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

"Summary", What is to be expected in this chapter Propably a bad idea to have started with this before i finished reviewing the literature. In this chapter we will present relevant literature needed to completely understand the proposed concept of chapter 3.

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.

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, wether its 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, of course varying in different aspects. In this work, we use Intel RealSense technology[1], namely the T256 Tracking Camera and the D455 RGB-D Camera. probably more detail on the camera right? oder eher in den background

In addition to the cameras, there is software that provdes a wide variety of usage. We are especially interested in detection of planes in complete environments. For that reason, we use the ROS [2] wrapper of Intels Realsense software, *realsense-ros* [3]

Realsense-ros internally uses a SLAM(Simultaneous Mapping and Localization) algorithm called RTAB-MAP [2] 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

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

included in the realsense package and its reported performance suffices for this work, especially since we dont focus on SLAM algorithms in this work. Furthermore noteworthy is the fact, that different algorithms eventually return different (kinds of) maps, which likely lead to different results in comparison to ours.

3.2 Selection Plane Detection Algorithms

First we need to assert the comparability between the algorithms introduced in 2. We report necesary criteria to both 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*
- *organized* or *structured point cloud*
- *(depth-) image*

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 first type of input offers a persistent view on the recorded environment.
For that reason, we disregard all algorithms which do not expect an unorganized point cloud as input.

Output Format of Detected Planes

Which specific representation the output takes 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.

Determinism

ND methoden, zb DL basierende haben grundsätzlich ein gewisses level an bias, welcher stark von der Wahl der Trainingsdaten abhängt. Ein weiterer grund gegen die Benutzung von Learning-basierenden methoden ist, dass wir nicht von einer (rechenstarken) GPU ausgehen können.

ransac ? OPS nutzt halt ransac

Paremeters, additional necessitites

Ebenfalls wichtig ist, ob ein algorithmus zusätzlich zum input noch weiteres wissen benötigt. 3DKHT ist stark von dem Level der octree subdivision abhängig, welches (ohne weitere Änderungen) predefined ist.

Availability

Zuletzt werden wir die verfügbarkeit als schwaches kriterium festlegen. Grundsätzlich muss ein algorithmus auf dem selben System, wie die anderen auch implementiert sein, damit die ausführende/underlying hardware als Faktor ausgeklammert werden kann. Aus dem Grund können algorithmen, welche wenig bis gar nicht beschrieben werden, nicht in diesen Vergleich aufgenommen werden.

RSPD - Robust Statistics Approach for Plane Detection

OPS - Oriented Point Sampling

3DKHT - 3-D Kernel-based Hough Transform

der bums braucht halt fixe parameter vielleicht kann ich den dynamisch-ish machen in abhängigkeit von der dimension der input wolke

OBRG - Octree-based Region Growing

PEAC - Probabilistic Agglomerative Hierarchical Clustering

Summary

hier ist ne tabelle:

	Input Data	Determinism	Availability	Output Format
RSPD	unordered PC	Deterministic	open source	inliers
OPS	unordered PC	mostly Deterministic	open source	normal, center, orientation
3DKHT	unordered PC	mostly Deterministic	open source	visual
OBRG	unordered PC	Deterministic?	not available	
PEAC	ordered PC	Deterministic	avalable	segmented cloud

Table 3.1: Selected 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 equal.

If we furthermore took PEAC into consideration, it would necessitate finding a data set which includes equivalent point clouds for the structured, as well as the unstructured case. Alternatively, we could transform an unordered point cloud into a series of ordered sub-clouds. Since this approach would disregard the global structure of the point cloud, we deem it not feasable for our use-case.

something about determinism of OPS(ransac) and hough transforms in general

How do we verify "real-time"?

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

Real-Time To be able to precisely define real-time, one has to consider possible hardware limitations, data flow, and simply how often it is needed to perform calculations in correspondence with the given use-case.

Gar nicht mehr unbedingt nötig, wenn die obere grenze eh der SLAM ist, oder? The D455 has a depth frame rate of up to 90, while the T256 only achieves a maximum frame rate of 30.

The recordings are not directly sent to the plane detection algorithm, but instead given to realsense-ros' internal SLAM algorithm, RTAB-MAP, which then performs calculations to update the map. Therefore, the upper limit is the frequency of how often RTAB-MAP updates the map, which by default is once per second. According to this upper limit, we declare an algorithm *real-time applicable*, if this algorithm achieves an average frame rate of minimum 1, e.g. the algorithm manages to process the entire point cloud and detect all planes in $1s$.

needed?
We can reduce complexity further by taking the specifications[4] of the D455 into account. The RMS error of the D455 is reported to be 2% at 4 meters distance to the sensor. Furthermore, the ideal distance is stated to range between $0.6 - 6$ meters. To maintain a dense and precise representation of our environment, we therefore limit the detection of planes to a radius of 6 meters from the current position.

Determine Real-Time Applicability 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 their 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.

Subsequently, we conduct experiments, this time including the temporal component, running and evaluating calculations on each distinct frame of time.

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.

Daher führen wir zunächst Experimente mit fokus auf Genauigkeit durch, dh wir klammern die zeitliche Komponente des Datensatzes aus.

[4] https://www.intelrealsense.com/compare-depth-cameras/

Anders gesagt, wir werden Experimente auf einem Datensatz durchführen, wobei genau einmal die komplette Punktwolke der Szene an den algorithmus gegeben wird. Anschliessend werden wir einen dynamischen Test durchführen, indem sich die Szene inkrementell aufbaut.

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 Hardware

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 [1], which runs Linux Ubuntu 20.04.5.
The laptop has the following specifications.

- CPU : AMD Ryzen 7 5800H @ 4.4GHz
- GPU : AMD RX Vega 8
- RAM : 16G

4.2 Intel RealSense

As stated in 3, we use the open source implementation of Intel RealSense.

4.3 Plane Detection Algorithms

RSPD

An open source implementation, written in C++, is provided by the authors Araújo and Oliveira[1] and can be found on GitHub [2]. We closely follow the instructions regarding

[1] Lenovo IdeaPad 5 Pro
[2] `https://github.com/abnerrjo/PlaneDetection`

the build process as described in the provided README.

OPS

The authors of *Oriented Point Sampling*, namely Sun and Mordohai, do not provide their original implementation. However, a C++ implementation can also be found, again on GitHub [3]

3D-KHT

Limberger and Oliveira, the authors of 3D-KHT, provide an implementation, in form of a Visual Studio project, of their algorithm on their website [4]. Since the laptop we use does not run Windows, and therefore cannot build the project, we use a package called *cmake-converter*[5] to convert the solution to a cmake project we can build using *make*.

OBRG

To our knowledge, no open-source implementation is available for the algorithm, which was introduced by Vo et al.[5]. We therefore use our own implementation.

sicherlich haben wir das geschafft. Weil C++ fucky wucky ist, haben wir das in python implementiert. Dazu haben wir diese hübschen libs genutzt...

[3] `https://github.com/victor-amblard/OrientedPointSampling`
[4] `https://www.inf.ufrgs.br/~oliveira/pubs_files/HT3D/HT3D_page.html`
[5] https://cmakeconverter.readthedocs.io/

5 Evaluation

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

5.1 Evaluation Protocol

The goal of this work is to determine wether or not real-time, precise plane detection is realistic. When selection an algorithm for said task a problem presents itself; Most algorithms had been tested on different, non-comparable conditions. We therefore select a dataset on which all plane detection algorithms will be tested on to achieve comparability. For the evaluation protocol itself, we took inspiration from the *Results* Section of Araújo and Oliveiras work on RSPD [1].

5.1.1 Metrics

To quantitatively evaluate each algorithms performance, we calculate the precision, recall and the f1 score. For that, we took inspiration from the work of [1] and their approach of evaluation. 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 voxelgrid 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 sort a given voxel v_i into the categories *True Positive(TP), False Positive(FP), False Negative(FN), True Negative(TN)* as follows.

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

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

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, e.g. RSPD needs to calculate normals for each sample, if normal vectors are not included in the dataset. To measure the detection time, we simply log the exact times before and after calculations and write the differences to a file.
a variation would be interesting probably right? like time per plane, time per sample etc

5.1.2 Dataset

To evaluate each plane detection algorithm on even grounds, we select the Stanford Large-Scale Indoor Spaces 3D Dataset(S3DIS). 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.

insert statistic here

[1] https://cloudcompare.org/

5.1.3 Real-Life Test

Bei dem Live test wird im Gebäude der FIN ein Datensatz aufgenommen. Der Scan wird $STUFF beinhalten. Zu diesem Scan gibt es keine Ground Truth, daher wird neben der Laufzeitanalyse lediglich eine qualitative Analyse der Präzision vorgenommen. Dafür überlagern wir die Punktwolke mit den gefundenen Ebenen.

5.2 Results

5.2.1 Results Dataset

Alle N Sequenzen des Datensatzs wurde X mal von jedem Algorithmus berechnet.

Hier sind die Ergebnisse:

5.2.2 Results Real-Life Test

Der FIN Datensatz wurde auch X mal von jedem Datensatz berechnet.

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. URL: https://github.com/abnerrjo/PlaneDetection.

[2] 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.

[3] 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.

[4] 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.

[5] 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.