Decoupled Representation of the Error and Trajectory Estimates for Efficient Pose Estimation

Xing Zheng, Mingyang Li, and Anastasios I. Mourikis
Dept. of Electrical and Computer Engineering, University of California, Riverside
E-mail: {xzheng|mli|mourikis}@ece.ucr.edu

***Abstract*—In this paper we present a novel approach for the parameterization of the trajectory of a moving platform, which facilitates the development of real-time pose-estimation methods. The key idea of the proposed approach is the *decoupling* of the parameterization of the trajectory *estimate* from the parameterization of the *error* in this estimate. Specifically, we represent the trajectory estimate as usual, via a set of pose states, each associated with a sensor reading (e.g., a laser scan or an image). The novelty of our approach lies in the representation of the estimation errors, for which we employ B-splines. This decoupled formulation, which we term Decoupled Estimate-Error Parameterization (DEEP) offers two key advantages. First, the use of a pose-based representation of the trajectory allows us to represent arbitrarily complex trajectories. Second, the use of B-splines for error representation allows us to *control the computational complexity of an estimator*, by selecting the density of the knots of the B-spline. We empirically demonstrate that, in the problem of visual-inertial localization, the DEEP formulation leads to substantial computational gains, while incurring only a small loss of estimation performance.**

I. Introduction

In application domains such as mobile robotics, unmanned aerial vehicles, and personal localization, the ability to accurately estimate the position and orientation of a moving platform in real time is crucial. As a result, substantial research efforts have focused on the development of accurate and computationally efficient localization algorithms. One of the key challenges these algorithms seek to address is the high dimensionality of the estimation problem at hand, which is caused by (i) the large number of variables (e.g., feature points) needed to model real-world environments, and (ii) the number of variables (i.e., platform poses) required to represent trajectories of long duration. In this work, we focus on the latter issue, and propose a general method that can significantly reduce computational cost, while incurring only a small loss of accuracy.

We start by noting that the vast majority of practical localization algorithms are *linearization-based* methods, which typically rely on a variation of an extended Kalman filter (EKF), or iterative minimization. In these methods the computational cost is a function of the dimension of the *error-state* vector. For instance, this dimension defines the size of the state-covariance matrix in an EKF, or the size of the Hessian in Newton-minimization methods. In current practice, the dimension of the estimator's error-state is directly determined by the number of poses in the trajectory, since one error-state vector (consisting, for instance, of the errors in position and orientation) is associated with each pose estimate. By contrast, in this work, we propose a new paradigm, where the number of poses included in the state vector, and the dimension of the estimator's error-state are *decoupled*.

To describe the main idea of the method, we point out that any estimator that employs linearization relies on the computation of (i) measurement residuals, and (ii) linearized expressions showing the dependence of the residuals on the estimation errors. The first step involves the state *estimates*, while the second the state *errors*. The key insight here is that we may use *different representations* for each of these. Specifically, the state of a system at a certain time instant, t, is expressed as $\mathbf{x}(t) = \hat{\mathbf{x}}(t) + \tilde{\mathbf{x}}(t)$, where $\hat{\mathbf{x}}(t)$ is the state estimate, and $\tilde{\mathbf{x}}(t)$ is the error in this estimate. In order to represent the estimates, we use the "traditional" representation, where one pose vector (e.g., consisting of position and orientation) is maintained for each time instant at which measurements are available. This representation is preferable, as it makes no assumptions on the form of the trajectory. On the other hand, for the error-state $\tilde{\mathbf{x}}(t)$, we employ a continuous-time representation using temporal B-spline functions. Since the estimation errors are expressed as a function of the B-spline control points, this in turn means that we can express all the estimator Jacobians as a function of these parameters.

The key result that we demonstrate in this paper is that, due to the nature of the estimation errors, a *low-dimensional* B-spline representation suffices in order to describe the errors well. Therefore, the estimator equations can be expressed in terms of a small number of B-spline control points (which now constitute the error-state vector of the estimator), leading to a reduction in computational cost. We term the proposed approach, in which a different representation is used for the trajectory estimates and their errors, Decoupled Estimate-Error Parameterization (DEEP).

The proposed formulation is a general one and can be employed in several classes of estimation problems. One of its key advantages is that it results in algorithms that can easily adapt to the availability of computational resources in a system. By reducing the dimension of the representation used for the error state (e.g., by placing the B-spline knots further apart in time), we can reduce the computational cost of the estimator, while incurring a penalty in terms of estimation accuracy (which, as shown in this paper, is only modest). Thus, the proposed formulation can allows us to easily explore the tradeoff between accuracy and computational cost, to select

the best option for a given application.

In order to demonstrate the DEEP formulation in a practical localization problem, we here present the design of an estimator for 3D localization using visual and inertial measurements. We start with one existing method in visual-inertial localization, namely the multi-state-constraint Kalman filter 2.0 (MSCKF 2.0) [17], and re-formulate it using the proposed DEEP framework. Through both Monte-Carlo simulations, as well as real-world experiments, we show that this yields substantial gains in computational efficiency, but only small loss of accuracy. In our simulation tests, for example, the approach is shown to reduce the computational cost (measured in terms of the number of floating-point operations) by 89%, while increasing the estimation errors by a mere 1%. Importantly, we demonstrate that the approach leads to a graceful degradation of performance, as the dimensionality of the error representation is reduced.

II. Related Work

Prior work on the topic of reducing the computational complexity of localization and mapping is extensive, and any attempt at presenting an overview within the limited space available would be necessarily incomplete. At a high level, most computation-reduction methods for EKF-based algorithms propose either exact or approximate reformulations of the estimator equations to reduce the computational cost of updating the filter's state-covariance matrix (see, e.g., [21, 11, 24, 8] and references therein). On the other hand, to reduce the computational cost of graph-based methods, key approaches have included exploiting the sparsity of the graph, improving the computational characteristics of the minimization algorithm, and focusing computation on only the relevant parts of the graph (e.g., [13, 6, 12, 16]). By contrast, we here focus on reducing dimensionality via changing the representation of the error-state, which is conceptually different. The DEEP formulation can, in principle, be employed with *any* linearization-based method which maintains estimates of a platform's trajectory, including the ones mentioned above.

Our approach is more closely related to methods which seek to reduce the number of poses included in the state vector in localization. This is typically accomplished either by simply using measurements less frequently, or by intelligently selecting only a subset of *keyframes* to include in the estimator [14, 22], or by pruning an already existing pose graph [15]. Our proposed approach differs in a key aspect: we do *not* aim at reducing the number of poses included in the estimated state vector. Instead, we decouple the representation of the trajectory (which is still represented as a set of poses) from the representation of the trajectory errors (which are represented by B-splines), and aim at *reducing the dimensionality of the error state only*. In our approach, the measurements recorded from all poses are processed, using a reduced-dimension representation for the pose errors. As a result, we avoid the loss of information that occurs when a number of poses and their associated measurements are discarded from the estimator.

The representation of the error states in localization has been studied in a number of previous works, which focus on addressing the fact that (most) orientation representations involve differentiable manifolds [10, 27, 9, 28]. However, in all these works, there exists a one-to-one correspondence between the number of state estimates and error states, and thus these approaches are not closely related to our work. The same holds for works where new state parameterizations are proposed (see, e.g. [5, 19]), resulting in a subsequent change in the representation of the state errors.

The approaches most closely related to ours are those that employ temporal basis functions to represent the trajectory in continuous time. Earlier works using spline-based representations of the trajectory can be found in [3, 2, 4]. The methodology was formalized in [7], and has been employed for pose estimation and calibration of rolling-shutter cameras in [23], and for visual-inertial SLAM in [18]. This continuous-time representation of the trajectory offers the advantage that, by increasing the number of basis functions (and thus the computational cost), one can model arbitrarily complex trajectories. The key difference between the DEEP formulation and the approach of the above works is that in the latter, there exists a one-to-one correspondence between the number of states in the representation (e.g., the B-spline parameters) and the number of error states. By contrast, in our proposed approach the dimensions of the estimated trajectory and of the error state are decoupled, which allows us to model complex trajectories at a lower computational cost (see Section V-B).

III. Continuous-Time Error-State Representation

In this section, we present the main idea of the DEEP formulation, which relies on a low-dimensional, continuous-time representation of the trajectory errors. As discussed in Section I, our choice is to employ a B-spline representation of the errors. Similarly to prior work [7], this is motivated by the need to have temporal basis functions with local support and simple analytical derivatives. In what follows, we briefly present the B-spline representation of a function, and then describe the way in which it is employed in our work.

A. B-Spline function representation

A B-spline function of degree k is a piecewise-polynomial function of degree k, defined over an interval $[t_0,\ t_n]$. The points $t_i, i = 0, \dots n$, are termed the *knots* of the B-spline, and in our work we assume that the knots are uniformly distributed (i.e., we employ a uniform B-spline). We employ quadratic and cubic B-splines. Over the time interval $[t_i, t_{i+1}]$, a quadratic B-spline function is given by [25]:

$$r_{2,i}(u) = \underbrace{\frac{1}{2}\begin{bmatrix}u^2 & u & 1\end{bmatrix}\begin{bmatrix}1 & -2 & 1\\ -2 & 2 & 0\\ 1 & 1 & 0\end{bmatrix}}_{\mathbf{B}_2(u)}\begin{bmatrix}c_i\\ c_{i+1}\\ c_{i+2}\end{bmatrix} \quad (1)$$

where $u = (t - t_i)/\delta t$, $\delta t = \frac{t_n - t_0}{n}$. In the above equation, c_i, c_{i+1} and c_{i+2} are the parameters, termed *control points*, which determine the form of the polynomial function. Similarly, a

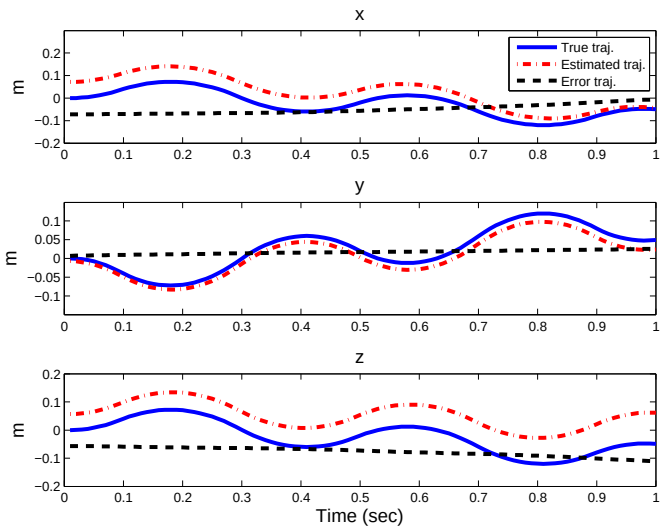

Fig. 1. Example trajectory and estimation errors during a one-second long interval of visual-inertial localization.

cubic B-spline function over the interval $[t_i, t_{i+1}]$ is defined as:

$$r_{3,i}(u) = \underbrace{\frac{1}{2}\begin{bmatrix} u^3 & u^2 & u & 1 \end{bmatrix}\begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 0 & 3 & 0 \\ 1 & 4 & 1 & 0 \end{bmatrix}}_{\mathbf{B}_3(u)}\begin{bmatrix} c_i \\ c_{i+1} \\ c_{i+2} \\ c_{i+3} \end{bmatrix} \tag{2}$$

The above equations define the B-spline functions in each of the intervals between knots. The value of the function at any time $t \in [t_0, t_n)$ is given by:

$$r(t) = \begin{bmatrix} \mathbf{0} & \mathbf{B}_k(u) & \mathbf{0} \end{bmatrix} \mathbf{c} \tag{3}$$

$$= \bar{\mathbf{B}}_k(t)\mathbf{c} \tag{4}$$

where k denotes the chosen degree of the B-spline, and $\mathbf{c}$ is the vector containing all the control points c_i, $i = 0, \ldots, n+k-1$. Note that the relationship between the function value, $r(t)$, and $\mathbf{c}$ is linear, which will be important for our formulation.

B. Key idea

B-splines are useful tools for representing (or approximating) a smooth function, a fact that motivated their use for trajectory modeling in [7] and derivative approaches. In these formulations, the dimension of the state vector, and therefore the computational cost of the estimator, depends on the spacing of the knots. When the knots are spaced closely in time, we are able to model complex trajectories. However, using a large number of knots increases the dimension of the state vector and the computational requirements, which is a key drawback. Moreover, since the required dimension of the knot vector depends on the trajectory itself, it may be hard to make this choice in advance (this is also discussed in [23]).

To motivate the DEEP formulation proposed here, in Fig. 1 we present the trajectory (x-y-z position) of a platform during a one-second long window from one of our simulations, which involves localization using visual and inertial measurements. Specifically, in these plots the blue solid lines represent the true trajectory; the red dash-dotted lines represent the trajectory estimate computed via inertial propagation in this time interval, starting from an initial inaccurate estimate; and the black dashed lines represent the errors in the estimates (i.e., the difference between the true and estimated position). The key point to notice in this plot is that, while the actual trajectory is quite "complex" in this time interval, the functions describing the estimation errors are much "smoother". In turn, this means that in order to obtain an accurate B-spline approximation of the actual trajectory, a *significantly larger knot vector would be required*, compared to that needed for a B-spline approximation to the trajectory errors. Our approach takes advantage of this observation.

Specifically, consider a platform equipped with proprioceptive sensors (e.g., an inertial measurement unit (IMU)) as well as exteroceptive ones (e.g., a camera). Let us assume that during a time interval $[t_0, t_n]$, M exteroceptive measurements are recorded (e.g., M images). In the DEEP formulation, the estimator maintains an estimate for the platform states (e.g., poses) corresponding to these M time instants, $\hat{\mathbf{x}}_i$, $i = 1, \ldots M$. An initial estimate for each state is computed via propagation (e.g., integration of the inertial measurements) from the latest available pose, and updated as more measurement information is processed. The novelty of our approach lies in the representation of the errors. In particular, we model the state error, $\tilde{\mathbf{x}}(t)$, $t \in [t_0, t_n]$ via a B-spline representation. As a result, the error in the estimate of the i-th pose, $\tilde{\mathbf{x}}_i$, which is required in the derivation of the linearized residual equations in the estimator, can be written as a linear function of the control-point vector (see (3)). Since all linearized residuals can be written in terms of this vector, it constitutes the "error-state vector" of the estimator.

If the number of knots needed to accurately describe the trajectory errors is smaller than M, then the dimension of the error-state vector in the DEEP formulation will be smaller than that of the "traditional" approach in which M pose errors are explicitly represented. As shown in the following sections, at least for the problem of visual-inertial localization, we can use a number of knots that is significantly smaller than M (e.g., five to ten times smaller), without incurring significant loss of estimation accuracy. The justification for this lies in the fact that, as shown in Fig. 1, the estimation errors tend to be "smooth" over short periods of time. Moreover, we note that the "complexity" of the estimation error is, to a large extent, independent from the complexity of the trajectory itself. This is useful, since it allows us to choose the knot density without having prior knowledge of the trajectory.

IV. Visual-Inertial Odometry in the DEEP Formulation

We now describe a visual-inertial localization algorithm that employs the DEEP approach described in the preceding section. Visual-inertial localization is a problem that has recently

attracted significant research interest, as the miniaturization of visual and inertial sensors has rendered them ideal for pose estimation in small-scale systems (e.g., smartphones and micro aerial vehicles). The algorithm we describe here is an adaptation of the MSCKF 2.0 algorithm [20, 17] that utilizes the DEEP framework. The MSCKF 2.0 has been shown in prior work to outperform alternative methods for visual-inertial odometry, and thus can be used as a benchmark for examining the performance of the DEEP formulation.

In the original MSCKF estimator, the state vector consists of a sliding window of M camera poses, which correspond to the time instants the last M images were recorded. The key characteristic of the approach is that all measurements of each feature are used simultaneously once the feature is lost from tracking, to apply constraints on the poses of the sliding window. The DEEP-MSCKF estimator we describe here follows the same approach.

A. Formulation

Consider a platform equipped with an IMU and a monocular camera, moving in an area populated with naturally-occurring point features, whose coordinates are not known a priori. Our goal is to estimate the position and orientation of the platform with respect to a global coordinate frame, $\{G\}$, using the inertial measurements and observations of these features. To derive the estimator's equations, we affix a coordinate frame $\{I\}$ to the IMU, and a coordinate frame $\{C\}$ to the camera. We here assume that the camera is intrinsically calibrated, and the relative rotation and translation between $\{I\}$ and $\{C\}$ are known.

The IMU state at time-step k is described by the vector:

$$\mathbf{x}_{I_k} = \begin{bmatrix} {}^{I_k}_{G}\bar{\mathbf{q}}^T & {}^{G}\mathbf{p}_k^T & {}^{G}\mathbf{v}_k^T & \mathbf{b}_{\mathbf{g}_k}^T & \mathbf{b}_{\mathbf{a}_k}^T \end{bmatrix}^T \tag{5}$$

where ${}^{I_k}_{G}\bar{\mathbf{q}}$ is the unit quaternion [29] representing the rotation from the global frame $\{G\}$ to the IMU frame $\{I\}$ at time-step k, ${}^{G}\mathbf{p}_k$ and ${}^{G}\mathbf{v}_k$ are the IMU position and velocity in the global frame, and $\mathbf{b}_{\mathbf{g}_k}$ and $\mathbf{b}_{\mathbf{a}_k}$ are the gyroscope and accelerometer biases, respectively, which are modeled as Gaussian random-walk processes.

The error in the estimate of the IMU state (5) is defined as [17]:

$$\tilde{\mathbf{x}}_{I_k} = \begin{bmatrix} {}^{G}\tilde{\boldsymbol{\theta}}_k^T & {}^{G}\tilde{\mathbf{p}}_k^T & {}^{G}\tilde{\mathbf{v}}_k^T & \tilde{\mathbf{b}}_{\mathbf{g}_k}^T & \tilde{\mathbf{b}}_{\mathbf{a}_k}^T \end{bmatrix}^T \tag{6}$$

where the standard additive error definition is used for the position, velocity and biases (i.e., if $\hat{\mathbf{y}}$ is the estimate of a variable $\mathbf{y}$, the estimation error is defined as $\tilde{\mathbf{y}} = \mathbf{y} - \hat{\mathbf{y}}$), while for the orientation errors we use a minimal 3-dimensional representation, defined by the equation:

$${}^{I_k}_{G}\bar{\mathbf{q}} \approx {}^{I_k}_{G}\hat{\bar{\mathbf{q}}} \otimes \begin{bmatrix} \frac{1}{2}{}^{G}\tilde{\boldsymbol{\theta}}_k \\ 1 \end{bmatrix} \tag{7}$$

where $\otimes$ denotes quaternion multiplication.

B. State vector of the DEEP-MSCKF

The *estimated* state vector of the DEEP-MSCKF contains an estimate for the current IMU state, and for the poses at the time instants the latest M images were recorded:

$$\hat{\mathbf{x}}_k = \begin{bmatrix} \hat{\mathbf{x}}_{I_k}^T & \hat{\boldsymbol{\pi}}_{k-1}^T & \hat{\boldsymbol{\pi}}_{k-2}^T & \cdots & \hat{\boldsymbol{\pi}}_{k-M}^T \end{bmatrix}^T \tag{8}$$

where $\hat{\boldsymbol{\pi}}_i = [{}^{I_i}_{G}\hat{\bar{\mathbf{q}}}^T \quad {}^{G}\hat{\mathbf{p}}_i^T]^T$, for $i = k-M, \ldots, k-1$, are the estimates of the IMU poses at the time instants the last M images were recorded.

The above state vector is identical to the state vector of the MSCKF 2.0 formulation. The difference lies in the representation of the errors in the M poses included in the estimated state vector. Specifically, instead of maintaining an individual error state for each of the M poses, in the DEEP formulation we model the position and orientation errors in the time interval these M poses were recorded by B-spline functions. The error-state of the DEEP-MSCKF therefore contains the control points of these B-splines. Specifically, the error-state vector at time-step k is defined as:

$$\tilde{\mathbf{x}}_k = [\tilde{\mathbf{b}}_{\mathbf{g}_k}^T \; \tilde{\mathbf{b}}_{\mathbf{a}_k}^T \; \underbrace{\mathbf{c}_{\mathbf{p}_k}^T \; \mathbf{c}_{\mathbf{p}_{k-N}}^T \cdots \mathbf{c}_{\mathbf{p}_{k-nN}}^T}_{\mathbf{x}_{\mathbf{p}_k}^T} \; \underbrace{\mathbf{c}_{\boldsymbol{\theta}_k}^T \; \mathbf{c}_{\boldsymbol{\theta}_{k-N}}^T \cdots \mathbf{c}_{\boldsymbol{\theta}_{k-(n-1)N}}^T}_{\mathbf{x}_{\boldsymbol{\theta}_k}^T}]^T \tag{9}$$

where $\mathbf{c}_{\mathbf{p}_j}$, $j = k - Nn, k - Nn + N, .., k$ and $\mathbf{c}_{\boldsymbol{\theta}_j}$, $j = k - (n-1)N, k - (n-2)N, .., k$ are 3×1 control-point vectors used to represent the position and orientation errors, respectively. To simplify the presentation, we here assume that a new knot is introduced every N camera images (note however, that having the knot interval be an integer multiple of the image period is not a strict requirement). In effect, N here is a "decimation factor", which determines the number of knots needed to cover the time interval in which the latest M images were recorded. Increasing N results in smaller size of the error-state vector (and thus lower computational cost), and vice versa. Note that in (9) the number of position control points is larger than the number of orientation control points by one. This is due to the fact that in our implementation we employ a quadratic B-spline to represent the orientation errors, and a cubic B-spline to represent the position errors. This choice was dictated by the fact that the IMU accelerometer provides measurements of the second-order derivative of the position, while the gyroscope provides measurements of the first-order derivative of the orientation.

With this representation, the errors of each individual pose in the estimated state vector in (8) are expressed as:

$${}^{G}\tilde{\boldsymbol{\theta}}_j = \mathbf{B}_{\boldsymbol{\theta}}(t_j)\mathbf{x}_{\boldsymbol{\theta}_k} \tag{10}$$

$${}^{G}\tilde{\mathbf{p}}_j = \mathbf{B}_{\mathbf{p}}(t_j)\mathbf{x}_{\mathbf{p}_k} \tag{11}$$

where $\mathbf{B}_{\boldsymbol{\theta}}(t_j) = \mathrm{kron}(\bar{\mathbf{B}}_2(t_j), \mathbf{I}_3)$ and $\mathbf{B}_{\mathbf{p}}(t_j) = \mathrm{kron}(\bar{\mathbf{B}}_3(t_j), \mathbf{I}_3)$, with kron denoting the Kronecker matrix product, and $\mathbf{I}_3$ the 3×3 identity matrix. Moreover, we note that the errors in the velocity can also be written as a linear

function of the control points:

$$^{G}\tilde{\mathbf{v}}_j = \mathbf{B}_{\mathbf{p}}^{(1)}(t_j)\mathbf{x}_{\mathbf{p}_k} \tag{12}$$

where $\mathbf{B}_{\mathbf{p}}^{(1)}(t_j)$ is the time-derivative of $\mathbf{B}_{\mathbf{p}}(t_j)$.

C. State augmentation

The DEEP-MSCKF filter proceeds as follows: let us consider that at time-step k (i.e., time t_k) a filter update is performed (see Section IV-D). From that point on, the estimator uses the IMU measurements for propagating the IMU state estimate $\hat{\mathbf{x}}_I$, using the integration equations described in [17]. Every time a new image is recorded, a new pose estimate $\hat{\boldsymbol{\pi}}_j$ is included in the estimated state vector (8), but the error-state, and corresponding covariance matrix, remains unchanged. A new set of control points is introduced in the error-state vector (9) at time t_{k+N}, i.e, once N images have been recorded. At this time, the error-state is augmented by including the vector:

$$\tilde{\mathbf{x}}_{new} = \begin{bmatrix} \tilde{\mathbf{b}}_{\mathbf{g}_{k+N}}^T & \tilde{\mathbf{b}}_{\mathbf{a}_{k+N}}^T & \mathbf{c}_{\mathbf{p}_{k+N}}^T & \mathbf{c}_{\boldsymbol{\theta}_{k+N}}^T \end{bmatrix}^T \tag{13}$$

which consists of the errors of the IMU biases at the current time, as well as the control points needed for representing the position and orientation errors in the time interval $[t_k, t_{k+N}]$.

To complete the augmentation, we must also augment the covariance matrix of the EKF. To this end, we must compute the covariance matrix of $\tilde{\mathbf{x}}_{new}$, as well as its cross-correlation with the other error-states. We begin with the expression relating the IMU-pose errors at time t_{k+N} and t_k:

$$\tilde{\mathbf{x}}_{I_{k+N}} = \boldsymbol{\Phi}_I(t_{k+N}, t_k)\tilde{\mathbf{x}}_{I_k} + \mathbf{w}_k \tag{14}$$

where $\mathbf{w}_k$ is the process-noise error, which is zero-mean Gaussian with covariance matrix $\mathbf{Q}_k$. The computation of $\boldsymbol{\Phi}_I(t_{k+N}, t_k)$ and $\mathbf{Q}_k$ is described in [17]. To obtain an expression relating $\tilde{\mathbf{x}}_{new}$ with the remaining terms in the error-state vector of the DEEP-MSCKF, we note that the IMU-state errors can be written as linear functions of the error-state, owing to the properties of the B-spline representation (see (10)-(12)):

$$\tilde{\mathbf{x}}_{I_k} = \boldsymbol{\Xi}_1\tilde{\mathbf{x}}_k \tag{15}$$

$$\tilde{\mathbf{x}}_{I_{k+N}} = \begin{bmatrix} \boldsymbol{\Xi}_2 & \boldsymbol{\Xi}_3 \end{bmatrix} \begin{bmatrix} \tilde{\mathbf{x}}_{new} \\ \tilde{\mathbf{x}}_k \end{bmatrix} \tag{16}$$

Using the above equations, along with (14), and solving for $\tilde{\mathbf{x}}_{new}$, we obtain:

$$\tilde{\mathbf{x}}_{new} = \boldsymbol{\Xi}_2^{\dagger}\left(\boldsymbol{\Phi}_I(t_{k+N}, t_k)\boldsymbol{\Xi}_1 - \boldsymbol{\Xi}_3\right)\tilde{\mathbf{x}}_k + \boldsymbol{\Xi}_2^{\dagger}\mathbf{w}_k$$

where $\boldsymbol{\Xi}_2^{\dagger}$ is the pseudoinverse of $\boldsymbol{\Xi}_2$. Denoting $\mathbf{A} = \boldsymbol{\Xi}_2^{\dagger}\left(\boldsymbol{\Phi}_I(t_{k+N}, t_k)\boldsymbol{\Xi}_1 - \boldsymbol{\Xi}_3\right)$, we can now write the covariance matrix of the augmented error-state:

$$\mathbf{P}_{k+N|k} = \begin{bmatrix} \mathbf{A}\mathbf{P}_{k|k}\mathbf{A}^T + \boldsymbol{\Xi}_2^{\dagger}\mathbf{Q}_k\boldsymbol{\Xi}_2^{\dagger T} & \mathbf{A}\mathbf{P}_{k|k} \\ \mathbf{P}_{k|k}\mathbf{A}^T & \mathbf{P}_{k|k} \end{bmatrix} \tag{17}$$

where $\mathbf{P}_{k|k}$ is the covariance matrix of the error-state vector after the update at time step k. As a final step, we remove from the error-state and the covariance matrix the terms corresponding to the IMU biases at time-step k, as these are no longer necessary.

D. Update

Once the error-state augmentation is complete, we proceed to process the feature measurements from the camera. Specifically, we process all measurements from features that (i) are no longer being tracked at time step $k+N$, or (ii) are still being tracked at time step $k+N$, but their oldest measurements were recorded during the time interval corresponding to the oldest available knots. The latter is necessary, since in the MSCKF after each update the oldest states (i.e., the oldest control points in this case) are removed, to maintain a sliding window of bounded length.

The update equations of the DEEP-MSCKF estimator follow closely those of the original MSCKF, with essentially the only difference being the way in which Jacobians are computed. Specifically, let us we denote the measurement of feature i from the j-th pose as:

$$\mathbf{z}_{ij} = \mathbf{h}(\boldsymbol{\pi}_j, {}^{G}\mathbf{p}_{f_i}) + \mathbf{n}_{ij} \tag{18}$$

where $\mathbf{n}_{ij}$ is zero-mean white Gaussian noise with covariance matrix $\sigma^2\mathbf{I}_2$, and ${}^{G}\mathbf{p}_{f_i}$ is the feature position in the global frame. Considering a feature that has been observed in m images, the first step in the update process is to employ all its measurements to obtain an estimate of its position, ${}^{G}\hat{\mathbf{p}}_{f_i}$, via triangulation. Next, we proceed to compute the residuals for all m measurements:

$$\mathbf{r}_{ij} = \mathbf{z}_{ij} - \mathbf{h}(\hat{\boldsymbol{\pi}}_j, {}^{G}\hat{\mathbf{p}}_{f_i}) \tag{19}$$

All these residuals are stacked in a block vector $\mathbf{r}_i$, whose block elements are $\mathbf{r}_{ij}$. To derive the EKF Jacobians, we begin by linearizing each of the above residuals:

$$\mathbf{r}_{ij} \simeq \mathbf{H}_{\boldsymbol{\pi}_{ij}}\tilde{\boldsymbol{\pi}}_j + \mathbf{H}_{\mathbf{f}_{ij}}{}^{G}\tilde{\mathbf{p}}_{f_i} + \mathbf{n}_{ij} \tag{20}$$

where $\tilde{\boldsymbol{\pi}}_j$ is the error in the j-th pose estimate, ${}^{G}\tilde{\mathbf{p}}_{f_i}$ is the feature position error, and $\mathbf{H}_{\boldsymbol{\pi}_{ij}}$ and $\mathbf{H}_{\mathbf{f}_{ij}}$ are the Jacobians of the measurement function with respect to the IMU pose and feature position, respectively. Employing the B-spline representation of the errors, we can write $\tilde{\boldsymbol{\pi}}_j$ as a *linear* function of the error-state:

$$\tilde{\boldsymbol{\pi}}_j = \begin{bmatrix} {}^{G}\tilde{\boldsymbol{\theta}}_j \\ {}^{G}\tilde{\boldsymbol{p}}_j \end{bmatrix} = \begin{bmatrix} \mathbf{B}_{\boldsymbol{\theta}}(t_j)\tilde{\mathbf{x}}_{\boldsymbol{\theta}_{k+N}} \\ \mathbf{B}_{\mathbf{p}}(t_j)\tilde{\mathbf{x}}_{\mathbf{p}_{k+N}} \end{bmatrix} = \mathbf{B}_j\tilde{\mathbf{x}}_{k+N}$$

Therefore, (20) becomes:

$$\mathbf{r}_{ij} \simeq \mathbf{H}_{\boldsymbol{\pi}_{ij}}\mathbf{B}_j\tilde{\mathbf{x}}_{k+N} + \mathbf{H}_{\mathbf{f}_{ij}}{}^{G}\tilde{\mathbf{p}}_{f_i} + \mathbf{n}_{ij} \tag{21}$$

and by stacking all m residuals we obtain:

$$\mathbf{r}_i \simeq \mathbf{H}_{c_i}\tilde{\mathbf{x}}_{k+N} + \mathbf{H}_{\mathbf{f}_i}{}^{G}\tilde{\mathbf{p}}_{f_i} + \mathbf{n}_i \tag{22}$$

where $\mathbf{H}_{c_i}$ is a matrix with block rows $\mathbf{H}_{\boldsymbol{\pi}_{ij}}\mathbf{B}_j$, $\mathbf{H}_{\mathbf{f}_i}$ is a matrix with block rows $\mathbf{H}_{\mathbf{f}_{ij}}$, and $\mathbf{n}_i$ is a block vector with elements $\mathbf{n}_{ij}$. Based on the linearized expression in (22), the update proceeds as in the original MSCKF, i.e., by removing ${}^{G}\tilde{\mathbf{p}}_{f_i}$ from the linearized expression. To achieve this, we define

a matrix $\mathbf{V}_i$ whose columns form a basis for the left nullspace of $\mathbf{H}_{\mathbf{f}_i}$, and define $\mathbf{r}_i^o$ as:

$$\mathbf{r}_i^o = \mathbf{V}_i^T \mathbf{r}_i \simeq \mathbf{H}_i^o \tilde{\mathbf{x}}_{k+N} + \mathbf{n}_i^o \tag{23}$$

where $\mathbf{H}_i^o = \mathbf{V}_i^T \mathbf{H}_{c_i}$ and $\mathbf{n}_i^o = \mathbf{V}_i^T \mathbf{n}_i$. For computational efficiency, we can compute $\mathbf{r}_i^o$ and $\mathbf{H}_i^o$ without explicitly computing $\mathbf{V}_i$ [20]. Once $\mathbf{r}_i^o$ and $\mathbf{H}_i^o$ are computed, we proceed by performing a Mahalanobis gating test to reject outliers, and features whose residuals pass the test are employed in an EKF update, analogously to [20], to compute the state correction and the updated covariance matrix. Once the correction to the position and orientation knot parameters is computed in this way, we proceed to compute the pose corrections by employing (10)-(12), and subsequently these pose corrections are applied to the estimates in the state vector (8). After the update is complete, the oldest control points are removed from the error-state vector, similarly to the original MSCKF.

As discussed in [20], the computational complexity of the MSCKF estimator is linear in the number of features, and *cubic* in the size of the state vector. This remains true in the DEEP formulation of the MSCKF, where now the computational complexity is cubic in the size of the error-state vector (9). We therefore see that by changing the frequency at which new knots are introduced (i.e., changing N), the computational cost of the estimator will be significantly impacted. As knots are placed more sparsely (i.e., N is increased), the size of the error-state vector and the computational cost of the estimator will decrease. However, placing knots more sparsely also reduces the fidelity with which the trajectory can be described, and this degrades accuracy. The key result, which we experimentally demonstrate in Sections V and VI, is that while with increasing N the reduction in computational cost is rapid, the loss of accuracy is only modest.

E. Ensuring consistency

As the analysis of [17] has shown, the performance of the MSCKF estimator can be significantly improved by computing the filter Jacobians in a way that ensures that the linearized system model has observability properties matching those of the underlying nonlinear system (i.e., the global pose and yaw are unobservable). In [17] it was shown that this can be achieved if the Jacobians are evaluated using a *single* estimate for each of the position and velocity states (the first available one). The resulting estimator, termed MSCKF 2.0, has improved accuracy and consistency[1]. We here follow a similar approach to improve the performance of the DEEP-MSCKF.

Specifically, similarly to [17], we use the same estimate for each position and velocity state when evaluating all Jacobians that involve it. However, for the DEEP-MSCKF approach, an additional modification is needed. It can be shown that to maintain the appropriate observability properties of the linearized system model, the position and velocity estimates used in Jacobian evaluation must conform to a B-spline model of the trajectory [30]. Therefore, prior to each update we perform a local B-spline fitting of the new segment of the trajectory, and then use the resulting estimates in evaluating all Jacobians involving these states. We stress that, similarly to [17], these estimates are only used in order to evaluate the Jacobians, and are not used in computing residuals.

[1]A recursive estimator is termed consistent when the ensemble mean of the estimation errors is zero, and their ensemble covariance matrix is equal to the one reported by the estimator [1].

V. Simulations

In this section we present the results from two sets of Monte-Carlo simulations to demonstrate the performance of the DEEP formulation of the MSCKF algorithm. In our simulator, we generate trajectories emulating a platform (e.g., a handheld device) moving in a building-sized environment, in a trajectory that involves significant rotations and accelerations. In each Monte-Carlo trial a 3-minute, approximately 260-m long trajectory is generated, as well as feature tracks and inertial measurements with characteristics matching those that we observe in real-world datasets. Specifically, the images are available at 20 Hz, IMU measurements at 100 Hz, and the average feature-track length is 7.4 frames. In each trial, different feature positions and different noise realizations are used. The noise parameters match those of the sensors found on a LG Nexus 4 mobile device.

A. DEEP formulation vs. MSCKF 2.0

We begin by comparing the proposed DEEP formulation of the MSCKF to the "standard", pose-based formulation in [17]. Since the DEEP-MSCKF is derived by employing an approximate representation of the errors, we expect the MSCKF 2.0 to attain the highest estimation accuracy. Our goal is to examine the tradeoff between the loss of accuracy and the computational gains as N changes. All estimators process the same data, and the sliding-window length at each time instant is equal to the current longest feature track (with a maximum allowable length corresponding 3 sec, or 60 poses).

The performance metrics we compute are (i) the root-mean-squared errors (RMSE) for the position and orientation, (ii) the computational cost of the methods, expressed in terms of number of floating-point operations (flops) needed for each augmentation/update cycle, and (iii) the normalized estimation error squared (NEES) for the pose errors. If we denote the pose error at time step k as $\tilde{\mathbf{x}}_p(k)$ and the corresponding 6×6 covariance matrix reported by the filter by $\mathbf{P}_p(k)$, the NEES at this time instant is defined as $\tilde{\mathbf{x}}_p(k)^T \mathbf{P}_p^{-1}(k) \tilde{\mathbf{x}}_p(k)$. Examining the NEES gives an insight into the magnitude of the unmodeled errors incurred by the DEEP parameterization. Specifically, if significant unmodeled errors exist, the covariance matrix reported by the estimator will be smaller than the covariance matrix of the actual errors (i.e., the estimator will be inconsistent [1]), and the NEES will increase. In a consistent estimator, the expected value of the pose NEES should equal six. Thus, by examining the deviation of the average NEES from this value, we can evaluate the significance of the unmodeled errors.

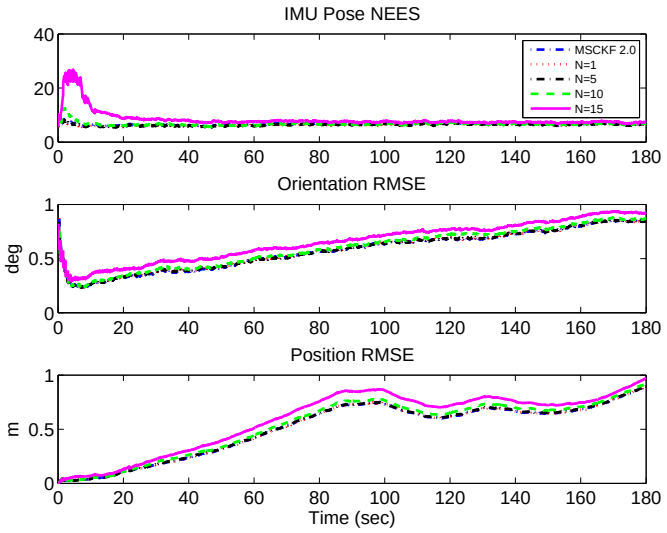

Fig. 2. Simulation results: the position and orientation RMSE, as well as the IMU-pose NEES over time. Plots are averages over 100 Monte-Carlo trials. The compared methods are the MSCKF 2.0 (blue dash-dotted line), and the DEEP-MSCKF with $N = 1$ (red dotted line), $N = 5$ (black dash-dotted line), $N = 10$ (green dashed line), and $N = 15$ (magenta solid line).

TABLE I
SIMULATION RESULTS: MSCKF 2.0 VS. DEEP-MSCKF FOR VARYING N

	MSCKF 2.0	DEEP-MSCKF			
		N=1	N=5	N=10	N=15
Orientation RMSE (deg)	0.577	0.578	0.577	0.596	0.657
Position RMSE (m)	0.505	0.504	0.509	0.529	0.585
NEES	6.63	6.38	6.36	6.68	8.41
Flops	100%	112.45%	11.18%	6.20%	4.83%

TABLE II
SIMULATION RESULTS: DEEP VS. TBF FORMULATION

	N=1		N=3		N=5	
	DEEP	TBF	DEEP	TBF	DEEP	TBF
Orientation RMSE (deg)	0.569	0.563	0.563	0.577	0.572	0.821
Position RMSE (m)	0.491	0.491	0.485	0.631	0.501	2.166
NEES	6.24	6.40	6.16	11.66	6.24	67.86

Fig. 2 shows the RMSE as well as the pose NEES for the different methods, averaged over 100 Monte-Carlo trials. In this plot, we compare the MSCKF 2.0 method to the DEEP-MSCKF, using values for the "decimation factor" $N = 1, 5, 10, 15$. From these plots, we can observe that up to $N = 10$, the average performance of the DEEP formulation is almost indistinguishable to that of the pose-based MSCKF formulation, while the estimation accuracy is noticeably lower for $N = 15$. To explore these results in more detail, in Table I we show the average RMSE and NEES for all algorithms, averaged over all Monte-Carlo trials and all time. Moreover, in this table we include the average flops for the different algorithms, expressed as a percentage of the flops needed in the MSCKF 2.0.

From the results of Table I we can clearly see that, as the knot density decreases in the DEEP-MSCKF (i.e., as N increases), the computational cost of the algorithm decreases rapidly, while the estimation errors increase slowly. For instance, when $N = 5$, we achieve 89% savings in computation, while incurring an approximately 1% increase in RMSE. Moreover, note that for values up to $N = 10$, the NEES remains close to the "ideal" value of 6. This indicates that the unmodelled errors resulting from the DEEP formulation are sufficiently small.

B. DEEP formulation vs. B-spline trajectory parameterization

We next compare the performance of the proposed DEEP formulation to a version of the MSCKF estimator that employs the temporal basis function (TBF) formulation of [18, 7, 23]. The latter estimator is similar to the one presented in Section IV, with the difference that the trajectory itself, rather than just the errors, are modeled by B-splines. This comparison is informative as both these approaches have practically identical computational cost, since for the same value of N they involve error-state vectors of the same size.

Table II shows the results of 100 Monte-Carlo simulations, comparing the two approaches for N ranging between one and five. We can observe that the two parameterizations have similar results when $N = 1$, but the performance of the TBF approach degrades rapidly for larger values of N. When $N = 5$, the position RMSE of the TBF formulation is more than quadruple that of the DEEP. Moreover, the NEES of the TBF formulation is larger than its ideal value by an order of magnitude.

Two important conclusions can be drawn from these results. First, it becomes clear that the good performance of the DEEP approach cannot be attributed to the use of a "smooth" trajectory in the simulator. If that were the case, then a B-spline representation of the trajectory (which is used in the TBF approach) would suffice, and would also yield good results. Second, we can see that the decoupling of the representation of trajectory *estimates* from that of the trajectory *errors*, which is the key novelty of the DEEP formulation, yields added representational power, compared to a standard B-spline representation of the trajectory. Since the trajectory estimates are represented by a set of poses, no assumption on the form of the trajectory itself is imposed, and this leads to the favorable accuracy-computation tradeoff observed in Table I.

VI. REAL-WORLD EXPERIMENT

We next present the results of a real-world experiment, in which a camera/IMU platform was mounted on a car driven in a residential area of Riverside, CA. The trajectory length was 5km, driven in 20 min, and sample images from the experiment are shown in Fig. 4. Shi-Tomasi features were extracted from images [26], and normalized cross-correlation was used for

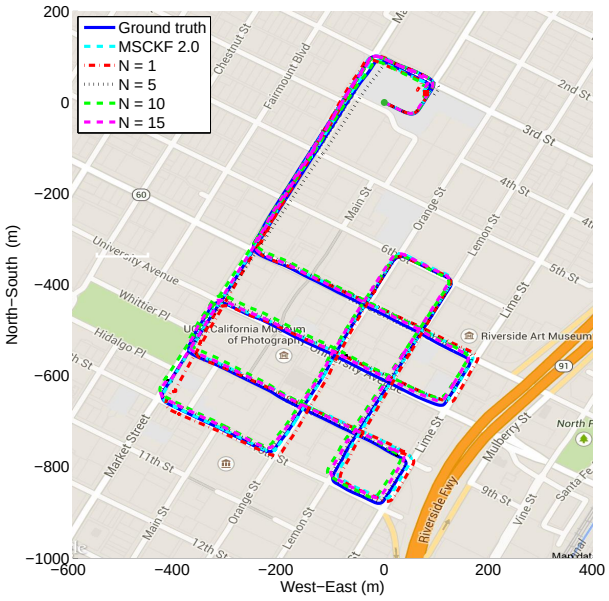

Fig. 3. Real-world experiment: trajectory estimates. The compared methods are the MSCKF 2.0 (dashed cyan line), and the DEEP-MSCKF with $N = 1$ (red dash-dotted line), $N = 5$ (black dotted line), $N = 10$ (green dashed line), and $N = 15$ (magenta dashed line). The solid blue line represents the ground truth trajectory.

Fig. 4. Sample images recorded during the experiment.

feature matching. All feature tracks were extracted in a pre-processing step, and used by all the algorithms compared, to ensure that all methods use exactly the same measurements.

Fig. 3 shows the trajectory estimated by the MSCKF 2.0 and the DEEP-MSCKF with N ranging from 1 to 15. The trajectory is plotted on a map of the area, along with the ground-truth trajectory provided by a high-precision GPS/INS solution. Moreover, in Fig. 5 we plot the position errors (distance between the computed estimates and ground truth) throughout the trajectory. Table III lists the average position errors throughout the trajectory, as well as the computational cost of the DEEP formulation with different values of N as a percentage of the cost of the MSCKF 2.0. We can observe that, as in the simulation results presented in the preceding section, the DEEP-MSCKF formulation results in accuracy that is similar to that of the MSCKF 2.0 algorithm, but has significantly lower computational cost when $N \geq 5$. In fact, for this specific dataset, the smallest average errors are obtained by the DEEP-MSCKF with $N = 5$. This is due to

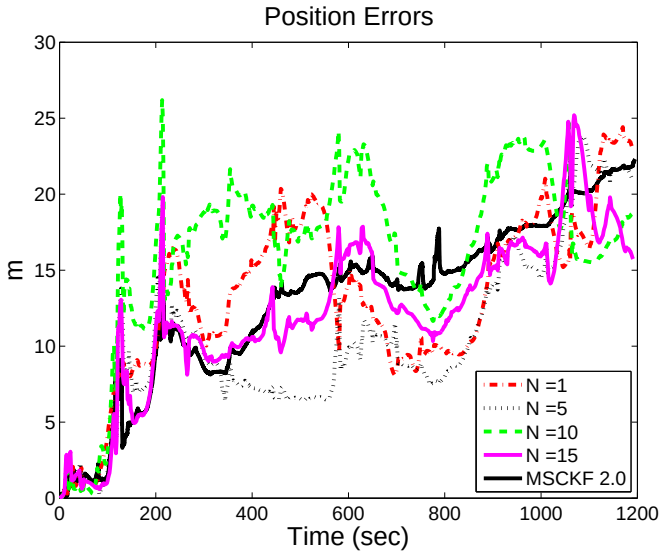

Fig. 5. Real-world experiment: position errors of the MSCKF 2.0 and the DEEP-MSCKF for increasing N.

the stochastic nature of the noise, and we expect that if several experiments were conducted, the accuracy would decrease monotonically with N, as in the Monte-Carlo simulations.

TABLE III
REAL-WORLD EXPERIMENT: DEEP-MSCKF VS. MSCKF 2.0

	MSCKF 2.0	N=1	N=5	N=10	N=15
Avg. position error (m)	13.01	13.20	10.65	16.29	12.20
Flops	100%	104.23%	7.71%	4.17%	3.26%

VII. CONCLUSION

In this paper, we have presented a novel formulation for the representation of the trajectory in pose-estimation problems. The key idea of the proposed DEEP approach is the decoupling of the representation of the trajectory *estimates* from that of the trajectory *errors*. Specifically, for the former a general, pose-based representation is employed, which imposes no assumptions on the form of the trajectory. On the other hand, for the latter a B-spline representation is used. This makes it possible to reduce the dimensionality of an estimator's error-state vector, simply by lowering the frequency at which knots are introduced. This general approach is used to design a novel visual-inertial odometry algorithm, which we term DEEP-MSCKF. By comparing the performance of this method to the "traditional" pose-based MSCKF formulation of [17], we demonstrate that the DEEP formulation yields substantial gains in computational efficiency, while resulting in only small loss of accuracy.

ACKNOWLEDGMENTS

This work was supported by the National Science Foundation (grant no. IIS-1117957, IIS-1253314 and IIS-1316934).

References

[1] Yaakov Bar-Shalom, Thiagalingam Kirubarajan, and X.-Rong Li. *Estimation with Applications to Tracking and Navigation*. John Wiley & Sons, Inc., New York, NY, USA, 2002.

[2] Charles Bibby and Ian Reid. A hybrid SLAM representation for dynamic marine environments. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 257–264, May 2010.

[3] Michael Bosse and Robert Zlot. Continuous 3D scan-matching with a spinning 2D laser. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 4312–4319, Anchorage, AK, May 2009.

[4] Michael Bosse, Robert Zlot, and Paul Flick. Zebedee: Design of a spring-mounted 3-D range sensor with application to mobile mapping. *IEEE Transactions on Robotics*, 28(5):1104–1119, 2012.

[5] Javier Civera, Andrew J. Davison, and J. M. Martinez Montiel. Inverse depth parametrization for monocular SLAM. *IEEE Transactions on Robotics*, 24(5):932–945, Oct. 2008.

[6] Frank Dellaert. Square root SAM. In *Proceedings of Robotics: Science and Systems*, Cambridge, MA, June 2005.

[7] Paul Furgale, Timothy D. Barfoot, and Gabe Sibley. Continuous-time batch estimation using temporal basis functions. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 2088–2095, May 2012.

[8] Jose Guivant and Eduardo Nebot. Optimization of the simultaneous localization and map building algorithm for real time implementation. *IEEE Transactions on Robotics and Automation*, 17(3):242–257, June 2001.

[9] Seungwoong Gwak, Junggon Kim, and Frank Chongwoo Park. Numerical optimization on the Euclidean group with applications to camera calibration. *IEEE Transactions on Robotics and Automation*, 19(1):65–74, Feb 2003.

[10] Christoph Hertzberg, René Wagner, Udo Frese, and Lutz Schröder. Integrating generic sensor fusion algorithms with sound state representations through encapsulation of manifolds. *Information Fusion*, 14(1):57–77, January 2013.

[11] Simon 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 (IROS)*, pages 1251–1256, Maui, HI, Oct. 29-Nov. 3 2001.

[12] Michael Kaess, Ananth Ranganathan, and Frank Dellaert. iSAM: Incremental smoothing and mapping. *IEEE Transactions on Robotics*, 24(6):1365 –1378, Dec. 2008.

[13] Michael Kaess, Hordur Johannsson, Richard Roberts, Viorela Ila, John Leonard, and Frank Dellaert. iSAM2: Incremental smoothing and mapping with fluid relinearization and incremental variable reordering. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 3281–3288, May 2011.

[14] Georg Klein and David Murray. Parallel tracking and mapping for small AR workspaces. In *Proceedings of the IEEE and ACM International Symposium on Mixed and Augmented Reality*, Nara, Japan, November 2007.

[15] Henrik Kretzschmar, Cyrill Stachniss, and Giorgio Grisetti. Efficient information-theoretic graph pruning for graph-based SLAM with laser range finders. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 865–871, Sept 2011.

[16] Rainer Kummerle, Giorgio Grisetti, Hauke Strasdat, Kurt Konolige, and Wolfram Burgard. g2o: A general framework for graph optimization. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 3607–3613, Shanghai, China, May 2011.

[17] Mingyang Li and Anastasios I. Mourikis. High-precision, consistent EKF-based visual-inertial odometry. *International Journal of Robotics Research*, 32(6):690–711, May 2013.

[18] Steven Lovegrove, Alonso Patron-Perez, and Gabe Sibley. Spline fusion: A continuous-time representation for visual-inertial fusion with application to rolling shutter cameras. In *Proceedings of the British Machine Vision Conference*. BMVA Press, 2013.

[19] Agostino Martinelli, Viet Nguyen, Nicola Tomatis, and Roland Siegwart. A relative map approach to SLAM based on shift and rotation invariants. *Robotics and Autonomous Systems*, 55(1):50–61, January 2007.

[20] Anastasios I. Mourikis and Stergios I. Roumeliotis. A multi-state constraint Kalman filter for vision-aided inertial navigation. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 3565–3572, Rome, Italy, Apr. 2007.

[21] Esha D. Nerurkar and Stergios I. Roumeliotis. Power-SLAM: a linear-complexity, anytime algorithm for SLAM. *International Journal of Robotics Research*, 30 (6):772–788, May 2011.

[22] Esha D. Nerurkar, Kejian J. Wu, and Stergios I. Roumeliotis. C-KLAM: Constrained keyframe-based localization and mapping. In *Proceedings of the Workshop on Multi-View Geometry in Robotics, held in conjunction with the Robotics: Science and Systems conference*, Berlin, Germany, June 2013.

[23] Luc Oth, Paul Furgale, Laurent Kneip, and Roland Siegwart. Rolling shutter camera calibration. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1360–1367, Portland, OR, June 2013.

[24] Lina M. Paz, José Guivant, Juan D. Tardós, and José Neira. Divide and conquer: EKF SLAM in O(n). *IEEE Transactions on Robotics*, 24(5):1107 –1120, Oct. 2008.

[25] David Salomon. *Computer Graphics and Geometric Modeling*. Springer-Verlag New York, Inc., Secaucus,

NJ, USA, 1st edition, 1999.

[26] Jianbo Shi and Carlo Tomasi. Good features to track. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 593–600, Seattle, WA, 1994.

[27] Malcolm D. Shuster. A survey of attitude representations. *Journal of the Astronautical Sciences*, 41:439–517, October 1993.

[28] Hannes Sommer, Cédric Pradalier, and Paul Furgale. Automatic differentiation on differentiable manifolds as a tool for robotics. In *Proceedings of the International Symposium of Robotics Research*, Singapore, 2013.

[29] Nikolas Trawny and Stergios I. Roumeliotis. Indirect Kalman filter for 3D attitude estimation. Technical Report 2005-002, Dept. of Computer Science & Engineering, University of Minnesota, Minneapolis, MN, Mar. 2005.

[30] Xing Zheng, Mingyang Li, and A. I. Mourikis. Decoupled representation of the error and trajectory estimates for efficient pose estimation: Supplemental materials. Technical report, University of California, Riverside, 2015.