# Survey and Experimental Comparison of RGB-D Indoor Robot Navigation Methods Supported by ROS and Their Expansion via Fusion with Wheel Odometry and IMU Data

Florian Spiess, Jonas Friesslich, and Tobias Kaupp
Faculty of Electrical Engineering and Information Technology University of Applied Sciences Wuerzburg - Schweinfurt 97421, Schweinfurt, Germany
Email: {florian.spiess; jonas.friesslich; tobias.kaupp}@fhws.de

Samuel Kounev
Faculty of Mathematics and Computer Science Julius-Maximilians-University of Wuerzburg 97074, Wuerzburg, Germany
Email: samuel.kounev@uni-wuerzburg.de

Norbert Strobel
Institute of Medical Engineering, University of Applied Sciences Wuerzburg - Schweinfurt, 97421, Schweinfurt, Germany
Email: norbert.strobel@fhws.de

***Abstract*—This paper presents an experimental evaluation and comparison of selected Visual Odometry (VO) and Visual-SLAM (V-SLAM) algorithms for indoor mobile robot navigation supported by the Robot Operating System (ROS). The focus is on algorithms involving RGB-D cameras. Since RGB-D cameras integrate color and depth information, they output coherent measurement data and facilitate an efficient processing pipeline. The various underlying methods of vision-based algorithms are described and evaluated on two datasets covering different indoor situations as well as various lighting and movement conditions. In general, V-SLAM algorithms yielded better results. They were superior with respect to handling drift, in particular when loop closures were involved. However, the results confirmed that VO algorithms could outperform V-SLAM methods under certain circumstances. This happened when there was a very good match between an algorithm's design objectives and the situation at hand. While the experiments showed that there is no single best algorithm for every scenario, ORB-SLAM2 is recommended as a robust stand-alone RGB-D based localization method available under ROS. Furthermore, we observed that the position estimation error could be reduced by around 67% on average when combining vision-based position estimates with sensor data obtained from wheel odometry and an inertial measurement unit (IMU), respectively. This clearly demonstrates the potential of sensor fusion techniques. The best results in case of sensor fusion were obtained with RGB-DSLAMv2.**

***Index Terms*— mobile robots, multisensor data fusion, data sets for robotic vision, RGB-D perception**

## I. Introduction

A mobile robot's capability to localize itself is an essential prerequisite for navigation and path planning. In this paper, *localization* describes the determination of a mobile robot's position without considering its orientation. The most widespread and cost-efficient localization method for mobile robots is wheel odometry. Almost all wheeled robots use it. However, the accuracy of the resulting position estimation is limited due to factors such as a finite wheel encoder resolution or slippage of the wheels relative to the ground. Due to the accumulation of errors, wheel odometry gets more unreliable with increasing path length. Another way to keep track of an object's position is to rely on camera information. In this context, it was Nister et al. [1] who proposed the term "Visual Odometry" to describe the estimation of camera motion from consecutive images. With the increasing popularity of commercially available, affordable camera modules integrating a depth sensor such as the Microsoft Kinect series or the Intel Realsense cameras, their use for visual odometry has increased substantially. These cameras provide RGB-D output streams that can be processed to compute camera motions. In addition to VO algorithms, there are Visual-SLAM methods. Whereas VO algorithms operate on a sequence of successive images to estimate motion, V-SLAM methods use the input images to generate a persistent representation of the surroundings. When constructing these global maps of the environment, V-SLAM methods usually benefit

Manuscript received March 11, 2020; revised November 1, 2020.


doi: 10.18178/ijmerr.9.12.1532-1540

from loop closures. They can be obtained by revisiting previously observed areas. Loop closure limits the drift of the camera path by readjusting position estimates. In other words, while VO aims for locally consistent movements, V-SLAM's goal is a globally consistent trajectory [2]. However, the increased robustness of V-SLAM algorithms comes at a price. They are in general more computationally expensive which can be problematic for mobile devices. In this paper, VO and V-SLAM position estimation methods are compared. The algorithms considered are listed in Table I. The underlying methods are further described in Section II. As wheel odometry and vision-based position estimation techniques are complementary [3], we also evaluated if fusion of these two can improve overall performance. In addition, data from a built-in inertial measurement unit (IMU) was included. The various sensor outputs were fused using an Extended Kalman Filter (EKF) to improve both the accuracy and the robustness of the state vector estimate [4]. For the experiments, a Mecanum-wheel-based research platform was used as shown Fig. 1. Attached to it was an Intel Realsense D435i [5] stereo depth camera. The platform was moved around inside and outside of the lab facilities to acquire datasets. They are made publicly available to add diversity to the current data pool. The contributions of this paper are:

1) Classification and experimental comparison of the most popular VO and V-SLAM algorithms supported under ROS using RGB-D camera data.
2) Experimental comparison of their performance with and without wheel odometry / IMU for various indoor navigation scenarios.
3) Demonstrating an improvement of the VO / V-SLAM position estimation results by 67% on average after applying EKF data fusion with wheel odometry readings and IMU data.
4) A benchmark dataset to facilitate research on sensor fusion comprising camera images as well as associated wheel odometry data and IMU measurements in conjunction with a suitably tuned parameter set for the algorithms listed in Table I; no such multi-sensor dataset has been previously made available.

TABLE I. OVERVIEW OF THE ALGORITHMS CONSIDERED IN THIS PAPER

| Algorithm | Category | Match | Method |
|---|---|---|---|
| RGB-DOdometry | VO | frame | image-based, dense |
| DVO [7] | VO | frame | image-based, dense with sensor and motion model |
| Fovis [8] | VO | keyframe | image-based, sparse |
| ICPOdometry [9] | VO | frame | depth-based, dense |
| RGB-DICPOdometry[10] | VO | frame | hybrid, dense joint-optimization |
| CCNY [11] | VO | model | hybrid, sparse 3D points |
| DVOSLAM [12] | V-SLAM | map | hybrid, dense |
| RTABMap [13] | V-SLAM | map | image-based, dense |
| ORB-SLAM2 [14] | V-SLAM | map | image-based, sparse |
| RGB-DSLAMv2 | V-SLAM | map | image-based, sparse |

From the vast amount of available algorithms, a representative set was chosen, listed in Table I. Two different motion trajectories were recorded in a laboratory environment using a Mecanum-wheeled robot which facilitates precise (omnidirectional) movements along straight paths as well as on curved tracks. The trajectories were set up such that the impact of different velocities and turn rates on visual path estimation methods could be studied. The remainder of the paper is organized as follows: Section II presents previous work on comparing visual odometry methods. Section III discusses the algorithms which were used to estimate the successive RGB-D camera positions. Experimental results can be found in Section IV. In the last Section, results are discussed and conclusions are drawn.

## II. RELATED WORK

Adopting the taxonomy proposed by [2], VO and V-SLAM methods fall into three categories depending on the type of data used to perform camera position estimation: image-based methods, depth-based methods, and hybrid methods. Those algorithms are subdivided into sparse and dense approaches. Sparse approaches operate on selected image features in 2D or 3D; dense approaches, involve all 2D pixels or 3D point cloud elements. In general, VO methods estimate the camera motion frame-to-frame, frame-to-keyframe, or frame-to-model. Frame-to-frame algorithms only align consecutive frames. The frame-to-keyframe approach takes the first RGB-D image of a sequence as a keyframe to which the consecutive frames are matched for motion estimation. If a significant drift is detected, for example based on a reduction of the number of matches, a new keyframe is selected. Frame-to-model strategies build a model / local map of the environment and match each new frame against it. This makes those algorithms somewhat similar to V-SLAM, as they provide the ability to relocalize should motion estimation fail in some instance. The main difference between frame-to-model VO and V-SLAM is that the VO model only has a limited size to keep the complexity under control. As a consequence, parts of the model are discarded once a maximum number of components is reached. In fast-motion scenarios frame-to-model algorithms can be more precise than their frame-to-frame counterparts because the model allows feature matching not only between consecutive frames but over the whole recorded scenario [2]. Furthermore, after the initial matching step a local optimization process can be run. This makes it possible to match a new frame not only to the latest one but also to multiple previously recorded frames or even multiple keyframes to arrive at a better position estimate minimizing drift. If all previously collected frames are considered, a VO algorithm becomes a V-SLAM technique. In case the matching process led to enough correspondence, V-SLAM algorithms can perform loop closure reducing the accumulated position error drastically. In their comparison [2], Fang and Zhang considered Fovis, DVO, MRSMAP [16], a proprietary algorithm developed by Occipital [17] and the three algorithms from the

OpenCV rgbd-module. The latter are evaluated in this paper again. OpenCV is an open programming library for image processing. All approaches were tested on the TUM dataset [18]. The authors found that image-based or hybrid methods perform better and are more robust than depth-based methods when the environment has rich visual features and the illumination is good. Under these conditions, Fovis, an image-based method utilizing sparse features, was the best choice for accuracy and speed. Under bad illumination conditions dense approaches were found to be superior. In case of featureless environments, depth-based methods were recommended. Additionally a degrading effect was observed if the motion between consecutive images was fast, or solely rotational. In [19], the authors tested and analyzed the performance of selected visual odometry algorithms designed for RGB-D sensors on the TUM dataset with respect to accuracy, time, and memory consumption. In contrast, this paper compared their performance with V-SLAM algorithms and considered data fusion. Similar to [2], they also observed that the performance of the algorithms under investigation strongly depended on the scene characteristics. They concluded that the image-based methods (e.g., Fovis) and the hybrid methods were most accurate when the environment had texture but no structure. In structured but low textured scenes, hybrid and depth-based methods were found to be preferable. The authors of [20] generated several synthetic datasets by moving a camera around various 3D virtual indoor environments containing structure but with little textures. Based on these simulated datasets, algorithms from the image and depth-based category were studied. Fovis, DVO, KinectFusion [21], RGB-D [1], and ICP+RGB-D [22] were evaluated. On noise-free rendered images, ICP, a depth-based method, estimated the camera trajectory best overall. From their work it can be concluded that ICP algorithms may perform best when datasets are mostly characterized by geometric features and little noise. In the more realistic simulation including noise, techniques such as DVO enforcing a photo-consistency constraint were more robust. In our work we use real environment setups , and comparisons were not only provided for VO but also for V-SLAM algorithms. The accuracy of several image-based and depth-based algorithms including Fovis, DVO was evaluated in [23]. Although the work of A. Handa et al. was targeted at micro aerial vehicles (MAVs), while our focus is on mobile ground robots and considers sensor fusion as well as V-SLAM, the authors pointed out several interesting observations relevant to our work. On the whole, image-based methods (such as Fovis and DVO) were found to be faster and more accurate than depth-based methods when the environments were well lit and had good texture features. Depth-based methods were recommended for dark environments, but they suffered from the rather short sensing range of the RGB-D camera used and the very noisy depth measurements. Ten of the most promising open source packages for vision- based state estimation algorithms were evaluated in [24]. Although there is no overlap between the V-SLAM algorithms covered in that paper and our work as different sensors were used, there are some lessons to be learned. First, one of the main challenges when comparing different algorithms is to configure their parameters appropriately. Second, the type of input images is another important factor influencing the results, as the performance of VO and V-SLAM techniques depend on the scene content, different illumination, and the presence of blur, caused by out-of-focus positioning as well as motion blur. Furthermore, some packages yielded different results for successive runs on the same dataset with the same parameter settings when real-time constraints were enforced. It turned out that this was caused by frame dropping. Various benchmark datasets for assessing visual odometry algorithms have been made available.


Figure 1. Mobile research platform from evocortex GmbH, Nuremberg, Germany [25]

They include the TUM RGB-D benchmark [19], EuroSec [26], or ICL-NUIM [21]. However, none of these datasets were collected with sensor fusion in mind. As a consequence, they neither include any wheel odometry data, nor do they provide any IMU measurements. The provided benchmark dataset improves this situation, comprising camera data as well as other sensor recordings to enable future research on different fusion approaches.

## III. Tested VO and V-SLAM Algorithms

In this section the algorithms listed in Table I are briefly dis-cussed. The different algorithms were chosen to provide a broad overview of the different methods for visual position estimation, comprising different methods for feature handling and matching with underlying data models (frame, keyframe, model, map). Furthermore by comparing ORB-SLAM2 and RGB-DSLAMv2, we analyzed if algorithms based on the same matching strategy and methodology had similar performance. RGB-DOdometry used an implementation described in [1]. [1] For every pixel in the RGB image, the corresponding intensity value (gray value) is calculated first. Then the corresponding positions of all pixels in the scene are computed. This can be interpreted as a surface of the observed area. In the next step, the camera motion

[1] https://github.com/opencv/opencv_contrib/blob/master/modules/rgbd/src/odometry.cpp

is derived based on minimizing the error (energy) between the intensity values of the motion-warped current surface and the surface of the previous image. DVO (Dense Visual Odometry) is an improved version of [1].[2] As in [1], the camera motion is estimated by enforcing the photo-consistency constraint between two consecutive images. However, unlike [1], the residual can be weighted based on a sensor model, which copes better with outliers. A constant motion model is further applied when calculating the transformation reducing the likelihood of jumps when estimating the camera motion. Fovis applies the FAST keypoint matching algorithm to detect features in the RGB image.[3] Afterwards, depth information is included as a third coordinate to those 2D positions were features were found. By comparing their feature descriptor values, features are then matched across frames using a mutual consistency check. Additionally, a keyframe technique can be used to reduce short-scale drift. ICPOdometry from OpenCV is based on KinectFusion [21].[1] To calculate the camera motion, this algorithm performs an ICP of all 3D depth points of the current image to align them with the previous depth frame. In the original version of KinectFusion the matching was applied not only to the previous frame but also to a model created from the data. OpenCV's RGB-DICPOdometry uses both algorithms, that is RGB-DOdometry and ICPOdometry.[1] It tries to find a compromise by minimizing the sum of both energy functions [5]. The mechanism is the same as introduced in [22]. CCNY RGB-D tools algorithm, abbreviated as CCNY, first utilizes one of its keypoint detector algorithms to identify keypoints in the intensity channel of the RGB image.[4] The next step is to calculate the uncertainty of the location of every point together with an average mean value for its possible location. Afterwards, the mean value and the uncertainties are compared against a model built from the previously observed features. An ICP algorithm is applied to find the transformation aligning the new set of features to the model. A Kalman Filter is subsequently used to update stored features and to add new ones to the model (up to a limit). DVOSLAM builds on top of DVO and uses a method to optimize the registration based on intensity and depth error. [5]An entropy-based method is used to select keyframes. This decreases the drift significantly. The same entropy metric is applied to validate loop closures. By applying a general graph SLAM solver, the position accuracy is increased. RTABMap evaluates RGB-D data to calculate camera motion.[6] Keypoints are detected and processed based on a combination of GoodFeaturesToTrack and the BRIEF algorithm. A nearest neighbor distance ratio test is applied to match the features against the feature map. The feature map itself is comprised of 3D features including their descriptors from latest keyframes. The location of the map features in the current frame is predicted using a motion model. The motion between the current frame and the feature map is then computed from the correspondences of the features from current frame and the feature map, applying OpenCV's Perspective-n-Point RANSAC. Whenever the number of keypoint matches falls below a threshold, the feature map is updated by including unknown features and adjusting positions of the previous ones. The amount of map features is limited, causing old features to be dropped [27]. The loop closure is based on a bag-of-words approach. It utilizes a subset of the features of the current frame and matches them to the feature map. ORB-SLAM2 employs multiple parallel threads to compute the camera position.[7] One thread calculates the position and orientation of the camera for each incoming frame by computing ORB feature correspondences between the current image and the local map using motion-only bundle adjustment. The map is then optimized by local bundle adjustment. A loop closure algorithm based on DBoW2 [28] is used to detect large loops and correct accumulated drift by optimizing the pose graph. Upon completion of these steps, a fourth thread is launched, that performs a full bundle adjustment to reconstruct the 3D camera trajectory. RGB-DSLAMv2 uses 3D landmark positions as features.[8] These features are compared across consecutive frames as well as to a list of keyframes. The results are further processed by RANSAC [29] to remove false positive matches. To verify the transformation estimate, the dense free-space information from the depth data is exploited. The list of keyframes is expanded with the new frame if there are not enough matches. Furthermore this list is used to detect loop closure. Upon receiving an initial pose graph from the SLAM front-end, g2o, which is a framework for graph optimization, is used to minimize errors of the provided graph and to compute an optimal trajectory.

## IV. Experiments and Analysis

### A. Hardware

The experiments were conducted using the indoor mobile robot shown in Fig. 1. An Intel Realsense D435i camera mounted on the robot's front was used to record the RGB-D data with a resolution of 1280x720 pixels and a frame rate of 26 fps. Processing involved the realsense2 *camera* package [30] and the Intel RealSense SDK 2.0 [31]. Before running the experiments, a map of the test environment was created using the robot's SICK TIM561 2D LiDAR. This map was used in combination with the Evocortex

[2] https://github.com/songuke/_dvo slam
[3] https://github.com/srv/fovis
[4] https://github.com/ccny-ros-pkg/ccny_ rgbd_ tools
[5] https://github.com/songuke/dvo_slam
[6] https://github.com/introlab/rtabmap
[7] https://github.com/appliedAI-Initiative/orb_ slam_ 2_ ros
[8] https://github.com/felixendres/rgbdslam _v2

Localizer SDK to obtain ground truth positions. The Evocortex Localizer SDK is reported to reach an accuracy of approximately 0.1% with respect to the distance traveled [32]. Since this in an order of magnitude better than typical results achieved with camera-based approaches, the positions returned by the Evocortex Localizer SDK can be considered as ground truth. All data was recorded on a laptop running Ubuntu 16.04 linked to the mobile robot via Ethernet. The evaluation was performed on a PC consisting of the following components: Threadripper 1950X, 64 GB RAM, 500 GB SSD, and Asus GeForce GTX 1080 TI.

### B. Software and EKF

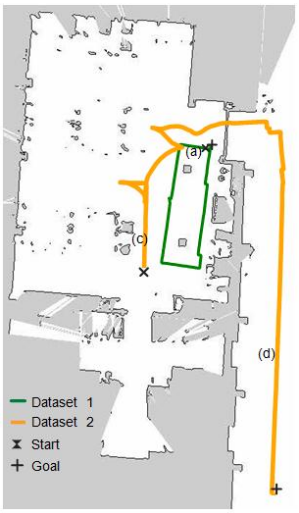
The ROS Kinetic Kame framework running on Ubuntu 16.04 LTS was used. The evaluated libraries were ported to ROS Kinetic, if necessary. All software packages except the three OpenCV packages were built with OpenCV version 3.3.1 as this shipped with ROS. The RGB-DICPOdometry, ICPOdometry and RGB-DOdometry components were built on OpenCV version 4.1.2. For those packages, the data was provided via the ROS-OpenCV interface and the output was converted to a ROS message. Since the aim was to simulate real-time conditions, the algorithms and their position estimates were evaluated on live data without applying any post-processing. As mentioned in [12], the parameter selection had a great influence on the performance as well as on the runtime of the algorithms. To arrive at a fair comparison, the parameters were individually tuned for each method. It was found that tuning could only be done meaningfully if no frames were skipped. Otherwise, the algorithms' performance was not stable, making it impossible to find an optimal parameter configuration. The parameter settings can be found in the Github repository.[9] To ensure that no frames were skipped, the recorded datasets were replayed with a slower data rate. A frame-process-rate was calculated by dividing the frame rate with the inverse of the slowdown-rate. It can be interpreted as a measure for the real-time performance of the algorithm. For each algorithm the frame-process-rate parameter is given in Table II. The parameters for each algorithm were tuned on dataset 1 and then also applied to the other dataset. The output of the visual motion estimation algorithms was fused with the odometry data of the mobile unit and IMU data coming from a sensor built into the Realsense camera. To this end, the *robot pose ekf* package was used.[10] As the robot only performs 2D-motion the underlying state vector consisted of ($x, y, yaw, v_x, v_y, v_{yaw}, a_x, a_y, a_{yaw}$), where yaw describes the rotation about the z axis. The variables $x$ and $y$ were measured in meters, while *yaw* was expressed in radians.

(a) Dataset 1 : Start position

(b) Dataset 1 : Middle part

(c) Dataset 2 : Laboratory

(d) Dataset 2 : Corridor




(e) Map of the laboratory environment. The respective positions at which the images 2a,2b,2c,2d were taken, are marked with identical letters.

Figure 2. Example pictures for dataset 1 & dataset 2 and the laboratory environment.

The EKF was set up as follows: From VO / V-SLAM the values of ($x, y, yaw$) were used as inputs to the *robot pose ekf* package. There, these values were first converted to their respective velocities ($v_x, v_y, v_{yaw}$) before passing them on to the EKF. The reason for this configuration was as follows: $x, y, yaw$ suffer from drift and therefore have increasing covariance values. For their derivatives, $v_x, v_y, v_{yaw}$, constant covariances can be assumed. Since most of the evaluated algorithms did not provide covariance matrices for their outputs, diagonal covariance matrices with fixed values for all algorithms were applied. A higher value for the yaw's variance was chosen to account for the higher rotational error relative to the translations. A good example for rather inaccurate yaw estimates can be found in Fig. 3a by looking at the CCNY trajectory. Similar to the position data of VO / V-SLAM algorithms, the wheel odometry data was converted to velocity values as well before being passed to the EKF. Again, a diagonal covariance matrix was assumed, but the main diagonal elements were set to a third of their VO / V-SLAM counterparts to account for the higher precision provided by the wheel odometry sensors. The IMU data was preprocessed using the *imu filter madgwick*, and $v_{yaw}$, $a_x$, $a_y$ were considered for

[9] https://github.com/Fugashu/compVisualPose/blob/master/ParameterOverview.pdf

[10] https://github.com/ros-planning/robot_pose_ekf

sensor fusion.[11] Note that no *x*, *y*, and *yaw* values were used for fusion directly. Only their derivatives were. As an initial value, the robot's starting position was used.

### C. Datasets

Fig. 2e shows an overview of the laboratory environment including the positions along the track where the pictures shown were taken. Dataset 1 was recorded with optimal conditions for VO / V-SLAM. This dataset was used for parameter tuning parameters as well as for evaluating how well the algorithms may perform. Dataset 2 was set up as a challenge for the VO or V-SLAM algorithms. While the first part was similar to the first dataset with respect to motion and environment, the second part, a hallway, was very different. It comprised only small structures and little texture was present. In the following the datasets are described in more detail. Dataset 1 is comprised of linear movements along a rectangular trajectory inside the robotics lab of the University of Applied Sciences Wuerzburg-Schweinfurt. When recording the first dataset, the robot was driven on a rectangular course without fast movements (max. velocity $0.25\,\frac{m}{s}$). Only minor rotational movements were carried out. This led to only small differences between consecutive images, thus, ensuring preferable conditions for VO / V-SLAM methods [12]. Recording was done in a typical lab environment characterized by chairs, tables, some robots, and lab materials arranged alongside the walls. It can be described as rich both in structures (corners, planes) and in textures. The experiment was performed under constant lightning conditions (ceiling lights). As this was a collision-free trajectory, the image center focused on fixed objects straight ahead, for example, two chairs and a table (see Fig. 2a). More objects rich in texture and structure were visible to the left and to the right (see Fig. 2b). During the experiment, the robot was carefully observed ensuring that only minor wheel slippage occurred to obtain very precise wheel odometry data. Dataset 2 also started inside the laboratory (see Fig. 2c), then exited through a door and utilized the Mecanum wheels to drive the robot parallel to a wall of the hallway with the RGB-D camera facing the wall (see Fig. 2d). The motions in negative x-direction were carried out to turn the robot around inside the lab so that the camera would face in negative x-direction to look at the hallway wall, as soon as it exited the door. This second run was significantly longer than the first with a maximum velocity of $0.29\,\frac{m}{s}$. The wall itself resembled a white plane, i.e., there was very little texture and not much variation in structure with the exception of two doors. The data was recorded as ROS bags containing the color (RGB), depth, camera info and IMU topics of the camera as well as the odometry of the robot. The ground truth was available through the *tf* topic and could be accessed through *frame_ id /map* and *child_frame_ id /pose_ localizer* messages. The download links of the datasets can be found on the Github page.[9]

[11] https://github.com/ccny-ros-pkg/imu_tools

### D. Evaluation Metric

The relative position error metric from [19] was applied to evaluate the relative translation error RMSE (rt), with a time interval of $\Delta t$ = 1 s between two compared positions. Since the estimated trajectory was transformed into the map coordinate system, only the relative trajectory error from [19] was used To calculate the absolute trajectory error, we proceeded as follows. The ground truth position $P(t)$ at time $t$ was computed using the ground truth position data $P(\hat{t}_1)$ and $P(\hat{t}_2)$ according to

$$P(t) = a \cdot P(\hat{t}_2) + (1 - a) \cdot P(\hat{t}_1) \quad (1)$$

The weighting factor $a$ was calculated as

$$a = \frac{t - \hat{t}_1}{\hat{t}_2 - \hat{t}_1} \quad (2)$$

In this equation $\hat{t}_1$ and $\hat{t}_2$ are the closest time stamps of the ground truth data right before and after the time stamp $t$ of the estimated camera position. The error $E(t)$ at time $t$ is the difference between the position data of the estimated trajectory, $Q(t)$, and the associated (interpolated) ground truth position, $P(t)$. It can be expressed as:

$$E(t) = Q(t) - P(t) \quad (3)$$

Finally, the absolution translation RMSE(at) follows as

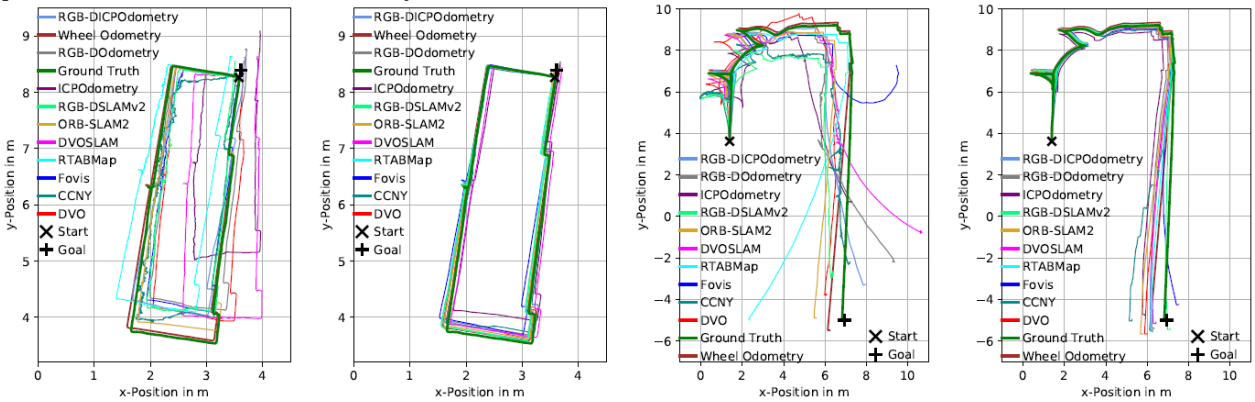
$$RMSE(E_{0:n}) := \left(\frac{1}{n}\sum_{i=0}^{n}\|E(t_i)\|^2\right)^{\frac{1}{2}} \quad (4)$$

Here, $n$ is the number of data points in the estimated trajectory, $t_i$ is the i-th timestamp of the estimated trajectory, and $E(t_i)$ is the error at the specific time.

### E. Evaluation

Besides performing a quantitative analysis, two-dimensional plots of the results are provided to offer additional insights. Fig. 3a and 3b show the experimental results for the first dataset before and after fusion with wheel odometry and IMU. In Table II, the RMSEs regarding the absolute translation and relative translation, as well as the Maximum, Mean, Median and Standard deviation of the translational error are listed before and after fusing with odometry and IMU, respectively. From the trajectories (Fig. 3d), it is evident that after fusion the paths are much closer to the ground truth. The results show a significantly lower RMSE (absolute trajectory (at)) and RMSE (relative trajectory (rt)) after fusion for all algorithms in both datasets. However, even after data fusion, wheel odometry was still superior in almost all cases. There are two reasons for this. First the tracks did not involve problematic situations for the wheel encoders such as slippage. The other reason is that even the second track with a length of 32 m is still rather short. This is why the accumulated drift of the wheel odometry was still low. Also note that when comparing the RMSE (rt) results, no VO of V-SLAM algorithm can reach the accuracy of wheel odometry, because the wheel odometry's short term drift is far lower than any accuracy achievable by

visual algorithms, if no special circumstances such as slippage or bumps cause problems. The results show that over short distances under good operating conditions, wheel odometry is very hard to beat. However, over longer distances, fusion with V-SLAM algorithms can help to keep wheel odometry's drift at bay as shown in dataset 2 for RGB-DSLAMv2. In this case, the maximum error after fusion reduced from 0.864 m to 0.484 m, i.e., by about 44%. The reason for the good performance was that the drift early on the track of RGB-DSLAMv2 was compensated by the wheel odometry while the drift in the last 30% of the track of the wheel odometry was compensated by RGB-DSLAMv2. This is an ideal case to demonstrate the benefits of senor fusion. Unlike V-SLAM methods, VO algorithms lack the ability to reduce drift. As a consequence, data fusion with VO techniques can at most serve as a fall-back when wheel odometry fails, e.g., in case of slippage.




(a) Dataset 1: Before fusion (b) Dataset 1: After fusion (c) Dataset 2: Before fusion (d) Dataset 2: After fusion

Figure 3. Comparison of vision based motion estimation algorithms before (Fig. 3a / Fig. 3c) and after fusion with odometry and IMU (Fig. 3b / Fig. 3d) on Dataset 1 / 2.

TABLE II. RMSE OF ABSOLUTE TRANSLATION(AT), RMSE OF RELATIVE TRANSLATION(RT),MAXIMUM TRANSLATION ERROR, MEAN TRANSLATION ERROR, MEDIAN TRANSLATION ERROR, STANDARD DEVIATION OF THE TRANSLATION ERROR BEFORE AND AFTER FUSION WITH EKF FOR DATASET 1 AND 2 (IN METERS). CORRESPONDING FRAME-PROCESS-RATES(FPS) IN HZ ARE SHOWN ON THE VERY RIGHT WITH VALUES CLOSER TO 26 INDICATE REAL-TIME PERFORMANCE.

| | Dataset 1 | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Algorithm | RMSE(at) | $RMSE(at)_{fused}$ | RMSE(rt) | $RMSE(rt)_{fused}$ | Max | $Max_{fused}$ | Mean | $Mean_{fused}$ | Median | $Median_{fused}$ | Std | $Std_{fused}$ | fps | rank |
| Wheel Odometry | 0.049 | - | 0.007 | - | 0.096 | - | 0.042 | - | 0.033 | - | 0.026 | - | - | |
| RGB-DOdometry | 0.458 | 0.106 (-76.75%) | 0.036 | 0.010 (-71.38%) | 0.686 | 0.191 (-72.11%) | 0.415 | 0.093 (-77.57%) | 0.446 | 0.101 (-77.23%) | 0.194 | 0.052 (-73.32%) | 2.6 | 7 |
| DVO | 0.373 | 0.074 (-80.14%) | 0.025 | 0.010 (-58.74%) | 0.579 | 0.148 (-74.49%) | 0.336 | 0.062 (-81.52%) | 0.358 | 0.056 (-84.30%) | 0.163 | 0.041 (-75.10%) | 6.5 | 5 |
| Fovis | 0.379 | 0.149 (-60.75%) | 0.040 | 0.012 (-70.63%) | 0.633 | 0.276 (-56.34%) | 0.331 | 0.127 (-61.72%) | 0.336 | 0.123 (-63.51%) | 0.185 | 0.078 (-57.80%) | 2.6 | 6 |
| ICPOdometry | 1.208 | 0.260 (-78.49%) | 0.071 | 0.017 (-76.73%) | 1.801 | 0.425 (-76.42%) | 1.114 | 0.235 (-78.93%) | 1.127 | 0.230 (-79.64%) | 0.468 | 0.112 (-76.15%) | 3.2 | 10 |
| RGB-DICPOdometry | 0.358 | 0.103 (-71.25%) | 0.029 | 0.010 (-65.19%) | 0.558 | 0.198 (-64.58%) | 0.326 | 0.088 (-72.99%) | 0.344 | 0.085 (-75.30%) | 0.149 | 0.054 (-64.07%) | 2.1 | 4 |
| CCNY | 0.246 | 0.108 (-56.25%) | 0.072 | 0.017 (-76.85%) | 0.491 | 0.218 (-55.55%) | 0.208 | 0.092 (-55.89%) | 0.173 | 0.083 (-52.11%) | 0.131 | 0.056 (-57.17%) | 20.8 | 2 |
| DVOSLAM | 0.685 | 0.107 (-84.43%) | 0.038 | 0.012 (-69.03%) | 0.973 | 0.146 (-84.96%) | 0.638 | 0.102 (-84.06%) | 0.664 | 0.106 (-84.09%) | 0.250 | 0.032 (-87.09%) | 3.9 | 9 |
| RTABMap | 0.461 | 0.074 (-84.03%) | 0.023 | 0.008 (-64.69%) | 0.695 | 0.136 (-80.46%) | 0.416 | 0.065 (-84.28%) | 0.450 | 0.065 (-85.56%) | 0.200 | 0.034 (-82.97%) | 6.5 | 8 |
| ORB-SLAM2 | 0.135 | 0.066 (-51.20%) | 0.017 | 0.009 (-47.62%) | 0.234 | 0.124 (-47.02%) | 0.115 | 0.057 (-50.59%) | 0.119 | 0.057 (-52.26%) | 0.072 | 0.034 (-52.81%) | 2.6 | 1 |
| RGB-DSLAMv2 | 0.323 | 0.070 (-78.24%) | 0.038 | 0.009 (-76.57%) | 0.570 | 0.139 (-75.65%) | 0.283 | 0.060 (-78.95%) | 0.269 | 0.060 (-77.52%) | 0.156 | 0.037 (-76.06%) | 26.0 | 3 |
| | Dataset 2 | | | | | | | | | | | | | |
| Algorithm | RMSE(at) | $RMSE(at)_{fused}$ | RMSE(rt) | $RMSE(rt)_{fused}$ | Max | $Max_{fused}$ | Mean | $Mean_{fused}$ | Median | $Median_{fused}$ | Std | $Std_{fused}$ | fps | rank |
| Wheel Odometry | 0.321 | - | 0.010 | - | 0.864 | - | 0.240 | - | 0.155 | - | 0.214 | - | - | |
| RGB-DOdometry | 1.335 | 0.442 (-66.93%) | 0.066 | 0.021 (-67.71%) | 3.849 | 0.844 (-78.07%) | 1.046 | 0.343 (-67.20%) | 0.899 | 0.241 (-73.21%) | 0.831 | 0.278 (-66.50%) | 2.6 | 4 |
| DVO | 1.286 | 0.464 (-63.93%) | 0.038 | 0.014 (-61.81%) | 1.993 | 1.177 (-40.91%) | 1.175 | 0.308 (-73.81%) | 1.273 | 0.139 (-89.12%) | 0.524 | 0.347 (-33.68%) | 6.5 | 3 |
| Fovis | 3.719 | 0.320 (-91.41%) | 0.068 | 0.015 (-77.07%) | 12.535 | 0.959 (-92.35%) | 1.955 | 0.272 (-86.08%) | 0.662 | 0.238 (-64.13%) | 3.164 | 0.167 (-94.71%) | 2.6 | 10 |
| ICPOdometry | 2.447 | 0.598 (-75.54%) | 0.074 | 0.019 (-74.48%) | 5.733 | 1.143 (-80.06%) | 2.023 | 0.478 (-76.38%) | 2.044 | 0.413 (-79.80%) | 1.376 | 0.360 (-73.80%) | 3.2 | 9 |
| RGB-DICPOdometry | 0.952 | 0.401 (-57.86%) | 0.044 | 0.015 (-67.22%) | 1.996 | 0.865 (-56.68%) | 0.817 | 0.294 (-64.03%) | 0.787 | 0.161 (-79.59%) | 0.487 | 0.273 (-44.06%) | 2.1 | 2 |
| CCNY | 1.394 | 0.628 (-54.92%) | 0.116 | 0.032 (-71.91%) | 2.808 | 1.703 (-39.37%) | 1.257 | 0.392 (-68.83%) | 1.256 | 0.207 (-83.56%) | 0.602 | 0.491 (-18.43%) | 20.8 | 7 |
| DVOSLAM | 1.628 | 0.355 (-78.23%) | 0.050 | 0.015 (-68.82%) | 5.677 | 0.765 (-86.53%) | 1.113 | 0.259 (-76.75%) | 0.894 | 0.141 (-84.18%) | 1.189 | 0.242 (-79.61%) | 3.9 | 8 |
| RTABMap | 1.387 | 0.370 (-73.31%) | 0.025 | 0.011 (-58.01%) | 4.584 | 0.859 (-81.25%) | 0.831 | 0.262 (-68.46%) | 0.366 | 0.179 (-51.08%) | 1.110 | 0.261 (-76.47%) | 6.5 | 6 |
| ORB-SLAM2 | 0.755 | 0.550 (-27.12%) | 0.032 | 0.013 (-58.39%) | 1.411 | 1.344 (-4.76%) | 0.646 | 0.353 (-45.31%) | 0.699 | 0.137 (-80.45%) | 0.391 | 0.422 (7.93%) | 2.6 | 1 |
| RGB-DSLAMv2 | 1.370 | 0.203 (-85.22%) | 0.059 | 0.011 (-81.73%) | 2.307 | 0.484 (-79.00%) | 1.259 | 0.167 (-86.76%) | 1.242 | 0.141 (-88.69%) | 0.540 | 0.115 (-78.73%) | 26.0 | 5 |

When comparing the results obtained with the VO and V-SLAM algorithms, we found that both image-based, dense approaches, i.e. RTABMap and RGB-DOdometry, yielded similar results. They were outperformed by DVO, the third algorithm of the image-based, dense category. A possible reason for this outcome is that the trajectories were well captured by the underlying motion and sensor model of the DVO algorithm. The V-SLAM algorithms were superior with respect to the translational error. This is expected as those algorithms limit drift by matching an incoming image not only to the most recent one but to keyframes or even a model. In fact, ORB-SLAM2 delivered the best results (45% better than the 2nd best). RGB-DSLAMv2 was also ahead of the VO algorithms with the exception of CCNY. Note, however, that CCNY is a hybrid approach making use of frame-to-model matching to keep drift under control. Surprisingly, DVOSLAM performed worse than its VO counterpart although the same parameters were set. Since DVOSLAM not only tries to optimize the intensity error but also the depth error, unreliable depth estimates may have caused the inferior performance. This hypothesis is confirmed by the fact that, as shown in Fig. 3a, the initial direction of both methods was the same. However, the path length predicted by DVOSLAM differed strongly from the real length. Note that DVO did not have this problem, again confirming that an insufficiently accurate depth estimate caused the inferior performance of the DVOSLAM method. Unreliable depth estimates could also be the reason why ICPOdometry, another depth-based matching method, fared rather poorly. Interestingly,

Fovis managed to outperform two V-SLAM techniques and most of the VO algorithms, although it does not build a map of the environment. It relies, however, on a frame-to-keyframe matching technique. This worked well for the trajectories traversed in the first dataset, where the same features were visible for extended time periods. In such a case, the V-SLAM algorithms performed only marginally better. In the second dataset (Tab II), a significantly higher translation error for all algorithms was observed. There are several reasons for this. First, the total trajectory length was about three times as much as in the first dataset (31.2 m vs. 12.8 m) leading to a stronger drift. There was also a long path parallel to an almost featureless flat, white wall spanning more than 50% of the total length (see Fig. 3c). As in the first dataset, the same V-SLAM approach, ORB-SLAM2, performed best. However, unlike in the first dataset, no places were revisited. This means that none of the V-SLAM algorithms could perform loop closure. As a consequence, the performance differences between the V-SLAM and VO algorithms were much smaller. As expected, in a featureless environment the dense methods, RTABMap, DVO, and RGBD-Odometry performed better than feature-based techniques. The inferior performance of Fovis clearly demonstrated this. Remarkably, the CCNY hybrid approach was clearly outperformed by RGB-DICPOdometry. This is because RGB-DICPOdometry uses all 3D points of an incom-ing frame to perform (dense) matching to other frames. Such an approach can cope well in an environment characterized by little texture. This does, however, not apply to CCNY as a feature-based approach. The jumps in the track of RGB-DSLAMv2 are related to the featureless environment, which caused wrong results for the comparison with previous frames. These jumps must be suppressed by filtering if the data is to be used for robot control. Among the VO methods, DVO yielded slightly better results than RGB-DOdometry when looking at the absolute translation error. When judged by the relative translation error, then DVO was clearly better. RGB-DICPOdometry, a hybrid approach, performed best among all VO methods, and it was the second best overall. The second performance criterion was the ability to perform in real-time. This was measured using the frame-process-rate as introduced in Section IV-B. Several algorithms were found to have a lower processing rate making them less suitable for real-time applications. Faster algorithms are, however, available as demonstrated by RGB-DSLAMv2 with 26 fps and CCNY with 21 fps. One reason for the somewhat slow processing speed of the other algorithms was the matrix size of the input streams. The image data had a resolution 3.5 times higher than what most of the algorithm were originally developed for (640x480). However, the higher image resolution led to better results by enabling more precise keypoint detection and better resolved depth values. The performance of RGB-DSLAMv2 is especially interesting because even with its underlying V-SLAM approach it can perform in real-time making it very suitable for practical robotic applications.

## V. Discussion and Conclusions

In this paper, the most common RGB-D based position estimation algorithms were classified first. Second, their performance was experimentally evaluated with respect to localization accuracy and ability to work in real-time using two indoor navigation scenarios. As expected, V-SLAM algorithms yielded more accurate results compared to VO algorithms under normal operating conditions. However, the present paper also shows that there can be situations in which appropriately designed VO algorithms can outperform V-SLAM techniques. A very strong drift in the predicted absolute position data in both data sets was observed when using visual odometry methods. This suggests that VO techniques may not yet be sufficiently reliable for position estimation over longer distances. Even for a rather short trajectory (12m), the position deviated by over 1 meter by the end of the run in one case. There are several ways to deal with this issue. First, one can use a V-SLAM algorithm and try to perform as many loop closures as possible to reduce drift. Second, when a VO approach is the method of choice, then another source for reliable relocalization is essential. This cannot be the wheel odometry as it is also subject to drift. Third, better vision sensors could be employed such as the new Intel RealSense LiDAR Camera L515, which is planned to become available mid-2020. Fourth, data fusion may be applied. The experiments revealed that by fusing vision-based position predictions with wheel odometry and IMU estimates, the translation error could be halved. More precisely, the average performance gain was 67.45%. The results also show that there is probably no single algorithm that performs best for every dataset. However, ORB-SLAM2 can be considered as a stand-alone solution that performs well in environment with high texture. It can even deal with low texture without losing too much accuracy. When it comes to real-time constraints, RGB-DSLAMv2 seems to be a good choice. Based on the findings in this work, a technique for switching between vision algorithms at run time, adapting to the situation at hand, could significantly boost the performance. Implementing such a technique is planned as part of our future work. Furthermore, a study of more sophisticated data fusion strategies could be considered. To this end, sensor modalities other than vision such as radar or ultrasound could be integrated. In addition, it is also conceivable to fuse outputs of sufficiently different and complementary VO or V-SLAM algorithms.

## Conflict of Interest

The authors declare no conflict of interest

## Author Contributions

1st and 2nd author conducted the research, analyzed the data, and wrote the paper. Authors 3rd, 4th,

5th wrote the paper. All authors had approved the final version.

## ACKNOWLEDGMENT

This work was supported by the Hans-Wilhelm Renkhoff Stiftung [33]

## REFERENCES


[1] D. Nister, O. Naroditsky, and J. Bergen, "Visual odometry," in *Proc. the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recogni- tion, 2004. CVPR 2004.*, vol. 1, June 2004, pp. I–I.

[2] Z. Fang and Y. Zhang, "Experimental evaluation of rgb- d visual odometry methods," *International Journal of Advanced Robotic Systems*, vol. 12, no. 3, p. 26, 2015.

[3] D. Scaramuzza and F. Fraundorfer, "Visual odometry [tutorial]," *IEEE Robotics Automation Magazine*, vol. 18, no. 4, pp. 80–92, Dec. 2011.

[4] B. Khaleghi, A. Khamis, F. O. Karray, and S. N. Razavi, "Multisensor data fusion: A review of the state-of-the-art," *Information Fusion*, vol. 14, no. 1, pp. 28–44, 2013.

[5] Intel, [Online]. Available: *Https://store.intelrealsense.com/buy-intel-realsense-depth-camera-d435.html*, Aug. 6, 2019

[6] F. Steinbrücker, J. Sturm, and D. Cremers. "Real-time visual odometry from dense RGB-D images," in *Proc. 2011 IEEE International Conference on Computer Vision Workshops (ICCV Workshops)*. Nov. 2011, pp. 719–722.

[7] C. Kerl, J. Sturm, and D. Cremers. "Robust odometry estimation for RGB-D cameras," in *Proc. 2013 IEEE International Conference on Robotics and Automation*. May 2013, pp. 3748–3754.

[8] A. S. Huang, A. Bachrach, P. Henry, M. Krainin, D. Maturana, D. Fox, and N. Roy, *Visual Odometry and Mapping for Autonomous Flight Using an RGB-D Camera*. Springer, Robotics Research, 2017, pp. 235–252.

[9] G. Bradski, *Icpodometry*. [Online]. Available: https://docs. opencv.org/4 2.0/d7/d83/classcv_1_ 1rgbd_1_ 1ICPOdometry. html, Jan. 15, 2020.

[10] *Rgbdicpodometry*. [Online]. Available: https://docs.opencv.org/4.2.0/d2/d0f/classcv_ 1_ 1rgbd_ 1_ 1RgbdICPOdometry.html, Jan. 15, 2020.

[11] I. Dryanovski, R. G. Valenti, and J. Xiao, "Fast visual odometry and mapping from rgb-d data," in *Proc. 2013 IEEE International Conference on Robotics and Automation*, 2013, pp. 2305–2310.

[12] C. Kerl, J. Sturm, and D. Cremers, "Dense visual slam for rgb-d cameras," in *Proc. 2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Nov. 2013, pp. 2100–2106.

[13] M. Labbé and F. Michaud, "Appearance-based loop closure detection for online large-scale and long-term operation," *IEEE Transactions on Robotics*, vol. 29, no. 3, pp. 734–745, June 2013.

[14] R. Mur-Artal and J. D. Tardós, "Orb-slam2: An open-source slam system for monocular, stereo, and rgb- d cameras," *IEEE Transactions on Robotics*, vol. 33, no. 5, pp. 1255–1262, Oct. 2017.

[15] F. Endres, J. Hess, J. Sturm, D. Cremers, and W. Burgard, "3-d mapping with an rgb-d camera," *IEEE Transactions on Robotics*, vol. 30, no. 1, pp. 177–187, Feb. 2014.

[16] J. Stückler and S. Behnke, "Multi-resolution surfel maps for efficient dense 3d modeling and tracking," *J. Vis. Comun. Image Represent.*, vol. 25, no. 1, pp. 137–147, Jan. 2014.

[17] O. Inc., *The Structure Sensor*. [Online]. Available: http://structure.io, Jan. 15, 2020.

[18] J. Sturm, N. Engelhard, F. Endres, W. Burgard, and D. Cremers, "A benchmark for the evaluation of rgb-d slam systems," in *Proc. of the International Conference on Intelligent Robot Systems (IROS)*, Oct. 2012.

[19] V. Angladon, S. Gasparini, V. Charvillat, T. Pribanić, T. Petković, M. Đonlić, B. Ahsan, and F. Bruel, "An evaluation of real-time rgb-d visual odometry algo- rithms on mobile devices," *Journal of Real-Time Image Processing*, vol. 16, no. 5, pp. 1643–1660, Oct. 2019.

[20] A. Handa, T. Whelan, J. McDonald, and A. J. Davison, "A benchmark for rgb-d visual odometry, 3d recon- struction and slam," in *Proc. 2014 IEEE International Conference on Robotics and Automation (ICRA)*, May 2014, pp. 1524–1531.

[21] R. A. Newcombe, S. Izadi, O. Hilliges, D. Molyneaux, D. Kim, A. J. Davison, P. Kohi, J. Shotton, S. Hodges, and A. Fitzgibbon, "Kinectfusion: Real-time dense sur- face mapping and tracking," in *Proc. 2011 10th IEEE International Symposium on Mixed and Augmented Reality*, Oct. 2011, pp. 127–136.

[22] T. Whelan, H. Johannsson, M. Kaess, J. J. Leonard, and J. McDonald, "Robust real-time visual odometry for dense rgb-d mapping," in *Proc. 2013 IEEE International Conference on Robotics and Automation*, May 2013, pp. 5724–5731.

[23] Z. Fang and S. Scherer, "Experimental study of odom- etry estimation methods using rgb-d cameras," in *Proc. 2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Sep. 2014, pp. 680–687.

[24] A. Quattrini Li, A. Coskun, S. M. Doherty, S. Ghasem- lou, A. S. Jagtap, M. Modasshir, S. Rahman, A. Singh, M. Xanthidis, J. M. O'Kane, and I. Rekleitis, "Exper- imental comparison of open source vision-based state estimation algorithms," in *2016 International Symposium on Experimental Robotics*, D. Kulić, Y. Nakamura, O. Khatib, and G. Venture, Eds., Cham: Springer Inter- national Publishing, 2017, pp. 775–786.

[25] Evocortex. [Online]. Available: *Evorobot*, https://evocortex.org/products/ evorobot/, Jan. 15, 2020.

[26] M. Burri, J. Nikolic, P. Gohl, T. Schneider, J. Re- hder, S. Omari, M. Achtelik, and R. Siegwart, "The euroc micro aerial vehicle datasets," *The International Journal of Robotics Research*, 2016.

[27] M. Labbé and F. Michaud, "Rtab-map as an open-source lidar and visual simultaneous localization and mapping library for large-scale and long-term online operation," *Journal of Field Robotics*, vol. 36, no. 2, pp. 416–446, 2019.

[28] D. Galvez-López and J. D. Tardos, "Bags of bi-nary words for fast place recognition in image sequences," *IEEE Transactions on Robotics*, vol. 28, no. 5, pp. 1188–1197, 2012.

[29] M. Fischler and R. Bolles, "Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography," *Commun. ACM*, vol. 24, no. 6, pp. 381–395, June 1981.

[30] Intel, *Intelrealsenseroswrapper*, version 2.2.12, Jan. 15, 2020. [Online]. Available: https://github.com/ IntelRealSense/realsense-ros.

[31] *Intel Realsense SDK*, version 2.0, Jan. 15, 2020. [Online]. Available: https://github.com/IntelRealSense/ librealsense.

[32] evocortex, *Localizersdk*, https://evocortex.org/products/ localizer-sdk/, Jan. 15, 2020.

[33] *Warema Renkhoff SE*, Jan. 26, 2020. [Online]. Available: https://www.warema-group.com/en/WAREMA group/ WAREMA Renkhoff SE.php.




**Florian Spieß** is a Ph.D. student at the chair of software engineering at the University of Würzburg. His research topics include people detection and localization. He researches in the field of mobile roboitcs and Industry 4.0 as well as Platooning and Intelligent Transportation Systems.