FAKULTÄT FÜR
INFORMATIK

Otto-von-Guericke-University Magdeburg

Faculty of Computer Science

Institute for Intelligent Cooperating Systems

Comparison of Real-Time Plane Detection Algorithms on Intel RealSense

Bachelor Thesis

Author:

Lukas Petermann

Examiner:

Prof. Frank Ortmeier

2nd Examiner

M.Sc. Marco Filax

Supervisor:

M.Sc. Maximilian Klockmann

Magdeburg, 32.13.2042

Contents

1 Introduction

Man-made environments usually contain planar structures to a large extent. They are a central component in numerous use cases in the fields of Augmented and Virtual Reality, as well as robotics.

1.1 Real-Time Plane Detection

What is Real-Time Plane Detection (RTPD) Some use cases are subject to temporal restriction of some degree. If the temporal constraint is minimal or a task must be executed as quickly as possible, the term *real-time* is often used. In the context of plane detection, real-time performance is often necessary when the subject is in motion. A possible scenario could involve an autonomous robot moving through a building. Here, the process of detecting planar surfaces in the vicinity is necessary as, otherwise, the robot could crash into a nearby wall or Door.

Problem of RTPD The technology for plane detection in real-time already exists. The Microsoft HoloLens is able to detect planes by processing meshes of the environment. The downside Ausserdem ist bisher kein einheitlicher vergleich angestellt worden, daher ist es nicht direkt möglich einen eindeutig besten algorithmen anhand einer literaturrecherche auszuwählen.

wie gehen wir das problem an Um ebenenfindung benutzerfreundlicher und allgemein inklusiver zu machen, wird in dieser arbeit daher die möglichkeit der ebenenfindung auf off-the-shelf hardware evaluiert.

Dies wird angegangen, indem verschiedene PDA in einer strukturierten literaturrecherche verglichen werden

1.2 Structure of this work

In the following chapter, the basics are explained. In the concept3, the problem is defined in more detail and a solution is proposed. The fourth chapter focuses on the implementation details of the previous chapter. The proceding chapter deals with the comparison of plane detection algorithms and the evaluation thereof. Finally, the results and future work are presented.

2 Background

In this chapter, we will present relevant literature needed to completely understand the proposed concept of chapter 3.

2.1 SLAM

SLAM (Simultaneous Localization And Mapping) algorithms aim to solve a usual problem in the field of unmanned robotics; A robot finds itself in an unknown environment and attempts to build a coherent map while keeping track of its location. The robot uses use-case-specific sensors to obtain a snapshot of its current surroundings, which it then uses to update and enhance its known map (Mapping). The robot then attempts to accurately estimate its position based on the updated map. The new information about its position is processed during the next map update. Over decades of research, varieties of different (combinations of) sensors have been employed to solve this problem more accurately and efficiently. Internal odometry sensors alone can be unreliable if the robot moves over uneven or slippery surfaces. For that reason, visual SLAM(V-SLAM) methods like *MonoSLAM*[6] or *Dense Visual SLAM*[8] integrate additional visual input of camera sensors into their algorithm.

2.2 Intel Realsense

In this work, we use the Intel RealSense tracking camera T256 and the RGB-Depth camera D455. An excerpt of the camera specifications can be found in Table 2.1. Furthermore, both cameras have an integrated Inertial Measurement Unit (IMU) which is used to compute its position in combination with visual input.

Furthermore, Intel provides a software development kit, namely RealSense SDK, which allows easy and efficient use of the cameras. The SDK runs on both Windows and Ubuntu, and a ROS adaptation is also provided in Intel's Github repository [1].

[1] https://github.com/IntelRealSense/realsense-ros

	Image	Type	Resolution	Diag. FOV	Shutter	Price	max. FPS
D455	Stereo	RGB-D	1280x720	111°	global	419$	90
T265	Stereo	Tracking	848 x 800	163°	global	199$	30

Table 2.1: Intel RealSense T256 and D455 camera specifications. More information and the complete Datasheets can be found on https://www.intelrealsense.com/.

RTAB-MAP

RealSense-ROS internally uses a SLAM algorithm for map building, namely RTAB-MAP (Real-Time Appearance-Based Mapping)[9]. Unlike purely visual-based SLAM algorithms, RTAB-MAP also takes input from odometry sensors, as well as an optional additional input in form of two- or three-dimensional lidar scan. All these inputs are combined during a synchronization step, and the results thereof are passed to RTAB-MAP's *Short-Term-Memory* (STM). The STM assembles a new node from the new inputs and inserts it into the map graph. Based on the newly inserted node, RTAB-MAP attempts to determine if the current location has already been visited earlier, also known as *loop closure*. If a loop closure is detected, i.e., RTAB-MAP detects the re-visiting of a known location, the map graph is optimized and thus minimized. In addition, the global map is reassembled in correspondence with the new information. The resulting map is published in the form of an unorganized point cloud.

RTAB-MAP's general workflow is shown below in Figure 2.1:

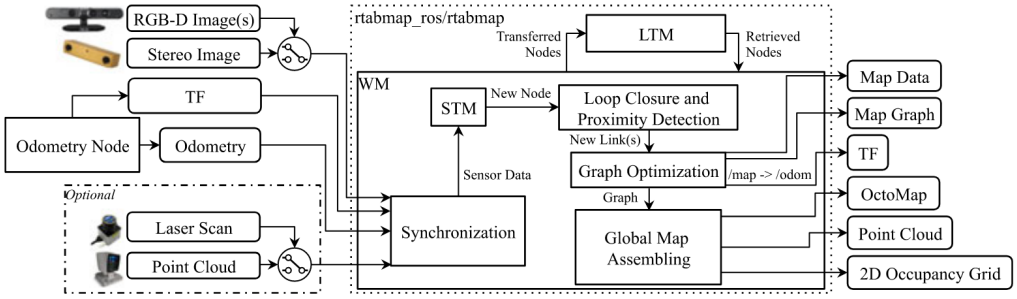


Figure 2.1: Block diagram of RTAB-MAP's main node. Taken from [9, Figure 1]

2.3 Plane Detection

introduction The field of plane detection has been around for decades. Most methods of detecting planar regions are based on one of three main categories:

- Hough Transform (HT)
- RANSAC (RC)
- Region Growing (RG)

Hough Transform

The original motivation behind the Hough transform was detecting lines in images or 4. All points are sequentially processed via a voting procedure to detect the best fitting line over a set of 2d points. Multiple lines with different orientations are fit through each given point p. Because a line in slope-intercept form parallel to the y-axis would lead to an infinite slope, the Hesse normal form is chosen as the primary line representation. In Hesse normal form, an individual line can be parameterized with a pair (r, θ), with r being the orthogonal distance origin to the plane and θ being the angle between the x-axis and the line that connects the origin to the closest point on the line. This pair is also called a *Hough Space* in this context. Votes are cast on the corresponding value of θ, depending on the number of inliers within a specific *Hough Space* (r_i, θ_i). The map that connects the votes to each θ is called an *accumulator*. Finally, the best fitting line is determined by the number of votes it received.

RANSAC

RANSAC (RAndom SAmple Consensus) has been researched for decades. While many use cases revolve around image processing, it is also heavily employed in many plane detection algorithms[17, 21, 4]. RANSAC is an iterative process. Each iteration randomly samples a certain amount of data points and fits a mathematical model through them. The level of outliers determines the quality of the obtained model and preserves the best overall model.

Within the context of plane detection in 3D point clouds, an approach could involve random sampling of 3 points, fitting a plane through them, and counting the number of points within a certain range of the plane[21]. The model, in that case, could be a cartesian plane equation. This

Region Growing

Region Growing methods are often used in the field of image or point cloud segmentation [15, 19]. RG-based segmentation methods aim to grow a set of disjoint regions from an initial selection of seed points. The regions increase in size by inserting neighboring

points based on an inclusion criterion. The quality of the resulting regions depends on the choice of seed points, e.g., a very noisy seed point could decrease overall quality [13].

In the context of this work, a criterion for region growth could be the distance or curvature between a region and its adjacent data points.

2.4 Plane Detection Algorithms

This section describes four algorithms that are used for the evaluation during this work.

2.4.1 RSPD

RSPD (Robust Statistics approach for Plane Detection [1]) is based on region growing. After taking an unorganized point cloud as input, the procedure is divided into three phases; *Split, Grow and Merge*.

Split The authors propose to use an octree to recursively subdivide the point cloud. The subdivision is repeated until every leaf node contains less than 0.1% of the total amount of points. This is followed by a planarity test, during which the octree is traversed bottom-up. If all eight children of a node n are leaf nodes and fail the planarity test, n replaces its children and becomes a leaf node of its own. This procedure is repeated until the root of the octree is reached.

Grow In preparation for the growth phase, a neighborhood graph (NG) over the entire point cloud is created. Every node of NG represents one point and an edge between two nodes exists only if a k-nearest-neighbor search detects both points being in the same neighborhood.

The graph construction is subsequently followed by a breadth-first-search, during which a point x is inserted into a planar patch p if it satisfies the following conditions:

- x is not included in any patch
- x satisfies the inlier conditions for p

Merge In the last phase, the previously grown patches are merged. Two planar patches P_1 and P_2 can be merged, if the following conditions are met:

- the octree nodes of P_1 and P_2 are adjacent
- $P_1.n$ and $P_2.n$ have a divergence within a tolerance range
- at least one inlier of P_1 satisfies the inlier condition from P_2 and vice versa

This phase returns all maximally merged planar patches, i.e. the final planes.

2.4.2 OPS

OPS (Oriented Point Sampling for plane detection [17]) accepts an unorganized point cloud as input. First, a sample of points is uniformly selected. The normal vectors of these points are estimated using SVD and the k nearest neighbors, which had been obtained by the use of a k-d tree. An inverse distance weight function is employed to prioritize neighboring points that are closer to the sample of which the normal vector is currently being estimated.

After normal estimation, one-point-RANSAC is used to find the largest plane. Usual RANSAC implementations sample three points to fit a plane, however, OPS fits a plane with only one sample point and its normal vector. Once a plane with the most inliers is obtained, its normal vector is re-estimated using SVD on all inliers and all inliers are removed from the point cloud. This process is repeated until the number of remaining points falls below a predefined threshold θ_N.

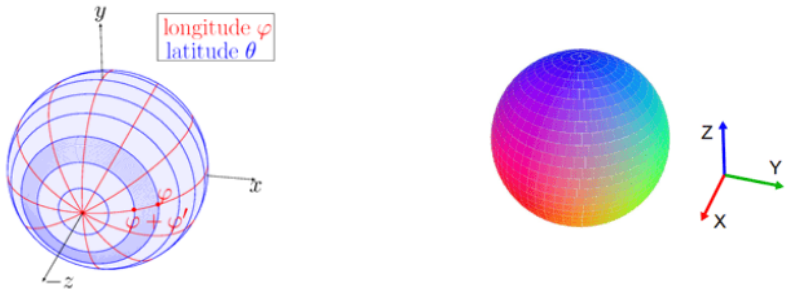
2.4.3 3D-KHT

Limberger and Oliveira propose a hough transform-based plane detection method, which accepts unorganized point clouds as input [10]. The point cloud is spatially subdivided. The authors propose the usage of octrees over k-d trees because the k-d tree lacks efficiency in creation and manipulation. Furthermore, the octree succeeds in capturing the shapes inside the point cloud, while the k-d tree does not.

Each leaf inside the octree continues subdividing until the points inside a leaf node are considered approximately coplanar, or the number of points is less than a predefined threshold. The authors recommend this threshold to value 30 for large point clouds. After the approximately coplanar nodes are refined by removing outliers, a plane is fit through the remaining points.

This plane π can, in polar coordinates, be uniquely described by a triple (ρ, θ, ϕ), with ρ being the orthogonal distance from the origin to the plane, θ being the azimuthal

angle, and ϕ being the inclination. Inspired by Borrmann et al.[5], an accumulator ball (Fig. 2.2b) is used for the voting procedure because the cells in polar regions are smaller (and therefore contain fewer normal vectors) in three-dimensional accumulator arrays, as portrayed in Figure 2.2a.



(a) (b)

Figure 2.2: Accumulator array (a), taken from [5, Figure 3]. Accumulator ball(b) used in 3D-KHT, taken from [10, Figure 5].

During the voting procedure, votes are not cast for each data point but rather on previously calculated approximately coplanar clusters. When casting a vote on a given cluster c_i with its plane (represented by (ρ, θ, ϕ)), the corresponding entry in the accumulator ball is updated. With this update, its neighboring clusters also receive a vote determined by the uncertainty value of c_i. Due to the non-discrete values of uncertainty, the votes are floating-point values as well.

All Peaks within the accumulator ball are detected in the last step. Because the votes tend to be sparsely distributed [10, Section 3.4], an auxiliary array A is used to memorize the entries inside the accumulator that are set. When an accumulator index is assigned a value for the first time, it is also added to A. Therefore, it is only necessary to iterate the auxiliary array to find peaks inside the accumulator. Furthermore, an intermediary smoothing step is performed by merging adjacent peaks inside the accumulator and storing them in A. Then, A is sorted in descending order. If a cell c in the accumulator has not yet been visited during iteration, c is considered a peak. In addition, c and its 26 neighboring cells are tagged as *visited*. That way, the most dominant plane, i.e., the one with the most votes, is detected first. Finally, the detected planes are sorted by the number of different clusters that voted for them.

2.4.4 OBRG

OBRG (Octree-Based Region Growing [19]) is also a method that employs region growing.

First, an unorganized point cloud is recursively subdivided using an octree. An octree node n repeatedly subdivides itself into eight children until the level of n supersedes a predefined maximum subdivision value or if the amount of contained points in n is less than a predefined minimum of included points. Saliency features are calculated for every leaf node in preparation for the region growing step. A normal vector is obtained by performing a principle component analysis (PCA) on the points inside each leaf node. The best-fitting plane of each leaf is defined by the mean normal vector and its center point. A residual value is obtained by taking the RMS of the distance of all included points to the plane.

For the region growing phase, all leaf nodes are selected as individual seed points. Starting from the seed with the lowest residual value, which relates to a low amount of noise, a neighboring leaf node n is inserted into the region if n does not belong to any region and the angular divergence between both normal vectors is smaller than a predefined threshold.

Lastly, a refinement step is employed. Fast refinement (FR) is performed on regions that succeed in a planarity test, i.e., 70%-90% of included points fit the best plane. FR is leaf-based, and all previously unallocated neighboring nodes that satisfy an inlier criterion are added to the region. General refinement (GR) is performed on regions that are considered non-planar. In contrast to the fast refinement, GR is point based. Therefore, points from neighboring and previously unallocated leaf nodes are considered and inserted into the region if they, too, satisfy the inlier criterion. The refinement process returns a complete set of planar regions.

3 Concept

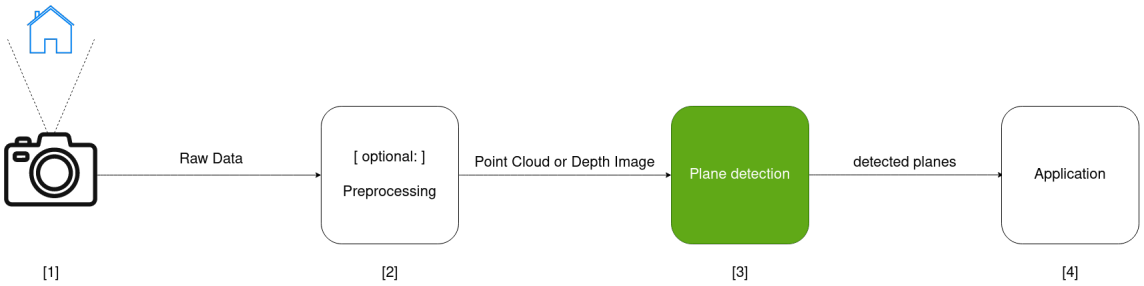


Figure 3.1: A camera records an environment. The recorded data is either preprocessed, or passed directly to the plane detection algorithm. The resulting planes are subsequently exported to an arbitrary application.

Within a given use case or scenario that involves the detection of planes, the user records the surroundings with a camera. The raw data is then either directly, or after some preprocessing steps, passed to a plane detection algorithm, which then hands the detected planes to an application for further use. This application could use those planes to define the playable area in an augmented reality (AR) video game or alternatively build a digital floor plan, or 3D model, of an apartment.

Especially in scenarios where the user moves through the environment, the time between recording and the planes reaching the application is crucial. If an autonomous robot moves through an apartment complex, the delay must be as short as possible to avoid a collision with a wall the robot has not yet detected. The problem can therefore be defined in such a way that planes must be detected in real-time and at the same time be sufficiently precise for the given use case.

To optimize the entire process, shown in 3.1, this work is focused on the plane detection step (green).

Furthermore, we focus on indoor environments, as motivated by Chapter 1. This includes the buildings we encounter in our normal lives, whether it is the home we live in, the office we work in, or a stripped-down version of a building during construction.

3.1 Used Sensors

To efficiently tackle the previously defined problem, and according to step one of figure 3.1, we need a camera sensor to record the environment and obtain data to detect planes therein.

Numerous cameras suffice for this task, each of which varies in different aspects. For this work, we use the Intel RealSense T256 Tracking Camera and the Intel RealSense D455 RGB-D Camera, because of their compatibility, since the T256 can be used in combination with any depth camera from the D400 series. Furthermore, both cameras can perform all position calculations which reduces the load on the hosting device, e.g., laptop or microcontrollers like Raspberry PI's.

Since we are especially interested in the detection of planes in complete environments, it is necessary to be able to build a map out of the data the cameras record continuously. In addition to the cameras, the Intel RealSense software provides a way to perform map assembly, see step 2 of Figure 3.1.We integrate *realsense-ros*, the ROS wrapper of Intel RealSense, into our process of plane detection.

Realsense-ros internally uses a SLAM algorithm called RTAB-MAP [9] for map-building. RTAB-MAP is responsible for building a coherent map from a continuous stream of data that is being recorded and published by the two cameras. It is worth noting, that the success of this work does not depend on the specific SLAM algorithm being chosen. We select RTAB-MAP because it is already included in the RealSense package and its reported performance suffices for this work, primarily since we don't focus on SLAM algorithms in this work.

We can further specify Figure 3.1 by inserting the aforementioned technology as follows:

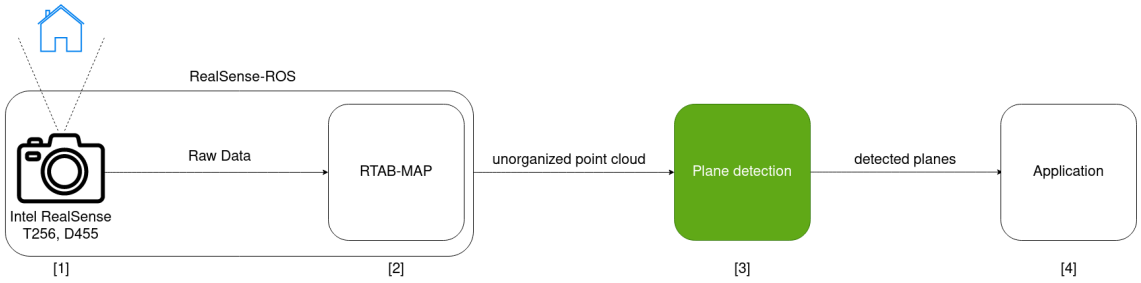


Figure 3.2: The concrete procedure of the plane detection process. The Intel RealSense Cameras, T256 and D455, record data ([1]), which is passed to RTAB-MAP ([2]). After map assembly, an unorganized point cloud is handed to a plane detection algorithm ([3]). The detected planes are given to a use-case-specific application ([4]).

3.2 Selection Plane Detection Algorithms

Naturally, appropriate algorithms are needed to perform real-time plane detection. First, we define meaningful criteria to select plane detection algorithms. Then, we use these criteria to select appropriate algorithms from the list of algorithms.

Type of Input

Usually, the data representation of the recorded environment passed to the plane detection algorithm falls under one of three categories:

- *unorganized* or *unstructured point cloud* (UPC)
- *organized* or *structured point cloud* (OPC)
- *(depth-) image* (D-/I)

As stated before, we focus on detecting planar structures in the entire environment rather than just distinct segments thereof. In addition, only the unorganized/unstructured point clouds offer a complete view of the recorded environment.

Detected Plane Format

Which specific representation the detected planes take the form of is also essential. If no uniform output type can be determined, consequently, no uniform metric for comparison can also be found. Since the algorithms process point clouds, we choose to stay within the realm of points, i.e., an arbitrary plane should be represented by the set of points included in the plane (inliers). The representation, being a list of points, enables further processing of the detected planes. A list of points would, in contrast to some plane equation, enable us to detect holes in planes, e.g., an open door or window, which can be helpful in any use case involving remodeling architectural elements. It also allows further filtering of planes based on a density value that we can calculate over the bounding box and the number of points, e.g., removing planes with a density lower than a certain threshold.

Learning based

Learning-based methods, e.g. Deep learning, generally have varying levels of bias, depending on the training data. Another reason against the use of learning-based methods is that we choose not to require a GPU to replicate our findings.

Availability

Lastly, we include the availability of an algorithm in our set of criteria. Each algorithm to be compared needs to run on the same system to exclude the underlying hardware as a factor from any experiments.

3.3 Algorithms

RSPD - Robust Statistics Approach for Plane Detection

RSPD expects an unorganized point cloud as input. The detected planes are written to file as either a list of inliers or represented by a center point, a normal vector and UV extents of the plane. As explained above 3.2, we select the inliers as the primary plane representation.

OPS - Oriented Point Sampling

OPS also takes an unorganized point cloud as input. Like RSPD, the default representation of a detected plane is a tuple of normals vector and center coordinate, while also reporting the orientation. without affecting the algorithm, we modify the output format to instead include the inliers.

3DKHT - 3-D Kernel-based Hough Transform

With 3D-KHT, as with other octree-based methods, the performance depends, to some degree, on the level of subdivision. In the provided implementation, the octree keeps dividing until either the number of points in the current node is lower than a set minimum, or the level of an octree node is higher than a predefined maximum.

A total of six different presets of parameters are included with the official implementation. Since this work does not focus on the evaluation and analysis of a single method, we performed experiments with all presets on Area 3 of the S3DIS[3] dataset, the results thereof can be seen in Figure 3.3 The two leftmost values for the maximum octree subdivision seem to yield the best results compared to the other values. Because we expect a greater subdivision level to yield better results on scenes of larger dimensions, we choose to perform all further experiments with an octree subdivision value of two.

irgendwas ergibt hier keinen sinn. zumindest passt meine erklärung nicht ganz zu der grafik

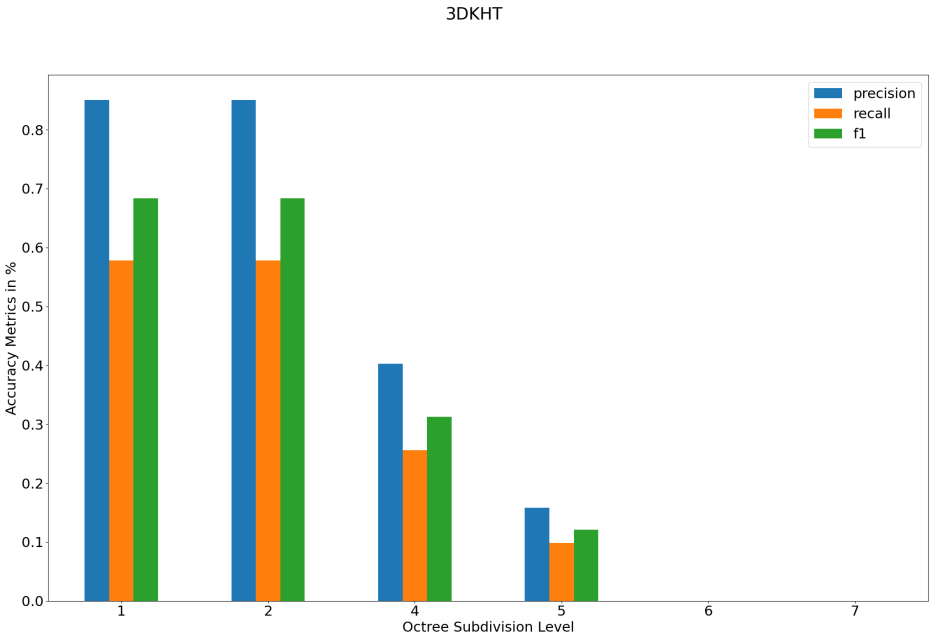


Figure 3.3: Results of 3D-KHT for different pre-set subdivision values

OBRG - Octree-based Region Growing

PEAC - Probabilistic Agglomerative Hierarchical Clustering

3.3.1 Summary: Plane Detection Algorithms

Summary Plane Detection Algorithms

To effectively compare the presented algorithms, the data on which each algorithm performs the plane detection should ideally be the same. Considering algorithms that run on anything other than UPC, would necessitate finding a data set that includes equivalent point clouds for both the structured and the unstructured case. Since these algorithms would disregard the global structure of the point cloud, we deem them not feasible for our use case and thus exclude them from our evaluation.

We exclude learning-based methods for the reasons previously stated in Subsection 3.2.

	Input Data	Plane format	Learning-Based	Availability
RSPD [1]	UPC	inliers	N	Y
OPS [17]	UPC	inliers	N	Y
3DKHT [10]	UPC	inliers	N	Y
OBRG [19]	UPC	inliers	N	N
PEAC [7]	OPC	inliers	N	Y
CAPE [15]	OPC	normal, d	N	Y
SCH-RG [14]	OPC	inliers?	N	N
D-KHT [18]	DI	inliers	N	Y
DDFF [16]	DI	indices	N	Y
PlaneNet [11]	I	normal, d	Y	Y
PLaneRecNet [20]	I	? / -	Y	Y
PlaneRCNN [12]	I	normal + ?	N	Y

Table 3.1: Plane Detection Algorithms

For an even comparison, the detected planes would have to be in the same format because, even for the same plane, representations could very well lead to different results, e.g., a plane in cartesian form compared to the same plane, described by its inliers. Asserting comparability, we exclude all methods which do not offer a plane representation by inliers.

Lastly, writing our own implementation of methods for which no implementation is available or for which the respective publication does not focus on the implementation details would go beyond the scope of this work. Finally, we end up with, and thus include, the following plane detection algorithms in our evaluation:

- RSPD
- OPS
- 3D-KHT
- OBRG

3.4 Real-Time

To determine whether or not an algorithm runs in real-time, we must first define the meaning of real-time.

We have to consider possible hardware limitations, data flow, and simply how often it is needed to perform calculations in correspondence with the given use case.

The recorded raw data is not directly sent to the plane detection algorithm but instead given to RTAB-MAP, which then performs calculations to update and publish the map. Therefore, the upper limit is the frequency of how often RTAB-MAP publishes those updates, which by default is once per second. According to this upper limit, we consider an algorithm *real-time applicable*, if it achieves an average frame rate of minimum 1, e.g., the algorithm manages to process the entire point cloud and detect all planes within one second.

3.5 Summary

Many applications have constraints in the form of a temporal component. Applications that include plane detection are no exception. The calculation of planes is an obvious bottleneck in the procedure shown in Figure 3.1. To evaluate to what extent it is possible to perform precise plane detection with a real-time constraint, we compare selected algorithms.

4 Implementation

To ensure a uniform comparison of the algorithms selected in the concept, the external circumstances must be identical for each algorithm. This chapter will detail the implementation thereof.

4.1 System Setup

It is necessary to perform all experiments on the same machine to ensure a consistent comparison. We implement all algorithms and further architecture on a Lenovo IdeaPad 5 Pro, which runs Linux Ubuntu 20.04.5. The laptop has an AMD Ryzen 7 5800H CPU and 16 GB of RAM.

We install the most recent ROS distribution, *ROS Noetic Ninjemys*, as well as *realsense-ros* with all additional dependencies.

4.2 Plane Detection Algorithms

We implement RSPD and OPS using their respective open source implementations on GitHub[1] [2]. Note that, while the implementation of RSPD is provided by the author, we could not determine whether the user who uploaded his implementation of OPS is affiliated with Sun and Mordohai. Both methods are implemented in C++ and depend on the C++ linear algebra library *Eigen*[3] and the C++ API of the Point-Cloud Library[4], *libpcl-dev*.

The authors of 3D-KHT, provide an implementation, in form of a Visual Studio project, on their website [5]. Since the laptop we use does not run Windows, we use *cmake-converter* [6] to convert the solution to a CMake project we can build using *make*.

[1] https://github.com/abnerrjo/PlaneDetection
[2] https://github.com/victor-amblard/OrientedPointSampling
[3] https://eigen.tuxfamily.org/index.php
[4] https://pointclouds.org/
[5] https://www.inf.ufrgs.br/ oliveira/pubs_files/HT3D/HT3D_page.html
[6] https://cmakeconverter.readthedocs.io/

OBRG

To our knowledge, no open-source implementation is available for the algorithm. We, therefore, use our own implementation.

We implement the algorithm using python. We choose to write our own octree implementation for spatial subdivision of our point cloud, since public libraries like *open3d* are limited in terms of leaf node functionality. The subdivision is followed by calculating the saliency features using *open3d*'s normal estimation function. We follow the pseudocode as stated in [19, Algorithm 1]. We modify the insertion into the set of regions by adding a containment check, to avoid redundancy of regions. By reducing the number of regions (incl. redundancies), we also reduce the total calculation time. Since the exact values of all thresholds have not been specified, we empirically select as follows:

- $r_{th} = 0.1$
- $\theta_{th} = 0.3$
- $d_{min} = 0.1$

To determine a region's planarity, we calculate the number of points that fit the best fitting plane within a predefined threshold. If that number supersedes the proposed 70%-90%, depending on the expected noise, the region is considered planar [19, Section 3.4].

It is worth noting that the choice of implementation using python is inferior considering calculation time when compared with an equivalent implementation in C++. Writing an optimized implementation in C++ would, therefore, go beyond the scope of this work, as the optimization of a single method is not our focus.

4.3 Dataset / Ground Truth

Since the selected data set focuses on semantic segmentation rather than detection of planar regions, we manually create a ground truth. We use the open-source 3D point cloud and mesh processing software CloudCompare. Since we cannot assume all walls to be planar or that, e.g., the tops or three adjacent tables always form the same number of planes (see Figure **??**), we have to view each point cloud and segment the included planes manually.

This process is very time-consuming, and to slightly reduce the time spent, we perform an initial analysis of all scenes within a given area and omit scenes that seem to inherit no noticeable differences.

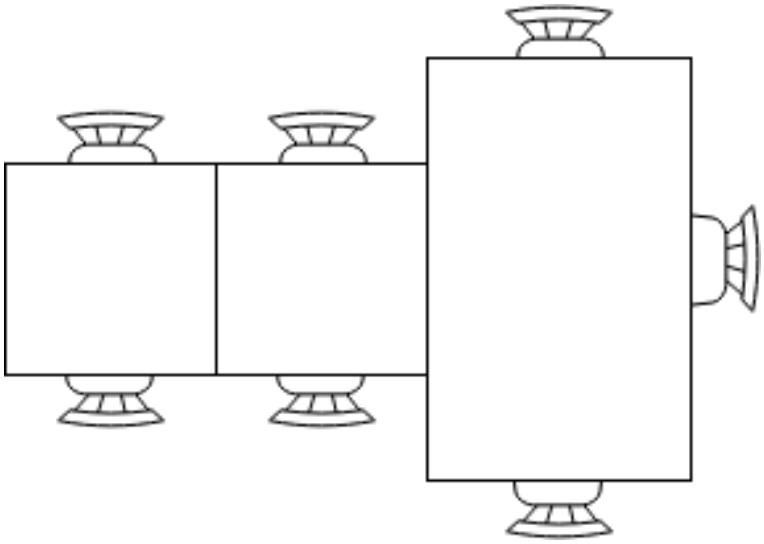


Figure 4.1: The given ground truth considers these tables to be three distinct objects. Within the context of plane detection, the three table tops would form exactly one plane.

5 Evaluation

In diesem kapitel werden zuvor ausgewählte algorithmen einheitlich verglichen und die resultierenden ergebnisse ausgewertet.

5.1 Evaluation Protocol

To determine the feasibility of performing real-time plane detection, we need to conduct experiments with the selected algorithms.

Another important factor for comparability is the data set, on which the experiments are conducted on. Because each publication from the presented algorithms uses a different data set for its evaluation, we cannot objectively select an algorithm to be the "best". Furthermore, to the best of our knowledge, there is no data set, that contains an incrementally growing and unordered point cloud with corresponding ground truth.

Thus, we first evaluate the algorithms on a dataset while excluding the temporal component. We do this by performing plane detection on whole point clouds, rather than incrementally growing ones.

We then perform experiments, this time including the temporal component, by performing calculations at each time step and evaluating them individually.

Lastly, through comparison, as well as analysis of those different experiments, a statement will be given as to whether and how well plane detection is possible in real-time.

For the evaluation of a given dataset, by comparing the test results with the ground truth, we took inspiration from Araújo and Oliveira Apr. 2020, especially their *Results* chapter.

5.1.1 Metrics

To quantitatively evaluate an algorithm's performance, we calculate the precision, recall and the f1 score. First, we regularize the original point cloud to reduce complexity and furthermore to avoid proximity bias, because of the inverse relationship between distance to sensor and cloud density. This regularization is obtained through voxelization of the point cloud.
With this voxel grid, we can now calculate corresponding sets of voxels for each point cloud representing a plane. In the next step, we compare our planes from the ground truth with the planes obtained from an algorithm to obtain a list of corresponding pairs of ground truth and found planes.
A grund truth plane p_{gt_i} is marked as *detected*, if any plane from the list of found planes achieves a voxel overlap of $\geq 50\%$. With this list of correspondences, we calculate precision, recall and the f1-score as explained in the following. For a given ground truth plane p_{gt_j} and a corresponding found plane p_{a_k} we can sort a given voxel v_i into the categories *True Positive(TP), False Positive(FP) and False Negative(FN)* as follows.

$$v_i \in p_{gt_j} \wedge v_i \in p_{a_k} \Rightarrow v_i \in TP$$
$$v_i \in p_{gt_j} \wedge v_i \notin p_{a_k} \Rightarrow v_i \in FN$$
$$v_i \notin p_{gt_j} \wedge v_i \in p_{a_k} \Rightarrow v_i \in FP$$

With those four rules, we can calculate the precision, recall and F1 score like this:

$$Precision = \frac{|TP|}{|TP| + |FP|}$$

$$Recall = \frac{|TP|}{|TP| + |FN|}$$

$$F1 = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall}$$

Aside from the accuracy, we also need to compare the time each algorithm needs to find its respective set of planes. For that, we measure the time spent in the plane detection phase, excluding any preprocessing or postprocessing steps. To measure the detection time, we log the exact times before and after calculations and write the difference to a file.

5.1.2 Dataset

We select the Stanford Large-Scale Indoor Spaces 3D Dataset(S3DIS)[2] to evaluate each plane detection algorithm on even grounds. S3DIS was recorded in three different buildings and divided into six distinct areas, including 272 different scenes. A detailed statistic of the included scene types can be found in Table 5.1. An individual scene has a complete unstructured point cloud and a list of annotated files representing semantically different objects that can be found therein. Since our focus is not on 3D semantic segmentation, we manually select planar regions using CloudCompare[1] to obtain a list of sub-clouds.

Furthermore, one could argue that an uneven distribution of scene types introduces a particular bias. While it is true that the distribution is quite uneven, the data set nevertheless reflects a realistic distribution of scene types since it is not realistic that a building contains only lecture halls. On the other hand, it is appropriate to assume that an office complex contains a substantial amount of hallways needed to connect all these offices.

Scene Categories	Area_1	Area_2	Area_3	Area_4	Area_5	Area_6	TOTAL
office	31	14	10	22	42	37	156
conference room	2	1	1	3	3	1	11
auditorium	-	2	-	-	-	-	2
lobby	-	-	-	2	1	-	3
lounge	-	-	2	-	-	1	3
hallway	8	12	6	14	15	6	61
copy room	1	-	-	-	-	1	2
pantry	1	-	-	-	1	1	3
open space	-	-	-	-	-	1	1
storage	-	9	2	4	4	-	19
WC	1	2	2	4	2	-	11
TOTAL	45	39	24	49	55	53	272

Table 5.1: S3DIS Disjoint Space Statistics

5.1.3 Real-Life Test

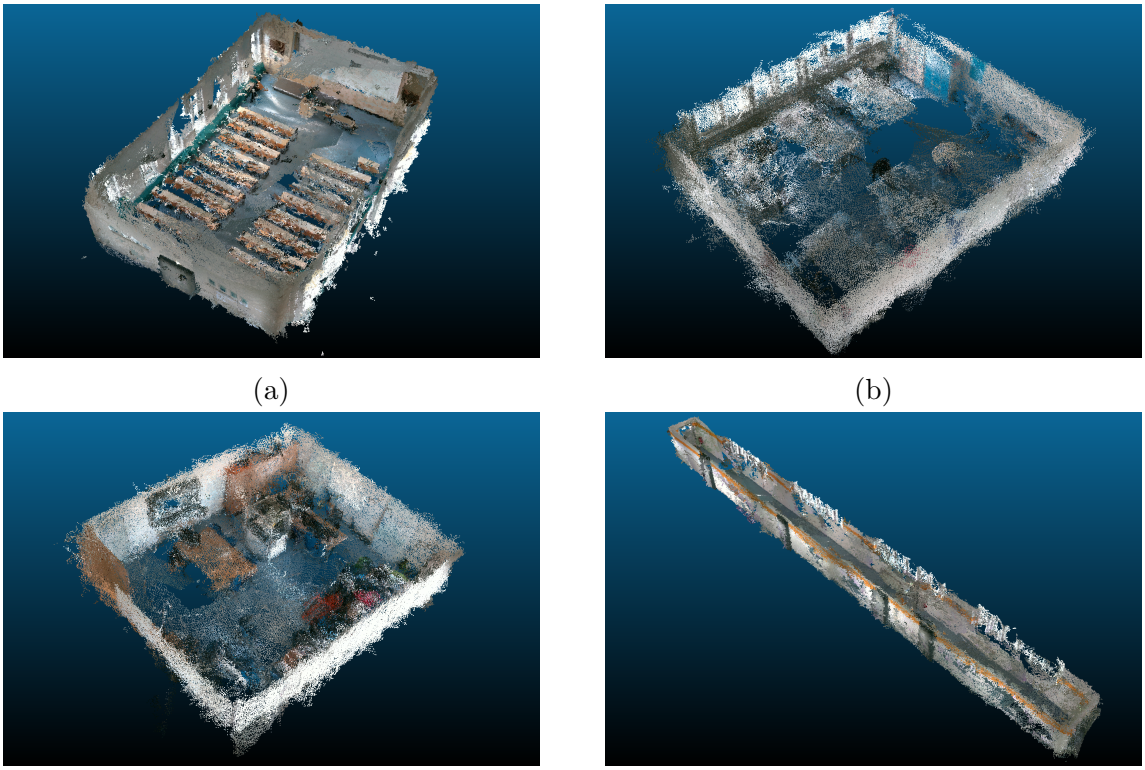
We record an incrementally growing data set in the Faculty of Computer Science at Otto-von-Guericke University Magdeburg. To perform a thorough comparison between the static and the dynamic experiment, we record a scene for each of the following scene types of S3DIS:

[1] https://cloudcompare.org/

- office
- conference room
- auditorium
- hallway

We disregard scene types where the number of occurrences in S3DIS is negligible(see Table 5.1). Furthermore, we do not create a recording for the storage type because the spatial extents are too small to introduce a temporal component effectively.

Running *realsense-ros* and holding our cameras, we walk through different parts of the building, scanning to the best of our ability. We save each incremental map update to a file for later usage.



(a) (b)

(c) (d)

Figure 5.1: The recordings for each scene type: (a)auditorium, (b) conference room, (c) office and (d) hallway.

We create a set of ground truth planes gt_{end} for only the most recent update of each scene, e.g., for the entire recording. The datasets for To prepare for the evaluation of a map m_t at a given time t, we crop all planes in gt_{end} by removing all points that are not

present in m_t, as shown in Figure 5.2. We speed up this expensive process by employing a KD-Tree neighbor search with a small search radius since we only need to know whether a certain point is present or not. Furthermore, we remove planes from the ground truth if the number of included points falls short of a threshold.

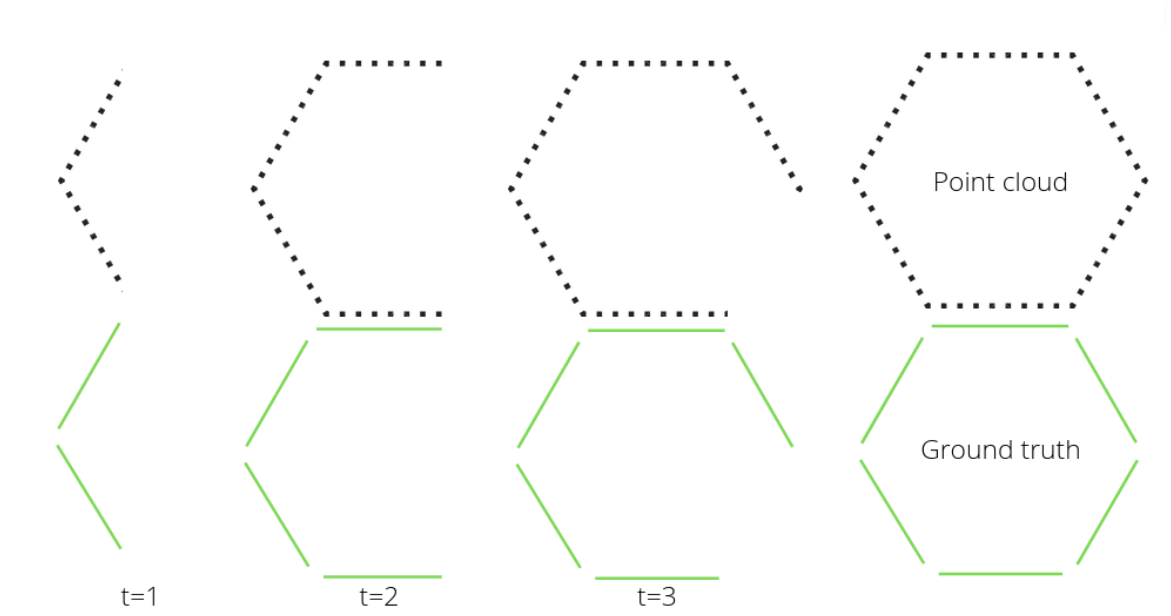


Figure 5.2: Dynamic ground truth generation. All planes that are included in *Ground Truth* are cropped depending on the available point cloud at each time t

5.2 Results

This section deals with the results of the experiments. The individual results of both experiments are presented and analized.

Results S3DIS Experiments

	Precision	Recall	F1	Time(s)
RSPD	0.85	0.89	0.86	0.90
OPS	0.87	0.70	0.77	16.75
3DKHT	0.75	0.43	0.53	0.91
OBRG	0.77	0.69	0.72	44.66

Table 5.2: Average results of each algorithm over the S3DIS dataset.

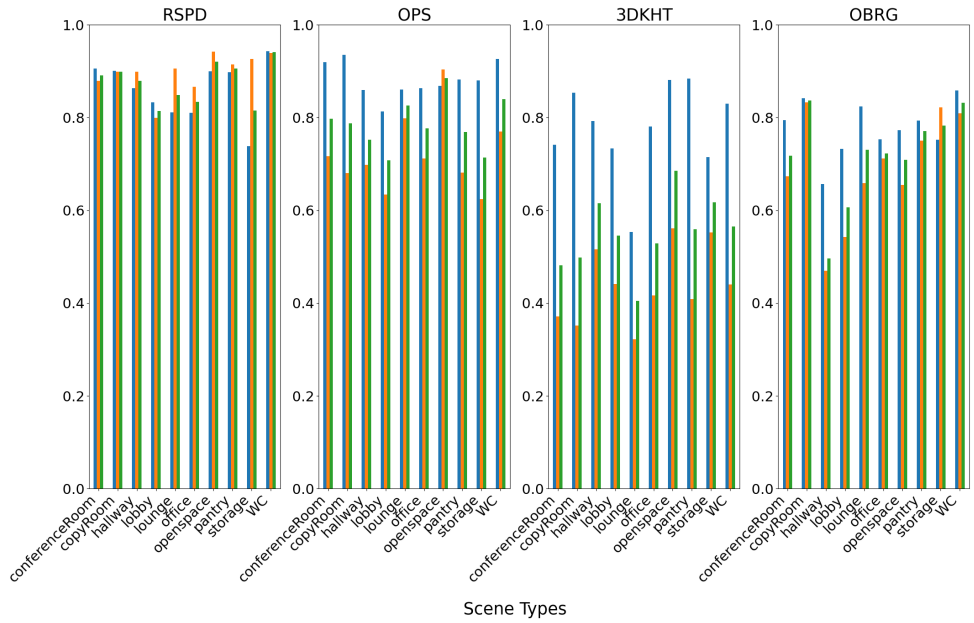


Figure 5.3: Average Accuracy for each scene type. The Precision is colored blue, recall is orange and the F1-score is green.

Every algorithm performs calculations on every scene included in S3DIS. The results of each algorithm on an individual scene type are reported in Figure 5.3 and Figure 5.4. As shown in Table 5.2, RSPD produces the overall best results with an average of 85% precision, 89% recall and an F1-score of 86%, as well as an average of 0.9 seconds of calculation time. The only other algorithm that achieves similar calculation times is 3D-KHT, which takes 0.91 seconds on average. Considering our definition of *real-time* in Section 3.4, RSPD and 3D-KHT are able to perform plane detecion in real-time. Still, 3D-KHT produces the worst overall accuracy results with an average precision of 75%, recall of 43%, and an F1-score of 53%.

With an average of about 17 seconds(OPS) and about 44 seconds(OBRG), both algorithms do not achieve real-time plane detection (see Table 5.2).

5.2.1 Results Real-Life Experiments

So far:

The calculation times of all algorithms except OBRG seem to be proportional to the size of the point cloud. The quality of plane detection, however, decraeses dramatically

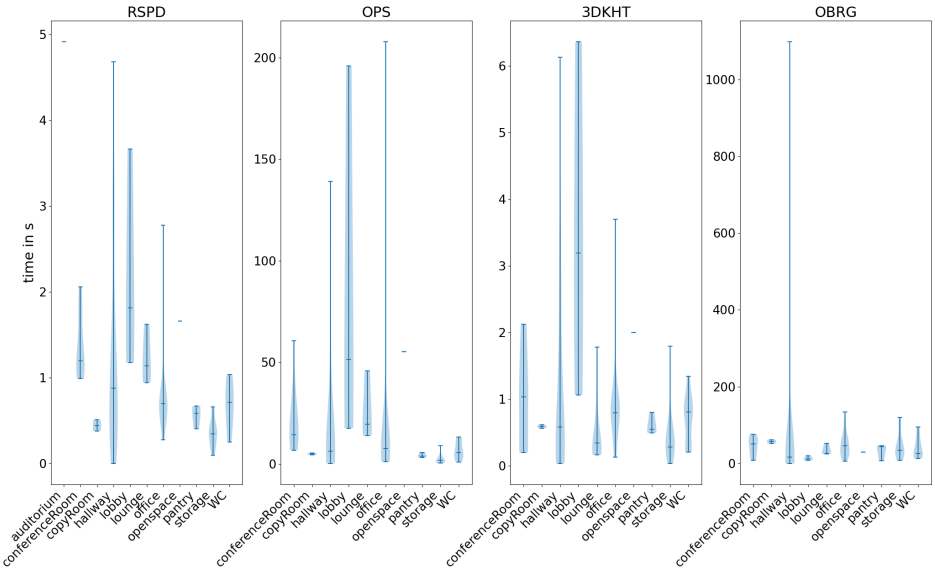


Figure 5.4: Average Time per scene type. Note, that the plots do not share the same y-axis.

in comparison to the Stanford Datasets. RSPD is the only dataset that is able to detect planes.

Summary Experiments

This section combines the preceding results of both experiments. RSPD is the only algorithm that produces comparable results to the Stanford experiment in the dynamic experiment. The remaining algorithms cannot reliably detect planes in an incrementally growing environment inheriting varying degrees of noise.

The reason for RSPD's dominance is likely caused by the inherent robustness against noise, as described in Section

Die ergebnisse der beiden experimente unterscheiden sich in folgendem punkt. Dazu sei gesagt, dass die experimente folgende übereinstimmungen haben. Das lässt sich so erklären. Alternative gründe davon könnten diese hier sein.

Ich denke RSPD ragt heraus, da hier besonders auf noise resistenz geachtet wurde.

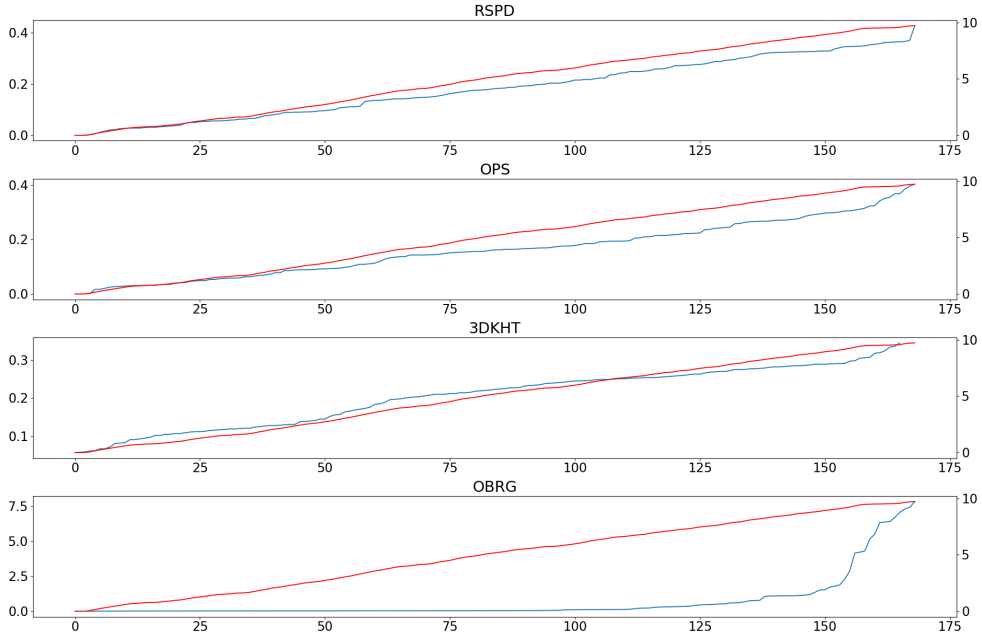


Figure 5.5: Calculation times(blue) of the hallway scene and cloud size(red) of each time step.

6 Conclusion and Future Work

Summary

Use case, scenario, real-world application, current problem

Daher das thema dieser arbeit

algorithmen

Testscenario / datensatz

Experimentaufbau

Ergebnisse der experimente

Fazit

Limitationen der ergebnisse

Algo x ist der beste, mögl verbesserungen

List of Figures

Bibliography

[1] Abner M. C. Araújo and Manuel M. Oliveira. "A robust statistics approach for plane detection in unorganized point clouds". en. In: *Pattern Recognition* 100 (Apr. 2020), p. 107115. ISSN: 00313203. DOI: 10.1016/j.patcog.2019.107115.

[2] I. Armeni et al. "Joint 2D-3D-Semantic Data for Indoor Scene Understanding". In: *ArXiv e-prints* (Feb. 2017). arXiv: 1702.01105 [cs.CV].

[3] Iro Armeni et al. "3D Semantic Parsing of Large-Scale Indoor Spaces". In: *Proceedings of the IEEE International Conference on Computer Vision and Pattern Recognition*. 2016.

[4] Ramy Ashraf and Nawal Ahmed. "FRANSAC: Fast RANdom Sample Consensus for 3D Plane Segmentation". en. In: *International Journal of Computer Applications* 167.13 (June 2017), pp. 30–36. ISSN: 09758887. DOI: 10.5120/ijca2017914558.

[5] Dorit Borrmann et al. "The 3D Hough Transform for plane detection in point clouds: A review and a new accumulator design". en. In: *3D Research* 2.2 (June 2011), p. 3. ISSN: 2092-6731. DOI: 10.1007/3DRes.02(2011)3.

[6] Davison. "Real-time simultaneous localisation and mapping with a single camera". In: *Proceedings Ninth IEEE International Conference on Computer Vision*. 2003, 1403–1410 vol.2. DOI: 10.1109/ICCV.2003.1238654.

[7] Chen Feng, Yuichi Taguchi, and Vineet R. Kamat. "Fast plane extraction in organized point clouds using agglomerative hierarchical clustering". en. In: *2014 IEEE International Conference on Robotics and Automation (ICRA)*. Hong Kong, China: IEEE, May 2014, pp. 6218–6225. ISBN: 978-1-4799-3685-4. DOI: 10.1109/ICRA.2014.6907776. URL: http://ieeexplore.ieee.org/document/6907776/.

[8] Christian Kerl, Jurgen Sturm, and Daniel Cremers. "Dense visual SLAM for RGB-D cameras". en. In: *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*. Tokyo: IEEE, Nov. 2013, pp. 2100–2106. ISBN: 978-1-4673-6358-7. DOI: 10.1109/IROS.2013.6696650. URL: http://ieeexplore.ieee.org/document/6696650/.

[9] Mathieu Labbé and François Michaud. "RTAB-Map as an open-source lidar and visual simultaneous localization and mapping library for large-scale and long-term online operation: LABBÉ and MICHAUD". en. In: *Journal of Field Robotics* 36.2 (Mar. 2019), pp. 416–446. ISSN: 15564959. DOI: 10.1002/rob.21831.

[10] Frederico A. Limberger and Manuel M. Oliveira. "Real-time detection of planar regions in unorganized point clouds". en. In: *Pattern Recognition* 48.6 (June 2015), pp. 2043–2053. ISSN: 00313203. DOI: 10.1016/j.patcog.2014.12.020. URL: https://www.inf.ufrgs.br/~oliveira/pubs_files/HT3D/HT3D_page.html.

[11] Chen Liu et al. "PlaneNet: Piece-Wise Planar Reconstruction from a Single RGB Image". en. In: *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*. Salt Lake City, UT: IEEE, June 2018, pp. 2579–2588. ISBN: 978-1-5386-6420-9. DOI: 10.1109/CVPR.2018.00273. URL: https://ieeexplore.ieee.org/document/8578371/.

[12] Chen Liu et al. "PlaneRCNN: 3D Plane Detection and Reconstruction From a Single Image". en. In: *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. Long Beach, CA, USA: IEEE, June 2019, pp. 4445–4454. ISBN: 978-1-72813-293-8. DOI: 10.1109/CVPR.2019.00458. URL: https://ieeexplore.ieee.org/document/8953257/.

[13] Aminah Abdul Malek et al. "Seed point selection for seed-based region growing in segmenting microcalcifications". en. In: *2012 International Conference on Statistics in Science, Business and Engineering (ICSSBE)*. Langkawi, Kedah, Malaysia: IEEE, Sept. 2012, pp. 1–5. ISBN: 978-1-4673-1582-1. DOI: 10.1109/ICSSBE.2012.6396580. URL: http://ieeexplore.ieee.org/document/6396580/.

[14] Hannes Mols, Kailai Li, and Uwe D. Hanebeck. "Highly Parallelizable Plane Extraction for Organized Point Clouds Using Spherical Convex Hulls". en. In: *2020 IEEE International Conference on Robotics and Automation (ICRA)*. Paris, France: IEEE, May 2020, pp. 7920–7926. ISBN: 978-1-72817-395-5. DOI: 10.1109/ICRA40945.2020.9197139. URL: https://ieeexplore.ieee.org/document/9197139/.

[15] Pedro F. Proença and Yang Gao. "Fast Cylinder and Plane Extraction from Depth Cameras for Visual Odometry". en. In: arXiv:1803.02380 (July 2018). number: arXiv:1803.02380 arXiv:1803.02380 [cs]. URL: http://arxiv.org/abs/1803.02380.

[16] Arindam Roychoudhury, Marceli Missura, and Maren Bennewitz. "Plane Segmentation Using Depth-Dependent Flood Fill". en. In: *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Prague, Czech Republic: IEEE, Sept. 2021, pp. 2210–2216. ISBN: 978-1-66541-714-3. DOI: 10.1109/IROS51168.2021.9635930. URL: https://ieeexplore.ieee.org/document/9635930/.

[17] Bo Sun and Philippos Mordohai. "Oriented Point Sampling for Plane Detection in Unorganized Point Clouds". en. In: arXiv:1905.02553 (May 2019). arXiv:1905.02553 [cs]. URL: https://github.com/victor-amblard/OrientedPointSampling.

[18] Eduardo Vera et al. “Hough Transform for real-time plane detection in depth images”. en. In: *Pattern Recognition Letters* 103 (Feb. 2018), pp. 8–15. ISSN: 01678655. DOI: 10.1016/j.patrec.2017.12.027.

[19] Anh-Vu Vo et al. “Octree-based region growing for point cloud segmentation”. en. In: *ISPRS Journal of Photogrammetry and Remote Sensing* 104 (June 2015), pp. 88–100. ISSN: 09242716. DOI: 10.1016/j.isprsjprs.2015.01.011.

[20] Yaxu Xie et al. “PlaneRecNet: Multi-Task Learning with Cross-Task Consistency for Piece-Wise Plane Detection and Reconstruction from a Single RGB Image”. en. In: arXiv:2110.11219 (Jan. 2022). number: arXiv:2110.11219 arXiv:2110.11219 [cs]. URL: http://arxiv.org/abs/2110.11219.

[21] Michael Ying Yang and Wolfgang Forstner. “Plane Detection in Point Cloud Data”. en. In: (), p. 16.