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 covers our procedure 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, 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 [1] wrapper of Intels Realsense software, *realsense-ros* [2]

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

[1] ROS-Robot Operating System
[2] 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.

3.1.2 Real-Time

To precisely define real-time, we need to consider the hardware restrictions of the Intel RealSense cameras as well as the fps restrictions of RTAB-MAP. RTAB-MAP publishes a new update of the current map with a default frame rate of 1Hz. Thus, the plane detection algorithm needs to process a pointcloud in a similar frequency, taking up to an entire second.

We can reduce complexity further by taking the depth accuracy of the D455 into account. The RMS error of the D455 is reported to be 2% at 4 meters distance to the sensor, with a linear progression. To maintain a dense and precise representation of our environment, we limit the detection of planes to a radius of 4 meters from the current position.

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.

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

3.2.1 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.

3.2.2 RSPD - Robust Statistics Approach for Plane Detection

3.2.3 OPS - Oriented Point Sampling

3.2.4 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

3.2.5 OBRG - Octree-based Region Growing

3.2.6 PEAC - Probabilistic Agglomerative Hierarchical Clustering

hier ist ne tabelle:

	Input Data	Determinism	Parameter Dependency	Availability
RSPD	unordered PC	Deterministic	independent	open source
OPS	unordered PC	mostly Deterministic	independent	open source
3DKHT	unordered PC	mostly Deterministic	very dependent	open source
OBRG	unordered PC	Deterministic?	?	not available
PEAC	ordered PC	Deterministic	?	avalable

Table 3.1: Selected Plane Detection Algorithms

Fazit

wir vergleichen RSPD, OPS und 3D-KHT. depending on my success in implementing OBRG myself we will add OBRG to the comparison. We exclude PEAC beacuse it excpects an ordered point cloud as input.

4 Implementation

What this chapter will cover In this chapter we will cover the realization of the aforementioned concept in chapter 3.

4.1 Used Sensors

There exists a broad range of sensors applicable for SLAM algorithms. Depending on the specific algorithm, Lidar [2], monocular [6] stereo [8] or even a combination of multiple Cameras can be integrated [3]. Different types of cameras ultimately lead to different kinds of input. Lidar, for example, returns dense Point Clouds, whereas a RGB-D camera would return colorful images. Of course, cameras are not the only sensors used in SLAM algorithms. Many systems make use of an inertial measurement Unit (IMU) [5, 7]

For this work we will be using the Intel RealSense T256[1] tracking camera as well as the Intel RealSense D455[2] depth camera. Not only do both have a built-in IMU, they also both have two imagers, which classifies them as stereo cameras. Another advantage of combining a fish-eye tracking camera (T256) with a RGB-D camera (D455) is that they support each other in situations a robot with only one of them would be unable to handle well.

4.2 Architecture

ROS

librealsense

We integrate RTAB-MAP into our architecture like this:...

[1] https://www.intelrealsense.com/tracking-camera-t265/
[2] https://www.intelrealsense.com/depth-camera-d455/

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.

[2] David Droeschel and Sven Behnke. "Efficient Continuous-Time SLAM for 3D Lidar-Based Online Mapping". In: *2018 IEEE International Conference on Robotics and Automation (ICRA)*. 2018, pp. 5000–5007. DOI: 10.1109/ICRA.2018.8461000.

[3] Adam Harmat, Inna Sharf, and Michael Trentini. "Parallel Tracking and Mapping with Multiple Cameras on an Unmanned Aerial Vehicle". In: *Intelligent Robotics and Applications*. Ed. by Chun-Yi Su, Subhash Rakheja, and Honghai Liu. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 421–432. ISBN: 978-3-642-33509-9.

[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] Stefan Leutenegger et al. "Keyframe-Based Visual-Inertial SLAM using Nonlinear Optimization". en. In: *Robotics: Science and Systems IX*. Robotics: Science and Systems Foundation, June 2013. ISBN: 978-981-07-3937-9. DOI: 10.15607/RSS.2013.IX.037. URL: http://www.roboticsproceedings.org/rss09/p37.pdf.

[6] Raul Mur-Artal, J. M. M. Montiel, and Juan D. Tardos. "ORB-SLAM: A Versatile and Accurate Monocular SLAM System". In: *IEEE Transactions on Robotics* 31.5 (Oct. 2015), pp. 1147–1163. ISSN: 1941-0468. DOI: 10.1109/TRO.2015.2463671.

[7] Raúl Mur-Artal and Juan D. Tardós. "ORB-SLAM2: An Open-Source SLAM System for Monocular, Stereo, and RGB-D Cameras". In: *IEEE Transactions on Robotics* 33.5 (Oct. 2017), pp. 1255–1262. ISSN: 1941-0468. DOI: 10.1109/TRO.2017.2705103.

[8] Vladyslav Usenko et al. "Direct visual-inertial odometry with stereo cameras". en. In: *2016 IEEE International Conference on Robotics and Automation (ICRA)*. Stockholm: IEEE, May 2016, pp. 1885–1892. ISBN: 978-1-4673-8026-3. DOI: 10.1109/ICRA.2016.7487335. URL: http://ieeexplore.ieee.org/document/7487335/.