iSAM2: Incremental Smoothing and Mapping with Fluid Relinearization and Incremental Variable Reordering

Michael Kaess, Hordur Johannsson, Richard Roberts, Viorela Ila, John Leonard, Frank Dellaert

Abstract— **We present iSAM2, a fully incremental, graph-based version of incremental smoothing and mapping (iSAM). iSAM2 is based on a novel graphical model-based interpretation of incremental sparse matrix factorization methods, afforded by the recently introduced Bayes tree data structure. The original iSAM algorithm incrementally maintains the square root information matrix by applying matrix factorization updates. We analyze the matrix updates as simple editing operations on the Bayes tree and the conditional densities represented by its cliques. Based on that insight, we present a new method to incrementally change the variable ordering which has a large effect on efficiency. The efficiency and accuracy of the new method is based on fluid relinearization, the concept of selectively relinearizing variables as needed. This allows us to obtain a fully incremental algorithm without any need for periodic batch steps. We analyze the properties of the resulting algorithm in detail, and show on various real and simulated datasets that the iSAM2 algorithm compares favorably with other recent mapping algorithms in both quality and efficiency.**

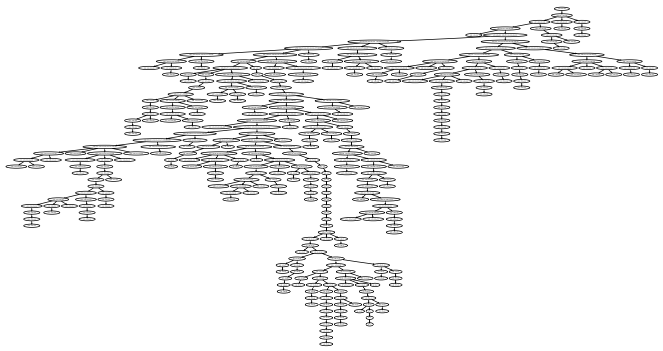
Fig. 1: iSAM2 is based on viewing incremental factorization as editing the graphical model corresponding to the posterior probability of the solution. In particular, the graphical model we use is the *Bayes tree* [17]. Above, we show a snapshot of this data structure at step 400 of the Manhattan sequence shown in Fig. 4. As a robot explores the environment, new measurements only affect parts of the tree, and only those parts are re-calculated.

I. Introduction

Knowing the spatial relationship between objects and locations is a key requirement for many tasks in mobile robotics, such as planning and manipulation. Such information is often not available prior to deploying a mobile robot, or previously acquired information is outdated because of changes in the environment. Automatically acquiring such information is therefore a key problem in mobile robotics, and has received much attention. The problem is generally referred to as simultaneous localization and mapping (SLAM).

In this work we focus on the estimation problem in SLAM. The SLAM problem consists of multiple components: processing the raw sensor data, performing data association and estimating the state of the environment. The first component is very application dependent, while the latter two are connected. We have previously addressed data association [1] in a way that is also applicable here.

For solving the estimation problem we look towards incremental, probabilistic inference algorithms. We prefer probabilistic algorithms to handle the uncertainty inherent in sensor measurements. Furthermore, for online operation in most mobile robot applications it is essential that current state estimates are available at any time, which prevents the use of offline batch processing.

A common solution to the estimation problem is smoothing and mapping or SAM [2], [3], also known as full SLAM [4] and is closely related to structure from motion [5] and bundle adjustment [6]. Smoothing estimates the complete robot trajectory and, if applicable, a map. This is in contrast to filtering which only recovers the current robot pose, but has been shown to be inconsistent [7] in the context of SLAM. To provide an efficient solution to the smoothing problem, a range of approaches have been proposed [3], [4], [8]–[16] that exploit the special sparsity structure of the problem or deploy approximations, or a combination of both. Common methods are either based on a graphical model approach or a sparse linear algebra solution.

In this work we present a novel *incremental* solution to SLAM that combines the advantages of the graphical model and sparse linear algebra perspective. Our approach, nicknamed iSAM2, is based on the insight that incremental factorization can be seen as editing the graphical model corresponding to the posterior probability of the solution. In particular, the graphical model we use is the *Bayes tree* [17], which is closely related to the junction tree. Figure 1 shows an example of the Bayes tree for a small sequence. As a robot explores the environment, new measurements only affect parts of the tree, and only those parts are re-calculated. While the Bayes tree data structure and the general incremental nonlinear optimization algorithm was previously introduced in [17], this paper focuses on the SLAM application, making the following contributions:

This work was partially funded under NSF grant 0713162 and under ONR grants N00014-06-1-0043 and N00014-10-1-0936. V. Ila has been partially supported by the Spanish MICINN under the Programa Nacional de Movilidad de Recursos Humanos de Investigación. M. Kaess, H. Johannsson, and J. Leonard are with the Computer Science and Artificial Intelligence Laboratory (CSAIL), Massachusetts Institute of Technology (MIT), Cambridge, MA 02139, USA {kaess, hordurj, jleonard}@mit.edu. R. Roberts, V. Ila, and F. Dellaert are with the School of Interactive Computing, Georgia Institute of Technology, Atlanta, GA 30332, USA {richard, vila, frank}@cc.gatech.edu

- We present the iSAM2 algorithm based on the Bayes tree data structure and tailored to SLAM applications.
- We evaluate the effect of different variable ordering strategies on efficiency.
- We evaluate the effect of the relinearization and update thresholds as a trade-off between speed and accuracy.
- We present a detailed comparison with other state-of-the-art SLAM algorithms in terms of computation and accuracy.

II. Related Work

The first smoothing approach to the SLAM problem was presented by Lu and Milios [18], where the estimation problem is formulated as a network of constraints between robot poses, and this was first implemented using matrix inversion [19]. A number of improved and numerically more stable algorithms have since been developed, based on well known *iterative* techniques such as relaxation [4], [20], [21], gradient descent [12], [22], preconditioned conjugate gradient [23], multi-level relaxation [8], and belief propagation [11]. Olson *et al*. [9] applied Gauss-Seidel to a relative formulation to avoid local minima in badly initialized problems that are difficult for other solvers. A hierarchical extension by Grisetti *et al*. [13] called TORO provides faster convergence by significantly reducing the maximum path length between two arbitrary nodes. However, the separation of translation and rotation leads to inaccurate solutions [15] that are particularly problematic for 3D applications.

More recently, some SLAM algorithms started to employ *direct* solvers based on Cholesky or QR factorization. Frese's [10] Treemap performs QR factorization within nodes of a tree, which is balanced over time. Sparsification prevents the nodes from becoming too large, which introduces approximations by duplication of variables. Dellaert *et al.* [3] presented Square Root SAM, which solves the estimation problem by Cholesky factorization of the complete, naturally sparse information matrix in every step using the Levenberg-Marquardt algorithm. Ni *et al.* [24] presented Tectonic SAM that divides the optimization problem into submaps while still recovering the exact solution. Grisetti *et al.* [15] recently presented a hierarchical pose graph formulation using Cholesky factorization that represents the problem at different levels of detail, which allows focusing on the affected area at the most detailed level, while still propagating any global effects at the coarser levels. Konolige et al. [16] recently also presented a Cholesky based algorithm called Sparse Pose Adjustment (SPA), which improves Square Root SAM [3] by a continuable Levenberg-Marquardt algorithm and a fast setup of the information matrix.

In [14] we introduced a more efficient version of Square Root SAM called iSAM that incrementally updates a QR matrix factorization using Givens rotations. iSAM avoids the cost of batch factorization in every step, while still obtaining the exact solution. To keep the solution efficient, periodic batch steps are necessary for variable reordering, which detracts from the online nature of the algorithm. In contrast, the new iSAM2 algorithm presented here incrementally reorders variables. This was made possible by combining insights from both the fields of sparse linear algebra and graphical models. iSAM2 also introduces incremental or fluid relinearization, that selectively relinearizes variables at every step as needed, completely eliminating the need for periodic batch steps. iSAM2 is implemented using the Bayes tree data structure [17], with incomplete Cholesky factorization performed within each node of the tree.

III. Problem Statement

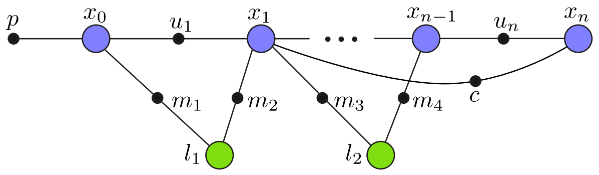

Fig. 2: Factor graph [25] formulation of the SLAM problem, where variable nodes are shown as large circles, and factor nodes (measurements) with small solid circles. This example illustrates both a loop closing constraint c and landmark measurements m, in addition to odometry measurements u and a prior p. Note that arbitrary cost functions can be represented, also including more than two factors.

We use a *factor graph* [25] to represent the SLAM problem in terms of graphical models. 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 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 of a SLAM factor graph is shown in Fig. 2, where the landmark measurements m, loop closing constraint c and odometry measurements u are examples of factors. Note that this formulation supports general probability distributions or cost functions of any number of variables allowing the inclusion of calibration parameters or spatial separators as used in T-SAM [24] and cooperative mapping [26].

When assuming Gaussian measurement models

$$f_i(\Theta_i) \propto \exp\left(-\frac{1}{2}\left\|h_i(\Theta_i) - z_i\right\|^2_{\Sigma_i}\right) \tag{2}$$

as is standard in the SLAM literature, the factored objective function we want to minimize (1) corresponds to the nonlinear least-squares criterion

$$\arg\min_{\Theta}\left(-\log f(\Theta)\right) = \arg\min_{\Theta} \frac{1}{2}\sum_i \left\|h_i(\Theta_i) - z_i\right\|^2_{\Sigma_i} \tag{3}$$

where $h_i(\Theta_i)$ is a measurement function and z_i a measurement, and $\|e\|^2_\Sigma \overset{\Delta}{=} e^T\Sigma^{-1}e$ is defined as the squared Mahalanobis distance with covariance matrix Σ.

In practice one always considers a linearized version of problem (3). For nonlinear measurement functions h_i (2), nonlinear optimization methods such as Gauss-Newton

Alg. 1 General structure of the smoothing solution to SLAM with a direct equation solver (Cholesky, QR). Steps 3-6 can optionally be iterated and/or modified to implement the Levenberg-Marquardt algorithm.

```
Repeat for new measurements in each step:
  1) Add new measurements.
  2) Add and initialize new variables.
  3) Linearize at current estimate $\Theta$.
  4) Factorize with QR or Cholesky.
  5) Solve by backsubstitution to obtain $\mathbf{\Delta}$.
  6) Obtain new estimate $\Theta' = \Theta \oplus \mathbf{\Delta}$.
```

iterations or the Levenberg-Marquardt algorithm solve a succession of linear approximations to (3) in order to approach the minimum. At each iteration of the nonlinear solver, we linearize around a linearization point Θ to get a new, *linear* least-squares problem in $\mathbf{\Delta}$

$$\arg\min_{\mathbf{\Delta}} \left(-\log f(\mathbf{\Delta})\right) = \arg\min_{\mathbf{\Delta}} \|A\mathbf{\Delta} - \mathbf{b}\|^2 \qquad (4)$$

where $A \in \mathbb{R}^{m \times n}$ is the measurement Jacobian consisting of m measurement rows and $\mathbf{\Delta}$ is an n-dimensional tangent vector [27]. This system can be solved based on the normal equations $A^T A\mathbf{\Delta} = A^T\mathbf{b}$ by Cholesky factorization of $A^T A = R^T R$, followed by forward and backsubstitution on $R^T\mathbf{y} = A^T\mathbf{b}$ and $R\mathbf{\Delta} = \mathbf{y}$ to first recover $\mathbf{y}$, then the actual solution $\mathbf{\Delta}$. Alternatively QR factorization yields $R\mathbf{\Delta} = \mathbf{d}$ which can directly be solved by backsubstitution (note that Q is not explicitly formed; instead $\mathbf{b}$ is modified during factorization to obtain $\mathbf{d}$, see [14]). Alg. 1 shows a summary of the necessary steps to solve the smoothing formulation of the SLAM problem with direct methods.

IV. The iSAM2 Algorithm

Our previous iSAM algorithm [14] reduces the complexity of the batch solution to smoothing by incrementally updating a matrix factorization. A batch solution performs unnecessary calculations, because it solves the complete problem at every step, including all previous measurements. New measurements often have only a local effect, leaving remote parts of the map untouched. iSAM exploits that fact by incrementally updating the square root information matrix R with new measurements. The updates often only affect a small part of the matrix, and are therefore much cheaper than batch factorization. However, as new variables are appended, the variable ordering is far from optimal, and fill-in soon accumulates. iSAM therefore performs periodic batch steps, in which the variables are reordered, and the complete matrix is factorized again. Relinearization is also performed during the periodic batch steps.

We have recently presented the Bayes tree data structure [17], which maps the sparse linear algebra perspective onto a graphical model view. Combining insights obtained from both perspectives of the same problem led us to the development of a fully incremental nonlinear estimation algorithm [17]. Here, we adapt this algorithm to better fit the SLAM problem. The most important change is the combination of the tree updates resulting from the linear update step and the fluid relinearization.

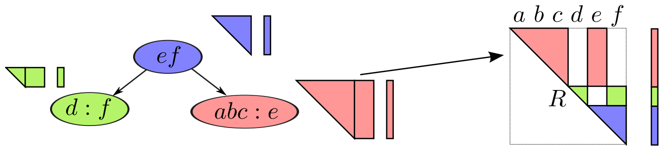

Fig. 3: Connection between Bayes tree (left) and square root information factor (right): Each clique of the tree contains the conditional probability densities of its frontal variables. In square root information form these are the same entries as in the corresponding matrix factor R.

Alg. 2 One step of the iSAM2 algorithm, following the general structure of a smoothing solution given in Alg. 1.

```
In/out: Bayes tree $\mathcal{T}$, nonlinear factors $\mathcal{F}$, linearization point $\Theta$, update $\mathbf{\Delta}$
In: new nonlinear factors $\mathcal{F}'$, new variables $\Theta'$
Initialization: $\mathcal{T} = \emptyset$, $\Theta = \emptyset$, $\mathcal{F} = \emptyset$
  1) Add any new factors $\mathcal{F} := \mathcal{F} \cup \mathcal{F}'$.
  2) Initialize any new variables $\Theta'$ and add $\Theta := \Theta \cup \Theta'$.
  3) Fluid relinearization with Alg. 5 yields affected variables $\mathcal{J}$, see
     Section IV-C.
  4) Redo top of Bayes tree with Alg. 3.
  5) Solve for delta $\mathbf{\Delta}$ with Alg. 4, see Section IV-B.
  6) Current estimate given by $\Theta \oplus \mathbf{\Delta}$.
```

The Bayes tree encodes the clique structure of the chordal graph that is generated by variable elimination [17]. The nodes of the Bayes tree encode conditional probability distributions corresponding to the variables that are eliminated in the clique. These conditionals directly correspond to rows in a square root information matrix, as illustrated in Fig. 3. The important difference to the matrix representation is that it becomes obvious how to reorder variables in this Bayes tree structure. For a sparse matrix data structure, in contrast, this task is very difficult. An example of an actual Bayes tree is shown in Fig. 1.

The iSAM2 algorithm is summarized in Alg. 2. Not all details of the algorithm fit into the space of this paper; instead we present the most important components and refer the reader to our Bayes tree work [17]. Note that the iSAM2 algorithm is different from our original work: iSAM2 is more efficient by combining the modifications of the Bayes tree arising from both linear updates and fluid relinearization in Alg. 3. In the following subsections we provide a summary of important components of iSAM2, as well as a quantitative analysis of their behavior in isolation.

Alg. 3 Updating the Bayes tree by recalculating all affected cliques. Note that the algorithm differs from [17]: Update and relinearization are combined for efficiency.

```
In: Bayes tree $\mathcal{T}$, nonlinear factors $\mathcal{F}$, affected variables $\mathcal{J}$
Out: modified Bayes tree $\mathcal{T}$'
  1) Remove top of Bayes tree, convert to a factor graph:
       a) For each affected variable in $\mathcal{J}$ remove the corresponding
          clique and all parents up to the root.
       b) Store orphaned sub-trees $\mathcal{T}_{orph}$ of removed cliques.
  2) Relinearize all factors required to recreate top.
  3) Add cached linear factors from orphans.
  4) Re-order variables, see Section IV-A.
  5) Eliminate the factor graph and create Bayes tree (for details see
     Algs. 1 and 2 in [17]).
  6) Insert the orphans $\mathcal{T}_{orph}$ back into the new Bayes tree.
```

A. *Incremental Variable Ordering*

Choosing a good variable ordering is essential for the efficiency of the sparse matrix solution, and this also holds for the Bayes tree approach. In [17] we have described an incremental variable ordering strategy that leads to better performance than a naive incremental variable ordering, and we perform a more detailed evaluation of its performance here. An optimal ordering minimizes the fill-in, which refers to additional entries in the square root information matrix that are created during the elimination process. In the Bayes tree, fill-in translates to larger clique sizes, and consequently slower computations. While finding the variable ordering that leads to the minimal fill-in is NP-hard [28] for general problems, one typically uses heuristics such as the column approximate minimum degree (COLAMD) algorithm by Davis *et al.* [29], which provide close to optimal orderings for many batch problems.

While performing incremental inference in the Bayes tree, variables can be reordered at every incremental update, eliminating the need for periodic batch reordering. This was not understood in [14], because this is only obvious within the graphical model framework, but not for matrices. The affected part of the Bayes tree for which variables have to be reordered is typically small, as new measurements usually only affect a small subset of the overall state space represented by the variables of the optimization problem.

To allow for faster updates in subsequent steps, our proposed incremental variable ordering strategy forces the most recently accessed variables to the end of the ordering. The naive incremental approach applies COLAMD locally to the subset of the tree that is being recalculated. In the SLAM setting we can expect that a new set of measurements connects to some of the recently observed variables, be it landmarks that are still in range, or the previous robot pose connected by an odometry measurement. The expected cost of incorporating the new measurements, i.e. the size of the affected sub-tree in the update, will be lower if these variables are closer to the root. Applying COLAMD locally does not take this consideration into account, but only minimizes fill-in for the current step. We therefore force the most recently accessed variables to the end of the ordering using the constrained COLAMD (CCOLAMD) ordering [29].

In Fig. 4 we compare our constrained solution to the naive way of applying COLAMD. The top row of Fig. 4 shows a color coded trajectory of the Manhattan simulated dataset [9]. The robot starts in the center, traverses the loop counter clockwise, and finally ends at the bottom left. The number of affected variables significantly drops from the naive approach (left) to the constrained approach (right), as red parts of the trajectory (high cost) are replaced by green (low cost). Particularly for the left part of the trajectory the number of affected variables is much smaller than before, which one would expect from a good ordering, as no large loops are being closed in that area. The remaining red segments coincide with the closing of the large loop in the right

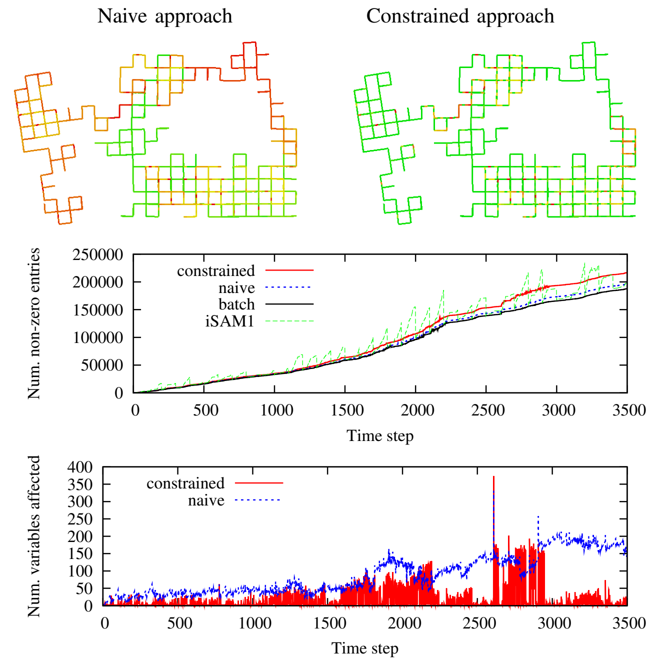

Fig. 4: Comparison of variable ordering strategies using the Manhattan world simulated environment [9]. By color coding, the top row shows the number of variables that are updated for every step along the trajectory. Green corresponds to a low number of variables, red to a high number. The naive approach of applying COLAMD to the affected variables in each step shows a high overall cost. Forcing the most recently accessed variables to the end of the ordering using constrained COLAMD [29] yields a significant improvement in efficiency. The center plot shows the fill-in over time for both strategies as well as the batch ordering and iSAM1. The bottom plot clearly shows the improvement in efficiency achieved by the constrained ordering by comparing the number of affected variables in each step.

part of the trajectory. The second row of Fig. 4 shows that the constrained ordering causes a small increase in fill-in compared to the naive approach, which itself is close to the fill-in caused by the batch ordering. The bottom figure shows that the number of affected variables steadily increases for the naive approach, but often remains low for the constrained version, though the spikes indicate that a better incremental ordering strategy can likely be found for this problem.

B. *Partial State Updates*

Recovering a nearly exact solution in every step does not require solving for all variables. New measurements often have only a local effect, leaving spatially remote parts of the estimate unchanged. We can therefore significantly reduce the computational cost by only solving for variables that actually change. Full backsubstitution starts at the root and continues to all leaves, obtaining a delta vector $\boldsymbol{\Delta}$ that is used to update the linearization point Θ. Updates to the Bayes tree from new factors and from relinearization only affect the top of the tree, however, changes in variable estimates occurring here can propagate further down to all sub-trees.

iSAM2 starts by solving for all variables contained in the modified top of the tree. As shown in Alg. 4, we then continue processing all sub-trees, but stop when encountering a clique that does not refer to any variable for which $\boldsymbol{\Delta}$ changed by more than a small threshold α. Note that the

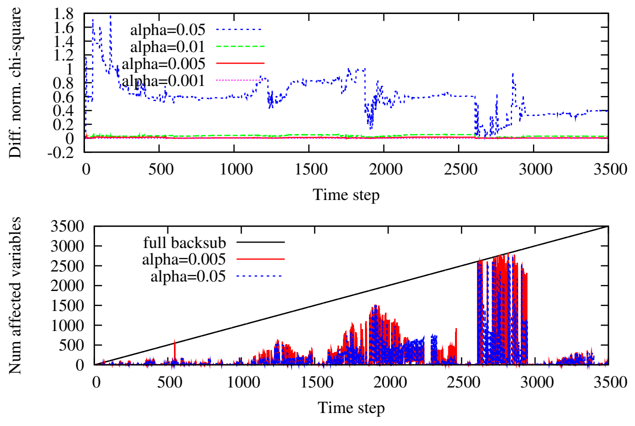

Fig. 5: How the backsubstitution threshold α affects accuracy (top) and computational cost (bottom) for the Manhattan dataset. For readability of the top figure, the normalized χ^2 value of the least-squares solution was subtracted. A small threshold such as 0.005 yields a significant increase in speed, while the accuracy is nearly unaffected.

Alg. 4 Partial state update: Solving the Bayes tree in the nonlinear case returns an update $\mathbf{\Delta}$ to the current linearization point Θ.

```
In: Bayes tree $\mathcal{T}$
Out: update $\mathbf{\Delta}$
Starting from the root clique $C_r = F_r$:
  1) For current clique $C_k = F_k : S_k$
     compute update $\mathbf{\Delta}_k$ of frontal variables $F_k$ from the local conditional density $P(F_k|S_k)$.
  2) For all variables $\mathbf{\Delta}_{k_j}$ in $\mathbf{\Delta}_k$ that change by more than threshold $\alpha$:
     recursively process each descendant containing such a variable.
```

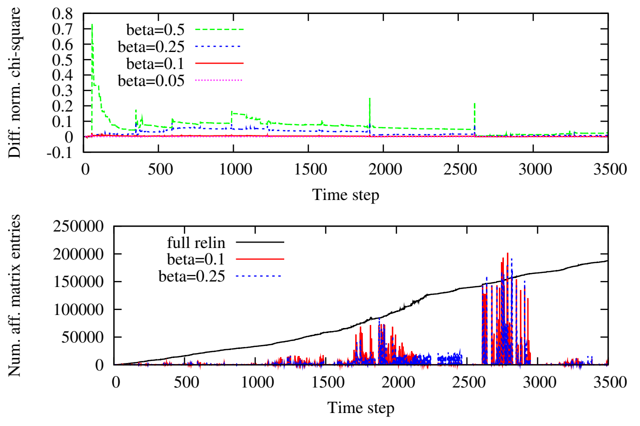

Fig. 6: How the relinearization threshold β affects accuracy (top) and computational cost (bottom) for the Manhattan dataset. For readability of the top figure, the normalized χ^2 value of the least-squares solution was subtracted. A threshold of 0.1 has no notable effect on the accuracy, while the cost savings are significant as can be seen in the number of affected nonzero matrix entries. Note that the spikes extend beyond the curve for full relinearization, because there is a small increase in fill-in over the batch variable ordering (see Fig. 4).

Alg. 5 Fluid relinearization: The linearization points of select variables are updated based on the current delta $\mathbf{\Delta}$.

```
In: linearization point $\Theta$, delta $\mathbf{\Delta}$
Out: updated linearization point $\Theta$, marked cliques $M$
  1) Mark variables in $\mathbf{\Delta}$ above threshold $\beta$: $J = \{\mathbf{\Delta}_j \in \mathbf{\Delta} | \mathbf{\Delta}_j \geq \beta\}$.
  2) Update linearization point for marked variables: $\Theta_J := \Theta_J \oplus \mathbf{\Delta}_J$.
  3) Mark all cliques $M$ that involve marked variables $\Theta_J$ and all their ancestors.
```

threshold refers to a change in the delta vector $\mathbf{\Delta}$, not the absolute value of the recovered delta $\mathbf{\Delta}$ itself. The absolute values of the entries in $\mathbf{\Delta}$ can be quite large, because, as described below, the linearization point is only updated when a larger threshold is reached. And to be exact, the different units of variables have to be taken into account, but one simple solution is to take the minimum over all thresholds. For variables that are not reached, the previous estimate $\mathbf{\Delta}$ is kept. An nearly exact solution is obtained with significant savings in computation time, as can be seen from Fig. 5.

C. *Fluid Relinearization*

The idea behind just-in-time or fluid relinearization is to 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 for direct equation solvers. For a variable that is chosen to be relinearized, all relevant information has to be removed from the Bayes tree and replaced by relinearizing the corresponding original nonlinear factors. For cliques that are re-eliminated we also have to take into account any marginal factors that are passed up from their sub-trees. We cache those marginal factors during elimination, so that the process can be restarted from the middle of the tree, rather than having to re-eliminate the complete system.

Our fluid relinearization algorithm is shown in Alg. 5. Note that because we combine the relinearization and update steps for efficiency, the actual changes in the Bayes tree are performed later. The decision is based on the deviation of the current estimate from the linearization point being larger than a threshold β. For simplicity we again use the same threshold for all variables, though that could be refined. A nearly exact solution is provided for a threshold of 0.1, while the computational cost is significantly reduced, as can be seen from Fig. 6.

D. *Complexity*

In this section we provide some general complexity bounds for iSAM2. The number of iterations needed to converge is typically fairly small, in particular because of the quadratic convergence properties of Gauss-Newton iterations near the minimum. We assume here that the initialization of variables is close enough to the global minimum to allow convergence - that is a general requirement of any direct solver method. For exploration tasks with a constant number of constraints per pose, the complexity is $O(1)$ as only a constant number of variables at the top of the tree are affected and have to be re-eliminated, and only a constant number of variables are solved for. In the case of loop closures the situation becomes more difficult, and the most general bound is that for full factorization, $O(n^3)$, where n is the number of variables (poses and landmarks if present). Under certain assumptions that hold for many SLAM problems, batch matrix factorization and backsubstitution can be performed in $O(n^{1.5})$ [30]. It is important to note that this bound does not depend on the number of loop closings. Empirically, complexity is usually much lower than these upper bounds because most of the time only a small portion of the matrix has to be refactorized in each step, as we show below.

V. Comparison to Other Methods

We compare iSAM2 to other state-of-the-art SLAM algorithms, in particular the iSAM1 algorithm [14], HOG-Man [15] and SPA [16]. We use a wide variety of simulated and real-world datasets shown in Fig. 7 that feature different sizes and constraint densities, both pose-only and including landmarks. All timing results are obtained on a laptop with Intel Core 2 Duo 2.4 GHz processor. For iSAM1 we use version 1.5 of the open source implementation available at `http://people.csail.mit.edu/kaess/isam`. For HOG-Man, we use svn revision 14 available at `http://openslam.org/`. For SPA, we use svn revision 32333 of ROS at `http://www.ros.org/`. For iSAM2 we use a research C++ implementation running single-threaded, using the CCOLAMD algorithm by Davis *et al.* [29], with parameters $\alpha = 0.001$ and $\beta = 0.1$ and relinearization every 10 steps. Source code for iSAM2 is available as part of the gtsam library at `https://collab.cc.gatech.edu/borg/gtsam/`.

Comparing the computational cost of different algorithms is not a simple task. Tight complexity bounds for SLAM algorithms are often not available. Even if complexity bounds are available, they are not necessarily suitable for comparison because the involved constants can make a large difference in practical applications. On the other hand, comparison of the speed of implementations of the algorithms depends on the implementation itself and any potential inefficiencies or wrong choice of data structures. We will therefore discuss not only the timing results obtained from the different implementations, but also compare some measure of the underlying cost, such as how many entries of the sparse matrix have to be recalculated. That again on its own is also not a perfect measure, as recalculating only parts of a matrix might occur some overhead that cannot be avoided.

A. Timing

We compare execution speed of implementations of the various algorithms on all datasets in Fig. 8, with detailed results in Table I. The results show that a batch Cholesky solution (SPA, SAM) quickly gets expensive, emphasizing the need for incremental solutions.

iSAM1 performs very well on sparse datasets, such as Manhattan and Killian, while performance degrades on datasets with denser constraints (number of constraints at least 5 times the number of poses), such as W10000 and Intel, because of local fill-in between the periodic batch reordering steps (see Fig. 4 center). An interesting question to ask is how many constraints are actually needed to obtain an accurate reconstruction, though this will be a function of the quality of the measurements. Note that the spikes in the iteration time plots are caused by the periodic variable reordering every 100 steps, which equals to a batch Cholesky factorization (SPA) with some overhead for the incremental data structures.

The performance of HOG-Man is between SPA and iSAM1 and 2 for most of the datasets, but performs better on W10000 than any other algorithm. Performance is generally better on larger datasets, where the advantages of hierarchical operations dominate their overhead.

iSAM2 consistently performs better than SPA, and similar to iSAM1. While iSAM2 saves computation over iSAM1 by only performing partial backsubstitution, the fluid relinearization adds complexity. Relinearization typically affects many more variables than a linear update (compare Figs. 4 and 6), resulting in larger parts of the Bayes tree having to be recalculated. Interesting is the fact that the spikes in iSAM2 timing follow SPA, but are higher by almost an order of magnitude, which becomes evident in the per iteration time plot. That difference can partially be explained by the fact that SPA uses a well optimized library for Cholesky factorization (CHOLMOD), while for the algorithms underlying iSAM2 no such library is available yet and we are using our own research implementation.

B. Number of Affected Entries

We also provide a computation cost measure that is more independent of specific implementations, based on the number of variables affected, and the number of entries of the sparse square root information matrix that are being recalculated in each step. The bottom plots in Figs. 5 and 6 show the number of affected variables in backsubstitution and the number of affected non-zero entries during matrix factorization. The red curve shows the cost of iSAM2 for thresholds that achieve an almost exact solution. When compared to the batch solution shown in black, the data clearly shows significant savings in computation of iSAM2 over Square Root SAM and SPA.

In iSAM2 the fill-in of the corresponding square root information factor remains close to that of the batch solution as shown in Fig. 4. The same figure also shows that for iSAM1 the fill-in increases significantly between the periodic batch steps, because variables are only reordered every 100 steps. This local fill-in explains the higher computational cost on datasets with denser constraints, such as W10000. iSAM2 shows no significant local variations of fill-in owing to the incremental variable ordering.

C. Accuracy

We now focus on the accuracy of the solution of each SLAM algorithm. There are a variety of different ways to evaluate accuracy. We choose the normalized χ^2 measure that quantifies the quality of a least-squares fit. Normalized χ^2 is defined as $\frac{1}{m-n}\sum_i \|h_i(\Theta_i) - z_i\|^2_{\Lambda_i}$, where the numerator is the weighted sum of squared errors of (3), m is the number of measurements and n the number of degrees of freedom. Normalized χ^2 measures how well the constraints are satisfied, approaching 1 for a large number of measurements sampled from a normal distribution.

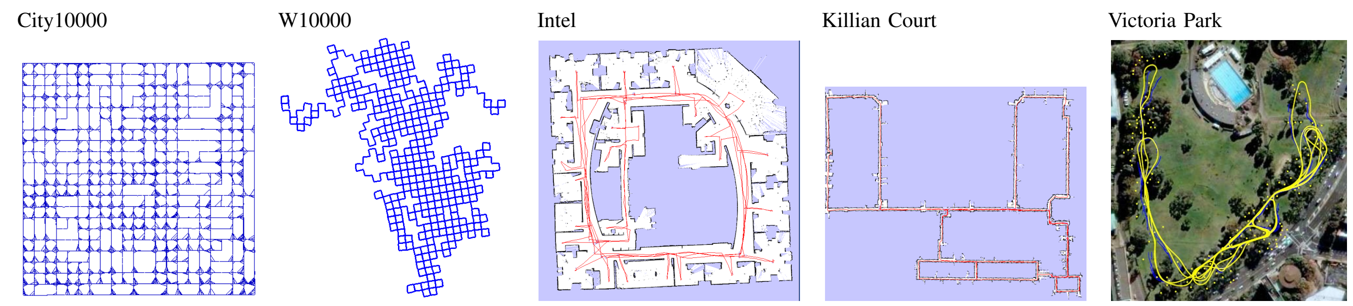

Fig. 7: Datasets used for the comparison, including simulated data (City10000, W10000), indoor laser range data (Killian, Intel) and outdoor laser range data (Victoria Park). See Fig. 4 for the Manhattan sequence.

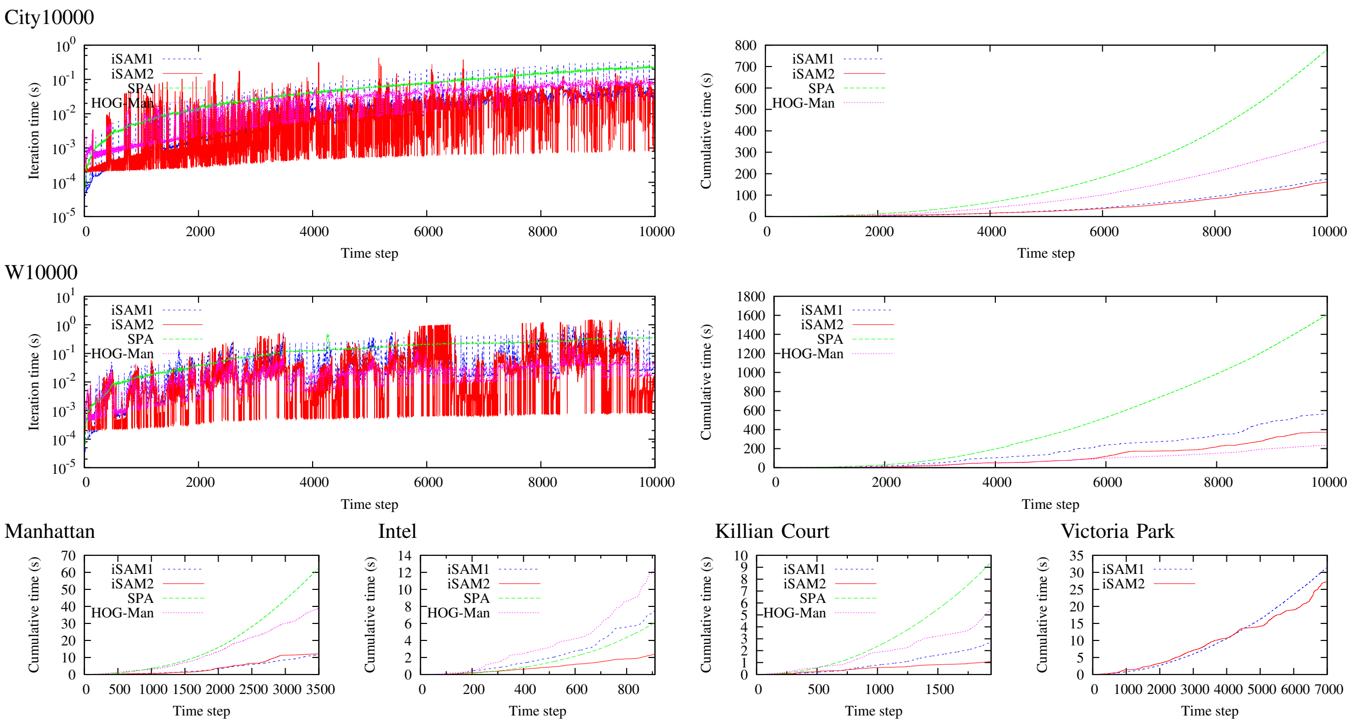

Fig. 8: Timing comparison between the different algorithms for all datasets, see Fig. 7. The left column shows per iteration time and the right column cumulative time. The bottom row shows cumulative time for the remaining datasets.

TABLE I: Runtime comparison for the different approaches (P: number of poses, M: number of measurements, L: number of landmarks). Listed are the average time per step together with standard deviation and maximum in milliseconds, as well as the overall time in seconds (fastest result shown in red).

			Algorithm	iSAM2		iSAM1		HOG-Man		SPA	
Dataset	P	M	L	avg/std/max [ms]	time [s]	avg/std/max [ms]	time [s]	avg/std/max [ms]	time [s]	avg/std/max [ms]	[s]
City10000	10000	20687	-	16.1 / 22.0 / 427	161	17.5 / 22.2 / 349	175	35.3 / 27.4 / 156	353	78.1 / 67.4 / 245	781
W10000	10000	64311	-	37.2 / 111 / 1550	372	56.5 / 86.6 / 1010	565	23.6 / 20.4 / 196	236	161 / 108 / 483	1610
Manhattan	3500	5598	-	3.46 / 12.8 / 217	12.1	3.32 / 4.69 / 63.7	11.6	11.2 / 9.85 / 49.0	39.0	17.9 / 12.7 / 52.2	62.5
Intel	910	4453	-	2.59 / 2.81 / 13.8	2.36	8.25 / 11.0 / 67.2	7.50	13.8 / 17.8 / 111	12.5	6.70 / 5.16 / 19.4	6.09
Killian Court	1941	2190	-	0.56 / 1.25 / 20.6	1.08	1.38 / 2.08 / 30.2	2.67	2.88 / 3.45 / 17.7	5.59	4.86 / 3.01 / 11.1	9.43
Victoria Park	6969	10608	151	3.91 / 13.4 / 527	27.2	4.49 / 6.57 / 112	31.3	N/A	N/A	N/A	N/A

The results in Fig. 9 show that the iSAM2 solution is very close to the ground truth given by batch (SAM or SPA). Small deviations are caused by relinearizing only every 10 steps, which is a tradeoff with computational speed. iSAM1 shows a larger spike in error that is caused by relinearization only being done only every 100 steps. HOG-Man is an approximate algorithm exhibiting consistently larger errors, even though visual inspection of the resulting map showed only minor distortions. Accuracy improves for more dense datasets, such as W10000, but is still not as good as iSAM2.

D. *Beyond 2D*

While we focused our evaluation in this paper on 2D data, iSAM also correctly deals with 3D data, with the first visual SLAM application of SAM shown in [3]. Over-parametrization and singularities are handled using exponential maps of Lie groups [27]. Fig. 10 shows a 3D dataset before and after incremental optimization. Note that a large range of orientations is traversed, as the simulated robot is always perpendicular to the surface of the sphere. For this example, rotations are parametrized as Euler angles. While

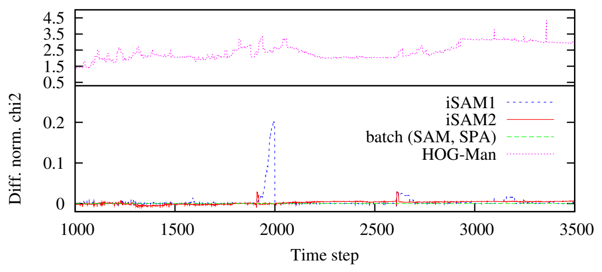

Fig. 9: Step-wise quality comparison of the different algorithms for the Manhattan world. For improved readability, the difference of normalized χ^2 from the least squares solution is shown.

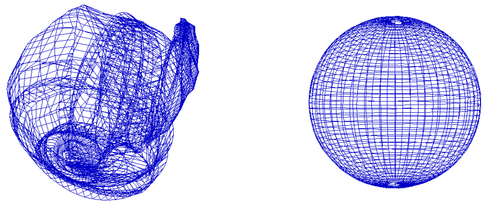

Fig. 10: Simulated 3D dataset (sphere2500, included in iSAM1 distribution): iSAM2 correctly deals with 6DOF using exponential maps of Lie groups. (left) The noisy data for a simulated robot driving along the surface of a sphere. (right) The sphere reconstructed by iSAM2. Note that the simulated poses are always perpendicular to the surface of the sphere.

incremental updates in Euler parametrization are free of singularities, absolute poses near singularities have explicitly been avoided in this dataset.

VI. Conclusion

We have presented iSAM2, a novel graph-based algorithm for efficient online mapping, based on the Bayes tree representation in [17]. We described a modification to make the algorithm better suited to SLAM. We performed a systematic evaluation of iSAM2 and a comparison with three other state-of-the-art SLAM algorithms. We expect our novel graph-based algorithm to 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 [1]. The graph-based structure is also suitable for exploiting parallelization that is becoming available in newer processors.

VII. Acknowledgments

We thank E. Olson for the Manhattan dataset, E. Nebot and H. Durrant-Whyte for the Victoria Park dataset, D. Haehnel for the Intel dataset and G. Grisetti for the W10000 dataset.

References

[1] M. Kaess and F. Dellaert, "Covariance recovery from a square root information matrix for data association," *Journal of Robotics and Autonomous Systems*, vol. 57, pp. 1198–1210, Dec 2009.

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

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

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

[5] R. Hartley and A. Zisserman, *Multiple View Geometry in Computer Vision*. Cambridge University Press, 2000.

[6] B. Triggs, P. McLauchlan, R. Hartley, and A. Fitzgibbon, "Bundle adjustment – a modern synthesis," in *Vision Algorithms: Theory and Practice* (W. Triggs, A. Zisserman, and R. Szeliski, eds.), LNCS, pp. 298–375, Springer Verlag, Sep 1999.

[7] 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)*, vol. 4, pp. 4238–4243, 2001.

[8] U. Frese, P. Larsson, and T. Duckett, "A multilevel relaxation algorithm for simultaneous localisation and mapping," *IEEE Trans. Robotics*, vol. 21, pp. 196–207, Apr 2005.

[9] 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.

[10] U. Frese, "Treemap: An $O(\log n)$ algorithm for indoor simultaneous localization and mapping," *Autonomous Robots*, vol. 21, no. 2, pp. 103–122, 2006.

[11] A. Ranganathan, M. Kaess, and F. Dellaert, "Loopy SAM," in *Intl. Joint Conf. on AI (IJCAI)*, (Hyderabad, India), pp. 2191–2196, 2007.

[12] J. Folkesson and H. Christensen, "Closing the loop with Graphical SLAM," *IEEE Trans. Robotics*, vol. 23, pp. 731–741, Aug 2007.

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

[14] M. Kaess, A. Ranganathan, and F. Dellaert, "iSAM: Incremental smoothing and mapping," *IEEE Trans. Robotics*, vol. 24, pp. 1365–1378, Dec 2008.

[15] G. Grisetti, R. Kuemmerle, C. Stachniss, U. Frese, and C. Hertzberg, "Hierarchical optimization on manifolds for online 2D and 3D mapping," in *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, (Anchorage, Alaska), May 2010.

[16] K. Konolige, G. Grisetti, R. Kuemmerle, W. Burgard, L. Benson, and R. Vincent, "Sparse pose adjustment for 2D mapping," in *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, Oct 2010.

[17] M. Kaess, V. Ila, R. Roberts, and F. Dellaert, "The Bayes tree: An algorithmic foundation for probabilistic robot mapping," in *Intl. Workshop on the Algorithmic Foundations of Robotics*, (Singapore), pp. 157–173, Dec 2010.

[18] F. Lu and E. Milios, "Globally consistent range scan alignment for environment mapping," *Autonomous Robots*, pp. 333–349, Apr 1997.

[19] J.-S. Gutmann and B. Nebel, "Navigation mobiler roboter mit laserscans," in *Autonome Mobile Systeme*, (Berlin), Springer Verlag, 1997.

[20] T. Duckett, S. Marsland, and J. Shapiro, "Fast, on-line learning of globally consistent maps," *Autonomous Robots*, vol. 12, no. 3, pp. 287–300, 2002.

[21] M. Bosse, P. Newman, J. Leonard, and S. Teller, "Simultaneous localization and map building in large-scale cyclic environments using the Atlas framework," *Intl. J. of Robotics Research*, vol. 23, pp. 1113–1139, Dec 2004.

[22] J. Folkesson and H. Christensen, "Graphical SLAM - a self-correcting map," in *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, vol. 1, pp. 383–390, 2004.

[23] K. Konolige, "Large-scale map-making," in *Proc. 21^{th} AAAI National Conference on AI*, (San Jose, CA), 2004.

[24] K. Ni, D. Steedly, and F. Dellaert, "Tectonic SAM: Exact; out-of-core; submap-based SLAM," in *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, (Rome; Italy), Apr 2007.

[25] F. Kschischang, B. Frey, and H.-A. Loeliger, "Factor graphs and the sum-product algorithm," *IEEE Trans. Inf. Theory*, vol. 47, Feb 2001.

[26] B. Kim, M. Kaess, L. Fletcher, J. Leonard, A. Bachrach, N. Roy, and S. Teller, "Multiple relative pose graphs for robust cooperative mapping," in *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, (Anchorage, Alaska), pp. 3185–3192, May 2010.

[27] B. Hall, *Lie Groups, Lie Algebras, and Representations: An Elementary Introduction*. Springer, 2000.

[28] S. Arnborg, D. Corneil, and A. Proskurowski, "Complexity of finding embeddings in a k-tree," *SIAM J. on Algebraic and Discrete Methods*, vol. 8, pp. 277–284, Apr 1987.

[29] T. Davis, J. Gilbert, S. Larimore, and E. Ng, "A column approximate minimum degree ordering algorithm," *ACM Trans. Math. Softw.*, vol. 30, no. 3, pp. 353–376, 2004.

[30] 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.