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 selbständig 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

List of Figures

List of Tables

List of Listings

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 simultanious 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 simultanious 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 simulatanious 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 completly unknown environment, while localizing itself within it only be using certain sensors. This paper is limited to evaluate monocular vSLAM algorithms, meaning that the algorithm is only working with a single RGB camera as sensor. Therefore, since nowadays RGB cameras are either a standard on drones or can easily be upgraded, these drones are 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 suggested. 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.

2. Introduntion to vSLAM Algorithms

SLAM is one of the most emerging research topics in robotics [6]. It is applied in various applications, such as in augmented reality to estimate the camera pose, autonomous navigation and computer vision-based online 3D modeling [11] [4]. The problem can be defined in a probabilistic way. The goal is to compute

$$\mathbb{P}\left(m_{t+1}, x_{t+1} | z_{1:t+1}, u_{1:t}\right)$$

Where, as can be found in figure 1, m_{t+1} is the map (pointcloud of the surroundings) at timepoint $t+1$, x_{t+1} the camera pose and position at timepoint $t+1$, $z_{1:t+1}$ all observations made to this timepoint and $u_{1:t}$ all historic control input. However, most modern SLAM methods don't require the control input anymore.

With their work on the representation and estimation of spatial uncertainty [9] Smith et al created the first relevant work in the field on SLAM in 1986. However, due to lacking computational recources, the engagement in this topic stayed aminly on a theoretical level.

> The result of this conversation was a recognition that consistent probabilistic mapping was a fundamental problem in robotics with major conceptual and computational issues that needed to be addressed. [2]

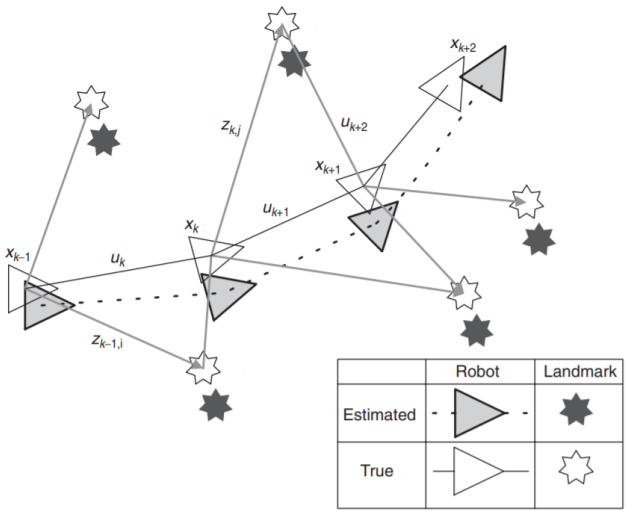


Figure 1: Overview of the SLAM concept. Source: [2]

This is because already then it was recognized, that camera pose and landmark positions had to be updated when moving further or optimizing the map and is a huge computational effort as the map grows.

As time progressed, so did the computational resources and the algorithms became more advanced. Huge evolvements were achieved after 2010 [11], mainly because of the increased use of augmented reality application, that may rely on real time vSLAM algorithms. The developed vSLAM methods can be devidet into two different classes: direct and feature based methods.

Direct algorithms use the entire image as input in order to compute the term in quotation **??**. Therefore, the term is computed by optimizing the photometric error. This is the error, that results from comparing the intensities of each pixel after transformation (optimization) [4].

> One of the main benefits of a direct formulation is that it does not require a point to be recognizable by itself, thereby allowing for a more finely grained geometry representation (pixelwise inverse depth). [4]

Feature based methods on the other hand, compute features for each frame, that serve as imput for further computation. These features usually are subsets of pixels, that have remarkable intensities and arrangement, such as corners (landmarks). These methods evaluate the upper term by computing the geometric error, since the feature positions are geomtric quantities. A main advantage of feature based methods is the rubostness over geometric distortions present in every-day-cameras [4].

Of all existing vSLAM algorithms, this work considers DSO, DSM and ORB SLAM for the evaluation of beeing a suited candide for flying a drone autonomously. This desicion is based on other research results in this area.

In the following section, an overview over how the algorithms work is given for each method. However, in order to understand this section, it is crutial to clarify basic definitions and vocabulary used in SLAM

2.1 Definitions

Keyframe

Most vSLAM Algorithms make usage of so called keyframes, as these keyframe-based approaches have proven to be more accurate [10]. Keyframes are specific selected images of the input sequences or video streams. In most cases, the keyframes store all of the existing map points and all of the computations and optimizations are made, based on the keyframes and data stored within them.

Covisibility Graph

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

Group of Rigid Transformations in 3D

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

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

and $R \in \mathbb{R}^{3x3}$ beiing a rotation matrix and $t \in \mathbb{R}^3$ a translational vector, is an element of SE(3).

Pointcloud

When we speak of pointcloud, we simply mean a formation of points in the threedimensional space, though having an x y and z coordinate. Later in the evaluation of the algorithms, to save these pointclouds, the PLY format has been chosen.

2.2 ORB-SLAM

ORB SLAM is a feature based, state of the art slam method. The first version was published in 2015 [7]. Here, an overview of the functionality of ORB SLAM is provided. The Algorithms runs on three threats simultanously. Each thread performes one of the following tasks: Tracking, Local Mapping and Loop Closing. An overview over the tasks can be found in figure 2. The explaination of these system components are described in the following subsections. A more detailed explaination can be found in the paper of Raul Mur-Artal et al [7].

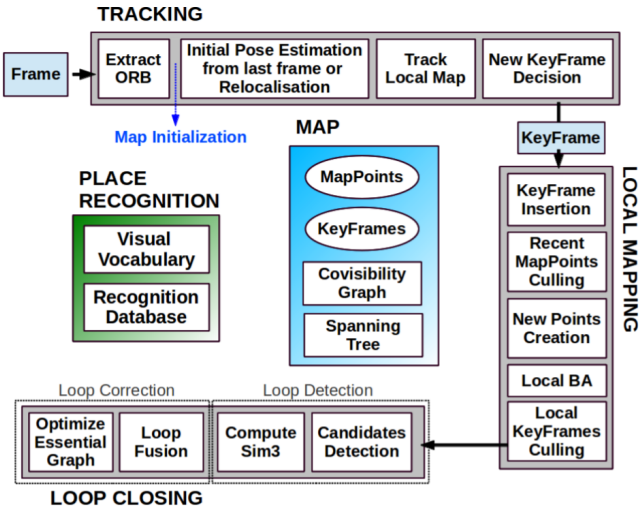


Figure 2: Overview of the system components extracted from [7]

2.2.1 Tracking

The tracking component determines the localization of the camera and decides, when a new keyframe is beeing inserted. As it is shown in figure 2, the tracking is performed in four steps.

1. Feature Extracting

   Features are extracted using Oriented FAST and Rotated BRIEF [8]. 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 pixels 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 devided 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 a 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 to 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.

2.2.2 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 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 occurances of features found in a 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 maximim and minimum distance at which the point can be observed is stored.

4. Local Bundle Adjustment

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

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

   . i is the respective Keyframe and j the index of the map Point. π_i is a projection function, calculation a transformation to project all keypoints on mappoints by minimizing a cost function, that can be found in [12].

   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 delets redundant keyframes, which decreases computational efforts, since computational complexity grows with the number of keyframes. All keyframes are deleted, where at least 90 percest of the map points can be found in at least three other keyframes.

2.2.3 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 covisibilty 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 computet, to map the map points from K_i on K_l. Since scale can drift, also the scale is computet in addition to the rotation matrix and translation using the mothod 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 edjes (shared map points) in the covisibilty graph.

4. Essential Graph Optimization

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

2.3 DSO-SLAM

Direct Sparse Odometry was developed in 2016 by the Technische Universität München.

2.3.1 Model Overview

The model optimzes the photometric error over a window of recent frames.

The camera is calibrated in two different ways. First a projection function is computed, considering the focal length and the pricipal point, in order to map a pixel point in the image on a 3D map point (and the other way around). Secondly, a photometric camera calibration is applied. This calibration accounts for camera specific non-linear response functions, that map scene irradiances on pixel intensities on the one hand and camera specific lens attenuations on the other hand.

The model relies on minimizing the photometric error. This error over all frames is given by

$$E_{\text{photo}} := \sum_{i \in \mathcal{F}} \sum_{p \in \mathcal{P}_i} \sum_{j \in \text{obs}(p)} E_{pj}$$

,

where $\mathcal{F}$ is the set of all frames, $\mathcal{P}_i$ the set of all pixel in frame i and $\text{obs}(p)$ the set of indices of frames, where the point p occurs. E_{pj} on the other hand is the difference in pixel intensities, calculated by the huber norm, after mapping the pixels on each other and applying an additional affine brightness transfer function. Also, E_{pj} does not include comparing the point p, but also computes the pixel intensitie difference of eight pixel neighbors of p, arranged in a spread pattern.

2.3.2 Visual Odometry Front-End

The front end determine the sets $\mathcal{F}$, $\mathcal{P}_i$ and obs (p). Also it initializes all parameters required to calcualte the term **??**. Finally it decides, when to remove points, outliers and keyframes. Frame and point management are closer described in the following.

1. Frame Managemant

   If a new keyframe is created, all map points are mapped into it. In case the root mean squared error of the current frame is more than twice as high than the one before, direct image alignment is assumed to have failed and initialization is tried again. The algorithm tries, to always work with seven active keyframes. All computations are made in reference to those keyframes.

   For creating a new keyframe, 5-10 frames per second are considered for creation. A keyframe must meet all of the following requirements:

   (a) the field of view changes.

   (b) Camera translation causes occlusions.

   (c) Camera exposure time changes significantly.

   When deciding when to marginalize a keyframe, following roles are applied to the active keyframes I_1,...,I_7, with I_1 beeing the most recent:

   (a) I_1 and I_2 are always kept

   (b) keyframes, whos amount of points contained in I_1 is less then 5 percent are marginalized

   (c) If still too many keyframes are active, the ones having the largest distance to I_1, or the worst distribuited in 3D space are removed.

2. Point Managemant

   DSO always tries to keep 2000 active points in the map. The first step is select candidates points of the frame. To obtain equally distributed points, the image is split into blocks. In each block, the point with the largest gradient of pixel intensity is selected, if it is greater than a threshhold, which is computed by adding seven to the overall median of the gradients. This is repeated twice while decreasing the threshhold and doubling the block size, in order to also include points with weaker gradient.

   Then, the point candidates are tracked in subsequent frames by minimizing E_{pj}. From the best match, the depth value is initialized.

   When old points are marginalized, candidate points are activated in a way that points in the active map are as evenly distribuited as possible. This is achieved by always selecting the point, which offers the largest distance to the next active point, after projecting it into the last active keyframe.

   Since outlier only consume unnecessary ressources, they are tried to be removed. For example, point for which E_{pj} surpasses a threshold are removed permanently.

2.4 DSM-SLAM

Direct Sparce Mapping SLAM was released in april 2019 and works similar to DSO slam but claims to be more robust when revisiting areas. The algorithm can be seperated into a tracking front-end and an optimization back-end that run on parallel threads.

2.4.1 Model Overview

DSM uses the same model as DSO. The model is described in section 2.3.1.

2.4.2 Front-End

The front-end, however, differs from DSO. While DSO cannot reuse points that have one beein marginalized, DSM suggests a method to activate and inactivate keyframes and points to its needs.

keyframe and mappoint selection

When selecting keyframes and mappoints, two criteria play a role: the temporal and covisibility criteria. The temporal parts regards N_t keyframes as recent sliding window approach, just like DSO. With similar critirias for keyframe selection as DSO, whenever a new keyframe is inserted, anotherone is removed (from the temporal part).

In order to regard reobserved points, DSM also considers N_c covisibil keyframes to fill the latest keyframe I_0 with map points, favouring map points in depleted areas. This is achived by the following steps:

1. Identify depleted areas

   All map points from the temporal part are mapped into the latest keyframe. For every pixel, the eucleadian distance to the closest map point is computed. Obvously, large distances suggest depleted areas.

2. covisibil Keyframe Selection

   Select the keyframe within the old keyframes with the most map points in the above selected depleted area. Map points, where the viewing angle

lies too far from the latest keyframe are removed from the local map. This can be determined from the pose $T_i \in \mathrm{SE}(3)$ that is saved for each keyframe I_i.

3. Update distance map

   Update the distance map, calculated in step 2, with the new selected keyframe.

4. iterate

   Iterate until N_c keyframes are selected for the covisibility part.

The entire keyframe and mappoint selection is displayd in figure 3. In this case, N_t is equal to four and N_c is equal to three.

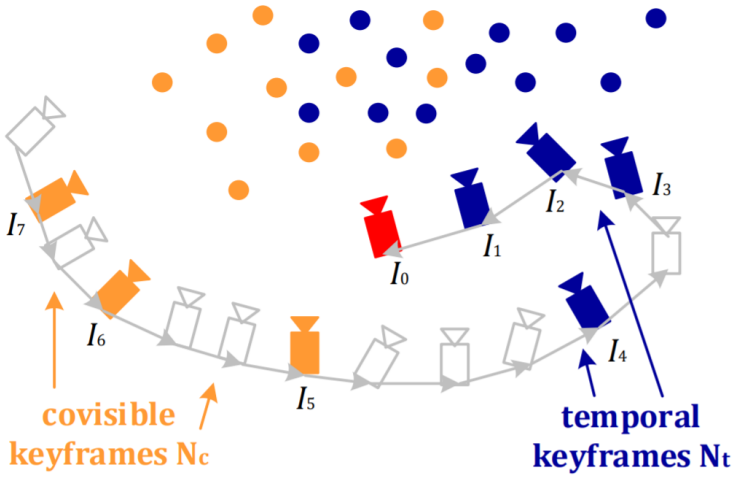


Figure 3: Keyframe and Point Selection of DSM. Source: [15]

Frame Tracking, New Keyframe desicion, New Map Point Tracking

Frame Tracking, New Keyframe desicion and New Map Point Tracking work similar to DSO slam. The main idea is always to manipulate the local map by minimizing the photometric error displayed in equation **??**.

However, unlike DSO SLAM, in each frame, the local map consisting of the $N_t + N_c$ keyframes (referenced from the latest keyframe) and contained map points is projected into the frame. Obviously, this also brings the difference compared to DSO slam, that keyframes and map points are not permanently culled from the map, and may be reactivated once the keyframe appears in the covisibilty graph. Also, the management of outliers is similar to DSO-SLAM, Trying to remove outliers as early as possible, in order to save computational resources.

3. Evaluation Methods

3.1 Datasets

3.1.1 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 resultion 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 obsticals. For the latter six sequences, the groundtruth of the environment exists as a dense pointcloud, as can be seen in figure 4.

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.

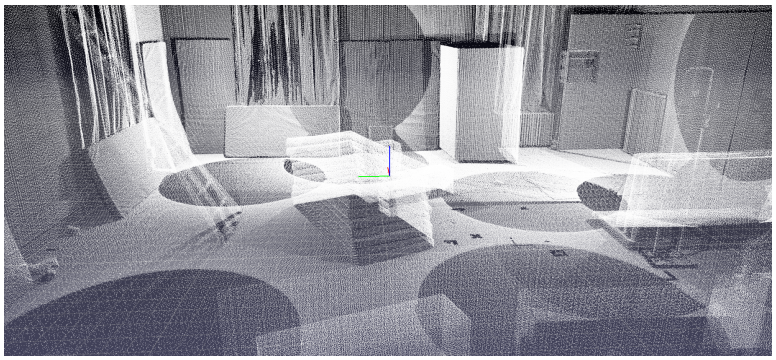


Figure 4: Pointcloud ground truth of sequence V1_01_easy visualized with python package pptk

3.2 Trajectory Comparison

3.2.1 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 innitilize 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 $\{\widehat{x_i}\}_{i=0}^{N-1}$ can not be compared to the ground truth points $\{x_i\}_{i=0}^{N-1}$ 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

$$S = \arg\min_{S'=\{R',t',s'\}} \sum_{i=0}^{N-1} \left\| x_i - s' R' \widehat{x_i} - t' \right\|^2$$

.

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

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 expresssion is calculated by using the method of Umeyama [13].

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

$$R = UWV^T$$
$$s = \frac{1}{\sigma_p^2}\mathrm{tr}\,(DW)$$
$$t = \mu_{\widehat{x}} - sR\mu_p$$

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 x, μ the mean and tr the trace of a matrix.

3.2.2 Positional Error

The error between $\{\widehat{x_i}\}_{i=0}^{N-1}$ and $\{x_i\}_{i=0}^{N-1}$ is computet after aligning them with the upper method using the computet parameters $S = \{R, t, s\}$, yielding

$$\widehat{x_i'} = sR\widehat{x_i} - t$$

. Then the distances between the points are evaluated using the eucledean norm:

$$e_i = ||\widehat{\mathbf{x}_\mathbf{i}'} - \mathbf{x_i}||_2$$

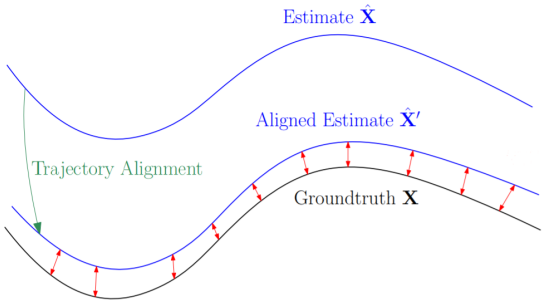


Figure 5: Trajectory error after alignment. Source: [14]

In figure 5 the computet errors are displayed.

These error terms are visualized over time and the overall mean is determined and again visualized with boxplots for each method. Additionally, the flight paths are plotted against each other after alignment, to gain visual information of the trajectories. This is done for all three axes.

3.3 Pointcloud evaluation

The algorithms were manipulated in a way, that after evaluating each sequence, they write a .PLY file with all map points to the device. These map points are then evaluated by the following methods. Obviously, this is only done for latter six sequences, where a groundtruth of the pointcloud exists. Additionally to the following methods, the point clouds are visually observed, trying to figure out, if the SLAM algorithms are also able to detect small obstacles, which is crucial for a successfull navigation.

3.3.1 Positional Error

Again, the map points are transformed using the method of umeyama. However, it is crucial to note that the computed $S = \{R, t, s\}$ is not a result of aligning the point clouds, but rather the parameters for aligning the trajectories are used. This is done, to ensure, that trajectory and point cloud are transformed in the same way, and fit in the same world reference.

To compare the the transformed computet point cloud $P' = \left\{\widehat{p'}_i\right\}_{i=0}^{M_{\text{eval}}-1}$ to the ground truth point cloud $\{p_i\}_{i=0}^{M_{\text{gt}}-1}$, for a point in the evalueated point cloud, the distance to the closest point in the ground truth point cloud is calculated. This is assumed to be the points ground truth position, since with several 100000-points in groundtruth, this point in the ground truth point cloud should not be too far away from the orthogonal projection of the evaluated point.

This situation is displayed in fiture 6, where it gets clear, that the more points are available in the ground truth point cloud, the smaller is the distance in between them and therefore the smaller the error e becomes.

Since calculating distances from several 100000 points to several 100000 points

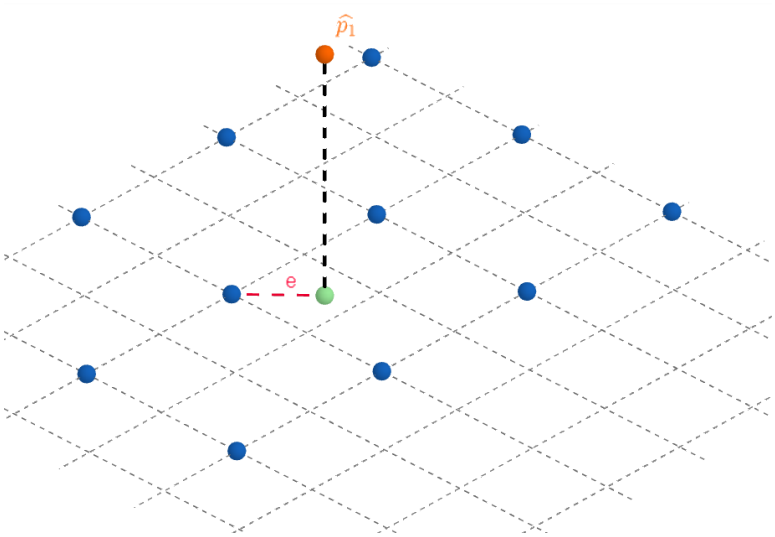


Figure 6: Orthogonal projection of a point in the evaluated pointcloud ($\widehat{p_1}$) on the planar of the groundtruth point cloud. The error e, determines how far the considerd point for the ground truth lies from the actual point of the ground truth.

is computational very expensive, and in the current setup applying it an all sequences and algorithms would require more than a day, only a subset of P' of 1000 points per sequence and algorithms is taken into consideration. The indices for the subset I_{sub} are sampled from a even distrubution of $i_{i=0}^{M_{\text{eval}}-1}$. Then, as mentioned, distances to the closest point in the ground truth point cloud is calculated for the sampled subset subset $P'_{\text{sub}} = \left\{\widehat{p'}_i\right\}_{i \in I_{\text{sub}}} \subset P'$.

The error term for $\widehat{p'_i} \in P'_{\text{sub}}$ is then given by

$$e_i = \underset{j \in i_{i=0}^{M_{\text{gt}}-1}}{\arg\min} \|\widehat{\mathbf{p}'_\mathbf{i}} - \mathbf{p_j}\|_2$$

.

These error terms are then plotted within a boxplot for each method over all sequences.

3.3.2 Density

As discribed in the second chapters, as a result of the functionality behind feature based methods, theire evaluated point clouds are significantly less dense. To quantify the density, for each algorithm and sequence the absolut number of points generated by the algorithm is accessed.

3.4 Computation Time

Since the computational performance of an algorithm is crutial to perform in real time, the absolute time that is needed to process each sequence is measured for each algorithm. The time required for initialization is substracted, since it is not decisive for the assessment, if the algorithm can be run in real time. For each sequence the resulting speed is additionally evaluated in computet frames per second.

3.5 Setup and Environment

3.5.1 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 for the computations. The operating system of the virtual machine is Ubuntu 18.04. All further setup information can be extracted from the github repository.

3.5.2 Flight Path Planning

4. Results

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

4.1 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 coordiante system. The x, y and z axis were observed seperatly. This method for comparing the trajectories is described in detail in section **??**.

In figure 7 the computed trajectories are plotted against the groundtruth for sequences MH01, V102 and V203 considering only the x and y coordinate. These sequences were selected, because they represent the results of the visual analalysis well. In the first sequence, MH01, all algorithms showed excellent results. One reason for this could be, that in the first sequence, the camera only

does very gentel movements, moving at an average speed of $0.41 ms^{-1}$.

The second image shows the sequence V102. Here

Furthermore, the eucleadian distances between position of the keyframe and the true postion 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 inserted is taken as reference point. This is justifiable, since the true position is sampled at a frequency of over 200 points per second.

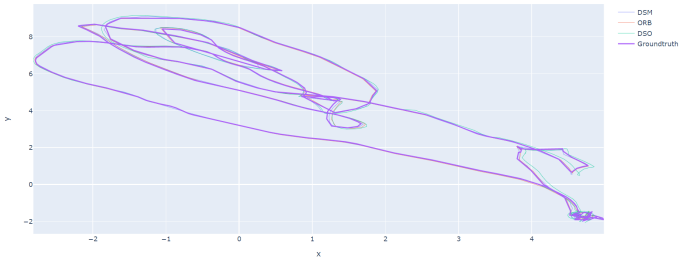
In figure 8 a boxplot of all computed distances over all sequences is displayed.

4.2 Pointcloud Evaluation

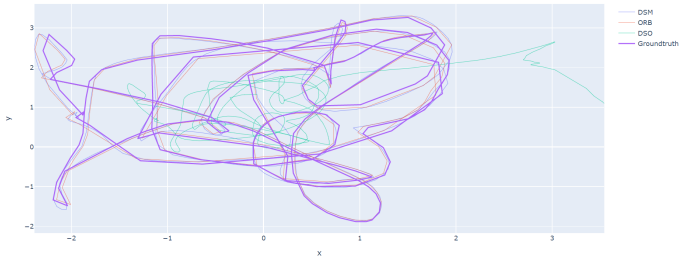
For the evaluation of the computed point clouds, these point clouds were first visually observed, as described in section **??**. Figure 10 shows the evaluated point clouds aligned with the ground truth point cloud for sequence V101. This is a sequence, where the tracking of the trajectory was successfull for all three algorithms, thus, the errors of the resulting point clouds can not be a result of errors in alignment.

What becomes clear at first glance is, that as mentioned, ORB generates only few points, since only found keypoints are mapped in feature based methods. To give these points better visibilty, the point size was doubled in the ORB image. DSM and DSO slam generate point clouds with significant higher density, where all structures of the room are clearly visible at first glance.

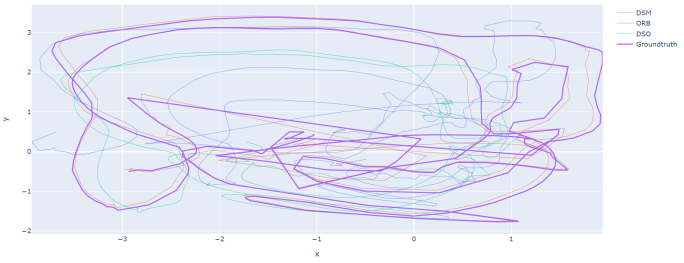
However, the advantace of ORB-SLAM over the other two direct methods is the recognizing of clear features in terms of strucural differences in the scenes. Though, DSO and DSM also regard the differences in pixel intensities ORB, as described in section **??**, detects the features on different scale levels and



(a) MH01



(b) V102



(c) V203

Figure 7: 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 displyed.

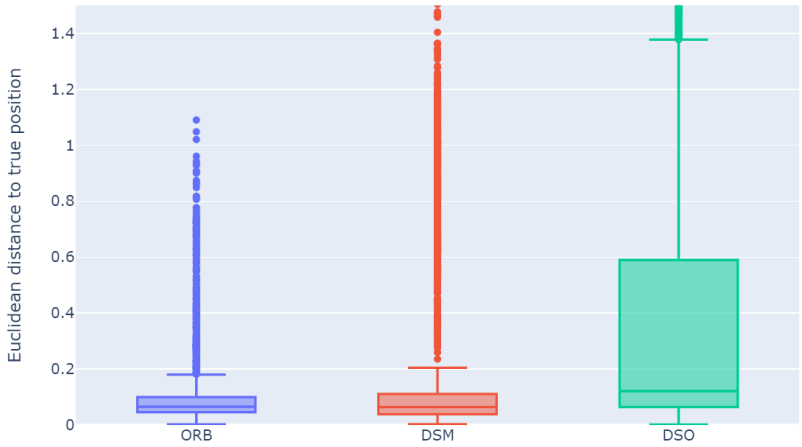


Figure 8: Boxplot of all euclideans distances between the ground truth position of the keyframe and the evaluated position after alignment with the method of Umeyama. Outliers greater than 1.5 are not displayed.

ensures, that the regarded features are in fact significant. This also became clear when observing the point cloud. All significant features, and therefore important features for autonomous navigation, were successfully marked with a computed point. For example, this can be seen at the leiter?? in image three of figure 10, where all sprossen contain at least on point.

After closely observing the point clouds, it became clear, that the point clouds of DSO, often times generate point clouds, where multiple layers of points were falsly generated, where all points had the same clear distance to the ground truth point cloud. This may be a result of working of DSO slam, where keyframes, that are marginalized, are removed permanently. When revisiting areas, the points are again generated. This means that all errors made in the sequence accumulate and when an area is revisited, significant errors in the point cloud can be made. This can be seen when looking at the third image closely. When looking at the matraccess in the middle of the room, the accumulated error expresses itself by points hovering in the air in a clear plane.

The density can also be expressed by numbers. The significant difference of the numbers of points can be seen in table 2. While ORB slam only generates

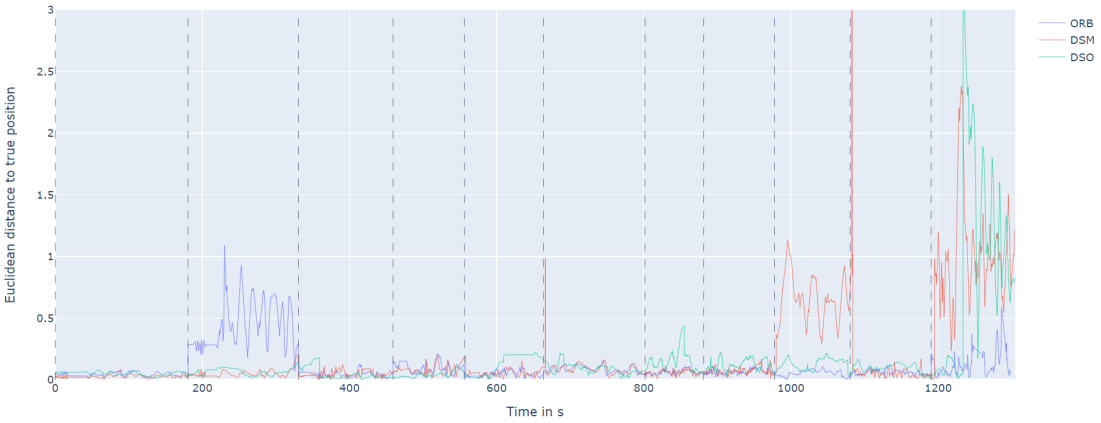
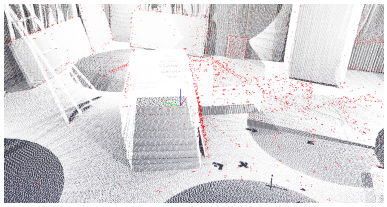


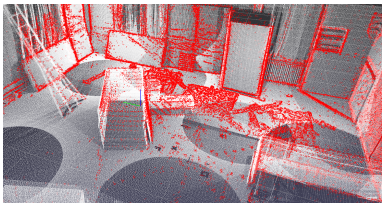
Figure 9: The positional error over time in meters. The vertical lines indicate the beginning of a new sequence

close to 10000 points in the sequences DSM slam generates more than 500000 points in most sequences and DSO more than 200000 in most sequences.

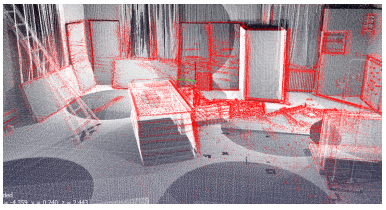
4.3 Calculation Time



(a) ORB



(b) DSM



(c) DSO

Figure 10: 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 wtice as large for better visability (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)

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

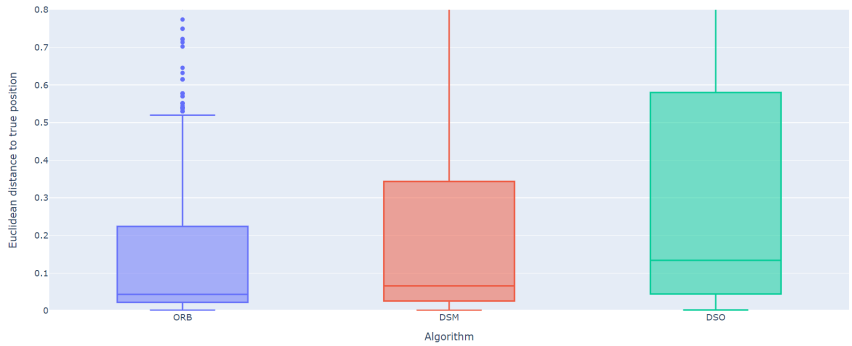
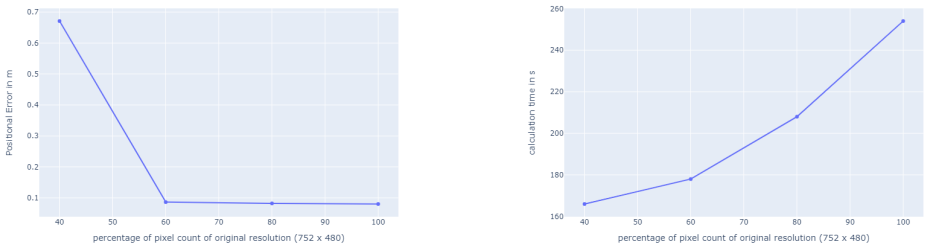


Figure 11: Boxplot of the eucleadian distances between an evaluated point and the closest point of the ground truth point cloud. For computational feasibility, for each sequence and algorithm, 500 points for evaluation are sampled randomly

5. Discussion

5.1 Conclusion of SLAM-Algorithm Evaluation



(a) trajectory error

(b) computation time

Figure 12: Influence of downsizing of the images on the trajectory error (a) and the computation time (b) for the sequence V101.

6. recommendation for further work regarding the full system

Furthemore, a framework to test and build an entire automated system is suggested. This framework includes a simulated envorinment, that realisicly makes it possible, to navigate a drone within an simulated environment. The environment is based on the Roboter Operating System (ROS) and is completely simulated.

The basic idea of the framework can be seen in figure 13. The main parts of the automated exploration system are the SLAM Algorithm and the flight path planning algorithm, that work simultaniously. The general concept is that each of the algorithms takes the output of the other as input.

In the folling section the ROS framework is proposed and described in detail.

6.1 ROS framework

ROS, as the name suggests, is a framework for a software infrastracture 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

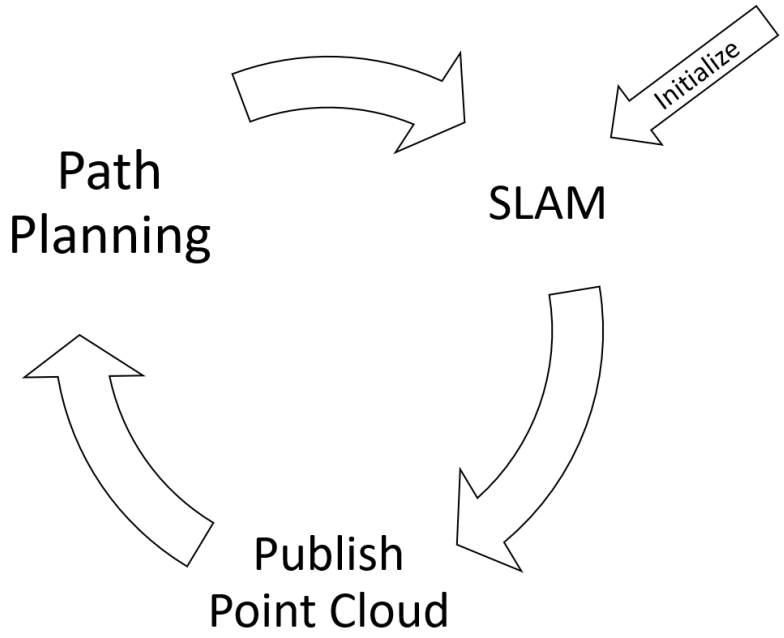


Figure 13: Automated SLAM system

robots regarding their physics.

6.1.1 TUM Simulation Node

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 eqipped with a bottom camera and a frot camera. The cameras each log their output to a ros topic. Additionally, message time stamps, the height sensor output, battery percentage, rotation velocity and accelaration are also logged to rostopics. While the drone can also be navigated using a playstation 3 controller, as shown in figure 15, 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 foreward.

Listing 1: drone navigation command

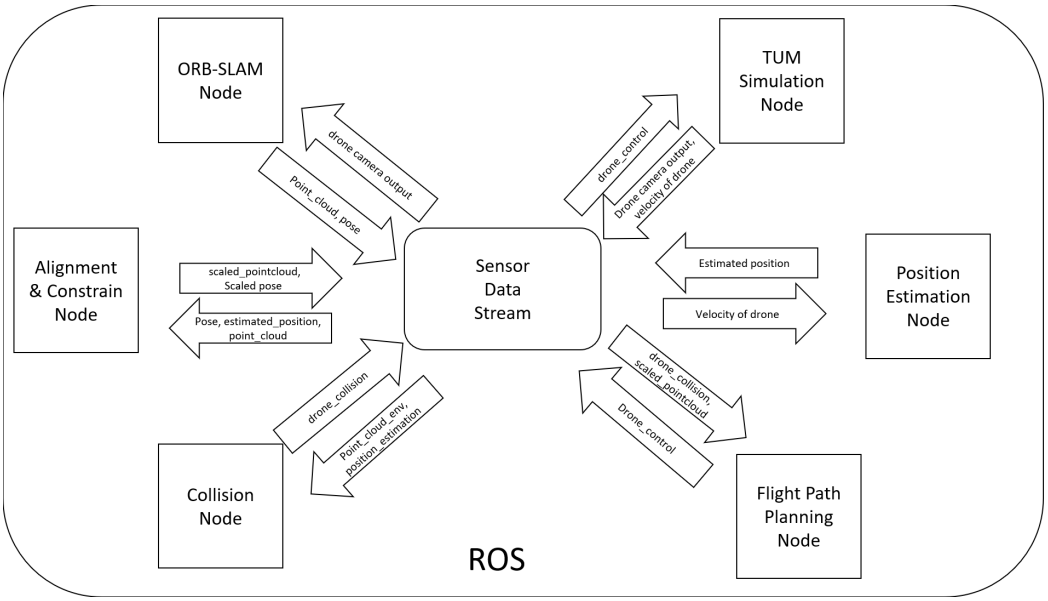


Figure 14: Overview over the suggested ROS framework

```
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}}'
```

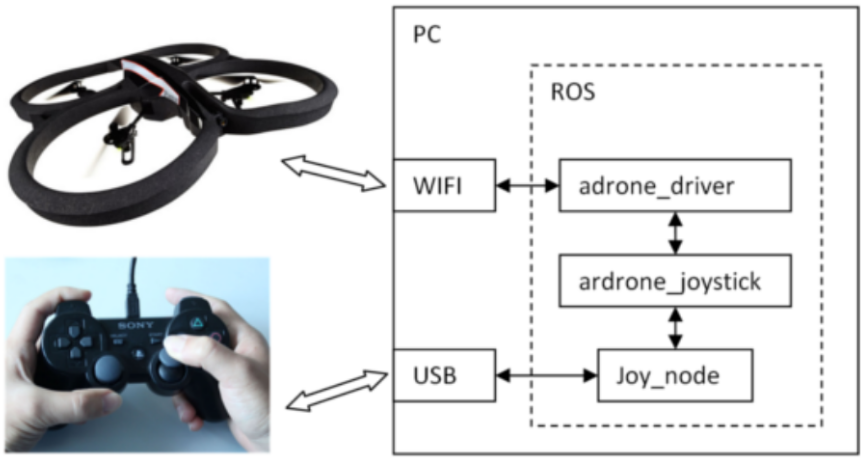


Figure 15: tum simulator setup. Source: *http* : *//wiki.ros.org/tum*$_s$*imulator*

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 lengt, 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.

6.1.2 ORB-SLAM Node

The ORB-SLAM algorithm runs in

Input

Output

6.1.3 Position Estimation Node

The position estimation node appriximates the position of the drone based on the valocity on the drone and the latest position. ORB-SLAM, used in the visual monocular mode is as mentioned not able to extract the true scale of the environment. Estimating the true position enables us to also scale the computed point cloud by the ORB SLAM node to its true scale. This method should be done in the initialization process, by only doing translational movements with the drone, such as a takeoff and a short foreward movement. No rotations should be performed with the drone since the drones navigation data, such as the veloxity, uses the bodyframe as reference frame. Thus doing rotations would result in an incorrect estimation of the position. Folling an overview over the input, output and computational functionality of the node.

Input

The node subscribes to the following topics:

1. /ardrone/navdata

   This topic is beeing published with the tum simulation node described in item **??** and is a messenger class developed for the ardrone. This node only uses the velocity vectors $v_x \in \mathbb{R}, v_y \in \mathbb{R}, v_z \in \mathbb{R}$ given in the unit mms^{-1}.

2. /drone_position_init

   The drone also subscribes to the /drone_position_init topic, published by itself. This topic includes messenges of the class PointStamped. This class includes the x, y and z coordinate of the point itself, and a timestamp. The node also needs the information from this topic, to read in the last position and update it based on the velocity, by doing the computations explained in the following section. The x y and z coordinates are transformed to meter.

Output

The node publishes to the following topics:

1. /drone_position_init

   This topic is explained in the upper section. The updated points are published in this topic

computation

As mentioned, the computation is made based on the current velocity

$$v_i = \begin{pmatrix} v_{i,x} \\ v_{i,y} \\ v_{i,z} \end{pmatrix} \in \mathbb{R}^3$$

and the latest position point

$$x_{i-1} = \begin{pmatrix} x_{i-1,x} \\ x_{i-1,y} \\ x_{i-1,z} \end{pmatrix} \in \mathbb{R}^3$$

. Also, the timedifference in seconds to the last point is extracted, which is easy, since all onject from the class PointStamped can be timestamped. The difference is given by $\Delta t_i = t_i - t_{i-1}$. The updated position is the yielded by

$$x_i = x_{i-1} + \frac{\Delta t_i * v_i}{1000}$$

.

Dividing by 1000 yields the unit meter.

This recursive methodoly is shown in figure 16

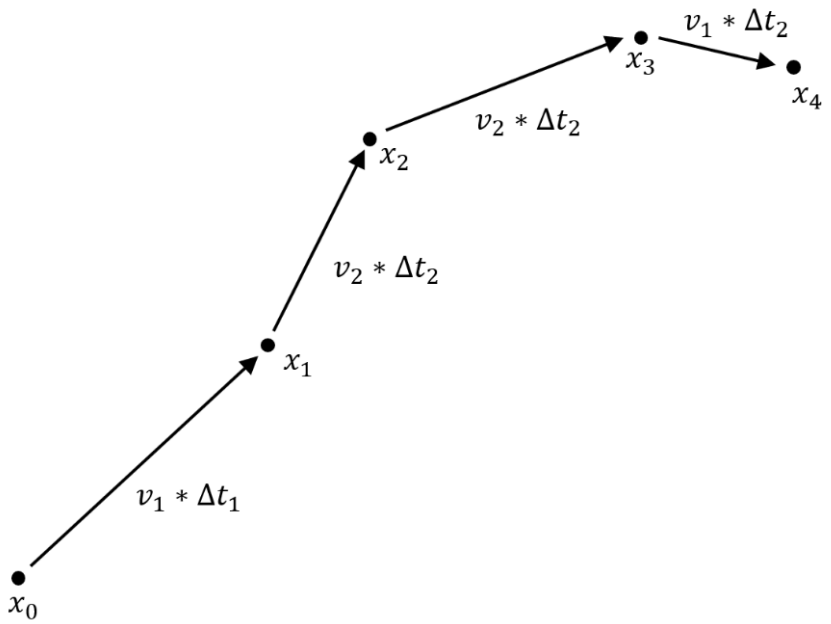


Figure 16: Calculation method for estimation the position in the initialization porcess in order to find the true scale.

The implementation is easily deployed and the update function, that is running in the main loop can be seen in listing 2.

Listing 2: Main part of the position estimation node

```
1  def update_position(self):
2  # get velocity in x, y and z direction
3  x_vel = self.navdata.vz
4  y_vel = self.navdata.vy
5  z_vel = self.navdata.vz
6
7  if x_vel is not None and y_vel is not None and
     z_vel is not None:
8                  # get time difference
9      curr_time = rospy.Time.now()
10     time_diff = (curr_time - self.position.header
         .stamp).to_sec()
11
12                 # create the new point
13     new_point = PointStamped()
14     new_point.header.stamp = curr_time
15     new_point.header.frame_id = "init"
16
17                 # update positions
18     new_point.point.x = self.position.point.x +
         time_diff * x_vel / 1000
19     new_point.point.y = self.position.point.y +
         time_diff * y_vel / 1000
20     new_point.point.z = self.position.point.z +
         time_diff * z_vel / 1000
21
22                 # publish
```

```
23                self.position_publisher.publish(new_point)
24                rospy.sleep(0.1)
```

6.1.4 Scale Recognition Node

Input

Output

6.1.5 Fight Path Planning Node

Input

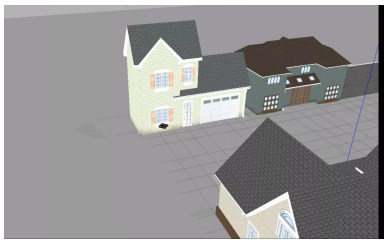
Output

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

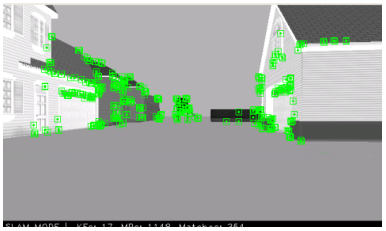
Listing 3: Launching the simulated environment

```
1 # launch the gazebo simulation
```



(a) gazebo simulation

(b) front camera simulation



(c) ORB applied on simulation

Figure 17: 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).

```
roslaunch cvg_sim_gazebo ardrone_testworld.launch

# launch ORB-SLAM
rosrun ORB_SLAM2 Mono ${PATH_TO_VOCABULARY} ${
    PATH_TO_SETTINGS_FILE}

# takeoff with drone
rostopic pub -1 /ardrone/takeoff std_msgs/Empty
```

In listing 3 the commands for launching the gazebo simulation, ORB-SLAM and the drone are displayed. After launching thouse applications, only the path planning algorithm based on the resulting point cloud is missing. However, multiple solutions for such algorithms exist [5], appying it on the system is not part of this paper and will be done in further research.

6.2.1 Known Issues

Bibliography

[1] M. Burri. The euroc micro aerial vehicle datasets. *The International Journal of Robotics Research*, 2016.

[2] H. DURRANT-WHYTE et al. Simultaneous localization and mapping: Part i. 2006.

[3] E. Eade. Lie groups for computer vision. 2002.

[4] J. Engel et al. Direct sparse odometry. 2016.

[5] A. Gasparetto et al. Path planning and trajectory planning algorithms: a general overview. 2015.

[6] B. Hiebert-Treuer. An introduction to robot slam (simultaneous localization and mapping). 2015.

[7] R. Mur-Artal. Orb-slam: a versatile and accurate monocular slam system. *IEEE TRANSACTIONS ON ROBOTICS*, 2015.

[8] E. Rublee. Orb: an efficient alternative to sift or surf. 2012.

[9] R. Smith et al. On the representation and estimation of spatial uncertainty. *The International Journal of Robotics Research*, 1986.

[10] H. Strasdat. Visual slam: Why filter? *Image and Vision Computing*, 2000.

[11] T. Taketomi et al. Visual slam algorithms: a survey from 2010 to 2016. *IPSJ Transactions on Computer Vision andApplications*, 2017.

[12] B. Triggs. Bundle adjustment – a modern synthesis. *International Workshop on Vision Algorithms*, 2000.

[13] S. Umeyama. Least-squares estimation of transformation parameters between two point patterns. *IEEE Trans. Pattern Anal. Mach. Intell.*, 1991.

[14] Z. Zhang et al. A tutorial on quantitative trajectory evaluation for visual(-inertial) odometry. 2009.

[15] J. Zubizarreta et al. Direct sparse mapping. 2019.