Concurrent Filtering and Smoothing: A Parallel Architecture for Real-Time Navigation and Full Smoothing

Stephen Williams*, Vadim Indelman*, Michael Kaess†, Richard Roberts*, John J. Leonard¶, and Frank Dellaert*

Abstract

We present a parallelized navigation architecture that is capable of running in real-time and incorporating long-term loop closure constraints while producing the optimal Bayesian solution. This architecture splits the inference problem into a low-latency update that incorporates new measurements using just the most recent states (filter), and a high-latency update that is capable of closing long loops and smooths using all past states (smoother). This architecture employs the probabilistic graphical models of Factor Graphs, which allows the low-latency inference and high-latency inference to be viewed as sub-operations of a single optimization performed within a single graphical model. A specific factorization of the full joint density is employed that allows the different inference operations to be performed asynchronously while still recovering the optimal solution produced by a full batch optimization. Due to the real-time, asynchronous nature of this algorithm, updates to the state estimates from the high-latency smoother will naturally be delayed until the smoother calculations have completed. This architecture has been tested within a simulated aerial environment and on real data collected from an autonomous ground vehicle. In all cases, the concurrent architecture is shown to recover the full batch solution, even while updated state estimates are produced in real-time.

1 Introduction

A high-quality estimate of a vehicle's position, orientation, velocity, and other state variables is essential for the success of autonomous robotic deployments.

* Institute for Robotics and Intelligent Machines, Georgia Institute of Technology, Atlanta, GA 30332, USA.

† Field Robotics Center, Robotics Institute, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA 15213, USA.

¶ Computer Science and Artificial Intelligence Laboratory, MIT, Cambridge, MA 02139, USA.

Common low-level control objectives (e.g. maintaining a target speed or following a specified trajectory) are dependent on the system's estimated state. This is particularly true for aerial vehicles where the unstable dynamics of the system requires constant adjustment of the vehicle's orientation and thrust simply to stay aloft, necessitating high rate and high quality state estimates. Additionally, planning objectives and future control plans are often affected by the current state of the robot. Determining the shortest path to a goal location or planning obstacle-free trajectories are influenced by the global position of the vehicle. Further, additional robot objectives, such as mapping the environment, benefit from the availability of accurate, low-latency navigation estimates. To achieve high-quality global position estimates, either sources of global measurements are required (e.g. GPS) or loop closure constraints must be introduced. Loop closures constraints are generated whenever the robot identifies it is revisiting a previous location. Such constraints allow the estimator to correct accumulated drift, resulting in better global position estimates.

Conventional navigation systems are able to provide either constant-time updates or the ability to incorporate loop closures, but not both. Filtering techniques, such as the extended Kalman filter (EKF), maintain constant-time updates by marginalizing out all past variables. Variations on the EKF, such as augmented state filters or fixed-lag smoothers, maintain only a subset of the past states to improve estimation accuracy and allow the incorporation of measurements involving these additional states. This is commonly done in vision-aided systems, where several sightings of a distinctive environmental feature are required to properly triangulate the feature location. To maintain real-time performance, the number of additional states must remain small, making the incorporation of loop closures to arbitrary states in the past impossible. In contrast, full smoothing methods, also commonly referred to as full simultaneous localization and mapping (SLAM) or bundle adjustment (BA), seek to estimate the value of all past states within the optimization. This allows the simple incorporation of loop closure constraints that reference any past state, but at the expense of computational complexity. Even while leveraging the inherent sparsity of the problem and utilizing the latest algorithms for performing incremental inference, constant-time updates in the presence of loop closures cannot be achieved.

Parallelization is the key to providing both real-time state updates with the ability to incorporate arbitrary loop closures. The basic idea is to have a low-latency process update the current state at a high rate while a slower background process performs full bundle adjustment. The challenge to such systems lies in the information exchange or synchronization of the two processes in a probabilistically sound way. Several systems have explored this type of parallel processing recently, such as the parallel tracking and mapping architecture (PTAM) Klein and Murray (2007) and the dual-layer estimator Mourikis and Roumeliotis (2008). However, neither of these system successfully explored the synchronization issue. Within PTAM, the synchronization problem is avoided by simply replacing the current tracking state with information from the bundle adjustment, while the synchronization method used in the dual-layer estimator is unclear as presented. Further, both systems were developed specifically for vision applications and may not be easily generalized to other sensor combinations.

In this paper, we propose a general inference framework, first presented in Kaess et al. (2012b), for performing concurrent filtering and smoothing (CFS). Similar to full smoothing approaches, an estimate is generated for all past states, allowing the easy incorporation of arbitrary loop closure constraints. However, unlike conventional smoothing, the problem is partitioned into two sets: a small set of the most recent states that is updated in real-time, and a large set of the remaining states that is updated at a much slower rate. This partitioning supports independent updates of the two variable sets, thus enabling parallel processing. Additionally, methods for performing the synchronization of the two partitions in a probabilistically sound manner are defined, as are methods for changing the current partitioning. By viewing filtering and smoothing simply as inference operations performed within a single graphical model of the entire smoothing problem, the CFS system is able to recover the exact full smoothing solution while providing updated state estimates in real-time. The cost of this parallelization is only an additional delay between the time a loop closure constraint is identified and the time the effects of the loop closure are propagated to the current filtering solution.

In Section 2, we review existing algorithms for estimating the navigation state, including existing methods of parallel inference. In Section 3, we formulate the addressed problem. In Section 4, we provide a brief review to the factor graphs, a graphical framework for probabilistic inference. In Section 5, we introduce the basic concurrent filtering and smoothing architecture, examining the different operations and sources of delays. In Section 6, we present implementation specifics of the concurrent filtering and smoothing system, including the nonlinear optimization methods and additional techniques that guarantee constant-time updates and synchronizations. In Section 7, we compare results from the concurrent system with a similar quality fixed-lag smoother and full batch optimization. The system is tested using both a simulated aerial environment and with data collected from an autonomous ground vehicle. Finally, in Section 8, we present our general conclusions about the concurrent architecture and propose future work.

2 Related Work

Many filtering-based solutions to the navigation problem exist that are capable of integrating multiple sensors (Bar-Shalom and Li, 1995). Examples include GPS-aided (Farrell, 2008) and vision-aided (Mourikis and Roumeliotis, 2007; Zhu et al., 2007; Jones and Soatto, 2011) inertial navigation. A recent overview of filtering and related methods to multi-sensor fusion can be found in (Smith and Singh, 2006). In addition to the extended Kalman filter (EKF) and variants such as the unscented Kalman filter, solutions also include fixed-lag smoothing (Maybeck, 1979). While delayed and/or out-of-sequence data can also be handled by filters (Bar-Shalom, 2002; Shen et al., 2009; Zahng and Bar-Shalom, 2011), fixed-lag smoothing additionally allows re-linearization (Ranganathan et al., 2007; Lupton and Sukkarieh, 2012). However, filtering and fixed-lag smoothing based navigation solutions are not able to probabilistically include loop closure constraints derived from camera, laser or sonar data; the most useful loop closures reach far back in time to states that are no longer represented by such methods.

Integration of loop closures requires keeping past states in the estimation and can be solved efficiently by smoothing. Originally in the context of the simultaneous localization and mapping (SLAM) problem, a set of landmarks has been estimated using the EKF (Smith et al., 1988, 1990; Moutarlier and Chatila, 1989). However, the EKF solution quickly becomes expensive because the covariance matrix is dense, and the number of entries grows quadratically in the size of the state. It has been recognized that the dense correlations are caused by elimination of the trajectory (Eustice et al., 2006; Dellaert and Kaess, 2006). The problem can be overcome by keeping past robot poses in the estimation instead of just the most recent one, which is typically referred to as full SLAM or view-based SLAM. Such a solution was first proposed by Lu and Milios (1997) and further developed by many researchers including Thrun et al. (2005); Eustice et al. (2006); Dellaert and Kaess (2006); Mahon et al. (2008); Konolige et al. (2010); Grisetti et al. (2007). Even though the state space becomes larger by including the trajectory, the problem structure remains sparse and can be solved very efficiently by smoothing Dellaert and Kaess (2006); Kaess et al. (2012a).

View-based SLAM solutions typically only retain a sparse set of previous states and summarize the remaining information. For high rate inertial data, summarization is typically done by a separate filter, often performed on an inertial measurement unit (IMU). Marginalization is used to remove unneeded poses and landmarks, good examples are given in (Folkesson and Christensen, 2004; Konolige and Agrawal, 2008). And finally, the so-called pose graph formulation omits explicit estimation of landmark locations, and instead integrates relative constraints between pairs of poses. Despite all the reductions in complexity, smoothing solutions are not constant time when closing large loops and are therefore not directly suitable for navigation purposes.

Constant time operation in presence of loop closure observations has been experimentally demonstrated in adaptive relative bundle adjustment Sibley et al. (2009); Mei et al. (2011). By resorting to a relative formulation and adaptively identifying which camera poses to optimize, the authors have been able to reduce the number of optimized variables. However, this method results in an approximate solution to the overall least-squares problem and leverages specific properties of the structure-from-motion problem. In contrast, we propose a general approach for fusing information from any available sensors, that uses parallelization to guarantee constant time operation and produces the exact solution, with a certain time delay, as would have been obtained by a full nonlinear optimization.

Similar to our work, filtering and smoothing has been combined in a single optimization by Eustice et al. (2006). Their exactly sparse delayed state filter retains select states as part of the state estimate, allowing loop closures to be incorporated. The most recent state is updated in filter form, allowing integration of sensor data at 10Hz. Their key realization was the sparsity of the information form, and an approximate solution was used to provide real-time updates. Our solution instead provides an exact smoothing solution incorporating large numbers of states, while being able to process high rate sensor data on the filtering side with minimum delay. Simplification algorithms such as Vial et al. (2011) are orthogonal to this work and can be used to further improve performance of the smoother.

A navigation solution requires constant processing time, while loop closures require at least linear time in the size of the loop; hence parallelization is needed. Klein and Murray (2007) proposed parallel tracking and mapping (PTAM) of a single camera, where localization and map updates are performed in parallel. This differs from the navigation problem because filtering is replaced by repeated re-localization of the camera with respect to the current map. Still, the key idea of performing slower map updates in parallel is directly applicable to navigation. In addition, the bundle adjustment (BA) (Triggs et al., 2000) used to optimize the map is mathematically equivalent to the smoothing solution deployed in our work: the only difference is the specific structure of the problem, which is more sparse in our navigation case. The same parallelization is also used in more recent dense solutions such as dense tracking and mapping by Newcombe et al. (2011b) and KinectFusion by Newcombe et al. (2011a). However, while this separation into relocalization and mapping works well for tracking a camera, it does not allow probabilistic integration of inertial sensor data as achieved by our work.

Probably the closest work in navigation is the dual-layer estimator by Mourikis and Roumeliotis (2008) for combined camera and inertial estimation that uses an EKF for fast processing and BA for limiting linearization errors. However, they do not sufficiently explain how the BA result is fed back into the filter, in particular how consistent feedback is performed without rolling back the filter. Rolling back measurements is made unnecessary in our formulation, which casts the concurrent filtering and smoothing processes as a single, parallelized optimization problem.

Our novel solution combines filtering and smoothing within a single estimation framework, while formulating it in such a way that both are performed concurrently. Hence the filter operates at constant time when integrating new sensor data, while updates from the slower smoother are integrated on the fly once they become available.

3 Problem Formulation

Our goal, in the context of navigation problem, is to produce the best possible estimate of the current state θ_t given all of the available sensor measurements, Z

$$\theta_t^* = \arg\max_{\theta_t} \; p(\theta_t \mid Z). \tag{1}$$

This optimization is typically performed using a recursive filtering approach, such as the extended Kalman filter, where past states are marginalized out each time new states arrive:

$$p\left(\theta_{t+1} \mid Z\right) = \int_{\theta_t} p\left(\theta_t, \theta_{t+1} \mid Z\right). \tag{2}$$

Filtering methods have the advantage of providing constant time updates, and can therefore process sensor data at a high rate. However, measurements from navigation aiding sensors, such as cameras or laser scanners, often involve past states that are no longer available inside the filter. Incorporating measurements from these sensors therefore requires maintaining the distribution also over the involved past states, leading to a fix-lag smoother (or augmented-state filter) formulation. Calculating the best estimate of the current state θ_t then involves first calculating the posterior over the states within the smoothing lag and then marginalizing out the past states.

In the general case, however, some of the incoming measurements may involve *arbitrary* states from the past. These measurements, typically referred to as loop closures, encode essential information but cannot be incorporated into the posterior if some of the involved states are not within the smoothing lag. Recovering the maximum *a posteriori* (MAP) estimate requires therefore maintaining the distribution over *all* past variables, leading to full smoothing or full SLAM formulations.

In the full SLAM problem, the joint probability distribution, $p\left(\Theta \mid Z\right)$, over all variables Θ is maintained, and the MAP estimate of these states is given by

$$\Theta^* = \arg\max_{\Theta} \; p\left(\Theta \mid Z\right). \tag{3}$$

The MAP estimate of the current state θ_t can be directly taken from Θ^*. While full smoothing approaches are capable in producing a MAP estimate also in the presence of loop closure observations, the involved computational complexity increases as more states are added to the above optimization. Even when using efficient incremental smoothing approaches constant-time high-rate performance quickly becomes infeasible.

Our goal in this work is to develop an approach that is capable of both recovering the MAP estimate of the navigation state at constant-time and high-rate, and being able to incorporate measurements involving arbitrary past states. Our general approach to address this problem is to factorize the full joint pdf, $p\left(\Theta \mid Z\right)$, into two conditionally independent components, one of which is designed to process high-rate observations, and the other to perform full smoothing in a parallel process. We therefore refer to these components as filter and smoother, respectively, and show in the sequel how operations between these components can be parallelized.

4 Graphical Inference Models

The concept of concurrent filtering and smoothing is most easily understood using graphical models. For probabilistic inference, the language of *factor graphs* has been shown to be convenient and appropriate Kschischang et al. (2001). The following section offers a brief review of using factor graphs for probabilistic inference, including the closely related data structures of Bayes nets and Bayes trees. For further reading, see Kaess et al. (2010a).

4.1 Factor Graphs

The full joint probability, $p\left(\Theta \mid Z\right)$, can be represented graphically using a *factor graph.* A factor graph is a bipartite graph $G = (\mathcal{F}, \Theta, \mathcal{E})$ with two node types: *factor nodes* $f_i \in \mathcal{F}$ and *variable nodes* $\theta_j \in \Theta$. An edge $e_{ij} \in \mathcal{E}$ exists if and only if factor f_i involves state variable θ_j. The factor graph G defines the factorization of a function $f(\Theta)$ as:

$$G = f\left(\Theta\right) = \prod_i f_i\left(\Theta_i\right) \tag{4}$$

where Θ_i is the set of all state variables θ_j involved in factor f_i, and independent relationships are encoded by edges e_{ij}. Clearly, a factor graph, G, can be used to represent the full joint density by representing each measurement likelihood, $p\left(z_i \mid \Theta_i\right)$, by a factor, f_i. Figure 1 illustrates an example factor graph consisting of five robot states and one landmark. Prior information on the first state is provided by factor p_1, odometry constraints between consecutive poses are encoded in factors u_i, camera projection factors v_i connect between the landmark and the states that observed it, and a loop closure constraint c_1 has been added between state x_1 and x_4. The graphical model captures the structure of the problem while abstracting away the details of the individual measurement equations. For example, if an inertial strapdown system was used to calculate sequential pose changes instead of wheel odometry, the measurement functions represented by the factors u_i would be considerably more complicated, but the factor graph representation of the full joint would be identical.

4.2 Inference

Finding the maximum *a posteriori* (MAP) estimate, Θ^*, from a factor graph proceeds in two phases: elimination and back-substitution Heggernes and Matstoms (1996). In the first step, Gaussian variable elimination is performed, whereby each variable is rewritten as a function of the remaining variables and then removed from the graph. To eliminate a variable θ_j from the factor graph, we first form the joint density $f_{joint}\left(\theta_j, N\left(\theta_j\right)\right)$, where $N\left(\theta_j\right)$ is defined as the open neighborhood of θ_j. This neighborhood consists of all factors adjacent to θ_j (connected by an edge), not including the variable θ_j itself. The variable θ_j is known as the frontal variable and the variables $N\left(\theta_j\right)$ are referred to as separator variables. Applying the chain rule, we obtain a conditional density

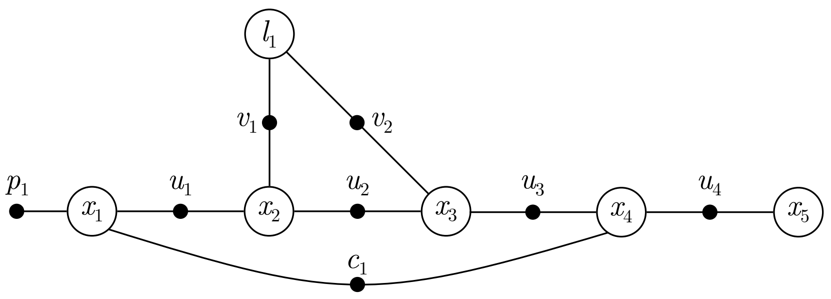


Figure 1: An example factor graph consisting of multiple robot states and two landmarks. Initial conditions on the first state are represented by p_1. Odometry factors, u_i, connect sequential robot states, while camera projection factors, v_i, connect between landmarks and the states that observed them. Finally, a loop closure constraint, c_1, exists between x_1 and x_4.

$p(\theta_j \mid N(\theta_j))$ and a new factor $f_{new}(N(\theta_j))$ that represents the marginal density on the separator variables $N(\theta_j)$ provided by the adjacent factors. After all variables are eliminated, the produced conditional densities form a new factored joint density, (5), known as a Bayes net.

$$p(\Theta) = \prod_j p(\theta_j \mid N(\theta_j)) \tag{5}$$

For the example in Figure 1, if we wish to eliminate variable x_3, we first form the joint density from all factors adjacent to x_3. In this case, the adjacent factors are u_2, u_3, and v_2. The separator variables are all variables adjacent to these factors, except for x_3: $N(x_3) = \{x_2, x_4, l_1\}$. After elimination, the variable x_3 and adjacent factors $\{u_2, u_3, v_2\}$ are removed from the graph, the calculated conditional density $p(x_3 \mid x_2, x_4, l_1)$ is stored in the Bayes net, and a new factor on the separator $f_{new}(x_2, x_4, l_1)$ is added to the remaining factor graph.

Before performing the elimination, an elimination order must be selected. While any order will ultimately produce an identical MAP estimate, the particular ordering selected affects the number of separator variables in each joint density and hence affects the computational complexity of the elimination step. Finding the best elimination order is NP-complete, but many heuristics exist for selecting good variable orderings, such as those based on minimum degree Heggernes and Matstoms (1996); Davis et al. (2004). It should be noted that the elimination order also affects the graph structure of the Bayes net. Figure 2 illustrates two possible structures produced by eliminating the factor graph from Figure 1 using different orderings.

After the elimination step is complete, back-substitution is used to obtain the MAP estimate of each variable. As seen in Figure 2, the last variable eliminated does not depend on any other variables. Thus, the MAP estimate of the last variable can be directly extracted from the Bayes net. By proceeding in reverse elimination order, the values of all the separator variables for each conditional

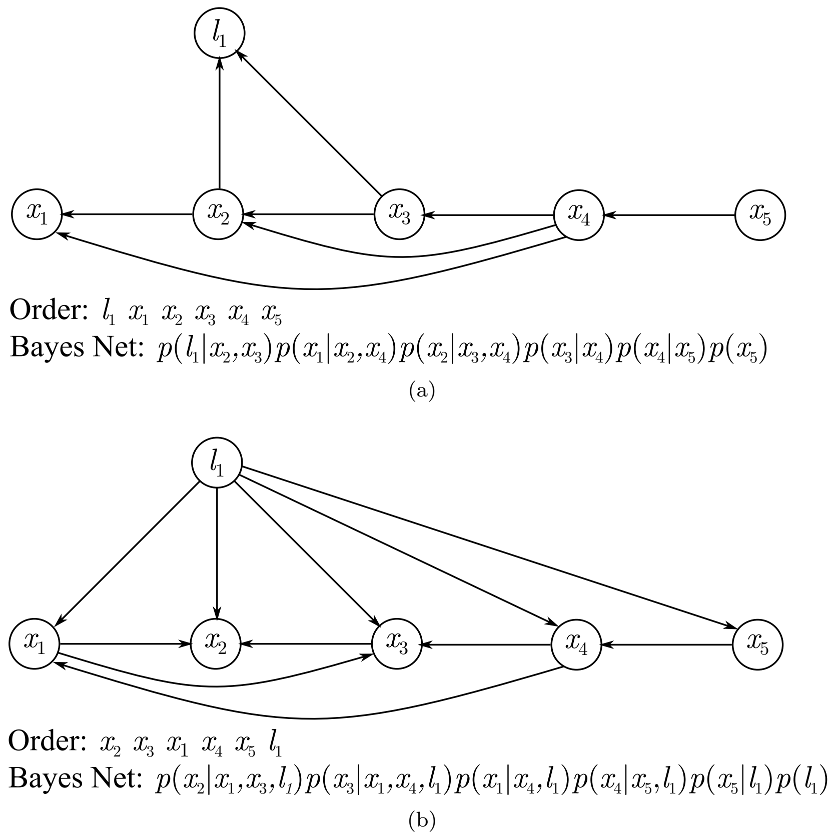

l_1
x_1
x_2
x_3
x_4
x_5
Order: $l_1\ x_1\ x_2\ x_3\ x_4\ x_5$
Bayes Net: $p(l_1|x_2,x_3)p(x_1|x_2,x_4)p(x_2|x_3,x_4)p(x_3|x_4)p(x_4|x_5)p(x_5)$
(a)
l_1
x_1
x_2
x_3
x_4
x_5
Order: $x_2\ x_3\ x_1\ x_4\ x_5\ l_1$
Bayes Net: $p(x_2|x_1,x_3,l_1)p(x_3|x_1,x_4,l_1)p(x_1|x_4,l_1)p(x_4|x_5,l_1)p(x_5|l_1)p(l_1)$
(b)

Figure 2: Example Bayes net structures produced by eliminating the same factor graph from Figure 1 using different variable orderings.

will always be available from the previous steps, allowing the estimate for the current frontal variable to be computed.

Topologically, the Bayes net is a chordal directed acyclic graph. By identifying cliques (groups of fully connected variables), the Bayes net may be rewritten as a Bayes tree. For full details on of the clique-finding algorithm, see Kaess et al. (2010a,b). Within the Bayes tree, each node represents the conditional density of the clique variables, Θ_j, given all of the separators, $N(\Theta_j)$:

$$p(\Theta) = \prod_j p(\Theta_j \mid N(\Theta_j)). \tag{6}$$

During elimination, the leaves of the tree are built first, and factors on the separator variables are passed up the tree to their parents. Back-substitution then proceeds top-down from the root clique, which is eliminated last, as it has no external dependencies. The solution of the frontal variables of the parent clique are passed down the tree to the children, which are guaranteed to depend only on the frontal variables of their ancestors. Figure 3 shows the Bayes tree representation of the Bayes nets shown in Figure 2. Like the Bayes net, the structure of the Bayes tree is affected by the selected variable ordering. While the Bayes net and Bayes tree representations are interchangeable, modeling the inference using a tree structure is often more convenient and intuitive: elimination passes information up the tree, while back-substitution propagates information down the tree.

5 Concurrent Filtering and Smoothing

The CFS algorithm provides a parallel architecture that enables both fast updates of the current solution while recovering the optimal solution of the full joint density even in the presence of loop closures. This is accomplished by applying a specific factorization of the full joint density that segments the problem into a small inference operation over the most recent states (i.e. a filter) and a larger inference problem over the remaining states (i.e. full smoothing). In the following sections, we explain this factorization in detail, showing how this factorization allows independent asynchronous updates of the filter and smoother segments. Methods for periodically synchronizing the two segments in a probabilistically consistent way are explained, and methods for changing the current segmentation are presented. The cost of exploiting this factorization for parallel processing are delays between the time a loop closure constraint is added to the smoother and the time this constraint impacts the current filtering solution. The source and nature of these delays are also discussed in the following.

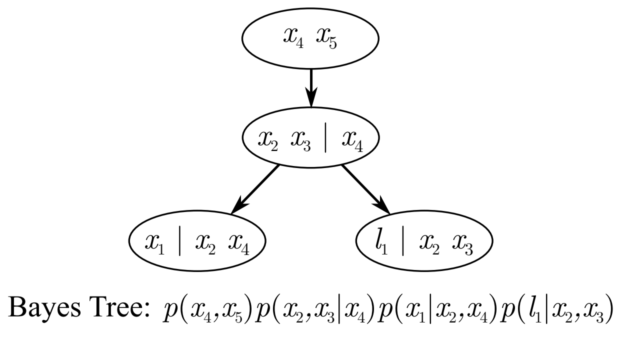
The entire algorithm is summarized in Algorithm 1, and explained in this section in detail.

5.1 Factorization

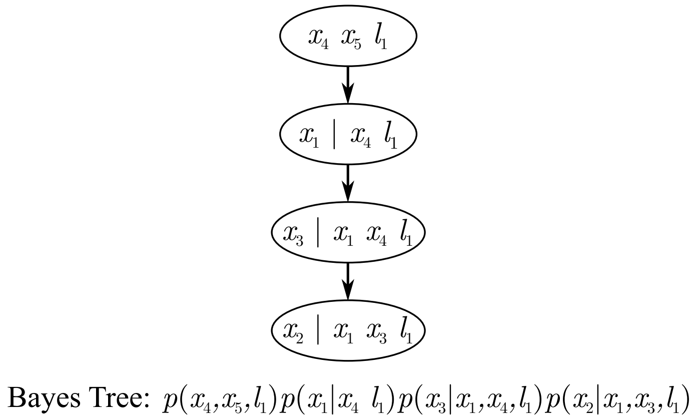
The CFS architecture chooses to factorize the full joint density into three groups: a small number of the most recent states referred to as the filter states, a large group of past states referred to as smoother states, and a small number of separator states that make the filter and smoother conditionally independent. This factorization is shown in (7).

$$p(\Theta) = p(\Theta_S \mid \Theta_{sep})\, p(\Theta_{sep})\, p(\Theta_F \mid \Theta_{sep}) \tag{7}$$

where Θ is the set of all state variables, Θ_F is the set of filter states, Θ_S is the set of smoother states, and Θ_{sep} is the set of separator states. This factorization accounts for all states such that $\Theta = \{\Theta_S, \Theta_{sep}, \Theta_F\}$. This factorization may be viewed as a generalized Bayes tree, as shown in Figure 4, where the large nodes depicted in the figure may themselves contain subtrees. Since the elimination step proceeds from the leaves of the tree to the root, as discussed in Section 4,



(a)



(b)

Figure 3: Example Bayes tree structures produced by eliminating the same factor graph from Figure 1 using the two different variable orderings from Figure 2.

```
while $true$ do
    Input: New filter measurements $Z'_F$ and smoother measurements $Z'_S$
    concurrently:
        update filter:
            /* Continuously update filter in real-time while
               smoother computes a loop closure update:        */
            while $Smoother\ is\ busy$ do
                Input: Incoming filter measurements $Z'_F$ and
                       measurement density $\tilde{p}(Z'_F|\Theta_{sep}, \Theta_F)$
                Input: Separator information $\tilde{p}(Z_F|\Theta_{sep})$ from previous
                       filter updates
                Input: Filter density $p(\Theta_F|\Theta_{sep}, Z_F)$
                Factor $\tilde{p}(Z_F|\Theta_{sep})p(\Theta_F|\Theta_{sep}, Z_F)\tilde{p}(Z'_F|\Theta_{sep}, \Theta_F)$ into
                $\tilde{p}(Z_F, Z'_F|\Theta_{sep})p(\Theta_F|\Theta_{sep}, Z_F, Z'_F)$.
                Output: $\tilde{p}(Z_F|\Theta_{sep}) \leftarrow \tilde{p}(Z_F, Z'_F|\Theta_{sep})$
                Output: $p(\Theta_F|\Theta_{sep}, Z_F) \leftarrow p(\Theta_F|\Theta_{sep}, Z_F, Z'_F)$
            end
        end update filter
        update smoother:
            /* Computationally-intensive smoother update, e.g.
               from visual loop closures                        */
            Input: Incoming smoother measurements $Z'_S$ and
                   measurement density $\tilde{p}(Z'_S|\Theta_{sep}, \Theta_S)$
            Input: Separator information $\tilde{p}(Z_S|\Theta_{sep})$ from previous
                   smoother updates
            Input: Smoother density $p(\Theta_S|\Theta_{sep}, Z_S)$
            Factor $\tilde{p}(Z_S|\Theta_{sep})p(\Theta_S|\Theta_{sep}, Z_S)\tilde{p}(Z'_S|\Theta_{sep}, \Theta_S)$ into
            $\tilde{p}(Z_S, Z'_S|\Theta_{sep})p(\Theta_S|\Theta_{sep}, Z_S, Z'_S)$.
            Output: $\tilde{p}(Z_S|\Theta_{sep}) \leftarrow \tilde{p}(Z_S, Z'_S|\Theta_{sep})$
            Output: $p(\Theta_S|\Theta_{sep}, Z_S) \leftarrow p(\Theta_S|\Theta_{sep}, Z_S, Z'_S)$
        end update smoother
    end concurrently

    /* Synchronize filter and smoother:                         */
    Filter sends $\tilde{p}(Z_F|\Theta_{sep})$ to smoother.
    Smoother sends $\tilde{p}(Z_S|\Theta_{sep})$ to filter.

    /* Advance separator:                                        */
    Factor $p(\Theta_S|\Theta_{sep}, Z)p(\Theta_{sep}|Z)p(\Theta_F|\Theta_{sep}, Z)$ into
    $p(\Theta'_S|\Theta'_{sep}, Z)p(\Theta'_{sep}|Z)p(\Theta'_F|\Theta'_{sep}, Z)$, where $\Theta'_S$, $\Theta'_{sep}$, and $\Theta'_F$ are
    the modified variables due to separator advancement.
end
```

Algorithm 1: Concurrent filtering and smoothing

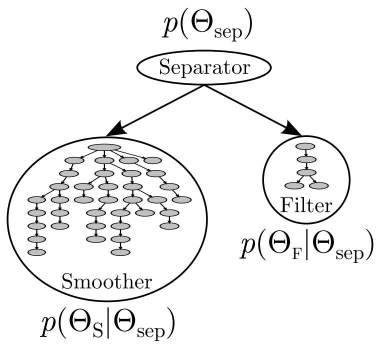

$p(\Theta_{\text{sep}})$
Separator
Smoother
$p(\Theta_{\text{S}}|\Theta_{\text{sep}})$
Filter
$p(\Theta_{\text{F}}|\Theta_{\text{sep}})$

Figure 4: The CFS factorization illustrated as a Bayes tree. The tree structure illustrates the conditional independence of the filter and smoother states, given the separator. The large nodes in the figure indicate that the filter and smoother may themselves contain significant sub-trees.

the tree representation clearly illustrates the independence of the filter on the smoother variables. This will be further discussed in Section 5.2.

The proposed factorization is general and can be applied to any system. The separator is defined as any set of states that effectively disconnects the filter states from the rest of the system. For any system, a minimal set of separator states can be identified based on the graph connectivity. Assuming the set of desired filter states is a design parameter and is therefore known, the set of separator states is simply the open neighborhood of the filter states, $N(\Theta_F)$, where an open neighborhood is defined as the set of variables adjacent to any variable in Θ_F, not including the variables in the set Θ_F. The smoother states are then the remaining states, $\Theta_S = \Theta \setminus \{\Theta_{sep} \cup \Theta_F\}$. As a small example, consider the sample factor graph from Figure 1. If it is desired that the filter consists of the last two states, $\Theta_F = \{x_4, x_5\}$, then the open neighborhood of Θ_F is $\Theta_{sep} = \{x_1, x_3\}$. This is demonstrated in Figure 5, where the desired filter nodes are colored in red. Any edge touching at least one filter node has been drawn using a dashed line. The open neighborhood of Θ_F is any node connected to a dashed edge, not including the Θ_F nodes themselves, i.e. $\{x_1, x_3\}$. This same set may also be discovered using the adjacency matrix from graph theory. The adjacency matrix is an $n \times n$ matrix, where each row and each column represents one of the n system variables. The adjacency matrix is constructed by placing non-zero entries in the elements corresponding to variables connected by edges. For undirected graphs, such as factor graphs, the adjacency matrix is symmetric. The adjacency matrix for the example factor graph from Figure 1 is shown in Figure 6. The filter variables $\{x_4, x_5\}$ correspond to the right-most and bottom-most rows, as indicated. The 2×2 block in the lower-right describes the inter-connections of the variables *within* the filter set, while the entries to the left (or above) this block describe the connections to variables *outside* the filter set. The indicated variables in this left (or upper) block are the neighbors of the filter set, $N(x_4, x_5) = \{x_1, x_3\}$. This is the same separator

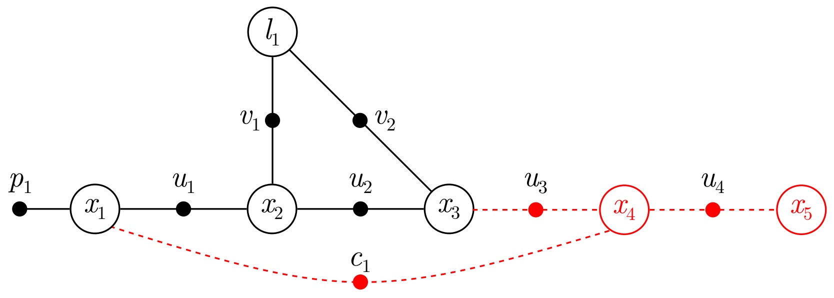


Figure 5: An illustration of the neighborhood of $\{x_4, x_5\}$. All adjacent edges are drawn using red dashed lines. The neighbors include any node connected to a dashed edge, not including the variable set itself. In this case, $\{x_1, x_3\}$.

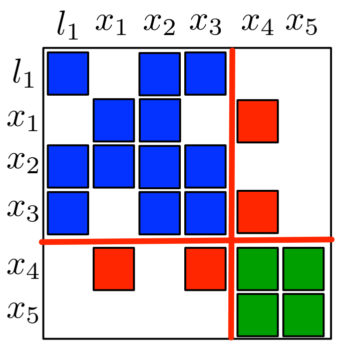


Figure 6: The adjacency matrix for the example factor graph from Figure 1. If the filter consists of variables $\{x_4, x_5\}$ (green), then the separator will be $\{x_1, x_3\}$ (red).

set found through the graphical method described in Figure 5.

A specific factorization may alternatively be viewed as a particular elimination order. As discussed in Section 4.2, any elimination order ultimately produces identical MAP estimates, but the selection of the elimination order affects the topology of the resulting Bayes net and Bayes tree, which indirectly impacts the computational performance of the inference operations. In order to produce the desired topology, shown in Figure 4, the separator states must be eliminated last, placing them at the root of the Bayes tree. The computational complexity of sparse elimination is bounded by the size of the largest clique in the Bayes tree, which is often the root clique. Since the proposed factorization in essence defines the root clique to be the separator, an investigation of the separator size is warranted. In the general case, the induced separator between an arbitrary set of states may be any size. Instead, we will examine two common cases in navigation.

Relative-Pose Aided Inertial Navigation An IMU is used to predict the next state given the previous state and a set of IMU measurements. The IMU factors are augmented with a set of factors from aiding sensors. These sensor factors are generally in the form of direct measurements on some of the state variables (i.e. GPS, compass, altimeter) or relative measurements between consecutive time instances (i.e. lidar scan matching, stereo visual odometry). An example of such a factor graph is shown in Figure 7, with the desired filter states of $\{x_6, x_7, x_8\}$. Menger's Theorem Menger (1927); Aharoni and Berger (2009) in graph theory states that the minimum size of a separator dividing a graph into two disconnected sets is equal to the maximum number of vertex-disjoint paths between the two sets. The decomposition of the factor graph into disjoint paths by sensor is also shown in Figure 7. The path from the older smoother states to the newer filter states using the IMU factors clearly involves variable nodes from the other sensor measurements and, hence, does not produce a vertex-disjoint path. Also, the absolute measurements, represented as unary factors on single states, do not induce any paths between the variable sets. Thus, assuming that the filter set is defined to be a continuous set of the most recent states, the number of disjoint paths is equal to the number of non-IMU relative measurement sensors. Therefore, in the example shown in Figure 7 the minimum size of the separator is 2: $\Theta_{sep} = \{x_4, x_5\}$.

Vision-Aided Inertial Navigation In the particular case of vision, it is also common that additional landmark variables are added to the system with individual factors connecting a single landmark to a single state. Figure 8 shows a simple example factor graph with IMU measurements and two landmark states. The desired filter states are again $\{x_6, x_7, x_8\}$. When landmark states are incorporated, the decomposition of the graph into disjoint paths is complicated, but it is bounded by the number of tracked landmarks plus the number of relative-pose sensors. In the example shown in Figure 8 the minimum size of the separator is 3: $\Theta_{sep} = \{x_5, l_1, l_2\}$.

5.2 Concurrent Incremental Filter and Smoother Updates

Using the suggested factorization it is possible to update the filter branch and the smoother branch concurrently and still recover a MAP state estimate. First, let us define the joint densities represented by each branch of the full joint density factored using the suggested concurrent factorization:

$$\begin{aligned} p_{filter}\left(\Theta_{sep}, \Theta_F | Z\right) &= p\left(\Theta_{sep}|Z\right) p\left(\Theta_F \mid \Theta_{sep}, Z\right) \\ p_{smoother}\left(\Theta_S, \Theta_{sep}|Z\right) &= p\left(\Theta_S \mid \Theta_{sep}, Z\right) p\left(\Theta_{sep}|Z\right), \end{aligned} \tag{8}$$

where Z represents all the available measurements.

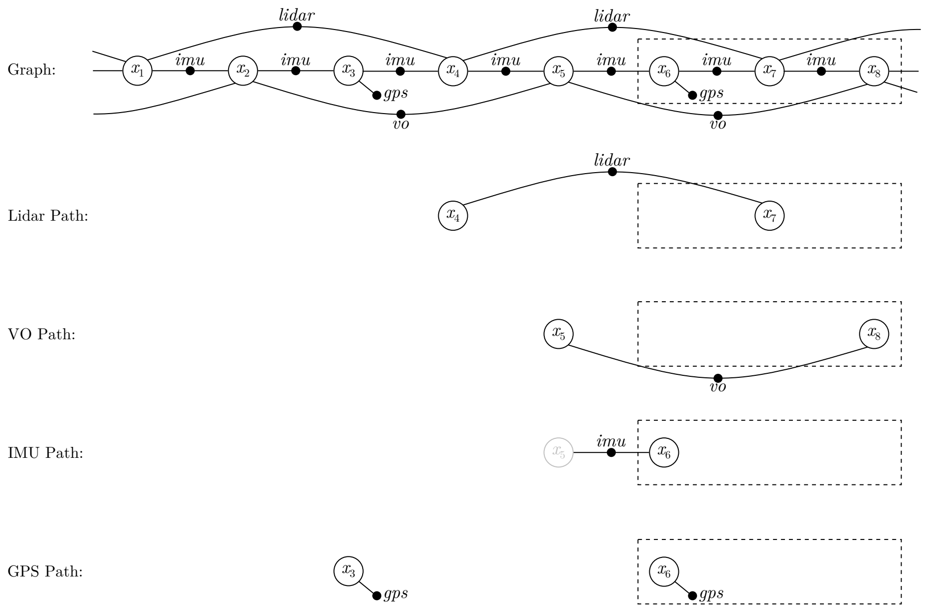


Figure 7: An example factor graph consisting of relative pose measurements (Lidar, VO, IMU) and direct state measurements (GPS). The desired filter states are $\{x_6, x_7, x_8\}$. The individual paths to one of the filter states are then shown per sensor. The Lidar and VO paths are shown to be disjoint (no variables in common). However, the IMU path shares variables with the other sensors, and the GPS sensor measurements do not form a path at all.

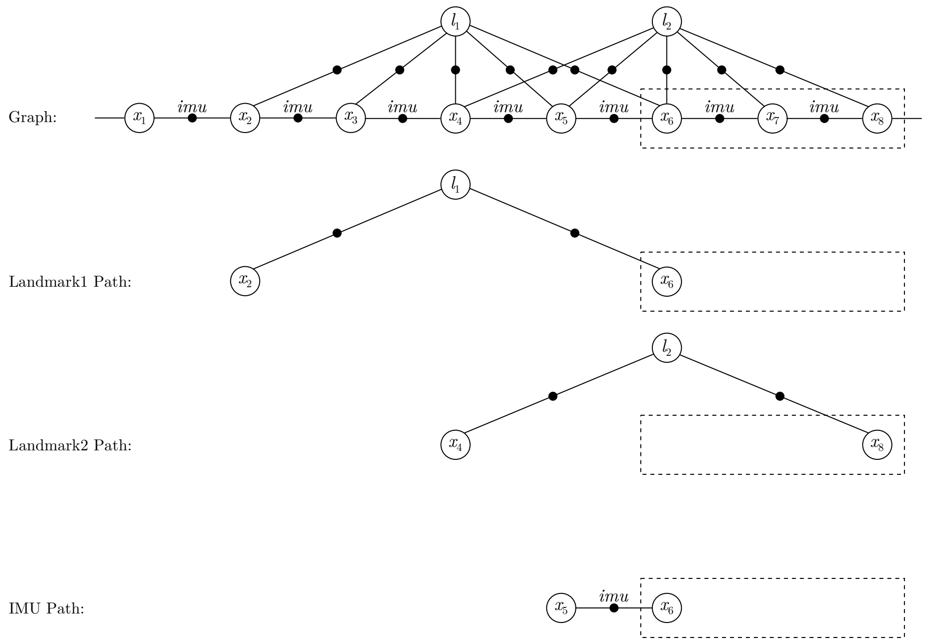


Figure 8: An example factor graph consisting of IMU measurements and observations of two landmarks. The desired filter states are $\{x_6, x_7, x_8\}$. Individual paths to one of the filter states are shown per landmark. In this case, there are many possible paths through each landmark, but all possible paths share the landmark variable.

Note that the joint density of both branches involve the marginal density $p\left(\Theta_{sep}|Z\right)$. This is the common separator, or the root of the Bayes tree shown in Figure 4, that is shared by both branches. During a full batch optimization, the separator would normally be updated with new information propagated upward from both branches. In order to enable parallel operation, both the filter and the smoother must maintain a local copy of the separator that is updated independently. A method is needed to separate the updates by source (filter or smoother) so that the proper updated information may later be exchanged between branches.

We can break up the definition of $p\left(\Theta_{sep}|Z\right)$ by separating the measurements Z into two groups: smoother measurements Z_S and filter measurements Z_F such that $Z = \{Z_S, Z_F\}$. Under this measurement segmentation, the joint density on the separator variables Θ_{sep} can itself be factored as:

$$p\left(\Theta_{sep} \mid Z\right) = p\left(Z_S \mid \Theta_{sep}\right) p\left(Z_F \mid \Theta_{sep}\right) p\left(\Theta_{sep}\right) \tag{9}$$

where each term in the separator factorization is obtained by marginalizing out all of the variables not included in Θ_{sep} from the smoother measurements or filter measurements respectively. Additionally, in the above equation we make the standard assumption of any two measurements being statistically independent of each other.

For simplicity we include the prior information $p\left(\Theta_{sep}\right)$ within the first two terms in Eq. (9) by appropriately splitting up $p\left(\Theta_{sep}\right)$ between these two terms. Introducing the notation $\tilde{p}\left(.\right)$ to represent these updated terms, we rewrite Eq. (9) as

$$p\left(\Theta_{sep} \mid Z\right) = \tilde{p}\left(Z_S \mid \Theta_{sep}\right) \tilde{p}\left(Z_F \mid \Theta_{sep}\right). \tag{10}$$

If we assume these marginal densities are available to both the smoother and the filter, then we can rewrite the joint density encoded by each branch as:

$$\begin{aligned} p_{filter}\left(\Theta_{sep}, \Theta_F \mid Z_S, Z_F\right) &= \tilde{p}\left(Z_S \mid \Theta_{sep}\right) \tilde{p}\left(Z_F \mid \Theta_{sep}\right) p\left(\Theta_F \mid \Theta_{sep}, Z_F\right) \\ p_{smoother}\left(\Theta_S, \Theta_{sep} \mid Z_S, Z_F\right) &= p\left(\Theta_S \mid \Theta_{sep}, Z_S\right) \tilde{p}\left(Z_S \mid \Theta_{sep}\right) \tilde{p}\left(Z_F \mid \Theta_{sep}\right) \end{aligned} \tag{11}$$

This factorization divides the terms into a set that only depends on the measurements assigned to the local branch (e.g. $\tilde{p}\left(Z_F \mid \Theta_{sep}\right)$ and $p\left(\Theta_F \mid \Theta_{sep}, Z_F\right)$ only involve filter measurements Z_F) and a set of terms that are assumed to be provided (e.g. $\tilde{p}\left(Z_S \mid \Theta_{sep}\right)$ was provided to the filter branch from the smoother). Thus, updating a single branch with a new factor is a local operation, independent of the other branch.

For example, if new measurements $Z_{F'}$ arrive involving the filter variables, then the set of filter measurements are augmented with these new measurements. To recompute the full joint, we only need to update the terms involving Z_F, namely $p\left(\Theta_F \mid \Theta_{sep}, Z_F\right)$ and $\tilde{p}\left(Z_F \mid \Theta_{sep}\right)$, with the new measurements (line 9 in Algorithm 1). Once complete, the filter and the full density can be assembled by reusing the previously calculated densities provided by the smoother branch, as in:

$$p_{filter}\left(\Theta_{sep}, \Theta_F \mid Z_S, Z_F, Z_{F'}\right) = \\ \tilde{p}\left(Z_S \mid \Theta_{sep}\right) \tilde{p}\left(Z_F, Z_{F'} \mid \Theta_{sep}\right) p\left(\Theta_F \mid \Theta_{sep}, Z_F, Z_{F'}\right)$$

$$p_{joint}\left(\Theta_S, \Theta_{sep}, \Theta_F \mid Z_S, Z_F, Z_{F'}\right) = \\ p\left(\Theta_S \mid \Theta_{sep}, Z_S\right) \tilde{p}\left(Z_S \mid \Theta_{sep}\right) \tilde{p}\left(Z_F, Z_{F'} \mid \Theta_{sep}\right) p\left(\Theta_F \mid \Theta_{sep}, Z_F, Z_{F'}\right) \quad (12)$$

The product $\tilde{p}\left(Z_S \mid \Theta_{sep}\right) \tilde{p}\left(Z_F, Z_{F'} \mid \Theta_{sep}\right)$ is equivalent to the $p\left(\Theta_{sep}\right)$ term in the original CFS factorization (7) as it defines the marginal on the separator given all of the measurements, $Z = \{Z_S, Z_F, Z_{F'}\}$. Despite the fact that only a subset of the factorization is recalculated, the exact joint density is recovered. Further, a single branch (filter or smoother) may be updated multiple times while still recovering the exact solution without requiring any communication with the other branch.

The branches may also be updated concurrently, adding new factors to the smoother branch while performing multiple updates to the filter branch. During concurrent updates, the filter branch is updated with new filter measurements *assuming the smoother branch remains* unchanged (lines 4-13 in Algorithm 1), while the smoother branch is updated with new measurements *assuming the filter branch remains unchanged* (lines 14-21). Let new filter measurements be denoted $Z_{F'}$ and new smoother measurements denoted $Z_{S'}$, then the resulting updated densities are:

$$p_{filter}\left(\Theta_{sep}, \Theta_F \mid Z_S, Z_F, Z_{F'}\right) = \\ \tilde{p}\left(Z_S \mid \Theta_{sep}\right) \tilde{p}\left(Z_F, Z_{F'} \mid \Theta_{sep}\right) p\left(\Theta_F \mid \Theta_{sep}, Z_F, Z_{F'}\right)$$

$$p_{smoother}\left(\Theta_S, \Theta_{sep} \mid Z_S, Z_{S'}, Z_F\right) = \\ p\left(\Theta_S \mid \Theta_{sep}, Z_S, Z_{S'}\right) \tilde{p}\left(Z_S, Z_{S'} \mid \Theta_{sep}\right) \tilde{p}\left(Z_F \mid \Theta_{sep}\right) \quad (13)$$

Each branch still obtains an exact solution, but the solutions are to problems with different measurement sets: $\{Z_S, Z_F, Z_{F'}\}$ for the filter versus $\{Z_S, Z_{S'}, Z_F\}$ for the smoother. We refer to this situation as being *unsynchronized.* Although the system is unsynchronized, it is important to note that the filter branch continues to incorporate new measurements, producing the exact MAP estimate of its particular subproblem (i.e. it does not include the most recent smoother measurements), at the same time the smoother is performing its update in the background. To recover the full joint density given *all* measurements, the two branches must first be synchronized, as described in the next section.

The computational complexity of the concurrent update step depends on the inference algorithm selected for the filter and smoother. Although the CFS architecture does not require a specific inference strategy, the goals of the CFS system do suggest specific choices. For the filter, an EKF variant would be a reasonable choice, providing fixed-time updates of complexity $O\left(n^3\right)$, where n is the number of states in the filter branch. If vision information is used, an augmented state filter or fixed-lag smoother might be employed that allows the linearization point of landmark variables to improve over time. For the smoother, the large number of states involved will generally make the added complexity of sparse linear algebra methods worthwhile. For a navigation system with a sparse graph and no loop closures, full smoothing methods can approach $O\left(m\right)$ complexity, where m is the number of states in the smoother branch. Additional performance gains could be achieved by employing incremental techniques, such as iSAM2 Kaess et al. (2011, 2012a). However, in the presence of loop closure constraints, the actual complexity of full smoothing methods, batch or incremental, is scenario dependent. As the synchronization procedure, discussed in the next section, can be performed only after the inference in smoother and filter branches is complete, the navigation solution is updated with loop closure information with a certain time delay. We elaborate on the root causes for this time delay in Section 5.5.

5.3 Synchronization

Periodic *synchronization* exchanges updated information between the filter and smoother after concurrent updates, thus recovering the optimal state estimate given *all* of the measurements. As discussed in the previous section, updating the smoother and filter in parallel leads to a mismatch in the measurements used by each branch. For example, when a loop closure constraint is added to the smoother, it can take several seconds for the smoother to incorporate the change across all past states. In the mean time, the filter continues to incorporate new high-speed measurements into the navigation solution. Once the smoother completes its optimization, the smoother does not include all of the recent high-speed measurements, nor does the filter include the loop closure information. The exchange of this updated information is referred to as synchronization.

Let Z_S be all of the measurements contained within the smoother branch, and Z_F be all of the measurements contained within the filter branch before any concurrent updates, such that $Z = Z_S \cup Z_F$. The two branches are synchronized as they both use identical measurement sets:

$$\begin{aligned} p_{filter}\left(\Theta_{sep}, \Theta_F \mid Z_S, Z_F\right) &= \tilde{p}\left(Z_S \mid \Theta_{sep}\right) \tilde{p}\left(Z_F \mid \Theta_{sep}\right) p\left(\Theta_F \mid \Theta_{sep}, Z_F\right) \\ p_{smoother}\left(\Theta_S, \Theta_{sep} \mid Z_S, Z_F\right) &= p\left(\Theta_S \mid \Theta_{sep}, Z_S\right) \tilde{p}\left(Z_S \mid \Theta_{sep}\right) \tilde{p}\left(Z_F \mid \Theta_{sep}\right) \end{aligned} \tag{14}$$

where the term $\tilde{p}\left(Z_S \mid \Theta_{sep}\right)$ is the marginal density on the separator variables calculated by marginalizing the smoother variables Θ_S from the smoother measurements Z_S, and similarly $\tilde{p}\left(Z_F \mid \Theta_{sep}\right)$ is the marginal density on the separator variables calculated by marginalizing the filter variables Θ_F from the filter measurements Z_F.

During the concurrent update phase, additional measurements are incorporated into the smoother and the filter. Let $Z_{S'}$ and $Z_{F'}$ be all of the new measurements received by the smoother and filter respectively. After concurrent updates, the system is in an unsynchronized state, where neither the filter density nor the smoother density contains the information from all of the received measurements:

$$p_{filter}\left(\Theta_{sep}, \Theta_F \mid Z_S, Z_F, Z_{F'}\right) = \tilde{p}\left(Z_S \mid \Theta_{sep}\right) \tilde{p}\left(Z_F, Z_{F'} \mid \Theta_{sep}\right) p\left(\Theta_F \mid \Theta_{sep}, Z_F, Z_{F'}\right)$$

$$p_{smoother}\left(\Theta_S, \Theta_{sep} \mid Z_S, Z_{S'}, Z_F\right) = p\left(\Theta_S \mid \Theta_{sep}, Z_S, Z_{S'}\right) \tilde{p}\left(Z_S, Z_{S'} \mid \Theta_{sep}\right) \tilde{p}\left(Z_F \mid \Theta_{sep}\right) \quad (15)$$

This process was described in detail in Section 5.2. Each branch contains an outdated marginal factor on the separator variables, $\tilde{p}\left(Z_S \mid \Theta_{sep}\right)$ or $\tilde{p}\left(Z_F \mid \Theta_{sep}\right)$, from the other branch. Thus, synchronizing the two branches merely requires updating the cached marginal on the separator with the updated version from the other branch: $\tilde{p}\left(Z_F, Z_{F'} \mid \Theta_{sep}\right)$ is sent from the filter to the smoother, while $\tilde{p}\left(Z_S, Z_{S'} \mid \Theta_{sep}\right)$ is sent from the smoother to the filter (lines 24-25 in Algorithm 1). After synchronization, both branches again use identical measurements:

$$p_{filter}\left(\Theta_{sep}, \Theta_F \mid Z_S, Z_{S'}, Z_F, Z_{F'}\right) = \tilde{p}\left(Z_S, Z_{S'} \mid \Theta_{sep}\right) \tilde{p}\left(Z_F, Z_{F'} \mid \Theta_{sep}\right) p\left(\Theta_F \mid \Theta_{sep}, Z_F, Z_{F'}\right)$$

$$p_{smoother}\left(\Theta_S, \Theta_{sep} \mid Z_S, Z_{S'}, Z_F, Z_{F'}\right) = p\left(\Theta_S \mid \Theta_{sep}, Z_S, Z_{S'}\right) \tilde{p}\left(Z_S, Z_{S'} \mid \Theta_{sep}\right) \tilde{p}\left(Z_F, Z_{F'} \mid \Theta_{sep}\right) \quad (16)$$

and the full joint can be recovered using information from both branches:

$$p_{joint}\left(\Theta_S, \Theta_{sep}, \Theta_F \mid Z_S, Z_{S'}, Z_F, Z_{F'}\right) = p\left(\Theta_S \mid \Theta_{sep}, Z_S, Z_{S'}\right) \tilde{p}\left(Z_S, Z_{S'} \mid \Theta_{sep}\right) \tilde{p}\left(Z_F, Z_{F'} \mid \Theta_{sep}\right) p\left(\Theta_F \mid \Theta_{sep}, Z_F, Z_{F'}\right) \quad (17)$$

As shown in (17), the synchronized system is equivalent to the full joint of a single system containing *all* measurements ($p\left(\Theta \mid Z\right)$), and thus the optimal MAP estimate is recovered. Since these marginal factors must be calculated by each branch during variable elimination as part of the normal inference process, the marginals can be cached by each branch with no additional computational requirement. Thus, synchronization is a constant time process, $O(1)$.

5.4 Advancing the Separator

After synchronization, the concurrent architecture has recovered the MAP state estimate given all of the incorporated measurements. However, as new measurements are added to the filter branch, the size of the filter branch increases. Over

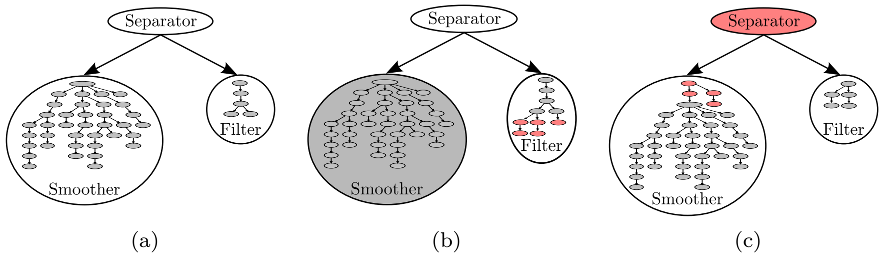


Figure 9: The evolution of the CFS Bayes tree over time. (a) The CFS begins in a consistent, synchronized state. (b) The smoother starts an update and thus is unavailable for a time. New states are accumulated in the filter branch (shown in red). (c) Once the smoother update completes, the separator is advanced, returning the filter to the desired size and transitioning new states to the smoother (shown in red). At this point, the separator is constructed from a different set of states than in (a) or (b).

time, the computation required to update the filter will exceed the real-time threshold for a given platform. To remedy this situation, a new separator is constructed that leaves the desired number of states in the filter branch, using the same process described in Section 5.1. This effectively advances the separator forward in time, as depicted in Figure 9. The factors and variables that move out of the filter branch are immediately sent to the smoother for further optimization (line 27 in Algorithm 1). This operation is equivalent to selecting a different elimination order in the filter, and consequently does not affect the state estimated by the system. Further, since the estimate does not change as a consequence of this reordering, the full filter does not need to be recomputed. Partial elimination results from the unchanged portion of the filter branch can be reused, as was recognized in incremental smoothing and mapping (iSAM2) Kaess et al. (2011, 2012a). Consequently, the computational complexity of moving the separator depends only on the number of states to be transitioned to the smoother branch.

5.5 Time Delays

As mentioned, the trade-off for utilizing the CFS factorization is a time delay between a loop closure observation and effect of the loop closure on the filter solution. One source of time delay is simply the computation time required to update the smoother with the loop closure constraint. There can be additional delay induced by the availability of the smoother at the time the constraint is first identified. In the worst case, a loop closure constraint is identified just after a smoother update has been initiated, at time t. Before this constraint may be added to the smoother, the smoother must first complete its current update. Assuming each smoother update requires Δ_s seconds to complete, the new constraint will not be inserted into the smoother until $t+\Delta_s$. The smoother must then perform a full second update before the smoother has incorporated the new constraint into the solution and synchronized with the filter. Thus, the filter will not be updated with the effects of the loop closure constraint until time $t + 2\Delta_s$.

A second, less obvious time delay is induced from the CFS factorization itself. As discussed in Section 5.1, the CFS factorization assumes no edge directly connects a filter state to a smoother state, thus allowing the filter and smoother to be conditionally independent. However, loop closure constraints are supposed to do just that: relate the current state to states in the distant past. In order to maintain the CFS factorization, these loop closures must be delayed until the involved state has transitioned from the filter into the smoother or separator. The size of this time delay will be dependent on the design of the filter. For example, if the states comprising the last Δ_f seconds are maintained in the filter (i.e. an augmented state filter or fixed-lag smoother), then the CFS architecture cannot incorporate the loop closure constraint into the smoother for at least Δ_f seconds. Thus, delays as long as $2\Delta_s + \Delta_f$ may be experienced between the time a loop closure is identified and the time the filtering solution is affected.

To demonstrate this effect, a simulated system consisting of three relative-pose measurement sources is considered. The filter is implemented as a fixed-lag smoother with a smoother lag $\Delta_f = 5.0\,s$, and the filter is updated at a rate of 10Hz. The smoother is operated with an artificial smoother delay of $\Delta_s = 10.0\,s$. Synchronization is scheduled to occur after every smoother update. A loop closure constraint between the most recent state and a past state within the smoother is identified at $t = 15.1\,s$, midway between the smoother updates at $t = 10.0\,s$ and $t = 20.0\,s$. Since this constraint would connect between a filter state and smoother state, the constraint cannot be added until the involved state transitions to the smoother. In this example, the state would be available to be moved to the smoother at $t = 20.1\,s$, $5.0\,s$ after the state first appeared in the filter. However, this transition will not actually occur until the next filter-smoother synchronization, scheduled at $t = 30.0\,s$. At $t = 30.0\,s$, the loop closure constraint will be added to the smoother, starting a new Δ_s smoother update cycle. At $t = 40.0\,s$, the smoother will complete this cycle and synchronize with the filter, updating the filter solution with the effects of the loop closure constraint. This means the constraint identified at $t = 15.1\,s$ will not affect the navigation solution until $t = 40.0\,s$, a delay of $24.9\,s$ or approximately $2\Delta_s + \Delta_f$. Note that this is the worst-case delay. Figure 10 compares the resulting trajectory from the CFS system to that of a full batch optimization where the loop closure is applied immediately. The timestamps of the different events are marked on the trajectory. Notice that after the CFS has fully integrated the loop closure, it again tracks the full batch solution.

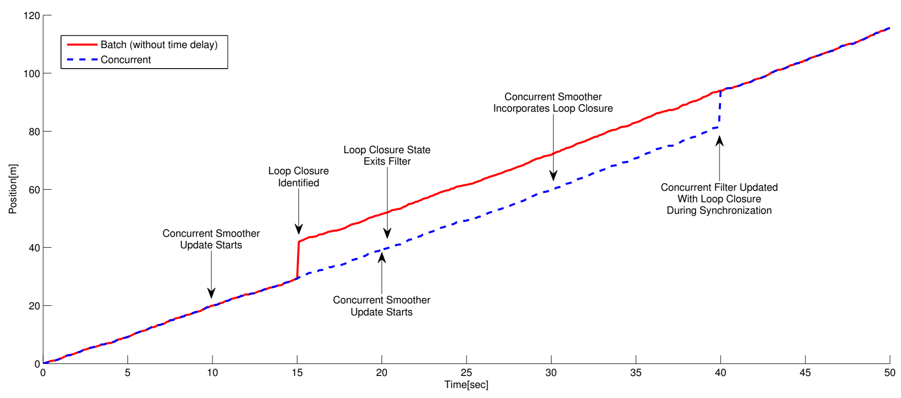


Figure 10: An example system demonstrating the time delays inherent in the CFS system. A loop closure constraint is identified at 15.1 s while the concurrent smoother is updating. Due to the concurrent filter lag, the loop closure state does not exit the filter until 20.1 s, after a new smoother update has been initiated at 20.0 s. Thus the loop closure does not get incorporated into the concurrent smoother until the third update at 30.0 s, and does not affect the filter until the concurrent smoother update has completed at 40.0 s.

6 Constant-Time Operation

Thus far, the basic concept of the CFS system has been discussed: a factorization that produces two conditionally independent branches, a method for performing concurrent updates of those branches, and a simple method for synchronizing the branches after a series of concurrent updates. However, the computational complexity of the filter update (Section 5.2) and of the separator advance (Section 5.4) both have an undesirable linear-time component. In this section, we discuss additional techniques that guarantee constant-time filter operation and synchronization.

6.1 Constant-time Filter Update

The factorization of the full joint density used by the concurrent filtering and smoothing architecture permits asynchronous updates of the smoother and the filter, but it does not guarantee constant-time updates for either. Specifically, the filter will accumulate additional states during times when the smoother is busy, causing the filter computation time to increase. For example, if new states are added at a rate of 10Hz and the smoother requires 1.5s to complete an update, then the size of the filter will increase by 15 states before the next synchronization is performed. After the synchronization occurs, the size of the filter will be reset to its nominal size. This cycle of accumulating states in the filter and later transitioning them to the smoother is illustrated in Figure 9.

This linear increase in the filter size between synchronizations is undesirable as it translates into increased computational requirements of the high-rate filter. This is complicated by the fact that the required update time of the smoother is unpredictable, depending on such things as the quality of initialization and the presence or absence of loop closures.

To combat this issue, a method of "pre-advancing" the separator can be implemented to maintain a constant-size filter. With this scheme, the filter constantly changes its local definition of the separator states to maintain a constant-size filter. States that should be part of the smoother are transitioned to the smoother side of the Bayes tree *within* the filter. At the same time, a marginal on the new separator from the smoother is computed using the smoother marginal on the old separator (provided during the last synchronization) and the information contained within the transitioned nodes.

At each timestep, the filter determines the minimal set of states that are to remain inside the filter. For an augmented state filter or fixed-lag smoother, this is simply the set of states newer than the defined filter lag. For vision-enabled systems, this might be the set of states associated with active feature tracks. We define this new set of filter states as $\Theta_{F'}$. Using the method defined in Section 5.1, the corresponding separator, $\Theta_{sep'}$, is identified. Any existing filter states that are not included in the new filter states or new separator states will eventually be transitioned to the smoother. Denote the set to be transitioned as $\Theta_{S'} = \{\Theta_{sep}, \Theta_F\} \setminus \{\Theta_{sep'}, \Theta_{F'}\}$, the transition variables. Also define the set of measurements that involve *only* transition states or new separator states as $Z_{S'}$. Since these measurements connect only new smoother states and new separator states, this is the set of measurements that should be sent to the smoother for further optimization. Using the old definition of the filter and separator states, the filter joint was factored as:

$$p_{filter}\left(\Theta_{sep}, \Theta_F\right) = \tilde{p}\left(Z_S \mid \Theta_{sep}\right) \tilde{p}\left(Z_F \mid \Theta_{sep}\right) p\left(\Theta_F \mid \Theta_{sep}, Z_F\right) \tag{18}$$

where Θ_F denotes all of the filter variables, and Z_F denotes all of the filter measurements. Using the new definition of the filter and separator states, the filter joint can be further factored as:

$$\begin{aligned} & p_{filter}\left(\Theta_{sep}, \Theta_F\right) = \\ & \qquad \tilde{p}\left(Z_S \mid \Theta_{sep}\right) \tilde{p}\left(Z_{S'} \mid \Theta_{S'}, \Theta_{sep'}\right) \tilde{p}\left(Z_{F'} \mid \Theta_{sep'}\right) p\left(\Theta_{F'} \mid \Theta_{sep'}, Z_{F'}\right) \end{aligned} \tag{19}$$

where the combination of the transition states $\Theta_{S'}$, the new separator states $\Theta_{sep'}$, and the new filter states $\Theta_{F'}$ span the set of original filter and separator states such that $\Theta_{sep} \cup \Theta_F = \Theta_{S'} \cup \Theta_{sep'} \cup \Theta_{F'}$. The final two terms in this factorization, $\tilde{p}\left(Z_{F'} \mid \Theta_{sep'}\right) p\left(\Theta_{F'} \mid \Theta_{sep'}, Z_{F'}\right)$, are equivalent to the final terms in the original factorization, $\tilde{p}\left(Z_F \mid \Theta_{sep}\right) p\left(\Theta_F \mid \Theta_{sep}, Z_F\right)$, except using the new definition of the separator states, filter states, and filter measurements. The estimate for the smoother marginal on the new separator, $\tilde{p}\left(Z_S, Z_{S'} \mid \Theta_{sep'}\right)$, can be obtained by marginalizing out the unneeded variables from the first two

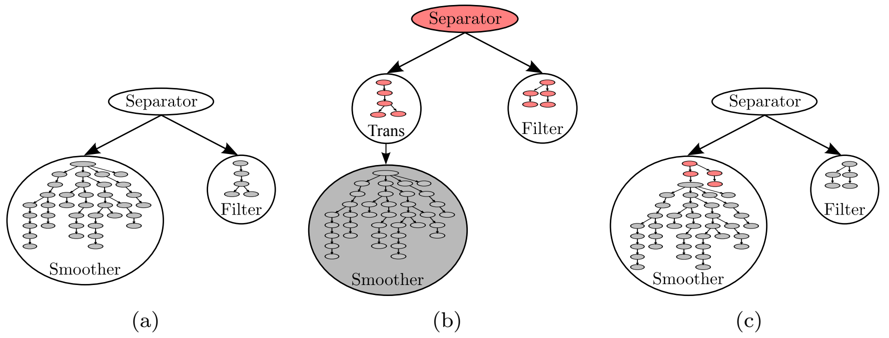


Figure 11: The evolution of the CFS Bayes tree using the separator "pre-advance" strategy. The CFS begins in a consistent, synchronized state. (b) The smoother starts an update and thus is unavailable for a time. As new states are introduced into the filter, the "pre-advance" strategy creates a new separator that maintains a constant-sized filter. The old states are transitioned to the smoother side of the Bayes tree, even though they are still contained within the filter states. (c) Once the smoother update completes, the transition states are moved into the smoother.

terms of this factorization:

$$\tilde{p}\left(Z_S, Z_{S'} \mid \Theta_{sep'}\right) = \int_{\neg\Theta_{sep'}} \tilde{p}\left(Z_S \mid \Theta_{sep}\right) \tilde{p}\left(Z_{S'} \mid \Theta_{S'}, \Theta_{sep'}\right) \tag{20}$$

Assuming the marginal on the old separator, $\tilde{p}\left(Z_S \mid \Theta_{sep}\right)$, provided to the filter from the smoother during the last synchronization, remains unchanged during sequential filter updates, then this operation of marginalizing out the states not contained within the new separator happens automatically during elimination. The method of "pre-advancing" may be interpreted as enforcing a specific elimination order within the filter branch, generating a new separator for the combined system. By caching the updated marginal on the new separator produced during elimination, no additional computation is incurred by the filter to maintain a constant state size. Also, since the variables being transitioned to the smoother are simply marginalized from the filter, then the behavior of the filter between synchronizations is identical to a fixed-lag or augmented state filter. Figure 11 illustrates the "pre-advance" cycle of accumulating transition states while maintaining a constant-size filter. Note that the same states are transitioned to the smoother at the same time regardless of whether "pre-advance" is used; only the definition of the filter states changes.

6.2 Constant-time Synchronization

As described in the previous section, a scheme of "pre-advancing" the separator has been implemented that allows the filter update to be constant-time. During each filter update, an elimination order is selected such that the older states are moved to the transition set, reducing the number of states included in the filter. This allows the filter to remain a constant size and hence allow constant-time updates.

However, this increases the computational complexity of the synchronization step as there are now transition states between the smoother-maintained separator and the filter-maintained separator. This situation is depicted in Figure 11, where the states of interest are labeled as transition nodes. When using the separator "pre-advance," none of the transition states are involved in the filter update. However, during synchronization, the updated smoother information must be propagated to the filter. All of the transition states, $\Theta_{S'}$, are involved in this calculation. The separator "pre-advance" effectively moves a linear-time operation out of the filter update and into the synchronization.

To remove this linear time delay from the synchronization process, a constant-time synchronization update can be implemented by maintaining a "shortcut" conditional between the smoother-maintained separator and the current filter-maintained separator. Under the separator "pre-advance", the full joint density is effectively factorized as:

$$\begin{aligned} p\left(\Theta\right) = p\left(\Theta_S \mid \Theta_{sep}, Z_S\right) \tilde{p}\left(Z_S \mid \Theta_{sep}\right) \cdot \\ \tilde{p}\left(Z_{S'} \mid \Theta_{S'}, \Theta_{sep'}\right) \tilde{p}\left(Z_{F'} \mid \Theta_{sep'}\right) p\left(\Theta_{F'} \mid \Theta_{sep'}, Z_{F'}\right) \end{aligned} \quad (21)$$

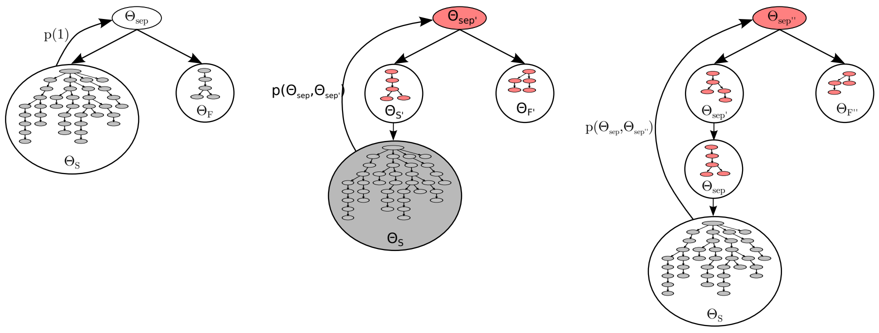
When the smoother completes an update, it will have recomputed its contribution to the full factored joint density, the conditional $p\left(\Theta_S \mid \Theta_{sep}, Z_S\right)$ and the marginal $\tilde{p}\left(Z_S \mid \Theta_{sep}\right)$. However, unlike the synchronization step described in Section 5.3, this marginal cannot be applied directly to the filter-maintained separator variables, $\Theta_{S'}$. Instead, a new marginal must be computed as:

$$\tilde{p}\left(Z_S, Z_{S'} \mid \Theta_{sep'}\right) = \int_{\neg\Theta_{sep'}} \tilde{p}\left(Z_S \mid \Theta_{sep}\right) \tilde{p}\left(Z_{S'} \mid \Theta_{S'}, \Theta_{sep'}\right) \quad (22)$$

Clearly, the computational complexity of this operation depends on the number of transition states that must be marginalized out to form the desired marginal density. Alternatively, this marginalization step can be reformulated as a recursive operation shown in (23). We use the notation $Z_{S'}$ and $Z_{S''}$ to represent the measurements to be transitioned at consecutive filter updates, and $\Theta_{sep'}$ and $\Theta_{sep''}$ to represent the separator states during those time periods.

$$\begin{aligned} \tilde{p}\left(Z_S, Z_{S'}, Z_{S''} \mid \Theta_{sep''}\right) &= \int_{\neg\Theta_{sep''}} \tilde{p}\left(Z_S \mid \Theta_{sep}\right) \tilde{p}\left(Z_{S'}, Z_{S''} \mid \Theta_{sep}, \Theta_{sep''}\right) \\ \tilde{p}\left(Z_{S'}, Z_{S''} \mid \Theta_{sep}, \Theta_{sep''}\right) &= \int_{\Theta_{sep'}} \tilde{p}\left(Z_{S'} \mid \Theta_{sep}, \Theta_{sep'}\right) \tilde{p}\left(Z_{S''} \mid \Theta_{sep'}, \Theta_{sep''}\right) \end{aligned} \quad (23)$$

We call the joint marginal density $\tilde{p}\left(Z_{S'}, Z_{S''} \mid \Theta_{sep}, \Theta_{sep''}\right)$ the "shortcut" marginal, as it allows a new version of the smoother marginal, $\tilde{p}\left(Z_S \mid \Theta_{sep}\right)$, to be converted into a marginal on the new separator, $\tilde{p}\left(Z_S, Z_{S'}, Z_{S''} \mid \Theta_{sep''}\right)$, *directly,*



(a) (b) (c)

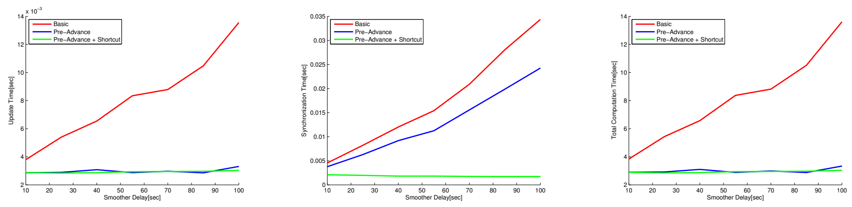
Figure 12: The evolution of the CFS Bayes tree using the separator "pre-advance" and the "shortcut" marginal. In addition to accumulating transition states over time, a shortcut is maintained that allows updated smoother information to be propagated directly to the root clique of the filter, *without* involving the transition states.

without including and then eliminating all of the accumulated transition states. Under the assumption that the number of variables in the separator remains approximately constant over time (i.e. the size of $\Theta_{sep''}$ is about the same as Θ_{sep}), then the calculation of the new separator marginal will also be constant-time. It should be noted that while the creation of the shortcut conditional allows for constant-time synchronization, the recursive calculation of the shortcut is less efficient than the direct computation described in (22).

Figure 12 shows the time evolution of the CFS Bayes tree with separator pre-advance and the shortcut marginal. As in Section 6.1, the filter accumulates transition states over time while the smoother is busy. Additionally, the shortcut marginal is updated recursively every timestep (Figure 12b). When the smoother completes its update, the shortcut marginal can be used to propagate the updated smoother information directly to the root clique in the filter (Figure 12c).

6.3 Timing Performance

When utilizing the proposed separator "pre-advance" and the synchronization "shortcut" conditional, different trade-offs are made between the computation time of a specific algorithm component and the total required computation. Figure 13 shows example computation times of the different components for the basic CFS system, CFS with separator "pre-advance", and CFS with separator "pre-advance" and the synchronization "shortcut". This example uses simulated data from three relative-pose measurement sources. Filter updates occur at 10Hz while the period between synchronization is artificially adjusted from 10.0 s to 100.0 s. As seen, the basic CFS system has an approximately linear increase in



(a) Filter Update Time (b) Synchronization Time (c) Total Computation Time

Figure 13: Update and synchronization performance with respect to the delay between concurrent smoother synchronizations using the basic CFS, CFS with separator pre-advance, and CFS with separator pre-advance and synchronization shortcut.

both the filter update time and synchronization time. Between synchronizations the basic system accumulates additional states within the filter, resulting in the observed linear trend. During synchronization, the basic system must also advance the separator to recreated the desired filter size. The number of states involved in this operation also increases with the time between synchronizations. By incorporating the separator pre-advance, the number of states within the filter is held constant, resulting is constant-time filter updates. However, as described in Section 6.2, any updated information from the smoother must be propagated through the transition variables first. Since the number of transition variables is proportional to the time between synchronizations, a linear trend in the synchronization time is evident. Finally, when using both the separator pre-advance and the synchronization shortcut, both the filter update and the synchronization time are approximately constant-time.

7 Results

In this section we investigate the estimation accuracy and the computational performance of the concurrent filtering and smoothing approach using both a synthetic dataset and real-world data. The proposed method is compared to full batch optimization and a fixed-lag smoother. All methods were implemented using the open-source factor graph library, GTSAM[1], using a Gauss-Newton iterative nonlinear solver. All tests were executed on a single core of an Intel i7-2600 processor with a 3.40GHz clock rate and 16GB of RAM memory.

7.1 Simulation

The CFS architecture was examined in a 100-trial Monte-Carlo study using an aerial vehicle simulation. A ground truth trajectory was created, simulating a flight of an aerial vehicle at a 20 m/s velocity and a constant height of 200 meter

[1]https://collab.cc.gatech.edu/borg/.

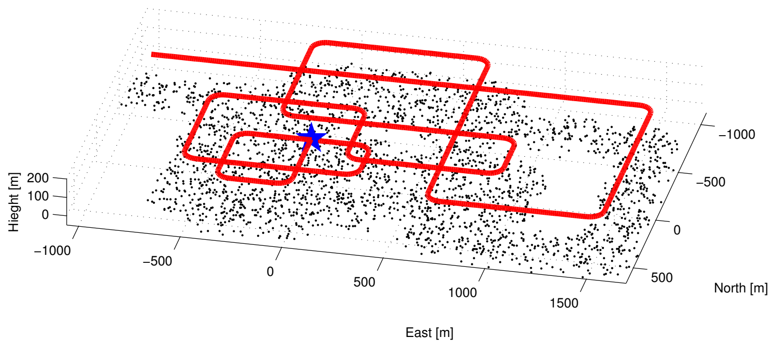


Figure 14: A visualization of the simulated ground truth trajectory of an aerial vehicle. Ground-based landmarks are observed from a downward-facing stereo camera. Star marking indicates beginning of trajectory.

above mean ground level. The trajectory consists of several segments of straight and level flight and maneuvers, as shown in Figure 14.

Based on the ground truth trajectory, ideal IMU measurements were generated at 100 Hz, while taking into account Earth's rotation and changes in the gravity vector. For each of the 100 Monte-Carlo runs, these measurements were corrupted with a constant bias and zero-mean Gaussian noise in each axis. Bias terms were drawn from a zero-mean Gaussian distribution with a standard deviation of $\sigma = 10$ mg for the accelerometers and $\sigma = 10$ deg/hr for the gyroscopes. The noise terms were drawn from a zero-mean Gaussian distribution with $\sigma = 100\,\mu g/\sqrt{Hz}$ and $\sigma = 0.001\,deg/\sqrt{hr}$ for the accelerometers and gyroscopes. Initial navigation errors were drawn from zero-mean Gaussian distributions with $\sigma = (10, 10, 15)$ meters for position (expressed in a north-east-down system), $\sigma = (0.5, 0.5, 0.5)$ m/s for velocity and $\sigma = (1.0, 1.0, 1.0)$ degrees for orientation. Instead of incorporating IMU measurements into the filter at the IMU rate of $100\,Hz$, sequential measurement are accumulated between the measurement times of the other aiding sensors, and added to the graph as a single factor Lupton and Sukkarieh (2012); Indelman et al. (2012, 2013b,a). This prevents the accumulation of excessive states within the filter or smoother, without a loss in estimation accuracy. Methods for generating a state estimate at IMU rate are available, if desired.

In addition to IMU, the aerial robot was assumed to be equipped with a stereo camera operating at 0.5Hz. Ideal visual observations were calculated by projecting landmarks, scattered on the ground with ± 50 meters elevation, onto the cameras. Zero-mean Gaussian noise, with $\sigma = 0.5$ pixels, was added to all visual measurements. Landmarks were observed on average by 5 views, with the

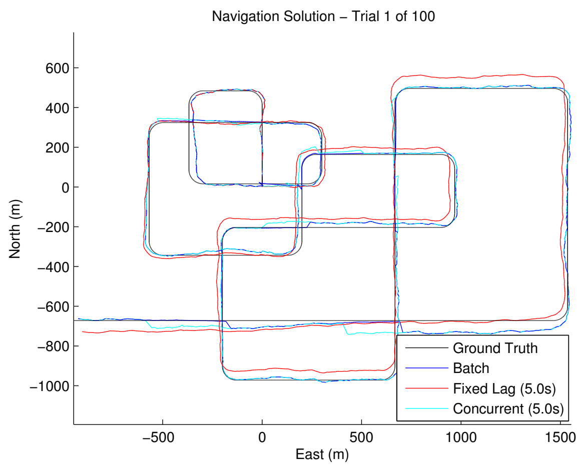


Figure 15: Trajectory generated by the CFS system, compared to ground truth, batch and fixed lag smoother trajectories from a typical trial in the Monte Carlo study.

shortest and longest landmark-track being 2 and 12, respectively. Additionally, loop closure measurements (i.e. landmark re-observations) were produced when possible. Within the 725 s simulated trajectory, 11 areas exist where the trajectory crosses itself, allowing the stereo camera to re-observe past landmarks.

For comparison, all measurements were processed using a full batch optimizer, a short-term fixed-lag smoother, and the proposed concurrent filtering and smoothing system. The full batch optimizer receives all of the measurements (IMU, visual odometry, and loop closure constraints) whenever they are measured, producing an updated output for each measurement time. For the purposes of this comparison, the calculation time of the batch optimizer is ignored, allowing the batch system to produce estimates as if it could run in real-time. The fixed-lag smoother performs nonlinear optimization over the states within a 5.0 s smoothing lag. After a state leaves the smoothing lag, it is marginalized from the system. Consequently, this fixed-lag smoother is incapable of processing the loop closure constraints and only the IMU and visual odometry measurements are provided. Similarly, the concurrent filter is designed as a fixed-lag smoother with a 5.0 s smoothing lag. IMU and visual odometry measurements are sent to the filter, while the loop closure constraints are provided directly to the smoother. However, as discussed in Section 5.5, there is a delay between the time the loop closure is identified and the time the concurrent filter incorporates the constraint into the solution. Both the concurrent filter and concurrent smoother use Gauss-Newton nonlinear optimization internally. As an example, Figure 15 shows the resulting trajectories from a typical trial in the Monte Carlo study.

Figures 16-18 show a comparison of the root mean square errors (RMSE)

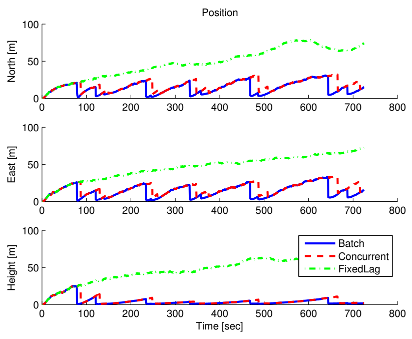


Figure 16: Position estimation errors produced by a fixed-lag smoother, the concurrent filter and smoother, and a full batch optimization.

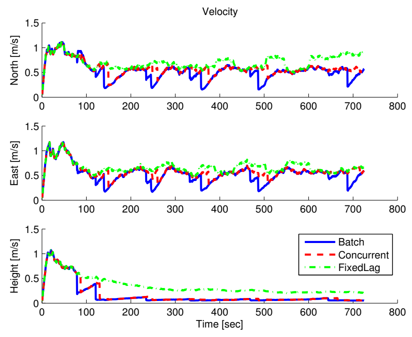


Figure 17: Velocity estimation errors produced by a fixed-lag smoother, the concurrent filter and smoother, and a full batch optimization.

produced by the full batch optimization, the fixed-lag smoother, and the concurrent filtering and smoothing system over the 100 trials. As shown, the performance of batch optimization, fixed-lag smoother, and concurrent system are all similar until the first loop closure is applied to the batch optimization at 78.6 s. The addition of this loop closure significantly reduces the estimations errors. Approximately 8.0 s later (5.0 s for the loop closure state to transition from the filter to the smoother plus 3.0 s of additional delay due to smoother availability), the concurrent errors are reduced to the batch optimization levels. The errors of the fixed-lag smoother, unable to benefit from the loop closure constraints, continue to increase over the entire trajectory. To further demonstrate that the CFS system recovers the batch solution, Figures 19-21 show the error of CFS and fixed-lag smoother relative to the batch solution. By plotting the differences with respect to the batch solution, it is clear that the CFS system recovers the batch solution, except for the delays around loop closure events.

Additionally, a metric known as the Kullback–Leibler divergence (KLD) Kullback and Leibler (1951) has been computed, comparing the CFS and the

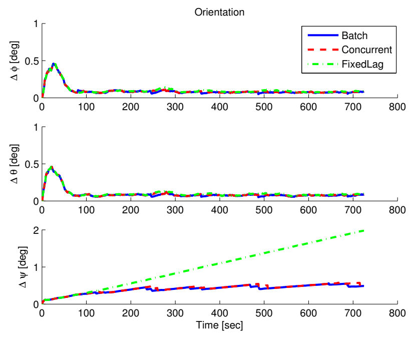


Figure 18: Orientation estimation errors produced by a fixed-lag smoother, the concurrent filter and smoother, and a full batch optimization.

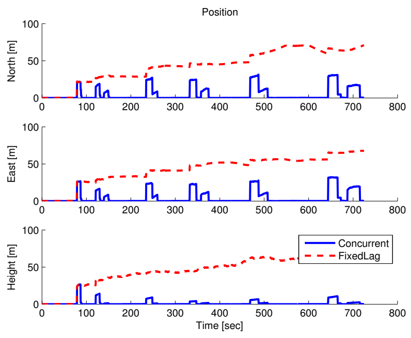


Figure 19: Position differences relative to the full batch solution for a fixed-lag smoother and the concurrent filter and smoother.

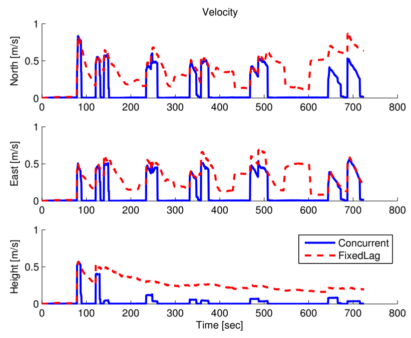


Figure 20: Velocity differences relative to the full batch solution for a fixed-lag smoother and the concurrent filter and smoother.

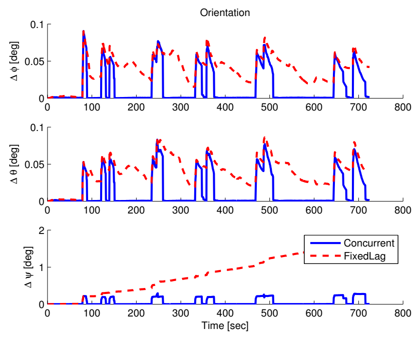


Figure 21: Orientation differences relative to the full batch solution for a fixed-lag smoother and the concurrent filter and smoother.

fixed-lag smoother results with that of the batch optimization. The KLD between two k-dimensional normal distributions $\mathcal{N}_1(\mu_1, \Sigma_1)$ and $\mathcal{N}_2(\mu_2, \Sigma_2)$ is defined as:

$$D_{\mathrm{KL}}(\mathcal{N}_1 \| \mathcal{N}_2) = \frac{1}{2}\left(\left[\mathrm{tr}\left(\Sigma_2^{-1}\Sigma_1\right) - k\right] + \left[(\mu_2 - \mu_1)^{\top} \Sigma_2^{-1}(\mu_2 - \mu_1)\right] - \ln\left(\frac{\det \Sigma_1}{\det \Sigma_2}\right)\right) \tag{24}$$

The KLD measures of the difference between two distributions up to the third moment. Thus, the errors in the covariance as well as the mean are included. Figure 22 shows the KLD of the 9-dimensional navigation state (6-dimensional pose and 3-dimensional velocity) produced by the CFS and fixed-lag covariance estimates compared with the batch covariances, averaged over the 100 Monte Carlo trials. Again, the CFS system is able to recover the full batch covariance between loop closure events while the fixed-lag smoother, without the aid of loop closure constraints, continues to diverge.

7.2 Karlsruhe Dataset

To test the proposed method on real-world data, we make use of the KITTI Vision Benchmark Suite (Geiger et al., 2012). These datasets were captured from the autonomous vehicle platform “Annieway” during traverses around the city of Karlsruhe, Germany. This platform consists of a car chassis outfitted with a stereo camera and a differential GPS/INS system. The differential GPS/INS data provides highly accurate ground truth position and orientation data. Additionally, raw IMU measurements are provided at 100 Hz, but are accumulated into factors only at camera rate, as was done in the simulated trials. Raw camera images from the stereo rig are available at 10Hz. A typical visual odometry pipeline was used in which image features are extracted and matched between sequential camera pairs using image descriptors. In this example, SIFT Lowe (1999) features and descriptors are used. As in the simulated aerial environment, the visual odometry measurements are in the form of frame-to-frame relative

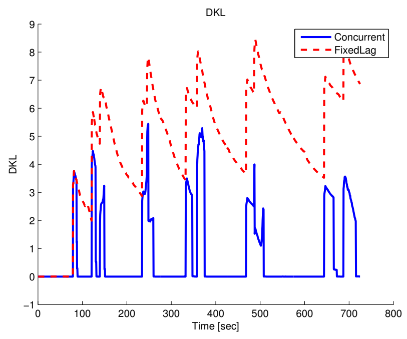


Figure 22: The average Kullback–Leibler divergence (KLD) of the CFS and fixed-lag smoother results compared with the full batch solution. The KLD measures the differences of two distributions up to the third moment, so errors in the covariance as well as the mean are captured.

pose transformation estimates. Figure 23 shows several typical camera images with the tracked features indicated in red. Loop closures were extracted using standard clustering techniques on the extracted feature descriptors. Figure 24 shows the ground truth trajectory (blue) overlaid on satellite images from Google Earth®. The identified loop closures are displayed in green.

As in the simulated scenario, the measurements are processed by a full batch optimizer, a short-term fixed-lag smoother, and the proposed concurrent filtering and smoothing system. The full batch optimizer receives all of the measurements (IMU, visual odometry, and loop closure constraints) whenever they are measured, producing an updated output for each measurement time. The fixed-lag smoother is only provided with the IMU and visual odometry measurements. Since camera measurements are produced at a much higher frequency in the ground data, the fixed-lag smoother is configured with a $2.0\,s$ lag, which includes measurements from the last 20 camera frames. Similarly, the concurrent filter is designed as a fixed-lag smoother with a $2.0\,s$ smoothing lag; identified loop closure constraints are sent directly to the concurrent smoother. The computed trajectories of all methods are shown in Figure 25.

The proposed CFS system closely tracks the full batch optimization results over the entire trajectory. As seen in Figure 25, the full batch trajectory and the CFS trajectory are nearly indistinguishable, except near loop closures. The fixed-lag smoothers, on the other hand, eventually drifts away from the optimal solution as it cannot take advantage of the loop closure constraints. This is shown in more detail in Figures 26a-26c which plot the position, velocity and orientation errors relative to the batch solution. As can be seen from these figures, the CFS system produces results very close to the batch estimate, with the most dramatic differences in the position and yaw estimates.

The effect of the time delays, described in Section 5.5, are also evident in the CFS trajectory. Figure 25b shows an expanded view of the first loop

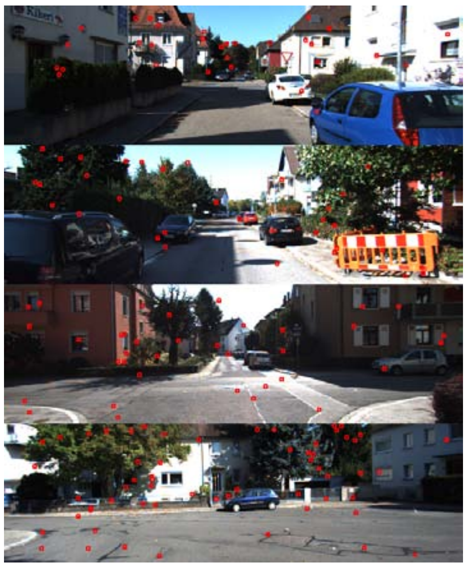


Figure 23: Typical camera images from the test KITTI dataset. Features tracked by the visual odometry system are indicated in red.

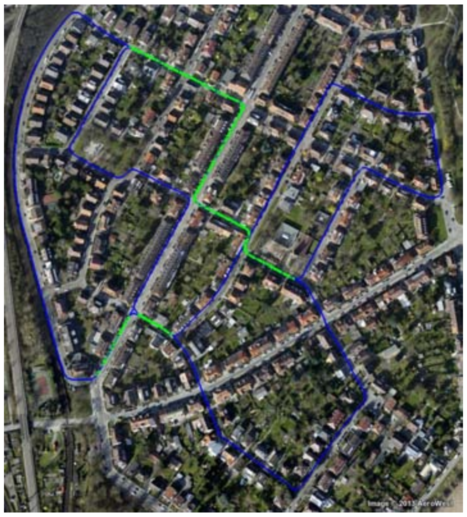


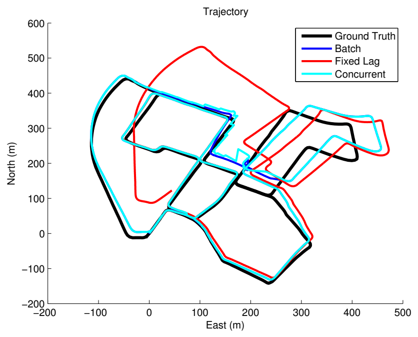
Figure 24: Ground truth trajectory (blue) with areas of identified feature correspondences (loop closures) identified (green).

closure event within this Karlsruhe scenario. The vehicle starts in the north-west corner, driving south. Once the vehicle rounds the corner, the loop closure system correctly identifies that the vehicle is retracing a previously driven path and adds loop closure constraints. The batch solution incorporates these loop closures immediately, correcting much of the drift accumulated during the first $161.8\,s$. However, the CFS system must first wait for the states involved in the loop closure to transition to the smoother (approximately $2.0\,s$ in this case) and wait for the smoother to complete the current update and process the new loop closure (a delay of up to $1.2\,s$ each as shown in Figure 28). The CFS system ultimately updates the navigation solution with the loop closure information at $165.5\,s$, an actual delay of $3.7\,s$.

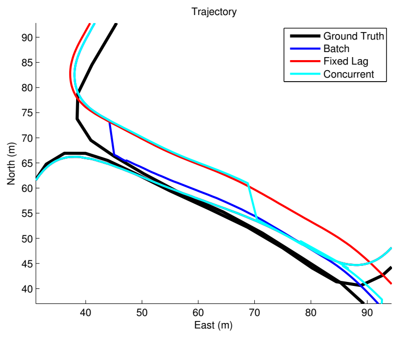
Also, each CFS filter-smoother synchronization event may be viewed as a single iteration of a nonlinear optimizer operating on the entire joint pdf. While a single iteration is often sufficient to achieve a near-optimal trajectory, corrections of large drifts may need multiple synchronization updates to fully converge. This is the situation depicted in Figure 25c. A large loop closure occurs between the vehicle at time $352.6\,s$ and part of the first loop at time $43.0\,s$. This loop closure corrects for drift accumulated over most of the trajectory. The full batch optimizer requires 5 iterations and over $11\,s$ to fully incorporate the first of the loop closures starting at $352.6\,s$. In contrast, the CFS system continues to produce navigation solutions at a rate of $10\,Hz$ while still converging to the batch solution over time. However, the CFS requires more iterations before it fully converges in this particular instance. Position updates due to these multiple iterations can be clearly seen in Figure 25c.

In case a smooth trajectory is required, e.g. for control algorithms, one alternative is to generate such a trajectory using a fixed-lag estimator. Although this solution will drift over time, it should be adequate for controlling the vehicle.

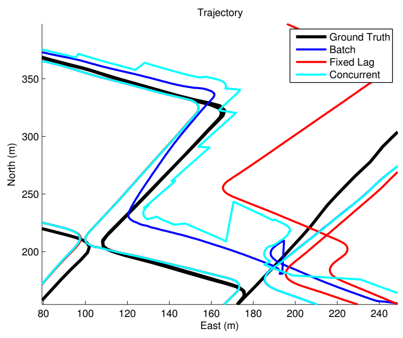
We also characterize the computation time required for each aspect of the CFS system. First, Figure 27 shows the calculation time required to perform each update for the full batch system, the fixed-lag smoother, and the CFS. Since the CFS is a parallel system, this time includes only the update time for the filter, and does not include the smoother update time (which happens in a separate thread) or the synchronization time. This time reflects the latency between the time a measurement arrives and the time each inference architecture has an updated navigation solution. Full batch optimization is not intended to be real-time; it exhibits a general linear time increase with large upward deviations caused by performing multiple optimization iterations per timestep. The fixed-lag smoother and the concurrent filter should both provide constant time updates. Further, since the fixed-lag smoother and concurrent filter are both configured for the same smoother lags, their update times should be similar. However, in addition to performing the same required operations as the fixed-lag smoother, the concurrent filter also caches factors for the smoother and calculates the shortcut marginal (see Section 6.2) at each iteration. As shown in the zoomed in view, this additional overhead is negligible and both the concurrent filter and fixed-lag smoother require approximately $6\,ms$ to perform each update. The timing variation shown for the filters is largely due to



(a) Trajectory

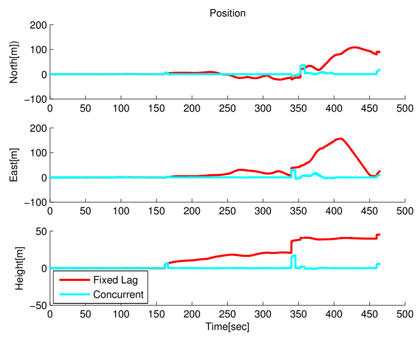


(b)

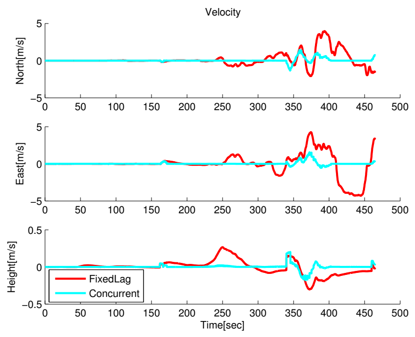


(c) Zoom 2

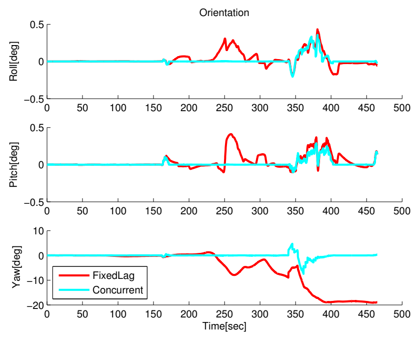
Figure 25: (a) The computed trajectory using full batch optimization, a fixed-lag smoother, and the CFS system. The CFS trajectory remains close to the batch solution, while the fixed-lag smoother trajectory diverges due to lack of loop closure constraints. (b) An expanded view of the first loop closure incorporated by the batch and CFS optimization, demonstrating the time delay of the CFS, described in Section 5.5. (c) An expanded view of a long-term loop closure correcting a large accumulated drift. While the batch optimization is able to perform multiple optimization iterations immediately, the CFS system converges to the batch solution over time as more synchronization events take place.



(a)

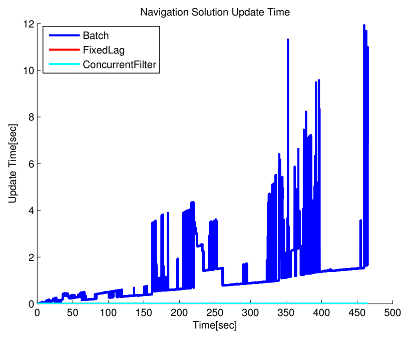


(b)

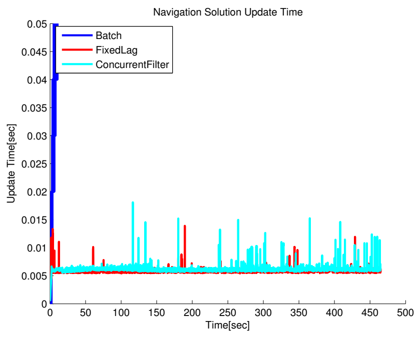


(c)

Figure 26: Position, velocity, and orientation errors relative to the batch solution. The CFS system recovers the batch solution, except near loop closures as a result of the time delays inherent in the real-time CFS. In particular, the large loop closure depicted in 25c occurs at $t = 352.6\,s$, corresponding to a period where the CFS solution is noticeably different than the batch. By $t = 400\,s$, the CFS has fully incorporated all of the loop closures in this extended period and again tracks the batch solution.



(a)



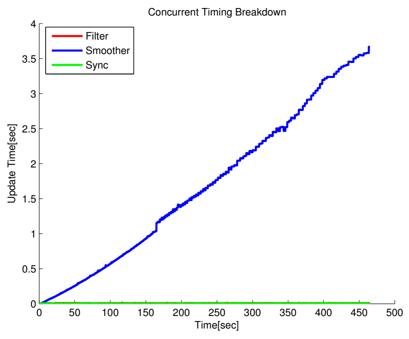
(b)

Figure 27: (a) The calculation time for each navigation update of the full batch optimization, fixed-lag smoother, and the concurrent filter. Loop closure constraints are evident in the smoother timing as large spikes in individual updates (e.g. at 165 s and 340 s) caused by the optimizer needing many iterations to converge. Compared to the full batch optimization update times, both the fixed-lag smoother and the CFS are small. (b) An expanded view of the the fixed-lag smoother and CFS update times, demonstrating constant-time operation for both systems. Despite additional overhead in the CFS, such as maintaining the "shortcut" marginal, their actual performance is comparable.

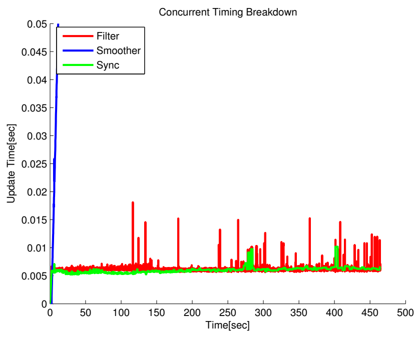
imprecisions in the CPU timer used in profiling. Figure 28 shows the timing break-down of the individual components of the concurrent architecture: filter update, smoother update, and synchronization. Again, the smoother update is not intended to be real-time, and is run in a background thread where it does not affect the filter operation. The concurrent smoother has been configured to only perform a single optimization iteration, allowing the CFS to synchronize more frequently. Thus, the large upward timing deviations observed in the full batch smoother of Figure 27 are not present. The filter update times are exactly those presented in Figure 27, and are provided for reference and scale of the synchronization times. Using "shortcut" marginal calculation described in Section 6.2, the synchronization time can be performed in approximately constant time. This is supported by the zoomed in view of Figure 28 where each synchronization requires approximately 6 ms, similar to the filter update time.

8 Conclusions

This paper presented a general parallel inference architecture that combines the fast updates typical of navigation filters with the optimality of full smoothing approaches. This is accomplished by factoring the full joint probability density used by batch smoothing approaches in a particular way, resulting in two conditionally independent subgraphs. Methods for updating each subgraph indepen-



(a)



(b)

Figure 28: (a) The calculation time of the individual component of the CFS system: smoother update, filter update, and synchronization. (b) An expanded view of just the filter and synchronization times, showing both low-latency and constant-time performance of both components.

dently and in parallel were presented, as well as a process for synchronizing the two subgraphs after concurrent updates have occurred. These processes are general, and are not tied to specific sensors, representations, or inference methods. It is suggested that one of the subgraphs, suggestively referred to as the 'filter' within this paper, consists of a small set of the most recent states. This allows fast, and even constant-time, updates of the current navigation estimate using newly received sensor measurements. The second subgraph, referred to as the 'smoother', consists of all other states and the original nonlinear observations obtained from the filter. By applying a full smoothing inference algorithm to the smoother subgraph in a background thread, the full smoothing solution of the entire joint density can be recovered without impacting the filter update cycle. In addition to recovering the optimal estimate, the presence of the smoother allows the incorporation of arbitrary loop closures into the optimization.

The cost of this parallelization is only a time delay between the instant a loop closure constraint is identified and the time the filter is adjusted accordingly. Most of this time delay is inherent to any parallel architecture running in real-time. Before the output navigation estimate can be impacted by the loop closure, the loop closure must first be processed by the smoother. Thus, any parallel real-time system will be delayed by at least the smoother update time. Further, if the smoother is updated constantly, then the system must also wait for the current smoother optimization to complete before inserting the loop closure and optimizing again. However, there is an additional delay in the CFS induced by its factorization. In order to maintain the conditional independence of the two subgraphs needed for concurrent updates, no linkage can be added between the filter variables and the smoother variables. This means that the system must also wait for the loop closure variable to propagate from the filter to the smoother before the constraint is added to the smoother. For fixed-lag smoother implementations, this is simply the smoother lag. Thus, the system designer does have control over the length of this additional lag.

A practical CFS system optimizes the full joint density by performing iterative, nonlinear least-squares optimization on the two independent subgraphs, then exchanging updated information by synchronizing the subgraphs. This synchronization may be viewed as the final step in a single optimization iteration over the *entire* graph. As such, it is generally better for the CFS system to perform synchronization more frequently with less optimal subgraphs, rather than waiting for the two subgraphs to fully converge before synchronization. Thus, the smoother implementation used throughout this paper was designed to perform only a single optimizer iteration before synchronizing with the filter.

The CFS architecture was demonstrated on both simulated data of an aerial vehicle and on real data collected from a ground vehicle. A Monte Carlo study was conducted using the simulation system, showing that the CFS system does recover the full batch solution, except during the system delays around loop closure events. Both the current state estimate and the covariances were examined. Similar results were obtained when working with the real data: the CFS estimate closely tracks the full batch estimate, except for some delay around loop closures. The computational requirements of the different CFS operations were examined, showing no significant difference between the CFS filter computation time and the computation time of an equivalent fixed-lag smoother, both well below the update rate of the involved sensors. The synchronization times of the CFS were also shown to be constant over the entire dataset, and well below the update rate of the involved sensors. This enables the CFS system to operate in real-time, even while the computation of the smoother grows approximately linearly.

8.1 Future Work

An obvious extension to the presented implementation would be to replace the batch optimization of the smoother branch with an incremental inference implementation, such as iSAM2 Kaess et al. (2012a). This should dramatically alter the time required for performing most of the smoother updates, converting the approximately linear-time complexity update operation shown in Figure 27 into an approximately constant-time update operation. Even though iSAM2 will still require significant processing time when loop closures are incorporated, the vast majority of the trajectory would be improved by more frequent synchronizations with significantly smaller time delays.

A far more interesting extension of CFS system involves transforming the formulation of the measurement factors when passing from the filter to the smoother during synchronization. Since computational speed is critical for the filter branch, an approximate but fast factor formulation could be used inside the filter. For example, for stereo vision processing, all of the feature observations between consecutive frames could be used to generate a single relative pose constraint. However, when this factor passes from the filter to the smoother, it could be transformed into a large set of projection factors, allowing the smoother to perform full bundle adjustment of the trajectory and scene structure. The factorization of the filter and smoother branch, presented in Section 5, specify conditional and marginal densities given specific sets of measurements. Since the relative pose constraint used in the filter and the set of projection factors used in the smoother involve identical measurements (a set of pixel locations), the CFS factorization is not violated in any way by this factor transformation. This method could allow a designer to make use of time-saving approximate formulations for many sensors, knowing that the filter output would ultimately be corrected by the smoother using the true measurement formulations.

Acknowledgements

We would like to thank Rakesh Kumar and Supun Samarasekera at Sarnoff/SRI International for their support and valuable discussions. This work was supported by the all source positioning and navigation (ASPN) program of the Air Force Research Laboratory (AFRL) under contract FA8650-11-C-7137. The views expressed in this work have not been endorsed by the sponsors.

References

Aharoni, R. and Berger, E. (2009). MengerÕs theorem for infinite graphs. *Inventiones mathematicae*, 176(1):1–62.

Bar-Shalom, Y. (2002). Update with out-of-sequence measurements in tracking: exact solution . *IEEE Trans. Aerosp. Electron. Syst.*, 38:769–777.

Bar-Shalom, Y. and Li, X. (1995). *Multitarget-multisensor tracking: principles and techniques.* YBS Publishing.

Davis, T., Gilbert, J., Larimore, S., and Ng, E. (2004). A column approximate minimum degree ordering algorithm. *ACM Trans. Math. Softw.*, 30(3):353–376.

Dellaert, F. and Kaess, M. (2006). Square Root SAM: Simultaneous localization and mapping via square root information smoothing. *Intl. J. of Robotics Research*, 25(12):1181–1203.

Eustice, R., Singh, H., and Leonard, J. (2006). Exactly sparse delayed-state filters for view-based SLAM. *IEEE Trans. Robotics*, 22(6):1100–1114.

Farrell, J. (2008). *Aided Navigation: GPS with High Rate Sensors.* McGraw-Hill.

Folkesson, J. and Christensen, H. (2004). Graphical SLAM – a self-correcting map. In *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, volume 1, pages 383–390.

Geiger, A., Lenz, P., and Urtasun, R. (2012). Are we ready for autonomous driving? the KITTI vision benchmark suite. In *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, pages 3354–3361, Providence, USA.

Grisetti, G., Stachniss, C., Grzonka, S., and Burgard, W. (2007). A tree parameterization for efficiently computing maximum likelihood maps using gradient descent. In *Robotics: Science and Systems (RSS)*.

Heggernes, P. and Matstoms, P. (1996). Finding good column orderings for sparse QR factorization. In *Second SIAM Conference on Sparse Matrices*.

Indelman, V., Melim, A., and Dellaert, F. (2013a). Incremental light bundle adjustment for robotics navigation. In *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*.

Indelman, V., Wiliams, S., Kaess, M., and Dellaert, F. (2012). Factor graph based incremental smoothing in inertial navigation systems. In *Intl. Conf. on Information Fusion, FUSION*.

Indelman, V., Wiliams, S., Kaess, M., and Dellaert, F. (2013b). Information fusion in navigation systems via factor graph based incremental smoothing. *Robotics and Autonomous Systems*, 61(8):721–738.

Jones, E. and Soatto, S. (2011). Visual-inertial navigation, mapping and localization: A scalable real-time causal approach. *Intl. J. of Robotics Research*, 30(4).

Kaess, M., Ila, V., Roberts, R., and Dellaert, F. (2010a). The Bayes tree: An algorithmic foundation for probabilistic robot mapping. In *Intl. Workshop on the Algorithmic Foundations of Robotics*.

Kaess, M., Ila, V., Roberts, R., and Dellaert, F. (2010b). The Bayes tree: Enabling incremental reordering and fluid relinearization for online mapping. Technical Report MIT-CSAIL-TR-2010-021, Computer Science and Artificial Intelligence Laboratory, MIT.

Kaess, M., Johannsson, H., Roberts, R., Ila, V., Leonard, J., and Dellaert, F. (2011). iSAM2: Incremental smoothing and mapping with fluid relinearization and incremental variable reordering. In *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, Shanghai, China.

Kaess, M., Johannsson, H., Roberts, R., Ila, V., Leonard, J., and Dellaert, F. (2012a). iSAM2: Incremental smoothing and mapping using the Bayes tree. *Intl. J. of Robotics Research*, 31:217–236.

Kaess, M., Wiliams, S., Indelman, V., Roberts, R., Leonard, J., and Dellaert, F. (2012b). Concurrent filtering and smoothing. In *Intl. Conf. on Information Fusion, FUSION*.

Klein, G. and Murray, D. (2007). Parallel tracking and mapping for small AR workspaces. In *IEEE and ACM Intl. Sym. on Mixed and Augmented Reality (ISMAR)*, pages 225–234, Nara, Japan.

Konolige, K. and Agrawal, M. (2008). FrameSLAM: from bundle adjustment to realtime visual mapping. *IEEE Trans. Robotics*, 24(5):1066–1077.

Konolige, K., Grisetti, G., Kuemmerle, R., Burgard, W., Benson, L., and Vincent, R. (2010). Efficient sparse pose adjustment for 2D mapping. In *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, pages 22–29, Taipei, Taiwan.

Kschischang, F., Frey, B., and Loeliger, H.-A. (2001). Factor graphs and the sum-product algorithm. *IEEE Trans. Inform. Theory*, 47(2).

Kullback, S. and Leibler, R. A. (1951). On information and sufficiency. *The Annals of Mathematical Statistics*, pages 79–86.

Lowe, D. (1999). Object recognition from local scale-invariant features. In *Intl. Conf. on Computer Vision (ICCV)*, pages 1150–1157.

Lu, F. and Milios, E. (1997). Globally consistent range scan alignment for environment mapping. *Autonomous Robots*, pages 333–349.

Lupton, T. and Sukkarieh, S. (2012). Visual-inertial-aided navigation for high-dynamic motion in built environments without initial conditions. *IEEE Trans. Robotics*, 28(1):61–76.

Mahon, I., Williams, S., Pizarro, O., and Johnson-Roberson, M. (2008). Efficient view-based SLAM using visual loop closures. *IEEE Trans. Robotics*, 24(5):1002–1014.

Maybeck, P. (1979). *Stochastic Models, Estimation and Control*, volume 1. Academic Press, New York.

Mei, C., Sibley, G., Cummins, M., Newman, P., and Reid, I. (2011). RSLAM: A system for large-scale mapping in constant-time using stereo. *Intl. J. of Computer Vision*, 94(2):198–214.

Menger, K. (1927). Zur allgemeinen kurventheorie. *Fundamenta Mathematicae*, 10(1):96–115.

Mourikis, A. and Roumeliotis, S. (2007). A multi-state constraint Kalman filter for vision-aided inertial navigation. In *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, pages 3565–3572.

Mourikis, A. and Roumeliotis, S. (2008). A dual-layer estimator architecture for long-term localization. In *Proc. of the Workshop on Visual Localization for Mobile Platforms at CVPR*, Anchorage, Alaska.

Moutarlier, P. and Chatila, R. (1989). An experimental system for incremental environment modelling by an autonomous mobile robot. In *Experimental Robotics I, The First International Symposium, Montréal, Canada, June 19-21, 1989*, pages 327–346.

Newcombe, R., Davison, A., Izadi, S., Kohli, P., Hilliges, O., Shotton, J., Molyneaux, D., Hodges, S., Kim, D., and Fitzgibbon, A. (2011a). Kinect-Fusion: Real-time dense surface mapping and tracking. In *IEEE and ACM Intl. Sym. on Mixed and Augmented Reality (ISMAR)*, pages 127–136, Basel, Switzerland.

Newcombe, R., Lovegrove, S., and Davison, A. (2011b). DTAM: Dense tracking and mapping in real-time. In *Intl. Conf. on Computer Vision (ICCV)*, pages 2320–2327, Barcelona, Spain.

Ranganathan, A., Kaess, M., and Dellaert, F. (2007). Fast 3D pose estimation with out-of-sequence measurements. In *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, pages 2486–2493, San Diego, CA.

Shen, X., Son, E., Zhu, Y., and Luo, Y. (2009). Globally optimal distributed Kalman fusion with local out-of-sequence-measurement updates. *IEEE Transactions on Automatic Control*, 54(8):1928–1934.

Sibley, G., Mei, C., Reid, I., and Newman, P. (2009). Adaptive relative bundle adjustment. In *Robotics: Science and Systems (RSS)*.

Smith, D. and Singh, S. (2006). Approaches to multisensor data fusion in target tracking: A survey. *IEEE Transactions on Knowledge and Data Engineering*, 18(12):1696.

Smith, R., Self, M., and Cheeseman, P. (1988). A stochastic map for uncertain spatial relationships. In *Proc. of the Intl. Symp. of Robotics Research (ISRR)*, pages 467–474.

Smith, R., Self, M., and Cheeseman, P. (1990). Estimating uncertain spatial relationships in Robotics. In Cox, I. and Wilfong, G., editors, *Autonomous Robot Vehicles*, pages 167–193. Springer-Verlag.

Thrun, S., Burgard, W., and Fox, D. (2005). *Probabilistic Robotics*. The MIT press, Cambridge, MA.

Triggs, B., McLauchlan, P., Hartley, R., and Fitzgibbon, A. (2000). Bundle adjustment – a modern synthesis. In Triggs, W., Zisserman, A., and Szeliski, R., editors, *Vision Algorithms: Theory and Practice*, volume 1883 of *LNCS*, pages 298–372. Springer Verlag.

Vial, J., Durrant-Whyte, H., and Bailey, T. (2011). Conservative sparsification for efficient and consistent approximate estimation. In *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, pages 886–893. IEEE.

Zahng, S. and Bar-Shalom, Y. (2011). Optimal update with multiple out-of-sequence measurements. In *Proc. of the SPIE, Signal Processing, Sensor Fusion, and Target Recognition XX*.

Zhu, Z., Oskiper, T., Samarasekera, S., Kumar, R., and Sawhney, H. (2007). Ten-fold improvement in visual odometry using landmark matching. In *Intl. Conf. on Computer Vision (ICCV)*.