The Bayes Tree: Enabling Incremental Reordering and Fluid Relinearization for Online Mapping

Michael Kaess, Viorela Ila, Richard Roberts, and Frank Dellaert

CSAIL

The Bayes Tree: Enabling Incremental Reordering and Fluid Relinearization for Online Mapping

Michael Kaess*, Viorela Ila†, Richard Roberts†, and Frank Dellaert†
*CSAIL, Massachusetts Institute of Technology, Cambridge, Massachusetts
†School of Interactive Computing, Georgia Institute of Technology, Atlanta, Georgia
Email: kaess@mit.edu, {vila, richard, frank}@cc.gatech.edu

Abstract

In this paper we present a novel data structure, the Bayes tree, which exploits the connections between graphical model inference and sparse linear algebra. The proposed data structure provides a new perspective on an entire class of simultaneous localization and mapping (SLAM) algorithms. Similar to a junction tree, a Bayes tree encodes a factored probability density, but unlike the junction tree it is directed and maps more naturally to the square root information matrix of the SLAM problem. This makes it eminently suited to encode the sparse nature of the problem, especially in a smoothing and mapping (SAM) context. The inherent sparsity of SAM has already been exploited in the literature to produce efficient solutions in both batch and online mapping. The graphical model perspective allows us to develop a novel incremental algorithm that seamlessly incorporates reordering and relinearization. This obviates the need for expensive periodic batch operations from previous approaches, which negatively affect the performance and detract from the intended online nature of the algorithm. The new method is evaluated using simulated and real-world datasets in both landmark and pose SLAM settings.

I. Introduction

Simultaneous localization and mapping (SLAM) refers to the problem of localizing a robot while simultaneously mapping its environment. The earliest and most popular SLAM methods employ the extended Kalman filter (EKF) [35, 28] which recursively estimates a Gaussian density over the current pose of the robot and the map. The computational complexity of the EKF becomes intractable fairly quickly, and hence a large number of efforts have focused on modifying and extending the filtering approach to cope with larger-scale environments [10, 19, 31, 2, 39, 37, 7]. However, *filtering* itself has been shown to be inconsistent when applied to the inherently non-linear SLAM problem [21]. Since this is mainly due to linearization choices that cannot be undone in a filtering framework, there has recently been considerable interest in the *smoothing* version of the SLAM problem [7, 8, 24].

Smoothing and Mapping (SAM) is the problem of not just estimating the map and robot's most current pose, but also the entire robot trajectory up to the current time. This problem is also referred to as the *full SLAM* problem by Thrun et al. [40]. Dellaert et

al. [7, 8, 26, 24] have shown how smoothing can be an efficient alternative to filtering-based methods, as in many cases keeping the trajectory around helps rather than hurts. The matrices associated with smoothing are typically very sparse, as opposed to the dense problems one obtains in filtering. Because of this sparsity, one can do much better than the cubic complexity associated with factorizing a dense matrix [26].

Kaess et al. [23, 24] introduced incremental smoothing and mapping (iSAM), which performs *fast incremental updates* of the square root information matrix, yet is able to compute the full map and trajectory at any time. New measurements are added using matrix update equations [18], so that previously calculated components of the square root information matrix are reused. Updating of matrix factorizations is a well-known technique in other areas, with applications in computer vision [25] and signal processing [29]. Golub and Loan [18] present general methods for updating matrix factorizations based on [17, 16], including the Givens rotation method used in iSAM.

However, incremental SAM suffers from two important drawbacks related to variable reordering and relinearization. First, for trajectories with loops, Kaess et al. [23, 24] need to perform *periodic variable reordering* to prevent unnecessary fill-in of the square root information matrix that would otherwise slow down the algorithm. Second, in non-linear problems such as SLAM the entire graph has to be periodically relinearized around the latest estimate to prevent inconsistency. In practice reordering and relinearization are combined in a periodic "batch reset" of incremental SAM which is expensive and detracts from the intended online nature of the algorithm.

In this paper **we propose a novel data structure, the Bayes tree, and a novel incremental factorization method which allows for incremental reordering and just-in-time relinearization.** Taking a graphical model perspective of the SLAM problem offers the possibility of creating more efficient methods to find the optimal map solution given noisy sensor readings. Looking at SLAM in terms of graphs has a rich history [3] and has especially lead to several novel and exciting developments in the past few years [11, 14, 12, 13].

Our approach is based on viewing matrix factorization as eliminating a factor graph into a Bayes net, which is the graphical model equivalent of the *square root information matrix*. Performing marginalization and optimization in Bayes nets is not easy in general. However, a Bayes net resulting from elimination/factorization is *chordal,* and it is well know that a chordal Bayes net can be converted into a tree-structured graphical model in which these operations are easy. The most well-known such data structure is the *clique tree* [34, 1], also known as the *junction tree* in the AI literature [5], which has already been exploited for distributed inference in SLAM [9, 31]. However, the new data structure we propose here, the **Bayes tree**, is *directed* and corresponds more naturally to the result of factorization in linear algebra. Exploiting this new data structure we propose an efficient algorithm for incremental factorization in on-line mapping.

II. PROBLEM STATEMENT

We use a *factor graph* [27] to represent the SLAM problem in terms of graphical models. Formally, 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* $\theta_j \in \Theta$. Edges $e_{ij} \in \mathcal{E}$ are always between factor nodes
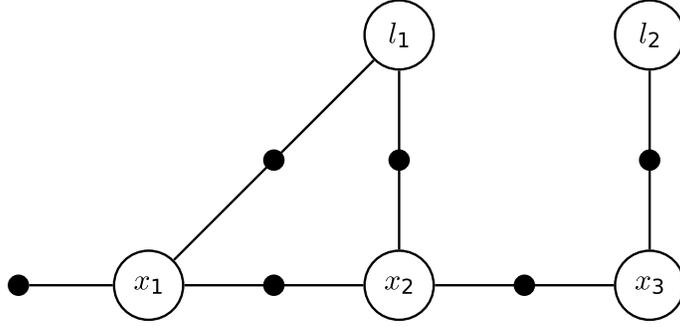
Figure 1: Factor graph for a small SLAM example, where a robot located at successive poses x_1, x_2, and x_3 makes observations on landmarks l_1 and l_2. In addition there is an absolute measurement on the pose x_1.

and variables nodes. A factor graph G defines the factorization of a function $f(\Theta)$ as

$$f(\Theta) = \prod_i f_i(\Theta_i) \tag{1}$$

where Θ_i is the set of variables θ_j adjacent to the factor f_i, and independence relationships are encoded by the edges e_{ij}: each factor f_i is a function of the variables in Θ_i. An example is shown in Figure 1.

When assuming Gaussian process and measurement models, as is standard in the SLAM literature [36, 4, 10], the factored objective function (1) we want to minimize corresponds to the nonlinear least-squares criterion

$$\log f(\Theta) = \frac{1}{2} \sum_i \|h_i(\Theta) - z_i\|^2 \tag{2}$$

The individual factors correspond to (a) the motion model,

$$\|g_t(x_{t-1}, u_t) - x_t\|_{\Lambda_t}^2 \tag{3}$$

where $g_t(.)$ is the process model at time t, corrupted by normally distributed zero-mean process noise with covariance matrix Λ_t, and (b) the measurement model,

$$\|h_{tk}(x_t, l_{tk}) - z_{tk}\|_{\Sigma_{tk}}^2 \tag{4}$$

where $h_{tk}(.)$ is a measurement equation for measurement k at time t, corrupted by normally distributed zero-mean measurement noise with covariance Σ_{tk}. Above $\|e\|_\Sigma^2 \triangleq e^T \Sigma^{-1} e$ is defined as the squared Mahalanobis distance given a covariance matrix Σ.

A crucial insight is that **inference can be understood as converting the factor graph to a Bayes net using the elimination algorithm.** Variable elimination [1, 5] originated in order to solve systems of linear equations, and was first applied in modern times by Gauss in the early 1800s [15].

In factor graphs, elimination is done via a *bipartite elimination game*, as described by Heggernes and Matstoms [20]. This can be understood as taking apart the factor graph and transforming it into a *Bayes net* [33]. One proceeds by eliminating one variable at a time,
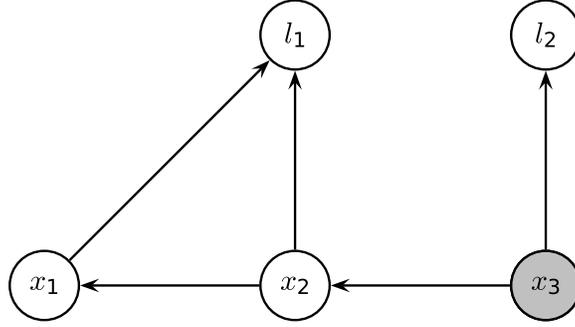
Figure 2: Bayes net resulting from eliminating the example factor graph in Figure 1 using the elimination ordering l_1, l_2, x_1, x_2, x_3. Note the *root* is shaded darker.

Algorithm 1 Eliminating variable θ_j from the factor graph.

1) Remove from the factor graph all factors $f_i(\Theta_i)$ that are adjacent to θ_j. Define the *separator* S_j as all variables involved in those factors, excluding θ_j.
2) Form the (unnormalized) joint density $f_{joint}(\theta_j, S_j) = \prod_i f_i(\Theta_i)$ as the product of those factors.
3) Using the chain rule, factorize the joint density $f_{joint}(\theta_j, S_j) = P(\theta_j|S_j)f_{new}(S_j)$. Add the conditional $P(\theta_j|S_j)$ to the Bayes net and the factor $f_{new}(S_j)$ back into the factor graph.

---

and converting it into a node of the Bayes net, which is gradually built up. After eliminating each variable, the reduced factor graph defines a density on the remaining variables. The pseudo-code for eliminating a variable θ_j is given in Algorithm 1. After eliminating all variables the Bayes net density is defined by the product of the conditionals produced at each step:

$$P(X) = \prod_j P(\theta_j|S_j) \tag{5}$$

The result of this process for the example in Figure 1 is shown in Figure 2.

A. Equivalence of Inference in Factor Graphs and QR

In Gaussian factor graphs, elimination is equivalent to sparse QR factorization of the measurement Jacobian. In practice one always considers a linearized version of problem (2). If the process models f_i and measurement equations h_k are non-linear and a good linearization point is not available, non-linear optimization methods such as Gauss-Newton iterations or the Levenberg-Marquardt algorithm will solve a succession of linear approximations to (2) in order to approach the minimum.

At each iteration of the non-linear solver, we linearize around a linearization point Θ_0 to get a new, *linear* least-squares problem in X with the objective function

$$\log f(X) = \frac{1}{2} \|AX - b\|^2 \tag{6}$$

4

where $A \in \mathbb{R}^{m \times n}$ is the measurement Jacobian consisting of m measurement rows and X is an n-dimensional tangent vector. The matrix A above is a sparse block-matrix, and its graphical model counterpart is a *Gaussian* factor graph with exactly the same structure as the non-linear factor graph. The probability density on X defined by this factor graph is the normal distribution

$$P(X) \propto e^{-\log f(X)} = \exp \left\{ -\frac{1}{2} \|AX - b\|^2 \right\} \tag{7}$$

In Gaussian factor graphs, the chain rule $f_{joint}(x_j, S_j) = P(x_j | S_j) f_{new}(S_j)$ in step 3 of Algorithm 1 can be implemented using Householder reflections or a Gram-Schmidt orthogonalization, in which case the entire elimination algorithm is equivalent to QR factorization of the entire measurement matrix A. To see this, note that the factor $f_{joint}(x_j, S_j)$ for $x_j \in \mathbb{R}$ defines a Gaussian density

$$f_{joint}(x_j, S_j) \propto \exp \left\{ -\frac{1}{2} \|ax_j + A_S S_j - b\|_{\Sigma}^2 \right\} \tag{8}$$

where the (dense but small) matrix $A_j = [a | A_S]$ is obtained by concatenating the vectors of partial derivatives of all factors connected to variable x_j. The desired conditional $P(x_j | S_j)$ is obtained by evaluating the joint (8) for a fixed value of S_j, yielding

$$P(x_j | S_j) \propto \exp \left\{ -\frac{1}{2\sigma^2} (x_j + r S_j - d)^2 \right\} \tag{9}$$

with $r \triangleq a^\dagger A_S$ and $d \triangleq a^\dagger b$, where $a^\dagger \triangleq \left(a^T \Sigma^{-1} a\right)^{-1} \Sigma^{-1} a^T$ is the *pseudo-inverse* of a, and the inverse variance $\sigma^{-2} = a^T \Sigma^{-1} a$. The new factor $f_{new}(S_j)$ is obtained by substituting $x_j = d - r S_j$ back into (8):

$$f_{new}(S_j) = \exp \left\{ -\frac{1}{2} \|A' S_j - b'\|_{\Sigma}^2 \right\} \tag{10}$$

where $A' \triangleq A_S - ar$ and $b' \triangleq b - ad$. The above is one step of Gram-Schmidt, interpreted in terms of densities, and the sparse vector r and scalar d can be recognized as specifying a single joint conditional density in the Bayes net, or alternatively a single row in the sparse square root information matrix.

III. THE BAYES TREE

The Bayes net resulting from elimination/factorization is *chordal,* and it can be converted into a tree-structured graphical model in which optimization and marginalization are easy. In this paper we introduce a new data structure, **the Bayes tree**, to better capture the equivalence with linear algebra and enable new algorithms in recursive estimation. A Bayes tree is a directed tree where the nodes represent *cliques* C_k of the underlying chordal Bayes net. In this respect Bayes trees are similar to clique trees, but a Bayes tree is directed and is closer to a Bayes net in the way it encodes a factored probability density. In particular, we define one conditional density $P(F_k | S_k)$ per node, with the *separator* S_k as the intersection $C_k \cap \Pi_k$ of the clique C_k and its parent clique Π_k, and the *frontal variables* F_k as the
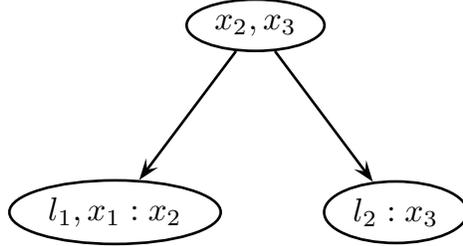
Figure 3: Bayes tree describing the clique structure in the Bayes net from Figure 2. A Bayes tree is similar to a junction tree, but is better at capturing the formal equivalence between sparse linear algebra and inference in graphical models.

Algorithm 2 Creating a Bayes tree from the Bayes net that was generated by eliminating the factor graph.

For each conditional density $P(\theta_j | S_j)$ of the Bayes net, in *reverse* elimination order:
If no parent ($S_i = \{\}$)
    start a new root clique F_r
else
    if frontal nodes of parent clique F_p is equal to S_i of conditional
        insert conditional into clique
    else
        start new clique

---

remaining variables, i.e. $F_k \overset{\Delta}{=} C_k \setminus S_k$. We write $C_k = F_k : S_k$. This leads to the following expression for the joint density $P(\Theta)$ on the variables Θ defined by a Bayes tree,

$$P(\Theta) = \prod_k P(F_k | S_k) \tag{11}$$

where for the root F_r the separator is empty, i.e., it is a simple prior $P(F_r)$ on the root variables.

Every chordal Bayes net can be transformed into a tree by discovering its cliques. Discovering cliques in chordal graphs is done using the maximum cardinality search algorithm by Tarjan and Yannakakis [38], which proceeds in reverse elimination order to discover cliques in the Bayes net. The algorithm for converting a Bayes net into a Bayes tree is summarized in Algorithm 2 and the corresponding Bayes tree for the small SLAM example in Figure 1 is shown in Figure 3.

IV. BAYES TREES FOR ONLINE MAPPING

In this section we use the Bayes tree in a novel algorithm for optimizing a set of nonlinear factors that grows over time, which is directly applicable to online mapping. We first present an algorithm for the case of linear constraints, and show how the Bayes tree is updated with those new factors. We then discuss how to perform relinearization where needed, a process
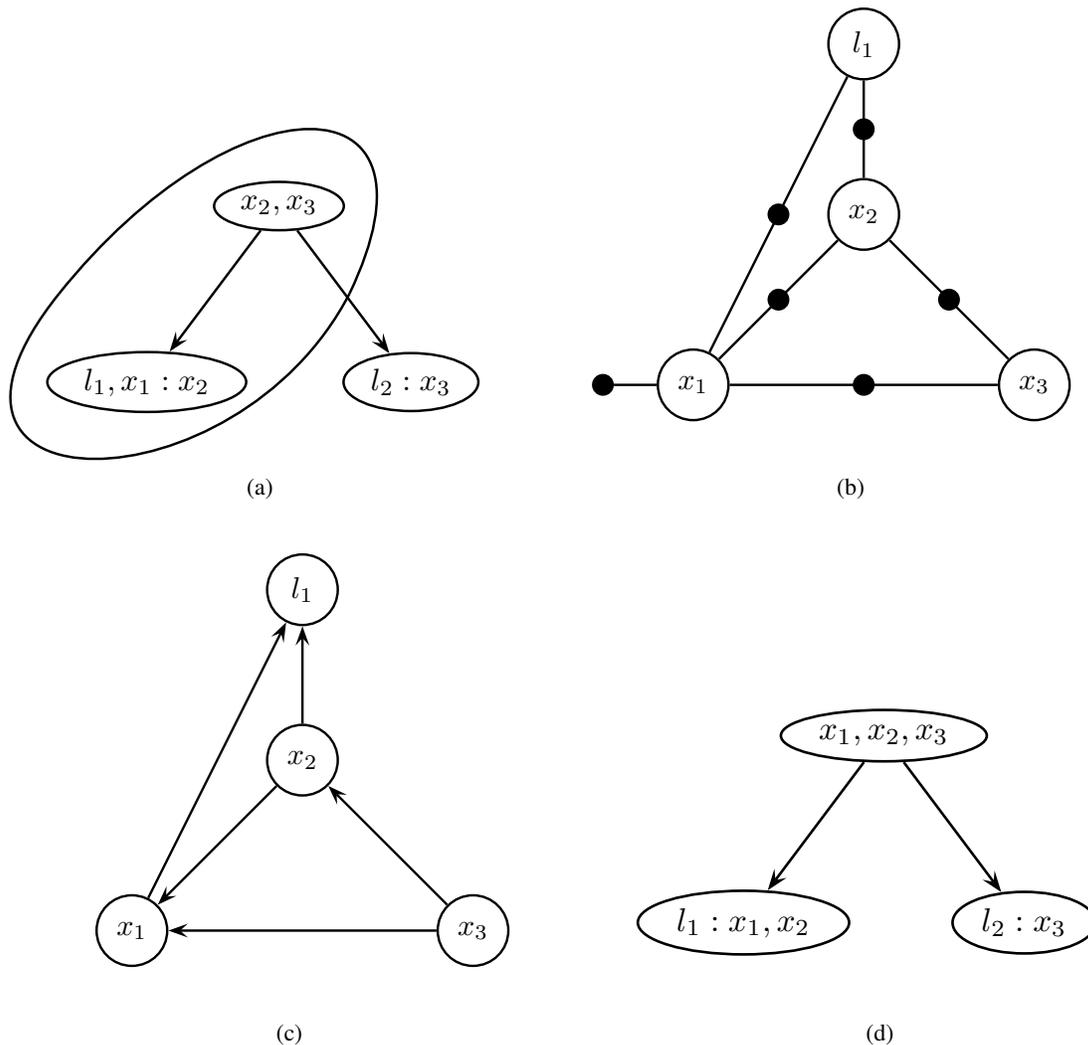
Figure 4: Updating a Bayes tree with a new factor, based on the example in Figure 3. (a) The affected part of the Bayes tree is highlighted for the case of adding a new factor between x_1 and x_3. Note that the right branch is not affected by the change. (b) The factor graph generated from the affected part of the Bayes tree. (c) The chordal Bayes net resulting from eliminating the factor graph. (d) The Bayes tree created from the chordal Bayes net, with the unmodified right branch from (a) added back in.

that we call *fluid relinearization*. We present a combined algorithm for adding nonlinear factors over time, while keeping the Bayes tree and the estimate up to date.

Incremental SAM The main technical idea is a new view of incremental factorization in terms of graphical models. In particular, we will now store and compute the square root information matrix R in the form of a Bayes tree $\mathcal{T}$. Incremental factorization/inference can be seen as converting the top part of the Bayes tree back to a factor graph, re-factoring only that part, and re-constituting the Bayes tree. To understand why only the top part of

---

Algorithm 3 Updating the Bayes tree with new factors $\mathcal{F}'$.

1) Remove top of Bayes tree and convert to a factor graph:
   a) For each affected variable, remove the corresponding clique and all parents up to the root.
   b) Store orphaned sub-trees $\mathcal{T}_O$ of removed cliques.
2) Add the new factors $\mathcal{F}'$ into the resulting factor graph
3) Re-order and eliminate the factor graph into a Bayes net (Algorithm 1), and re-assemble into a new Bayes tree (Algorithm 2)
4) Insert the orphans back into the new Bayes tree.

---

the Bayes tree is affected, we discuss the case of adding a new measurement represented by the factor $f'(x_j, x_{j'})$ between two variables x_j and $x_{j'}$. A property of the Bayes tree is that this will only affect the paths between those variables and the root. The sub-trees below the cliques containing x_j and $x_{j'}$ remain unaffected, as is any sub-tree of the root not containing x_j or $x_{j'}$.

Figure 4 (a) shows that adding the new factor between $\mathbf{x}_1$ and $\mathbf{x}_3$ in the small SLAM example from Figure 3 only affects the left branch of the tree. It is easy to convert the affected part of the Bayes tree back to a factor graph, add the new constraint(s), re-order, and re-eliminate. The unaffected sub-trees can then be reattached to the new Bayes tree. Figure 4(a)-(d) shows how these steps are applied to the small SLAM example. The entire process of updating the Bayes tree with a new factor is described by Algorithm 3.

Incremental Reordering Using the Bayes tree structure, reordering can be done at every incremental update, eliminating the need for periodic batch reordering. This was not understood in [24], because this is only obvious within the graphical model framework, but not for matrices. In addition, by maintaining a Bayes tree data structure in an online mapping framework, we inherit its beneficial properties in terms of marginalization and optimization, which can both be done in an easy, recursive manner. Optimization starts at the root and proceeds recursively through the children. Each clique that is visited is solved by backsubstitution, based on the internal representation arising from incomplete QR factorization.

Fluid Relinearization The idea behind just-in time or *fluid relinearization* is very similar to incremental updates, and indeed we combine both components into one algorithm. We can keep track of the validity of the linearization point for each variable, and only relinearize when needed. This represents a departure from the conventional linearize/solve approach that currently represents the state of the art, and can be viewed as a completely new algorithm for non-linear optimization.

For incremental updates in the previous section we have removed the affected parts of the Bayes tree, that were then updated, converted back to a Bayes tree, and combined with the unmodified sub-trees of the original Bayes tree. For relinearization we can follow much the same scheme. For a variable that is chosen to be relinearized, all relevant information has to be removed from the Bayes tree. The original nonlinear factors are then recovered and linearized at the new linearization point. Then we again follow the same path of updating with new factors, converting back to a Bayes tree, and finally combining with the orphaned

---
Algorithm 4 Full online mapping algorithm with incremental variable reordering and fluid relinearization.

Input / output: nonlinear factors $\mathcal{F}$, Linearization point Θ, Bayes tree $\mathcal{T}$, marked variables J, update X_J

Initialization: $\mathcal{F}_0 = \emptyset$, $\Theta_0 = \emptyset$, $\mathcal{T}_0 = \emptyset$, $J_0 = \emptyset$

Iterate whenever $J \neq \emptyset$ or new factors $\mathcal{F}'$ available:

1) Update linearization point for marked variables: $\Theta_J := \Theta_J + X_J$.
2) Add any new factors $\mathcal{F} := \mathcal{F} \cup \mathcal{F}'$.
3) Initialize any new variables Θ_{new} and add $\Theta := \Theta \cup \Theta_{new}$.
4) Also mark affected variables J' as invalid: $J := J \cup J'$.
5) Remove top plus one layer of children (see text) of Bayes tree $\mathcal{T}$ according to J, remembering orphaned sub-trees $\mathcal{T}_O$ and removed conditionals $J_{removed}$.
6) Find all nonlinear factors $\mathcal{F}_{core} = \{f \in \mathcal{F} | var(f) \subseteq J_{removed}\}$ that were removed above.
7) Linearize those factors and add cached factors from orphan roots into Gaussian factor graph: $G = linearize(\mathcal{F}_{core}, \Theta) + cached(\mathcal{T}_0)$.
8) Eliminate G and append orphans $\mathcal{T}_O$ to create a new Bayes tree $\mathcal{T}$ according to Algorithm 2.
9) Obtain $X = solve(\mathcal{T}, \alpha)$ by back-substitution with threshold α.
10) Mark variables as invalid if X_J above threshold: $J = \{x \in X | x \geq \beta\}$.
---

sub-trees.

Algorithm Summary The goal of our algorithm is to obtain an estimate Θ for the map and trajectory, given a set of nonlinear constraints that expands over time, represented by nonlinear factors $\mathcal{F}$. New factors $\mathcal{F}'$ can arrive at any time and may add new variables Θ_{new} to the estimation problem. We take the most recent estimate Θ as linearization point to solve a linearized system as subroutine in an iterative nonlinear optimization scheme. The linearized system is represented by the Bayes tree $\mathcal{T}$. The overall algorithm is shown in Algorithm 4, and a more detailed explanation follows.

In **step 1** we start by updating the linearization point for selected variables, which in the first iteration will not perform any operation - we will revisit this point at the end. In **step 2** we add the new factors $\mathcal{F}'$ to our existing and initially empty set of factors $\mathcal{F}$, which lists all nonlinear constraints we have encountered so far. Remembering the original factors $\mathcal{F}$ allows selective relinearization later. In **step 3** we initialize any new variables Θ_{new} based on the current linearization point, as predicted by the first measurement that is applied to it. The initial estimates are then added to the current linearization point Θ. In **step 4** we mark any variables connected to the new factors as invalid. Some of the variables may already have been marked as invalid in the previous iteration when the linearization point has been updated.

In **step 5** we remove the cliques corresponding to each invalidated variable J, including the complete path to the root, which is the same operation as for the linear case in Algorithm 3. However, this time we also remove each child clique of every removed clique, as variables to be relinearized can also be part of the child clique (but not further down

9

the chain based on the definition of the clique tree). Overall this leads to the removal of additional variables beyond the set J, and the full set of removed variables is summarized in $J_{removed}$. As in the linear update algorithm, we are left with a set of orphaned sub-trees $\mathcal{T}_O$ that are not affected by the changes and are remembered for later insertion back into the modified tree.

In contrast to the linear update algorithm, we cannot reuse the conditional densities that were removed from the Bayes tree, as they combine various factors, some of which are invalidated because of a change in linearization point for one of the involved variables. But how can we restart the elimination in the middle of the tree? From the original factorization we can see that all factors that only involve variables $J_{removed}$ that have been removed must have been eliminated inside the removed part. We call this set $\mathcal{F}_{removed}$, and it is recovered from the nonlinear factors $\mathcal{F}$ in **step 6**. But what about other factors that contain not only removed variables? These would have already been combined with other factors during the elimination process within the children. A summary of these factors is passed into the core part from the orphaned sub-trees. Those are the new factors f_{new} in Algorithm 1, that were created when eliminating the last variable for each of the orphaned sub-trees. Basically, if we were to re-eliminate all variables in the same order, those same factors would be passed in again from each orphaned sub-tree. As the orphaned sub-trees are not affected by our current changes, we can therefore simply cache those factors during elimination and reuse them now. In **step 7** those cached factors are combined with the linearized set of core factors $\mathcal{F}_{core}$ in a single Gaussian factor graph G that contains all the information that was removed in step 5, but now relinearized and updated with the new factors.

In **step 8** we create a Bayes tree from the Gaussian factor graph G that contains all relinearized and new information, using Algorithm 2. The new variable ordering makes sure that structural changes introduced by the new factors do not lead in larger clique sizes than necessary. Or in other words, it avoids fill-in in the sparse square root information matrix represented by the Bayes tree, but without recalculating the complete matrix as was necessary in previous work. The untouched parts of the original Bayes tree, the orphaned sub-trees $\mathcal{T}_O$ get inserted back into the modified core of the Bayes tree. Note that they may end up in different places than before, because the core was reorganized. The location to insert an orphaned tree is easy to find, as it is given by the clique corresponding to the separator variable of the orphaned tree that was eliminated first in the new ordering.

In **step 9** we solve the linearized system represented by the Bayes tree in a recursive fashion. Starting at the root, for which we have all information to recover the solution by backsubstitution, we proceed recursively through all children. At each point we already have a solution for the variables in the separator (parents). For large systems it can get expensive to recover the complete solution. Also, local changes in one part of the map typically only affect areas nearby, except for loop closings. We therefore stop propagation into the children once the change in a clique falls below a threshold α.

Relinearizing all variables at each step would result in a batch algorithm, as the complete system would have to be relinearized and re-eliminated. In **step 10** we determine when a change is significant enough to warrant a relinearization of the corresponding variable, based on a threshold β. Note that the update X always specifies the deviation of the estimate from the linearization point. Therefore undetected drifting over time is not possible, instead the update will eventually reach the threshold and the linearization point is updated. The
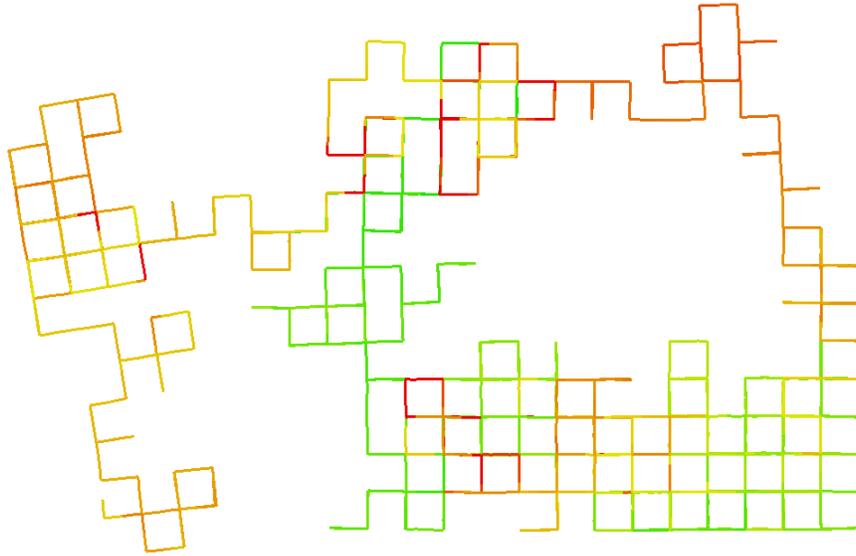
Figure 5: Manhattan world simulated environment after optimization. Color coded with the number of variables that are updated for every step along the trajectory. Green corresponds to a low number of cliques, red to a high number.

actual change in linearization point is deferred to step 1 of the next iteration, so that the returned Bayes tree $\mathcal{T}$ and linearization point Θ are always consistent. Therefore we can always obtain a solution by adding the current update X_J to the linearization point Θ. We can also obtain a full solution independent of thresholds by recovering a full update X, albeit at a higher cost.

V. EXPERIMENTAL RESULTS

This section describes the experiments that validate the presented approach, first using a synthetic dataset for which ground truth is available to evaluate the quality of the resulting map, and then using two publicly available real datasets. We compared our estimation and timing results with the state of the art incremental method in the literature [24] in order to highlight the advantages of fluid relinearization and incremental reordering. We used the column approximate minimum degree ordering (COLAMD) algorithm by Davis et al. [6]. The results are obtained with a C++ implementation running under Mac OS X on an Intel Core 2 at 2.4 GHz.

We have evaluated our algorithm on the Manhattan world from [30], courtesy of E. Olson. Using this simulated dataset, the quality of the estimated map is easy to appreciate, and, since the ground truth is available, adequate metrics are used to analyze the quality of the resulting map. We achieve the same final normalized chi-square value as the original algorithm, but intermediate results are often better due to more frequent updating of the linearization. Figure 5 shows the estimated trajectory for the simulated Manhattan world. The trajectory is colorized by the number of variables that had to be recalculated at that step, where green represents a small number of variables (order of 10), yellow a moderate number, and red finally a large number (order of hundreds of variables).
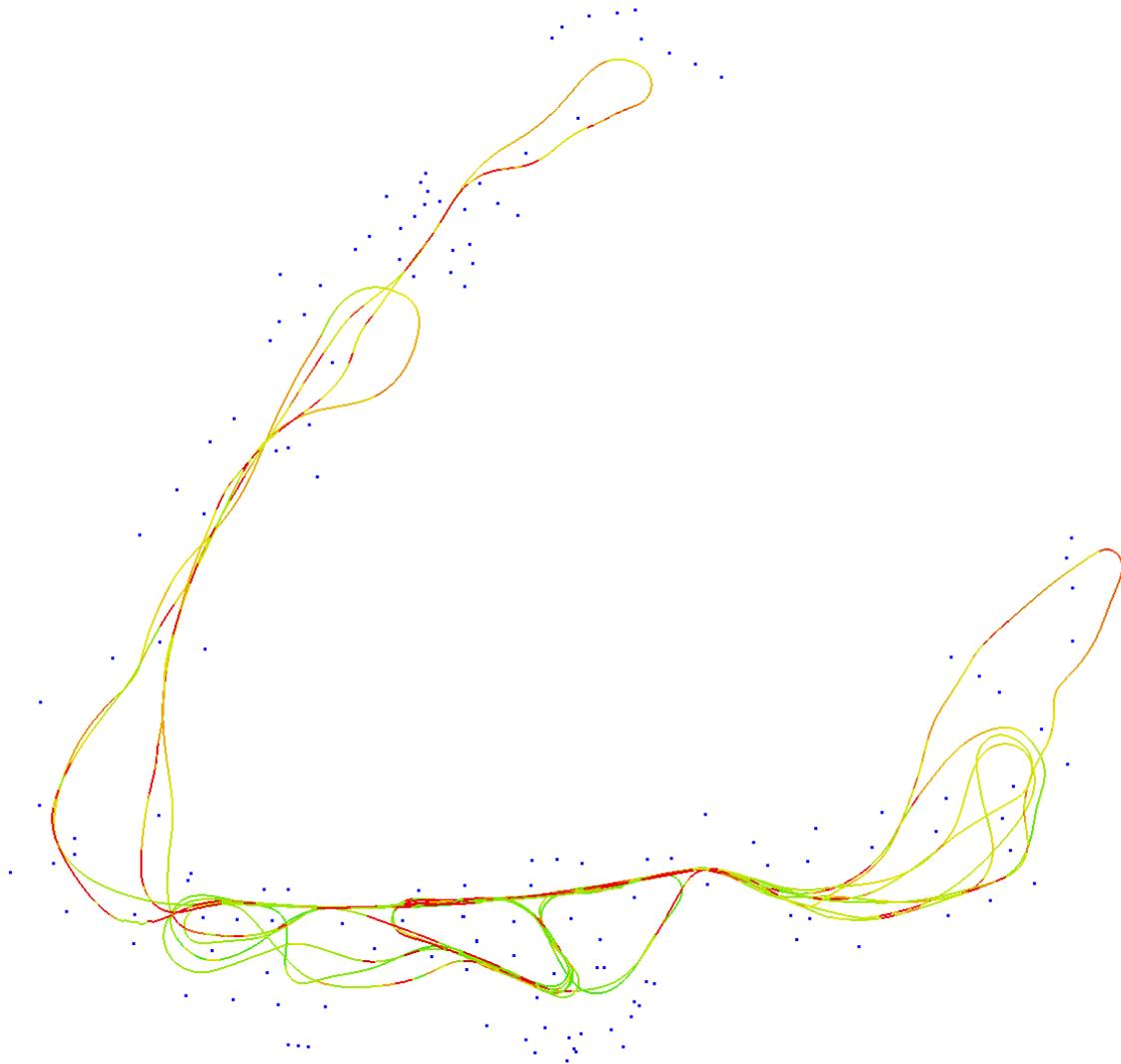
11

Figure 6: The Victoria Park dataset color coded with the number of variables that are updated for every step along the trajectory. Green corresponds to a low number of cliques, red to a high number.

We also applied our Bayes tree algorithm to the Victoria Park dataset, an often used benchmark SLAM dataset [24, 32] courtesy of E. Nebot and H. Durrant-Whyte. This dataset includes about 6900 laser scans with the corresponding odometry readings. The laser scans are processed to detect the trunks of the trees in the park, which are used as landmarks. Figure 6 shows the final trajectory estimate together with the detected landmarks, again using the same color coding for the number of variables that had to be recalculated in each step. A relatively small portion of the trajectory is colorized in intense red, mainly the part at the bottom where the vehicle closed loops multiple times, visiting the same location up to eight times. We finally also present results for the Intel dataset, courtesy of
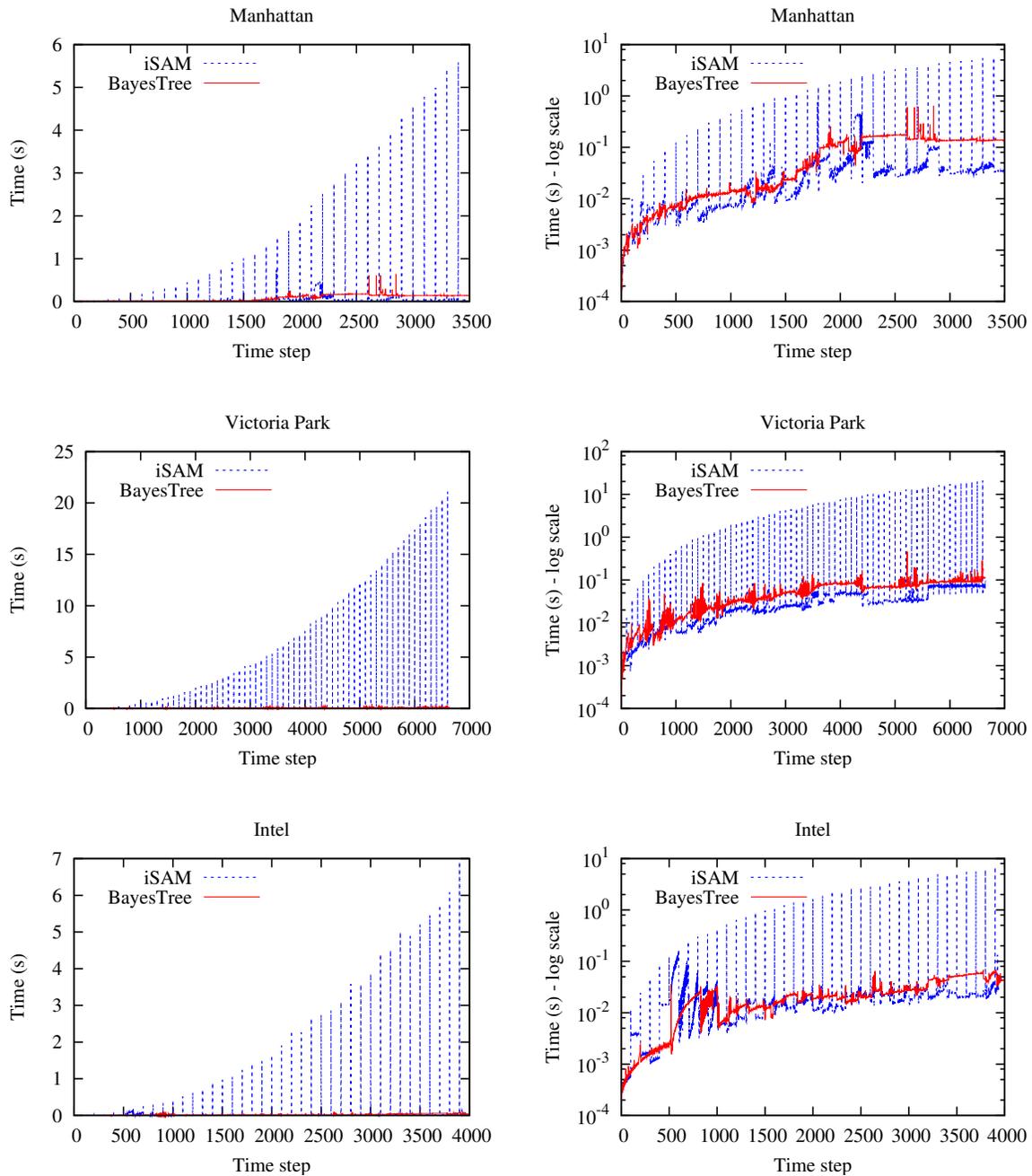
Figure 7: Timing comparison between our new algorithm (red) and the original iSAM algorithm (blue). The horizontal axis shows the step number, the vertical axis the time in seconds. The top row shows results for the Manhattan world data, the center row for the Victoria Park dataset and the bottom row for the Intel dataset. For each row, the same data are shown in linear scale on the left and log scale on the right. The original iSAM algorithm included a batch step every 100 iterations, which is clearly visible from the spikes.
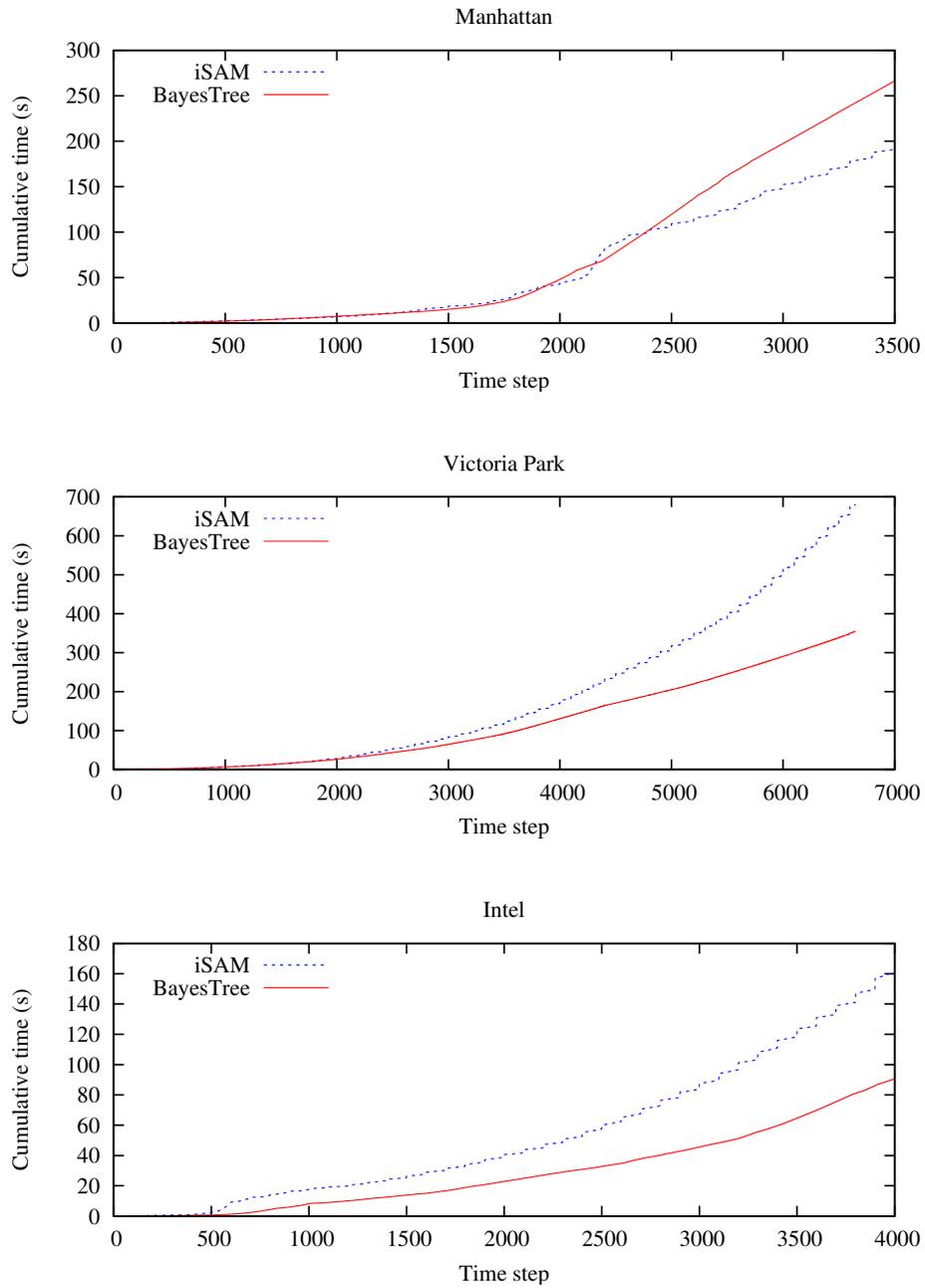
13

Figure 8: Cumulative time comparison for all three datasets. For two out of the three datasets our new algorithm provides an improvement in speed, in addition to its advantages of eliminating periodic batch operations and performing fluid relinearization.
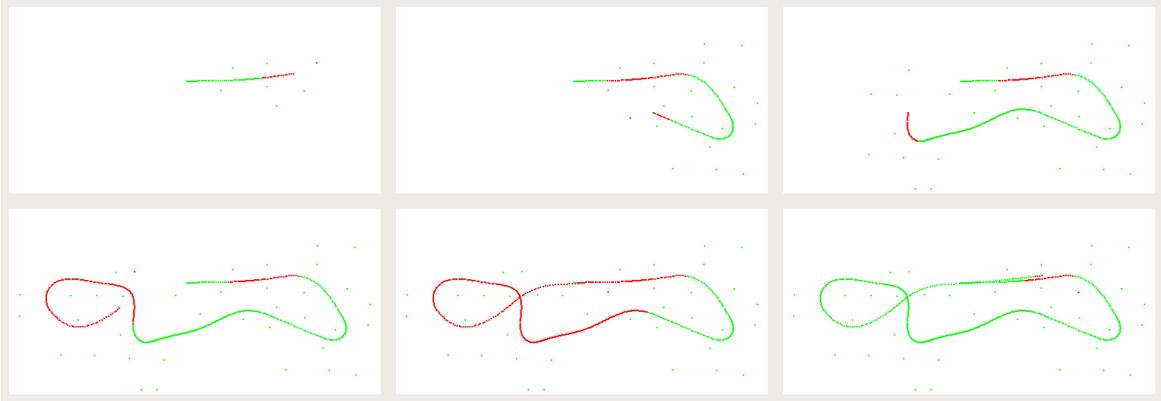
Figure 9: Illustration of the proposed novel graph-based incremental SAM algorithm on part of the Victoria Park data-set. The red nodes indicate which variables were affected by new measurement constraints at each iteration, and represent the subset of the factor graph that needs to be re-factorized. At times of loop closing (bottom center figure), larger parts of the trajectory might have to be recalculated. The number of affected variables for the complete dataset is shown by color coding in Figure 6.

D. Haehnel and D. Fox. This dataset was preprocessed by laser scan-matching, resulting in a pose graph formulation without landmarks, including about 4000 poses.

In Figure 7, we compare timing for both datasets between our algorithm and an implementation of the original incremental SAM algorithm [24]. The results show that our fully incremental algorithm does not suffer from the spikes caused by the batch relinearization step. Our algorithm also performs better in terms of cumulative time for two out of the three datasets, as shown in Figure 8, while providing the additional advantage of continuously updating the linearization point of all variables with significant changes. Further insights into the algorithm are provided by Figure 9, that shows the parts of the trajectory that are updated for select frames at the beginning of the Victoria Park sequence.

VI. CONCLUSION

We have presented a novel data structure, the Bayes tree, that provides new insights into previous online mapping algorithms based on sparse linear algebra. The graphical model perspective and insights in its detailed connection to linear algebra have lead us to a fully incremental algorithm that performs incremental variable reordering just-in-time, as well as selective relinearization. Our novel graph-based algorithm should also allow for better insights into the recovery of marginal covariances, as we believe that simple recursive algorithms in terms of the Bayes tree are formally equivalent to the dynamic programming methods described in [22]. The graph based structure also provides a starting point for exploiting parallelization that is becoming available in newer processors.

References

[1] J. Blair and B. Peyton. An introduction to chordal graphs and clique trees. In J. George, J. Gilbert, and J.-H. Liu, editors, *Graph Theory and Sparse Matrix Computations*, volume 56 of *IMA Volumes in Mathematics and its Applications*, pages 1–27. Springer-Verlag, New York, 1993.

[2] M. Bosse, P. Newman, J. Leonard, M. Soika, W. Feiten, and S. Teller. An Atlas framework for scalable mapping. In *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, pages 1899–1906, Sep 2003.

[3] R. Brooks. Visual map making for a mobile robot. In *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, volume 2, pages 824 – 829, March 1985.

[4] J. Castellanos, J. Montiel, J. Neira, and J. Tardós. The SPmap: A probabilistic framework for simultaneous localization and map building. *IEEE Trans. Robot. Automat.*, 15(5):948–953, 1999.

[5] R. G. Cowell, A. P. Dawid, S. L. Lauritzen, and D. J. Spiegelhalter. *Probabilistic Networks and Expert Systems*. Statistics for Engineering and Information Science. Springer-Verlag, 1999.

[6] T. Davis, J. Gilbert, S. Larimore, and E. Ng. A column approximate minimum degree ordering algorithm. *ACM Trans. Math. Softw.*, 30(3):353–376, 2004. ISSN 0098-3500.

[7] F. Dellaert. Square Root SAM: Simultaneous location and mapping via square root information smoothing. In *Robotics: Science and Systems (RSS)*, 2005.

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

[9] F. Dellaert, A. Kipp, and P. Krauthausen. A multifrontal QR factorization approach to distributed inference applied to multi-robot localization and mapping. In *Proc. 22nd AAAI National Conference on AI*, Pittsburgh, PA, 2005.

[10] M. Dissanayake, P. Newman, H. Durrant-Whyte, S. Clark, and M. Csorba. A solution to the simultaneous localization and map building (SLAM) problem. *IEEE Trans. Robot. Automat.*, 17(3):229–241, 2001.

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

[12] J. Folkesson, P. Jensfelt, and H. Christensen. Graphical SLAM using vision and the measurement subspace. In *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, Aug 2005.

[13] U. Frese. Treemap: An $O(\log n)$ algorithm for indoor simultaneous localization and mapping. *Autonomous Robots*, 21(2):103–122, 2006.

[14] U. Frese, P. Larsson, and T. Duckett. A multilevel relaxation algorithm for simultaneous localisation and mapping. *IEEE Trans. Robotics*, 21(2):196–207, April 2005.

[15] C. Gauss. *Theoria Motus Corporum Coelestium in Sectionibus Conicis Solem Mabientium [Theory of the Motion of the heavenly Bodies Moving about the Sun in Conic Sections]*. Perthes and Besser, Hamburg, Germany, 1809. English translation available at http://name.umdl.umich.edu/AGG8895.0001.001.

[16] W. Gentleman. Least squares computations by Givens transformations without square roots. *IMA J. of Appl. Math.*, 12:329–336, 1973.

[17] P. Gill, G. Golub, W. Murray, and M. Saunders. Methods for modifying matrix factorizations. *Mathematics and Computation*, 28(126):505–535, 1974.

[18] G. Golub and C. V. Loan. *Matrix Computations*. Johns Hopkins University Press, Baltimore, third edition, 1996.

[19] J. Guivant and E. Nebot. Optimization of the simultaneous localization and map building algorithm for real time implementation. *IEEE Trans. Robot. Automat.*, 17 (3):242–257, June 2001.

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

[21] S. Julier and J. Uhlmann. A counter example to the theory of simultaneous localization and map building. In *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, volume 4, pages 4238–4243, 2001.

[22] M. Kaess and F. Dellaert. Covariance recovery from a square root information matrix for data association. *Journal of Robotics and Autonomous Systems*, 57:1198–1210, Dec 2009. doi: 10.1016/j.robot.2009.06.008.

[23] M. Kaess, A. Ranganathan, and F. Dellaert. iSAM: Fast incremental smoothing and mapping with efficient data association. In *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, pages 1670–1677, Rome, Italy, April 2007.

[24] M. Kaess, A. Ranganathan, and F. Dellaert. iSAM: Incremental smoothing and mapping. *IEEE Trans. Robotics*, 24(6):1365–1378, Dec 2008.

[25] Z. Khan, T. Balch, and F. Dellaert. MCMC data association and sparse factorization updating for real time multitarget tracking with merged and multiple measurements. *IEEE Trans. Pattern Anal. Machine Intell.*, 28(12):1960–1972, December 2006.

[26] P. Krauthausen, F. Dellaert, and A. Kipp. Exploiting locality by nested dissection for square root smoothing and mapping. In *Robotics: Science and Systems (RSS)*, 2006.

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

[28] J. Leonard, H. Durrant-Whyte, and I. Cox. Dynamic map building for an autonomous mobile robot. *Intl. J. of Robotics Research*, 11(4):286–289, 1992.

[29] F. Ling. Givens rotation based least squares lattice related algorithms. *IEEE Trans. Signal Processing*, 39(7):1541–1551, Jul 1991.

[30] E. Olson, J. Leonard, and S. Teller. Fast iterative alignment of pose graphs with poor initial estimates. In *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, May 2006.

[31] M. Paskin. Thin junction tree filters for simultaneous localization and mapping. In *Intl. Joint Conf. on AI (IJCAI)*, 2003.

[32] L. M. Paz, P. Pinies, J. D. Tardós, and J. Neira. Large scale 6DOF SLAM with stereo-in-hand. *IEEE Transactions on Robotics*, 24(5):946–957, 2008.

[33] J. Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, 1988.

[34] A. Pothen and C. Sun. Distributed multifrontal factorization using clique trees. In *Proc. of the Fifth SIAM Conf. on Parallel Processing for Scientific Computing*, pages 34–40. Society for Industrial and Applied Mathematics, 1992.

[35] R. Smith, M. Self, and P. Cheeseman. A stochastic map for uncertain spatial relationships. In *Int. Symp on Robotics Research*, 1987.

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

[37] J. Tardós, J. Neira, P. Newman, and J. Leonard. Robust mapping and localization in indoor environments using sonar data. *Intl. J. of Robotics Research*, 21(4):311–330, 2002.

[38] R. Tarjan and M. Yannakakis. Simple linear-time algorithms to test chordality of graphs, test acyclicity of hypergraphs and selectively reduce acyclic hypergraphs. *SIAM J. Comput.*, 13(3):566–579, 1984.

[39] S. Thrun, Y. Liu, D. Koller, A. Ng, Z. Ghahramani, and H. Durrant-Whyte. Simultaneous localization and mapping with sparse extended information filters. *Intl. J. of Robotics Research*, 23(7-8):693–716, 2004.

[40] S. Thrun, W. Burgard, and D. Fox. *Probabilistic Robotics*. The MIT press, Cambridge, MA, 2005.