

[66] Hassan Facoiti, Ahmed Boumezzough, and Said Safi. Computer vision radar for autonomous driving using histogram method. *Advances in Science, Technology and Engineering Systems Journal*, 7(4):42–48, 2022.

[67] Rostislav Hulik, Michal Spanel, Pavel Smrz, and Zdenek Materna. Continuous plane detection in point-cloud data based on 3d hough transform. *Journal of visual communication and image representation*, 25(1):86–97, 2014.

[68] Fayez Tarsha-Kurdi, Tania Landes, and Pierre Grussenmeyer. Hough-transform and extended ransac algorithms for automatic detection of 3d building roof planes from lidar data. In *ISPRS Workshop on Laser Scanning 2007 and SilviLaser 2007*, volume 36, pages 407–412, 2007.

[69] Evangelos Maltezos and Charalabos Ioannidis. Automatic extraction of building roof planes from airborne lidar data applying an extended 3d randomized hough transform. *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, 3:209–216, 2016.

[70] Tiago Gomes, Diogo Matias, André Campos, Luís Cunha, and Ricardo Roriz. A survey on ground segmentation methods for automotive lidar sensors. *Sensors*, 23(2):601, 2023.

[71] Bertrand Douillard, J Underwood, Narek Melkumyan, S Singh, Shrihari Vasudevan, C Brunner, and A Quadros. Hybrid elevation maps: 3d surface models for segmentation. In *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1532–1538. IEEE, 2010.

[72] Alireza Asvadi, Paulo Peixoto, and Urbano Nunes. Detection and tracking of moving objects using 2.5 d motion grids. In *2015 IEEE 18th International Conference on Intelligent Transportation Systems*, pages 788–793. IEEE, 2015.

[73] Qingquan Li, Liang Zhang, Qingzhou Mao, Qin Zou, Pin Zhang, Shaojun Feng, and Washington Ochieng. Motion field estimation for a dynamic scene using a 3d lidar. *Sensors*, 14(9):16672–16691, 2014.

[74] Michael Himmelsbach, Thorsten Luettel, and H-J Wuensche. Real-time object classification in 3d point clouds using point feature histograms. In *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 994–1000. IEEE, 2009.

[75] Zhongzhen Luo, Martin V Mohrenschildt, and Saeid Habibi. A probability occupancy grid based approach for real-time lidar ground segmentation. *IEEE Transactions on Intelligent Transportation Systems*, 21(3):998–1010, 2019.

[76] Xiao Hu, F Sergio A Rodriguez, and Alexander Gepperth. A multi-modal system for road detection and segmentation. In *2014 IEEE Intelligent Vehicles Symposium Proceedings*, pages 1365–1370. IEEE, 2014.

[77] Hyungtae Lim, Minho Oh, and Hyun Myung. Patchwork: Concentric zone-based region-wise ground segmentation with ground likelihood estimation using a 3d lidar sensor. *IEEE Robotics and Automation Letters*, 6(4):6458–6465, 2021.

[78] Yihuan Zhang, Jun Wang, Xiaonian Wang, and John M Dolan. Road-segmentation-based curb detection method for self-driving via a 3d-lidar sensor. *IEEE transactions on intelligent transportation systems*, 19(12):3981–3991, 2018.

[79] Michael Himmelsbach, Felix V Hundelshausen, and H-J Wuensche. Fast segmentation of 3d point clouds for ground vehicles. In *2010 IEEE Intelligent Vehicles Symposium*, pages 560–565. IEEE, 2010.

[80] Shrihari Vasudevan, Fabio Ramos, Eric Nettleton, and Hugh Durrant-Whyte. Gaussian process modeling of large-scale terrain. *Journal of Field Robotics*, 26(10):812–840, 2009.

[81] Bertrand Douillard, James Underwood, Noah Kuntz, Vsevolod Vlaskine, Alastair Quadros, Peter Morton, and Alon Frenkel. On the segmentation of 3d lidar point clouds. In *2011 IEEE International Conference on Robotics and Automation*, pages 2798–2805. IEEE, 2011.

[82] Phuong Chu, Seoungjae Cho, Sungdae Sim, Kiho Kwak, and Kyungeun Cho. A fast ground segmentation method for 3d point cloud. *Journal of information processing systems*, 13(3):491–499, 2017.

[83] Jens Rieken, Richard Matthaei, and Markus Maurer. Benefits of using explicit ground-plane information for grid-based urban environment modeling. In *2015 18th International Conference on Information Fusion (Fusion)*, pages 2049–2056. IEEE, 2015.

[84] Frank Moosmann, Oliver Pink, and Christoph Stiller. Segmentation of 3d lidar data in non-flat urban environments using a local convexity criterion. In *2009 IEEE Intelligent Vehicles Symposium*, pages 215–220. IEEE, 2009.

[85] Kiin Na, Jaemin Byun, Myongchan Roh, and Bumsu Seo. The ground segmentation of 3d lidar point cloud with the optimized region merging. In *2013 International Conference on Connected Vehicles and Expo (ICCVE)*, pages 445–450. IEEE, 2013.

[86] Bisheng Yang and Zhen Dong. A shape-based segmentation method for mobile laser scanning point clouds. *ISPRS journal of photogrammetry and remote sensing*, 81:19–30, 2013.

[87] Igor Bogoslavskyi and Cyrill Stachniss. Fast range image-based segmentation of sparse 3d laser scans for online operation. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 163–169. IEEE, 2016.

[88] Frederik Hasecke, Lukas Hahn, and Anton Kummert. Flic: Fast lidar image clustering. *arXiv preprint arXiv:2003.00575*, 2020.

[89] Mingfang Zhang, Daniel D Morris, and Rui Fu. Ground segmentation based on loopy belief propagation for sparse 3d point clouds. In *2015 International Conference on 3D Vision*, pages 615–622. IEEE, 2015.

[90] Weixin Huang, Huawei Liang, Linglong Lin, Zhiling Wang, Shaobo Wang, Biao Yu, and Runxin Niu. A fast point cloud ground segmentation approach based on coarse-to-fine markov random field. *IEEE Transactions on Intelligent Transportation Systems*, 23(7):7841–7854, 2021.

[91] Lukas Rummelhard, Anshul Paigwar, Amaury Nègre, and Christian Laugier. Ground estimation and point cloud segmentation using spatiotemporal conditional random field. In *2017 IEEE Intelligent Vehicles Symposium (IV)*, pages 1105–1110. IEEE, 2017.

[92] Sukai Wang, Huaiyang Huang, and Ming Liu. Simultaneous clustering classification and tracking on point clouds using bayesian filter. In *2017 IEEE International Conference on Robotics and Biomimetics (ROBIO)*, pages 2521–2526. IEEE, 2017.

[93] Dean A Pomerleau and David S Touretzky. Advances in neural information processing systems 1. *ch. ALVINN: An Autonomous Land Vehicle in a Neural Network*, pages 305–313, 1989.

[94] R Qi Charles, Hao Su, Mo Kaichun, and Leonidas J Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *2017 IEEE conference on computer vision and pattern recognition (CVPR)*, pages 77–85. IEEE, 2017.

[95] Charles Ruizhongtai Qi, Li Yi, Hao Su, and Leonidas J Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. *Advances in neural information processing systems*, 30, 2017.

[96] Yecheng Lyu, Lin Bai, and Xinming Huang. Real-time road segmentation using lidar data processing on an fpga. In *2018 IEEE international symposium on circuits and systems (ISCAS)*, pages 1–5. IEEE, 2018.

[97] Martin Velas, Michal Spanel, Michal Hradis, and Adam Herout. Cnn for very fast ground segmentation in velodyne lidar data. In *2018 IEEE International Conference on Autonomous Robot Systems and Competitions (ICARSC)*, pages 97–103. IEEE, 2018.

[98] RIEGL Laser Measurement Systems GmbH. RIEGL VZ-400. `http://www.riegl.com/uploads/tx_pxpriegldownloads/10_DataSheet_VZ-400_2017-06-14.pdf`, 2017. Accessed: 23.07.2024.

[99] Michael Bleier. Lecture notes in Advanced Sensory Systems and Sensor Data Processing, University Würzburg. `https://wuecampus.uni-wuerzburg.de/moodle/course/view.php?id=59606#section-2`, June 2023. Accessed: 31.08.2024.

[100] Johannes Schauer and Andreas Nüchter. Collision detection between point clouds using an efficient k-d tree implementation. *Advanced Engineering Informatics*, 29(3):440–458, 2015.

[101] Igal Galperin and Ronald L Rivest. Scapegoat trees. In *Proceedings of the fourth annual ACM-SIAM Symposium on Discrete algorithms*, pages 165–174, 1993.

[102] Adrien-Marie Legendre. Nouvelles méthodes pour la détermination des orbites des comètes. appendice sur la méthodes des moindres carrés. *Didot, Paris*, 80, 1805.

[103] C.F. Gauss and C. Haase. *Theorie der Bewegung der Himmelskörper welche in Kegelschnitten die Sonne umlaufen, übertr. von C. Haase.* Oxford University, 1865. Original: Theoria Motus Corporum Coelestium in sectionibus conicis solem ambientium 1809.

[104] David J Ittner. 3-d surface discrimination from local curvature measures. In *Proceedings of Computer Vision and Pattern Recognition Conference*, pages 119–123, 1985.

[105] SW Lu, AKC Wong, and M Rioux. Recognition of 3-d objects in range images by attributed hypergraph monomorphism and synthesis. *IFAC Proceedings Volumes*, 18(16):325–330, 1985.

[106] Hugues Hoppe, Tony DeRose, Tom Duchamp, John McDonald, and Werner Stuetzle. Surface reconstruction from unorganized points. In *Proceedings of the 19th annual conference on computer graphics and interactive techniques*, pages 71–78, 1992.

[107] Dana H Ballard. Generalizing the hough transform to detect arbitrary shapes. *Pattern recognition*, 13(2):111–122, 1981.

[108] John Illingworth and Josef Kittler. A survey of the hough transform. *Computer vision, graphics, and image processing*, 44(1):87–116, 1988.

[109] Thomas C Henderson and Wu So Fai. The 3-d hough shape transform. *Pattern recognition letters*, 2(4):235–238, 1984.

[110] Yolande Belaïd and Roger Mohr. Planes and quadrics detection using hough transform. In *7th International Conference on Pattern Recognition (ICPR'84)*, pages 1101–1103. IEEE, 1984.

[111] Richard O Duda and Peter E Hart. Use of the hough transformation to detect lines and curves in pictures. *Communications of the ACM*, 15(1):11–15, 1972.

[112] Lei Xu, Erkki Oja, and Pekka Kultanen. A new curve detection method: randomized hough transform (rht). *Pattern recognition letters*, 11(5):331–338, 1990.

[113] Philip HS Torr and Andrew Zisserman. Mlesac: A new robust estimator with application to estimating image geometry. *Computer vision and image understanding*, 78(1):138–156, 2000.

[114] Ondrej Chum and Jiri Matas. Matching with prosac-progressive sample consensus. In *2005 IEEE computer society conference on computer vision and pattern recognition (CVPR'05)*, volume 1, pages 220–226. IEEE, 2005.

[115] Ondrej Chum and Jirı Matas. Randomized ransac with td, d test. In *Proc. British Machine Vision Conference*, volume 2, pages 448–457, 2002.

[116] José María Martínez-Otzeta, Itsaso Rodríguez-Moreno, Iñigo Mendialdua, and Basilio Sierra. Ransac for robotic applications: A survey. *Sensors*, 23(1):327, 2022.

[117] Hesai Technology Co., Ltd. PandarXT-32: 32-Channel Medium-Range Mechanical LiDAR User Manual. `http://www.oxts.com/wp-content/uploads/2021/01/Hesai-PandarXT_User_Manual.pdf`,, 2020. Accessed: 08.08.2024.

[118] BMAX. Maxmini B3 Plus. `https://www.bmaxit.com/Maxmini-B3-Plus-pd722218588.html`,, 2024. Accessed: 08.08.2024.

[119] Paul Heinisch. LiDAR Distortion Correction on Spherical Robots. `https://github.com/deepcodin/AMDC`, 2023. Accessed: 16.08.2024.

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 2024