Snake-SLAM: Efficient Global Visual Inertial SLAM using Decoupled Nonlinear Optimization

Darius Rückert[1] and Marc Stamminger[1]

***Abstract*— Snake-SLAM is a scalable visual inertial SLAM system for autonomous navigation in low-power aerial devices. The tracking front-end features map reuse, loop closing, relocalization, and supports monocular, stereo, and RGBD input. The keyframes are reduced by a graph-based simplification approach and further refined using a novel deferred mapping stage to ensure a sparse yet accurate global map. The optimization back-end decouples IMU state estimation from visual bundle adjustment and solves them separately in two simplified sub problems. This greatly reduces computational complexity and allows Snake-SLAM to use a larger local window size than existing SLAM methods. Our system implements a novel multi-stage VI initialization scheme, which uses gyroscope data to detect visual outliers and recovers metric velocity, gravity, and scale. We evaluate Snake-SLAM on the EuRoC dataset and show that it outperforms all other approaches in efficiency while also achieving state-of-the-art tracking accuracy.**

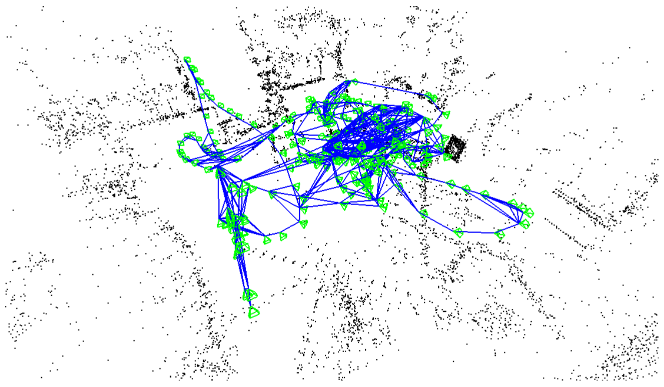
Fig. 1: Estimated trajectory and global map of the EuRoC V13 sequence.

I. INTRODUCTION

Multi-sensor pose estimation is at the core of unmanned aircraft systems. It allows drones and robots to localize themselves in an unknown environment only from sensor input. Special interest lies in visual inertial (VI) SLAM systems because metric scale is recovered and the global map ensures long-term drift-free localization. Using such a system allows autonomously discovering drones, for example, to return to the starting location with an error of less than 10 cm. However, running such methods on low power devices comes at a cost [1]. The high number of parameters slow down optimization-based methods and currently available systems have to reduce tracking quality to match real time constraints.

In this work, we fill this gap by releasing Snake-SLAM, which is able to run on battery powered aerial devices as well as high-end desktop computers. The good scalability, which the name *Snake* refers to, is a result of algorithmic improvements and an efficient parallel implementation. Snake-SLAM features large scale loop closure, relocalization, and supports monocular, RGBD, and stereo input. All IMU states, consisting of scale, bias, gravity, and velocity, are estimated on the fly allowing a metric reconstruction from monocular camera input. Snake-SLAM implements various improvements to the feature-based SLAM pipeline, for example, a novel map simplification technique and an additional deferred mapping stage. We also present a decoupled optimization approach for global VI-SLAM, which estimates the required parameters in two separate subproblems. This allows a very accurate pose estimation while also achieving better computational efficiency than existing approaches. In summary, the contributions of our work are:

- A decoupled VI-SLAM formulation (Section IV)
- A robust VI initialization scheme that uses gyroscope data to detect outlier keyframes (Section IV-B).
- A graph and confidence-based map simplification (Section V).
- The integration of deferred local mapping into real-time SLAM (Section VI).
- A highly efficient open-source implementation that outperforms current state-of-the-art methods (Section VII and VIII).

II. RELATED WORK

In recent years, many different visual SLAM systems have been published. In this section, we differentiate between filter-and optimization-based, between direct and indirect, and between sliding window and global map approaches. The input may differ between monocular, RGBD, and stereo cameras and some approaches make use of additional sensors, such as IMU and GPS.

Filter-based methods rely on the Extended Kalman Filter (EKF) to estimate camera pose and landmark positions simultaneously [2], [3]. As the filter complexity grows quadratically with the number of landmarks, only few points are tracked during real-time operation [4]. A structureless approach, using the Multi-State Constraint Kalman Filter (MSCKF) [5], allows marginalizing the landmarks out of the state vector. Lynen et al. [6] track feature points and estimate the camera pose using a visual inertial sliding window estimator based on MSCKF. Sun et al. [7] extend this concept to stereo input and show good results on fast moving aerial devices.

[1]Visual Computing, University of Erlangen-Nurenberg, Germany
darius.rueckert@fau.de, marc.stamminger@fau.de

Optimization-based approaches formulate the pose estimation problem as a nonlinear cost function. LSD-SLAM [8] aligns consecutive frames directly by minimizing the photometric error of image patches. The absence of traditional corner and blob-like features allows LSD-SLAM to work even when only image gradients are available. SVO [9] uses similar direct image alignment but falls back to a feature extractor on selected keyframes. DSO [10] reduces complexity of the photometric minimization by selecting only a sparse set of patches distributed evenly over the image. The pose is optimized in a fixed-size sliding window. Variants of DSO include IMU measurements [11], detect and close large scale loops [12], support stereo inputs [13], and add compensation for rolling shutter effects [14].

Optimization-based approaches with feature points rely on bundle adjustment (BA), which minimizes the reprojection error between estimated 3D points and the 2D image features. OKVIS [15] extracts Harris corner points [16] and matches them in a brute-force manner to a local map in a sliding window. The camera pose is computed by solving a nonlinear optimization problem including the landmark observations and IMU measurements. Qin et al. follow a similar approach with VINS-Mono [17] but track features using the KLT sparse optical flow algorithm [18] and implement a more robust initialization phase. They also include large scale loop closure based on BoW place recognition [19] and additional BRIEF descriptors [20] on keyframes. In a second publication, they extend their algorithm to stereo cameras and optional GPS input [21].

Instead of using a fixed size sliding window, the current frame's pose can also be estimated by aligning it to a global map. This map is usually refined over time and can grow indefinitely. Several methods [22]–[24] use RGBD input to build a dense volumetric model of the scene. New frames are then aligned by rendering the model and applying the iterative closest point algorithm [25]. The seminal work of Klein et al. show with PTAM [26] that the idea of aligning a novel frame to a global map can be implemented for monocular camera input. In each frame, the existing 3D points are projected into the predicted camera frame and matched to local 2D features. Using these correspondences, the pose is refined by nonlinear optimization. In a background thread, a local mapping procedure finds new epipolar matches between keyframes and applies local bundle adjustment (LBA). The more sophisticated ORB-SLAM2 [27] follows a concept similar to PTAM but furthers improves accuracy and is able to process monocular, RGBD, and stereo input. For each frame, FAST corner points [28] with associated ORB feature descriptors [29] are extracted. The novel frame is aligned to the global map and keyframes are inserted based on a heuristic. In contrast to PTAM, they also remove redundant keyframes and are able to close large scale loops using BoW place recognition [19]. It has been shown that ORB-SLAM2 achieves very high accuracy [30] but also is computational demanding due to the global nature and per-frame ORB-feature extraction. VI-ORBSLAM [31] makes use of the IMU to further improve tracking accuracy and is able to

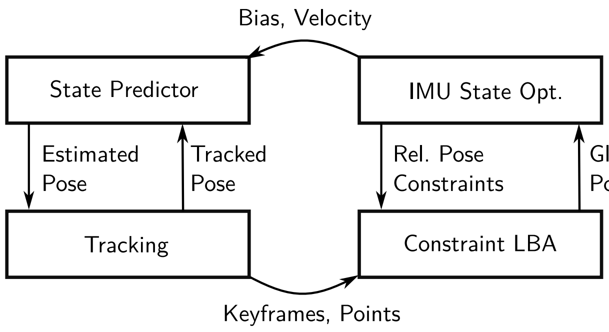

Fig. 2: High-level overview of Snake-SLAM, showing the tracking front-end on the left and the decoupled optimization back-end on the right.

estimate scale and gravity direction. However, VI-ORBSLAM takes 15 seconds of visual-only tracking to initialize and they do not provide source code. ORB-SLAM3 [32] is similar to VI-ORBSLAM with multiple minor improvements and public source code. They show convincing results on various datasets, featuring a very low absolute pose error and high robustness.

In recent years, researchers have experimented with replacing stages of a traditional tracking pipeline with deep neural networks. In VINet [33], the stream of monocular RGB images and IMU measurements is processed by large neural networks in a LSTM-like manner. The output is the relative transformation of the new frame towards the previous frame. Other approaches, such as [34]–[37], also predict depth images in addition to the relative image transformation. This improves tracking accuracy, because the network is guided to learn a geometric representation of the 2D image. The main drawback of current deep learning tracking methods is that they are computationally very demanding and do not build a globally consistent map.

Snake-SLAM, which is presented in the following chapters, uses a global algorithm similar to VI-ORBSLAM, but includes a faster and more robust initialization, the optimization is decoupled, and our system runs in real-time on mobile hardware.

III. SYSTEM OVERVIEW

Snake-SLAM is a full VI-SLAM pipeline taking monocular, RGBD, or stereo images as input and computing a pose estimate for every frame. The frames are aligned to a globally consistent map of keyframes and 3D points. Our contributions, i.e. the decoupled VI solver (Section IV), map simplification (Section V) and deferred mapper (Section VI), are presented in-depth after this overview.

For every new frame, we extract ORB features on the unrectified grayscale image. If the input is set to stereo, matches are found between both images and the depth is added to the keypoint. After feature preprocessing, the novel pose is predicted from the previous keyframe, the last frame's position, and the current IMU state. Global 3D points are then projected to the predicted pose and matched to the extracted keypoints. Using these matches, the pose is refined by nonlinear optimization. A new keyframe is inserted into the map, if tracking is weak or more than 0.5 s have passed since the previous keyframe was added. However, if no camera motion is measured, keyframes are not inserted to avoid flooding the map with redundant information. After keyframe insertion, new 3D points are created using triangulation and existing map points are fused if they project to the same 2D feature. The optimization back-end (see Figure 2 right) refines the map, by first computing the IMU state variables and then solving a constraint local bundle adjustment (BA). Non-essential keyframes are culled using our graph-based heuristic (see Section V) and further refined in the deferred mapper (see Section VI). The loop closing module detects loop candidates and closes them using pose graph optimization [38].

IV. DECOUPLED VISUAL INERTIAL OPTIMIZATION

Snake-SLAM incorporates an optimization-based mapping back-end, which minimizes the visual error $\mathbf{E}_{\text{VIS}}$ and IMU error $\mathbf{E}_{\text{IMU}}$ on a global map of keyframes and 3D points. The state of a keyframe consists of the camera pose $\mathbf{P} \in \mathbf{SE}(3)$, velocity $\mathbf{v} \in \mathbb{R}^3$, and IMU biases $\mathbf{b}_a \in \mathbb{R}^3, \mathbf{b}_g \in \mathbb{R}^3$. Each map point stores its world space position $\mathbf{x} \in \mathbb{R}^3$ and we keep track of the global gravity direction $\mathbf{g} \in \mathbb{R}^3$ and a scale factor $s \in \mathbb{R}$. After an initialization phase, this state is estimated iteratively by running a few iterations of a Levenberg-Marquardt solver each time a new keyframe is inserted. To reduce the complexity of the nonlinear problem, we separate the state estimation into two subproblems. First, we estimate the IMU states consisting of velocities, biases, gravity direction, and scale using the IMU error term between two consecutive keyframes i and $i+1$:

$$\underset{\mathbf{b}_a,\mathbf{b}_g,\mathbf{v},\mathbf{g},s}{\operatorname{argmin}} \sum_{i}^{n-1} ||\mathbf{E}_{\text{IMU}}(i,i+1)||^2 \tag{1}$$

After that, we use the estimated IMU parameters to compute the relative transformations $\Delta\mathbf{T}_i$ mapping from camera i to camera $i+1$. These relative transformations are added as constraints into visual bundle adjustment (BA) resulting in the second nonlinear optimization problem:

$$\underset{\mathbf{P},\mathbf{x}}{\operatorname{argmin}} \sum_{i}^{n} \sum_{obs(i)} ||\mathbf{E}_{\text{VIS}}(i,obs)||^2 + \sum_{i}^{n-1} ||\mathbf{E}_{\text{REL}}(i,i+1,\Delta\mathbf{T}_i)||^2 \tag{2}$$

$$\underset{\mathbf{b}_a,\mathbf{b}_g,\mathbf{v},\mathbf{g},s,\mathbf{P},\mathbf{x}}{\operatorname{argmin}} \sum_{i}^{n} \sum_{obs(i)} ||\mathbf{E}_{\text{VIS}}(i,obs)||^2 + \sum_{i}^{n-1} ||\mathbf{E}_{\text{IMU}}(i,i+1)||^2 \tag{3}$$

The separation into (1) and (3) is very efficient because (1) is almost linear and (3) has the same complexity as visual-only BA. The linearity of (1) allows us to update the IMU states with a single LM iteration after each keyframe. The additional relative pose constraint in (3) does not increase BA complexity because neighboring keyframes also have common map points. This means, after constructing the sparse Schur complement $\mathbf{S}_{\mathbf{P}}$, the mixed Jacobians are added to existing non-zero entries.

In the following section, we describe the error terms used in (1) and (3) in more detail. After that, we highlight our robust initialization scheme and explain the pose estimation of new frames.

A. Visual Inertial Error Terms

The inertial measurement unit (IMU) records metric linear acceleration $\mathbf{a}$ and angular velocity ω at regular time steps Δt. To construct the IMU error term, we preintegrate the IMU measurements between two consecutive keyframes following the approach of [39]. Given an IMU measurement $(\mathbf{a},\omega)$ the new rotation $\mathbf{R}$, position $\mathbf{p}$ and velocity $\mathbf{v}$ can be computed using the current accelerometer and gyroscope bias $(\mathbf{b}_a,\mathbf{b}_g)$:

$$\begin{aligned}
\mathbf{R}_{k+1} &= \mathbf{R}_k \exp((\omega - \mathbf{b}_g)\Delta t) \\
\mathbf{v}_{k+1} &= \mathbf{v}_k + (\mathbf{a} - \mathbf{b}_a)\Delta t \\
\mathbf{p}_{k+1} &= \mathbf{p}_k + \mathbf{v}_k\Delta t + \frac{1}{2}\mathbf{R}_k(\mathbf{a} - b_a)\Delta t^2
\end{aligned} \tag{4}$$

By initializing $\mathbf{R}_0, \mathbf{v}_0, \mathbf{p}_0$ to identity and iterating Eq. (4) over all measurements between keyframe i and j, we get the predicted delta rotation $\Delta\mathbf{R}_{ij}$, position $\Delta\mathbf{p}_{ij}$, and velocity $\Delta\mathbf{v}_{ij}$. After integration of the gravity $\mathbf{g}$ and initial velocity $\mathbf{v}_i$, the predicted state $(\mathbf{R}_j^{est}, \mathbf{v}_j^{est}, \mathbf{p}_j^{est})$ of keyframe j is:

$$\begin{aligned}
\mathbf{R}_j^{est} &= \mathbf{R}_i\Delta\mathbf{R}_{ij} \\
\mathbf{v}_j^{est} &= \mathbf{v}_i + \mathbf{g}\Delta t + \mathbf{R}_i\Delta\mathbf{v}_{ij} \\
\mathbf{p}_j^{est} &= \mathbf{p}_i + \mathbf{v}_i\Delta t + \frac{1}{2}\mathbf{g}\Delta t^2 + \mathbf{R}_i\Delta\mathbf{p}_{ij}
\end{aligned} \tag{5}$$

The IMU induced errors of rotation, velocity, and position are the difference between the predicted state of Eq. (5) and current state of keyframe j.

$$\begin{aligned}
\mathbf{E}_{\text{R}}(i,j) &= \log(\mathbf{R}_j^{est}\mathbf{R}_j^{-1}) \\
\mathbf{E}_{\text{v}}(i,j) &= \mathbf{v}_j^{est} - \mathbf{v}_j \\
\mathbf{E}_{\text{p}}(i,j) &= \mathbf{p}_j^{est} - \mathbf{p}_j
\end{aligned} \tag{6}$$

The final IMU error which is used to solve equation (1) is obtained by inserting the weights γ that capture the noise characteristics of the IMU sensor and scaling everything by the inverse time delta.

$$\mathbf{E}_{\text{IMU}}(i,j) = \frac{1}{\Delta t} \begin{bmatrix} \gamma_r \ \mathbf{E}_{\text{R}}(i,j) \\ \gamma_v \ \mathbf{E}_{\text{v}}(i,j) \\ \gamma_p \ \mathbf{E}_{\text{p}}(i,j) \end{bmatrix} \tag{7}$$

The computationally most expensive part of (1) is computing and deriving the preintegration defined in equation (4). To this end, we recompute Eq. (4) only if the change in bias $(\Delta\mathbf{b}_a, \Delta\mathbf{b}_g)$ is above a threshold. Otherwise we use the first order Jacobians to approximate the predicted state:

$$\begin{aligned}
\mathbf{R}_j^{est} &= \mathbf{R}_i\Delta\mathbf{R}_{ij}\exp(\mathbf{J}_{\Delta R}^g\Delta\mathbf{b}_g) \\
\mathbf{v}_j^{est} &= \mathbf{v}_i + \mathbf{g}\Delta t + \mathbf{R}_i(\Delta\mathbf{v}_{ij} + \mathbf{J}_{\Delta\mathbf{v}}^a\Delta\mathbf{b}_a + \mathbf{J}_{\Delta\mathbf{v}}^g\Delta\mathbf{b}_g) \\
\mathbf{p}_j^{est} &= \mathbf{p}_i + \mathbf{v}_i\Delta t + \frac{1}{2}\mathbf{g}\Delta t^2 + \mathbf{R}_i(\Delta\mathbf{p}_{ij} + \mathbf{J}_{\Delta p}^a\Delta\mathbf{b}_a + \mathbf{J}_{\Delta p}^g\Delta\mathbf{b}_g)
\end{aligned} \tag{8}$$

The second nonlinear subproblem (3) consists of the visual error $\mathbf{E}_{\text{VIS}}$ and the relative pose error $\mathbf{E}_{\text{REL}}$. The visual error is defined as the distance between the reprojected 3D point and the measured 2D keypoint

$$\mathbf{E}_{\text{VIS}}(i,k) = \pi(\mathbf{P}_i \mathbf{m}) - \mathbf{x}_k \tag{9}$$

where $\mathbf{x}_k$ is the keypoint position, $\mathbf{m}$ the global 3D point, $\mathbf{P}$ the camera pose, and π the camera model mapping the view space points to image coordinates. For the camera model, we use the pinhole projection with an 8-parameter RadTan distortion model [40]. The relative pose error using equation (7) and (8) is

$$\begin{aligned} \mathbf{T}_{ij} &= \mathbf{T}_{BC}\mathbf{P}_i^{-1}[\mathbf{R}_j^{est}, \mathbf{p}_j^{est}]\mathbf{T}_{CB} \\ \mathbf{E}_{\text{REL}}(i,j) &= \log(\mathbf{T}_{ij}\mathbf{P}_j\mathbf{P}_i^{-1}) \end{aligned} \tag{10}$$

where $\mathbf{T}_{BC}$ is the transformation from camera space to IMU space. Note here, that $\mathbf{E}_{\text{REL}}$ is identical to the cost used in pose graph optimization (PGO) during loop closure. Therefore, equation (3) can also be seen as a combined BA-PGO with the relative pose constraints coming from the IMU.

B. Robust Initialization

A good initial state estimation of the IMU parameters is essential for accurate camera tracking and precise scale recovery. The method described in [31] demonstrates good results for global VI systems, though they delay the initialization 15 seconds into tracking to make sure all variables are observable. We found that this delay is not required because gyroscope bias can be computed after a few keyframes. The initialization of scale and gravity direction take more time but early estimates might be beneficial to some applications.

In Snake-SLAM, the tracking begins in visual-only mode and the gyroscope bias initialization is started once 8 keyframes have been inserted into the map. To compute the global bias, we minimize the rotational error $\mathbf{E}_{\text{R}}$ of equation (7). This initialization procedure, which is similar to [31], works well unless the visual-only initialization is erroneous. To recover from bad monocular initializations, we apply the Cauchy robust cost function ρ with an adaptive threshold h_ρ.

$$\mathbf{b}_g = \underset{\mathbf{b}_g}{\operatorname{argmin}} \sum_{j=1} \rho(\log(\mathbf{R}_j^{est}\mathbf{R}_j^{-1}), h_\rho) \tag{11}$$

After each new keyframe and iteration, the threshold h_ρ is updated to the clamped and scaled RMSE of Eq. (11).

$$\begin{aligned} \gamma &= \sqrt{\frac{1}{N}\sum_{j=1} ||\log(\mathbf{R}_j^{est}\mathbf{R}_j^{-1})||^2} \\ h_\rho^{new} &= \max(\min(h_\rho, \frac{3}{4}\gamma), h_g) \end{aligned} \tag{12}$$

The updated threshold is then used to erase outlier keyframes from the map. Starting after 3 iterations of Eq. (11) the currently first keyframe is removed, if its rotational error is above $1.5h_\rho$. Once the RMSE is below $h_g = 0.008$ or a maximum number of iterations is reached, the gyroscope bias initialization is finished. From here on, the measured angular velocities are used in pose prediction, pose refinement, and the constraint bundle adjustment (3). A few iterations of global BA ensure that the map complies with the new constraints.

Accelerometer initialization starts once the rotational pose constraints have been integrated into the map. Each time a new keyframe is inserted, we refine the bias $\mathbf{b}_a$, gravity $\mathbf{g}$, scale s and velocity $\mathbf{v}$. These parameters are initialized using the linear system defined in [31] and then iteratively refined with (1). Once the error is below a threshold h_a or a maximum number of iterations is reached, IMU initialization is completed. After that, local mapping continues normally by an alternating minimization of (1) and (3). To counteract the ambiguity of IMU states, we linearly increase the weight of the predicted relative translation in $\mathbf{E}_{\text{REL}}$ until $t = 25\,\text{s}$. Additionally, the scale is held constant and only refined at fix timestamps $t = 15\,\text{s}$, $t = 30\,\text{s}$, and $t = 45\,\text{s}$.

C. Camera Pose Estimation

The pose of new frames is computed similar to [27]. Map points found in the last frame and last keyframe are projected to a predicted pose and matched to 2D features. Using these matches, the camera pose is refined by minimizing a visual inertial cost function. After refinement, more 3D points are projected and the pose optimization is repeated.

Once IMU initialization is completed, we predict the camera pose using equation (4) and (5). Since the accelerometer initialization is often weak at the beginning, we interpolate the predicted position with the position of a constant velocity model. The interpolation weight is changed to gradually favor the IMU prediction in the same way as the translational weight is changed during constraint BA.

For pose refinement, we use a slightly modified version of the decoupled visual inertial cost function (3)

$$\underset{\mathbf{P}}{\operatorname{argmin}} \sum_{obs} ||\mathbf{E}_{\text{VIS}}(obs)||^2 + ||\log(\mathbf{P}_{IMU}^{est}\mathbf{P}^{-1})||^2 \tag{13}$$

where only the camera pose $\mathbf{P}$ is optimized. The relative pose constraint is replaced by an absolute pose error between the current and predicted pose. The interpolated pose is not used in (13) because the motion model would bias the result.

V. GRAPH-BASED MAP SIMPLIFICATION

Obtaining a stable camera tracking during rapid motion of flying drones or head-mounted displays requires many keyframes in the local history. Muetry et al. [27], [31] made the observation that inserting as many keyframes as possible and then culling them based on a redundancy heuristic yields stable and accurate pose estimations. The problem of their heuristic is that pose graph connectivity is not guaranteed.

To that end, we add an additional graph-connectivity constraint, which ensure that the map never breaks apart. First, we define the global map quality C_{global} as the minimal edge weight in the maximal spanning tree of the covisibility graph G_{vis}.

$$C_{global} = \min_{edge} \text{ SPAN}(G_{vis}) \tag{14}$$

The covisibility graph G_{vis} contains all keyframes as vertices and an edge between two keyframes with a weight equal to the number of pairwise feature matches. A new keyframe K_i is essential if the map quality would fall below a threshold th_{map}

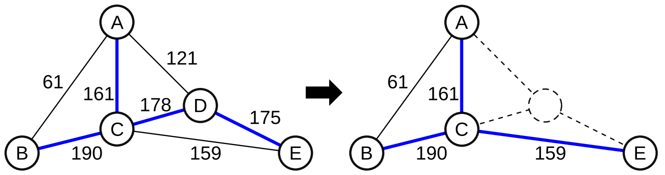

Fig. 3: Graph-based map simplification. Blue edges show the maximal spanning tree. The keyframe D is removed, because the minimal edge weight in the maximal spanning tree stays above the threshold $th_{map} = 140$.

without it. Therefore, we cull K_i if the following conditions holds:

$$\min_{edge} \operatorname{SPAN}(G_{vis} \setminus K_i) > th_{map} \tag{15}$$

To reduce complexity, we approximate Eq. (15) by only using a local sub-graph in the neighborhood of K_i. As a culling threshold we use $th_{map} = 140$. A special case occurs if a node has only a single connection in the local spanning tree. For these boundary keyframes, we use a different culling criterion to keep good frames with lots of innovation. A border keyframe K_i is culled if the triangulation angle to its spanning tree neighbor K_j is small, the edge weight $e_{ij} < 80$, or the keyframe contains mostly redundant observations.

The algorithm is visualized in Figure 3 for the inner keyframe D. First, the local covisibility graph around D is extracted and the maximal spanning tree is build (thick blue edges). After that, the vertex D is removed and the spanning tree is rebuild. The minimal spanning tree edge weight after removal is 159, which is larger than th_{map}. Therefore, D is not essential and can be removed. In the case of visual inertial tracking, we also ensure that the time between two keyframes is less than 0.5 seconds during initialization phase and less than 1.1 s after that.

VI. DEFERRED MAPPER

When a new keyframe is inserted into the map, new keypoints are created by computing feature matches between the novel and previous keyframes. This approach is problematic because right after insertion the relative pose error of a keyframe is maximal. Therefore, novel points have a high probability of being outliers and correct matches might not be found. On the EuRoC dataset V11 [41], we have measured that the tracking error decreases over time by around 10%. On the more difficult V12 dataset the error decreases by around 25%.

Therefore, we propose to add a *Deferred Mapper* module into keyframe-based SLAM, which triangulates and optimizes matches a second time, once the keyframe pose has stabilized. A similar idea under the name *Retriangulation* has been successfully used in offline structure-from-motion systems to improve keypoint density in old parts of the map [42]. The deferred local mapper in Snake-SLAM is delayed 10 keyframes, which means that when keyframe K_i is inserted the keyframe K_{i-10} is processed. This keyframe refinement is similar to the local mapping and consist of the following steps:

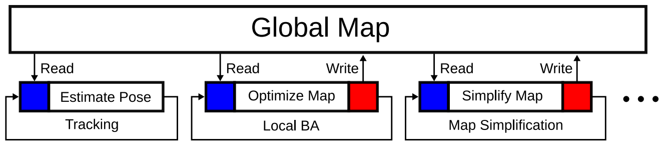

Fig. 4: Each module operates on a local copy of the global map. The red and blue parts indicate critical sections.

1) **Relinking.** All observed points of the keyframes are projected into the image and matched to nearby keypoint. If a 2D point with smaller reprojection error and feature distance is found, the observation is updated to link to the new keypoint.
2) **Outlier Removal.** Observations with a reprojection error over th_r are removed. This step is performed before (3.) and (4.) to allow the keypoints to immediately find new matching neighbors.
3) **Map Search.** The local map around the keyframe is extracted and projected into the image. New observations are created and old points are fused if a good feature match is found. All points in the local map must have at least one link to a keyframe, which has already been processed by the deferred mapper.
4) **Retriangulate.** Similar to the initial triangulation during keyframe insertion new epipolar matches are found and triangulated. Retriangulation is also only applied if both keyframes are older than the deferred mapper delay of 10.

VII. IMPLEMENTATION

Running global SLAM on micro aerial devices requires high efficiency of all steps in the tracking pipeline. This section covers our implementation approach and gives further insights into the algorithmic choices of Snake-SLAM. The first part (Section VII-A), describes the data structures used for each module as well as the required intra-module synchronization. After that, the various numerical solvers are highlighted (Section VII-B). Lastly, our parallel implementation approach is presented in Section VII-C.

A. Local Maps

Each module of the SLAM pipeline accesses the global map at some point. For example, the tracking stage tries to find matches between the newest frame's keypoints and the triangulated 3D points. A naive implementation is problematic due to race conditions and the interesting 3D points might be scattered over system memory. Therefore, each module in Snake-SLAM has a local working copy of the map containing only the relevant information. The working copy of the tracking module contains a few hundred 3D points with their associated feature descriptors. This increases memory coherence and reduces synchronizations points to one at the beginning of a module and one at the end. Figure 4 shows the workflow and critical sections of different SLAM modules. The tracking module reads from the global map once before it estimates the new frame camera pose. Local BA reads parts of the map and then optimizes its own working copy before copying the result back to the global map. Read operations on the map are much more common than writes because modules that also write updates are only executed once per keyframe. Therefore, to reduce contention of the critical section, we use the asymmetric locking mechanism of shared mutexes. Multiple threads are allowed to enter the critical section simultaneously as long as none of them writes to the global map. If a write operation is issued, the map is completely locked for all other threads until it is completed.

B. *Numerical Solver*

In multiple stages of the SLAM pipeline, solving nonlinear optimization problems takes the majority of time. During tracking the new frame's pose is aligned to the global map using Eq. (13). After keyframe insertion, the IMU state is refined (1) and the map is optimized by constraint bundle adjustment (3). During loop closing, accumulated drift is removed by pose graph optimization. An efficient solution to these problems requires precise derivatives and a linear solver that exploits the underlying structure.

We compute the derivatives with respect to geometric transformations by expressing linearized pose increments as Lie-algebra elements $x \in \mathfrak{se}(3)$. This removes all trigonometric function evaluations while still providing an accurate derivative around x_0.

To solve the structured linear systems of equations, we make use of recursive matrix algebra [43]. The Hessian structure of a given problem is encoded into a single C++ type by using recursive templates. For example, the recursive type for our constraint bundle adjustment with 6 camera parameters and 3 point parameters is:

```
using BAMatrix = MixedMatrix22<
  SparseMatrix<Matrix<double, 6, 6>>,
  SparseMatrix<Matrix<double, 6, 3>>,
  SparseMatrix<Matrix<double, 3, 6>>,
  DiagonalMatrix<Matrix<double, 3, 3>>>;
```

This approach eliminates all unnecessary overhead because the problem structure is analyzed at compile time and the most optimal solver is picked. In the given example, the block diagonal matrix in the bottom right is exploited by constructing the Schur-complement. Additional performance gains are acquired on modern compilers due to the statically sized inner blocks resulting in unrolled and vectorized code. Previous work [43] has shown that the recursive formulation outperforms more general optimization frameworks, such as Ceres [44] and G2O [45], by a factor of 2.

C. *Parallelization*

Examining current trends of hardware development shows that parallelization plays an increasingly important role. Modern software systems must take advantage of multiple threads to reach the best performance. The architecture of Snake-SLAM is inherently parallel because each stage in the tracking pipeline runs independently in a separate thread. The optimization back-end is parallelized by a simultaneous execution of the two subproblems (1) and (3). Further tests have shown that feature detection, camera tracking, and local mapping benefit from multiple threads. For example, during the triangulation of new 3D points, feature matches have to be searched across multiple keyframes in the past. This is parallelized by assigning each keyframe pair to a single thread. In Snake-SLAM, the in-module parallelization is implemented with OpenMP [46].

In total, Snake-SLAM makes use of 12 threads, which currently exceeds the core count of most consumer processors. However, we show in Section VIII-B that Snake-SLAM runs very well on mobile quad core system. In the near future, when 8 or 16 core CPUs become more common, we expect our approach to show even better results.

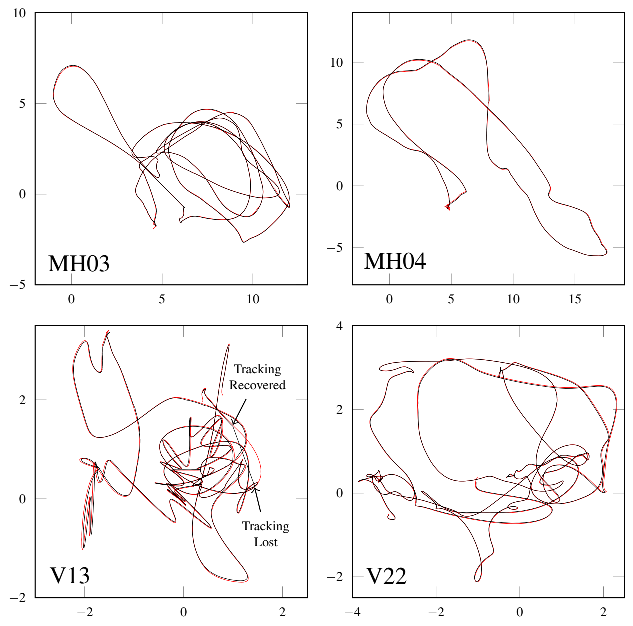

Fig. 5: The estimated trajectory of monocular Snake-SLAM (black) and ground truth (red) for various EuRoC sequences.

VIII. Evaluation

To validate Snake-SLAM, we have conducted multiple experiments. In the first part (Section VIII-A), we evaluate the tracking accuracy and compared it to other SLAM systems. In Section VIII-B, a detailed performance analysis is given. We measure the timings of each step as well as the required system power consumption during real-time operation (see Section VIII-C). Finally, we show in Section VIII-D that Snake-SLAM is well suited for dense reconstruction tasks.

A. *Tracking Accuracy*

The tracking accuracy of Snake-SLAM is evaluated on the public EuRoC dataset in monocular and stereo mode. The computed trajectory is aligned to the ground truth by a $\mathbf{SE}(3)$ transformation using the method of [48]. Following the standard evaluation approach of SLAM systems [49], we record 25 runs and select the trajectory with the median absolute root mean square error (RMSE). For the other SLAM system, we use the results presented in the respective publications.

Monocular + IMU		MH01	MH02	MH03	MH04	MH05	V11	V12	V13	V21	V22	V23
Snake-SLAM (ours)	RMSE (m)	**0.016**	**0.021**	**0.023**	**0.091**	**0.028**	**0.018**	0.019	**0.023**	**0.021**	0.016	**0.021**
	Scale Error (%)	0.9	0.2	0.4	0.7	0.1	0.8	0.9	1.1	0.6	0.6	0.5
ORB-SLAM 3 [32]	RMSE (m)	0.032	0.053	0.033	0.099	0.071	0.043	**0.016**	0.025	0.041	**0.015**	0.037
	Scale Error (%)	0.7	1.0	0.3	1.0	0.6	1.5	0.5	1.1	0.5	0.3	0.9
VI ORB-SLAM [31]	RMSE (m)	0.075	0.084	0.087	0.217	0.082	0.027	0.028	×	0.032	0.041	0.074
	Scale Error (%)	0.5	0.8	1.5	3.4	0.5	0.9	0.8	×	0.2	1.4	0.7
VI DSO [11]	RMSE (m)	0.062	0.044	0.117	0.132	0.121	0.059	0.067	0.096	0.040	0.062	0.174
	Scale Error (%)	1.1	0.5	0.4	0.2	0.8	1.1	1.1	0.8	1.2	0.3	0.4
VINS-Mono [17]	RMSE (m)	0.084	0.105	0.074	0.122	0.147	0.047	0.066	0.180	0.056	0.090	0.244
SelfVIO [34]	RMSE (m)	0.19	0.15	0.21	0.16	0.29	0.08	0.09	0.10	0.11	0.08	0.11

Stereo + IMU		MH01	MH02	MH03	MH04	MH05	V11	V12	V13	V21	V22	V23
Snake-SLAM (ours)	RMSE (m)	**0.013**	**0.021**	**0.018**	0.091	**0.018**	**0.012**	**0.012**	**0.015**	**0.018**	**0.012**	**0.014**
	Scale Error (%)	0.1	0.5	0.5	0.2	0.7	0.4	0.3	0.4	0.4	0.2	0.2
ORB-SLAM 3 [32]	RMSE (m)	0.037	0.031	0.026	**0.059**	0.086	0.037	0.014	0.023	0.037	0.014	0.029
	Scale Error (%)	0.7	0.2	0.2	0.4	1.0	0.6	0.6	0.6	1.4	0.2	0.8
VINS Fusion [21]	RMSE (m)	0.166	0.152	0.125	0.280	0.284	0.076	0.069	0.114	0.066	0.091	0.096
Basalt [47]	RMSE (m)	0.080	0.060	0.050	0.100	0.080	0.040	0.020	0.030	0.030	0.020	×
OKVIS [15]	RMSE (m)	0.160	0.220	0.240	0.340	0.470	0.090	0.200	0.240	0.130	0.160	0.290

TABLE I: Absolute trajectory RMSE on the EuRoC dataset for monocular VI tracking (top) and stereo VI tracking (bottom). *RMSE* is the absolute pose error after rigid trajectory alignment.

The EuRoC micro aerial vehicle dataset [41] consists of 11 sequences recorded on a flying drone. It includes 20 Hz monochrome stereo images, IMU measurements, and ground truth poses. Six sequences (prefixed with V) are captured indoors in a single room. The other five (prefixed with MH) are captured in a machine hall. The ground truth of the machine hall sequences was quantified by a Leica MS50 laser tracker. As the frequency of this sensor is low and the raw measurements are noisy, we use the provided smoothed and interpolated ground truth estimates. For the indoor sequences, we use the raw ground truth data of the Vicon 6D motion capture system. As mentioned by the dataset authors, the time synchronization between camera and ground truth is inaccurate [41]. They suggest a temporal aligned of the camera trajectory, which we apply before measuring the pose error. Our computed temporal offsets in seconds for the *MH* sequences are $(0.006, 0.011, 0.012, 0.029, 0.032)$ for the *V1* sequences are $(0.010, 0.013, 0.042)$ and for the *V2* sequences are $(-0.217, -0.209, -0.200)$. These offsets match closely to the numbers computed in [31] and are included in the trajectory comparison. To simulate autonomous drone localization, we ran Snake-SLAM on a Jetson Nano mini-computer, which has acceptable weight and power restrictions to be placed on micro aerial devices.

Table I shows the absolute trajectory RMSE without scale correction (SE3-alignment). Snake-SLAM consistently outperforms the competitors in both monocular and stereo mode. On the machine hall sequences, the average RMSE of monocular Snake-SLAM is 0.036 m and on the indoor sequences 0.020 m. ORB-SLAM3, which uses a similar tracking front-end as our approach, achieves an average RMSE of 0.058 m and 0.030 m respectively. VI-DSO and the neural network-based SelfVIO show significantly larger absolute trajectory errors than Snake-SLAM. However, this is expected because odometry methods do not build a global map and are therefore less accurate in sequences with many loops. The scale error averages between 0.1% and 1.1%, which is similar to the scale error of the competitors.

Figure 5 contains monocular trajectory plots (black) of all tracked frames on four EuRoC sequences. Underneath, we include the ground truth poses in red. The plot starts, once monocular initialization has completed and does not include the last few frames because our system automatically removes keyframes which have not been refined by the deferred mapper (see Section VI). In all sequences, the estimated trajectory aligns precisely to the ground truth without visible scale error or pose drift. Near the end of the difficult V13 sequence (bottom left), tracking is lost during an outward rotation of the drone. After a few frames, the pose is recovered correctly before the sequence ends. We found a similar behavior in the V23 scene, where tracking is lost and recovered multiple times due to rapid rotations and strong motion blur. Even though only around 70% of all frames are tracked in this sequence, we can see in Table I that Snake-SLAM achieves a very low pose error and recovers the absolute scale precisely. This shows, that our pose recovery works well and the optimization back-end is able to produce a high-quality map from multiple partial sequences.

B. Timings

In this section, we take a detailed look at each pipeline step and evaluate the performance on a Jetson Nano single board computer, a thin and light laptop, and a high-end

System	Jetson Nano	Laptop	Desktop
CPU	ARM A57	R5 4650U	i9-7940X
Cores	4 x 1.43 GHz	6 x 2.1 GHz	14 x 3.1 GHz
RAM	4 GB	16 GB	64 GB
GPU	Maxwell	-	RTX 2080 Ti
CUDA Cores	128	-	2560
VRAM	4 GB (Shared)	-	8 GB
CPU Util.	220 %	59 %	13 %
GPU Util.	28.17 %	-	2.6 %
RAM Usage	24.8 %	8.3 %	2.1 %
Power	4.25 W	15 W	95 W
Uncapped FPS	50	130	322

TABLE II: Performance overview of Snake-SLAM.

	Snake-SLAM (ours)			VI ORB-SLAM	
	Nano	Laptop	Desktop	Laptop	Desktop
Feature Detection	11.42	6.41	0.41	13.67	11.95
Tracking	7.27	1.97	1.24	9.45	8.49
Local Mapping	18.83	5.97	3.65	74.81	60.01
Constraint LBA	76.21	15.23	12.49	107.63	95.44
IMU Solving	34.07	10.97	7.55	0.00	0.00
Map Simplification	1.75	1.16	0.71	9.62	7.71
Deferred Mapping	17.12	5.10	2.60	0.00	0.00
Loop Detector	1.52	0.62	0.32	6.70	5.97

TABLE III: Mean time in ms for the different SLAM stages on the EuRoC MH03 dataset in monocular mode.

desktop. All timings are measured on the EuRoC MH03 sequence with Snake-SLAM running in monocular mode and 1000 features per frame. This scene consists of around 2400 frames recorded at 20 FPS. As shown in Figure 5 (top left), the MH03 sequence contains multiple loops therefore generating a very dense pose graph. Feature detection is run on the GPU for the Jetson Nano and Desktop. The Laptop does not have a CUDA-capable graphics card, therefore we execute feature detection on the CPU. All other steps are run on the CPU. The program is compiled by Clang 9.0 using all optimizations and architecture specific instructions including ARM Neon for the Nano and AVX512 for the desktop.

A short performance overview, including more detailed system specifications, is given in Table II. As all three platforms can process the input stream above the target frame rate of 20, we have added an uncapped FPS mode, which processes new frames as fast as possible. The uncapped mean frame rate of Snake-SLAM on the Jetson Nano, Laptop, and Desktop are 50, 130, and 322 respectively. Limiting the frame rate to 20, reduces the CPU utilization of the desktop to 13% of a single core. The Laptop requires 59% CPU utilization due to the CPU ORB feature detection. In comparison, ORB-SLAM2 has between 300% and 400% CPU utilization on similar systems.

Table III shows the measured mean times of each step in the SLAM-pipeline. On the Jetson Nano, the optimization back-end takes the majority of time with 76 ms for constraint LBA and 34 ms for the IMU solver. However, these steps are only executed once per keyframe, which are inserted in average every 500 ms. The feature detection requires 11.4 ms. The laptop has build-in a more powerful CPU, significantly

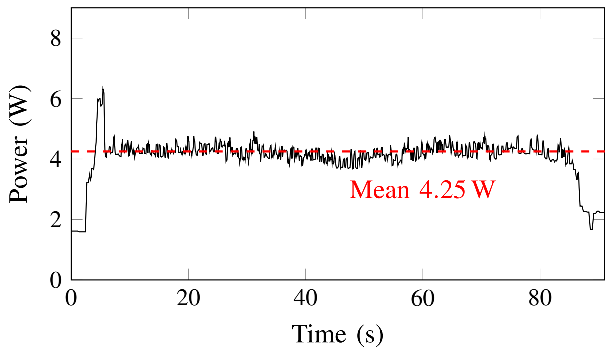

Fig. 6: Total system power consumption of Snake-SLAM on the Jetson Nano for the MH04 sequence in monocular mode.

reducing processing time by 40-70% compared to the Jetson Nano. On the desktop system, constraint LBA takes 12.5 ms, IMU solving takes 7.5 ms and feature detection requires only a fraction of a millisecond due to the powerful GPU. The laptop and desktop can process all steps faster than the delta time between two consecutive frames ($\Delta t_f = 50$ms), even though some of them are executed only once per keyframe ($\Delta t_{kf} \approx 500$ms).

The performance comparison towards VI ORB-SLAM shows large differences across the field even though we have tuned their implementation to produce an roughly identical amount of keyframes. The stages feature detection, tracking, local mapping, and LBA of VI ORB-SLAM on the desktop system show comparable performance characteristics than Snake-SLAM running on the Jetson Nano. This is remarkable because the Jetson Nano is a single board computer with only a fraction of the power consumption. However, we have to acknowledge that the comparison of the feature detection is not fair since our implementation runs on the GPU and could be easily integrated into VI ORB-SLAM.

C. Power Consumption

In the previous section, we have shown that Snake-SLAM is very efficient and can run on a Jetson Nano single board computer. When integrating such a system into battery-powered robots or drones, power consumption is a major concern. Ideally, the current drawn is constant and doesn't exceed a given threshold. We have tested Snake-SLAM in a controlled environment running on a Jetson Nano B01. The dataset is loaded on-the-fly from a wall-powered external USB 3 SSD. Idle power consumption is 1.6 W.

Figure 6 shows the measured power consumption on the Jetson Nano for the MH04 dataset in monocular mode. During the first few frames, the initialization phase is executed, which consumes the most power at around 6 W. The average current drawn during this phase is 1.16 A with a maximum measurement of 1.4 A. After the initialization, power consumption drops to an average of 4.25 W and 0.83 A. Note that at $t = 45$ s the capturing drone flies into a dark area of the machine hall. At this point, less features are extracted and power consumption drops by 0.5 W.

For a mobile integration, we recommend at least two 3.7 V

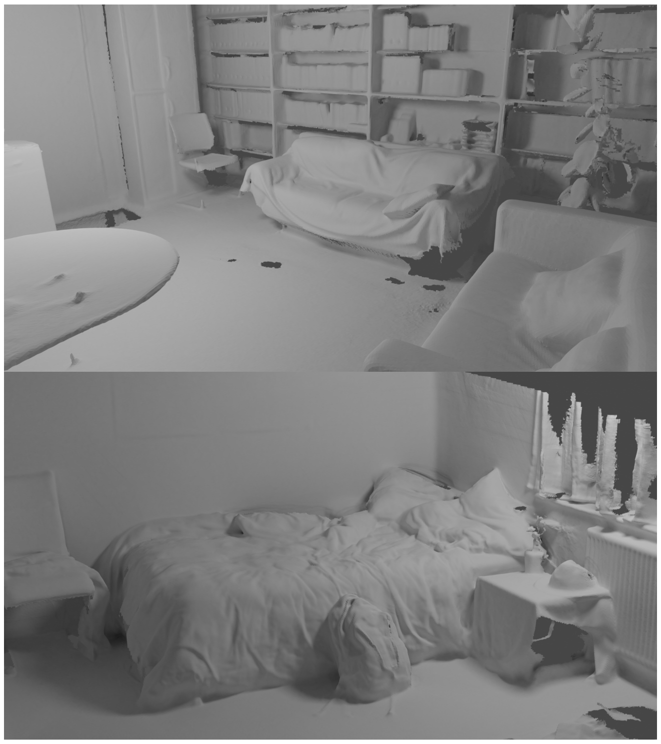
Fig. 7: Indoor scans of a living room (top) and bed room (bottom) with 3000 and 3500 frames respectively.

lithium ion cells in series with a switching 5 V DC regulator. Respective cells are usually rated for up to 10 A continuous discharge current, which is more than enough for the measured 1.4 A peak during monocular initialization. This setup weighs around 240 grams (100 g battery + 140 g Jetson Nano) and should last for 5 hours of nonstop monocular tracking.

D. Dense Reconstruction

Due to the global tracking paired with loop closure and relocalization, Snake-SLAM is well suited for dense reconstructions tasks. In our experiments, an operator first captures the scene resulting in a global map of keyframes and 3D points. After that, all intermediate frames are realigned using equation (3). The optimized frames are then fused into a truncated signed distance field (TSDF) [23] and the isosurface is extracted using the Marching-Cubes algorithm [50].

Figure 7 shows two room-scale indoor scans of a living room (top) and bed room (bottom). They were captured by an Azure Kinect camera [51] which outputs 30 Hz RGB-D images paired with 200 Hz IMU measurements. The scenes were reconstructed from 3000 and 3500 frames respectively using a voxel size of 0.5 cm. In both reconstructions, the wall and floor are almost perfectly planar and fine detail is visible on the bed, sofa, and shelf. These results are beneficial to intelligent robots or drones that navigate through indoor environments. Dense surfaces are identified as obstacles and a volumetric data structure could be used as input to neural classification networks.

IX. CONCLUSION AND OUTLOOK

Snake-SLAM is a VI-SLAM method capable of onboard navigation in micro aerial devices. New frame poses are estimated relative to a globally consistent map, allowing zero-drift localization in already discovered areas. The map is optimized using a decoupled nonlinear solver, which refines IMU states and map states separately. We show that this approach is highly efficient without a compromise to tracking accuracy. Snake-SLAM achieves best or second best results compared to state-of-the-art SLAM system on all sequences of the EuRoC dataset. The high efficiency and low power consumption makes Snake-SLAM viable on all kinds of mobile devices such as drones, robots, phones, or AR headsets. Using an RGB-D camera, we are able to densely reconstruct environments to further aid navigation in unknown terrain.

In future work we want to tackle the integration of additional sensors that are commonly found on UAVs. Especially the use of GPS should further increase robustness and accuracy on long-lasting exploration missions. A nice property of our optimization-based tracking back-end is that GPS and other sensors can be implemented straight forward by adding additional constraints to Eq. (1). The decoupled optimization approach ensures that the complexity of BA is not increased and real-time operation can still be assumed.

The source code of Snake-SLAM and all computed trajectories used in Table I are available on GitHub.

`https://github.com/darglein/Snake-SLAM`

References

[1] J. Delmerico and D. Scaramuzza, “A benchmark comparison of monocular visual-inertial odometry algorithms for flying robots,” in *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2018, pp. 2502–2509.

[2] E. S. Jones and S. Soatto, “Visual-inertial navigation, mapping and localization: A scalable real-time causal approach,” *The International Journal of Robotics Research*, vol. 30, no. 4, pp. 407–430, 2011.

[3] M. Bloesch, S. Omari, M. Hutter, and R. Siegwart, “Robust visual inertial odometry using a direct ekf-based approach,” in *2015 IEEE/RSJ international conference on intelligent robots and systems (IROS)*. IEEE, 2015, pp. 298–304.

[4] D. Scaramuzza and Z. Zhang, “Visual-inertial odometry of aerial robots,” *arXiv preprint arXiv:1906.03289*, 2019.

[5] A. I. Mourikis and S. I. Roumeliotis, “A multi-state constraint kalman filter for vision-aided inertial navigation,” in *Proceedings 2007 IEEE International Conference on Robotics and Automation*. IEEE, 2007, pp. 3565–3572.

[6] S. Lynen, T. Sattler, M. Bosse, J. A. Hesch, M. Pollefeys, and R. Siegwart, “Get out of my lab: Large-scale, real-time visual-inertial localization.” in *Robotics: Science and Systems*, vol. 1, 2015.

[7] K. Sun, K. Mohta, B. Pfrommer, M. Watterson, S. Liu, Y. Mulgaonkar, C. J. Taylor, and V. Kumar, “Robust stereo visual inertial odometry for fast autonomous flight,” *IEEE Robotics and Automation Letters*, vol. 3, no. 2, pp. 965–972, 2018.

[8] J. Engel, T. Schöps, and D. Cremers, “Lsd-slam: Large-scale direct monocular slam,” in *European conference on computer vision*. Springer, 2014, pp. 834–849.

[9] C. Forster, M. Pizzoli, and D. Scaramuzza, “Svo: Fast semi-direct monocular visual odometry,” in *2014 IEEE international conference on robotics and automation (ICRA)*. IEEE, 2014, pp. 15–22.

[10] J. Engel, V. Koltun, and D. Cremers, “Direct sparse odometry,” *IEEE transactions on pattern analysis and machine intelligence*, vol. 40, no. 3, pp. 611–625, 2017.

[11] L. Von Stumberg, V. Usenko, and D. Cremers, "Direct sparse visual-inertial odometry using dynamic marginalization," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2018, pp. 2510–2517.

[12] X. Gao, R. Wang, N. Demmel, and D. Cremers, "Ldso: Direct sparse odometry with loop closure," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2018, pp. 2198–2204.

[13] R. Wang, M. Schworer, and D. Cremers, "Stereo dso: Large-scale direct sparse visual odometry with stereo cameras," in *Proceedings of the IEEE International Conference on Computer Vision*, 2017, pp. 3903–3911.

[14] D. Schubert, N. Demmel, V. Usenko, J. Stuckler, and D. Cremers, "Direct sparse odometry with rolling shutter," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 682–697.

[15] S. Leutenegger, S. Lynen, M. Bosse, R. Siegwart, and P. Furgale, "Keyframe-based visual–inertial odometry using nonlinear optimization," *The International Journal of Robotics Research*, vol. 34, no. 3, pp. 314–334, 2015.

[16] C. G. Harris, M. Stephens, *et al.*, "A combined corner and edge detector." in *Alvey vision conference*, vol. 15, no. 50. Citeseer, 1988, pp. 10–5244.

[17] T. Qin, P. Li, and S. Shen, "Vins-mono: A robust and versatile monocular visual-inertial state estimator," *IEEE Transactions on Robotics*, vol. 34, no. 4, pp. 1004–1020, 2018.

[18] B. D. Lucas, T. Kanade, *et al.*, "An iterative image registration technique with an application to stereo vision," 1981.

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

[20] M. Calonder, V. Lepetit, C. Strecha, and P. Fua, "Brief: Binary robust independent elementary features," in *European conference on computer vision*. Springer, 2010, pp. 778–792.

[21] T. Qin, S. Cao, J. Pan, and S. Shen, "A general optimization-based framework for global pose estimation with multiple sensors," *arXiv preprint arXiv:1901.03642*, 2019.

[22] 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 surface mapping and tracking," in *2011 10th IEEE International Symposium on Mixed and Augmented Reality*. IEEE, 2011, pp. 127–136.

[23] M. Nießner, M. Zollhöfer, S. Izadi, and M. Stamminger, "Real-time 3d reconstruction at scale using voxel hashing," *ACM Transactions on Graphics (ToG)*, vol. 32, no. 6, pp. 1–11, 2013.

[24] A. Dai, M. Nießner, M. Zollhöfer, S. Izadi, and C. Theobalt, "Bundlefusion: Real-time globally consistent 3d reconstruction using on-the-fly surface reintegration," *ACM Transactions on Graphics (ToG)*, vol. 36, no. 4, p. 1, 2017.

[25] Z. Zhang, "Iterative point matching for registration of free-form curves and surfaces," *International journal of computer vision*, vol. 13, no. 2, pp. 119–152, 1994.

[26] G. Klein and D. Murray, "Parallel tracking and mapping on a camera phone," in *2009 8th IEEE International Symposium on Mixed and Augmented Reality*. IEEE, 2009, pp. 83–86.

[27] 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, 2017.

[28] E. Rosten and T. Drummond, "Machine learning for high-speed corner detection," in *European conference on computer vision*. Springer, 2006, pp. 430–443.

[29] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski, "Orb: An efficient alternative to sift or surf," in *2011 International conference on computer vision*. Ieee, 2011, pp. 2564–2571.

[30] L. Jinyu, Y. Bangbang, C. Danpeng, W. Nan, Z. Guofeng, and B. Hujun, "Survey and evaluation of monocular visual-inertial slam algorithms for augmented reality," *Virtual Reality & Intelligent Hardware*, vol. 1, no. 4, pp. 386–410, 2019.

[31] R. Mur-Artal and J. D. Tardós, "Visual-inertial monocular slam with map reuse," *IEEE Robotics and Automation Letters*, vol. 2, no. 2, pp. 796–803, 2017.

[32] C. Campos, R. Elvira, J. J. Gómez, J. M. M. Montiel, and J. D. Tardós, "ORB-SLAM3: An accurate open-source library for visual, visual-inertial and multi-map SLAM," *arXiv preprint arXiv:2007.11898*, 2020.

[33] R. Clark, S. Wang, H. Wen, A. Markham, and N. Trigoni, "Vinet: Visual-inertial odometry as a sequence-to-sequence learning problem," *arXiv preprint arXiv:1701.08376*, 2017.

[34] Y. Almalioglu, M. Turan, A. E. Sari, M. R. U. Saputra, P. P. de Gusmão, A. Markham, and N. Trigoni, "Selfvio: Self-supervised deep monocular visual-inertial odometry and depth estimation," *arXiv preprint arXiv:1911.09968*, 2019.

[35] R. Li, S. Wang, Z. Long, and D. Gu, "Undeepvo: Monocular visual odometry through unsupervised deep learning," in *2018 IEEE international conference on robotics and automation (ICRA)*. IEEE, 2018, pp. 7286–7291.

[36] Y. Almalioglu, M. R. U. Saputra, P. P. de Gusmao, A. Markham, and N. Trigoni, "Ganvo: Unsupervised deep monocular visual odometry and depth estimation with generative adversarial networks," in *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, 2019, pp. 5474–5480.

[37] E. J. Shamwell, S. Leung, and W. D. Nothwang, "Vision-aided absolute trajectory estimation using an unsupervised deep network with online error correction," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2018, pp. 2524–2531.

[38] G. Grisetti, R. Kummerle, C. Stachniss, and W. Burgard, "A tutorial on graph-based slam," *IEEE Intelligent Transportation Systems Magazine*, vol. 2, no. 4, pp. 31–43, 2010.

[39] C. Forster, L. Carlone, F. Dellaert, and D. Scaramuzza, "On-manifold preintegration for real-time visual–inertial odometry," *IEEE Transactions on Robotics*, vol. 33, no. 1, pp. 1–21, 2016.

[40] G. BRADSKI, "The opencv library," *Dr Dobb's J. Software Tools*, vol. 25, pp. 120–125, 2000. [Online]. Available: https://ci.nii.ac.jp/naid/10028167478/en/

[41] M. Burri, J. Nikolic, P. Gohl, T. Schneider, J. Rehder, S. Omari, M. W. Achtelik, and R. Siegwart, "The euroc micro aerial vehicle datasets," *The International Journal of Robotics Research*, vol. 35, no. 10, pp. 1157–1163, 2016.

[42] J. L. Schonberger and J.-M. Frahm, "Structure-from-motion revisited," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 4104–4113.

[43] D. Rückert and M. Stamminger, "An efficient solution to structured optimization problems using recursive matrices," in *Computer Graphics Forum*, vol. 38, no. 8. Wiley Online Library, 2019, pp. 33–39.

[44] S. Agarwal, K. Mierle, and Others, "Ceres solver," http://ceres-solver.org.

[45] 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*, 2011, pp. 3607–3613.

[46] L. Dagum and R. Menon, "Openmp: an industry standard api for shared-memory programming," *IEEE computational science and engineering*, vol. 5, no. 1, pp. 46–55, 1998.

[47] V. Usenko, N. Demmel, D. Schubert, J. Stückler, and D. Cremers, "Visual-inertial mapping with non-linear factor recovery," *IEEE Robotics and Automation Letters (RA-L)*, vol. 5, 2020, accepted for presentation at IEEE International Conference on Robotics and Automation (ICRA) 2020, to appear, arXiv:1904.06504.

[48] S. Umeyama, "Least-squares estimation of transformation parameters between two point patterns," *IEEE Transactions on Pattern Analysis & Machine Intelligence*, no. 4, pp. 376–380, 1991.

[49] J. Sturm, N. Engelhard, F. Endres, W. Burgard, and D. Cremers, "A benchmark for the evaluation of rgb-d slam systems," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2012, pp. 573–580.

[50] W. E. Lorensen and H. E. Cline, "Marching cubes: A high resolution 3d surface construction algorithm," *ACM siggraph computer graphics*, vol. 21, no. 4, pp. 163–169, 1987.

[51] Microsoft, "Azure kinect dk documentation," https://docs.microsoft.com/en-us/azure/kinect-dk/, 2020.