Autonomous flying with a commercially available micro-air-vehicle (MAV) below 100 grams

Internship report

Jordy van Appeven, i6147983

Abstract

Micro-air-vehicles (MAVs) are flying vehicles that are restricted in size and weight. The subset of quad-copters is a quickly growing and popular area of research. The goal of this study is to reach autonomous flying using an off-the-shelf, commercially available MAV. Requirements of the MAV platform are: it should weigh less than 100 grams, financial costs of around €150 and most importantly open source access through a software development kit(SDK). Evaluating the autonomous flying of the platform is done by testing its simultaneous localization and mapping (SLAM) performance. SLAM is the process of mapping an unknown environment while keeping track of its own position with respect to this map. If a robot is aware of its surroundings it can start interacting, for example when moving without causing any collisions. The SLAM performance is quantified by the accuracy between an estimated trajectory compared to the ground-truth. Because of the small size and low weight restrictions, on-board processing is limited to low-level control for stabilization and hovering. High-level control is done on a ground-station, receiving sensor data to estimate a current position and transmitting control commands to fly the MAV. Retrieving a ground-truth trajectory is done using a stereo-camera instead of an external tracking system seen in related work. From a number of potential MAV platforms the Parrot Mambo is eventually tested, which is equipped with on-board stabilization. One of the risks of this platform is receiving inaccurate- or an insufficient amount of sensor data. This is solved by developing an external sensor module consisting of a Raspberry Pi, camera and internal-measurement-unit (IMU) attached onto the MAV. In the experiments two localization paradigms are tested, based on an extended Kalman filter and a smoothing approach using non-linear optimization.

1 Introduction

Unmanned-air-vehicles (UAVs), specifically quadcopters, are a quickly growing and popular research area. Their extra dimension of movement and constant in-air hovering result in complex flight dynamics and -kinematics, while having a limited number of mechanical components involved [8]. This makes UAVs a suitable hardware platform for research and education in varying topics such as model identification and control theory. But also at a higher-level in robotics UAVs are attractive ; e.g. in the area of autonomous robotic systems, towards multi-agent swarms and for interaction with it's surroundings, for instance during search and rescue missions or at construction sites. With a weight below 100 grams a much smaller and cheaper subset of UAVs are the micro-air-vehicles (MAVs). The use of these flying robots is more effective than general UAVs in confined area's, while at the same time offering a much safer environment in case of collisions. The flying robots are also accessible in research and education as they cost less than €150. Additionally recent legal developments might cause an even further shift towards the use of MAV's. By proposing a requirement to register UAVs above 250 grams and a pilot license for UAVs above 900 grams [2], when flying outside. To further test the suitability of these small MAV's within research and education their ability towards autonomous flying is tested in this work. This evaluation is demonstrated according to their performance of simultaneous localization and mapping (SLAM). Since SLAM is a key-aspect for autonomous robots. Since a robot is required to be aware of it's surroundings before it can start interacting, for instance when moving around without causing any collisions.

1.1 Problem statement

Before getting to the problem definition, a brief overview is given of the previous work towards autonomous UAVs using different implementations of SLAM. Starting by the application of time-of-flight(e.g. LiDAR) or depth cameras (e.g. stereo-cameras, Microsoft Kinect) which, combined with an IMU, achieving accurate on-board localization and mapping in an unknown environment [11][15]. However these systems are heavy platforms with a weight around 3 kg and high financial costs. Other visual-inertial-odometry (VIO) methods decrease the weight and costs by using monocular-camera systems [7][26][24][20][19][27], involving just a single camera and an IMU. It has been demonstrated that this approach works on commercially available UAVs due to the use of an open-source software-developer-kit (SDK), without the need for any hardware modifications. [9][31] The success of these existing solutions towards autonomous UAVs combined with the development of more accessible MAV platforms bring up the following research question: ”Can we apply existing methods of autonomous flying to commercially available MAV platforms with a weight below 100 grams?”. Related to this question and within the scope of this work the following sub-questions will be answered:

- What commercial MAV platforms are available shipped with open-source SDK, a weight below 100 grams and a price under €150?
- Which existing SLAM solutions can be implemented on these platforms?
- How can the localization accuracy on these platforms be quantified?

1.2 State-of-the-art

A clear picture of previous and related to autonomous UAVs helps in organizing the approach towards the goal of this work. This is visualized in figure 1, plotting specific approaches towards autonomous UAVs according to their weight against the complexity and financial costs of the approach. Furthermore the size of each marker defines their success in autonomous flying. The first approach that is widely used, even across commercial UAV platforms, is the use of GPS in order to achieve localization accurate to a few centimeters, through an absolute position estimate. Successful autonomous flying based on GPS is available on platforms with a weight from 500 grams. A disadvantage of these systems is their useability to areas covered by GPS signal. Other approaches use time-of-flight (ToF) sensors, such as LiDAR [11]. Adding high financial costs above €1000 for a 2D ToF sensor [11]. Also ToF sensors come with a heavy weight. Use cases employing vision based systems recently gained an increase in popularity, not only with stereo-camera's but even systems up to 13 camera's on-board [3].

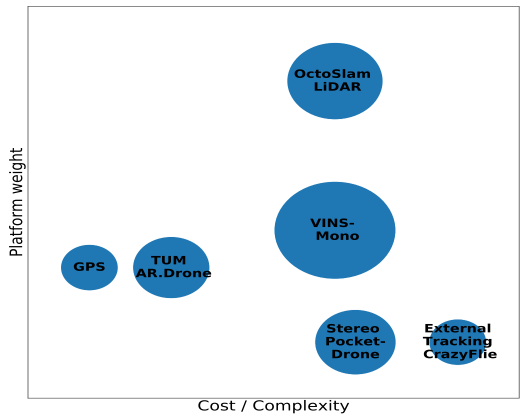


Fig. 1: Overview previous approaches of autonomous UAVs, with the size corresponding to their success in autonomous flying.

Instead of large and heavy UAVs the other end of the spectrum focuses on MAVs below 100 grams. The restriction in payload capacity asks for a completely different approaches towards autonomous flying as demonstrated by several implementations on the CrazyFlie platform [8][1]. These systems reach very precise autonomous flying through the use of external tracking systems (infrared- or ultra-wideband technology[32]). The downsides of these systems are high financial costs, around €50.000, and restrict deployment to predefined environments. Another approach with a different MAV platform illustrates impressive work by implementing a low-weight stereo-camera combined with novel solutions focused on the high computational requirements for the processing of stereo images [22]. Due to their complexity their approach is however outside the scope of this study, which is towards an accessible platform suitable for research and education. More promising approaches to be used with commercially available MAV's consist of a monocular-camera combined with an IMU. As is demonstrated by earlier work through robust localization and accurate figure flying [4]. Implemented on the commercial Parrot AR.Drone with a weight of 420 grams transmitting sensor measurements at 150 Hz to a ground-station. Furthermore images are processed by the camera tracking system PTAM (parallel tracking and mapping) [17]. Fusing of the sensor measurements and tracked images is done through an extended Kalman filter. An implementation of this work is available through the 'TUM AR.Drone' library. Successive work extends on this towards autonomous exploration [31] of an unknown environment. Implemented on the newer Parrot Bebop platform with a higher weight of 500 grams and longer flight-time of 25 minutes. This extension requires modifications in the original 'TUM AR.Drone' library, which hasn't been released as open-source. The biggest change lies in the camera tracking system PTAM replaced by LSD-SLAM (Large-scale direct monocular SLAM). Which instead of a sparse map with a low number of key-points returns a semi-dense map representation. More suitable autonomous navigation and object avoidance. This is due to the direct processing of pixel intensities instead of tracking key-points between consecutive images. More recent work replaces the traditional EKF by Bayesian smoothing approaches [29], implemented through a non-linear optimization framework [5][18][16]. According to a benchmark comparison [6] 'VINS-Mono' [24] is a very competitive approach based on Bayesian smoothing. Offering a complete system for robust and accurate localization, while having a slight increase in computational costs compared to other implementations. The original work is demonstrated by fusing data between IMU and monocular-camera (mvBlueFOX-MLC200w) on-board of a self developed UAV platform with DJI A3 flight controller. Even during aggressive flight maneuvers the results are successful, flying an eight-figure at velocities up to 2.8 m/s. Similar work, based on the same non-linear optimization, extends the autonomous flight maneuvers to the mapping of free- and occupied space [12][33] and eventually with autonomous navigation in cluttered environments [13][14]. The VINS-mono seems very promising to implement on a MAV, especially with the minimal sensor setup consisting of an IMU and monocular-camera.

1.3 Approach

After extensive research a final choice led to the Parrot Mambo as commercial MAV platform for autonomous flying, shipped with an open-source SDK for €150 and a weight of 72 grams. Additionally the platform is equipped with on-board stabilization software. An alternative to the platform is the DJI Tello, however this wasn't released at the time. A severe risk of the Parrot Mambo platform are noisy and low-frequency sensor measurements transmitted to the ground-station. Therefore an additional sensor module is developed and attached to the MAV, consisting of a Raspberry Pi Zero, IMU and wide-angle camera. Detailed specifications for both setups are given in 2.1 and 2.2. Furthermore two localization methods are tested on the MAV platform, TUM AR.Drone and VINS-Mono. Quantifying the localization accuracy is done by comparing an estimated trajectory with the ground-truth. An external stereo-camera is used to retrieve this ground-truth. This is different from other methods that use an expensive tracking system. Evaluation of the accuracy is based on two performance metrics, an absolute-pose-error and relative-pose-error both returning statistical analysis. Implementation of all systems in this work is done using the Robot-Operating-System (ROS) [25]. It's purpose is to communicate between all processes running on the ground-station, Raspberry Pi and a server, used for external-tracking.

1.4 Outline

The outline through this study is as follow. Starting with a description of all the hardware components in sections 2.1 and 2.2. Followed by an explanation of the SLAM packages in section 2.3 and 2.4. In subsection 2.5 the process to obtain the ground-truth is illustrated. And subsection 2.6 describes corresponding evaluation metrics between an estimated trajectory and the ground-truth. Then section 3 contains the experiment related to estimate the precision of the ground-truth setup. While 3 evaluates the localization accuracy for each sensor setup. Finally this study is concluded in chapter 5.

2 Methods

First the sensor setups used in this study are specified. Consisting of the Parrot Mambo's on-board sensors and a Raspberry Pi sensor module. Then two SLAM implementations are explained. First the EKF and then the Bayesian smoothing using non-linear optimization. Followed by an illustration of the ground-truth setup and finally the definition of error metrics.

2.1 Parrot Mambo specifications

The on-board flight controller of the Parrot Mambo achieves a stable hovering state by using it's downward facing ultrasonic sensor and optical flow camera. The fusing of all sensor measurements results in an estimated orientation, altitude, and velocity in x, y, z direction corresponding to the local-coordinate frame of the MAV. These estimates are received at a ground-station with a frequency of 2 Hz, either by Bluetooth or WiFi connection. Since the flight controller of the Parrot Mambo is closed-source it is not clear how the fusing of sensor measurements is done, however it could be similar to the implementation on an older Parrot AR.Drone platform [4].

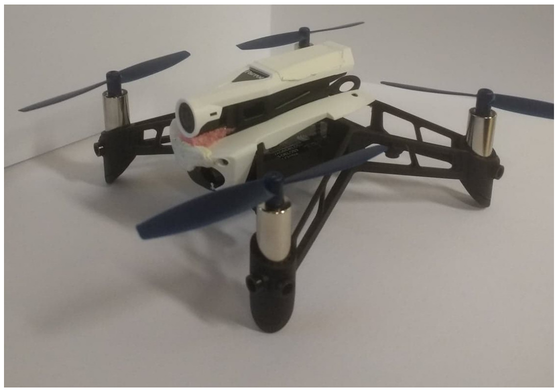


Fig. 2: Parrot Mambo drone with FPV camera.

Next to these default sensors a first-persion-view (FPV) camera can be attached (figure 2), sending images at maximum 20 FPS in either 720p or 1080p resolution. This Parrot Mambo FPV edition, including the extra camera, is available for €150 and has a total weight of 72 grams. Communication between a ground-station and the Parrot Mambo is possible by the official SDK released by Parrot. The SDK is available for the progamming languages objective-C, Java and C. In this work however the unofficial 'PyParrot' [21] interface for python will be used. Adding additional functionality compared to the official SDK. Besides just receiving data, the ground-station can transmit control commands to fly the MAV. These control commands consists of 5 variables, 4 directional values between -1.0 and 1.0 and a time duration to execute the command. The directional values correspond to a fraction of the maximum pitch, roll, yaw and thrust velocities in either positive or negative direction. Aligned with right-handed Euler angles (3).

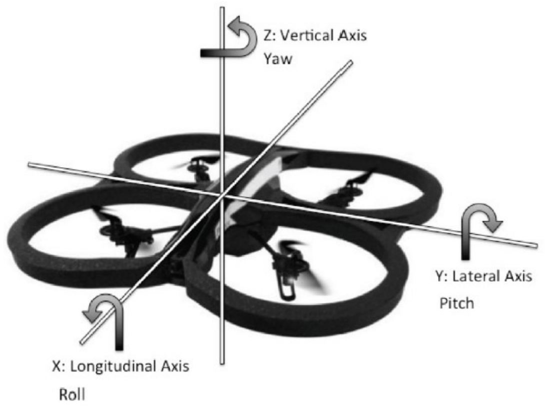


Fig. 3: Coordinate-frame of Parrot Mambo drone, according to right-handed Euler angles.

2.2 Raspberry Pi sensor module

To overcome the risk of insufficient or inaccurate sensor data transmitted from the MAV to the ground-station, a separate sensor module is developed shown in figure 4.

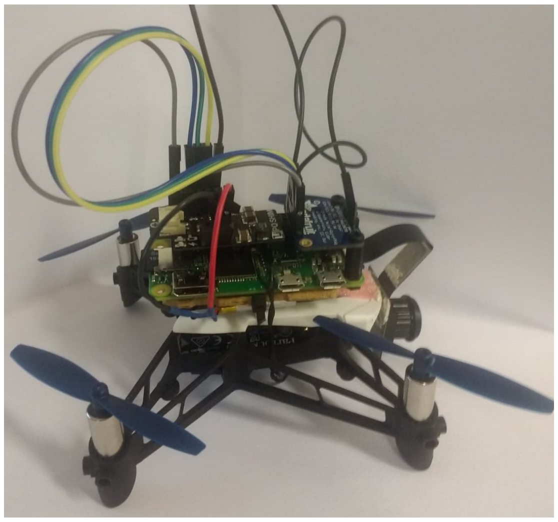


Fig. 4: Parrot Mambo with Raspberry Pi sensor module.

For this purpose the Raspberry Pi Zero can be used as an embedded computing board. Adding a battery, power supply, camera and IMU makes it possible to collect raw sensor measurements at high frequencies. These raw measurements are necessary by more recent visual-inertial (VI) SLAM packages. Since the processing power of the Raspberry Pi Zero isn't sufficient to run these packages on-board, all data is transmitted to a ground-station using a WiFi connection. A complete overview with all components in the sensor module is shown in table 1, together with their specifications 1.

Component	Description	Price	Weight
Raspberry Pi zero W		€10.71	11.5 grams
ZeroCam FishEye	Camera for Raspberry Pi Zero, wide-angle 100-120 degree field of view	€17.45	3 grams
Adafruit IMU BNO055	9-axis Inertial measurement unit, accelerometer, gyroscope, geomagnetic sensor	€30.52	3 grams
LiPo battery 190 mAh	External battery pack, 24.5 mm x 23.5 mm x 4.2 mm	€6.11	4.5 grams
Pimoroni LiPo SHIM	Power supply SHIM (shove hardware in the middle) to power Raspberry Pi with LiPo battery	€12.25	2.6 grams
Jumper cables	Male/Female wires 6 x 150mm	€0.70	4 grams
	Total	€77.74	28.6 grams

Tab. 1: Specifications for all components of developed sensor module

2.3 Extended Kalman filter approach

One approach for localization of the MAV is by using an EKF. The EKF is an extension on the Kalman filter so that it can be applied to a system with non-linear dynamics. Not only MAVs' but most dynamical systems are non-linear. The extension consists of a linear approximation of the non-linear system; e.g. using a first-order Taylor expansion (formula 1). This requires computing the Jacobian of the non-linear dynamics. This approximation is valid only locally close to the linearized point. The EKF used in this work is implemented in the TUM AR.Drone package. This packages estimates the state variables according to formula 2.

$$f(x) = f(x_0) + f'(x_0)(x - x_0) \tag{1}$$

$$\boldsymbol{X}_t = \left[x_t, y_t, z_t, \dot{x}_t, \dot{y}_t, \dot{z}_t, \Phi_t, \Theta_t, \Psi_t, \dot{\Psi}_t\right]^T \tag{2}$$

the (x_t, y_t, z_t, Θ_t, Φ_t and Ψ_t) correspond to a position and orientation at time t. Additionally the x_t, y_t, z_t and yaw (Ψ_t) velocity are estimated all in world-coordinates. Furthermore the behavior of the MAV system is explained. Consisting of a linear prediction- and a non-linear observation model.

2.3.1 Prediction model

The prediction model is used to estimate the MAVs' state when no sensor measurements or camera images are available. This is valuable for two reasons. First of all there's a time delay between the the moment measurements take place and theyre processed into a state update. Secondly it is used to estimate the effect of a certain control command. A time delay takes place during the processing and transmion of data between the MAV platform and a ground-station. Besides predicting the effect of control commands to reach a desired behavior, there's also a delay between transmitting controls and execution by the MAV. The prediction model in the TUM AR.Drone package is based on the linear function in formula 3. This linear function is sufficiently accurate because of the low velocities that are involved.

$$\begin{aligned} \ddot{x}(\boldsymbol{x}_t) &= c_1 R(\Phi_t, \Theta_t, \Psi_t)_{1,3} - c_2 \dot{x}_t \\ \ddot{y}(\boldsymbol{x}_t) &= c_1 R(\Phi_t, \Theta_t, \Psi_t)_{2,3} - c_2 \dot{y}_t \\ \dot{\Phi}(\boldsymbol{x}_t, \boldsymbol{u}_t) &= c_3 \bar{\Phi}_t - c_4 \Phi_t \\ \dot{\Theta}(\boldsymbol{x}_t, \boldsymbol{u}_t) &= c_3 \bar{\Theta}_t - c_4 \Theta_t \\ \ddot{\Psi}(\boldsymbol{x}_t, \boldsymbol{u}_t) &= c_5 \bar{\dot{\Psi}}_t - c_6 \dot{\Psi}_t \\ \ddot{z}(\boldsymbol{x}_t, \boldsymbol{u}_t) &= c_7 \bar{\dot{z}}_t - c_8 \dot{z}_t \end{aligned} \tag{3}$$

The horizontal accelerations $(\ddot{x}, \ddot{y})$ are estimated by projecting the z-axis in the local-coordinate frame to the x-/y plane in global-coordinates. This is calculated using the roll- (Θ_t) and pitch (Φ_t) orientation. Then the current velocities $(\dot{x}_t, \dot{y}_t)$ are subtracted which represent a drag-force. Furthermore the pitch- $(\dot{\Phi})$ and roll velocity $(\dot{\Theta})$ are estimated by a difference between set control commands $(\bar{\Phi}_t, \bar{\Theta}_t)$ and current states (Φ_t, Θ_t). A larger difference is proportional to a higher velocity. Similarly control commands $(\bar{\dot{\Psi}}_t)$ and $(\bar{\dot{z}}_t)$ are used to estimate the yaw- $(\ddot{\Psi})$ and throttle $(\ddot{z})$ acceleration. Parameters c_1 to c_8 are platform dependent and need to be estimated. This is done by recording all sensor measurements, control inputs and corresponding filter states during a test flight. Replaying this data is used to minimize the error between two filters. One filter is based on observed states, while the other filter is based purely on the prediction model. This is done by trial-and-error.

2.3.2 Observation model

The estimated states in the state vector $\boldsymbol{X}_t$ are corrected by using observations from sensor measurements and camera images. For this both observations need to be transformed to matching the variables in the state vector. These transformations are described by an observation model. Starting with the odometry model, based on the sensor measurements in formula 4. Because the observed velocities $(\hat{v}_{x,t}, \hat{v}_{y,t})$ are according a local-coordinate frame of the MAV they are transformed into global velocities $\dot{x}_t$ and $\dot{y}_t$. Next the estimated pitch- (Φ_t) and roll (Θ_t) orientation are directly observed by the sensor measurements $(\boldsymbol{z}_I, t)$.

Hence no transformation is required. Finally the vertical- $(\hat{v}_z, t)$ and yaw $(\hat{\dot{\Psi}}_t)$ velocities are computed by differentiation of consecutive height- $(\hat{h}_t, \hat{h}_{t-1})$ and yaw- $(\hat{\Psi}_t, \hat{\Psi}_{t-1})$ measurements. Also shown in formula 4.

$$\boldsymbol{z}_{I,t} = \left[\hat{v}_{x,t}, \hat{v}_{y,t}, \frac{\hat{h}_t - \hat{h}_{t-1}}{\delta_t - 1}, \hat{\Phi}_t, \hat{\Theta}_t, \frac{\hat{\Psi}_t - \hat{\Psi}_{t-1}}{\delta t - 1}\right]^T \tag{4}$$

The visual observation model is based on feature tracking by PTAM, which computes a relative state observation in the camera-coordinate frame. Computing this relative state measurement is explained in the original work: ”Parallel Tracking and Mapping for Small AR Workspaces” [17]. This observation is relative since a scale is unobservable from camera images. Hence the relative observation is scaled by a factor λ, estimated from odometry observations. Since this scaled state observation is still in the camera-coordinate frame $(\boldsymbol{E}_{C,t})$ it is transformed to the local-coordinate frame $(\boldsymbol{E}_{DC})$. Followed with a conversion in notation from the rotation matrix (SE(3)) to Euler angles $(\mathbb{R}^6)$ through $f(.)$. The estimated states in formula 5 can now be corrected by this visual observation in formula 6.

$$h_p(\boldsymbol{x}_t) = \left[x_t, y_t, z_t, \Phi_t, \Theta_t, \Psi_t\right]^T \tag{5}$$

$$\boldsymbol{z}_{P,t} = f(\boldsymbol{E}_{DC}, \boldsymbol{E}_{C,t}) \tag{6}$$

2.4 Visual-inertial Bayesian smoothing

One of the assumptions in Kalman filtering for predicting a state $\boldsymbol{x}_k$ at time k is an independence from all previous states and measurements except for the most recent ones (formula 7). Each state $\boldsymbol{x}_k$ is fixed after marginalization of $\boldsymbol{x}_{k-1}$ and $\boldsymbol{y}_k$. An alternative to this filtering approach is Bayesian smoothing [29]. In Bayesian smoothing the current state $\boldsymbol{x}_k$ is dependent on all previous states $\boldsymbol{x}_{0:k-1}$, but also on future states $\boldsymbol{x}_{k+1:T}$ (formula 8), hence the state $\boldsymbol{x}_k$ is not fixed. Instead initial state estimates are repeatedly updated when new states and -measurements become available, by formulating a non-linear optimization model. This method is more accurate, especially when working with a highly non-linear system. Since fixed EKF estimations are only linearized once, which might be very inaccurate. The repeadetly updating of initial state estimates is also effective for estimating additional variables. For example gyroscope- and accelerometer biases $(\boldsymbol{b}_\omega, \boldsymbol{b}_a)$, since they dependend on state estimates which are more accurate [24]. One of the limitations with this approach is computations become infeasible by continuously adding new states, appending more dependencies to include. This is solved in a sliding window formulation, further explained in 2.4.2.

$$p(\boldsymbol{x}_k | \boldsymbol{y}_k, \boldsymbol{x}_{k-1}) \tag{7}$$

$$p(\boldsymbol{x}_k | \boldsymbol{y}_{1:T}, \boldsymbol{x}_{0:T}), \text{ for } k = 1, ..., T \tag{8}$$

2.4.1 Preprocessing IMU- and feature observations

Before adding observations to the optimization model the corresponding sensor measurements are preprocessed. Observations either originate from raw IMU measurements, with accelerations $\hat{\boldsymbol{a}}_t$ in x, y, z directions and angular velocities $\hat{\boldsymbol{\omega}}_t$ in quaternion notation, or camera images (..). The preprocessing takes place between two image frames. For camera images this consists of the detection- and tracking of features $(\lambda_0, .., \lambda_m)$ corresponding with the current key-frame $\boldsymbol{x}_k$. Since IMU measurements come at a much higher frequency (200 Hz) than camera images (30 fps) they are pre-integrated between 2 image frames according to formula 9.

$$
\begin{aligned}
\boldsymbol{p}^w_{b_{k+1}} &= \boldsymbol{p}^w_{b_k} + \boldsymbol{v}^w_{b_k}\delta t_k + \iint_{t\in[t_k,t_{t+1}]} (\boldsymbol{R}^w_t(\hat{\boldsymbol{a}}_t - \boldsymbol{b}_{a_t} - \boldsymbol{n}_a) - \boldsymbol{g}^w)dt^2 \\
\boldsymbol{v}^w_{b_{k+1}} &= \boldsymbol{v}^w_{b_k} + \int_{t\in[t_k,t_kk+1]} (\boldsymbol{R}^w_t(\hat{\boldsymbol{a}}_t - \boldsymbol{b}_{a_t} - \boldsymbol{n}_a) - \boldsymbol{g}^w)dt \\
\boldsymbol{q}^w_{bk+1} &= \boldsymbol{q}^w_{b_k} \otimes \int_{t\in[t_k,t_{k+1}]} \frac{1}{2}\Omega(\hat{\boldsymbol{w}}_t - \boldsymbol{b}_{\omega_t} - \boldsymbol{n}_w)\boldsymbol{q}^{b_k}_t dt, \\
\Omega(\boldsymbol{\omega}) &= \begin{bmatrix} -\lfloor\omega\rfloor_\times & \omega \\ -\omega^T & 0 \end{bmatrix}, \lfloor\omega\rfloor_\times = \begin{bmatrix} 0 & -\omega_z & \omega_y \\ \omega_z & 0 & -\omega_x \\ -\omega_y & \omega_x & 0 \end{bmatrix}
\end{aligned}
\tag{9}
$$

Here $\boldsymbol{p}^w_{b_k}, \boldsymbol{v}^w_{b_k}$ and $\boldsymbol{q}^w_{b_k}$ are respectively the position, velocity and orientation at $t = b_k$ in the world-coordinate frame. With pre-integration of IMU measurements between two image frames b_k and b_{k+1}, in other words integration over the interval $[t_k, t_{k+1}]$. Within the integral, $\boldsymbol{R}^w_t$ rotates raw IMU measurements from the body frame to the world frame, before subtracting a gravity acceleration $\boldsymbol{g}^w$. Furthermore the additive noise $\boldsymbol{n}_a$ and $\boldsymbol{n}_w$) is assumed to be Gaussian, while biases $\boldsymbol{b}_{a_t}$ and $\boldsymbol{b}_{w_t}$) are modeled as a random walk with the derivative set as a Gaussian distribution, for respectively accelerometer and gyroscope measurements. Integration of the measurements can be done using any numerical method; e.g. Euler, mid-point or Runge Kuta 4 estimating the mean for each of the variables. Covariances can be computed recursively by a first-order approximation using the Jacobian. This is shown in the original paper [24], together with an approach to prevent frequent re-integration. Re-integratoin is an issue, since everytime biases $(\boldsymbol{b}_{a_t}, \boldsymbol{b}_{\omega_t})$ or rotation $\boldsymbol{R}^w_t$ is updated, re-integration takes place. A final observation model as returned by pre-integration of IMU measurements is shown in formula 10.

$$
\begin{bmatrix} \hat{\boldsymbol{\alpha}}^{b_k}_{b_{k+1}} \\ \hat{\boldsymbol{\beta}}^{b_k}_{b_{k+1}} \\ \hat{\gamma}^{b_k}_{b_{k+1}} \\ \boldsymbol{0} \\ \boldsymbol{0} \end{bmatrix} = \begin{bmatrix} \boldsymbol{R}^{b_k}_{b_{k+1}}(\boldsymbol{p}^w_{b_{k+1}} - \boldsymbol{p}^w_{b_k} + \frac{1}{2}\boldsymbol{g}^w\delta t_k^2 - \boldsymbol{v}^w_{b_k}\delta t_k) \\ \boldsymbol{R}^{b_k}_w(\boldsymbol{v}^w_{b_{k+1}} + \boldsymbol{g}^w\delta t_k - \boldsymbol{v}^w_{b_k}) \\ \boldsymbol{q}^{w^{-1}}_{b_k} \otimes \boldsymbol{q}^w_{b_{k+1}} \\ \boldsymbol{b}_{a_{b_{k+1}}} - \boldsymbol{b}_{a_{b_k}} \\ \boldsymbol{b}_{w_{b_{k+1}}} - \boldsymbol{b}_{w_{b_k}} \end{bmatrix} \tag{10}
$$

Every time a new key frame b_{k+1} is accepted pre-integrated mean values for position ($\hat{\boldsymbol{\alpha}}^{b_k}_{b_{k+1}}$), velocity ($\hat{\boldsymbol{\beta}}^{b_k}_{b_{k+1}}$) and orientation ($\hat{\boldsymbol{\gamma}}^{b_k}_{b_{k+1}}$), with corresponding and covariances, are included in the optimization model. Accepting new key frames takes place if the number of tracked features between current frame and key-frame isn't enough or when the average parallax of tracked features is large enough. Both are specified by a certain threshold. Features are detected cornerpoints by the approach of Shi and Tomasi [28] with a minimum distance around 30 pixels. Tracking these features is done using the KLT sparse optical flow algorithm [30] for 100 to 300 features. Detected features are undistorted by the internal camera matrix, with calibration parameters according to the pinhole model. Outliers are rejected by the RANSAC algorithm combined with epipolar constraints [10]. At last tracked features are projected to an unit sphere corresponding to the pose representation, this is also further detailed in the original paper [24].

2.4.2 Tightly-coupled optimization model

Pose estimation using Bayesian smoothing is done by optimizing over the full state vector in formula 11. Variables included in this vector are IMU states $\boldsymbol{x}_k$ at time k. Containing position ($\boldsymbol{p}^w_{b_k}$), velocity ($\boldsymbol{v}^w_{b_k}$), orientation($\boldsymbol{q}^w_{b_k}$), and biases ($\boldsymbol{b}_a, \boldsymbol{b}_g$) in world-coordinate frame at time b_k related to one of the n accepted key-frames. Furthermore there are m features being tracked with a value for λ_l representing the inverse-depth of the l^{th} feature at it's first observation. The optimization in VINS-Mono is solved by Ceres Solver [5]. The full state vector is updated by optimization of the non-linear cost-function in formula 12.

$$\begin{aligned} X &= \left[\boldsymbol{x}_0, \boldsymbol{x}_1, ...\boldsymbol{x}_n, \boldsymbol{x}^b_c, \lambda_0, \lambda_1, ...\lambda_m\right] \\ \boldsymbol{x}_k &= \left[\boldsymbol{p}^w_{b_k}, \boldsymbol{v}^w_{b_k}, \boldsymbol{q}^w_{b_k}, \boldsymbol{b}_a, \boldsymbol{b}_g\right], k \in [0, n] \\ \boldsymbol{x}^b_c &= \left[\boldsymbol{p}^b_c, \boldsymbol{q}^b_c\right] \end{aligned} \tag{11}$$

In total n key frames and m features are included. Here λ represents the inverse depth of the l'th feature at it's first observation. The states are updated by the optimization of visual-inertial bundle adjustment, minimizing the following cost function to return a maximum posteriori estimate (MAP):

$$\min_X \left\{ ||\boldsymbol{r}_p - \boldsymbol{H}_p X||^2 + \sum_{k\in B} \left|\left|\boldsymbol{r}_B(\hat{\boldsymbol{z}}^{b_k}_{b_{k+1}}, X)\right|\right|^2_{\boldsymbol{P}^{b_k}_{b_{k+1}}} + \sum_{(l,j)\in C} \rho(\left|\left|\boldsymbol{r}_C(\hat{\boldsymbol{z}}^{C_j}_l, X\right|\right|^2_{\boldsymbol{P}^{c_j}}) \right\},$$

$$\rho(s) = \begin{cases} 1 \text{ , for } s \geq 1 \\ 2\sqrt{s} - 1 \text{ , for } s < 1 \end{cases} \tag{12}$$

This cost-function consists of a sum of 3 parts, a prior (contained in $\boldsymbol{r}_p, \boldsymbol{H}_p$) and a Mahalanoris norm for observation residuals for respectively camera images (r_C) and IMU measurements (r_B). To restrict computational costs a sliding-window is defined defining the full state vector. Consisting of features that are observed at least twice and all pre-integrated IMU measurements. A detailed explanation of both residuals can be found in the original work [24]. The prior contains marginalized IMU observations and image features. When a new key-frame is accepted the current oldest key-frame is removed from the sliding-window. This key-frame is marginalized into the prior, together with corresponding IMU observations. All unaccepted key-frames are directly marginalized. Marginalization is done according to the Schur complement.

2.5 Obtaining a ground-truth trajectory

The ground-truth trajectory consists of absolute position estimates in world-coordinates X, Y, Z of the MAV. To observe these states a position in pixel-coordinates u, v is transformed to world-coordinates by formula 13 [10]. A stereo-camera is used in order to observe the required depth measurements Z. Calibration of the stereo-camera retrieves the calibration parameters a_x, a_y, u_0, v_0 to complete the transformation.

$$\begin{aligned} X &= \frac{u - u_0}{a_x} \times Z \\ Y &= \frac{v - v_0}{a_y} \times Z \end{aligned} \tag{13}$$

The ZED camera is used as stereo-camera to get precise position measurements, with a resolution of 2048x1080 pixels and at 15 fps. The tracking algorithm is originally implemented for a Microsoft Kinect camera to achieve successful feedback control on a Crazyflie platform [8]. The algorithm start by generating a static background based on a collection of images. Then for each pixel a mean and standard deviation are computed. A binary image illustrating the difference between incoming images and the static background reveals new objects that entered the field-of-view. To detect the correct object the size of each contours is combined with depth values and a size is estimated. An object with a size within the specified tolerance is labeled as the MAV.

2.6 Evaluation metrics for localization

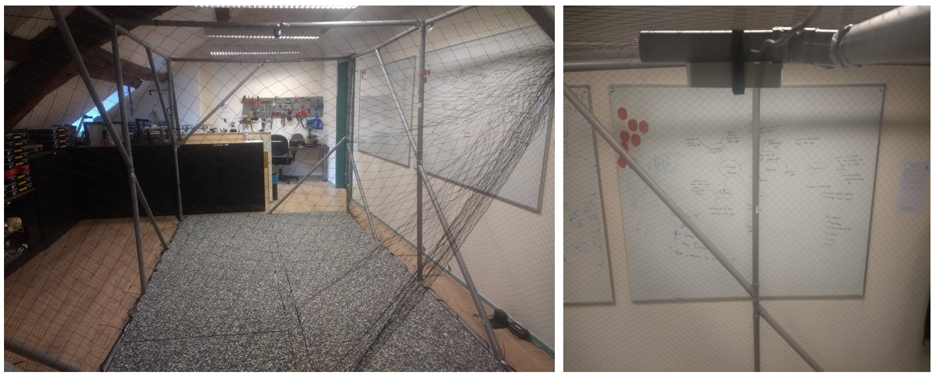
The localization estimates by each of the setups with corresponding methods are evaluated by their accuracy. To retrieve this accuracy an error is computed for each position using X, Y, Z coordinates between an estimated trajectory and the ground-truth. Before computing the errors the trajectories need to be aligned. Since there is no fixed offset between the ground-truth and an initialization of the estimated trajectory, which is different at each run. Aligning the trajectories is done according Sim(3) defined in formula 14. Returning a rotation $\boldsymbol{R}$, translation $\boldsymbol{t}$ and optional scaling factor $\boldsymbol{s}$. The scaling factor is usually required for monocular SLAM, because the scale is not directly opserved. An optimal Sim(3) alignment is computed by solving a least-squares estimator. For this 'evo' [23] is used, a python package for odometry and SLAM evaluation.

$$T = \left[\begin{array}{c|c} \boldsymbol{R} & \boldsymbol{t} \\ \hline \boldsymbol{0} & \boldsymbol{s}^{-1} \end{array}\right] \in \mathrm{Sim}(3) \qquad (14)$$
$$\boldsymbol{R} \in SO(3), \boldsymbol{t} \in \mathbb{R}^3, \boldsymbol{s} \in \mathbb{R}$$

After alignment of the trajectories two different errors are computed. The absolute pose error is calculated as an absolute trajectory error. By directly comparing estimated and ground-truth positions. Statistics can be calculated using these errors, which represent the global consistency of a trajectory. In addition a relative pose error compares deltas between two poses. Then a translational- or rotational drift can be computed to evaluate the local accuracy.

3 Precision ground-truth setup

Computing the precision of the ground-truth setup involves setting up the ZED-camera. This is mounted in the quad-copter safety cage, with it's optical center faced downwards figure 5.



(a) Quad-copter safety cage (b) ZED-camera faced downwards

Fig. 5: Ground-truth setup

3.1 Experiments

The experiment is executed by computing the difference between a relative distance from estimations and an absolute distance from measurements. For this the distance between wooden blocks of a fixed size are used, while estimating their position in world-coordinates. The blocks have a dimension of 10.6 cm x 3.9 cm x 3.9 cm and the ZED-camera is attached in the middle of the safety cage at a height of 2.05 m. Then estimated world-coordinates are retrieved by selecting points in a pointcloud, corresponding to the wooden blocks. An example of this pointcloud is illustrated in 6.

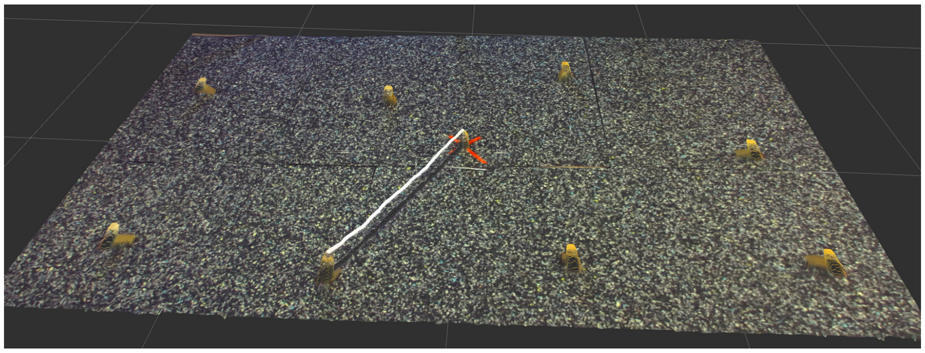
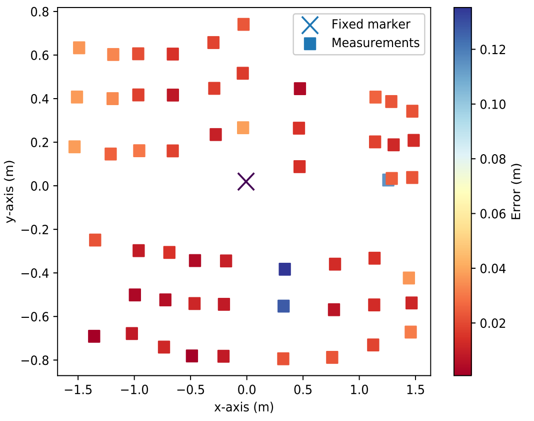


Fig. 6: Point cloud for position estimates in world-coordinates.

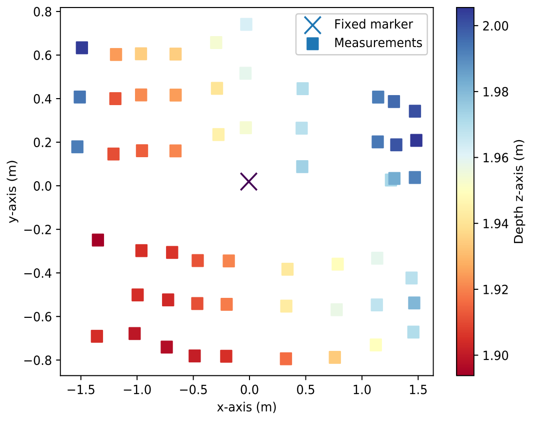
Additionally the distance from each block to the center marker is measured using a ruler. A comparison between the estimated- and measured distances represent the precision of the ZED-Camera. To achieve a high quality the ZED camera will be set at a resolution of 2048 x 1080 pixels which is fixed at 15 fps by the ZED-camera SDK. Other settings are an ultra quality with filling mode, making sure a depth measurement is computed for every pixel in the image.

3.2 Results

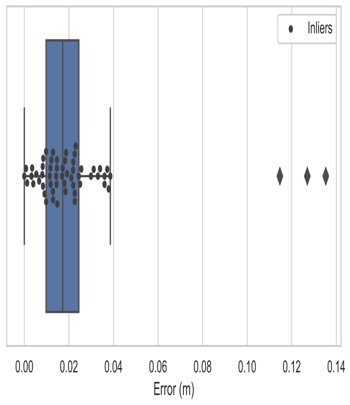
A total of 60 data points are collected with their estimated world-coordinates and measured distance to the center marker. Figures 7a and 7b show all blocks in world-coordinates, respectively with their error values and depth estimation. The depth values seem not to be equal at different positions of the floor. With a 10 cm difference in height between the lower left- and upper right corner. The error values however seem equally distributed over all positions of the image. Furthermore the boxplot in figure 7c illustrates the statistical analysis of the error values. With a mean value of 1.8 cm. There are also some clear outliers which an error value around 12 cm.



(a) Error between estimated and measured positions



(b) Depth estimation for each measurement



(c) Boxplot of error between estimated and measured positions

Fig. 7: Estimated block positions in world-coordinates by ZED-camera

3.3 Discussion

From the data that has been collected it is interesting that the different depth estimates across the map didn't influence the measurement error. Since projecting the pixel values back to the world-coordinate frame involves these depth values (formula 13). Hence the other explanation is that the floor inside the quad-copter cage isn't straight. Also a precision of +/- 1.8 cm isn't competitive to any of the external tracking systems. Since they offer an accuracy up to a few millimeters. However for the use-case within this paper it is sufficient.

4 Evaluation SLAM performance

The SLAM performance is evaluated for the estimated trajectories by the on-board sensors and using the Raspberry Pi sensor module. Using respectively a filtering- and smoothing based approach. The resulting trajectories are then compared with a ground-truth by the stereo-camera system.

4.1 Experiments

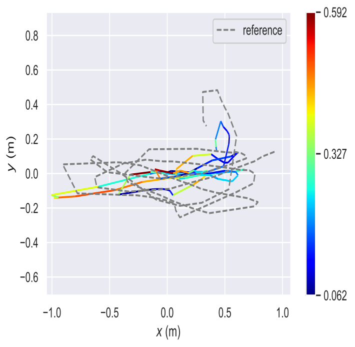
Several experiments are executed each attempting to fly a specific pattern. These flight patterns consist of a cross-, square- and circle shaped trajectory. While flying these trajectories are stored together with corresponding ground-truth. All positions in a trajectory are stored including timestamps, this is used for synchronization between an estimated- and ground-truth trajectory. Since both SLAM methods need some time to initialize. Hence the MAV is first flown outside reach of the ground-truth setup. Then when the SLAM is initialized one of the flight patterns is flown inside field-of-view of the stereo-camera. Now when analyzing the trajectories it's convenient to use matching timestamps from the first moment the MAV is within reach of the ground-truth setup. Besides synchronization the trajectories use a SIM(3) alignment as explained in 2.6.

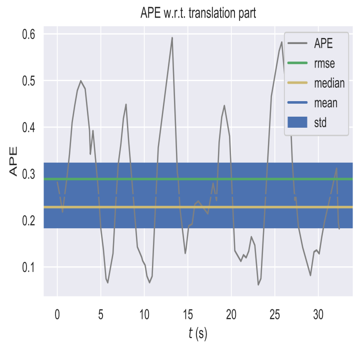
4.2 Results

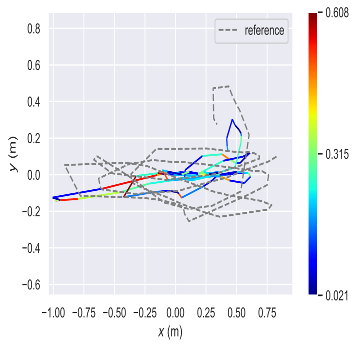
The results are split by the different flight patterns. Then for each flight pattern the trajectories are aligned with and without scaling factor. Both results are analyzed by an absolute- and relative-pose-error (2.6).

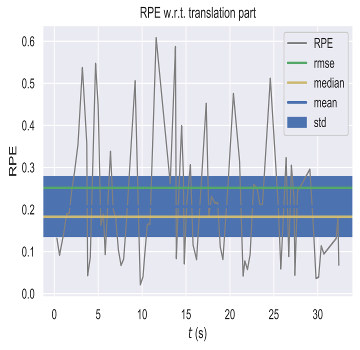
4.2.1 Cross-shape pattern

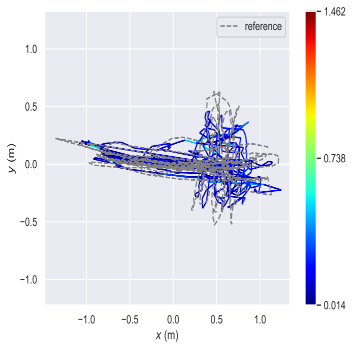
Results for the cross-shaped flight pattern with scale alignment are shown in figure 8 and without scale alignment in figure 9. Showing a lower root-mean-squared-error (RMSE) using the on-board sensors compared to the Raspberry Pi module for all error-metrics, with and without scale alignment. Furthermore the flight trajectory with on-board sensors has a longer time duration, three times the length of the Raspberry Pi module. Another result are higher velocities with the on-board sensors. This is seen in the APE statistics, with a larger number of oscillations for the on-board sensors. Lastly the scale estimation by the on-board sensors seems to be initialized smaller than the actual scale. This is seen in the difference between figures 8j and 9j

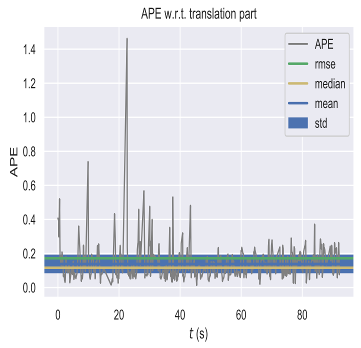

(a) Absolute-pose-error Raspberry Pi

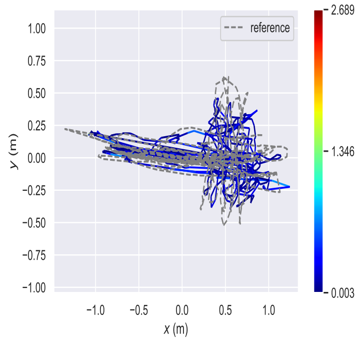

(b) Statistics APE Raspberry Pi

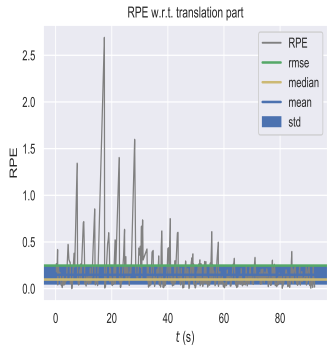

(c) Relative-pose-error Raspberry Pi

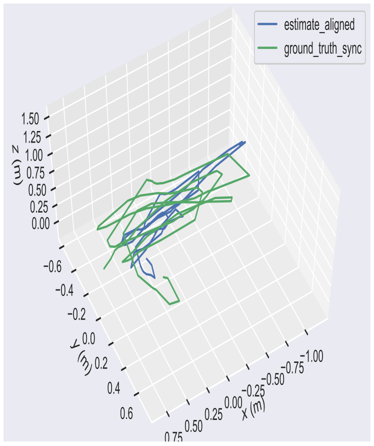

(d) Statistics RPE Raspberry Pi


(e) Absolute-pose-error on-board


(f) Statistics APE on-board


(g) Relative-pose-error on-board


(h) Statistics RPE on-board


(i) Raspberry Pi aligned trajectories

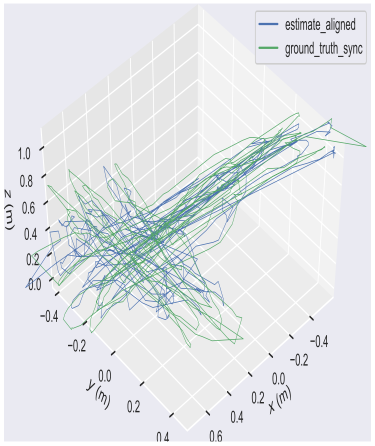
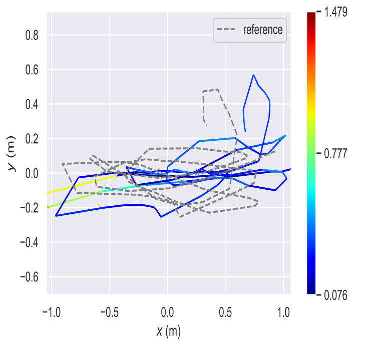
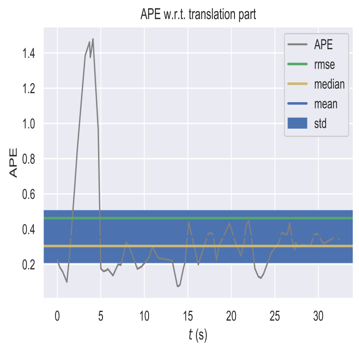
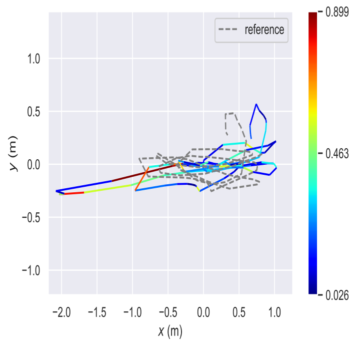
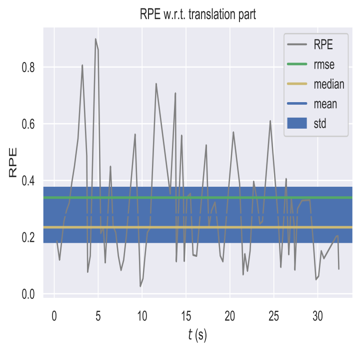

(j) On-board aligned trajectories

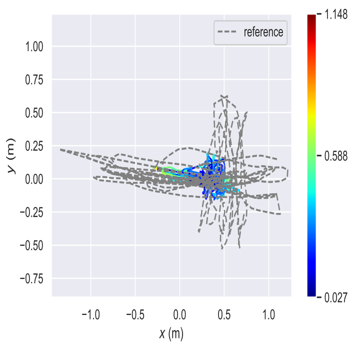
Fig. 8: Ground-truth and estimation for cross-shaped trajectories aligned by SIM(3) including scale, VINS-Mono with Raspberry Pi sensors and EKF with on-board sensors

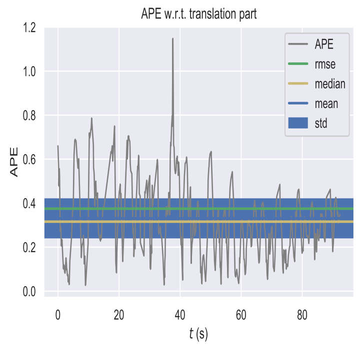

(a) Absolute-pose-error Raspberry Pi

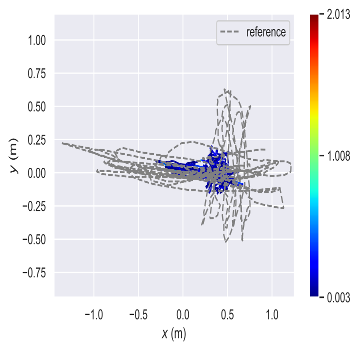

(b) Statistics APE Raspberry Pi

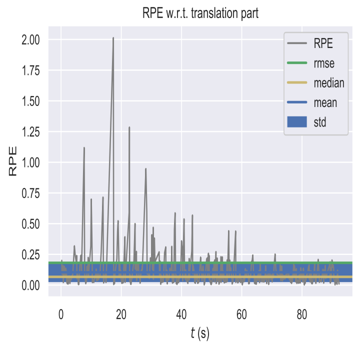

(c) Relative-pose-error Raspberry Pi

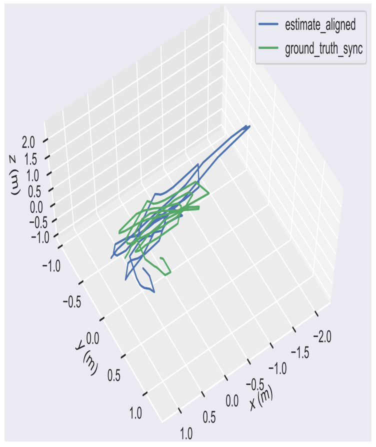

(d) Statistics RPE Raspberry Pi


(e) Absolute-pose-error on-board


(f) Statistics APE on-board


(g) Relative-pose-error on-board


(h) Statistics RPE on-board


(i) Raspberry Pi aligned trajectories

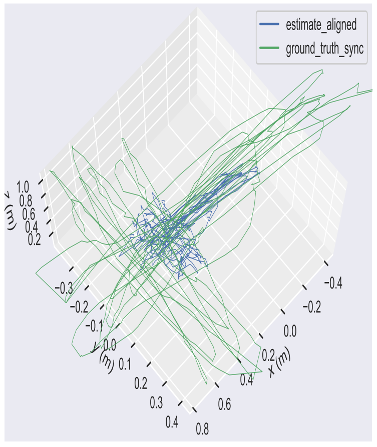
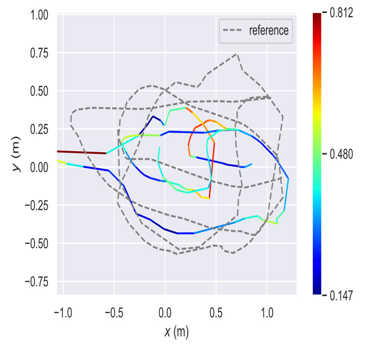

(j) On-board aligned trajectories

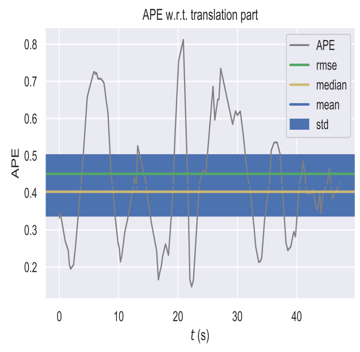
Fig. 9: Ground-truth and estimation for cross-shaped trajectories aligned by SIM(3) excluding scale, VINS-Mono with Raspberry Pi sensors and EKF with on-board sensors

4.2.2 Square-shaped pattern

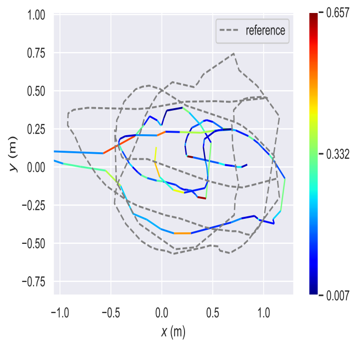
Results for the square-shaped flight pattern with scale alignment are shown in figure 10 and without scale alignment in figure 11. The trajectories lengths flown for this pattern are much more equal than with the other patterns. With just a difference of approximately 10 seconds longer by the on-board sensors. Though the on-board sensors give higher peaks for both APE as RPE. But overall the RMSE look similar for both sensor setups. Looking at the aligned trajectories with no scaling it seems the on-boards sensors give a correct estimation of the true scale. Since the error values are similar as with the scaled trajectory. The Raspberry Pi sensors give a higher error when not considering the scaling factor.



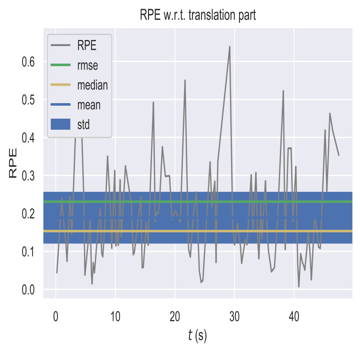
(a) Absolute-pose-error Raspberry Pi



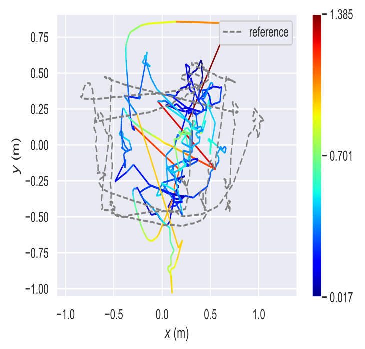
(b) Statistics APE Raspberry Pi



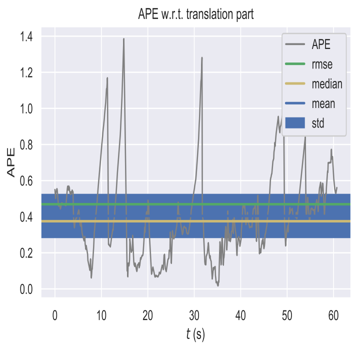
(c) Relative-pose-error Raspberry Pi



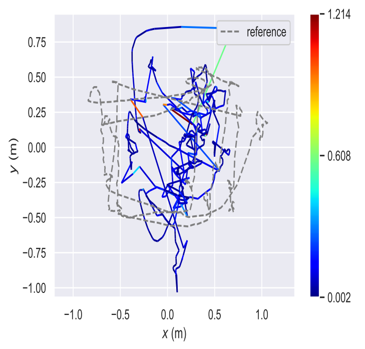
(d) Statistics RPE Raspberry Pi



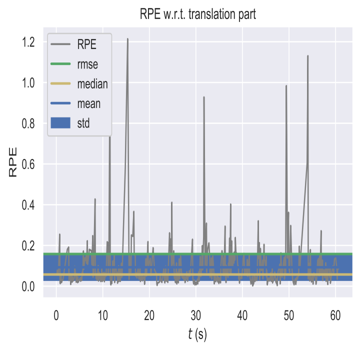
(e) Absolute-pose-error on-board



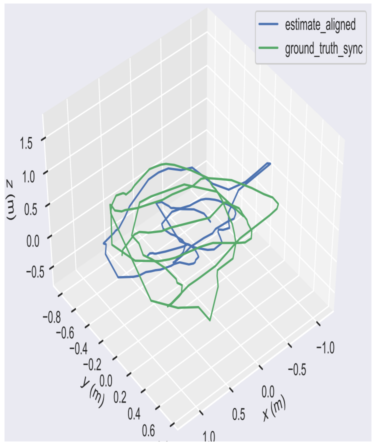
(f) Statistics APE on-board



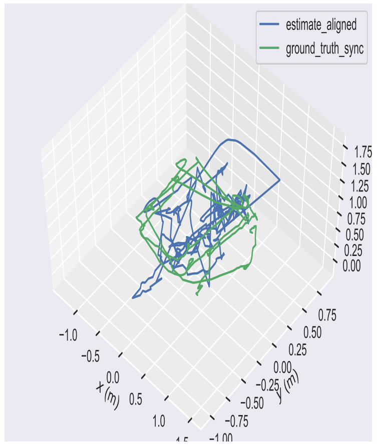
(g) Relative-pose-error on-board



(h) Statistics RPE on-board

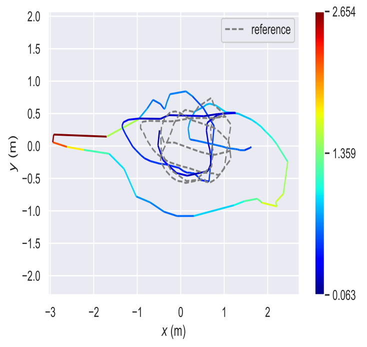


(i) Raspberry Pi aligned trajectories

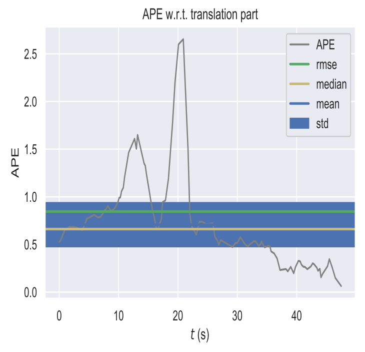


(j) On-board aligned trajectories

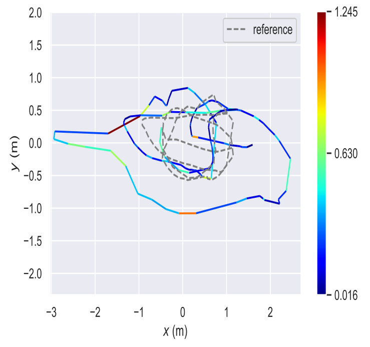
Fig. 10: Ground-truth and estimation for square-shaped trajectories aligned by SIM(3) including scale, VINS-Mono with Raspberry Pi sensors and EKF with on-board sensors



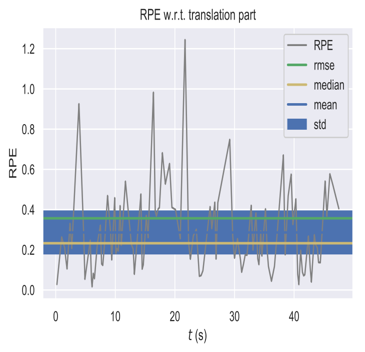
(a) Absolute-pose-error Raspberry Pi



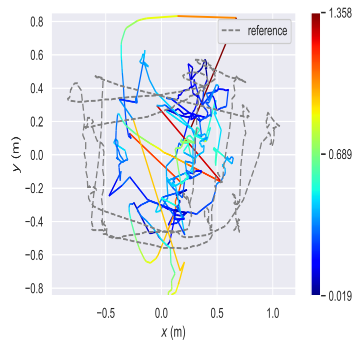
(b) Statistics APE Raspberry Pi



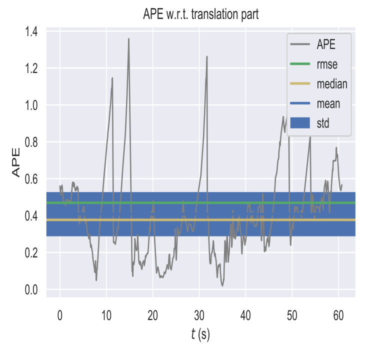
(c) Relative-pose-error Raspberry Pi



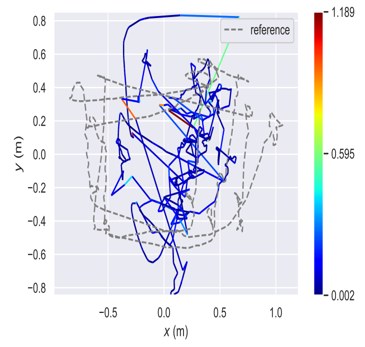
(d) Statistics RPE Raspberry Pi



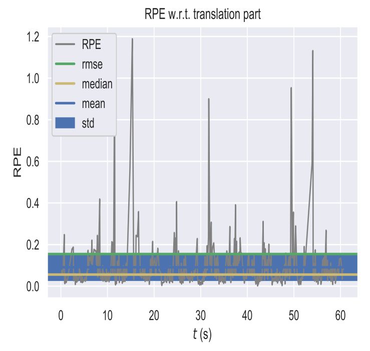
(e) Absolute-pose-error on-board



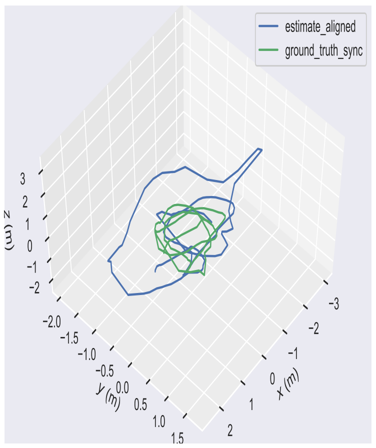
(f) Statistics APE on-board



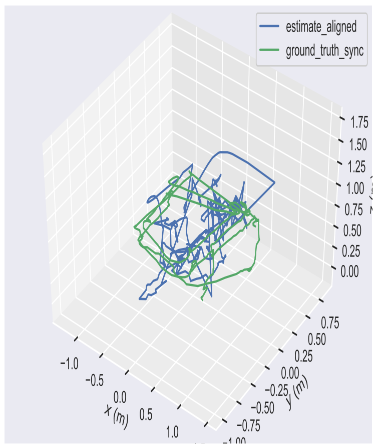
(g) Relative-pose-error on-board



(h) Statistics RPE on-board



(i) Raspberry Pi aligned trajectories

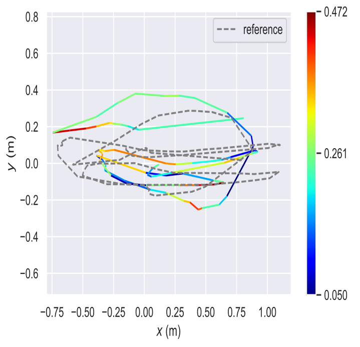


(j) On-board aligned trajectories

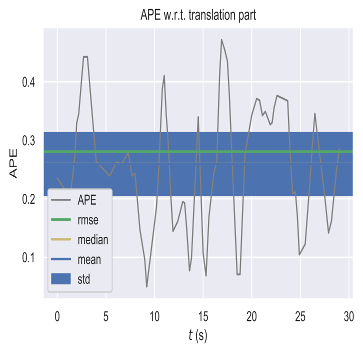
Fig. 11: Ground-truth and estimation for square-shaped trajectories aligned by SIM(3) excluding scale, VINS-Mono with Raspberry Pi sensors and EKF with on-board sensors

4.2.3 Circle-shaped pattern

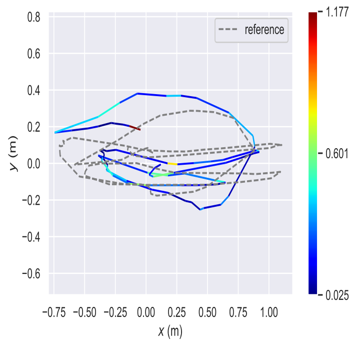
Results for the circle-shaped flight pattern with scale alignment are shown in figure 12 and without scale alignment in figure 13. They are similar to the previous patterns. Showing a lower RMSE in all comparisons. As well as an higher velocity indicated by the oscillations and a lower estimated- than true scaling.



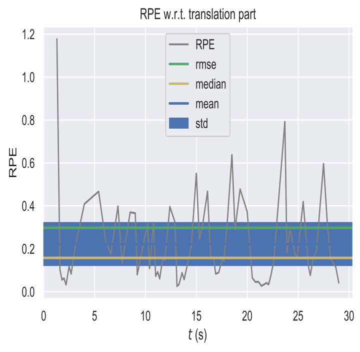
(a) Absolute-pose-error Raspberry Pi



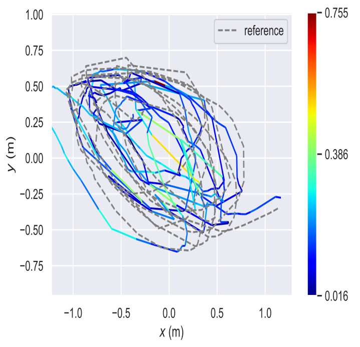
(b) Statistics APE Raspberry Pi



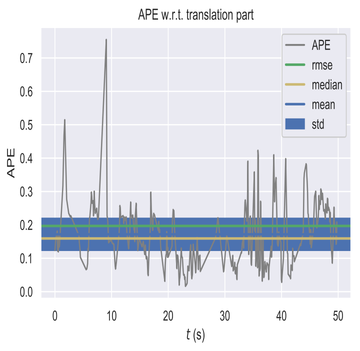
(c) Relative-pose-error Raspberry Pi



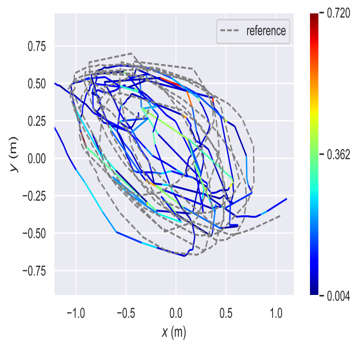
(d) Statistics RPE Raspberry Pi



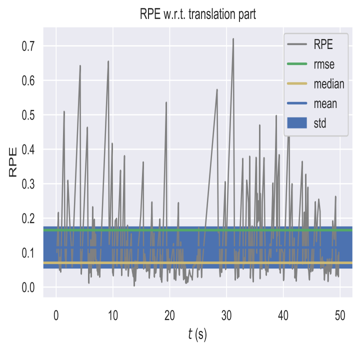
(e) Aboslute-pose-error on-board



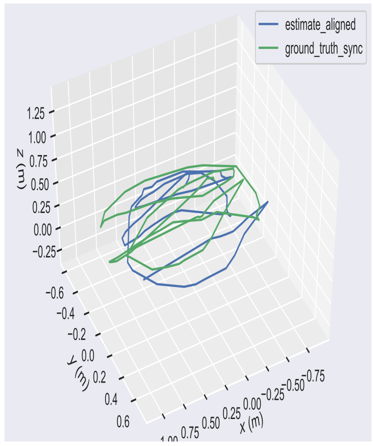
(f) Statistics APE on-board



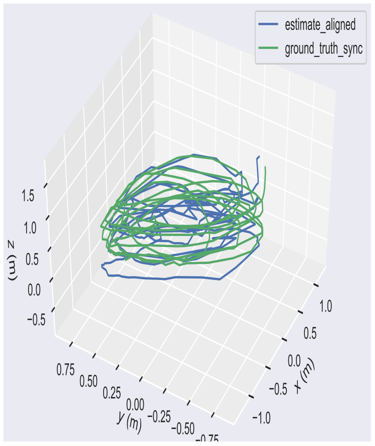
(g) Relative-pose-error on-board



(h) Statistics RPE on-board

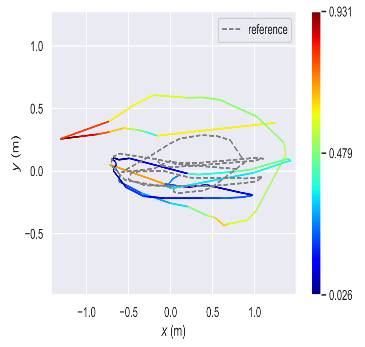


(i) Raspberry Pi aligned trajectories

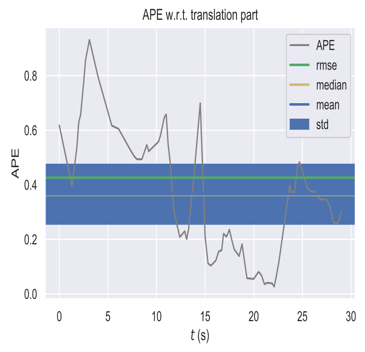


(j) On-board aligned trajectories

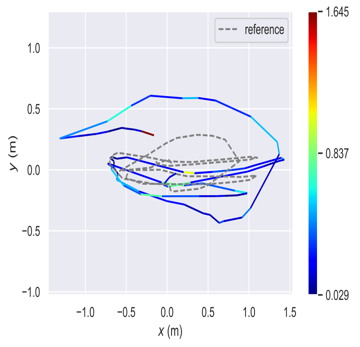
Fig. 12: Ground-truth and estimation for circle-shaped trajectories aligned by SIM(3) including scale, VINS-Mono with Raspberry Pi sensors and EKF with on-board sensors



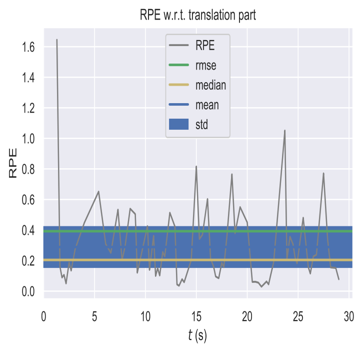
(a) Absolute-pose-error Raspberry Pi



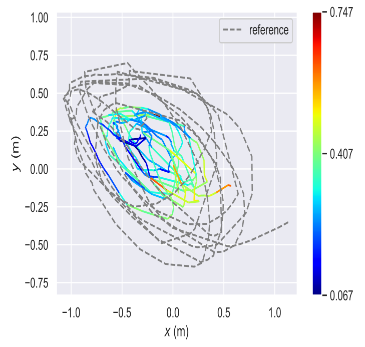
(b) Statistics APE Raspberry Pi



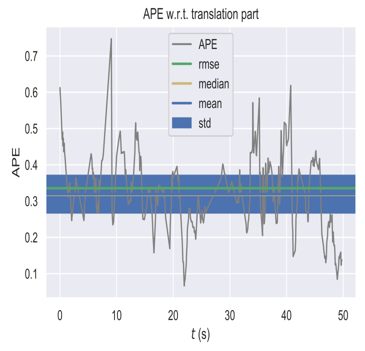
(c) Relative-pose-error Raspberry Pi



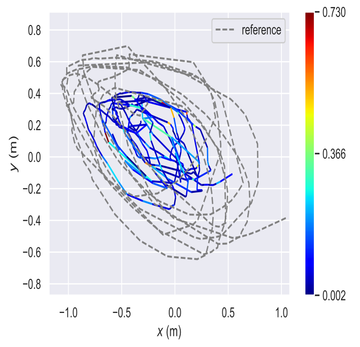
(d) Statistics RPE Raspberry Pi



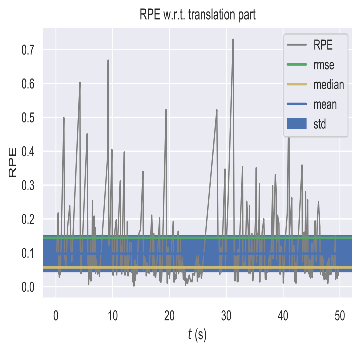
(e) Aboslute-pose-error on-board



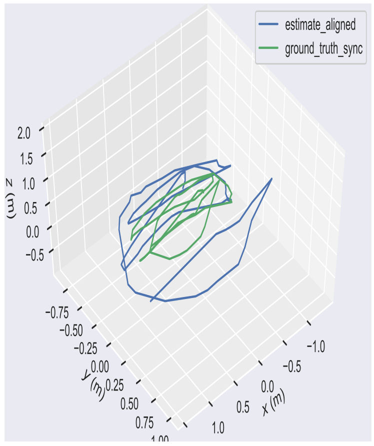
(f) Statistics APE on-board



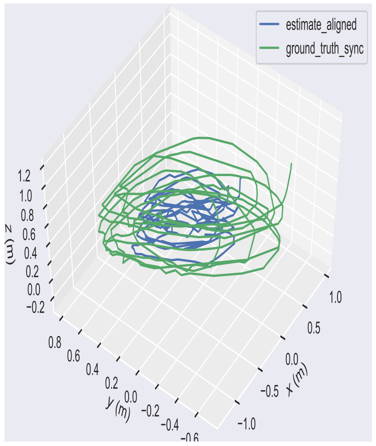
(g) Relative-pose-error on-board



(h) Statistics RPE on-board



(i) Raspberry Pi aligned trajectories



(j) On-board aligned trajectories

Fig. 13: Ground-truth and estimation for circle-shaped trajectories aligned by SIM(3) excluding scale, VINS-Mono with Raspberry Pi sensors and EKF with on-board sensors

4.3 Discussion

Summarizing the results it's fair to say the on-board sensors are more accurate than the Raspberry Pi module. However it has to be taken into account that the Raspberry Pi module is still a prototype. Making it difficult to run experiments due to a loose connection between the battery and the Raspberry, resulting in a reboot. Also mounting the sensor module isn't optimal, resulting in unstable flight dynamics. The VINS-Mono smoothing method also describes that using a global-shutter camera gives much better results than the rolling-shutter camera, used for the sensor-module. Another important aspect is the inaccurate time synchronization between the IMU and camera on the sensor module. The timestamps are now computed when being transmitted by the Raspberry Pi, instead of the actual time of retrieving the image or sensor measurements. A last aspect of this experiment is that both sensor setups are tested by different SLAM implementations. Hence for a better comparison might be done by applying the same SLAM method on both setups. Though this isn't possible for the VINS-Mono approach because it requires raw IMU measurements at a high frequency above 100 Hz.

5 Conclusion

In this work an overview is retrieved of available MAV platforms within restrictions of: a weight below 100 grams, financial costs less than €100 and an open-source SDK. This resulted in the Parrot Mambo and DJI Tello, with the Mambo as final test platform. Focusing at the on-board sensor setup of the Parrot Mambo it became clear not any SLAM solution can be used. Recent Bayesian smoothing based packages were not suitable for the on-board sensors of the MAV platform. Due to a flight-controller only offering limited access through the SDK. With a the publishing of sensor data at a low frequency of 2 Hz. Traditional EKF based solutions turned out to be more promising. Through previous studies that successfully demonstrated autonomous flying on commercially available platforms using the TUM AR.Drone package. Modifying the package also turned out to be suitable for the MAV platform in this work. By developing an additional sensor module it became possible to test the SLAM package based on Bayesian smoothing. Eventually both SLAM approaches were tested. Using a stereo-camera to retrieve a ground-truth and compute a comparison with the estimated trajectories. The ground-truth setup is tested to be precise up to approximately +/- 1.8 cm. The results of a comparison between SLAM methods turned out different than expected. With an higher accuracy for the traditional EKF method than the modern Bayesian smoothing in every experiment. Though it is clear the Raspberry Pi sensor module is not idea in it's current state. With 3 major improvements to do. First of all properly fixing the module onto the MAV. Also a proper time synchronization between IMU and camera is essential. And finally the use of a global-shutter camera over the current rolling-shutter. Considering the localization results it's not decisive to use the Parrot Mambo as a final choice as autonomous flying platform. It would still be interesting to test the alternative DJI Tello. Which transmits sensor data at a higher frequency of 10 Hz. A final step would be to test both platforms with the current EKF implementation. Autonomously flying to set waypoints using a simple PID controller, which is already part of the Parrot AR.Drone package.

References

[1] Crazyflie – position control, university of augsburg. access: 6.02.2019.

[2] Regulatory framework to accommodate unmanned aircraft systems in the european aviation system. access: 6.02.2017.

[3] Skydio – advanced autonomous drone in the world. access: 6.02.2019.

[4] *The Navigation and Control Technology Inside the AR.Drone Micro UAV*, Milano, Italy, 2011.

[5] Sameer Agarwal and Keir Mierle. *Ceres Solver: Tutorial & Reference*. Google Inc.

[6] Jeffrey A. Delmerico and Davide Scaramuzza. A benchmark comparison of monocular visual-inertial odometry algorithms for flying robots. *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2502–2509, 2018.

[7] O. Dunkley, J. Engel, J. Sturm, and D. Cremers. Visual-inertial navigation for a camera-equipped 25g nano-quadrotor. In *IROS2014 Aerial Open Source Robotics Workshop*, 2014.

[8] Oliver Montague Welton Dunkley. Visual inertial control of a nano-quadrotor. Master's thesis, Technical University Munich, Germany, Sept. 2014.

[9] Jakob Engel and Daniel Cremers. Scale-aware navigation of a lowcost quadrocopter with a monocular camera. In *Robotics and Autonomous Systems (RAS*, 2014.

[10] David A. Forsyth and Jean Ponce. *Computer Vision: A Modern Approach*. Prentice Hall Professional Technical Reference, 2002.

[11] J. Fossel, D. Hennes, D. Claes, S. Alers, and K. Tuyls. Octoslam: A 3d mapping approach to situational awareness of unmanned aerial vehicles. In *2013 International Conference on Unmanned Aircraft Systems (ICUAS)*, pages 179–188, May 2013.

[12] Xingyin Fu, Feng Zhu, Qingxiao Wu, Yunlei Sun, Rongrong Lu, and Ruigang Yang. Real-time large-scale dense mapping with surfels. In *Sensors*, 2018.

[13] F. Gao, Y. Lin, and S. Shen. Gradient-based online safe trajectory generation for quadrotor flight in complex environments. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3681–3688, Sep. 2017.

[14] F. Gao, W. Wu, Y. Lin, and S. Shen. Online safe trajectory generation for quadrotors using fast marching method and bernstein basis polynomial. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 344–351, May 2018.

[15] Albert S. Huang, Abraham Bachrach, Peter Henry, Michael Krainin, Daniel Maturana, Dieter Fox, and Nicholas Roy. Visual odometry and mapping for autonomous flight using an rgb-d camera. In *ISRR*, 2011.

[16] M. Kaess, A. Ranganathan, and F. Dellaert. iSAM: Incremental smoothing and mapping. *IEEE Trans. on Robotics (TRO)*, 24(6):1365–1378, December 2008.

[17] Georg Klein and David Murray. Parallel tracking and mapping for small AR workspaces. In *Proc. Sixth IEEE and ACM International Symposium on Mixed and Augmented Reality (ISMAR'07)*, Nara, Japan, November 2007.

[18] R. Kümmerle, G. Grisetti, H. Strasdat, K. Konolige, and W. Burgard. G2o: A general framework for graph optimization. In *2011 IEEE International Conference on Robotics and Automation*, pages 3607–3613, May 2011.

[19] Yi Lin, Fei Gao, Tong Qin, Wenliang Gao, Tianbo Liu, William Wu, Zhenfei Yang, and Shaojie Shen. Autonomous aerial navigation using monocular visual-inertial fusion: Lin et al. *Journal of Field Robotics*, 35, 07 2017.

[20] Yonggen Ling, Tianbo Liu, and Shaojie Shen. Aggressive quadrotor flight using dense visual-inertial fusion. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1499–1506, May 2016.

[21] A. McGovern. *PyParrot*, 2018.

[22] K. McGuire, G. de Croon, C. De Wagter, K. Tuyls, and H. Kappen. Efficient optical flow and stereo vision for velocity estimation and obstacle avoidance on an autonomous pocket drone. *IEEE Robotics and Automation Letters*, 2(2):1070–1076, April 2017.

[23] G. Michael. *evo*, 2018.

[24] T. Qin, P. Li, and S. Shen. Vins-mono: A robust and versatile monocular visual-inertial state estimator. *IEEE Transactions on Robotics*, 34(4):1004–1020, Aug 2018.

[25] Morgan Quigley, Brian P. Gerkey, Ken Conley, Josh Faust, Tully Foote, Jeremy Leibs, Eric Berger, Rob Wheeler, and Andrew Ng. Ros : an open-source robot operating system. 2009.

[26] S. Shen, N. Michael, and V. Kumar. Tightly-coupled monocular visual-inertial fusion for autonomous flight of rotorcraft mavs. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5303–5310, May 2015.

[27] S. Shen, N. Michael, and V. Kumar. Tightly-coupled monocular visual-inertial fusion for autonomous flight of rotorcraft mavs. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5303–5310, May 2015.

[28] Jianbo Shi and Tomasi. Good features to track. In *1994 Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pages 593–600, June 1994.

[29] Simo Särkkä. *Bayesian Filtering and Smoothing.* Institute of Mathematical Statistics Textbooks. Cambridge University Press, 2013.

[30] Carlo Tomasi and Takeo Kanade. Detection and tracking of point features. Technical report, International Journal of Computer Vision, 1991.

[31] L. von Stumberg, V. Usenko, J. Engel, J. Stückler, and D. Cremers. From monocular slam to autonomous drone exploration. In *2017 European Conference on Mobile Robots (ECMR)*, pages 1–8, Sep. 2017.

[32] Chen Wang, Handuo Zhang, Thien-Minh Nguyen, and Lihua Xie. Ultra-wideband aided fast localization and mapping system. 09 2017.

[33] K. Wang, W. Ding, and S. Shen. Quadtree-accelerated real-time monocular dense mapping. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1–9, Oct 2018.