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

List of Figures

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

introduction

1. aktueller stand: gute und schnelle ebenenfindung wird oft gebraucht, gibt es auch schon/ist möglich
2. problem: oft sind die speziellen sensoren sehr kostenspielig
3. Daher die frage: (wie gut) ist das ganze auf bezahlbarer hardware möglich?
4. Nötig, um die Frage zu beantworten:
   - Welche Kamera(s)?
   - Welcher algorithmus?
   - Was heisst "real-time" überhaupt?
5. Problem an letzterem: nicht möglich einen einheitlich besten algorithmus auszuwählen, da ...
6. Lösung: wir wählen algorithmen aus und vergleichen diese einheitlich um die frage aus 3. zu beantworten

1.2 Intel RealSense

- Wir müssen zuerst sensoren auswählen, mit denen wir die umgebung aufnehmen
- Da, wie vorher angesprochen, der preis des sensors oft ein problem ist, wählen wir eine relativ billige (im vergleich)
- die intel sensoren sind vergleichsweise bezahlbar.
- genauer gesagt nutzen wir ...
- Zu den sensoren wird eine kostenfreie software bereit gestellt
- Über diese software lassen sich die kameras ansteuern. dazu ist in dieser software ein slam algorithmus namens rtabmap implementiert
- mit rtabmap können wir den strom aus rohdaten zu einer bestehenden karte verarbeiten, was uns ermöglicht ebenen der kompletten umgebung zu finden anstatt nur von dem aktuellen blickwinkel

2 Background

In this chapter, we 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*[8] or *Dense Visual SLAM*[16] integrate additional visual input of camera sensors into their algorithm.

RTAB-MAP

RealSense-ROS internally uses a SLAM algorithm for map building, namely RTAB-MAP (Real-Time Appearance-Based Mapping)[17]. 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:

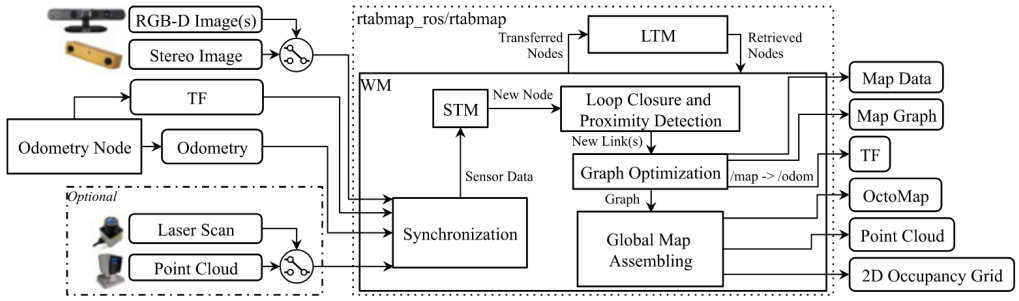


Figure 2.1: Block diagram of RTAB-MAP's main node. Taken from [17, Figure 1]

2.2 Intel Realsense

In this work, we use the Intel RealSense tracking camera T265 and the RGB-Depth(RGB-D) camera D455. A tracking camera is generally used to observe the environment and usually has a wider field of view (FOV). The primary motivation for using RGB-D cameras is depth perception. The primary differences and similarities between the T265 and the D455 are reported in Table 2.1. Beide Kameras sind stereo, die T265 hat 2 fisheye lenses und die D455 hat 2 imagers. Dazu hat die D455 noch einen RGB sensor und einen infrarot sensor. Mit dem IR sensor und den beiden imagern wird ein tiefenbild berechnet. Durch die fisheye lenses hat die T265 mit 163° ein deutlich breiteres Sichtfeld als die D455 mit nur 111°. Die maximale FPS anzahl der D455 ergibt sich aus den individuellen FPS werten der imager sensoren und dem RGB sensor, welche beide einen maximalwert von 90 haben. Dazu sei gesagt, dass bei steigender auflösung die maximale Framerate sinkt und 90FPS nur mit einer maximalen auflösung von 640x480 möglich ist. Furthermore, both cameras have an integrated Inertial Measurement Unit (IMU) which is used to compute its position in combination with visual input.

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(Robot Operating System [1]) adaptation is also provided in Intel's Github repository [2].

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

	Image	Type	max. Resolution	D-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 T265 and D455 camera specifications. More information and the complete Datasheets can be found on https://www.intelrealsense.com/.

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 [18, 2]:

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

Hough Transform

The original motivation behind the Hough transform was detecting lines in images [13]. 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[9].

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.

In the context of plane detection in 3D point clouds, a plane would be uniquely identified by the 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. Since more parameters are needed to describe a plane in 3D, the accumulator must be adapted. Therefore, a three-dimensional accumulator is used, whereas the specific shape has been discussed [6].

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[30, 34, 5]. 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[34]. The model, in that case, could be a cartesian plane equation.

Region Growing

Region Growing methods are often used in the field of image or point cloud segmentation [24, 32]. 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 values 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 [21]. 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 Robust Statistics approach for Plane Detection

Robust Statistics approach for Plane Detection (RSPD) [2] 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 *and*
- x satisfies the inlier conditions for p:
  - The distance d of x to p is smaller than a threshold θ_d (see Eq. 2.1) *and*
  - The angle ϕ between the normals vectors of x and p is less than a threshold θ_a (see Eq. 2.2).

$$d = |(x - p.center) \cdot p.normal| < \theta_d \tag{2.1}$$

$$\phi = acos(|x.normal, p.normal|) < \theta_a \tag{2.2}$$

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 *and*
- at least one inlier of P_1 satisfies the inlier conditions(see Eq. 2.1+2.2) from P_2 and vice versa.

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

2.4.2 Oriented Point Sampling

Oriented Point Sampling (OPS) [30] 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 using 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.

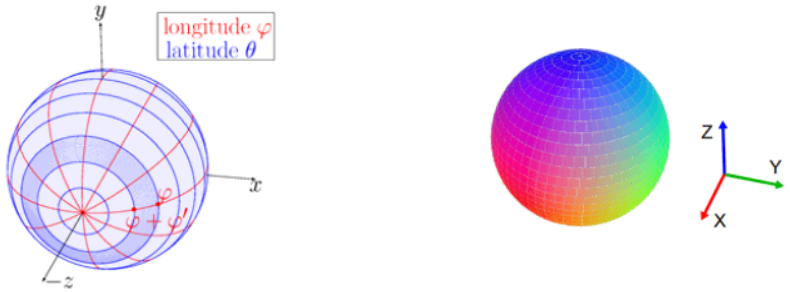
After normal estimation, one-point-RANSAC is used to find the largest plane. Usual RANSAC implementations sample three points to fit a plane. However, OPS fits a plane with only one sample point and its normal vector. Once a plane with the most inliers is obtained, its normal vector is re-estimated using 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 [18]. 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). Inspired by Borrmann et al.[6], 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 [6, Figure 3]. Accumulator ball(b) used in 3D-KHT, taken from [18, 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 [18, 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 [32]) 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.

Dataset	Input Format	Real	Indoor	GT
SegComp [15]	DI	N	/	planes
S3DIS [4]	UPC	Y	Y	objects
NYU V2 [27]	DI	Y	Y	classes
Kinect [23]	OPC	Y	Y	planes
ICL-NUIM [12]	DI	Y	Y	trajectory
SYNBEP [26]	OPC	N	/	planes
ARCO [14]	OPC	Y	Y	/
SUN [28]	DI	Y	Y	objects
Leica [1]	UPC	Y	N	planes
TUM [29]	DI	Y	Y	trajectory

Table 2.2: Popular Datasets. The GT(Ground Truth) column specifies what the ground truth of each dataset represents.

2.4.5 PEAC - Probabilistic Agglomerative Hierarchical Clustering

2.4.6 CAPE - Fast Cylinder and Plane Extraction

2.4.7 SCH-RG - Plane Extraction using Spherical Convex Hulls

2.4.8 D-KHT - Hough Transform for Real-Time Plane Detection

2.4.9 DDFF - Depth Dependent Flood Fill

2.4.10 PlaneNet

2.4.11 PlaneRecNet

2.4.12 PlaneRCNN

2.5 Datasets

Through extensive research of current literature, we compiled a list of popular datasets (see Table 2.2).

2.6 Evaluation Metrics

Wenn man sachen segmentiert oder muster erkennen möchte usw. benutzt man oft zur evaluierung metriken, die die qualität der benutzten methode beschreiben. Usual metrics are *precision, recall* and the *f1-score*. In general, *precision* describes how many of the results are relevant, i.e., the percentage of correctly calculated values(see Eq. 2.3). *Recall* describes the ratio of relevant results to all relevant data, i.e. the likelihood of a result being relevant(see Eq. 2.4). Lastly, the *f1-score* is the harmonic mean of the former two metrics (see Eq. 2.5).

$$Precision = \frac{|\{\text{correct values}\} \cap \{\text{obtained values}\}|}{|\{\text{obtained values}\}|} \tag{2.3}$$

$$Recall = \frac{|\{\text{correct values}\} \cap \{\text{obtained values}\}|}{|\{\text{correct values}\}|} \tag{2.4}$$

$$\textit{f1-score} = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall} \tag{2.5}$$

In the context of this work, we calculate *precision, recall* and the *f1-score* as follows. Required are the original point cloud PC, the corresponding list of ground truth planes GT and the planes obtained from a plane detection algorithm A. First, we regularize the PC to reduce complexity and 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 list of points that represent a plane. In the next step, we compare our planes from GT with A to obtain a list of corresponding pairs of ground truth and found planes. A ground truth plane gt_i is marked as *detected* if any plane from the list of found planes achieves a minimum voxel overlap of 50%. With this list of correspondences, we calculate *precision, recall* and the *f1-score*.

For a given ground truth plane gt_j and a corresponding detected plane 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.

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

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

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

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

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.

3 Concept

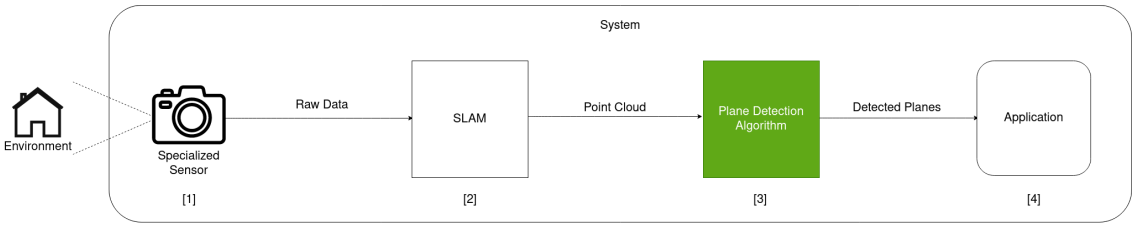


Figure 3.1: The procedure of the plane detection process. The specialized sensor records data ([1]), which is passed to a SLAM algorithm ([2]). After map assembly, a point cloud is handed to a plane detection algorithm ([3]). The detected planes are given to a use-case-specific application ([4]).

Internal Introduction Many AR and VR Systems integrate plane detection into their software, some use it only to calculate the ground floor while others use plane detection to build a smaller model of the environment. Die genauen Anwengunsmöglicheiten sind hierbei jedoch endlos. Figure 3.1 shows a generic block diagram of such a VR/AR system including plane detection. In general, the environment is continuously recorded by a specialized sensor which is usually a camera([1]). A SLAM algorithm then integrates the new data into its already existing map([2]). The map, in form of a point cloud, is subsequently passed to a plane detection algorithm([3]). The algorithm performs the necessary steps to detect all planes inside the current map and passes the planes to the application([4]). The application would then further process those planes, e.g., by creating a live visualization of them or by assisting the movement of visually impaired people [7].

When creating such a system, the choice of PDA is naturally of great importance. The problem is that most published algorithms are not inherently comparable. Often different datasets or metrics are used, which precludes comparison by quantification. Alternatively, algorithms are not comparable by internal functionality because many methods require different inputs, and the format of the planes differs accordingly. All in all, selecting a single "best" algorithm is impossible solely based on the metrics presented in their respective work.

To answer the question of which algorithm is best and whether it is real-time capable, we make a unified comparison of PDAs. To perform this evaluation, we need the following things:

1. Appropriate plane detection algorithms,
2. a useful dataset *and*
3. a definition of *real-time*.

The following sections are dedicated to them.

3.1 Selection of Plane Detection Algorithms

Since, as already noted, most algorithms differ in certain aspects, it is not possible to compare them all uniformly. Furthermore, not all algorithms have the same motivation and therefore focus on different things. For example, testing an algorithm like *Underwater SLAM* [11] for performance in small indoor environments would be pointless. It is, therefore, necessary to first define objective criteria to superficially determine which algorithm seems to be relevant for the context of this work.

3.1.1 Criteria

In the following paragraphs, we define and outline appropriate criteria for the objective assessment of plane detection algorithms.

Type of Input The first criterion is the type of input expected by a plane detection algorithm. Usually, the data representation of the recorded environment falls into one of three categories:

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

The fundamental difference between UPC and OPC is their format. Each point cloud has a *width* and a *height* parameter. An unorganized point cloud c is generally equal to an unordered 1D array of 3D coordinates, i.e., $width = |c|$ and $height = 1$. In contrast, the memory layout of an organized point cloud is a 2D array, where the width and height depend on the resolution of the used sensor. Taking the maximum resolution of the T265(see Table 2.1) as an example, the OPC would have a *width* and *height*

of 1280 and 720, respectively. Intuitively, the value at index (0,0) would be in the top-left corner, and the value at index (*width,height*) would be in the bottom-right corner.

Depth images are inherently similar to organized point clouds, given their resolution and two-dimensional structure. The primary difference is that the values stored in the array are distances to the sensor instead of 3D coordinates.

Consequently, since unorganized point clouds are not limited in their dimension, they are more suitable for capturing entire environments.

As stated before, we focus on detecting planar structures in the entire environment rather than just distinct segments. 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 Some methods are learning-based, e.g., they use deep learning to detect planes in point clouds or images. Their ability to generalize depends on the choice of training data. If an algorithm is perfectly trained on a dataset that consists of only tables, the algorithm would find all table tops but might fail to detect planes that do not have a certain number of legs attached to them. todo:elaborate

3.1.2 Plane Detection Algorithms

A list of state-of-the-art algorithms is compiled through comprehensive research of the current literature on plane detection (see Table 3.1). explanation of table

Im folgenden werden aus den zuvor aufgestellten kriterien die für diese arbeit sinnvollsten werte(?"ich nehme UPC aus UPC, OPC, DI... idk wie ich das nennen soll) ausgewählt und anhand dessen unpassende algorithmen von der evaluierung ausgeschlossen.

Plane Detection Algorithm	Input Data	Plane Format	Learning-Based
RSPD [2]	UPC	inliers	N
OPS [30]	UPC	inliers	N
3DKHT [18]	UPC	inliers	N
OBRG [32]	UPC	inliers	N
PEAC [10]	OPC	inliers	N
CAPE [24]	OPC	normal, d	N
SCH-RG [22]	OPC	inliers	N
D-KHT [31]	DI	inliers	N
DDFF [25]	DI	inliers	N
PlaneNet [19]	I	normal, d	Y
PlaneRecNet [33]	I	reconstructed scene	Y
PlaneRCNN [20]	I	normal, d	Y

Table 3.1: Plane Detection Algorithms

Addressing the criterion of input type, we are only interested in performing plane detection in complete environments. We hereby consider organized point clouds or images inappropriate because they do not offer a complete view on a scene. We hereby exclude *PEAC, CAPE, SCH-RG, D-KHT, DDFF, PlaneNet, PlaneRecNet* and *PlaneRCNN* from our evaluation.

Secondly, the detected planes need to be in the same format because, even for the same plane, different representations could very well lead to different results. Assume a plane in cartesian form and a plane represented by its inliers. The calculated metrics may differ significantly because the plane in cartesian form is infinitely dense. In contrast, the plane described by its inliers allows for holes and non-rectangular shapes, e.g., doorways or a round table. We thereby determine *inliers* as the preferred plane format and exclude all methods which do not comply, namely *CAPE, PlaneNet, PlaneRecNet*, and *PlaneRCNN*.

Finally, we end up with, and thus include, the following plane detection algorithms in our evaluation:

- **RSPD**
- **OPS**
- **3D-KHT**
- **OBRG**

3.2 Datasets

@marco Sollte hier dann nicht der anfang der evaluation hin? also bzgl warum wir das aufteilen in temporal und nicht temporal? dann wäre aber halt die intro der eval quasi leer

3.2.1 S3DIS Experiment

This subsection deals with the selection of the dataset without the temporal component.

In Section 3.1, we determine unorganized point clouds (UPC) as input for the selected algorithms. Furthermore, this work focuses on real, indoor environments. Additionally, a ground truth is necessary for the evaluation. We select the Stanford Large-Scale Indoor Spaces 3D Dataset(S3DIS)[3] from the list of datasets to evaluate each plane detection algorithm on even grounds. Since our focus is not on 3D semantic segmentation and the provided ground truth is focused on semantic segmentation rather than planes, we manually select planar regions using CloudCompare[1]. 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 3.2. An individual scene has a complete unstructured point cloud and a list of annotated files representing semantically different objects that can be found therein.

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 dataset nevertheless reflects a realistic distribution of scene types since it is not realistic if a building contains only lecture halls. Inversely, it is appropriate to assume that an office complex contains a substantial amount of hallways needed to connect all offices.

[1] https://cloudcompare.org/

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 3.2: S3DIS Disjoint Space Statistics

3.2.2 FIN Experiment

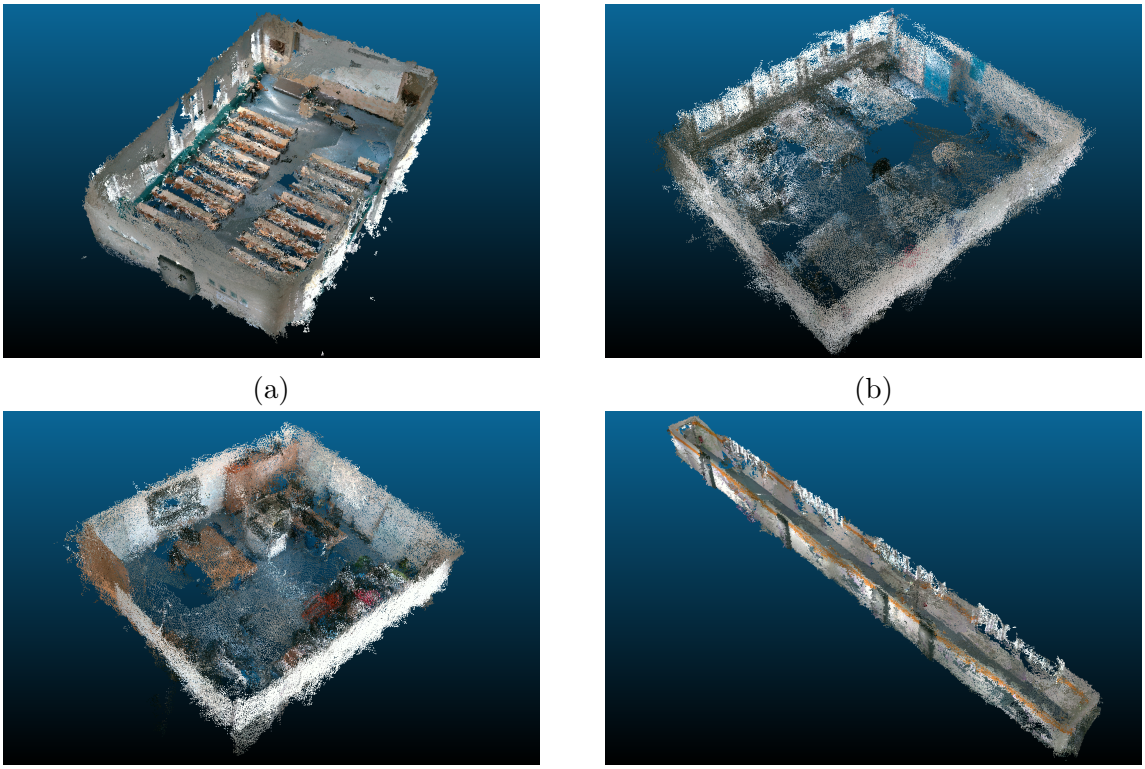
We record an incrementally growing dataset in the Faculty of Computer Science at Otto-von-Guericke University Magdeburg. To perform a thorough comparison between the FIN and S3DIS, we record a scene for each of the following scene types:

- office
- conference room
- auditorium
- hallway

We focus on these four scene types because they are the most common in a real environment.

The recorded point clouds can be seen in Figure 3.2.

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



(a) (b) (c) (d)

Figure 3.2: The recordings for each scene type: (a)auditorium, (b) conference room, (c) office and (d) hallway.

Since no ground truth exists for a novel dataset like this, 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. 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 3.3. 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.

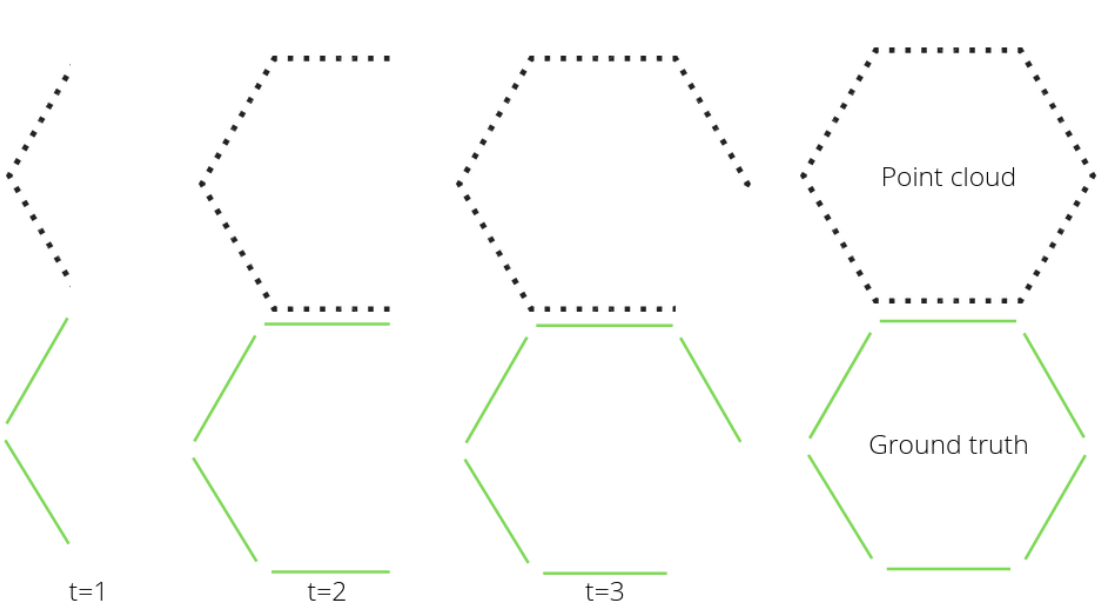


Figure 3.3: Dynamic ground truth generation. All planes that are included in *Ground Truth* are cropped depending on the available point cloud at each time t

3.3 Definition 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.

3.4 Summary

Many applications have constraints in the form of a temporal component. Augmented or Virtual Reality applications that include plane detection are no exception. In addition to time constraints, good quality is usually tightly coupled to expensive sensors. To evaluate to what extent it is possible to perform precise plane detection with a real-time constraint on off the shelf hardware, 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[1] and OPS[2] using their respective open source implementations on GitHub. 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/

4.2.1 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 [32, 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 [32, 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 Ground Truth Segmentation

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. Because 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 4.1), 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.

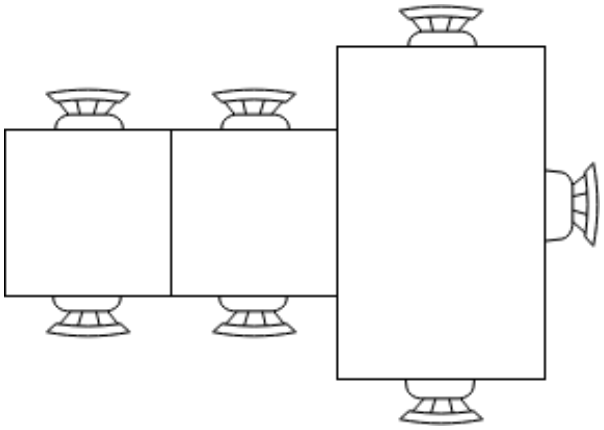


Figure 4.1: The given ground truth considers these tables to be three distinct objects. Within the context of plane detection, the three table tops would form exactly one plane.

5 Evaluation

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

5.1 Protocol

- wie im konzept gesagt vergleichen wir algorithmen um $FRAGE zu beantworten
- zunächst definieren wir, wie und woran wir die performanz der algos vergleichen
- da die algos verschiedene datensätze nutzen sind sie nicht direkt vergleichbar
- daher führen wir einheitliche vergleiche durch
- es gibt keinen passenden dynamischen datensatz (inkr. aufbau + punktwolke+ GT) daher machen wir zwei experimente:
  - statisch: direkt ganze wolke da
  - dynamisch: inkrementell aufbauend
- schlussendlich werden wir die resultate beider experimente vergleichen und hinsichtlich der $FRAGE auswerten

Because most publications of the selected algorithms use a different dataset during evaluation, they are not objectively comparable. Furthermore, to the best of our knowledge, there is no dataset, 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(static dataset). 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(dynamic dataset), 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 [2, Section 4].

5.2 Results

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

5.2.1 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 5.1: Average results of each algorithm over the S3DIS dataset.

Every algorithm performs calculations on every scene included in S3DIS. The results of each algorithm on an individual scene type are reported in Figure 5.1 and Figure 5.2. As shown in Table 5.1, 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 3.3, 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 5.1).

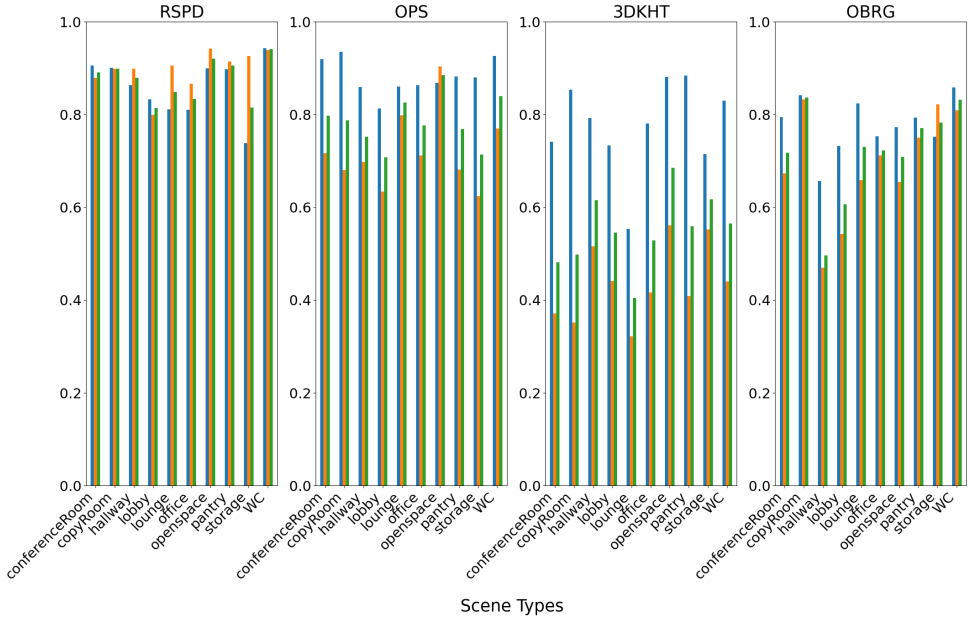


Figure 5.1: Average Accuracy for each scene type. The Precision is colored blue, recall is orange and the F1-score is green.

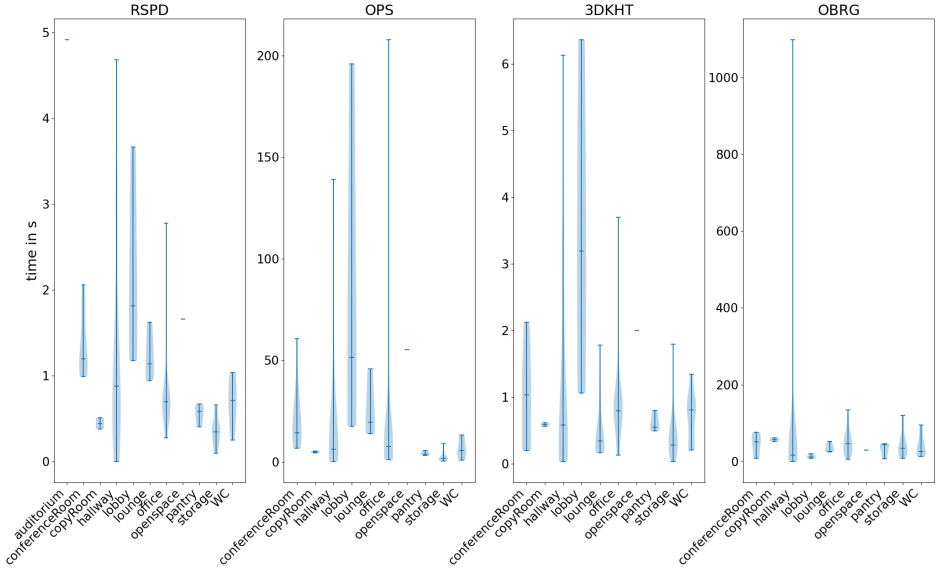


Figure 5.2: Average Time per scene type. Note, that the plots do not share the same y-axis.

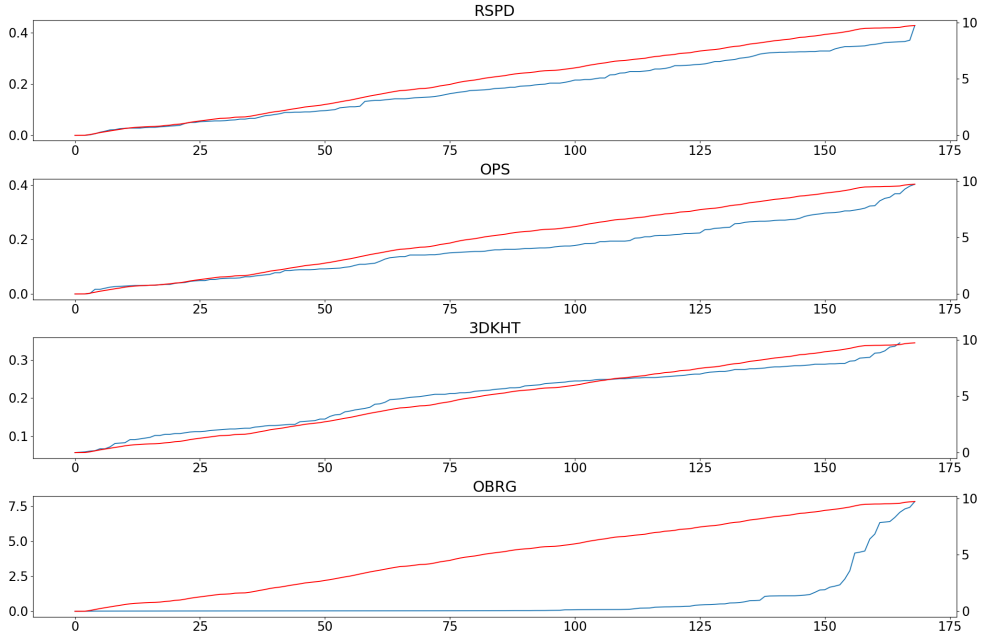


Figure 5.3: Calculation times(blue) of the hallway scene and cloud size(red) of each time step.

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

5.2.3 Summary Results

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.

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

Bibliography

[1] URL: https://shop.leica-geosystems.com/de/leica-blk/blk360/dataset-downloads.

[2] 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.

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

[4] 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.

[5] 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.

[6] 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.

[7] Aparicio Carranza et al. "Plane Detection Based Object Recognition for Augmented Reality". en. In: May 2021. DOI: 10.11159/cdsr21.305. URL: https://avestia.com/CDSR2021_Proceedings/files/paper/CDSR_305.pdf.

[8] 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.

[9] Richard O. Duda and Peter E. Hart. "Use of the Hough Transformation to Detect Lines and Curves in Pictures". In: *Commun. ACM* 15.1 (Jan. 1972), pp. 11–15. ISSN: 0001-0782. DOI: 10.1145/361237.361242. URL: https://doi.org/10.1145/361237.361242.

[10] 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/.

[11] Felipe Guth et al. “Underwater SLAM: Challenges, state of the art, algorithms and a new biologically-inspired approach”. In: *5th IEEE RAS/EMBS International Conference on Biomedical Robotics and Biomechatronics*. 2014, pp. 981–986. DOI: 10.1109/BIOROB.2014.6913908.

[12] A. Handa et al. “A Benchmark for RGB-D Visual Odometry, 3D Reconstruction and SLAM”. In: *IEEE Intl. Conf. on Robotics and Automation, ICRA*. Hong Kong, China, May 2014.

[13] Peter E. Hart. “How the Hough transform was invented [DSP History]”. In: *IEEE Signal Processing Magazine* 26.6 (2009), pp. 18–22. DOI: 10.1109/MSP.2009.934181.

[14] Alejandro Hidalgo-Paniagua et al. “A Comparative Study of Parallel RANSAC Implementations in 3D Space”. en. In: *International Journal of Parallel Programming* 43.5 (Oct. 2015), pp. 703–720. ISSN: 0885-7458, 1573-7640. DOI: 10.1007/s10766-014-0316-7.

[15] Adam Hoover et al. “An Experimental Comparison of Range Image Segmentation Algorithms”. In: *IEEE Trans. Pattern Anal. Mach. Intell.* 18 (July 1996), pp. 673–689. DOI: 10.1109/34.506791.

[16] 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/.

[17] 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.

[18] 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.

[19] 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/.

[20] 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/.

[21] 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/.

[22] 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/.

[23] Bastian Oehler et al. “Efficient Multi-resolution Plane Segmentation of 3D Point Clouds”. en. In: *Intelligent Robotics and Applications*. Ed. by Sabina Jeschke, Honghai Liu, and Daniel Schilberg. Vol. 7102. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 145–156. ISBN: 978-3-642-25488-8. DOI: 10.1007/978-3-642-25489-5_15. URL: http://link.springer.com/10.1007/978-3-642-25489-5_15.

[24] 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.

[25] 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/.

[26] Alexander Schaefer et al. “A Maximum Likelihood Approach to Extract Finite Planes from 3-D Laser Scans”. In: *2019 IEEE/RSJ International Conference on Robotics and Automation (ICRA)*. IEEE. May 2019. URL: http://ais.informatik.uni-freiburg.de/publications/papers/schaefer19icra.pdf.

[27] Nathan Silberman et al. “Indoor Segmentation and Support Inference from RGBD Images”. In: *Computer Vision – ECCV 2012*. Ed. by Andrew Fitzgibbon et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 746–760. ISBN: 978-3-642-33715-4.

[28] Shuran Song, Samuel P. Lichtenberg, and Jianxiong Xiao. "SUN RGB-D: A RGB-D scene understanding benchmark suite". In: *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2015, pp. 567–576. DOI: 10.1109/CVPR.2015.7298655.

[29] J. Sturm et al. "A Benchmark for the Evaluation of RGB-D SLAM Systems". In: *Proc. of the International Conference on Intelligent Robot Systems (IROS)*. Oct. 2012.

[30] 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.

[31] 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.

[32] 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.

[33] 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.

[34] Michael Ying Yang and Wolfgang Forstner. "Plane Detection in Point Cloud Data". en. In: *Proceedings of the 2nd int conf on machine control guidance* 1 (Feb. 2010), p. 16.

Declaration of Academic Integrity

I hereby declare that I have written the present work myself and did not use any sources or tools other than the ones indicated.

Datum: .....

(Signature)