A Resource-aware Vision-aided Inertial Navigation System for Wearable and Portable Computers

Dimitrios G. Kottas, Ryan C. DuToit, Ahmed Ahmed,
Chao X. Guo, Georgios Georgiou, Ruipeng Li, and Stergios I. Roumeliotis

***Abstract*— In this paper, we address the problem of deploying Vision-aided Inertial Navigation Systems (VINS) on resource-constrained platforms such as cell phones and wearable computers. In particular, we consider the case of a sliding-window extended Kalman filter (EKF)-based estimator and focus on optimizing its use of the available processing resources. This is achieved by first classifying visual observations based on their feature-track length and then assigning different portions of the CPU budget for processing subsets of the observations belonging to each class. Moreover, we introduce a processing strategy where "spare" CPU cycles are used for (re)-processing all or a subset of the observations corresponding to the same feature, across multiple, overlapping, sliding windows. This way, feature observations are used by the estimator more than once for improving the state estimates, while consistency is ensured by marginalizing each feature only once (i.e., when it moves outside the camera's field of view). The ability of the proposed feature classification and processing approach to adjust to the availability of processing resources is demonstrated experimentally on a Samsung S4 cell phone and on the Google Glass where VINS operates in real-time while occupying only half of the CPU cycles of one of the ARM processor's cores.**

I. Introduction and Related Work

Navigating in GPS-denied areas (e.g., spacecraft [1] or personal localization [2]) often requires combining visual observations from a camera, with inertial measurements, from an Inertial Measurement Unit (IMU) in what is known as a Vision-aided Inertial Navigation System. Existing approaches to VINS rely either on filtering or bundle-adjustment (BA) optimization methods. BA methods optimize over the entire trajectory of the sensor platform along with a map of environment, comprising all observed landmarks, which leads to processing and memory requirements that inevitably increase with time [3]. Filtering approaches can be divided into two categories: (i) Vision-aided Inertial Odometry (VIO) algorithms that optimize over the most recent camera poses [1], keeping a bounded computational cost, and (ii) EKF-SLAM approaches that maintain a map of the environment [4]. Existing VIO methods are able to exploit all information provided via feature tracks within their window, for estimating the motion of the platform. However, these methods need to wait for a feature track to reach its maximum length (ideally the whole length of the optimization window) before employing it in the correction of the platform's trajectory estimate. Furthermore, to the best of our knowledge, no rule exists for deciding, based on the available computational resources, *which* features should be processed at each time step and *when* they should contribute to the estimator's state and uncertainty estimates. Our work addresses these limitations through the following contributions:

- A modified Multi-State Constrained Kalman Filter (MSC-KF) framework which repetitively uses visual observations as they become available, for improving the filter's state estimate during multiple EKF updates, before they are marginalized and their information is used to reduce the estimator's uncertainty.
- A method for *classifying* visual measurements based on their span (i.e., the number of times that a feature is observed within a sequence of consecutive camera poses).
- A policy for deciding when and how to process a feature track given the current processing constraints.

In what follows, we provide a brief description of the key components comprising the proposed algorithm as well as its experimental validation on resource-constrained platforms.

II. Algorithm Description

In this section, we present the main modules of the proposed EKF-based VINS. First, we describe a modified MSC-KF, which allows the (re)processing of a feature track during different epochs of the sliding window. Second, we present the categorization of feature tracks that we employ for prioritizing the contribution of each feature to the filter's state estimate and covariance. Finally, we describe a measurement-selection policy for deciding, which measurements will be processed based on the available computational resources.

The state vector $\mathbf{x}$ comprises three parts, $\mathbf{x} = \begin{bmatrix}\mathbf{x}_{I_k}^T & \mathbf{x}_C^T & \mathbf{x}_F^T\end{bmatrix}^T$, corresponding to (i) the current IMU pose, $\mathbf{x}_{I_k}$, (ii) the set of M past camera poses, included in the filter's sliding window, $\mathbf{x}_C = \begin{bmatrix}\mathbf{x}_{C_1}^T \dots \mathbf{x}_{C_M}{}^T\end{bmatrix}^T$, and (iii) the mapped landmarks, currently included in the state vector, $\mathbf{x}_F = \begin{bmatrix}\mathbf{x}_{f_1}^T \dots \mathbf{x}_{f_L}^T\end{bmatrix}^T$.

A. Visual Observations in Sliding Window Estimators

Within a sliding window filtering framework, the same measurement is available for processing at different time-instants, during different epochs of the sliding window of camera poses. Following the traditional EKF framework, such a feature track can be employed only in a single update for correcting the state estimate *and* reducing its covariance. For existing VIO algorithms, a feature track is processed only once, when it has reached its maximum tracking length [1]. In practice, this will cause a depletion of features available for an EKF update, leading to low performance. In order to process any feature track spanning at least two poses of our sliding window, we introduce the method of *State-only* EKF updates. Specifically, among the different sliding window epochs, corresponding to different EKF updates, we choose only one for "absorbing" the feature track's information in the filter's covariance matrix (i.e., marginalization when the feature track is lost), while we are able to correct our

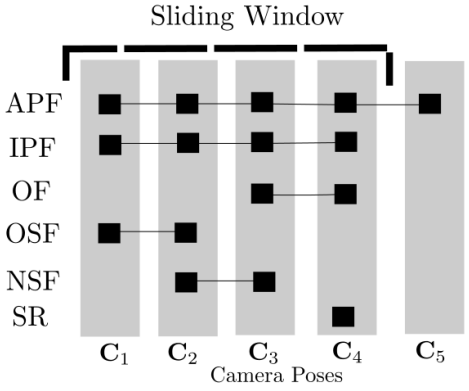

Fig. 1. Feature categories based on their span across the sliding window of camera poses.

state estimate during all state updates, up to that point. Due to space limitations, we describe the key idea that allows the processing of the same visual observation over multiple EKF updates. An EKF update step, processing m visual measurements, can be written as a single Gauss-Newton iteration, optimizing over x, the cost function [5],

$$c(\mathbf{x}) = c_p(\mathbf{x},\mathbf{P}) + \sum_{i=1}^{m} c_i(\mathbf{z}_i,\mathbf{x}) = ||\mathbf{x}-\hat{\mathbf{x}}||_{\mathbf{P}} + \sum_{i=1}^{m} ||\mathbf{z_i}-\mathbf{h}(\mathbf{x})||_{\mathbf{R}_i}$$

comprising of a prior term, c_p[1], and a set of quadratic terms c_i, corresponding to visual measurements $\mathbf{z_i}$, $i = 1 \dots m$. Upon employing a measurement $\mathbf{z_i}$ for updating the filter's covariance, the corresponding cost term c_i is absorbed into the filter's prior, represented by the cost term, c_p, of the next EKF update. In contrast by using a measurement $\mathbf{z_i}$ for updating *only* our state, the corresponding cost term c_i is exploited for improving our state estimate, without being absorbed into c_p. Hence the measurement $\mathbf{z_i}$, along with the corresponding cost term c_i, remain available for the next epoch of the sliding window filter and the next EKF update. As opposed to the feature depletion encountered in the regular MSC-KF framework, such a strategy leads to a plethora of visual measurements available for processing during a single EKF update, which presents two critical questions:

- At which one, among all tracking epochs, should a feature track both correct the filter's state estimate and reduce its uncertainty?
- Which features, should be processed such that the computational cost of the filter remains within the available resources?

B. Feature Tracks' Classification

Before using any visual observations, we classify all feature tracks within the current window of camera poses. Specifically, features are divided into 6 disjoint sets $\mathbb{F}_{SR}$, $\mathbb{F}_{APF}$, $\mathbb{F}_{IPF}$, $\mathbb{F}_{OSF}$, $\mathbb{F}_{OF}$, and $\mathbb{F}_{NSF}$ (see Fig. 1):

- SLAM re-observations (SRs) denoted by $\mathbb{F}_{SR}$, correspond to observations of features being mapped.
- Active Persistent Features (APFs) denoted by $\mathbb{F}_{APF}$, correspond to feature tracks that span the whole window of camera poses and are actively tracked beyond the size of the current sliding window. Such features have currently reached their maximum length. Hence they are the most informative regarding the motion of the camera poses, and they can also be initialized as new SLAM features such that they evolve to SRs.
- Inactive Persistent Features (IPFs) denoted by $\mathbb{F}_{IPF}$, correspond to feature tracks that span the whole window of camera poses, but are not tracked beyond the size of the current sliding window.
- Old Short Features (OSFs), denoted by $\mathbb{F}_{OSF}$, correspond to feature tracks that start from the oldest image but do not reach the newest (most recent) camera pose.
- Opportunistic Features (OFs) denoted by $\mathbb{F}_{OF}$, correspond to feature tracks that do not start from the oldest camera pose but are observed in the newest image. Such features have not yet reached their maximum potential tracking length, and may evolve to PFs.
- New Short Features (NSFs), denoted by $\mathbb{F}_{NSF}$, correspond to feature tracks that do not start from the oldest image and do not reach the newest image.

C. Processing Methods

During each epoch of the sliding window, feature tracks are assigned to 4 disjoint sets $\mathbb{P}_{SR}, \mathbb{P}_{SO}, \mathbb{P}_{SC}, \mathbb{P}_{SI}$ corresponding to different methods of processing within the same EKF update. Specifically, these methods correspond to:

- SLAM re-observations ($\mathbb{P}_{SR}$) Processing of measurements to mapped features, using the regular EKF SLAM measurement model. Clearly, features belonging to $\mathbb{F}_{SR}$ are the only ones that can reach this group.
- State & Covariance Update VIO Features ($\mathbb{P}_{SC}$) Camera measurements processed using the regular MSC-KF measurement model, contributing both to a state correction and a reduction of the estimator's uncertainty. As in the regular MSC-KF, features that have reached their maximum length across all sliding window epochs, are used for contributing a state correction as well as new information for the estimator, by reducing its covariance. Following the feature classification described previously, this corresponds to selecting features from $\mathbb{F}_{IPF}$ and $\mathbb{F}_{OSF}$.
- SLAM Initialization Features ($\mathbb{P}_{SI}$) Active persistent features that span beyond the length of the sliding window ($\mathbb{F}_{APF}$) should reach this set, since they have both reached their maximum length and can evolve into SR features in the future, upon their successful initialization as mapped features.
- State-only Update Features ($\mathbb{P}_{SO}$) Any of the features, belonging to the remaining groups, will contribute *only* to a state correction. Intentionally the filter's updated covariance will not reflect any information provided from these features, since they have either not reached their maximum potential (i.e., OFs) or because they will contribute to the filter's covariance, later on once a better

[1]Note that we consider that all IMU measurements, over the optimization window, have been integrated and incorporated into c_p

linearisation point has been determined for them (i.e., NSFs).

D. Processing Budget

In an ideal scenario that we have access to infinite computational resources, the sets of features to be processed (i.e., $\mathbb{P}_{SR}, \mathbb{P}_{SO}, \mathbb{P}_{SC}, \mathbb{P}_{SI}$) should be allowed to reach their maximum size, such that all available visual observations are employed in an EKF update. In many cases, however, VINS needs to be deployed on resource-constrained platforms that need to operate in real-time. Motivated by this, we bound the *maximum number of CPU cycles* allowed for each EKF update. We define a maximum CPU budget of allowed floating-point operations $\mathbb{B}_{MAX}$, corresponding to the 4 different types of processing methods:

$$\mathbb{B}_{MAX} = \mathbb{B}^{SR}_{MAX} + \mathbb{B}^{SC}_{MAX} + \mathbb{B}^{SI}_{MAX} + \mathbb{B}^{SO}_{MAX}.$$

For each family of measurements we determine the amount of CPU operations required for its processing, denoted by $\mathbb{B}$, as a function of the corresponding set $\mathbb{P}$:

$$\mathbb{B}_{SR}(\mathbb{P}) = |\mathbb{P}|^3,\ \mathbb{B}_{SC}(\mathbb{P}) = \sum_{\mathbf{f}\in\mathbb{P}} \ell(\mathbf{f})^3$$
$$\mathbb{B}_{SI}(\mathbb{P}) = \sum_{\mathbf{f}\in\mathbb{P}} \ell(\mathbf{f})^3 + |\mathbb{P}|^2,\ \mathbb{B}_{SO}(\mathbb{P}) = \sum_{\mathbf{f}\in\mathbb{P}} \ell(\mathbf{f})^3$$

where $|\mathbb{S}|$ denotes the cardinality of the set $\mathbb{S}$, and $\ell(\mathbf{f})$ the length of a feature track $\mathbf{f}$ within the estimator's window. For clarity, we omitted lower-order terms, depicting only the leading terms of the computational cost for each family of measurements. The proposed[2] estimation policy examines the available feature tracks and assigns them to one of the four available groups, after checking that the projected computational cost does not exceed the allocated CPU budget for this EKF update. Such a measurement selection policy is depicted for the simple case of re-observations of SLAM features ($\mathbb{P}_{SR}$) in Alg. 1.

Algorithm 1 Measurement Selection for SLAM features

$\mathbb{P}_{SR} \leftarrow \emptyset$
for $\mathbf{f} \in \mathbb{F}_{SR}$ **do**
  if $\mathbb{B}_{SR}(\mathbb{P}_{SR} \cup \mathbf{f}) \leq \mathbb{B}^{SR}_{MAX}$ **then**
    $\mathbb{P}_{SR} \leftarrow \mathbb{P}_{SR} \cup \mathbf{f}$
  end if
end for

III. Experimental Results

We validated the computational efficiency of the proposed algorithm over two resource-constrained navigation platforms. In the first case, the filter was running online on Samsung S4 (SGH-M919), which operates on Android 4.3, and is equipped with an IMU and a rolling shutter camera. It features a 1.9GHz quad core ARMv7 CPU with 2GB RAM. As it is depicted in Table. I, the filter, along with the image processing module, was able to operate *two times faster than real-time*, for relatively heavy configurations (i.e., a sliding window of 10 images, 20 SLAM features included in the state vector, with $|\mathbb{P}_{SC}| + |\mathbb{P}_{SI}| \leq 30,\ |\mathbb{P}_{SO}| \leq 50,\ |\mathbb{P}_{SR}| \leq 20$). Inertial measurements were sampled and propagated at 100Hz while camera images were acquired at 15Hz.

TABLE I
Single-Threaded Online Samsung S4 VINS
Total CPU time per second: $15\times10+5\times47+5\times18=475$ (ms)
Loop-closure error 0.5-1.5% of distance travelled

Module	Frequency (Hz)	Time (ms)
Harris Corner Extr.	5	18
KLT	15	10
EKF	5	47

TABLE II
Single-Threaded Offline VINS running on Google Glass
Total CPU time per second: $15\times20+5\times45+5\times5=550$ (ms)

Module	Frequency (Hz)	Time (ms)
Harris (scaled images)	5	5
KLT	15	20
EKF	5	45

Pushing the ability of our framework to tune its configuration based on the available computational resources, we tested our VINS implementation on a much more constrained platform, Google Glass. Google Glass operates on Android 4.0 and features a dual core Texas Instruments OMAP 4430 CPU, clocked at 1.2GHz with 682MB of RAM. Our preliminary results use the following configuration: Sliding window size of 7 images, 0 SLAM features, with $|\mathbb{P}_{SC}| \leq 20,\ |\mathbb{P}_{SO}| \leq 30$). As presented by the preliminary (offline) results at Table. II, the system was able to operate *almost two times faster than real-time*, albeit for a lighter configuration. All linear algebra operations where carried out using the Eigen C++ Library or pure C array operations. Image processing modules, such as the Harris Corner Extraction and the Lucas-Kanade (KLT) feature tracking, were implemented in ARM NEON assembly. All modules were wrapped in an object-oriented C++ framework, allowing easy organization of measurements, state estimates and corresponding uncertainties.

[2]Note that the distribution of resources ($\mathbb{B}_{MAX}$) between $\mathbb{B}^{SR}_{MAX}$, $\mathbb{B}^{SI}_{MAX}$, $\mathbb{B}^{SC}_{MAX}$, $\mathbb{B}^{SO}_{MAX}$, is predetermined based on the motion profile and the availability of lengthy feature tracks (e.g., more SLAM features are used when "flying" over the same region).

References

[1] A. I. Mourikis, N. Trawny, S. I. Roumeliotis, A. E. Johson, A. Ansar, and L. Matthies, "Vision-aided inertial navigation for spacecraft entry, descent, and landing," *IEEE Trans. on Robotics*, vol. 25, pp. 264–280, Apr. 2009.

[2] J. A. Hesch, D. G. Kottas, S. L. Bowman, and S. I. Roumeliotis, "Towards consistent vision-aided inertial navigation," in *Proc. of the 10th International Workshop on the Algorithmic Foundations of Robotics*, (Cambridge, Massachusetts), pp. 559–574, June 13–15 2012.

[3] E. D. Nerurkar, K. J. Wu, and S. I. Roumeliotis, "C-KLAM: Constrained Keyframe Localization and Mapping for long-term navigation," in *Workshop on Long-term Autonomy of the IEEE International Conference on Robotics and Automation*, (Karslruhe, Germany), May 10 2013.

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

[5] A. Jazwinski, "Stochastic processes and filtering theory," *Mathematics in science and engineering*, no. 64, 1970.