Optimization-Based Estimator Design for Vision-Aided Inertial Navigation

Mingyang Li and Anastasios I. Mourikis
Dept. of Electrical Engineering, University of California, Riverside
E-mail: mli@ee.ucr.edu, mourikis@ee.ucr.edu

***Abstract*—This paper focuses on the problem of real-time pose tracking using visual and inertial sensors in systems with limited processing power. Our main contribution is a novel approach to the design of estimators for these systems, which optimally utilizes the available resources. Specifically, we design a hybrid estimator that integrates two algorithms with complementary computational characteristics, namely a sliding-window EKF and EKF-SLAM. To decide which algorithm is best suited to process each of the available features at runtime, we learn the distribution of the feature number and of the lengths of the feature tracks. We show that using this information, we can predict the expected computational cost of each feature-allocation policy, and formulate an objective function whose minimization determines the optimal way to process the feature data. Our results demonstrate that the hybrid algorithm outperforms each individual method (EKF-SLAM and sliding-window EKF) by a wide margin, and allows processing the sensor data at real-time speed on the processor of a mobile phone.**

I. Introduction

In this paper we address the problem of tracking the 3D-pose of small, resource-constrained systems in unknown environments. Specifically, we are interested in estimating the motion of miniature devices, similar in size to a mobile phone. In contrast to medium- and large-scale systems, (e.g. mobile robots, UAVs, autonomous cars), small devices have limited computational capabilities and battery life, factors which make the pose-estimation problem challenging. In the absence of GPS, the types of sensors that can be used for pose estimation in small-scale systems are quite restricted. In our work, we opt for using visual and inertial measurements. This is motivated by the fact that cameras and inertial measurement units (IMUs) are small, lightweight, and inexpensive sensors, which can operate in almost any environment. Our goal is to estimate the moving platform's trajectory *only*, using the inertial measurements and the observations of naturally occurring point features. We do not assume that a map of the area is available, nor do we aim to build such a map. Thus, the problem we address is analogous to visual odometry, with the added characteristic that an IMU is available. We term the approach *visual-inertial odometry*.

The key challenge that needs to be addressed when designing an estimator for this task is the limited availability of processing power. Feature detection algorithms can track hundreds of features in images of natural scenes, but processing all this data in real time is challenging, particularly for a small, resource-constrained device. To date, the existence of resource limitations has been typically addressed by *proposing* a motion estimation algorithm (e.g., EKF-SLAM, sliding-window iterative estimation), and testing whether it can operate within the given resource constraints. If it can, success is declared. However, this design paradigm has the shortcoming that it cannot ensure optimal use of the available resources: even though one algorithm may operate using a certain amount of CPU time, it is possible that a different algorithm can either (i) produce a more accurate estimate in the same amount of time, or (ii) compute an estimate of the same precision faster. Clearly, this situation is undesirable.

An additional difficulty that arises when designing an estimator for any localization task is that, typically, the computational efficiency of different methods depends strongly on the nature of each particular dataset. For instance, one algorithm may outperform all others when the environment is feature-rich and the vehicle is moving slowly, while a different algorithm may be the fastest in feature-sparse environments under fast motion. This makes algorithm selection a difficult task, for which no general, systematic methods exist to date.

To address the limitations described above, in this work we propose a *new paradigm* for the design of motion estimation algorithms. Specifically, we propose designing a hybrid estimator that incorporates two (or more, in general) algorithms with complementary computational characteristics. This makes it possible to decide, in real time, to process different measurements (e.g., different features) by a different algorithm. Since the optimal choice for each measurement will depend on the characteristics of the sensor data, in the proposed framework we employ *statistical learning* to learn these characteristics. Gathering statistical information allows us to compute the *expected* cost of any strategy for allocating measurements to algorithms. To identify the optimal strategy we solve, in real time, an *optimization problem* whose objective function is the expected computation time.

In this paper, we apply the approach described in the preceding paragraph to the problem of visual-inertial odometry. Specifically, we design a hybrid extended Kalman filter (EKF), which integrates EKF-SLAM with a sliding-window EKF estimator. As explained in Section III, these two estimators process the same measurement information in different ways, and have complementary computational characteristics. We show that the optimal choice of algorithm to process each individual feature depends on the distribution of the feature-track lengths of *all* features. This distribution is not known in

advance (it depends on the environment, the camera motion, as well as the feature tracker used), and therefore we learn it from the image sequence. Using this information, the optimal strategy for processing the feature measurements can be computed by solving a one-variable optimization problem, as shown in Section V. Our results demonstrate that the hybrid algorithm outperforms each individual method (EKF-SLAM and sliding-window EKF) by a wide margin. In fact, the hybrid filter allows processing the sensor data at real-time speed on a mobile phone, something that is not possible with either of the individual algorithms.

II. Related Work

A huge number of methods have been proposed for pose estimation, and thus any attempt at a general literature survey in the limited space available would be incomplete. We therefore here focus on methods that seek to optimize, in some sense, the computational efficiency of localization. To the best of our knowledge, no prior work exists that employs learning of the feature tracks' characteristics to *explicitly* optimize the computational cost of estimation. Past work has focused primarily on SLAM, and can broadly be categorized as:

a) Exact reformulations of the SLAM equations: Typical methods decompose the computations into smaller parts, and selectively carry out the only necessary computations at each time instant (see e.g., [1]–[4]). Typically, in these methods the currently visible features are updated normally at every time step, while the remaining ones are updated only "on demand" or when re-visited. In the case of visual-inertial odometry, however, where the state vector contains only the actively tracked features and no loop-closure is considered, these methods are not applicable.

b) Approximations of the SLAM equations: On the other hand, several methods exist that employ approximations to the SLAM equations (e.g., [5]–[7] and references therein), to reduce the required computations. In contrast to these methods, which trade-off information for efficiency, our proposed approach involves no information loss, and no approximations, other than the inaccuracies due to the EKF's linearization.

c) Feature selection methods: A different family of approaches seek to reduce the computational cost of localization by processing only the most valuable, in terms of uncertainty reduction, measurements (e.g., [8], [9]). Only processing a small subset of carefully selected measurements can often result in high accuracy. However, since the remaining measurements are simply discarded, loss of information inevitably occurs. In our proposed approach, *all* the available measurements are processed, and no localization information is lost.

III. Estimators for visual-inertial odometry

We now examine the possible choices of algorithms for processing the feature observations in visual-inertial odometry. Since real-time performance is necessary, any candidate algorithm must have bounded computational complexity, irrespective of the duration of the trajectory. Within this class, practically all algorithms proposed to date employ either an EKF (e.g., [10]–[12]), or iterative minimization over a window of states (e.g., [13]–[15]). The latter approaches iteratively relinearize the measurement equations to better deal with their nonlinearity, which, however, incurs a high computational cost. In our recent work [16] we showed that, by choosing linearization points that preserve the system's observability properties, EKF-based visual-inertial odometry can attain *better* accuracy than iterative-minimization methods, at only a fraction of the computational cost. Therefore, in this paper we only focus on EKF-based algorithms.

Within the class of EKF methods, there are two possible formulations of the estimator. Specifically, we can employ the EKF-SLAM approach (e.g. [17]–[19] and references therein), in which the state vector contains the current IMU state as well as feature positions. To keep the computations bounded, features that leave the field of view must be removed from the state vector, leaving only the currently observed ones [19]. Other EKF algorithms instead maintain a sliding window of camera poses in the state vector, and use the feature observations to apply probabilistic constraints between these poses (e.g., [10], [20], [21]). Out of this class of methods, the multi-state-constraint Kalman filter (MSCKF) algorithm [10] uses the all the information provided by the feature measurements, which makes it our preferred method.

Both the MSCKF and EKF-SLAM use exactly the same information, and only differ in the way they organize the computations, and in linearization (more on this in Section VI). If the measurement models were linear, both methods would yield exactly the same result, equal to the MAP estimate of the IMU pose [22]. With respect to computational cost, however, the two methods differ dramatically. For the EKF-SLAM algorithm the cost at each time-step is cubic in the number of features (since all features in the state vector are observed). In contrast, the MSCKF has computational cost that scales linearly in the number of features, but cubically in the length of the feature tracks (see Section IV-B). Therefore, if many features are tracked, each in a small number of frames, the MSCKF approach is preferable, but if few features are tracked over long image sequences, EKF-SLAM would result in lower computational cost[1]. Clearly, EKF-SLAM and the MSCKF algorithm are complementary, with each being superior in different circumstances. This motivates us to integrate both algorithms in a single, hybrid filter, as described next.

IV. The hybrid MSCKF/SLAM Algorithm

In this section we present the hybrid MSCKF/SLAM estimator for visual-inertial odometry. We begin our presentation by briefly outlining the MSCKF algorithm, which was originally proposed in [10]. In our work, we employ the modified MSCKF algorithm, which attains increased accuracy and consistency by choosing linearization points that ensure the correct observability properties [16].

[1]Similar conclusions can be reached from the analysis of [23], which compares EKF-SLAM to iterative optimization over a window of poses.

A. The MSCKF algorithm for visual-inertial odometry

The MSCKF does not include the feature positions in the state vector, but instead uses the feature measurements to directly impose constraints between the camera poses. The state vector of the MSCKF is:

$$\mathbf{x}_k^T = \begin{bmatrix} \mathbf{x}_{I_k}^T & \mathbf{x}_{C_1}^T & \mathbf{x}_{C_2}^T & \cdots & \mathbf{x}_{C_m}^T \end{bmatrix}^T \tag{1}$$

where $\mathbf{x}_{I_k}$ is the current IMU state, and $\mathbf{x}_{C_i}$, $i = 1 \dots m$ are the camera poses (positions and orientations) at the times the last m images were recorded. The IMU state is defined as:

$$\mathbf{x}_I = \begin{bmatrix} \bar{\mathbf{q}}^T & \mathbf{p}^T & \mathbf{v}^T & \mathbf{b}_g^T & \mathbf{b}_a^T \end{bmatrix}^T \tag{2}$$

where $\bar{\mathbf{q}}$ is the unit quaternion describing the rotation from the global frame to the IMU frame, $\mathbf{p}$ and $\mathbf{v}$ denote the IMU position and velocity, respectively, while $\mathbf{b}_g$ and $\mathbf{b}_a$ are the gyroscope and accelerometer biases.

The MSCKF uses the IMU measurements to propagate the current IMU state and the covariance matrix, $\mathbf{P}_{k+1|k}$, while the feature measurements are used for EKF updates. Let's assume that the i-th feature, which has been observed in ℓ images, has just been lost from tracking (e.g., it went out of the field of view). At this time, we use all the measurements of the feature to carry out an EKF update. Let the observations of the feature be described by the nonlinear equations $\mathbf{z}_{ij} = \mathbf{h}(\mathbf{x}_{C_j}, \mathbf{f}_i) + \mathbf{n}_{ij}$, for $j = 1 \dots \ell$, where $\mathbf{f}_i$ is the feature position (described by the inverse-depth parameterization [24]) and $\mathbf{n}_{ij}$ is the noise vector, modelled as zero-mean Gaussian with covariance matrix $\sigma^2\mathbf{I}_2$. Using all the measurements we compute a feature position estimate $\hat{\mathbf{f}}_i$, and then compute the residuals $\tilde{\mathbf{z}}_{ij} = \mathbf{z}_{ij} - h(\hat{\mathbf{x}}_{C_j}, \hat{\mathbf{f}}_i)$, $j = 1 \dots \ell$. By linearizing, these residuals can be written as:

$$\tilde{\mathbf{z}}_{ij} \simeq \mathbf{H}_{ij}\tilde{\mathbf{x}}_{C_j} + \mathbf{H}_{f_{ij}}\tilde{\mathbf{f}}_i + \mathbf{n}_{ij},\ j = 1 \dots \ell \tag{3}$$

where $\tilde{\mathbf{x}}_{C_j}$ and $\tilde{\mathbf{f}}_i$ are the estimation errors of the j-th camera pose and i-th feature respectively, and the matrices $\mathbf{H}_{ij}$ and $\mathbf{H}_{f_{ij}}$ are the corresponding Jacobians. Since the feature is not included in the MSCKF state vector, we proceed to marginalize it out. For this purpose, we first form the vector containing the ℓ residuals from all the feature's measurements:

$$\tilde{\mathbf{z}}_i \simeq \mathbf{H}_i\tilde{\mathbf{x}} + \mathbf{H}_{f_i}\tilde{\mathbf{f}}_i + \mathbf{n}_i \tag{4}$$

where $\tilde{\mathbf{z}}_i$ and $\mathbf{n}_i$ are block vectors with elements $\tilde{\mathbf{z}}_{ij}$ and $\mathbf{n}_{ij}$, respectively, and $\mathbf{H}_i$ and $\mathbf{H}_{f_i}$ are matrices with block rows $\mathbf{H}_{ij}$ and $\mathbf{H}_{f_{ij}}$, for $j = 1 \dots \ell$. Subsequently, we define the residual vector $\tilde{\mathbf{z}}_i^o = \mathbf{V}^T\tilde{\mathbf{z}}_i$, where $\mathbf{V}$ is a matrix whose columns form a basis of the left nullspace of $\mathbf{H}_{f_i}$. From (4), we obtain:

$$\tilde{\mathbf{z}}_i^o = \mathbf{V}^T\tilde{\mathbf{z}}_i \simeq \mathbf{V}^T\mathbf{H}_i\tilde{\mathbf{x}} + \mathbf{V}^T\mathbf{n}_i = \mathbf{H}_i^o\tilde{\mathbf{x}} + \mathbf{n}_i^o \tag{5}$$

Once $\tilde{\mathbf{z}}_i^o$ and $\mathbf{H}_i^o$ are available, we proceed to carry out a Mahalanobis gating test for the residual $\tilde{\mathbf{z}}_i^o$, by computing:

$$\gamma = (\tilde{\mathbf{z}}_i^o)^T \left(\mathbf{H}_i^o\mathbf{P}_{k+1|k}(\mathbf{H}_i^o)^T + \sigma^2\mathbf{I}\right)^{-1}\tilde{\mathbf{z}}_i^o \tag{6}$$

and comparing it against a threshold given by the 95-th percentile of the χ^2 distribution. By stacking together the residuals of all features that pass this gating test, we obtain:

$$\tilde{\mathbf{z}}^o = \mathbf{H}^o\tilde{\mathbf{x}} + \mathbf{n}^o \tag{7}$$

where $\tilde{\mathbf{z}}^o$, $\mathbf{H}^o$, and $\mathbf{n}^o$ are block vectors/matrices, with block rows $\tilde{\mathbf{z}}_i^o$, $\mathbf{H}_i^o$, and $\mathbf{n}_i^o$, for $i = 1 \dots n$, respectively. We can now use the above residual, which only involves the camera poses, for an EKF update. However, if a large number of features are processed at each time instant (the common situation), further computational savings can be achieved. Specifically, in [10] it is shown that instead of using the above residual, we can equivalently compute the thin QR factorization of $\mathbf{H}^o$, written as $\mathbf{H}^o = \mathbf{Q}\mathbf{H}^r$, and then employ the residual $\tilde{\mathbf{z}}^r$ for updates, defined as:

$$\tilde{\mathbf{z}}^r = \mathbf{Q}^T\tilde{\mathbf{z}}^o = \mathbf{H}^r\tilde{\mathbf{x}} + \mathbf{n}^r \tag{8}$$

where $\mathbf{n}^r$ is the $r \times 1$ noise vector, with covariance matrix $\sigma^2\mathbf{I}_r$. Once the residual $\tilde{\mathbf{z}}^r$ and the matrix $\mathbf{H}^r$ have been computed, we compute the state correction and update the covariance matrix via the standard EKF equations:

$$\boldsymbol{\Delta}\mathbf{x} = \mathbf{K}\tilde{\mathbf{z}}^r \tag{9}$$

$$\mathbf{P}_{k+1|k+1} = \mathbf{P}_{k+1|k} - \mathbf{K}\mathbf{S}\mathbf{K}^T \tag{10}$$

$$\mathbf{S} = \mathbf{H}^r\mathbf{P}_{k+1|k}(\mathbf{H}^r)^T + \sigma^2\mathbf{I}_r \tag{11}$$

$$\mathbf{K} = \mathbf{P}_{k+1|k}(\mathbf{H}^r)^T\mathbf{S}^{-1} \tag{12}$$

B. Computational complexity of the MSCKF

The way in which the feature measurements are processed in the MSCKF is optimal, in the sense that no approximations are used, except for the EKF's linearization [22]. This is true, however, *only* if the sliding window of states, m, contains at least as many states as the longest feature track. If it does not, then the measurements that occurred more than m timesteps in the past cannot be processed. Therefore, to use all the available feature information, the MSCKF must maintain a window of states long enough to include the longest feature tracks.

We now examine the dependence of the computational requirements of the MSCKF on the number of features and the lengths of the feature tracks. The computation time of the MSCKF is dominated by the following operations:
1) The computation of the residual and Jacobian matrix in (5), for each feature. If n features are processed, with feature track lengths $\ell_i, i = 1 \dots n$, this requires $\mathcal{O}(\sum_{i=1}^n \ell_i^3)$ operations.
2) The Mahalanobis test, requiring $\mathcal{O}(\sum_{i=1}^n \ell_i^3)$ operations.
3) The computation of the residual and the Jacobian matrix in (8), which, by exploiting the structure in $\mathbf{H}^o$, can be implemented in approximately $\mathcal{O}(\sum_{i=1}^n \ell_i^3)$ operations.
4) The computation of the Kalman gain and the update of the covariance matrix, which require $\mathcal{O}(r^3/6 + r(15+6m)^2)$ operations. Here $15 + 6m$ is the size of the state covariance matrix, and r is the number of rows of $\mathbf{H}^r$. It can be shown that, in general, r (which equals the number of independent constraints for the camera poses) is given by [25]:

$$r = 2(\ell_{(1)} + \ell_{(2)} + \ell_{(3)}) - 7 \tag{13}$$

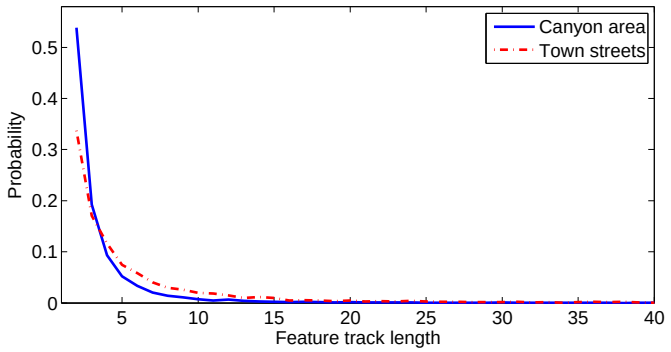

Fig. 1. Distribution of the feature track lengths in two parts of the Cheddar Gorge dataset [26]. The features detected were Harris corners, matched by normalized cross-correlation.

where $\ell_{(1)}$, $\ell_{(2)}$, and $\ell_{(3)}$ are the three longest feature tracks.

From the above we see that, while the computational cost of the MSCKF is linear in the number of features, it is at least quadratic in the size of the sliding window, m. In fact, if the size of the sliding window is chosen to be equal to the longest feature tracks, then r is also $\mathcal{O}(m)$, and the overall complexity becomes *cubic* in m. This demonstrates a shortcoming of the MSCKF: to preserve all measurement information, the complexity of the algorithm scales cubically as a function of the *longest* feature track length.

In real-world datasets the distribution of the feature-track lengths is non-uniform, with many features tracked for short durations, and very few stabler features tracked over longer periods. For instance, Fig. 1 shows the distribution of the feature track lengths in two parts of a real-world dataset [26]. It can be seen that, even though the longest feature tracks reach 40 images, the vast majority of features is tracked for a small number of frames (the percentage of features tracked in five or less images is 88% and 69% in the two cases shown, respectively). To be able to process the small percentage of features with long track lengths, the MSCKF must maintain a long sliding window, which is computationally inefficient. In what follows, we show how we can integrate the MSCKF algorithm with EKF-SLAM, to address this limitation.

C. The hybrid MSCKF/SLAM algorithm

The alternative way of processing feature measurements in the EKF is to include them in the state vector, and use their observations as in the standard visual-SLAM formulation. As discussed in Section III, this approach has computational properties complementary to those of the MSCKF: while the MSCKF is better at processing many features tracked for a few frames, EKF-SLAM is faster when few features are tracked for many frames. This motivates us to combine both approaches in a single, "hybrid" estimator. Specifically, we formulate a filter whose state vector at time-step k contains the current IMU state, a sliding window of m camera poses, and s_k features:

$$\mathbf{x}_k = \begin{bmatrix} \mathbf{x}_{I_k}^T & \mathbf{x}_{C_1}^T & \cdots & \mathbf{x}_{C_m}^T & \mathbf{f}_1^T & \cdots & \mathbf{f}_{s_k}^T \end{bmatrix}^T \quad (14)$$

This provides us with the ability to choose whether a feature will be processed using the MSCKF approach, or whether it will be included in the state vector and processed as in EKF-SLAM. By analyzing in detail the computational requirements of each EKF update, it can be shown that, when many features are present, there is nothing to be gained by initializing in the state vector any feature observed fewer than m times [25]. Thus, the optimal (in terms of computational requirements) strategy for using the features turns out to be a simple one: if feature i's track is lost after fewer than m frames (i.e., $l_i < m$), then the feature is processed using the MSCKF equations, as described in Section IV-A. On the other hand, if a feature is still actively being tracked after m images, it is initialized into the state vector, and used for SLAM. Thus, the *only* choice that one needs to make is the size of the sliding window, m. In Section V we show how this can be chosen optimally.

At a given time step, the EKF update is carried out using a residual vector constructed by stacking together the residual computed from the "MSCKF features" and from the s_k observations of the "SLAM features":

$$\tilde{\mathbf{z}}_k = \begin{bmatrix} \tilde{\mathbf{z}}_k^r \\ \tilde{\mathbf{z}}_{1m} \\ \vdots \\ \tilde{\mathbf{z}}_{s_k m} \end{bmatrix} \simeq \begin{bmatrix} \mathbf{H}^r \\ \mathbf{H}_{1m} \\ \vdots \\ \mathbf{H}_{s_k m} \end{bmatrix} \tilde{\mathbf{x}}_k + \mathbf{n}_k = \mathbf{H}_k \tilde{\mathbf{x}}_k + \mathbf{n}_k \quad (15)$$

In the above equation, $\tilde{\mathbf{z}}_{jm}$, for $j = 1 \dots s_k$ are the residuals of the observations of the s_k SLAM features from the latest camera state (state m), and $\mathbf{H}_{jm}$, for $j = 1 \dots s_k$ are the associated Jacobian matrices (see (3)). Each of these residuals is a 2×1 vector, while each $\mathbf{H}_{jm}$ is a $2\times(15+6m+3s_k)$ matrix (here $15+6m+3s_k$ is the size of the state covariance matrix). The residual $\tilde{\mathbf{z}}_k$ and the Jacobian $\mathbf{H}_k$ are used for updating the state and covariance matrix, similarly to (9)-(12).

For initializing new features, the m measurements of a feature are used to triangulate it and to compute its initial covariance and the cross-correlation with other filter states. In our implementation, we use the inverse-depth feature parameterization [24], due to its superior linearity properties. The latest camera clone, $\mathbf{x}_{C_m}$, is used as the "anchor" state, based on which the inverse-depth parameters are computed. If the feature is still actively being tracked at the time its anchor state needs to be removed from the sliding window, the anchor is changed to the most recent camera state, and the covariance matrix of the filter is appropriately modified. The details of feature initialization as well as the "anchor change" process are shown in [25]. Finally, the hybrid MSCKF/SLAM algorithm is described in Algorithm 1.

V. Optimizing the Performance of the Hybrid EKF

In this section we show how the size of the sliding window, m, can be selected so as to minimize the computational cost of the hybrid MSCKF/SLAM filter. As the results in Section VI show, the choice of m has a profound effect on the time requirements of the algorithm. With a suitable choice, the hybrid method can significantly outperform each of its individual components.

Algorithm 1 Hybrid MSCKF/SLAM algorithm

Propagation: Propagate the state vector and covariance matrix using the IMU readings.

Update: Once camera measurements become available:

- Augment the state vector with the latest camera pose.
- For features to be processed in the MSCKF (feature tracks of length smaller than m), do the following
  - For each feature to be processed, calculate the residual and Jacobian matrix in (5).
  - Perform the Mahalanobis gating test in (6).
  - Using all features that passed the gating test, form the residual vector and the Jacobian matrix in (8).
- For features that are included in the state vector, compute the residuals and measurement Jacobian matrices, and form the residual $\tilde{\mathbf{z}}_k$ and matrix $\mathbf{H}_k$ in (15).
- Update the state vector and covariance matrix, via (9)-(12), using the residual $\tilde{\mathbf{z}}_k$ and Jacobian matrix $\mathbf{H}_k$.
- Initialize features tracked in all m images of the sliding window.

State Management:

- Remove SLAM features that are no longer tracked, and change the anchor pose for SLAM features anchored at the oldest pose.
- Remove the oldest camera pose from the state vector. If no feature is currently tracked for more than m_o poses (with $m_o < m-1$), remove the oldest $m - m_o$ poses.

A. Operation count for each update

By carefully analyzing the computations needed, we calculate the number of floating-point operations per update of the hybrid algorithm:

$$f_k = \alpha_1 \sum_{i=1}^{n} \ell_i^3 + \alpha_2 (r + 2s_k)^3 + \alpha_3 (r + 2s_k)(15 + 6m + 3s_k)^2 + l.o.t \qquad (16)$$

where the α_i's are known constants, n is the number of features used in the MSCKF, r is defined in (13), and $l.o.t.$ stands for lower-order terms[2]. The three terms shown above correspond to the computation of the MSCKF residual, the Cholesky factorization of the matrix $\mathbf{S}$ in (12), and the covariance update equation, respectively. Note that (16) also models the probability of failure for the Mahalanobis test.

It is interesting to examine the properties of (16). First, we note that since r represents the number of constraints provided by the MSCKF features for the poses in the sliding window, it is bounded above by $6m-7$: the total number of unknowns in the sliding window is $6m$, and the feature measurements cannot provide any information about the global pose or scale, which correspond to 7 degrees of freedom. If many features are available, we will have $r \approx 6m-7$, and thus:

$$f_k \approx \alpha_1 \sum_{i=1}^{n} \ell_i^3 + \alpha_2 (6m + 2s_k - 7)^3 + \alpha_3 (6m + 2s_k - 7)(15 + 6m + 3s_k)^2$$

This approximate expression makes it possible to gain intuition about the behavior of the computational cost of the hybrid method: as the size of the sliding window, m, increases, more features will be processed by the MSCKF, and fewer features will be included in the state vector of the filter. Thus, as m increases, the term $\sum_{i=1}^{n} \ell_i^3$ will increase, but s_k will decrease rapidly. These two opposing trends result in the performance curves shown in Section VI (e.g., Fig. 3).

B. Minimizing the average operation count

We now turn our attention to determining the optimal value of m, in order to minimize the runtime of the hybrid EKF estimator. Equation (16) provides us with the operation count of one filter update, so at first glance, it may appear that one needs to minimize this equation with respect to m, at each time instant. However, that would be an ill-posed problem. To see why, consider the case where, at time step k, the sliding window has length $m = 20$, and ten features exist that have been continuously tracked in 20 images. At this time, we have the option of either increasing the size of the sliding window, or including the ten features in the state vector. Which is best depends on the *future* behavior of the feature tracks: if these features end up being tracked for a very large number of frames ($\gg 20$), then it would be preferable to include the features in the state vector. If, on the other hand, the features end up being tracked for only 21 frames, it would be preferable to increase m by one.

Clearly, it is impossible to obtain future information about any particular feature track. We can however, collect statistical information about the properties of the feature tracks, and use this information to minimize the *expected* cost of the EKF updates. This is precisely the approach we implement. Specifically, during the filter's operation, we collect statistics to learn the probability mass function (pmf) of the feature track lengths, $p(\ell_i)$, the probability of failure of the Mahalanobis gating test, as well as the pmf of the number of features tracked in the images. Using the learned pmfs, we compute the average number of operations needed for each EKF update, $\bar{f}(m)$, by direct application of the definition of the expected value of a function of random variables.

The value of m that yields the minimum $\bar{f}(m)$ can be found by exhaustive search among all possible values. However, we have found that the cost curve in practical cases is quasiconvex, which means that it has a unique minimum (see Fig. 2). Therefore, to reduce the time needed for the optimization, we perform the optimization by local search starting from a known good initial guess (e.g., the last computed threshold). Since the statistical properties of the feature tracks can change over time (see Fig. 1), we perform the learning of the pmfs as well as the

[2]The full expression for the operation count consists of tens of individual terms, whose inclusion would merely complicate the presentation, and not add much insight. Note however, that in the optimization described Section V-B, the complete expression is used.

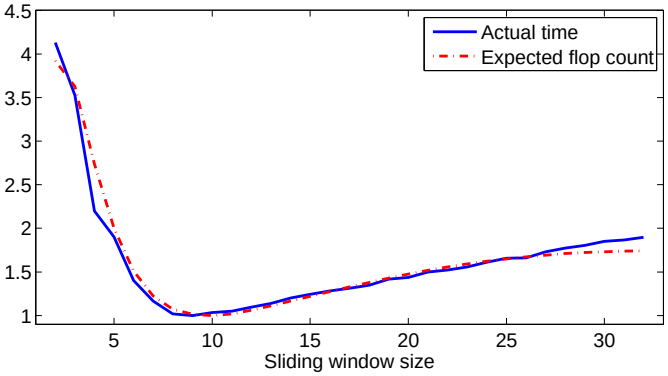

Fig. 2. Comparison of actual runtime vs. expected flop count. Since the curves have different units, each has been normalized with respect to its minimum value.

selection of the optimal threshold in consecutive time windows spanning a few seconds (15 sec in our implementation).

It is worth noting that in modern computers the use of flop counts to model computational cost is not always suitable, as performance is affected by several factors including vectorization, cache access patterns, data locality, etc. However, we have experimentally verified that in the algorithms considered here, and for our implementation, the computed flop counts closely follow the observed runtimes. Specifically, Fig. 2 shows the actual runtime of the hybrid filter, as well as the value $\bar{f}(m)$ calculated analytically, for a specific trajectory. We can observe that the two curves are very similar, with the only significant differences observed at the two "extremes" of very small or very large m. These regions are less important, however, as they fall outside the typical operational region of the hybrid filter. Thus, using $\bar{f}(m)$ as the objective function to minimize is appropriate.

VI. Experimental results

A. Simulated data

We generate simulation data that closely match real-world datasets, to have as realistic a test environment as possible. Our first simulation environment is based on the dataset of [26], which consists of a 29.6-km long trajectory through canyons, forested areas, and a town. We generate a ground truth trajectory (position, velocity, orientation) that matches the vehicle's actual trajectory, as computed by a high-precision INS system. Using this trajectory, we subsequently generate IMU measurements corrupted with noise and bias characteristics identical to those of the Xsens MTi-G sensor used in the dataset. Moreover, we generate monocular feature tracks with statistical characteristics similar to those of the features extracted in the actual dataset. The number of features observed in each image and the feature track distribution change in each part of the trajectory, as in the actual dataset (see Fig. 1). Overall, there are 232 features observed in each image on average, and the average feature track length is 5.6 frames. The IMU measurements are available at 100 Hz, while the camera frame rate is 20 Hz, as in the actual dataset. All the results reported are averages over 50 Monte-Carlo trials.

Fig. 3 plots the results from the application of the hybrid filter in this setting. Specifically, the top plot shows the average runtime for each update of the hybrid algorithm. The solid blue line is the average time when the threshold m is chosen in advance, and kept constant for the entire trajectory. The red dashed line denotes the runtime achieved when applying the optimization process described in Section V, which optimally chooses the threshold in each time window, to adapt to local feature properties. This plot shows that, by optimizing the value of m in real time, we can attain performance higher than that of any fixed threshold. Note that when m is large, no features are initialized, and thus the right-most part of the plot gives the performance of the plain MSCKF (similarly, for small m we obtain pure EKF-SLAM). Therefore, from this plot we can see that the optimal hybrid filter has a runtime 37.17% smaller than that of the MSCKF, and 72.8% smaller than EKF-SLAM.

In the second subplot of Fig. 3 we plot the RMS position error, averaged over all Monte-Carlo trials and over the duration of the trajectory. We can observe that the plain MSCKF results in the highest accuracy. We attribute this to two causes. First, in the MSCKF features are explicitly marginalized, and thus no Gaussianity assumptions are needed for the pdf of the feature position errors (as is the case in SLAM). Second, all the measurements of each feature are used jointly in the MSCKF, which means that outliers can be more easily detected, and better linearization points can be computed. By combining the MSCKF with EKF-SLAM some accuracy may be lost, as the errors for the features included in the state vector are now assumed to be Gaussian. However, we can see that if the size of the sliding window increases above a moderate value (e.g., 9 in this case), the change in the accuracy is almost negligible. Intuitively, when a sufficient number of observations is used to initialize features, the feature errors' pdf becomes "Gaussian enough" and the accuracy of the hybrid filter is very close to that of the MSCKF. Based on these results, in our optimization we do not allow the value of m to fall below a certain threshold (set to 7 in our implementation).

The timing results presented in Fig. 3 are obtained on a laptop computer with a Core i7 processor at 2.13GHz, and a single-threaded C++ implementation. Clearly, if the data were to be processed on this computer, the timing performance would easily allow for real-time implementation (the hybrid filter requires fewer than 4 msec per update with optimal m). However, our primary interest is in implementing pose estimation on small portable devices. For this reason, we conducted a second set of tests, in which the data were processed on a Samsung Galaxy S2 mobile phone, equipped with a 1.2-GHz Exynos 4210 processor. For these tests, we ported our C++ implementation to Android using the Android NDK. The simulation data were produced in a similar way as described above, by emulating a real-world dataset collected while driving in a residential area of Riverside, CA. The vehicle trajectory and statistics of the dataset (e.g., feature distribution, feature track length, and so on) are different, allowing us to test the proposed algorithm in different situations.

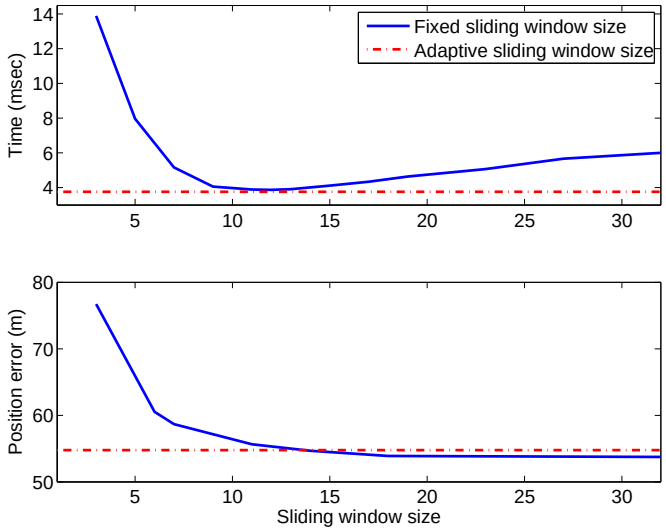

Fig. 3. Monte-Carlo simulation results: Timing performance and rms position accuracy of the hybrid filter, for changing values of m. Timing measured on a laptop computer.

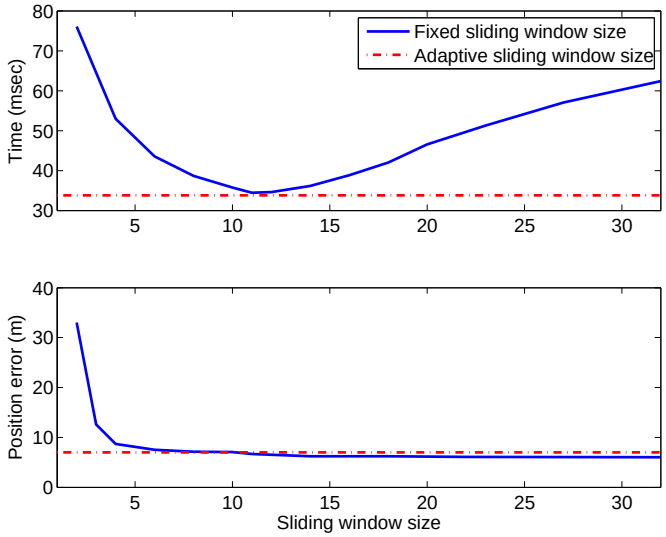

Fig. 4. Monte-Carlo simulation results: Timing performance and rms position accuracy of the hybrid filter, for changing values of m. Timing measured on a Samsung Galaxy S2 mobile phone. Note that the dataset used here is different from the one used in Fig. 3, hence the different accuracy.

Fig. 4 shows the results of the hybrid filter in this setting. These results are very similar to what was observed in the first simulation, with the optimal hybrid filter outperforming each of the individual algorithms by a wide margin (runtime 45.8% smaller than the MSCKF, and 55.6% smaller than SLAM). More importantly, however, we observe that the hybrid filter is capable of processing the data at real-time speeds, even on the much less capable processor of the mobile phone. Specifically, the average time needed for each update of the hybrid filter with optimally chosen thresholds is 33.78 msec, corresponding to a rate of 30 images per second. Since the images are recorded at 20 Hz, this means that the proposed hybrid estimator is capable of real-time processing, something that is not possible with the any of the individual methods.

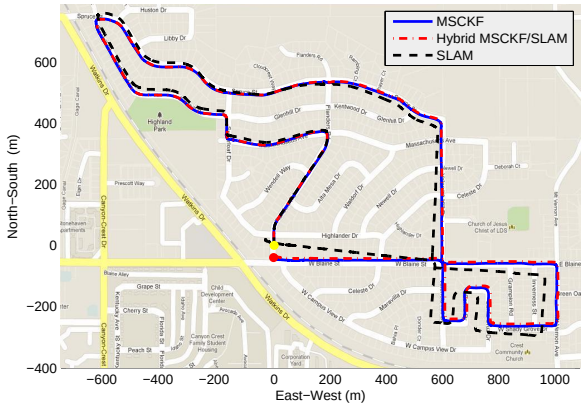

Fig. 5. The trajectory estimates of the MSCKF, EKF-SLAM, and the hybrid algorithm in the real-world experiment.

B. Real-world data

In addition to the synthetic datasets described above, we employed our proposed algorithm for processing the feature measurements recorded during a real-world experiment. During this experiment an IMU/camera platform was mounted on top of a car and driven on city streets. The sensors consisted of an Inertial Science ISIS IMU and a PointGrey Bumblebee2 stereo pair (only a single camera's images are used). The IMU provides measurements at 100 Hz, while the camera images were stored at 10 Hz. Harris feature points are extracted, and matching is carried out by normalized cross-correlation. The vehicle trajectory is approximately 5.5 km long, and a total of 7922 images are processed.

In this dataset, to compensate for the low frame rate of the images, we use lower feature-detection thresholds, which leads to a larger number of features. Specifically, 540 features are tracked in each image on average, and the average track length is 5.06 frames. This substantially increases the overall computational requirements of all algorithms. When run on the mobile phone's processor, the average time per update is 139 msec for the MSCKF, 774 msec for EKF-SLAM, and 77 msec for the hybrid filter with optimally selected thresholds. In Fig. 5 the trajectory estimates for each of the three methods are plotted on a map of the area where the experiment took place. We observe that the accuracy of the MSKCF and the hybrid filter are similar, and substantially better than that of the EKF-SLAM.

In Fig. 6 we plot the computed values for the optimal m during the experiment. This figure shows that, due to the changing properties of the feature tracks' distribution, the optimal value varies considerably over time, justifying the need for periodic re-optimization. As a final remark, we note that the optimization process itself is computationally inexpensive. In our implementation, the optimal threshold is re-computed every 15 sec, and this process takes 0.31 msec, on average, on the mobile phone. Therefore, the optimization takes up fewer than 0.003% of the overall processing time, while resulting in substantial performance gains.

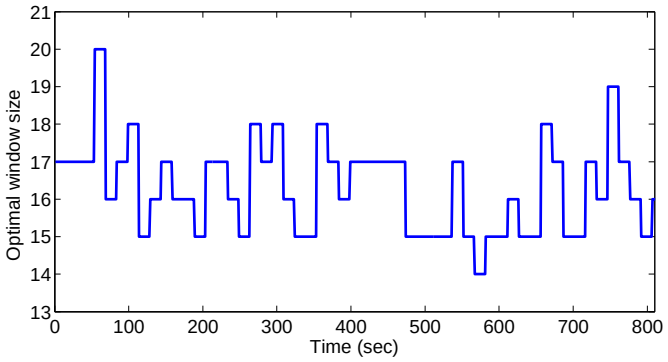

Fig. 6. The threshold selected via the optimization process for the real-world experiment.

VII. Discussion and Concluding Remarks

In this work, we presented a hybrid MSCKF/SLAM estimator for visual-inertial odometry, as well as a methodology for optimizing the estimator's computational efficiency. The results presented in the preceding section demonstrate that the proposed algorithm offers dramatic performance gains over both the MSCKF and EKF-SLAM individually. In all the cases examined, the algorithm was able to attain runtimes that allow for real-time performance, even on the low-end (by today's standards) processor of a mobile phone. These results validate the learning-based optimization approach for designing pose estimation algorithms outlined in Section I.

In this paper we have exclusively focused on optimizing the performance of the estimator (the estimation "back end" as it is sometimes called). The "front end" of visual feature extraction was not considered in this work. Obviously, for real-time operation on a resource-constrained device both of these components must be optimized, and this is the subject of our ongoing work.

As a final remark, we would like to stress that the learning-based performance optimization approach proposed in this paper is general, and can be applied to different estimators, and augmented with additional information. For instance, one can model the correlations between the track lengths of different features, which are now assumed independent. Moreover, one can use additional information (e.g., the distinctiveness of features) to predict the track length of each feature individually, or its likelihood of passing the Mahalanobis gating test. Such improvements will allow further optimization of the algorithm's performance.

References

[1] J. E. Guivant and E. M. Nebot, "Optimization of the simultaneous localization and map building algorithm for real time implementation," *IEEE Transactions on Robotics and Automation*, vol. 17, no. 3, pp. 242–257, June 2001.

[2] L. M. Paz, J. D. Tardos, and J. Neira, "Divide and conquer: EKF SLAM in O(n)," *IEEE Transactions on Robotics*, vol. 24, no. 5, pp. 1107 –1120, Oct. 2008.

[3] M. Walter, R. Eustice, and J. Leonard, "Exactly sparse extended information filters for feature-based SLAM," *International Journal of Robotics Research*, vol. 26, no. 4, pp. 335–359, Apr. 2007.

[4] M. Kaess, A. Ranganathan, and F. Dellaert, "iSAM: Incremental smoothing and mapping," *IEEE Transactions on Robotics*, vol. 24, no. 6, pp. 1365 –1378, Dec. 2008.

[5] E. D. Nerurkar and S. I. Roumeliotis, "Power-SLAM: a linear-complexity, anytime algorithm for SLAM," *International Journal of Robotics Research*, vol. 30, no. 6, pp. 772–788, May 2011.

[6] S. J. Julier, "A sparse weight Kalman filter approach to simultaneous localisation and map building," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, Maui, HI, Oct. 29-Nov. 3 2001, pp. 1251–1256.

[7] S. Thrun, Y. Liu, D. Koller, A. Ng, Z. Ghahramani, and H. Durrant-Whyte, "Simultaneous localization and mapping with sparse extended information filters," *International Journal of Robotics Research*, vol. 23, no. 7-8, pp. 693–716, Aug. 2004.

[8] H. Strasdat, C. Stachniss, and W. Burgard, "Which landmark is useful? Learning selection policies for navigation in unknown environments," in *Proceedings of the IEEE International Conference on Robotics and Automation*, Kobe, Japan, May 2009, pp. 1410 –1415.

[9] A. J. Davison and D. W. Murray, "Simultaneous localisation and map-building using active vision," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 24, no. 7, pp. 865–880, Jul. 2002.

[10] A. I. Mourikis and S. I. Roumeliotis, "A multi-state constraint Kalman filter for vision-aided inertial navigation," in *Proceedings of the IEEE International Conference on Robotics and Automation*, Rome, Italy, Apr. 2007, pp. 3565–3572.

[11] J. Kelly and G. Sukhatme, "Visual-inertial sensor fusion: Localization, mapping and sensor-to-sensor self-calibration," *International Journal of Robotics Research*, vol. 30, no. 1, pp. 56–79, Jan. 2011.

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

[13] T. Dong-Si and A. I. Mourikis, "Motion tracking with fixed-lag smoothing: Algorithm and consistency analysis," in *Proceedings of the IEEE International Conference on Robotics and Automation*, Shanghai, China, May 2011, pp. 5655 – 5662.

[14] G. Sibley, L. Matthies, and G. Sukhatme, "Sliding window filter with application to planetary landing," *Journal of Field Robotics*, vol. 27, no. 5, pp. 587–608, Sep. 2010.

[15] K. Konolige and M. Agrawal, "FrameSLAM: From bundle adjustment to real-time visual mapping," *IEEE Transactions on Robotics*, vol. 24, no. 5, pp. 1066 –1077, Oct. 2008.

[16] M. Li and A. I. Mourikis, "Improving the accuracy of EKF-based visual-inertial odometry," in *Proceedings of the IEEE International Conference on Robotics and Automation*, St Paul, MN, May 2012, pp. 828–835.

[17] A. Davison, I. Reid, N. Molton, and O. Stasse, "MonoSLAM: Real-time single camera SLAM," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 29, no. 6, pp. 1052–1067, Jun. 2007.

[18] P. Pinies and J. D. Tardos, "Scalable SLAM building conditionally independent local maps," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, San Diego, CA, Nov. 2007, pp. 3466–3471.

[19] R. Munguia and A. Grau, "Monocular SLAM for visual odometry," in *Proceedings of the IEEE International Symposium on Intelligent Signal Processing*, Alcala Henares, Spain, Oct. 2007, pp. 1–6.

[20] D. D. Diel, P. DeBitetto, and S. Teller, "Epipolar constraints for vision-aided inertial navigation," in *IEEE Workshop on Motion and Video Computing*, Breckenridge, CO, Jan. 2005, pp. 221–228.

[21] S. I. Roumeliotis, A. E. Johnson, and J. F. Montgomery, "Augmenting inertial navigation with image-based motion estimation," in *Proceedings of the IEEE International Conference on Robotics and Automation*, Washington D.C, May 2002, pp. 4326–33.

[22] M. Li and A. I. Mourikis, "Consistency of EKF-based visual-inertial odometry," University of California Riverside, Tech. Rep., 2011, www.ee.ucr.edu/∼mourikis/tech_reports/VIO.pdf.

[23] H. Strasdat, J. M. M. Montiel, and A. J. Davison, "Real-time monocular SLAM: Why filter?" in *Proceedings of the IEEE International Conference on Robotics and Automation*, Anchorage, AL, May 2010, pp. 2657–2664.

[24] J. Montiel, J. Civera, and A. Davison, "Unified inverse depth parametrization for monocular SLAM," in *Proceedings of Robotics: Science and Systems*, Philadelphia, PA, Aug. 16-19 2006, pp. 81–88.

[25] M. Li and A. I. Mourikis, "Optimization-based estimator design for vision-aided inertial navigation: Supplemental materials," 2012, www.ee.ucr.edu/∼mli/SupplMaterialsRSS2012.pdf.

[26] R. Simpson, J. Cullip, and J. Revell, "The Cheddar Gorge data set," Tech. Rep., 2011.