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

SLAM ALgos includes table of different ones

PDAs includes table of different ones

DataSets includes table of different ones

3 Concept

Introduction

What this chapter is about This chapter covers...

Chapter Structure It is Structured as follows ...

3.1 Used Sensors

Why these ones

open source ros wrapper realsense-ros

Detailed Information i guess?

3.2 SLAM

Why SLAM is needed

map building reduces complexity for plane detection

SLAM SOTA

RTAB-MAP, ORBSLAM etc..

bc we are using visual, stereo etc.

We choose RTAB-MAP bc.. just makes sense because its already included in realsense-ros and also has a wide variety of options regarding output formats

3.3 Requirements

Problem/Use case We want to find planes quickly..

Definition Real-time

Because camera runs at 30fps .. Daher definieren wir in dieser arbeit "echtzeit" als

Precision

What does precision mean in general
What is a "good" precision Werden wir $später definieren fokus wird auf echtzeit gelegt wird und wir gucken was wir im rahmen der echtzeit im besten fall an präz raus holen können dazu sind allein die angegeben werte nicht 100%ig aussagekräftig weil größtenteils verschiedene datensätze benutzt wurden somit werden wir in dieser arbeit einen einheitlichen vergleich von PDAs unter besonderer berücksichtigung des echtzeitkriteriums machen

3.4 Plane Detection Algorithms

Intro?

SOTA aufgrund von $SACHEN filtert sich die gesamte liste runter auf diese enge auswahl $DARUM werden wir den fokus auf diese N algorithmen legen

3.5 Summary

we want to answer the question if real time plane detection is viable on OTS hardware like the intel realsense. to answer the q, we selected $SLAM which gives us $OUTPUT which we use as input for out $PDAs

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 [5] stereo [7] 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) [4, 6]

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.

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

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

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] 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.

[5] 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.

[6] 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.

[7] 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/.