2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)
Madrid, Spain, October 1-5, 2018

Key-Frame Strategy During Fast Image-Scale Changes and Zero Motion in VIO Without Persistent Features

Eren Allak[1], Alexander Hardt-Stremayr[1] and Stephan Weiss[1]

***Abstract*— Many of today's Visual-Inertial Odometry (VIO) frameworks work well under regular motion but have issues and need special treatment under special motion. Here, *special* does not imply bad or corrupted data but stands for increased difficulty to treat clean data. Common *special* motion for VIO are large feature displacement due to fast motion close to a scene and zero motion phases not providing sufficient baseline.**

In this paper we present a feature and frame selection approach which seamlessly handles all motion scenarios without the need of (error prone) motion case identification and subsequent case-specific heuristics. We further show that this approach allows to eliminate features in the state vector (persistent features) altogether while still being able to inherently handle zero motion phases. This reduces computational complexity while maintaining the ability to hover in place.

We integrate our frame selection approach into our own VIO algorithm and compare its performance against three state-of-the-art algorithms with real data on a real platform. While our approach shows slightly higher global drift it is the only algorithm that can reliably estimate the pose over a large motion spectrum from fast scale change down to zero motion.

I. INTRODUCTION

Robot pose estimation in an unknown environment is an important, well researched, yet still active field. Recently, so-called Visual-Inertial Odometry (VIO) frameworks fusing camera measurements and readings from an Inertial Measurement Unit (IMU) for robot pose estimation have reached a remarkable level of maturity. A key aspect of their success is the lower computational complexity compared to full grown Simultaneous Localization And Mapping (SLAM) approaches. This allows VIO algorithms run onboard computationally constrained platforms in real-time for accurate closed-loop control and navigation. The lower computational complexity, however, comes at the cost of global pose information: VIO algorithms would require a (costly) loop closure component to re-gain this information.

A naive implementation of a VIO algorithm can quickly drift in position and heading as the motion at a current time instant may not contain sufficient information for an estimate of the incremental step (e.g. pure rotation, still phases, large image motion). To mitigate both the issue of drift and loss of information while at the same time maintain the low computational complexity, different strategies have been proposed in the past. One main strategy is to only keep a map of the local environment and discard the information that is no longer used (this is also known as *keeping the features in the state* or so-called *persistent features*). The soft-terms "local" and "no longer used" immediately give rise to a number of parameters, thresholds, and heuristics one can define and choose. Varying these elements usually results in vast differences of performance although the same base algorithm is used.

In this paper, we suggest an approach which inherently chooses the best information available for motion estimation over a large span of different motion situations. Our contribution is a frame and feature selection for VIO that can inherently handle all motions from fast image scale changes to zero motion without requiring persistent features nor motion-case specific heuristics or state-machines.

Our approach uses a parallax-based metric for information weighting and only requires one single parameter (the minimum desired parallax between two keyframes), does not need to use persistent features (i.e. features in the state vector), and does not need to differentiate between still phases and (informative) motion through heuristics. This leads to several advantages: i) not requiring persistent features allows to maintain a very low computational complexity. We provide timings of our implementation proving that our approach is real-time capable even on computationally constrained platforms. ii) We demonstrate that our algorithm uses exact the same process whether large motion, slow motion or no motion is present. This results in a very lean framework capable of inherently handling visual-inertial data with fast and large scale change up to no motion at all. We show these capabilities in this paper using real indoor and outdoor data in our own tightly coupled feature based VIO for a UAV flight.

The rest of the paper is organized as follows. Section II describes the related work. Section III and IV presents our algorithm. Finally, Section V shows the experimental results and Section VI presents the conclusion.

II. RELATED WORK

Visual Inertial Odometry (VIO) algorithms are already widely used in robotics [1], [2], [3]. With the advent of powerful vision and inertial sensors, and high-performance System on Chip (SoC) processing units, a large body of work has been published on estimators fusing visual and inertial data for real-time state estimation. While earlier SoCs only allowed for the computationally less complex loosely coupled estimator structures [4] today's SoCs allow for more

[1]Eren Allak, Alexander Hardt-Stremayr and Stephan Weiss are with the Department of Smart Systems Technologies in the Control of Networked Systems Group, Alpen-Adria-Universität Klagenfurt, 9020 Klagenfurt, Austria {eren.allak, alexander.hardt-stremayr, stephan.weiss}@aau.at

The research leading to these results has received funding from the Austrian Ministry for Transport, Innovation and Technology (BMVIT) under grant agreement n. 855777 (MODULES) and from the ARL within the BAA W911NF-12-R-0011 under grant agreement W911NF-16-2-0112.

complex tightly coupled approaches [2]. With the difference that in loosely coupled approaches, the visual and inertial data is processed individually before fusion, whereas tightly coupled approaches the raw sensor information is directly fused together.

In addition to the distinction of loosely and tightly coupled we can classify VIO algorithms into non-linear optimization (also known as Bundle Adjustment (BA)) and filter-based approaches. BA based algorithms keep a measurement history and optimize over the whole history while filter-based algorithms only consider the latest measurement [5].

Visual-Inertial Bundle Adjustment algorithms are both available as feature-based approaches [6], [7] as well as direct intensity-based approaches [8]. Among the feature- and BA-based VIO approaches, both OKVIS [9] and VINS-Mono [10], [11] can be considered as currently best performing algorithms in the research community. For this reason, we will compare our algorithm against these frameworks in Section V. We will show that although our approach has a slightly higher drift, it can much better cope with a variety of different motions and requires much less computation power.

Filter-based algorithms can be distinguished between those having features in the state and those having keyframes in the state. ROVIO[12], keeps a bearing vector and a distance to each currently active feature in the state. It has been designed to operate on image patch intensities in [13].

In terms of computational complexity, it is considered as one of the most robust (hence the name ROVIO) VIO with very low computational complexity. For this reason, we include it together with OKVIS and VINS-Mono to the set of algorithms against which we compare our approach in Section V. We will show that our approach is slightly more costly than ROVIO, but can handle different motion scenarios much more robustly with less drift.

The multi-state constaint kalman filter (MSCKF)[1] by Mourikis et.al. established an approach to keep older poses (keyframes) in the state instead of detected features. This approach shares some advantages (being able to reevaluate old poses) and some disadvantages (a keyframe selection strategy is needed) with BA algorithms. MSCKF has been developed further in [2] by improving the consistency and changing the keyframe handling into a simple first in - first out selection strategy compared to the previous dropping of a third of keyframes when a certain amount was reached.

This approach is still an active field of research: Both [14] and [15] use stereo information to calculate feature depth instead of triangulating it based on older keyframes and feature measurements. [16] adapt it by using intensity changes at edges instead of more commonly used point features and [17] apply it on omnidirectional cameras. In [18], instead of the Extended Kalman Filter, the Iterated Cubature Kalman Filter is used and compared to an Unscented Kalman Filter variant.

A. Selected Frameworks for Comparison

Among the large variety of VIO algorithms, we have chosen three we consider have the best performance in one or more of the categories robustness, computational complexity, accuracy, and which are publicly available such that we can run direct comparisons with real data on the real platform (see Section V). The comparison to the MSCKF implementation of Ke Sun et al.[14] is not indicative, because of the advantage of stereo over monocular vision. In the monocular case the feature triangulation and the estimation problem need to be solved simultaneously, which is particularly difficult and requires a method for keyframe and feature selection as presented here. While in the stereo case the feature triangulation is decoupled from the estimation problem, since a second camera provides enough parllax for triangulation. In the following the three algorithms are described in some detail.

- OKVIS[9] provides a tightly-coupled visual-inertial feature based bundle adjustment approach. It keeps a set of most recent frames as well as another set of keyframes. Whenever the area spanned by matched points covers less than 50 - 60% of the area spanned by detected points, a new keyframe is generated. The oldest keyframe is marginalized.
- VINS-Mono[10], [11] is, similar to OKVIS, a tightly-coupled visual-inertial feature based bundle adjustment approach. It keeps a set of keyframes in a sliding window. If the average translation-induced parallax over all features compared to the latest keyframe is more than 20 rotation-compensated pixels, a new keyframe is inserted and the oldest keyframe is marginalized. Initialization is done by a separate initialization module which aligns an IMU based trajectory with a trajectory generated by Structure from Motion for images with enough parallax.
- ROVIO[12] is a feature based filter approach and does not use keyframes. Instead, features are selected to be kept in the state and updated with the latest measurement. New features are selected by identifying high-gradient candidates not close to existing features. The feature management depends on their global (how often has the feature been tracked) and local (how often has the feature been tracked when expected and how often was it in the field of view) quality.

III. ALGORITHM OVERVIEW

This section serves as an overview about the framework used for our our implementation of the suggested feature and frame selection approach. It mainly serves for completeness and does not claim major novelties.

A. Filter-Based Tightly Coupled Visual-Inertial Odometry

We use our own filter-based tightly coupled visual-inertial odometry framework which is based on [1]. In the following we very briefly sketch the set-up of such a framework.

1) State Vector: The state vector $\mathbf{X}$ consists of the current 6DoF pose of the IMU in the world frame, expressed as position p_w^i and attitude as a quaternion q_w^i with the 3D velocity vector v_w^i. In addition, the state vector contains the IMU intrinsic calibration states b_ω and b_a as the biases of

the gyroscopes and accelerometers respectively. Furthermore, the vector p_i^c and quaternion q_i^c describe the camera-IMU extrinsic translation and rotation respectively (i.e. the pose of the camera in the IMU frame).

To be able to triangulate features in the world and predict their position within a given image, we additionally keep a set of N previous IMU poses in the state. Those are expressed as position $p_w^{i_j}$ and quaternion $q_w^{i_j}$.

$$\mathbf{X} = \left[{p_w^i}^T {v_w^i}^T {q_w^i}^T {b_\omega}^T {b_a}^T {p_w^{i_1}}^T {q_w^{i_1}}^T \dots {p_w^{i_N}}^T {q_w^{i_N}}^T \right] \quad (1)$$

2) State Prediction: The following differential equations govern the state:

$$\dot{p}_w^i = v_w^i \quad (2)$$

$$\dot{v}_w^i = C_{(q_w^i)}^T (a_m - b_a - n_a) - g \quad (3)$$

$$\dot{q}_w^i = \frac{1}{2} \Omega(\omega_m - b_\omega - n_\omega) q_w^i \quad (4)$$

$$\dot{b}_\omega = n_{b_\omega}, \dot{b}_a = n_{b_a}, \dot{p}_i^c = 0, \dot{q}_i^c = 0, \dot{p}_w^{i_j} = 0, \dot{q}_w^{i_j} = 0 \quad (5)$$

With g as the gravity vector in the world frame and $\Omega(\omega)$ as the quaternion multiplication matrix of ω. The Rotation matrix of the quaternion q is $C_{(q)}$. We assume the camera and IMU to be mounted on a rigid body. Also, the previous camera poses are independent of the current IMU readings. Hence the zero dynamics in (5).

More details on the system propagation and covariance matrices F_d and Q_d respectively can be found in [19]. The discretized propagation matrix F_j for the past poses is the identity matrix I.

3) State Update: The update step is performed according to [1]. We use previous poses and corresponding feature measurements (so-called feature trails, see IV-A) in order to triangulate the 3D position of the selected features in world frame.

Within feature trails, we differ between processed and unprocessed measurements. A measurement is initially unprocessed. After it has been used in an update step, it is set to processed. The main difference is that while for the state update processed measurements could be discarded we are still able to use them for 3D triangulation and measurement prediction.

This position is re-projected into the current image and the residual as the pixel distance between projected and measured 2D position in the image is computed. To predict the 2D feature position p_{2D} in the image as the estimated measurement $\hat{z}$ we use

$$p_c = C_{(q_i^c)}^T C_{(q_w^i)}^T \; p_w - C_{(q_i^c)}^T C_{(q_w^i)}^T p_w^i - C_{(q_i^c)}^T p_i^c \quad (6)$$

$$\hat{z} = p_{2D} = \frac{1}{p_{c_Z}} \begin{bmatrix} p_{c_X} \\ p_{c_Y} \end{bmatrix} \quad (7)$$

With p_w and p_c as the triangulated 3D point in the world and camera frame respectively. The Kalman Filter update steps and matrices directly follow from this measurement equation assuming a calibrated camera and rectified images. For more details on the update step, please refer to [1], [2]. The main novelty in this formulation being, that we include the camera-IMU extrinsic calibration $p_w^{i_1}$ and $q_w^{i_1}$ in the state vector in our approach to obtain a self-calibrating system like in our loosely coupled counter part [19].

B. Feature Detection and Tracking

To provide the above sketched measurements, we implemented a simple feature tracker. In each frame, features are identified in the image using the FAST corner detector. This is followed by matching image patches with a patch size of 16 with sub-pixel accuracy. Subsequently found feature positions are inserted into a container which grows into a so-called feature trail.

All currently known trails are provided to the update step of the filter algorithm as an input to our proposed novel selection process (see Section IV). A trail contains the image IDs where the feature has been observed and a 2D image positions.

C. Initialization

At start-up, the algorithm aligns the measured acceleration vector to the gravity vector to generate a initial estimate for the rotation of the IMU in the world frame. By subtracting the rotated gravity vector from the measured acceleration vector, acceleration biases are estimated. Tests have shown that this initialization provides better starting values than setting biases to zero although roll/pitch inclination and IMU biases cannot be distinguished from each other from a theoretical point of view. Furthermore we collect acceleration data to compute the statistics of the acceleration measurements, assuming the system is initially at rest.

We use the IMU readings right after start-up for system state-propagation such that no special treatment or sophisticated initialization machinery has to be used in the beginning. As long as the system (i.e. the camera) is motionless no trails are generated and thus no motion is detected in the image. This is directly applied to the system (inherently resulting in a zero-motion update). The moment the system moves, features will be tracked and motion updates will be generated accordingly. The IMU readings continue to be integrated to predict the system dynamics. This initialization approach inherently allows for longer still phases in the beginning and at the same time a seamless start once the system begins to move.

IV. KEY-FRAME STRATEGY

This section details our main contribution on the selection of features and camera frames to seamlessly handle different motion scenarios without the need of motion case identification and subsequent case-specific heuristics. It also allows to leave out features in the state vector (persistent features) while still being able to inherently handle zero motion phases.

A. Feature Selection Concepts

We define a *feature trail* as set of observations of the same 3D point in consecutive camera images. The trail

information contains the image ID and corresponding 2D pixel coordinates (with sub-pixel accuracy) of all images in which the 3D point has been observed as well as the camera 6DoF poses. Instead of using all available feature trails per new image frame, we select a set of trails we consider *best*. A *good* feature trail

- has a large baseline (parallax) to give the best available information for subsequent most accurate triangulation.
- has an even distribution of parallax between camera poses for robustness against removal of a pose (see Figure 1)
- yields the most information for the state update when projected into the new frame

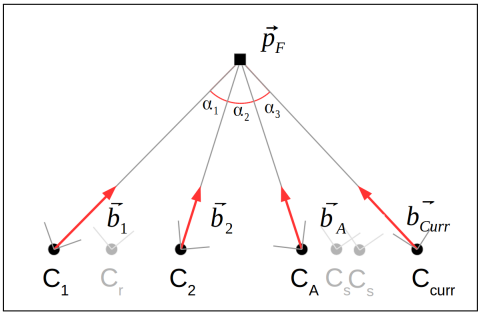

Fig. 1. All bearing vectors $\vec{b}_i$ seeing the feature $\vec{p}_F$ build the set $\mathcal{B}$, which is in turn used to calculate the parallaxes α_j. Two desirable goals are maximizing the sum of α_j and having each α_j near the mean. C_r was replaced by C_{curr} and C_S was skipped because not enough $\alpha_{A,C_{curr}}$.

We introduce the following definitions for the subsequent calculations:

$\mathcal{T}$ is the set of all feature trails, containing 2D pixel positions for each feature trail i and each camera position (keyframe) j.

C_j are keyframes describing both the set of 2D feature positions within the keyframe as well as the pose of the camera in the world [1].

$\mathcal{K}$ is the set of known keyframes in the state vector.

C_{curr} is the current camera frame, corresponding to the propagated pose for the latest measurement. Although not in $\mathcal{K}$, it is still used in the calculations.

$\vec{b}_j^i$ is the vector of the feature i in the camera frame j in rectified and normalized image coordinates.

$\alpha_{j_1 j_2}$ is the angle between two cameras j_1 and j_2 seeing the same feature and corresponds to the parallax between two frames. $\alpha(\vec{b}_{j_1}, \vec{b}_{j_2})$ is the angle between cameras j_1 and j_2 within a given feature trail.

$\mathcal{B}_i$ is the set of all bearing vectors for one feature trail i.

For each feature trail, both α_{max} and α_{mean} are calculated according to

[1]Note that, for simplicity and legibility, we refer here directly to the camera pose in the world. In our VIO algorithm, we use the IMU poses instead and apply the online estimated transformations between IMU and camera maintaining the self-calibration aspect of our framework.

$$\alpha_{max}(\mathcal{B}) = \max_{\vec{b}_i, \vec{b}_j \in \mathcal{B}} \alpha(\vec{b}_i, \vec{b}_j) \tag{8}$$

$$\alpha_{mean}(\mathcal{B}) = \frac{\sum_{i=0}^{N_b} \sum_{j=i+1}^{N_b} \alpha(\vec{b}_i, \vec{b}_j)}{N_b \cdot (N_b - 1)/2} \tag{9}$$

α_{max} (8) selects the maximum parallax within a set of bearing vectors while α_{mean} (9) calculates the mean parallax between all vectors. N_b is the number of feature measurements in $\mathcal{B}$. We do not make any assumptions on the camera trajectory. Thus we use all possible combinations of camera poses that observe the same feature to determine α_{max}.

α_{max} is used to identify the best features with respect to triangulation (assuming an outlier detection and rejection routine is in place) while α_{mean} is used to describe the best features with respect to equal distribution of parallax. Both will be used in the keyframe selection step.

We use a leightweight linear approach to triangulate features. To detect and reject outliers, we first triangulate the candidate feature using a subset of measurements. The result is reprojected into the frames not used for the calculation. Only if the distance between reprojection and measurement is smaller than a certain threshold, the measurement is considered as an inlier. If the feature is marked as inlier in at least four frames, the entire trail is considered valid.

B. Keyframe Selection

1) Idea: In order to decide whether or not to *augment* the set of key frames $\mathcal{K}$ with the current frame C_{curr} we can ask the question whether the best keyframe to *remove* is actually the current frame or not.

Thus, our keyframe selection algorithm calculates the best candidate for removal as follows. Similar to the quality of a feature trail, the best candidate keyframe has a set of desired properties. When removing the candidate keyframe, ideally (for all feature trails)

- the maximum feature trail parallax decreases minimally
- the mean parallax decreases minimally
- when replacing the removed candidate with the current frame C_{curr}, there is a net quality gain regarding parallax following Equations (8) and (9)

For these properties to be fulfilled, we define the metrics $\alpha_{max,KF}$, $\Delta\alpha_{max}$, and $\Delta\alpha_{mean}$.

$$\alpha_{max,KF}(\mathcal{B}) = \arg\max_{\vec{b}_i, \vec{b}_j \in \mathcal{B}} \alpha(\vec{b}_i, \vec{b}_j) \tag{10}$$

$$\Delta\alpha_{max}(C_i) = |\min(\frac{\pi}{2}, \alpha_{max}(\mathcal{B})) - \alpha_{max}(\mathcal{B} \setminus \vec{b}_i)| \tag{11}$$

$$\Delta\alpha_{mean}(C_i) = \alpha_{mean}(\mathcal{B} \setminus \vec{b}_i) \tag{12}$$

All metrics (10, 11, and 12) are calculated for each feature trail i and check how the corresponding $\alpha_{max,mean}$ would change assuming the candidate frame j, identified by $\vec{b}_j^i$, is removed.

Keyframes that maximize (11) contribute towards the maximal available parallax and should not be replaced. Similarly,

(12) computes the mean parallax if the bearing vector $\vec{b}_j$ from the keyframe C_j is not considered. Keyframes that maximize this metric can be replaced, since without them the mean parallax is higher. (10) returns the indices of the two keyframes that have the biggest parallax to a feature.

For one feature and its bearing vectors $\mathcal{B}$, (11) and (12) are computed for all keyframes C_j in $\mathcal{K}$. Keyframes C_j that minimize (11) and maximize (12) are candidates to be replaced.

We want to compute how each keyframe C_j is influencing the parallax when all trails $\mathcal{T}$ are considered. Therefore we need to sum up for each keyframe their contributions to the maximal parallax (8) and the mean parallax (9) over all trails and norm it with the total number of trails seen by the keyframe C_j.

Figure 2 shows that F_2 and F_3 have a very small maximal parallax and are seen by C_3 to C_5. Without normalization, either C_1 or C_2 would be selected for replacement as the summed up parallax is smaller than C_3 to C_5. This would lead to an unusable feature triangulation in the worst case.

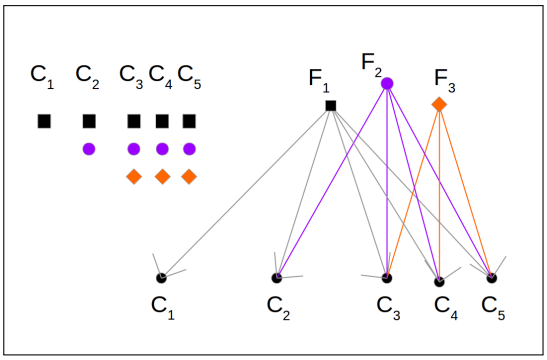

Fig. 2. Assume C_5 is the newest frame. We normalize our parallax metrics because newer keyframes (C_3 to C_5) achieve more parallax by a large number of feature trails with small maximal parallax. Although F_2 and F_3 can not be as good triangulated as F_1, they still increase the overall seen parallax of C_3 to C_5.

2) Algorithm: The algorithm selecting a candidate keyframe for removal has the following steps to be performed before every update.

1) If a slot has not yet been used, the current frame is inserted as a keyframe.

We have a fixed-sized set of N_{max} possible old poses. If such a slot is still unoccupied, we use it directly for the current pose C_{curr} and end the selection process. Otherwise, iterate over all feature trails and all camera poses (keyframes). For each camera pose, calculate the metrics how the parallax α and therefore the assumed triangulation quality would change if the keyframe C_j is removed:

2) Loop through all camera poses $C_j \in \mathcal{K}$
   a) Loop through all feature trails $t \in \mathcal{T}$; if there is a feature in C_j:
      i) Calculate $\Delta\alpha_{max}, \Delta\alpha_{mean}$ for this trail and this camera pose.
      ii) $\Delta\alpha_{max,sum}(C_j) += \Delta\alpha_{max}$
      iii) $\Delta\alpha_{mean,sum}(C_j) += \Delta\alpha_{mean}$
      iv) $counter(C_j)++$
   b) Normalize $\Delta\alpha_{max,sum}(C_j)/counter$ and $\Delta\alpha_{mean,sum}(C_j)/counter$

This calculates the metric values of keyframes for all features and all trails. The order of the iteration as described here (first $\mathcal{K}$, then $\mathcal{T}$) is to improve the understanding of the algorithm.

3) If there is at least one unused keyframe, select it as a candidate to replace and end the selection.

In case features seen from C_j in all trails are already processed, then replace C_j and end the selection.

4) Set threshold $th = \pi/128$ for minimum desired parallax between two subsequent keyframes.

Experience has shown that a parallax of $\pi/16$ will yield a triangulation exact enough for our algorithm. This is equivalent to a baseline five times smaller than the scene distance. We have a N_{max} of ten keyframes, if 8 of them see the feature, the threshold is $\pi/128$ as a parallax between two subsequent camera poses. This is the only tuning parameter in our approach.

5) Set C_A to the closest keyframe to the current camera position C_{curr}
6) If the parallax improvement between C_A and C_{curr} is smaller than threshold th, do not insert C_{curr} as a keyframe.

If the current frame does not yield an additional parallax of th, it would remove an existing frame which provides a better parallax. Therefore, we perform the update on the current frame but it is not added to the state.

7) Find keyframes with $\Delta\alpha_{max,sum}(C_j)$ below the threshold th
8) If those exist, select the keyframe with the largest $\Delta\alpha_{mean,sum}(C_j)$
9) If those don't exist, select the keyframe with the smallest $\Delta\alpha_{max,sum}(C_j)$

$\Delta\alpha_{max,sum}(C_j)$ is the (mean over all feature trails) change in the maximal parallax on removal of this keyframe. If this change is smaller than th and the current frame introduces a change larger than th, it is preferred to discard one of those states.

Such states may have been introduced by either relaxed conditions at the start or by the end of feature trails near the camera.

Assuming such a state can be found, the keyframe with the largest $\Delta\alpha_{mean,sum}(C_j)$ should be discarded. In this case, by removing the keyframe, the remaining parallax values between two frames will be most evenly distributed.

If no such state can be found, the keyframe with the smallest $\Delta\alpha_{max,sum}(C_j)$ will be discarded. This is done to maximize the resulting overall parallax.

Although it is possible to skip the threshold check for the current frame before calculating other candidates, we found that having this check improves the filter performance in hover and near-hover cases.

C. Feature Selection for Filter Update

After the selection of a candidate keyframe, we select a subset of feature trails to perform the update.

$\mathcal{T}$ contains the trails of all found features. For every trail $i \in \mathcal{T}$, we calculate the maximal parallax $\alpha_{max}(\mathcal{B}_i)$ with $\mathcal{B}_i$ consisting of all poses found in the set of old poses and the current pose $\{\mathcal{K}, C_{curr}\}$. N_i is the number of unprocessed feature measurements per trail.

If $\alpha_{max}(\mathcal{B}_i)$ is smaller than the above defined minimum parallax for depth estimation $\alpha_{TH} = \pi/16$, we set the value $V(i)$ of this trail to zero: $V(i) = 0$.

The assumption is that with less parallax than $\pi/16$, the triangulation is not good enough to generate a usable prediction of the measurement. If it is larger, we set it to

$$V(i) = \alpha_{max}(\mathcal{B}_i) \cdot \frac{N_i}{size(\mathcal{K}) + 1}$$

This prefers large parallax trails with the largest amount of updated poses (current and old). All trails are sorted according to this metric and the m best trails with a value greater than zero are used for the update in this iteration. The reason for selecting a subset of all available feature trails is that large parallax trails yield a good triangulation and a better update than other trails. The remaining trails can improve over time and are used when they have enough parallax .

For each old pose, a list of processed trails is kept. If a trail measurement has not yet been processed, its residual is added to the vector of residuals and the entries in the measurement Jacobian are stacked into the overall measurement matrix for the filter update step.

Typically, most of the measurements have already been processed resulting in a small H matrix which enables a fast calculation. If more than one trail measurement is being processed, we use a nullspace calculation (and QR decomposition) similar to [1] to reduce further computational complexity.

After finishing the update step, the feature measurements used for the update are set to the processed state.

D. Keyframe Augmentation

If the candidate to discard is the current frame, no augmentation is performed. If it is an old frame, the current estimated and updated pose is stored as an old pose together with the corresponding covariance entries. This way, we achieve a seamless keyframe handling strategy not requiring any active distinction of hovering or other motion phases.

V. RESULTS WITH REAL DATA

We performed specific test runs using real data and an AscTec quadcopter with a special focus on different motion within the same test run (e.g. with large image scale changes and still phases). For this, we used a MEMS-IMU in combination with a rigidly mounted VGA global shutter greyscale camera (only software synchronized). For the indoor datasets, we additionally used a motion tracking system to generate ground truth. To generate large image scale changes we performed so-called "take-off" and "landing" maneuvers while pointing the camera down-ward (see Figure 4). In particular, the take-off part represents a challenging situation as the framework is still in convergence phase while experiencing large image scale changes.

A. Indoor Tests

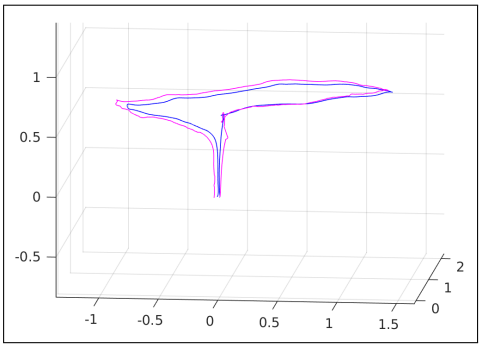

Fig. 3. 3D plot of the rectangular indoor trajectory dataset. The estimate is in purple and the ground truth is in blue. The trajectory is about 8m in total. The relative error (i.e. drift) of our approach is about 1.8%

Fig. 4. Image scale change during take-off phase for the rectangular indoor trajectory dataset. The red circle marks the same area in all four images underlining the vast scale change during this maneuver.

The quadcopter was moved on a rectangular trajectory of approximately 1×2m and 1m height. We start our algorithm when the quadcopter is on the ground. Despite the large scale changes in the beginning, our algorithm provides accurate pose estimates at all times (Figure 3). The position error at the end of the trajectory is approximately 15cm at total trajectory length of 8m (i.e. 1.8% drift) which is only slightly larger than reported by other approaches. Ours, however, seamlessly copes with both take-off and landing, as well as hover motion during the data set without requiring persistent features nor special tratment for different motion. The outcome of 13 different indoor trajectories is summarized in the table I. As it can be seen in Figure 5 the average computation time for updates is around 4ms.

B. Long Hovering without Persistent Features

The algorithm does not need to store and manage persistent features that other algorithms require in order to estimate the pose when there is no baseline in the camera movement. One could argue that this might lead to failure once all trails with enough parallax are lost. However, we show with a 5min

TABLE I

INDOOR TRAJECTORIES WITH GROUND TRUTH.

Final Position Error [m]	Trajectory Length [m]	Relative Error [%]
0.3274	7.3494	4.46 %
0.2948	7.3853	3.99 %
0.1662	6.8833	2.41 %
0.1311	7.1140	1.84 %
0.1268	7.1143	1.78 %
0.1639	6.4386	2.55 %
0.1612	7.2942	2.21 %
0.4861	7.4332	6.54 %
0.1975	7.6182	2.59 %
0.0524	6.8306	0.77 %
0.2857	5.4097	5.28 %
0.3920	6.6872	5.86 %
0.1886	7.3773	2.56 %

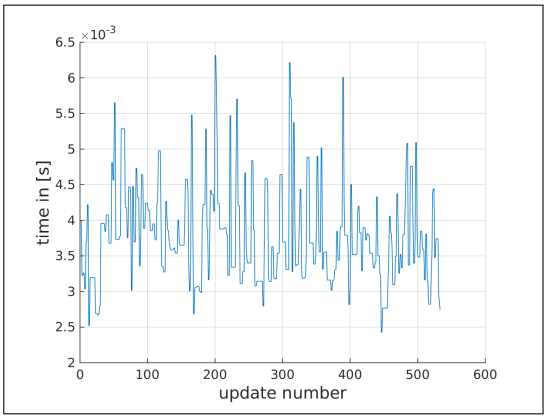

Fig. 5. The filter updates only need about 4ms on a i7 CPU @ 2.1GHz.

flight test, that this is rarely the case in practice (Figure 6). This is possible because we select keyframes to be replaced according to the quality with respect to triangulation and because we allow already processed feature measurements to be used for triangulation. In the hover case, we are able to use older features with the newest camera measurement without keeping persistent features in the state. The feature trails are robustly found for long periods of time including the take-off (see Figure 6).

C. Comparison against other VIO algorithms during large image-scale changes

As explained in Section II, we use Rovio[13], Okvis[9] and VINS-Mono [10] for comparison purposes. As before, in the indoor dataset, we focus on having large scale changes (e.g. take-off) and hover phases both in the same run. Within this frame, for each estimator, we tried to find the most favorable condition for each algorithm in order to allow a fair comparison: We varied the starting time between initialization and the first velocity peak to match each algorithm's requirement on initialization and we used different parameters for each algorithm with the goal of estimating the trajectory best possible. The best achieved (i.e. cherry-picked) run was then used to compare against our algorithm.

The performance of the algorithms can be seen in Figure 7 and 8. Okvis diverged for every take-off data set. Rovio managed to stabilize for two data sets after initially diverging. On

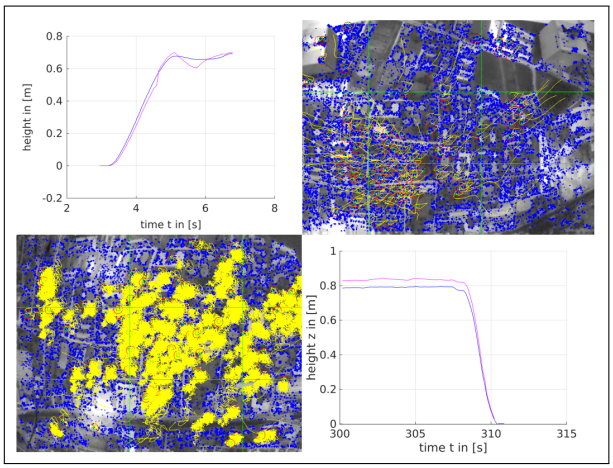

Fig. 6. Hovering without persistent features for 5min. Top left: take-off phase (blue=truth, magenta=estimate). Top right: found features during and after take-off. Bottom left: same features still found and tracked after 5min uninterrupted flight. Bottom right: landing phase (same coloring).

the third it also diverged. VINS-Mono did not initialize right away. It usually initialized once a steady position at hover was reached but still had convergence issues (see oscillations in 7). All algorithms were working properly, when they were initialized at hovering conditions not experiencing large scale changes in the beginning.

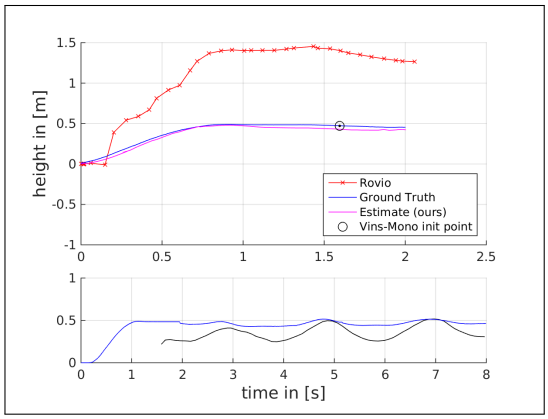

Fig. 7. Top: Comparison of VIO algorithms at take-off: Rovio stabilizes only after severe convergence issues. VINS-Mono only initializes once in hover mode at a certain scene distance. Convergence issues are shown in the bottom plot. Once converged, VINS-Mono has comparable performance to our approach with even slightly less drift, but higher computational complexity. OKVIS was not able to handle these data sets at all.

D. Outdoor High Altitude Trajectory

The proposed algorithm was also tested outdoors over a grass field to show its ability to cope in real, larger and self-similar environments. Figure 9 shows that the algorithm copes well with the homogeneous grass texture and the long take-off experiencing large scale changes. The trajectory was about 100m long at a varying height between 5 to 10m. The comparison with GPS data shows that the estimated trajectory has a RMSE of $\{1.59, 0.99, 1.00\}$m in x-, y-, and z-direction respectively (Figure 9). Also shown is the

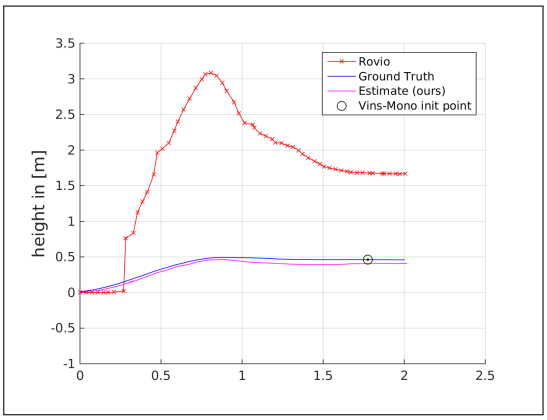

Fig. 8. Comparison of VIO algorithms at take-off. VINS-mono initializes later and has comparable performance. Rovio diverges first, then stabilizes.

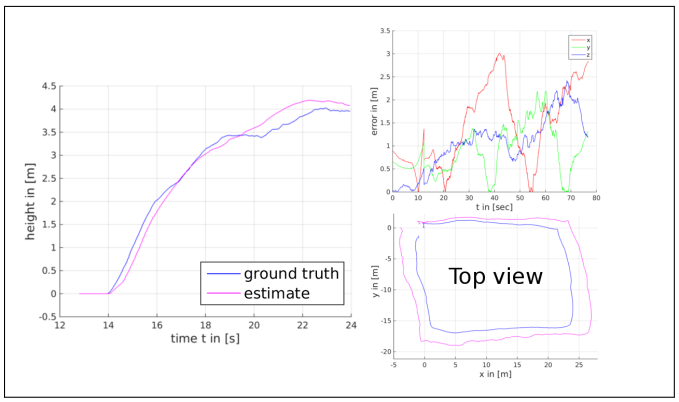

Fig. 9. Take-off and rectangular trajectory outdoor for long distance and high altitude. Error plots of filter compared to GPS.

estimation of the height compared to an on-board differential pressure sensor. The plots indicate that the height was estimated correctly and the algorithm was able to manage the scale change at take-off, landing, and throughout the trajectory very well.

VI. CONCLUSIONS

We have shown that with our proposed keyframe and feature selections strategy, we outperform current state-of-the-art VIO algorithms in situations with large image scale changes.

Our proposed keyframe and feature selections strategy additionally eliminates the need of active motion phase identification (e.g. hover modes) as it seamlessly works through all different motion scenarios without parameter adaptation nor case selection strategies. Our comparisons against other VIOs with real data show that this leads to improved performance of our algorithm.

Both indoor and outdoor datasets show that the algorithm is able to estimate the pose despite large scale changes and larger distances and without a mapping component (e.g. persistent features) with a reasonable drift.

Using only 11 poses (one current pose and 10 keyframes) and at most 5 features in every update triggered by a camera image, we are still able to correctly estimate the state during the hover case for more than 5 minutes, achieve the drift error mentioned above in the indoor dataset (1.8%) and are able to track the position over homogeneous terrain for $100m$ in the outdoor dataset.

REFERENCES

[1] A. I. Mourikis and S. I. Roumeliotis, "A multi-state constraint kalman filter for vision-aided inertial navigation," in *Robotics and automation, 2007 IEEE international conference on*. IEEE, 2007, pp. 3565–3572.

[2] M. Li and A. I. Mourikis, "High-precision, consistent ekf-based visual-inertial odometry," *The International Journal of Robotics Research*, vol. 32, no. 6, pp. 690–711, 2013.

[3] L. E. Clement, V. Peretroukhin, J. Lambert, and J. Kelly, "The battle for filter supremacy: a comparative study of the multi-state constraint kalman filter and the sliding window filter," in *Computer and Robot Vision (CRV), 2015 12th Conference on*. IEEE, 2015, pp. 23–30.

[4] S. Weiss, R. Brockers, and L. Matthies, "4DoF drift free navigation using inertial cues and optical flow," *IEEE International Conference on Intelligent Robots and Systems*, pp. 4180–4186, 2013.

[5] H. Strasdat, J. M. M. Montiel, and A. J. Davison, "Visual SLAM: Why filter?" *Image and Vision Computing*, vol. 30, no. 2, pp. 65–77, 2012. [Online]. Available: https://www.doc.ic.ac.uk/ ajd/Publications/strasdat_etal_ivc2012.pdf

[6] Y. Ling, T. Liu, and S. Shen, "Aggressive quadrotor flight using dense visual-inertial fusion," in *Robotics and Automation (ICRA), 2016 IEEE International Conference on*. IEEE, 2016, pp. 1499–1506.

[7] 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.

[8] A. Concha, G. Loianno, V. Kumar, and J. Civera, "Visual-inertial direct slam," in *Robotics and Automation (ICRA), 2016 IEEE International Conference on*. IEEE, 2016, pp. 1331–1338.

[9] 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.

[10] T. Qin, P. Li, and S. Shen, "Vins-mono: A robust and versatile monocular visual-inertial state estimator," *arXiv preprint arXiv:1708.03852*, 2017.

[11] Y. Lin, F. Gao, T. Qin, W. Gao, T. Liu, W. Wu, Z. Yang, and S. Shen, "Autonomous aerial navigation using monocular visual-inertial fusion," *Journal of Field Robotics*, vol. 35, no. 1, pp. 23–51, 2018.

[12] M. Bloesch, S. Omari, M. Hutter, and R. Siegwart, "Robust visual inertial odometry using a direct ekf-based approach," in *Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on*, Sept 2015, pp. 298–304.

[13] M. Bloesch, M. Burri, S. Omari, M. Hutter, and R. Siegwart, "Iterated extended kalman filter based visual-inertial odometry using direct photometric feedback," *The International Journal of Robotics Research*, vol. 36, no. 10, pp. 1053–1072, 2017.

[14] 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, April 2018.

[15] F. Pang, Z. Chen, L. Pu, and T. Wang, "Depth enhanced visual-inertial odometry based on multi-state constraint kalman filter," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Sept 2017, pp. 1761–1767.

[16] H. Yu and A. I. Mourikis, "Edge-based visual-inertial odometry," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Sept 2017, pp. 6670–6677.

[17] M. Ramezani, K. Khoshelham, and L. Kneip, "Omnidirectional visual-inertial odometry using multi-state constraint kalman filter," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Sept 2017, pp. 1317–1323.

[18] T. Nguyen, G. K. I. Mann, A. Vardy, and R. G. Gosine, "Likelihood-based iterated cubature multi-state-constraint kalman filter for visual inertial navigation system," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Sept 2017, pp. 4410–4415.

[19] S. Weiss and R. Siegwart, "Real-time metric state estimation for modular vision-inertial systems," in *Robotics and Automation (ICRA), 2011 IEEE International Conference on*. IEEE, 2011, pp. 4531–4537.