OTTO VON GUERICKE UNIVERSITÄT MAGDEBURG

INF

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

2 Background

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

2.1 Intel Realsense

https://www.intelrealsense.com/compare-depth-cameras/

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. Over decades of research on this particular problem, many different methodologies have been developed, usually regarding the sensors used to aid the robot's navigation. Since odometry sensors like rotary encoders that measure the rotation of the robot's wheels are unreliable when it comes to uneven or slippery ground, visual sensors also needed to be involved.

RTAB-MAP RealSense-ROS internally uses a SLAM algorithm for map building, namely RTAB-MAP (Real-Time Appearance-Based Mapping)[5]. 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, the results of which are passed to RTAB-MAP's *Short-Term-Memory* (STM). Das STM baut aus den neuen inputs einen neuen knoten, welcher in den map graphen eingefügt wird. Basierend auf dem hinzufügen des neuen knotens wird die loop closure berechnet. sollte eine loop closure detected werden wird der graph optimiert und minimiert. Dazu wird die globale map in abhängigkeit der neuen kenntnisse reassembled. Das resultat davon wird als punktwolke veröffentlicht.

Er funktioniert grundsätzlich so:

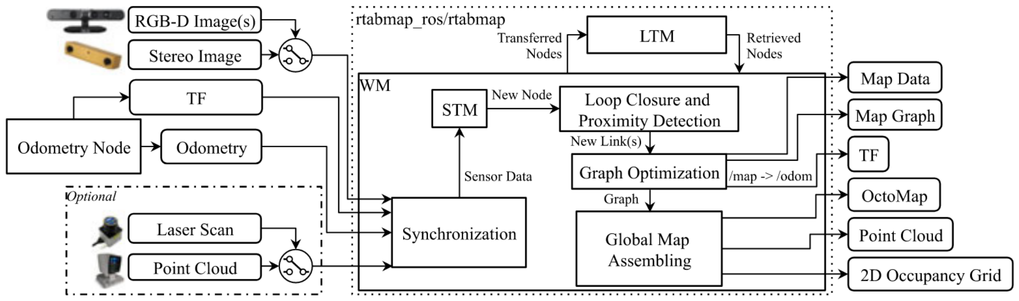


Figure 2.1: Block diagram, gestolen ausm rtabmap paper [5, Figure 1]

2.2 Plane Detection

introduction The field of plane detection has been around for decades. Most methods of detecting planar regions can be sorted into one of three main categories:

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

Hough Transform

HT wurde ursprünglich introduced um lines in bildern zu finden. das ganze wurde aber inzwischen erweitert auf weitere formen im zwei und sogar drei dimensionalen raum. Möchte man in 2d punktdaten eine gerade detektieren, legt man durch jeden datenpunkt für jede orientierung eine gerade in hesse normal form. Dabei werden stimmen für die winkel abgegeben, welche ähnliche distanzen zum ursprung erzeugen. Am ende hat die best fitting grade den winkel, für den die meisten punkte gestimmt haben. Die datenstruktur, in dem die Stimmen gesammelt werden, wird auch accumulator genannt.

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[13, 17, 3]. 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. In the context of this work, the model parameters would be some plane equation or a combination of normal vectors and center points.

Region Growing

Region Growing methods are often used in the field of image or point cloud segmentation [11, 15]. RG-based segmentation methods aim to obtain a set of disjoint regions by initially selecting a set of seed points and incrementally inserting neighboring points based on an inclusion criterion. Therein, the quality of results depends on the choice of seed points, e.g., a very noisy seed point could decrease overall quality [9]. 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.

2.3 Plane Detection Algorithms

In dieser section werden die algorithmen näher erläutert, die im späteren kontext dieser arbeit im fokus stehen werden.

2.3.1 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 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 by becoming 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.

2.3.2 OPS

OPS (Oriented Point Sampling for plane detection [13]) 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.

2.3.3 3D-KHT

Limberger and Oliveira propose a hough transform-based plane detection method, which accepts unorganized point clouds as input [6]. 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 terms of creation and manipulation. Furthermore, the octree succeeds in capturing the shapes inside the point cloud, while the k-d tree does not.

Using the octree, the point cloud is subdivided until either 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.

The approximately coplanar nodes are refined by removing outliers and subsequently fitting a plane through the remaining points.

In dieser Phase werden zuerst gaussische trivariate kernel berechnet. Dabei wird die Ebene in spherical coordinates umgeschrieben. hier noch den rest einfügen!

2.3.4 OBRG

OBRG (Octree-Based Region Growing [15]) 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 superseded 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 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.

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 (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

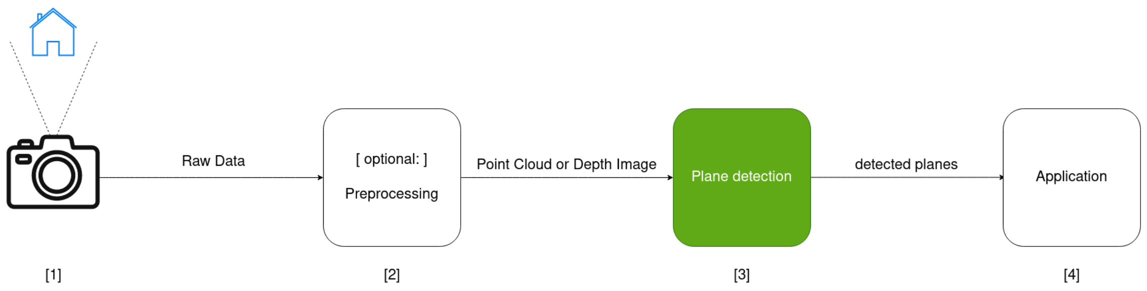


Figure 3.1: A camera records an environment. The recorded data is either preprocessed, or passed directly to the plane detection algorithm. The resulting planes are subsequently exported to an arbitrary application.

Within a given use case or scenario, the user records the surroundings with a camera, and the raw data is 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 3-D 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 car drives through an urban environment, the delay must be as short as possible to avoid a collision with a wall the car 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 will focus 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.

3.0.1 Used Sensors

To efficiently tackle the previously defined problem, and according to figure 3.1, phase 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.

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, siehe concept figure, step 2. We integrate *realsense-ros*, the ROS wrapper of Intel RealSense, into our process of plane detection.

Realsense-ros internally uses a SLAM(Simultaneous Mapping and Localization) algorithm called RTAB-MAP [5] 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.

Mit intel realsense und rtabmap lässt sich Figure 3.1 konkreter beschreiben, siehe Figure 3.2

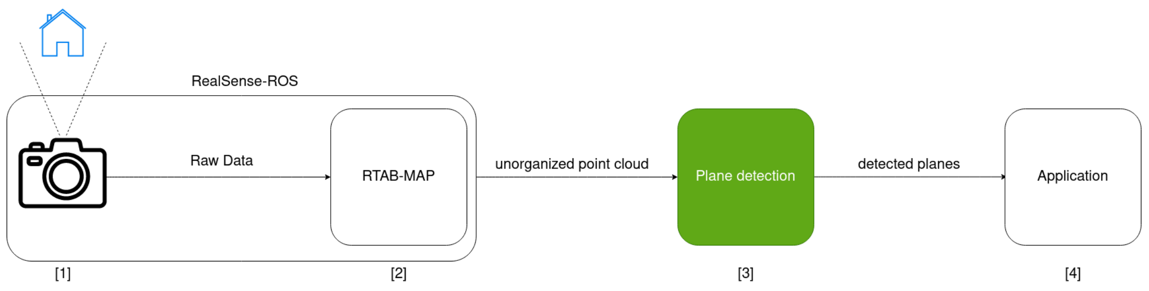


Figure 3.2: Updated process with sensors and what not

3.1 Selection Plane Detection Algorithms

Naturally, appropriate algorithms are needed to perform real-time plane detection, as depicted in Figure 3.1, step 3. 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.

3.2 Algorithms?

RSPD - Robust Statistics Approach for Plane Detection

OPS - Oriented Point Sampling

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 a data set, the results thereof can be seen in Figure 3.3

Because the leftmost two presets, with an octree subdivision value of 1 and 2, respectively, seem to yield similar results, but a higher value of subdivision might result in better performance in larger point clouds, we will perform all calculations of 3D-KHT with an octree subdivision value of 2.

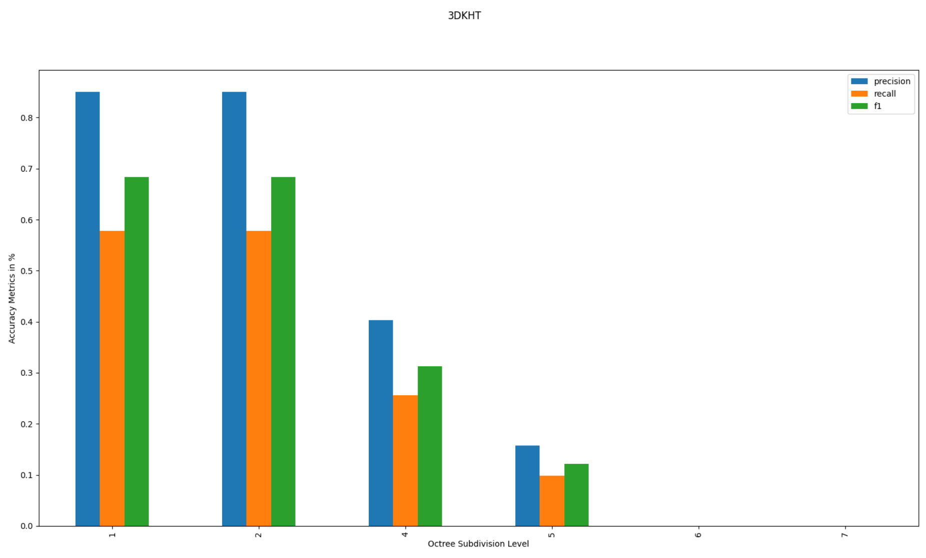


Figure 3.3: Results of 3D-KHT for different pre-set subdivision values

	Input Data	**Plane format**	**Learning-Based**	**Availability**
RSPD [1]	UPC	inliers	N	Y
OPS [13]	UPC	inliers	N	Y
3DKHT [6]	UPC	inliers	N	Y
OBRG [15]	UPC	inliers	N	N
PEAC [4]	OPC	inliers	N	Y
CAPE [11]	OPC	normal, d	N	Y
SCH-RG [10]	OPC	inliers?	N	N
D-KHT [14]	DI	inliers	N	Y
DDFF [12]	DI	indices	N	Y
PlaneNet [7]	I	normal, d	Y	Y
PLaneRecNet [16]	I	? / -	Y	Y
PlaneRCNN [8]	I	normal + ?	N	Y

Table 3.1: Plane Detection Algorithms

OBRG - Octree-based Region Growing

PEAC - Probabilistic Agglomerative Hierarchical Clustering

3.2.1 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 3.1.

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

3.3 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. 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.

Wir schreiben den algorithmus in python. Wir hatten in betracht gezogen, vorgefertigte octree implementierungen zb von open3d zu benutzen, aber diese haben uns im endeffekt nicht genug freiraum geboten. Daher implementieren wir unseren eigenen octree für die räumliche aufteilung der punktwolke. Wir teilen, wie in 2.3.4 beschrieben, einen octree knoten so lange in 8 gleichgroße kinder auf, bis ein predefined maximum erreicht ist oder die anzahl der included punkte kleiner als ein schwellwert ist. Um die saliency features der leaf nodes zu berechnen, speziell für die normal vektoren, benutzen wir die open3d normal estimation. wir folgen dem pseudocode wie angegeben, mit der ausnahme, dass wir das hinzufügen in eine region angepasst haben. Da das wort "insert" raum für interpretation lässt, haben wir eine zweite liste geführt, die angibt, ob ein leaf schon benutzt wurde. Wenn ja, wird die region gesucht, bei dem das leaf schon benutzt wurde, anstatt eine neue region zu erstellen. Dieser Schritt kompliziert zwar das region growing ein wenig, reduziert jedoch die totale laufzeit des algorithmus, da keine redundante regionen mehr vorkommen können. Da die residual threshold, angular threshold und dmin des papers nicht genauer spezifiziert wurden, waren wir gezwungen empirisch zu bestimmen, welche werte gut funktionieren:

- res th = 0.1
- ang difference = 0.3
- d min = 0.1

in deren paper wurde auch ein planarity level von 0.7 -0.9 vorgeschlagen, wir nehmen einfach pauschal 0.8. oder wir führen tests analog zu 3dkht durch, kann auch sein.

zu guter letzt mussten wir uns das general refinement übelst aus dem ärmel leiern, scheint aber größtenteils zu funzen. GR betrifft sowieso scheinbar nur die stellen, wo zwei ebenen aufeinander treffen würden, bzw winkel.

Uns ist bewusst dass python eher weniger echtzeitfähig ist, aber die optimierung eines spezifischen algorithmuses steht nciht im vordergrund dieser arbeit und würde dementsprechend zu future work gehören.

4.3 Dataset / Ground Truth

Da der benutzte Datensatz den fokus auf semantische segmentierung legt und nicht auf ebenen findung, legen wir unsere eigene ground truth an. Dafür benutzen wir cloudcompare. Da wir ausserdem nicht pauschal davon ausgehen können, dass ”wände” immer planar sind, oder dass drei tische immer verschiedene ebene sind (also zusammenstehen, eine ebene formend, aber dennoch verschiedene tische sind) sind wir gezwungen jede punktwolke manuell anzusehen und zu editieren.

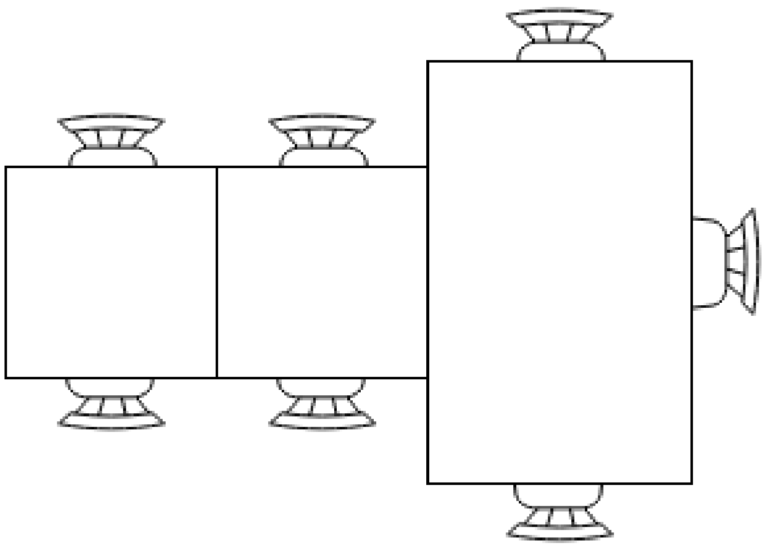


Figure 4.1: Three tables = three planes?

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.

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

5.1.2 Dataset

To evaluate each plane detection algorithm on even grounds, we select the Stanford Large-Scale Indoor Spaces 3D Dataset(S3DIS)[2]. The Dataset had been recorded in three different buildings, which then were divided into six distinct areas. Those six areas include a total of 270 different types of rooms, e.g. offices, hallways, WCs and two auditoriums to name a few.
Each room has a complete unstructured point cloud in form of a list of XYZ values, as well as a list of annotated files representing semantically different objects that can be found in this point cloud of the room. Since our focus lies not on 3D semantic segmentation, we manually select planar regions using CloudCompare[1], obtaining a list of sub-clouds.

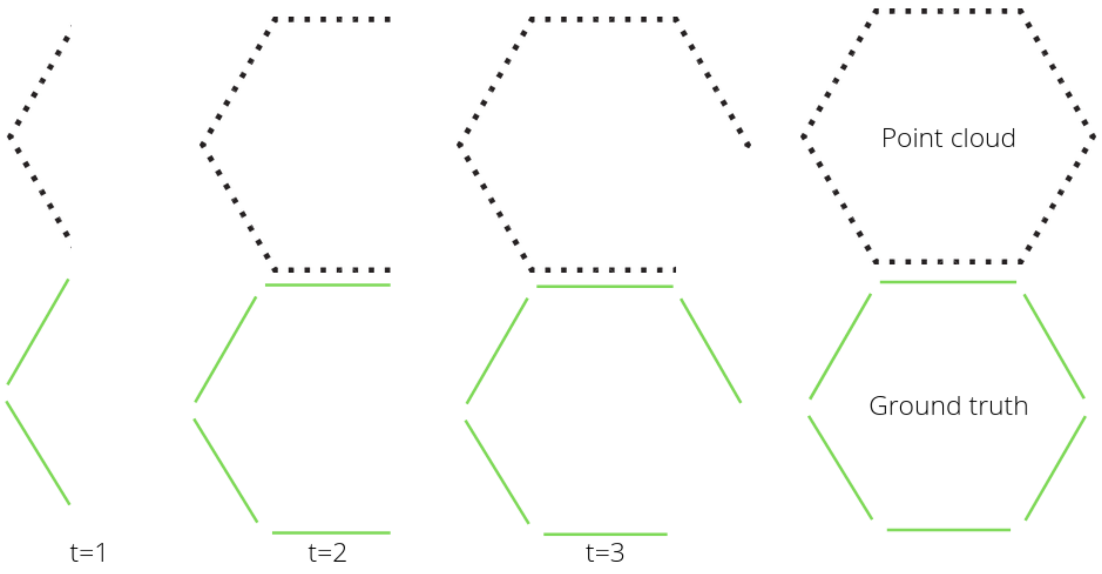
5.1.3 Real-Life Test

We record an incrementally growing data set in the Faculty of Computer Science at Otto-von-Guericke University Magdeburg. 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.

We create a set of ground truth planes $gt_{end}$for only the most recent update, e.g., for the entire recording.

[1]https://cloudcompare.org/

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. 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.



5.2 Results

5.2.1 Results Dataset

Area 1 Hier sind die Ergebnisse von Area 1:

Area 2 Hier sind die Ergebnisse von Area 2:

Area 3 Hier sind die Ergebnisse von Area 3:

Area 4 Hier sind die Ergebnisse von Area 4:

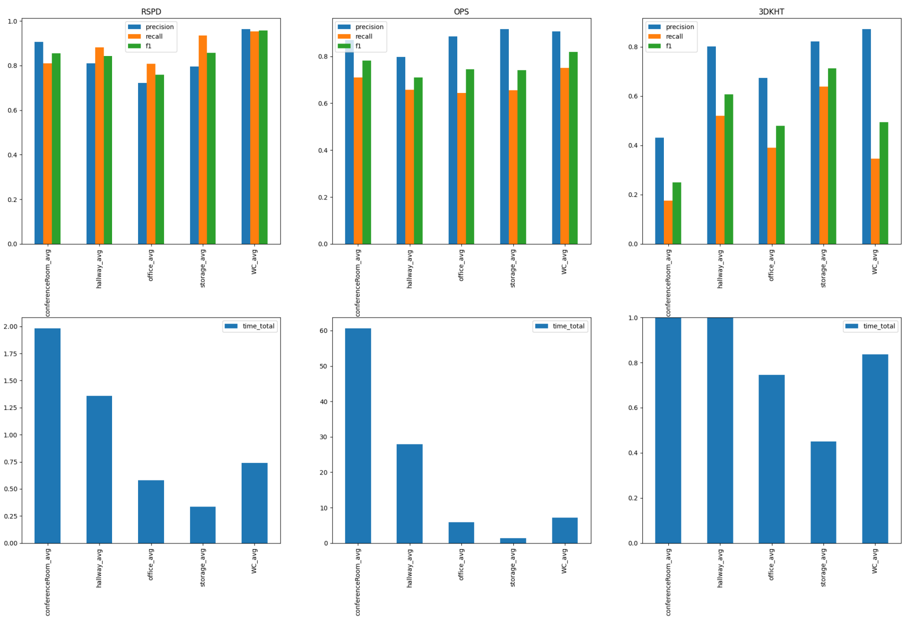


Figure 5.1: Results Area 1

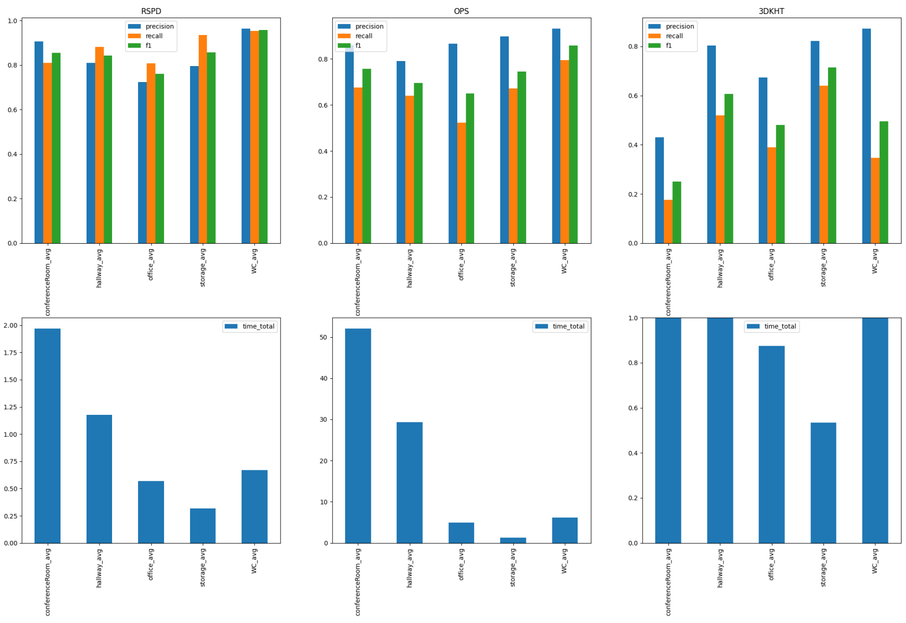


Figure 5.2: Results Area 2

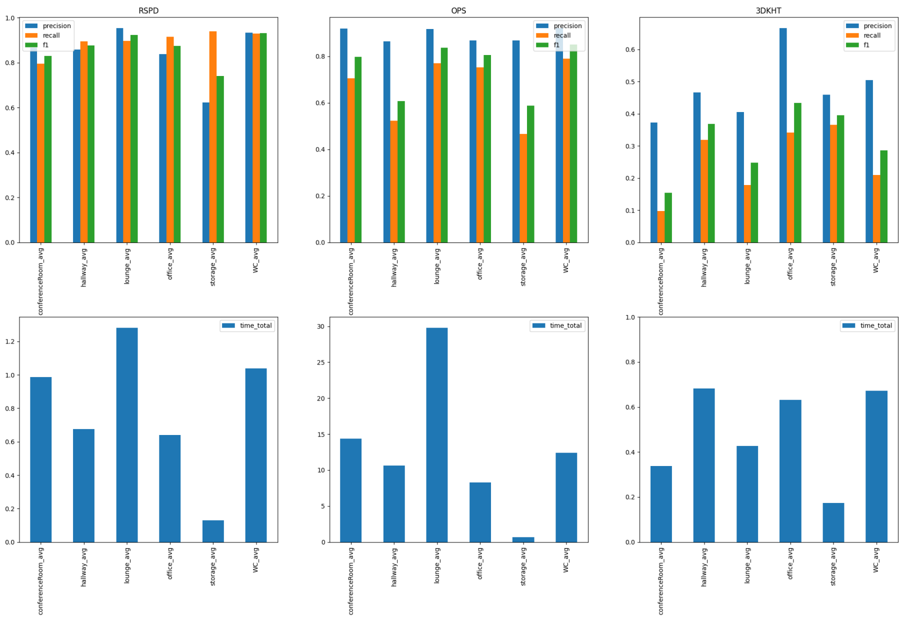


Figure 5.3: Results Area 3

5.2.2 Results Real-Life Test

Hier sind die Ergebnisse:

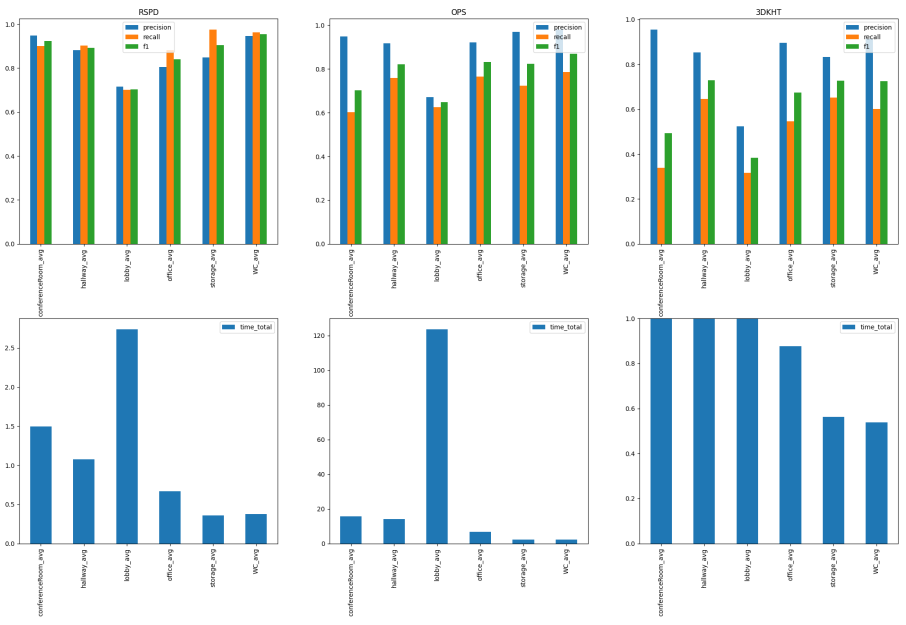


Figure 5.4: Results Area 4

6 Conclusion

Possibly redundant with results section in evaluation

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] 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.

[4] 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/.

[5] 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.

[6] 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.

[7] 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/.

[8] 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/.

[9] 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/`.

[10] 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/`.

[11] 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`.

[12] 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/`.

[13] 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`.

[14] 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`.

[15] 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`.

[16] 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`.

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