Vision-aided Inertial Navigation for Resource-constrained Systems

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*— In this paper we present a resource-adaptive framework for real-time vision-aided inertial navigation. Specifically, we focus on the problem of visual-inertial odometry (VIO), in which the objective is to track the motion of a mobile platform in an unknown environment. Our primary interest is navigation using miniature devices with limited computational resources, similar for example to a mobile phone. Our proposed estimation framework consists of two main components: (i) a hybrid EKF estimator that integrates two algorithms with complementary computational characteristics, namely a sliding-window EKF and EKF-based SLAM, and (ii) an adaptive image-processing module that adjusts the number of detected image features based on the availability of resources. By combining the hybrid EKF estimator, which optimally utilizes the feature measurements, with the adaptive image-processing algorithm, the proposed estimation architecture fully utilizes the system's computational resources. We present experimental results showing that the proposed estimation framework is capable of real-time processing of image and inertial data on the processor of a mobile phone.**

I. Introduction

In this paper we focus on the problem of high-precision real-time pose tracking, in unknown and GPS-denied environments. Our aim is to estimate the trajectory of a moving platform using inertial measurements and visual 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. As a result, 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 (VIO). We are specifically focusing on pose tracking for miniature, resource-constrained 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 real-time pose estimation a challenging task. The problem becomes particularly acute when visual sensing is used, for two main reasons. First, processing images to extract and match corner features can take a considerable amount of time. Even on desktop-grade CPUs, performing image processing in real time can be a difficult task unless highly optimized code is used. Second, most feature detection algorithms can typically detect and track hundreds of features in images of natural scenes. Processing all these measurements in an estimator in real time is a computationally demanding task.

To date, most pose estimation algorithms have been designed for and tested on computers with laptop- or desktop-grade CPUs. On these systems, which can be easily carried by medium- and large-sized mobile robots, the large amount of computational power available can simplify the task of estimator design. One can *propose* a given motion estimator and feature extraction method (e.g., EKF-based SLAM using Harris corners), and subsequently test whether the resulting system attains real-time performance. If real-time performance is achieved, no further action is typically necessary. However, the situation is very different in small-scale systems, where processing resources are severely limited, and battery life is a critical issue. First, in these systems it is harder to meet real-time constraints, due to the less capable CPUs available. Moreover, even if real-time performance is achieved, to prolong battery life one must make sure that the pose estimation algorithm requires as little computation as possible.

To address these challenges, in this paper we describe a resource-adaptive approach to VIO, consisting of two main components. The first component is a hybrid extended Kalman filter (EKF) estimator that integrates EKF-based SLAM with a sliding-window VIO estimator [1]. These two estimators process the same measurement information in different ways, and have complementary computational characteristics. The hybrid EKF adaptively chooses which of the two estimators should be used to process each of the available measurements, so as to minimize the required computation. As explained in Section III-B, the optimal choice of algorithm to process each individual feature depends on the distribution of the feature-track lengths of all features tracked in the images. Since this distribution is impossible to predict in advance (it depends on the environment, the camera motion, as well as the feature tracker used), we *learn* it from the image sequence. Using this information, the optimal strategy for processing the feature measurements can be computed by solving, in real time, a one-variable optimization problem.

The hybrid EKF estimator is able to extract the maximum possible accuracy from the available measurements, at the minimum possible cost. However, when the number of available feature measurements is too large, it may still be impossible for the algorithm to operate in real time. For this reason, the hybrid EKF is coupled with an adaptive feature extraction algorithm, in which the number of detected features is actively controlled to meet real-time constraints. This is accomplished by measuring the estimator's CPU usage and using it to adjust the sensitivity threshold of feature extraction. This is a general approach, and can be used with any feature detection method (e.g., SIFT [2], Harris corners [3], Shi-Tomasi features [4], or FAST features [5]). For any choice of method, it will adapt to the computational requirements of the feature extractor, and automatically de-

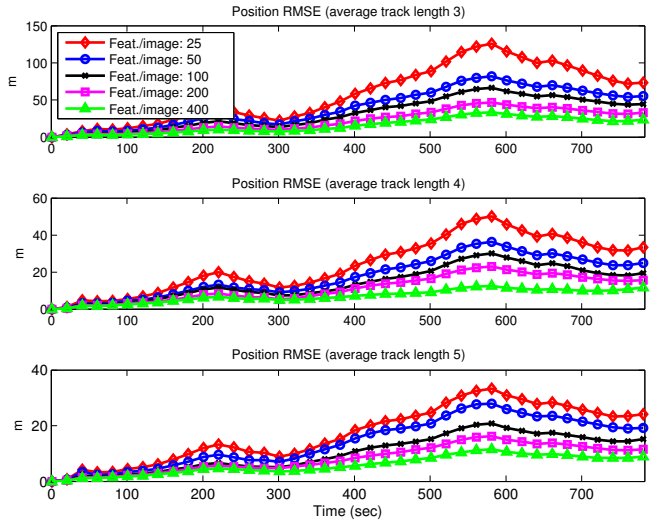


Fig. 1: Position accuracy using VIO with varying numbers of features and average track lengths. Note the different scale of the y axis in the three subplots.

termine the maximum number of features that can be processed in real time. When a more computationally efficient feature detector is employed, more features can be used, and thus higher precision is attained. Our testing showed that (as expected) the most computationally efficient algorithm out of those mentioned above is FAST, and its use leads to the highest estimation precision.

To test the computational efficiency of the proposed resource-adaptive VIO estimator, we have used it to carry out pose estimation on the processor of a mobile phone (Samsung Galaxy S2). Our results show that the estimator can easily attain real-time operation, being able to process images containing an average of 250 FAST features at 10 Hz using a single processor core, and yielding position error of 0.55% of the travelled distance, in a 5.8-km-long trajectory.

II. Related Work

Methods that seek to optimize, in some sense, the computational efficiency of pose estimation can be categorized broadly as follows. First, a number of approaches exist that try to re-organize the computations in an estimator so as to minimize the total processing required. Typical methods involve decomposing the computations into smaller parts, and selectively carrying out the necessary computations at each time instant (see e.g., [6], [7]). 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 VIO, however, where the state vector contains only the actively tracked features and no loop-closure is considered, these methods are not applicable. The methods discussed above usually employ exact reformulations of the estimator equations, which lead to no loss of accuracy. On the other hand, several methods exist that employ approximations (e.g., [8]–[10] and references therein), to reduce the required computations. In contrast to these methods, which trade-off information for efficiency, the hybrid estimator described in Section III involves no approximations, other than the inaccuracies due to the EKF's linearization.

The computational cost of localization can alternatively be lowered by reducing the number of feature measurements processed. For instance, several pose estimation algorithms only track features for two or three consecutive frames, use these tracks to estimate displacement, and fuse the displacement estimates with the inertial measurements, if available (see e.g., [11], [12]). So-called *keyframe* methods only process measurements from a subset of spatially distributed camera poses, and discard all other measurements (e.g., [13], [14]). Finally, one can select a subset of all available features to process, for example by identifying the most valuable ones in terms of pose information (e.g., [15], [16]). While all these are reasonable ways to lower the computational cost of an estimator, it should be clear that discarding measurements results in loss of information, and thus reduced accuracy.

To demonstrate the effect of discarding feature measurements on the localization accuracy, we briefly present the results of a Monte-Carlo simulation test. In this test, a camera/IMU system moves in a feature-rich environment and the measurements are processed for state estimation using the algorithm in [17]. The trajectory of the camera and the features' spatial distribution are designed to emulate the characteristics of a vehicle moving in an urban environment. Fig. 1 shows the average position-estimation accuracy (averaged over 50 Monte-Carlo trials), with varying average number of features tracked per image and varying average feature-track lengths. This figure shows that, as expected, using more features, or features with longer tracks, leads to increased accuracy. What is important, however, is to observe that even when a large number of features is tracked, increasing the number of features can still result in significant improvement (e.g., increasing the average number of features per image from 200 to 400 leads to a 37.1% reduction in RMSE, when the average feature-track length is 4). Similar comments can be made regarding the average track length.

These results show that for high-precision VIO it is essential to use as many features as possible. To achieve this, in this paper we employ a hybrid EKF estimator that adaptively determines the optimal way to process the feature measurements, so that the algorithm's computational requirements are minimized. This estimator is coupled with a feature extraction method that adaptively determines the maximum number of features that can be processed in real time. As shown in the results of Section V the combination of the adaptive estimator back-end with the adaptive image-processing front-end can result in real-time, high-precision VIO, even on the less-capable processor of a mobile phone. We note that in recent years, there has been significant interest in the augmented-reality research community in developing methods for real-time tracking using mobile phones (see [18] for a review). These methods typically require known 3D positions of the features being tracked. In contrast, in our work we use naturally occurring features with unknown positions, and fuse their observations with inertial measurements. We use the mobile phone as a convenient

resource-constrained computing platform, but our methods can be used in any computing system.

III. The hybrid MSCKF/SLAM Algorithm

We now present the hybrid EKF estimator, which adaptively determines the optimal way to process the available feature measurements to minimize computation [1]. This estimator integrates two different approaches for processing the feature measurements: the multi-state-constraint Kalman filter (MSCKF) [17], [19] and EKF-based SLAM. These two algorithms use the feature measurements' information in very different ways. On the one hand the MSCKF explicitly marginalizes out the features, and maintains a state vector comprising a sliding window of poses. On the other hand, EKF-SLAM marginalizes out all but the current pose, and maintains a state vector comprising the visible landmarks (since we are interested in VIO, and not loop closing, all landmarks that fall out of the field of view are discarded).

Both the MSCKF and EKF-SLAM use exactly the same information; if the measurement models were linear, both methods would yield exactly the same result, equal to the MAP estimate of the IMU pose [17]. The methods differ in the assumptions they make about the feature-error pdf (more on this in Section III-B), and crucially, in terms of their computational properties. 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). On the other hand, the MSCKF has computational cost that scales linearly in the number of features, but cubically in the length of the feature tracks [19]. Therefore, if many features are tracked in a small number of frames the MSCKF approach is preferable, but if few features are tracked over long image sequences, EKF-SLAM results in lower computational cost.

We thus see that EKF-SLAM and the MSCKF algorithm are complementary, with each being superior in different circumstances. By integrating both algorithms in a single hybrid filter, we are able to decide, in real time, which algorithm will be used to process each of the available features in order to minimize the computational cost. To improve the estimation accuracy and consistency, the hybrid filter, employs fixed linearization points for each state, in both the MSCKF and EKF-SLAM Jacobians. This ensures the correct observability properties of the linearized model, as discussed in [17].

A. Description of the hybrid MSCKF/SLAM algorithm

We here briefly describe the hybrid estimator (see Algorithm 1), and the interested reader is referred to [1] for more details. The state vector of the hybrid estimator at time-step k contains the current IMU state, a sliding window of m camera poses, and s features:

$$\mathbf{x}_k = \begin{bmatrix} \mathbf{x}_{I_k}^T & \mathbf{x}_{C_1}^T & \cdots & \mathbf{x}_{C_m}^T & \mathbf{f}_1^T & \cdots & \mathbf{f}_s^T \end{bmatrix}^T \quad (1)$$

where $\mathbf{x}_{I_k}$ is the current IMU state, $\mathbf{x}_{C_j}$, $j = 1 \dots m$ are the camera poses (positions and orientations) at the times the last m images were recorded, and $\mathbf{f}_i, i = 1 \dots s$ are the features, represented using an inverse-depth parameterization. The IMU state, $\mathbf{x}_I$, contains the IMU pose, velocity, and the gyroscope and accelerometer biases.

Algorithm 1 Hybrid MSCKF/SLAM algorithm

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

Update: When camera measurements become available:

- Augment the state vector with the latest camera pose, and begin image processing.
- For each of the features to be processed in the MSCKF (features whose tracks are complete after m or fewer images), do the following
  - Triangulate the feature using all its observations.
  - Compute the feature-reprojection residuals, and project the residual vector onto the left nullspace of the feature Jacobian matrix, to obtain the MSCKF residual and Jacobian.
  - Perform a Mahalanobis gating test.
- Stack the residuals and Jacobians of all the MSCKF features that passed the gating test, and form the combined MSCKF residual vector and the Jacobian matrix.
- For the SLAM features included in the state vector, compute the residuals and measurement Jacobians.
- Perform an EKF update, using all the SLAM-feature residuals and the combined MSCKF residual.
- Initialize into the state vector features that are still actively tracked after m images.

State Management:

- Remove SLAM features that are no longer tracked.
- 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.

When an IMU measurement is received, it is used to propagate the filter state and covariance. For processing the feature tracks, two possibilities exist: if a feature's track is lost after m or fewer images, it is used for an MSCKF update, while if it still being tracked after m frames, it is initialized in the state vector and used for SLAM. We have shown that, for any given m, this is the optimal feature-processing policy in terms of computation. At each time step, the hybrid filter processes a number of features by the MSCKF approach, and others with the EKF-SLAM approach. For each of these types of features, the appropriate residuals and Jacobian matrices are computed, and a Mahalanobis gating test is performed. All the features that pass the gating test are then employed for an EKF update. At the end of this process, SLAM features that are no longer visible, and older camera poses, all of whose features have been processed, are removed.

B. Optimizing the computational cost of the hybrid EKF

As explained, the motivation for integrating the MSCKF and EKF-SLAM formulations is to leverage the computational advantages of both methods, by processing each of the available features with the best suited algorithm. Since for a given sliding window size, m, the optimal policy of processing features is known (i.e., process feature tracks of

length up to m by the MSCKF, and initialize longer-tracked features in the state vector), the only design parameter that determines the computation cost of the filter is m.

To demonstrate how m affects the performance of the method, Fig. 2 shows the timing and position accuracy attained for different values of m. To obtain these plots, we generated simulated data with properties (trajectory, sensor noise characteristics, feature number, spatial distribution, and track lengths) similar to those observed in a real-world dataset. These data were processed on a Samsung Galaxy S2 phone using a single-threaded C++ implementation, in which the value of m was a parameter. The blue lines in Fig. 2 show the timing and accuracy results, averaged over 50 Monte-Carlo trials for each choice of m. We point out that when m is very large, all features are processed using the MSCKF, therefore the right-most parts of the curves show the performance of the "pure" MSCKF. Similarly, the left-most parts of the curves show the performance of pure EKF-SLAM.

Two main conclusions can be drawn from Fig. 2. First, it becomes clear that by appropriately choosing m we can obtain an algorithm that is faster than both the MSCKF and EKF-SLAM, by a significant margin. Second, we note that the MSCKF yields higher accuracy than EKF-SLAM, since 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). By combining the MSCKF with EKF-SLAM some accuracy may be lost, as the errors for those features included in the state vector are now assumed to follow a Gaussian distribution. However, we can see that if the size of the sliding window increases above a moderate value (e.g., 10 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.

The results of Fig. 2 show that an optimal value of m exists, which minimizes the computation of the hybrid filter. To determine this optimal value, it is necessary to have an expression for the filter's computational cost, as a function of m. In [1] it is shown that an *analytical* expression for the cost (flop count) of the filter can be obtained, if the distribution of the feature-track lengths is known. Therefore, we collect statistics during the filter's operation that allow us to *learn* the probability mass function (pmf) of the feature-track lengths. Using this learned pmf, the optimization of the analytical cost function can be performed very efficiently by simple exhaustive search (note that this is a one-variable optimization problem, in a very limited domain). Since the statistical properties of the feature tracks will change over time, we perform the learning of the pmf as well as the selection of the optimal threshold in consecutive time windows spanning a few seconds (15 sec in our implementation). In Fig. 2 the red dashed line shows the time requirements of the hybrid filter when the size of the sliding window is adaptively controlled via this learning and optimization approach. We can observe that (as expected) the adaptive algorithm is faster than any choice of a fixed m.

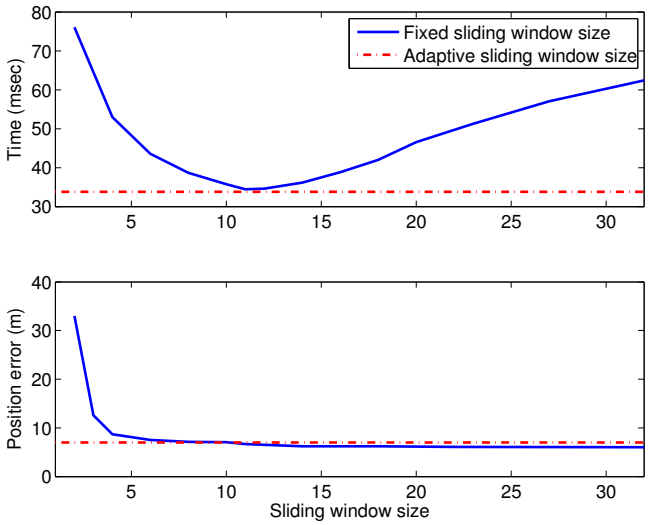


Fig. 2: Monte-Carlo simulation results: Timing performance and RMS position accuracy of the hybrid filter, for changing values of m. Timing measured on a Samsung Galaxy S2 mobile phone.

IV. Image Processing Module

In this section, we present the image-processing module of our proposed VIO estimator. Specifically, we first discuss the choice of possible feature extraction algorithms, and then describe the controller used to adaptively determine the number of features detected.

A. Feature extraction algorithms

Several feature extraction algorithms have been proposed to date. For this paper, we have evaluated the following ones: (i) the FAST detector [5], (ii) the Shi-Tomasi feature detection algorithm [4], (iii) the Harris corner detector [3], and (iv) the SIFT keypoint detection algorithm [2]. Besides feature detection accuracy and repeatability, the most important characteristic of the feature detection algorithm, in the context of real-time VIO, is computational efficiency. Therefore, we first evaluated the time requirements of each of the methods on the Samsung Galaxy S2 mobile phone on which the experimental testing was done. Table I shows the average time needed by the different feature extraction methods per image. In this test the images are of size 640×480 pixels, and for a fair comparison between different algorithms, their thresholds and parameters are adjusted so that each extracts approximately 500 features. All algorithms (except Opt-ST) were implemented using the built-in OpenCV functions. For efficiency, we used only one octave for the SIFT algorithm. In Table I, ST refers to the Shi-Tomasi feature detector, while Opt-ST is an optimized version of this detector that we developed as described in the Appendix.

In Table I, we observe that SIFT is significantly slower than the other algorithms, and it is not eligible for real-

TABLE I: Average feature extraction time for the different algorithms

	FAST	Harris	ST	Opt-ST	SIFT
Time (msec)	11.3	69.9	72.7	43.1	1015.0

time navigation at a high frame rate[1]. On the other hand, FAST is the fastest one (as expected, based on the results of [5]), while the optimized version of the ST detector is approximately 40% faster than the original algorithm. From these results we conclude that for real-time image processing (e.g., for being able to process images at 10 Hz at least), the eligible algorithms are FAST, Opt-ST, ST, and Harris. Even though the features detected in the images by these algorithms are not exactly the same, our tests have shown that when they are used in conjunction with the hybrid filter, the estimation accuracy does not exhibit significant differences (if the same number of features is used).

For feature matching we employ normalized cross-correlation, since it is computationally efficient. When visual navigation is carried out at a high frame rate, the feature templates' scaling and perspective deformation between consecutive images are not significant. Thus, costly feature matching algorithms based on invariant feature detectors (e.g., those provided by SIFT) are not considered. Our tests have shown that the features detected by FAST, Harris, and Opt-ST result in almost identical feature-track distributions (differences are in the order of 3%). This suggests that, at least for the image sequences in our experiments, the properties of the feature tracks are determined mostly by the scene and trajectory, rather than by the method used for feature detection (similar conclusions were reached in [20]).

B. Adaptive threshold selection

As discussed in Section III-B, the hybrid MSCKF/SLAM filter is able to optimize the computational time given the available feature characteristics. Ideally, we would like to be able to use all features that our extraction algorithm can detect, as increasing the number of features increases accuracy (see Fig. 1). However, if thousands of features are tracked per image, the hybrid filter will eventually become too slow for real-time pose estimation. Thus, in this section, we describe a control approach that ensures that the number of features processed is the maximum possible for a given environment, under the real-time constraints. This approach is applicable to any of the feature extraction algorithms described in the previous section.

Our approach employs a proportional controller (p-controller) to adaptively control the sensitivity threshold used by the feature extraction algorithm. The input to this controller is the average time, t_a, needed by the entire estimation process (image processing, feature matching, and filter update) in the last N timesteps. The controller is expressed as:

$$\Delta T_c = K_p(t_a - t_{\text{target}}) \quad (2)$$

[1]We note that the SIFT algorithm can be optimized, and it has been used for real time feature extraction on a mobile phone [18]. However, since scale invariance (a key advantage of SIFT over the other algorithms considered) is not required in our application, this was not pursued in our work.

Fig. 3: Sample images from the dataset used for testing

where t_{target} is the desired time for estimation at each step, K_p is the control gain, and ΔT_c is the change in the image extraction threshold. Intuitively, if the time needed by the estimation process is lower than t_{target}, the threshold can be lowered, which will result in more features being detected. In our implementation, we do not allow the threshold to fall below a minimum value, under which the detected features are due to image noise. Moreover, we choose t_{target} to be slightly smaller than the image period, to allow for variations in the runtime due to the inherent randomness.

By combining the hybrid MSCKF/SLAM estimator, which optimally utilizes the feature measurements, with the adaptive image-processing algorithm, which at each time instant detects the maximum number of features that can be processed in real time, the proposed estimator fully utilizes the system's computational resources. Clearly, a more efficient feature extraction algorithm is preferable, as it will allow more time to be allocated to the estimator, which in turn means that a higher number of features will be processed, and the accuracy will be increased. Therefore, out of the algorithms discussed in the preceding section, the FAST feature detector is better suited for real-time VIO. In the next section, we present experimental results demonstrating the performance of the system with real-world data.

V. Real-world Experiment

To evaluate our proposed VIO estimation architecture, we tested its performance and compared it against alternative approaches, by processing data on a Samsung Galaxy S2 mobile phone. To be able to test different methods on the same data, for these tests we recorded a dataset using a camera/IMU platform mounted on top of a car, and processed the data off-line. The sensors used for data recording consist of an Inertial Science ISIS IMU, a PointGrey Bumblebee2 stereo pair (only a single camera's images are used), and a Xsens MTi-G unit (used to record ground truth only). The IMU provides measurements at 100 Hz, while the camera images are stored at 10 Hz. The vehicle trajectory is approximately 5.78 km long, and a total of 4825 images are recorded. Sample images from the dataset are shown in Fig. 3.

In our testing, we compared the following four estimator/feature extraction combinations: 1) the MSCKF filter with FAST features, 2) the EKF-SLAM filter with FAST features, 3) the hybrid MSCKF/SLAM filter with FAST features, and 4) the hybrid MSCKF/SLAM filter with Opt-ST features. Since the Harris "cornerness" function is an approximation to that computed by ST, and its implementation is slower than the Opt-ST algorithm, we do not consider this feature detector in our tests. For all the above algorithms, we

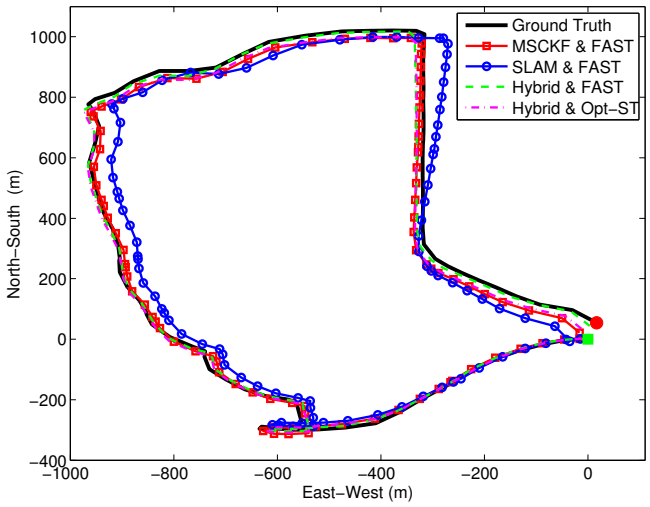


Fig. 4: Trajectory estimates computed by the different VIO methods, and GPS/INS ground truth. The green square is the starting point of the trajectory, and the red circle is the ending point.

TABLE II: Statistics for different filter and feature detector combinations

Filter	Features	Avg. feat. number	RMS Position error (m)
MSCKF	FAST	76.7	19.85
SLAM	FAST	61.5	34.52
Hybrid	FAST	250.1	11.57
Hybrid	Opt-ST	176.6	18.84

employed the adaptive feature-threshold selection algorithm described in Section IV-B to ensure that they all utilize the same amount of CPU resources. Since the image sampling period is 100 msec, we set $t_{\text{target}} = 80$ msec in the controller. All implementations are single-threaded in C++, ported to Android using the Android NDK.

Fig. 4 shows the vehicle trajectory estimates computed by the different algorithms, as well as the ground truth reported by the GPS/INS unit. From this figure, we observe that the proposed hybrid filter with the FAST feature detector, and the hybrid filter with the Opt-ST detector obtain more accurate trajectory estimates than the other approaches tested here. To provide a more detailed analysis of the results, in Table II we show the average number of features per-frame that can be used in each method within the time allotted, as well as the average position errors for the duration of the trajectory.

Several interesting observations can be made based on these results. First, note that the average number of features that each algorithm can process varies by a factor of four from the lowest one (EKF SLAM with FAST) to the highest one (hybrid filter with FAST). The number of features processed has direct implications on the accuracy attained by the different methods. For example, for the two approaches that are based on the hybrid filter, the accuracy increases with the number of features, corroborating the results of Fig. 1. It is worth pointing out that this result occurs due to the adaptive nature of the proposed architecture: when a more computationally efficient feature extraction method is used, the CPU time that is "freed up" is automatically allocated to the estimator. As a result, the estimator can now process a larger number of features, and obtain higher accuracy.

The benefits of the adaptive nature of the proposed estimation architecture are also seen in the comparison of the performance between the MSCKF and the hybrid EKF. As shown in Fig. 2, when the filters process the *same data*, the MSCKF is slightly more accurate. However, when the two algorithms are allocated the same amount of *CPU time* the hybrid filter is clearly preferable, as it is able to optimally utilize the limited amount of resources available, and process more features. Note that in this experiment, the hybrid filter with the FAST feature extractor is able to attain worst-case position errors of just 0.55% of the travelled distance, even though it is implemented on a resource-constrained mobile device.

The results of Fig. 4 and Table II demonstrate that the combination of the hybrid MSCKF/SLAM estimator with the FAST feature detector outperforms the other methods in terms of accuracy. For this estimator, in Fig. 5a we plot the size of the sliding window, m, in the hybrid filter for the duration of the experiment. This figure demonstrates that, due to the changing properties of the feature number and the feature tracks' distribution, the optimal value varies over time, justifying the need for periodic re-optimization.

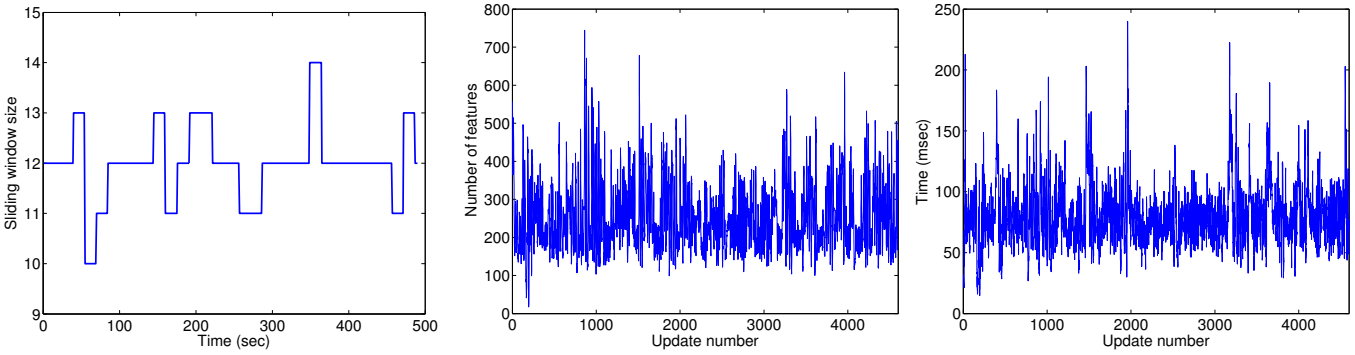
Fig. 5b shows the number of features processed at every timestep by the MSKCF/SLAM filter using the FAST feature detector, while Fig. 5c shows the time needed for each image update (combined time for image extraction, matching, and EKF update). Although the run time sometimes exceeds the image period (100 ms), the adaptation performed via the controller described in Section IV-B rapidly reduces it within a few timesteps. As a result, the measured average time for updates is 83 msec in this dataset. Note that, if it was required that no update take more than 100 msec, that could be easily implemented by either choosing a more conservative t_{target} in the controller, or by simply ignoring certain features in the EKF update.

VI. Conclusion

In this work we presented a framework for real-time visual-inertial odometry on resource-constrained devices. The proposed estimation architecture couples two components, each of which adaptively determines the best possible way to utilize the available resources. On one hand, we employ a hybrid EKF estimator that integrates a sliding-window EKF filter with EKF-based SLAM, and determines, in real time, which of the two approaches is best suited to process each of the available features. On the other hand, we designed an adaptive feature-extraction algorithm, which controls the number of features detected, depending on the available computing resources and the characteristics of the environment. Our experimental results demonstrate that the proposed approach is capable of high-precision VIO in real time, even when the data is processed by the less-capable processor of a mobile phone.

Acknowledgments

This work was supported by the National Science Foundation (grant no. IIS-1117957), the UC Riverside Bourns



(a) (b) (c)

Fig. 5: Results from the hybrid MSCKF/SLAM filter with FAST features: (a) The size of the sliding window chosen by the hybrid filter during the experiment. (b) Number of features extracted per image. (c) Combined per-update time for the filter and image processing.

College of Engineering, and the Hellman Family Foundation.

APPENDIX

We here describe the optimized Shi-Tomasi (Opt-ST) feature extraction algorithm. Recall that the ST algorithm forms the following matrix for each pixel:

$$\mathbf{M} = \begin{bmatrix} \sum I_x^2 & \sum I_x I_y \\ \sum I_x I_y & \sum I_y^2 \end{bmatrix} \quad (3)$$

where I_x and I_y denote the horizontal and vertical image derivatives, and summation occurs in a window centered at the pixel. The "cornerness" measure used by the ST algorithm is the minimum eigenvalue, $\lambda_{\min}$, of the matrix $\mathbf{M}$, which is quite computationally expensive to compute. The points that are local maxima and have value above a pre-specificed threshold T_c are chosen as features. Our optimized version of the algorithm does not compute $\lambda_{\min}$ at each pixel. To see which pixels can be ignored, we note that for any unit vector $\mathbf{v}$, the term $\mathbf{v}^T\mathbf{M}\mathbf{v}$ is an upper bound to $\lambda_{\min}$. Therefore we can first evaluate the easily-computed bounds $\mathbf{v}^T\mathbf{M}\mathbf{v}$ for $\mathbf{v} = [0 \;\; 1]^T$, $\mathbf{v} = [1 \;\; 0]^T$, $\mathbf{v} = [\sqrt{2}/2 \;\; \sqrt{2}/2]^T$ and $\mathbf{v} = [-\sqrt{2}/2 \;\; \sqrt{2}/2]^T$, and if all these bounds are above T_c, only then do we compute $\lambda_{\min}$. In this way, we can typically eliminate about 95% of an image's pixels, and the computational cost is significantly decreased.

REFERENCES

[1] M. Li and A. I. Mourikis, "Optimization-based estimator design for vision-aided inertial navigation," in *Proceedings of Robotics: Science and Systems*, Sydney, Australia, Jul. 2012.

[2] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *International Journal of Computer Vision*, vol. 260, no. 2, pp. 91–110, Nov. 2004.

[3] C. Harris and M. Stephens, "A combined corner and edge detector," in *Proceedings of the 4th Alvey Vision Conference*, Manchester, UK, Aug. 31 - Sep. 2 1988, pp. 147–151.

[4] J. Shi and C. Tomasi, "Good features to track," in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, Seattle, WA, June 1994, pp. 593–600.

[5] E. Rosten, R. Porter, and T. Drummond, "Faster and better: a machine learning approach to corner detection," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 32, no. 1, pp. 105–119, 2010.

[6] 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.

[7] 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.

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

[9] 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.

[10] 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.

[11] 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.

[12] 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. 4326–4333.

[13] 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.

[14] G. Klein and D. Murray, "Parallel tracking and mapping for small AR workspaces," in *The International Symposium on Mixed and Augmented Reality*, Nara, Japan, Nov. 2007, pp. 225–234.

[15] 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.

[16] 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, July 2002.

[17] 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.

[18] D. Wagner, G. Reitmayr, A. Mulloni, T. Drummond, and D. Schmalstieg, "Real-time detection and tracking for augmented reality on mobile phones," *IEEE Transactions on Visualization and Computer Graphics*, vol. 16, no. 3, pp. 355–368, May 2010.

[19] 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.

[20] J. Klippenstein and H. Zhang, "Performance evaluation of visual SLAM using several feature extractors," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, St. Louis, MO, Oct. 2009, pp. 1574–1581.