



Evaluation of stand-of-the-art monocular visual simultanious and mapping approaches

Masterarbeit

zur Erlangung des akademischen Grades
Master of Science in Engineering (M.Sc.)

Eingereicht bei:

Fachhochschule Kufstein Tirol Bildungs GmbH
Data Science & Intelligent Analytics

Verfasser/in:

Julian Bialas, BSc

1910837917

Erstgutachter : Prof. (FH) PD Dr. Mario Döller
Zweitgutachter : Robert Kathrein, MSc

Abgabedatum:

06. July 2020

Eidesstattliche Erklärung

Ich erkläre hiermit, dass ich die vorliegende Masterarbeit selbstständig und ohne fremde Hilfe verfasst und in der Bearbeitung und Abfassung keine anderen als die angegebenen Quellen oder Hilfsmittel benutzt sowie wörtliche und sinngemäße Zitate als solche gekennzeichnet habe. Die vorliegende Masterarbeit wurde noch nicht anderweitig für Prüfungszwecke vorgelegt.

Kufstein, 06. July 2020

Julian Bialas, BSc

Sperrvermerk

Ich habe die Sperrung meiner Masterarbeit beantragt, welche von der Studiengangsleitung genehmigt wurde.

Kufstein, 06. July 2020

Julian Bialas, BSc

Contents

1	Introduction	1
1.1	Related Work	2
2	Methods	3
2.1	vSLAM Algorithms	3
2.1.1	Definitions	3
2.1.2	ORB-SLAM	4
2.1.3	DSM-SLAM	8
2.1.4	DSO-SLAM	8
2.2	Calculations	8
2.2.1	Trajectory Alignment	8
2.3	Datasets	10
2.3.1	EuRoC Dataset	10
2.4	Setup and Environment	11
2.4.1	Evaluation	11

2.4.2 Flight Path Planning	11
3 Results	12
3.1 Trajectory Evaluation	12
3.2 Pointcloud Evaluation	16
3.3 Calculation Time	16
4 Discussion	18
4.1 Conclusion of SLAM-Algorithm Evaluation	18
4.2 Framework for an entire automated System	18
4.2.1 Current setup	20

List of Figures

1	Overview of the system components extracted from [3]	4
2	Pointcloud ground truth of sequence V1_01_easy visualized with python package pptk	10
3	Ground truth flight path and evaluated flight path of each algorithm after alignment with the method of Umeyama in the x and y axis in meters. Left the sequence MH01, middle the sequence V102 and right the sequence V203 is displayed.	13
4	Boxplot of all euclidean distances between the ground truth position of the keyframe and the evaluated position after alignment with the method of Umeyama. Outliers greater than 1.5 are not displayed.	14
5	The positional error over time in meters. The vertical lines indicate the beginning of a new sequence	14
6	The groundtruth of the Pointcloud from Sequence V101 (white points) and the evaluated points by each algorithm (red points). The points in Figure (a) are four times as large for better visibility (ORB-SLAM generates only few points).	15

7	Boxplot of the euclidean distances between an evaluated point and the closest point of the ground truth point cloud. For computational feasibility, for each sequence and algorithm, 500 points for evaluation are sampled randomly	16
8	Influence of downsizing of the images on the trajectory error (a) and the computation time (b) for the sequence V101.	19
9	tum simulator setup. Source: http://wiki.ros.org/tum_simulator	20
10	Automated SLAM system	21
11	The drone in a gazebo simulation in a), the output of the front camera of the drone in b) and the ORB-SLAM algorithm applied on the front camera output in with the detected ORB features marked green c).	22

List of Tables

List of Listings

1	drone navigation command	19
2	Launching the simulated environment	21

List of Acronyms

HTML HyperText Markup Language

JS JavaScript

FH Kufstein Tirol

Data Science & Intelligent Analytics

Abstract of the thesis: **Evaluation of stand-of-the-art monocular visual simultaneous and mapping approaches**

Author: Julian Bialas, BSc

First reviewer: Prof. (FH) PD Dr. Mario Döller

Second reviewer: Robert Kathrein, MSc

06. July 2020

FH Kufstein Tirol

Data Science & Intelligent Analytics

Kurzfassung der Masterarbeit: **Evaluation of stand-of-the-art monocular visual simultaneous and mapping approaches**

Verfasser: Julian Bialas, BSc

Erstgutachter: Prof. (FH) PD Dr. Mario Döller

Zweitgutachter: Robert Kathrein, MSc

06. July 2020

1. Introduction

Multiple applications exist for the autonomous exploration and mapping tasks for drones; such as search and rescue-, inspection- and surveillance operations [?]. The focus in this paper is on checking the feasabilty to perform an exploration and mapping task with a drone. An approach with a combination of a vSLAM (visual simulataneous localization and mapping) algorithm and a path planning algorithm is assessed. As the name suggests, the aim of a SLAM algorithm is to create a map of its completely unknown environment, while localizing itself within it only be using certain sensors. This paper is limited to monocular vSLAM, meaning that the algorithm is only working with a single RGB camera as sensor. This makes the drone very affordable, making it highly available for a larger user group. The main part of this work is the evaluation of vSLAM algorithms in order to find the most suitable method, that meets the requirements for this autonomous navigation task. DSO (Direct Sparse Odometry) SLAM, DSM (Direct Sparce Mapping) SLAM and ORB (Oriented FAST and Rotated BRIEF) SLAM were investigated regarding accuracy of the resulting trajectory estimation and pointcloud and computational speed. Furthermore, a ROS (Roboter Operating System) setup to combine the suitable vSLAM algorithm with a flight path planning algorithm is introduced. This implementation includes a simulation of a drone equiped with a RGB camera within a simulated environment, making it easy to apply flight path planning algorithms without any hardware. Finally a recommendation regarding the flight path algorithm is given for future work.

1.1 Related Work

While some papers exist, evaluating SLAM methods, most papers restrict on the trajectory. For applications involving autonomous exploration it is also crucial, that these algorithms deliver a satisfactory dense and accurate pointcloud. The generated pointclouds have barely been studied.

2. Methods

2.1 vSLAM Algorithms

2.1.1 Definitions

Covisibility Graph

updating the edges resulting from the shared map points with other keyframes.

Keyframe

Most SLAM Algorithms make usage of so called keyframes, as these keyframe-based approaches have proven to be more accurate [5].

Direct and Feature based methods

Group of Rigid Transformations in 3D

$SE(3)$ is the group of rigid transformations in 3D space [2]. Each Matrix $T \in \mathbb{R}^{4x4}$ with

$$T = \begin{bmatrix} R & t \\ 0 & 1 \end{bmatrix}$$

and $R \in \mathbb{R}^{3 \times 3}$ being a rotation matrix and $t \in \mathbb{R}^3$ a translational vector, is an element of $\text{SE}(3)$.

2.1.2 ORB-SLAM

ORB SLAM is a feature based, state of the art slam method. The first version was published in 2015 [3]. Here, an overview of the functionality of ORB SLAM is provided. The Algorithms runs on three threads simultaneously. Each thread performs one of the following tasks: Tracking, Local Mapping and Loop Closing. An overview over the tasks can be found in figure 1. The explanation of these system components are described in the following subsections. A more detailed explanation can be found in the paper of Raul Mur-Artal et al [3].

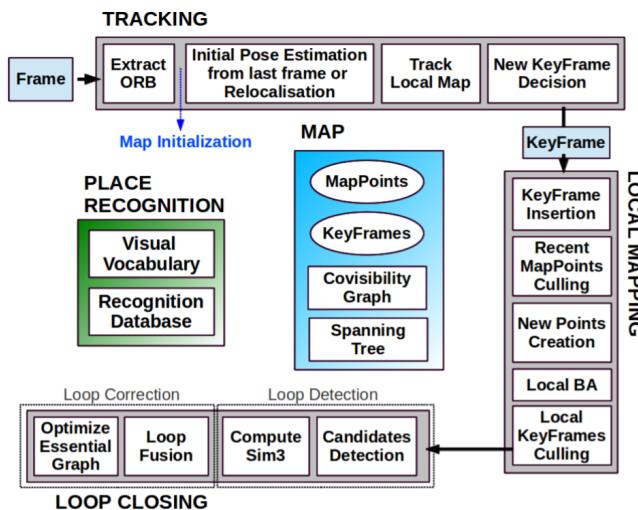


Figure 1: Overview of the system components extracted from [3]

Tracking

The tracking component determines the localization of the camera and decides, when a new keyframe is being inserted. As it is shown in figure 1, the tracking is performed in four steps.

1. Feature Extracting

Features are extracted using Oriented FAST and Rotated BRIEF [4]. This method starts by searching for FAST (Features from Accelerated and Segments Test). Herefor, for each pixel x in the image, a circle of 16 pixels around that pixel is considered and checked if at least eight of these 16 pixels have major brightness differences. If so, the pixel x is considered as a keypoint, since it is likely to be an edge or corner. This is repeated again and again after downsizing the image up to a scale of eight. To extract features evenly distributed over the image, it is divided into a grid, trying to extract five features per cell. Extracting features this way, makes the algorithm more stable to scale invariance. Next the orientation of the extracted feature is calculated using a intensity centroid. Finally the features are converted into binary vectors (ORB descriptor) using a modified version, which is more robust to rotation, of BRIEF descriptors (Binary robust independent elementary feature).

2. Initial Pose Estimation

A constant velocity model is first run to predict the camera pose. Then, the features of the last frame are searched. If no matches are found, a wider area around the last position is searched.

3. Track Local Map

When the camera pose is estimated, map point correspondences are searched in the local map, containing keyframes that contain the observed map points and the keyframes from the covisibility graph. The

pose is then corrected with all matched mappoints.

4. New Keyframe Decision

To insert the current frame as a keyframe, the following conditions have to be met: more than 20 frames have to be passed from the last relocalization or keyframe insertion (when not idle), the current frame tracks at least 50 points or less than 90 percent of the points of the keyframe in the local map with the most shard mappoints.

Local Mapping

Whenever a new Keyframe K_i is inserted, the map is updated.

1. KeyFrame Insertion

The keyframe is inserted in the covisibility graph. Then the spanning tree is updated using the keyframe with the most common points with K_i . Finally the keyframe is represented as a bag of words using the DBoW2 implementation. Therefor, the image is saved by the number of occurrences of features found in a predefined vocabulary of features. When the vocabulary is created with images general enough, it can be used for most environments.

2. Recent Map Points Culling

A mappoint is removed from the map, when it is found in more than 25/it must be observed from more than two keyframes if more than one keyframe has passed from map point creation.

3. New Map Point Creation

A map point is created by calculating the triangulation of the connected keyframes in the covisibility graph. For each map point, the 3D coordinate in the world coordinate system, its ORB descriptor, the viewing direction,

the maximum and minimum distance at which the point can be observed is stored.

4. Local Bundle Adjustment

The keyframe poses $T_i \in \text{SE}(3)$ and Map Points $X_j \in \mathbb{R}^3$ are optimized by minimizing the reprojection error to the matched keypoints $x_{i,j} \in \mathbb{R}^2$. The error is computed by the following term:

$$e_{i,j} = x_{i,j} - \pi_i(T_i, X_j)$$

. i is the respective Keyframe and j the index of the map Point. π_i is a projection function, calculating a transformation to project all keypoints on map points by minimizing a cost function, that can be found in [6]. In case of full BA (used in the map initialization) we optimize all points and keyframes, by the exception of the first keyframe which remain fixed as the origin. In local BA all points included in the local area are optimized, while a subset of keyframes is fixed. In pose optimization, or motion-only BA, all points are fixed and only the camera pose is optimized. At this point, a local BA is performed.

5. Local Keyframe Culling

With difference to other SLAM algorithm, ORB slam deletes redundant keyframes, which decreases computational efforts, since computational complexity grows with the number of keyframes. All keyframes are deleted, where at least 90 percent of the map points can be found in at least three other keyframes.

Loop Closing

The loop closing is computed based on the last inserted keyframe K_i .

1. Loop Candidates Detection

First the similarity of K_i to its neighbours in the covisibility graph is computed by using the bag of words representation and a loop candidate K_l might be chosen.

2. Similarity Transformation

In this step the transformation is computed, to map the map points from K_i on K_l . Since scale can drift, also the scale is computed in addition to the rotation matrix and translation using the method of horn.

3. Loop Fusion

Here, duplicated map points are fused and the keyframe pose T_ω is corrected by the transformation calculated in the previous step. All map points of K_l are projected in K_i . All keyframes affected by the fusion will update the edges (shared map points) in the covisibility graph.

4. Essential Graph Optimization

Finally the loop closing error is distributed over the essential graph.

2.1.3 DSM-SLAM

2.1.4 DSO-SLAM

2.2 Calculations

2.2.1 Trajectory Alignment

In order to compare the evaluated position of the camera at a given time with the ground truth of the position, the trajectories need to be aligned. This is because most SLAM Algorithms initialize the origin of their coordinate system with the camera position from the first frame. Whereas the ground truth of the

trajectory uses a different origin. As a consequence, evaluated points $\{\widehat{p}_i\}_{i=0}^{N-1}$ can not be compared to the ground truth points $\{p_i\}_{i=0}^{N-1}$. Also, as described in the vSLAM Algorithms section,

the minority of the existing vSLAM algorithms are recognizing the true scale of the coordinate system. For those two reasons, the target is to find $S = \{R, t, s\}$, while R being a rotation matrix, t a translation vector and s a scaling factor, such that

$$S = \arg \min_{S'=\{R', t', s'\}} \sum_{i=0}^{N-1} \|p_i - s'R'\widehat{p}_i - t'\|^2$$

In other words, the evaluated points are rotated, translated and scaled in a way, that the sum squared error over the point distances is minimized. The upper expression is calculated by using the method of Umeyama [7].

Similar to principal component analysis, Umeyama uses the singular value decomposition of the covarianve matrix Σ of p and \widehat{p} . Thus, $\Sigma = UDV^T$ is yielded. Umeyama proves, that R, t and s can be calculated as followed:

$$\begin{aligned} R &= UWV^T \\ s &= \frac{1}{\sigma_p^2} \text{tr}(DW) \\ t &= \mu_{\widehat{p}} - sR\mu_p \end{aligned}$$

with

$$W = \begin{cases} I, & \text{if } \det(U) \det(V) = 0 \\ \text{diag}(1, 1, -1), & \text{otherwise} \end{cases}$$

σ_p beeing the standard deviation of p , μ the mean and tr the trace of a matrix.

2.3 Datasets

2.3.1 EuRoC Dataset

For the evaluation of the vSLAM Algorithms, the EuRoC dataset [1] was used. The dataset contains eleven video sequences, recorded with a micro aerial vehicle at 20 frames per second. The sequences have a resolution of 752x480 pixels. For each Sequence, RGB images from two cameras exist. However, since the evaluation focuses on monocular SLAM methods, only the left camera was considered. Also the available inertial and camera pose data was not taken in consideration. The first five sequences were recorded in the machine hall at ETH Zürich, and the other six were recorded in a room, that was provided with additional obstacles. For the latter six sequences, the groundtruth of the environment exists as a dense pointcloud, as can be seen in figure 2.

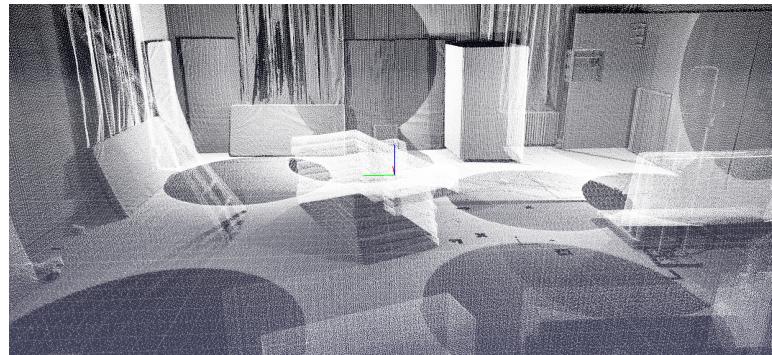


Figure 2: Pointcloud ground truth of sequence V1_01_easy visualized with python package pptk

Finally the true position of the camera is known at a high frequency of over 200 points per second. An overview of the sequences is shown in table 1.

Table 1: Overview of the sequences included in the EuRoC Dataset

Sequence Name	Duration in s	Average Velocity in ms^{-1}	Pointcloud available
MH_01_easy	182	0.44	No
MH_02_easy	150	0.49	No
MH_03_medium	132	0.99	No
MH_04_difficult	99	0.93	No
MH_05_difficult	111	0.88	No
V1_01_easy	144	0.41	Yes
V1_02_medium	83.5	0.91	Yes
V1_03_difficult	105	0.75	Yes
V2_01_easy	112	0.33	Yes
V2_02_medium	115	0.72	Yes
V2_03_difficult	115	0.75	Yes

2.4 Setup and Environment

2.4.1 Evaluation

The entire evaluation is run on a virtual machine. The host system is a lenovo yoga with eight GB of RAM and the basic model (8250U CPU @1.6 GHz 1.80GHz) of an eight core i5. The operating system of the host machine is Windows 10 Home. The virtual machine is given 5 GB of Ram and 4 cores. The operating system of the virtual machine is Ubuntu 18.04. All further setup information can be extracted from the github repository.

2.4.2 Flight Path Planning

3. Results

The three SLAM algorithms were evaluated regarding the computed trajectories, the resulting pointclouds and the computational complexity.

3.1 Trajectory Evaluation

In order to evaluate the quality of the trajectory computed by the algorithms, the trajectory firstly had to be transformed into the world reference of the ground truth data. This is because the algorithms usually initialize the origin with the first (key)frame and the groundtruth data doesn't. Furthermore, monocular visual slam algorithms are generally not capable to extract the true scale. The alignment was performed using the method of Umeyama, described in the ?? section. After alignment, as a first indicator for the accuracy of the computed trajectories, the trajectories were visually observed by plotting the true position and the evaluated position into a coordinate system. The x, y and z axis were observed separately.

In figure 3 the computed trajectories are displayed for sequences MH01, V102 and V203.

Furthermore, the euclidean distances between position of the keyframe and the true position of the latter are computed. For a keyframe, the entry of the groundtruth data with the lowest distance in time to the time the keyframe was

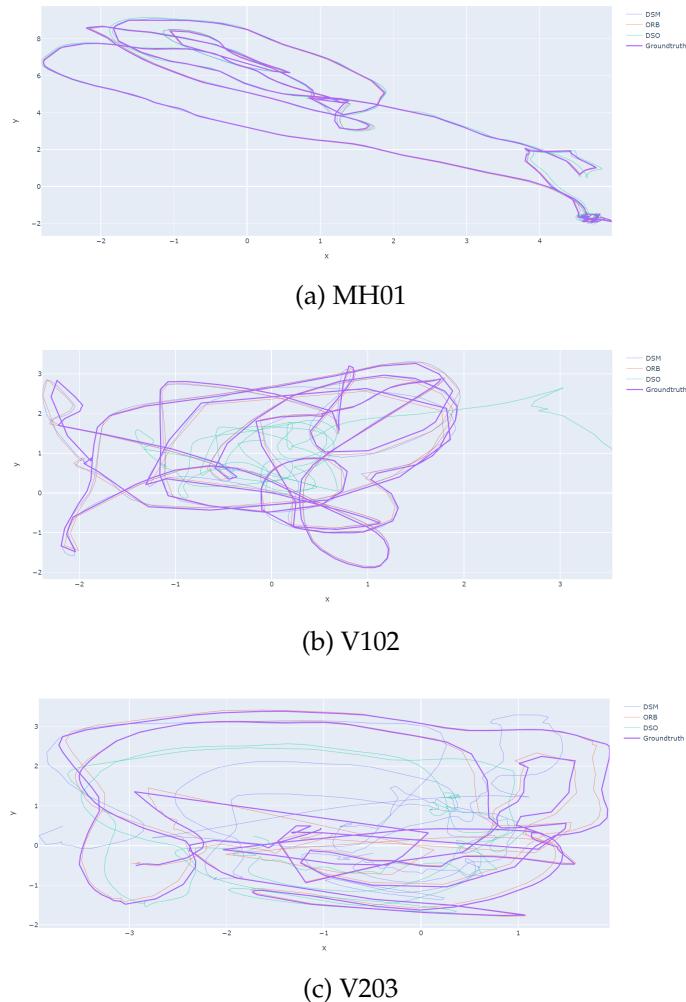


Figure 3: Ground truth flight path and evaluated flight path of each algorithm after alignment with the method of Umeyama in the x and y axis in meters. Left the sequence MH01, middle the sequence V102 and right the sequence V203 is displayed.

inserted is taken as reference point. This is justifiable, since the true position is sampled at a frequency of over 200 points per second.

In figure 4 a boxplot of all computed distances over all sequences is displayed.

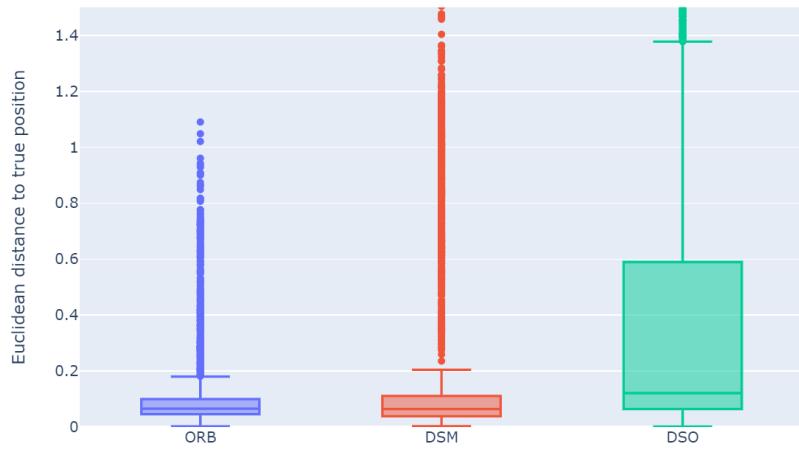


Figure 4: Boxplot of all euclidean distances between the ground truth position of the keyframe and the evaluated position after alignment with the method of Umeyama. Outliers greater than 1.5 are not displayed.

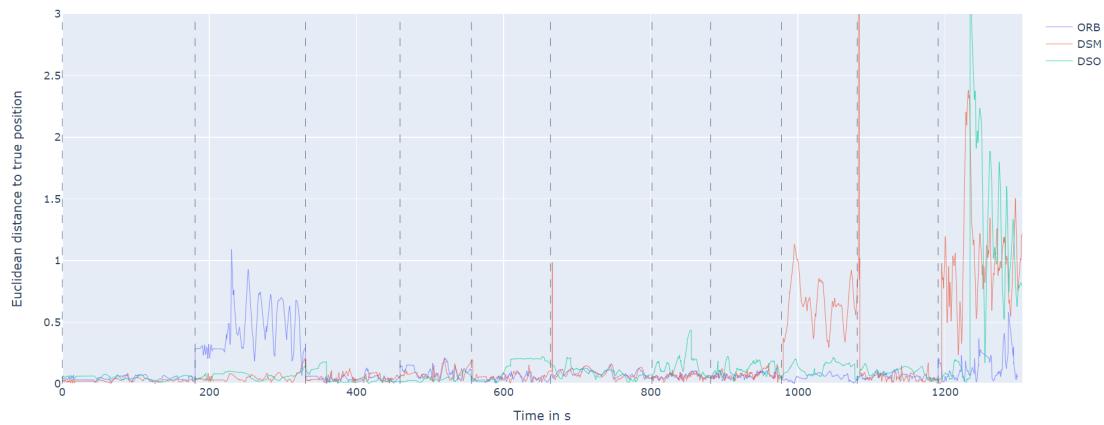


Figure 5: The positional error over time in meters. The vertical lines indicate the beginning of a new sequence

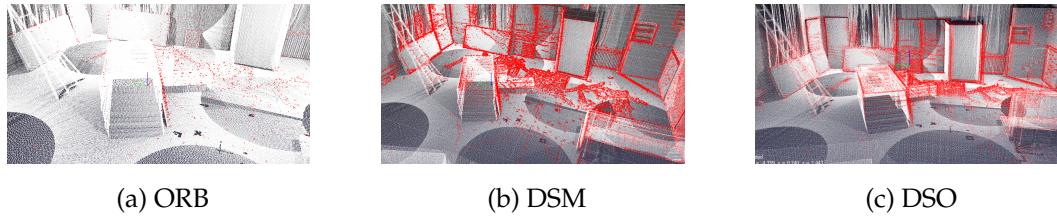


Figure 6: The groundtruth of the Pointcloud from Sequence V101 (white points) and the evaluated points by each algorithm (red points). The points in Figure (a) are four times as large for better visibility (ORB-SLAM generates only few points).

Table 2: Number and accuracy of evaluated points of each algorithm

Sequence Name	ORB	DSM	DSO
MH_01_easy	8958 (/)	675720 (/)	361633 (/)
MH_02_easy	8692 (/)	700920 (/)	343804 (/)
MH_03_medium	/ (/)	614264 (/)	371752 (/)
MH_04_difficult	7943 (/)	495752 (/)	208445 (/)
MH_05_difficult	8373 (/)	517712 (/)	232415 (/)
V1_01_easy	7075 (0.049)	6108440 (0.066)	374257 (0.066)
V1_02_medium	6517 (0.042)	648440 (0.187)	366513 (1.458)
V1_03_difficult	/ (/)	775080 (0.092)	448212 (0.459)
V2_01_easy	/ (/)	584552 (0.58)	247905 (0.086)
V2_02_medium	/ (/)	733992 (0.078)	490608 (0.104)
V2_03_difficult	/ (/)	921312 (0.645)	465396 (0.677)

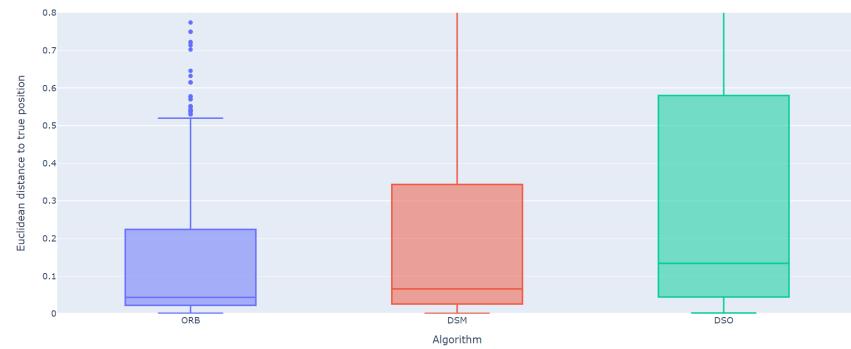


Figure 7: Boxplot of the euclidean distances between an evaluated point and the closest point of the ground truth point cloud. For computational feasibility, for each sequence and algorithm, 500 points for evaluation are sampled randomly

3.2 Pointcloud Evaluation

3.3 Calculation Time

Table 3: Computation Time (excluded time needed for initialization) of each Sequence and Algorithm

Sequence Name	Computation Time in s ORB	Computation Time in s DSM	Computation Time in s DSO
MH_01_easy	257	1098	749
MH_02_easy	209	984	690
MH_03_medium	198	1369	707
MH_04_difficult	165	896	504
MH_05_difficult	193	825	633
V1_01_easy	253	1383	905
V1_02_medium	150	1550	820
V1_03_difficult	186	2262	1134
V2_01_easy	187	1045	612
V2_02_medium	162	1675	1522
V2_03_difficult	143	1600	793

4. Discussion

4.1 Conclusion of SLAM-Algorithm Evaluation

4.2 Framework for an entire automated System

Furthermore, a framework to test and build an entire automated system is suggested. This framework includes a simulated simulated environment, that realistically makes it possible, to navigate a drone within an simulated environment. The environment is based on the Roboter Operating System (ROS) and for the simulations the tum_simulation ros package, developed by the Technische Universität München, was resorted to.

ROS, as the name suggests, is a framework for a software infrastructure within a robot. With the right drivers installed, it can access and use the robots hardware and serves as a messenger system between robot components. Ros packages make it easy to reuse important functionalities. Gazebo on the other hand is a 3D dynamic simulator for robotics. It can accurately and efficiently simulate robots regarding their physics.

Thus, with the tum_simulation package you can navigate an AR.drone 1.0 and 2.0 in different worlds created with a gazebo node. This drone is equipped with a bottom camera and a front camera. The cameras each log their output to a ros topic. Additionally, message time stamps, the height sensor output, battery

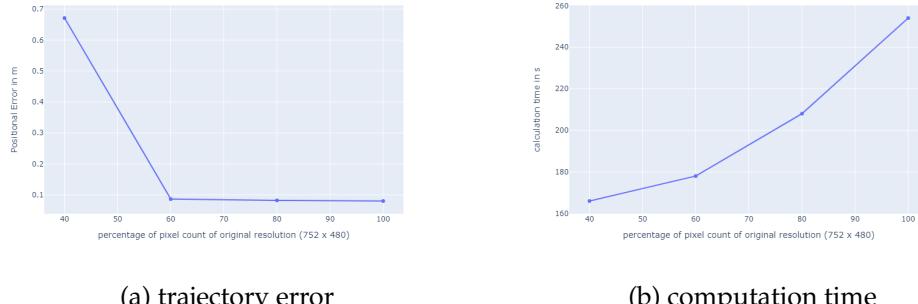


Figure 8: Influence of downsizing of the images on the trajectory error (a) and the computation time (b) for the sequence V101.

percentage, position, rotation velocity and acceleration are also logged to rostopics. While the drone can also be navigated using a playstation 3 controller, as shown in figure 9, showing a section of the tum_simulation package content structure, for an automated system, the drone should rather be addressed using the command line interface. For example the command shown in sourcecode listing 1 will make the drone fly forward.

Listing 1: drone navigation command

```
1   rostopic pub -r 10 /cmd_vel geometry_msgs/Twist '{  
    linear: {x: 1.0, y: 0.0, z: 0.0}, angular: {x:  
        0.0,y: 0.0,z: 0.0}}'
```

By accessing the camera output of the drone via rostopics, SLAM algorithms can be applied on the generated images. Some SLAM-Algorithms, such as an modified version of the ORB-SLAM-Algorithm ([ORB_SLAM2_ROS](#)), publish the created pointcloud again in an own rostopic. The fight path planning algorithm on the other hand takes the publishes pointcloud as input and navigates the drone using the command line interface.

This circle of a fully automated SLAM system is shown in figure 10. For the initialiazation of the process, the manual step of flying the drone may

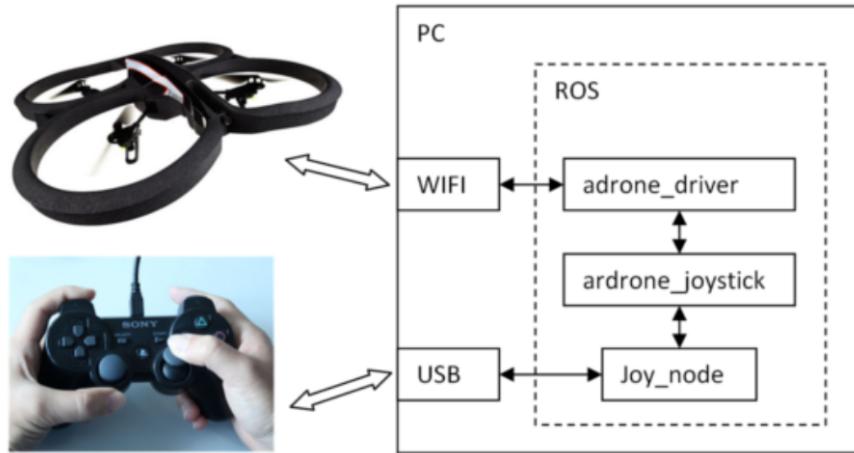


Figure 9: tum simulator setup. Source: http://wiki.ros.org/tum_simulator

be required. This is because the ORB-SLAM system needs a translational movement to initialize the algorithm. Though, in most cases, a normal take off will be sufficient.

In figure [?] the simulated environment with gazebo can be found in figure a. Here, the drone (in black) is flying in front of the building. The output of the front camera is shown in figure b. Therefore, the output calibration data of the front camera, such as the focal length, can be found in an own rostopic. In figure c, the ORB-SLAM-Algorithm was applied to the output of the front camera. Green dots represent the finding of an ORB-feature.

4.2.1 Current setup

Currently the framework is set up in an environment provided by theconstruct-sim.com. This platform is enabling ROS-developers to program in preconfigured ROS-environments. The environment comes with the possibility to open terminal consols, a file management system, a simulator, that automatically detects, when a gazebo simulation is running. Also, you have a graphical interface for other graphical applications, such as the viewer of ORB-SLAM. The current environments is set up with ROS kinetic and Ubuntu 16.04.6 LTS

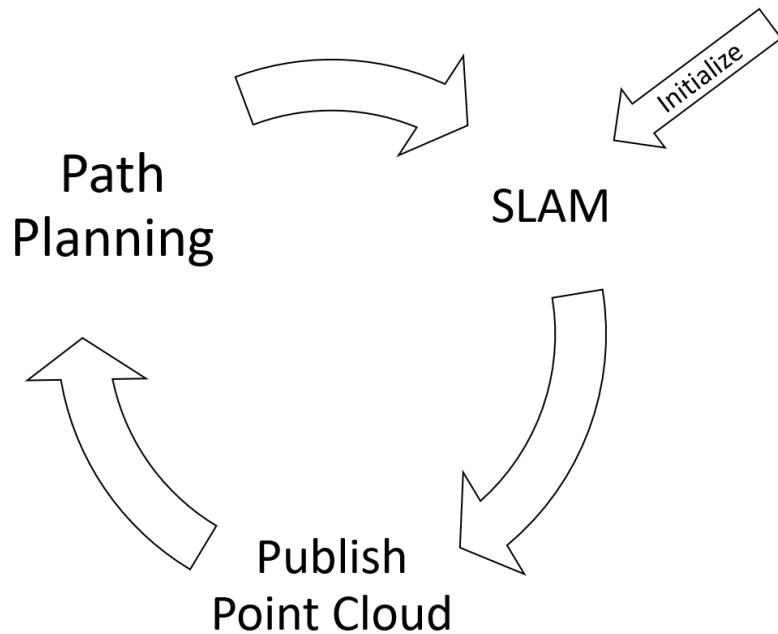


Figure 10: Automated SLAM system

(Xenial). The tum_simulator, ORB-SLAM and all of their dependencies are already installed.

Listing 2: Launching the simulated environment

```
1      # launch the gazebo simulation
2          roslaunch cvg_sim_gazebo ardrone_testworld.launch
3
4      # launch ORB-SLAM
5          rosrun ORB_SLAM2 Mono ${PATH_TO_VOCABULARY} ${
6              PATH_TO_SETTINGS_FILE}
7
8      # takeoff with drone
9          rostopic pub -1 /ardrone/takeoff std_msgs/Empty
```

In listing 2 the commands for launching the gazebo simulation, ORB-SLAM and the drone are displayed. After launching those applications, only the path

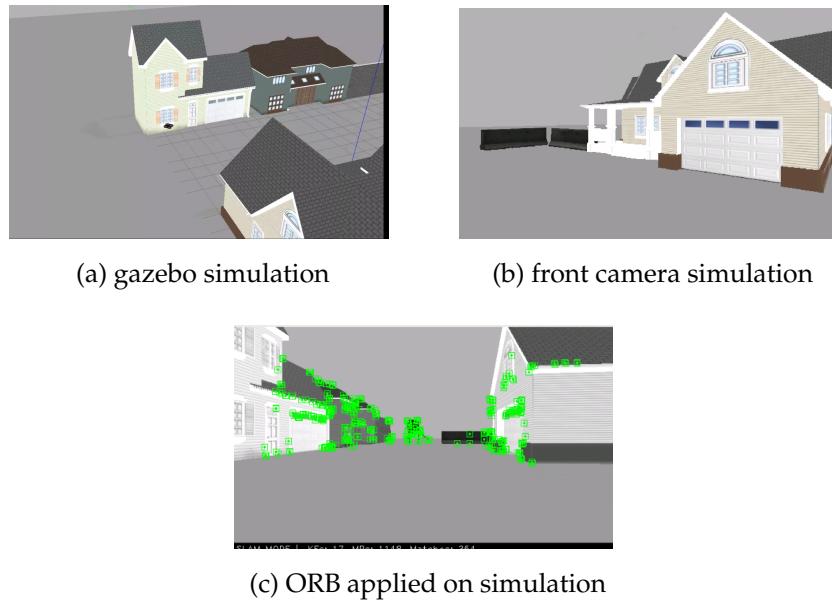


Figure 11: The drone in a gazebo simulation in a), the output of the front camera of the drone in b) and the ORB-SLAM algorithm applied on the front camera output in with the detected ORB features marked green c).

planning algorithm based on the resulting point cloud is missing. However, multiple solutions for such algorithms exist, applying it on the system is not part of this paper and will be done in further research.

Bibliography

- [1] M. Burri. The euroc micro aerial vehicle datasets. *The International Journal of Robotics Research*, 2016.
- [2] E. Eade. Lie groups for computer vision. 2002.
- [3] R. Mur-Artal. Orb-slam: a versatile and accurate monocular slam system. *IEEE TRANSACTIONS ON ROBOTICS*, 2015.
- [4] E. Rublee. Orb: an efficient alternative to sift or surf. 2012.
- [5] H. Strasdat. Visual slam: Why filter? *Image and Vision Computing*, 2000.
- [6] B. Triggs. Bundle adjustment – a modern synthesis. *International Workshop on Vision Algorithms*, 2000.
- [7] S. Umeyama. Least-squares estimation of transformation parameters between two point patterns. *IEEE Trans. Pattern Anal. Mach. Intell.*, 1991.