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, oder Simultaneous Localization And Mapping, ist ein Problem aus der Robotik und befasst sich damit, wie ein unbemannter roboter eine karte der umgebung aufbaut und gleichzeitig die eigene position in relation dazu korrekt erfasst.

RTAB-MAP rtabmap ist der intern benutze slam algorithmus von realsense-ros. Er unterscheidet sich insbesondere hierdurch von HIERANDERENSLAMEINFÜGEN.

Er funktioniert grundsätzlich so:

2.2 Plane Detection

introduction plane detection algorithmen lassen sich im groben in 3 hauptkategorien einordnen:

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

Hough Transform

RANSAC

RANSAC ist kurz für RAndom SAmple Consensus und die grundlegende idee dahinter ist das iterative schätzen eines mathematischen modells. Im kontext dieser arbeit würde dementsprechend beispielsweise eine Ebenengleichung geschätzt werden. Es gibt viele Varianten von RAnsac, alle haben jedoch gemeinsam, dass eine gewisse anzahl an punkten (mehr oder weniger) zufällig aus dem datensatz gewählt werden und versucht wird durch diese das jeweilige modell zu bilden. Yang and Forstner[15] samplen in jeder iteration 3 random punkte, Sun and Mordohai [11] wählen nur einen zufälligen punkt zu einem zuvor berechneten normalenvektor um eine Ebene zu bilden. Dazu gibt es in den meisten fällen ein Kriterium, wie oft das ganze durchgeführt werden soll.

Region Growing

Regio Growing findet oft benutzung in der image segmentation, kann jedoch auch im 3d raum zum finden von ebenen benutzt werden. Grundsätzlich werden beim RG zunächst eine Auswahl an Seed points getroffen. Im anschluss werden von jedem seed point aus benachbarte Daten betrachtet, und zu der region des seed points hinzugefügt, wenn bestimmte voraussetzungen erfüllt sind. im kontext dieser arbeit würden diese vorraussetzungen beispielsweise einen fokus auf coplanarität legen, d.h. die normalenvektoren des seed points und des nachbarn unterscheiden sich weniger als ein vordefinierter Schwellwert.

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 gehört zu den region growing verfahren und besteht im grunde aus 3 Phasen; Teilen, Wachsen, Zusammenfügen.

Split Für die spatial subdivision wird ein octree aus der ungeordneten punktwolke konstruiert. Dieser unterteilt sich immer weiter in kleinere sub-trees, bis die beinhalteten punkte des jeweiligen sub-trees einen Grenzwert unterschreiten.

Im Anschluss wird eine planarity test durchgeführt, bis ein sub-tree einen planar patch enthält. Dieser Sub-tree schreitet voran in die nächste Phase (grow).

Grow Für die wachstumsphase wird ein nachbarschaftsgraph (NG) über die gesamte wolke erstellt, insofern dass ein knoten des graphen einen punkt innerhalb der wolke repräsentiert. Knoten des nachbarschaftsgraphen werden verbunden, wenn die dazugehörigen punkte durch eine knn suche gefunden werden (die authoren benutzen k=50).

Nach der NG konstruktion wird von einem planaren patch P_i aus eine breitensuche gemacht und jeder punkt s, der die folgende kriterien erfüllen, wird zu P_i hinzugefügt:

- s gehört bisher zu keinem patch
- die abweichung von s zu der ebene P_i ist kleiner als die definierten Schwellwerte

Nun kann s jedoch zu mehreren patches gehören, wodurch sich das problem ergibt, zu welcher ebene s hinzugefügt werden sollte. Um dem vorzubeugen, werden die patches anhand deren normalen abweichung sortiert, sodass zuerst die patches mit dem geringsten noise anteil zuerst wachsen.

Merge Die patches aus den vorherigen phasen müssen noch gemergt werden. Betrachtet man zwei planare patches P_1 und P_2, gibt es essentiell drei konditionen, unter welchen diese beiden verbunden werden dürfen:

- die Octree nodes von P_1 und P_2 müssen benachbart sein
- $P_1.n$ und $P_2.n$ sollten in ähnliche richtungen zeigen
- min. ein inlier aus P_1 sollte die inlier condition von P_2 erfüllen und vice versa

Schlussendlich liefert die Merge phase alle maximal zusammengefügten patches zurück.

2.3.2 OPS

Oriented point Sampling akzeptiert eine unorganized point cloud als input. Zuerst wird aus der gesamtwolke ein kleiner anteil random gewählt. Die normalvektoren dieser punkte werden mit SVD geschätzt, nachdem die k nächsten nachbarn mithilfe eines kd-trees gesucht wurden. Dazu wird eine inverse-distance gewichtsfunktion benutzt, sodass nahegelegene punkte einen höheren stellwert haben als weiter weg gelegene.

Anschliessend wird per 1P RANSAC die ebene mit den meisten inliern gefunden. Im vergleich zu traditioneller Ransac ebenenfindung braucht OPS nicht 3 punkte, sondern nur einen punkt und den normalenvektor. Zuletzt wird der normalenvektor der gefundenen ebene neu geschätzt, dafür wird erneut die SVD auf alle inlier angewandt. Wurde erfolgreich eine Ebene gefunden, werden die beinhalteten punkte aus der gesamtwolke entfernt. Dieser prozess wird so lange wiederholt, bis die gesamtanzahl der punkte einen Schwellwert θ_N unterschreitet.

Die resultierenden gefundenen ebenen werden über ihren Mittelpunkt, sowie den normalenvektor dargestellt.

2.3.3 3D-KHT

Der 3D-Kernel Hough transform gehört natürlich zu den Hough transform verfahren. Auch dieser Algorithmus teilt die punktwolke zunächst rekursiv in kleinere teile auf mithilfe eines octrees. Das aufteilen der Octree nodes wird erst gestoppt, wenn die enthaltenen samples approximately coplanar sind, oder aber die anzahl der enthaltenen samples, s_{ms}, einen Schwellwert unterschreitet. Die authoren empfehlen für große wolken einen minimum von $s_{ms} = 30$ [5]. Ist eine octree node approx. coplanar, werden zuerst samples entfernt, wessen distanz zur ebene größer als $\frac{octree-node-length}{10}$ ist, und im Anschluss eine ebene gefittet.

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 maybe OBRG

OBRG gehört natürlich ebenfalls zu den region growing verfahren.

Die unorganized point cloud wird zunächst mit einem octree rekursiv in kleinere fragmente zerteilt. eine octree node wird so lange zerteilt, bis für eine octree node n eines der zwei folgenden kriterien erfüllt ist;

- node level überschreitet max subdivision level
- number of included samples unterschreitet min included schwellwert

Anschliessend werden saliency features berechnet. Für jede leaf node wird die normale, sowie ein residual wert berechnet. Die normale wird mit PCA der included samples berechnet. Mithilfe der Normale und dem center der samples wird eine fitting plane

definert. die residual value der leaf node ist das RMS aller abweichungen der punkte von der fitting plane.

Im nächsten schritt werden die leaf nodes geclustert. die Liste der blattknoten wird nach deren residual werten sortiert. Betrachte man einen beliebigen blattknoten l, so werden zunächst alle 26 benachbarten B blattknoten von l in betracht gezogen. ein benachbarter blattknoten $b_i \in B$ wird der zu der Region von l hinzugefügt, wenn beide folgenden kriterien erfüllt sind:

- b_i gehört noch zu keiner region
- angular divergence von $b_i.n$ und $l.n \leq \theta_{ang}$

Weiterhin wird die aktuelle Region für weitere berechnungen berücksichtigt, wenn die anzahl der beinhalteten punkte einen Grenzwert M überschreitet.

Im folgenden werden die Regionen ihrer Größe nach sortiert und im nächsten schritt verfeinert. Die verfeinerung wird unterteilt in zwei Arten; fast(FR) und general(GR). Welche der Arten auf eine Region angewandt werden, entscheidet sich danach, ob die Region planar($\rightarrow FR$) ist oder nicht ($\rightarrow GR$). Eine Region wird als planar angesehen, wenn eine mindestanzahl an samples mit einer toleranz in die best fitting plane der Region passen. Die authoren geben an, dass sich diese mindestanzahl zwischen 70% und 90% befinden sollte, je nach anteil an noise im datensatz. Für beide verfeinerungen werden lediglich die blattknoten berücksichtigt, welche die jeweilige region eingrenzen, also weniger als 8 nachbarn haben. Sie unterscheiden sich jedoch darin, dass beim fast refinement auf leaf basis verfeinert wird, beim general refinement aber auf punktbasis.

In beiden fällen erhält man vollständige regionen, aus denen problemlos die best fitting plane, als auch die included samples extrahiert werden können.

3 Concept

This chapter deals with the realization of analysis. We introduce the definition of real-time and the use case with respect to this work. Plane detection algorithms are selected for comparison, as well as the metrics they will be judged upon.

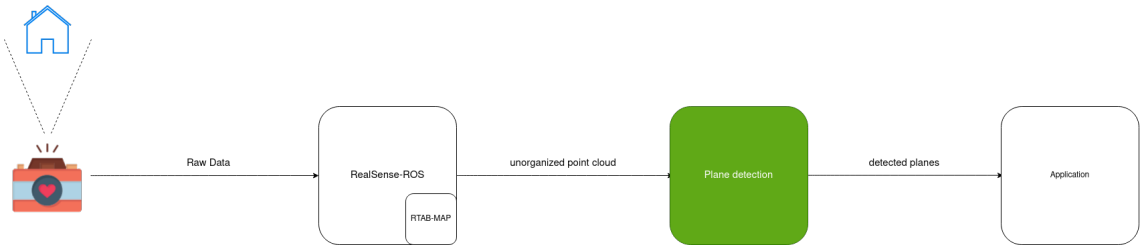


Figure 3.1: A camera records an environment and passes the raw data to RealSense-ROS. After RTAB-MAP updates the map, it is given to the plane detection algorithms (green). The resulting planes will then be exported as needed, given the specific use case, described in Chapter 1.

3.1 Scenario / UseCase

Taking motivation from chapter 1, we focus on indoor environments during this work. 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.

3.1.1 Used Sensors

To be able to perform plane detection, we need special hardware that is able to accurately record the surroundings. Numerous different cameras suffice for this task, each one of course varying in different aspects. For this work, we use the Intel RealSense T256 Tracking Camera, as well as the Intel RealSense D455 RGB-D Camera. The reason for this is the compatibility of the two cameras since the T256 can be used in combination with any depth camera from the D400 series.

In addition to the cameras, the Intel RealSense software provides a wide variety of usage. We are especially interested in the detection of planes in complete environments. Therefore, it is necessary to be able to build a map out of the data the cameras record continuously. This is why we integrate *realsense-ros*, the ROS wrapper of Intel RealSense, into our plane detection. Realsense-ros internally uses a SLAM(Simultaneous Mapping and Localization) algorithm called RTAB-MAP [4] 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, especially since we don't focus on SLAM algorithms in this work.

3.2 Selection Plane Detection Algorithms

First, we need to assert the comparability between the algorithms introduced in 2. We report necessary criteria, both to shorten the list of algorithms, as well as verify comparability.

Type of Input

Popular representations, which the recorded environment can take the form of, can be grouped into three main categories of input:

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

As stated before, we focus on the detection of planar structures in the entirety of a scene, rather than just singular segments thereof. In addition, only the unorganized/unstructured point clouds offer a complete view of the recorded environment.
For that reason, we disregard all algorithms which do not expect an unorganized point cloud as input.

Detected Plane Format

Which specific representation the detected planes take the form of is also important. If no uniform type of output can be determined, consequently no uniform metric for comparison can be found as well. 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. Having 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 useful for any use case involving remodeling architectural elements. It also enables 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 from any experiments.

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 octree has been divided at s_{level} times.
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 3.2

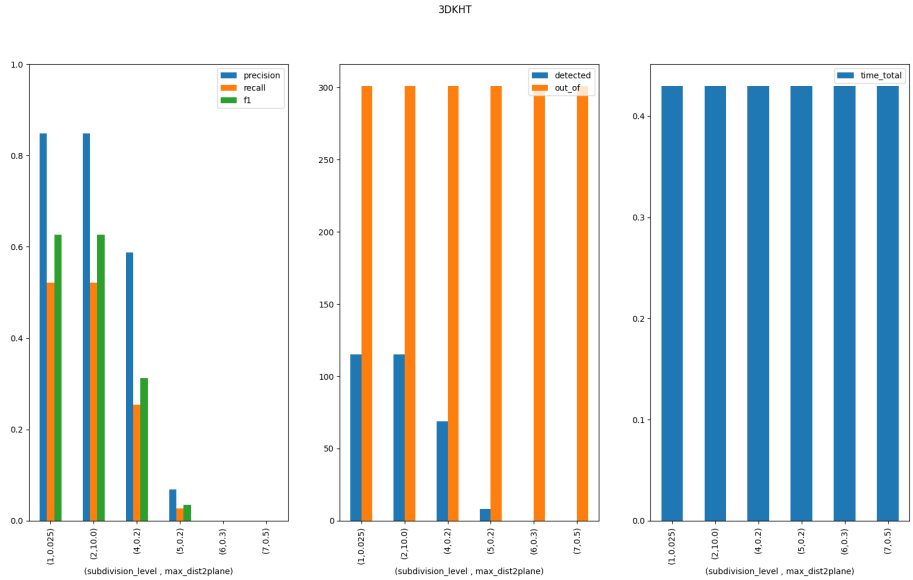


Figure 3.2: Test results of 3D-KHT with different parameters

Because the leftmost two presets, with an s_{level} value of 1 and 2, respectively, seem to yield equal results, but a higher value of subdivision might result in better performance in larger environments, we henceforth perform all calculations of 3D-KHT with an s_{level} value of 2.

	Input Data	Plane format	Learning-Based	Availability
RSPD [1]	UPC	inliers	N	Y
OPS [11]	UPC	inliers	N	Y
3DKHT [5]	UPC	inliers	N	Y
OBRG [13]	UPC	surely inliers	N	N
PEAC [3]	OPC	inliers	N	Y
CAPE [9]	OPC	normal, d	N	Y
SCH-RG [8]	OPC	inliers?	N	N
D-KHT [12]	DI	inliers	N	Y
DDFF [10]	DI	indices	N	Y
PlaneNet [6]	I	piecewise planar sth	Y	Y
PLaneRecNet [14]	I	? / -	Y	Y
PlaneRCNN [7]	I	normal + ?	N	Y

Table 3.1: Plane Detection Algorithms

OBRG - Octree-based Region Growing

PEAC - Probabilistic Agglomerative Hierarchical Clustering

Summary

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.

If we furthermore consider algorithms that run on anything other than UPC, it would necessitate finding a data set that includes equivalent point clouds for the structured, as well as 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.

For the reasons previously stated in 3.2, we also exclude learning-based methods.

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 How do we verify "real-time"?

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

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 recordings are 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 in $1s$.

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 method in python, heavily relying on someLib for computation of something.

4.3 Dataset / Ground Truth

um eine ground trutht in form von einer liste von ebenen zu jedem datensatz zu bekommen, segmentieren wir manuell die input punktwolken des datensatzes mithilfe von cloudcompare.

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 pointcloud.
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.

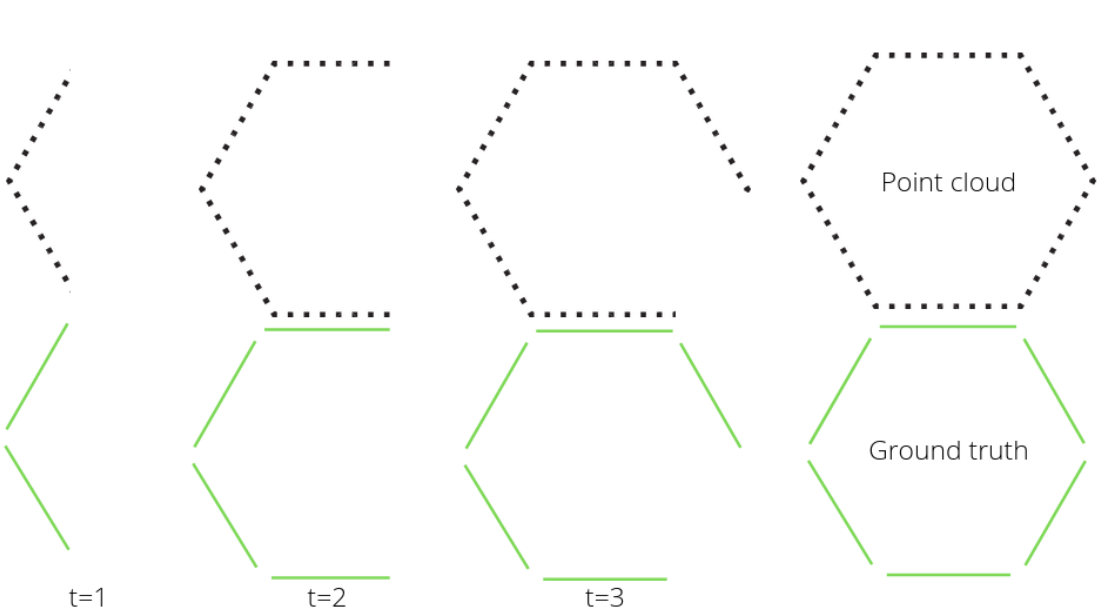
5.1.3 Real-Life Test

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

Hier sind die Ergebnisse:

5.2.2 Results Real-Life Test

Hier sind die Ergebnisse:

6 Conclusion

Possibly redundant with results section in evaluation

7 References

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] 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/.

[4] 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.

[5] 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.

[6] 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/.

[7] 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/.

[8] 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/.

[9] 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.

[10] 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/.

[11] 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.

[12] 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.

[13] 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.

[14] 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.

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