Noname manuscript No.
(will be inserted by the editor)

Efficient Cholesky Factor Recovery for Column Reordering in Simultaneous Localisation and Mapping

S. Touchette · W. Gueaieb · E. Lanteigne

the date of receipt and acceptance should be inserted later

Abstract Simultaneous Localisation And Mapping problems are inherently dynamic and the structure of the graph representing them changes significantly over time. To obtain the least square solution of such systems efficiently, it is desired to maintain a good column ordering such that fill-ins are reduced. This comes at a cost since general ordering changes require the complete re-computation of the Cholesky factor. While some methods have obtained good results with reordering at loop closing, the changes are not guaranteed to be limited to the scope of the loop, leading to suboptimal performance. In this article, it is shown that the Cholesky factorisation of an updated matrix can be efficiently recovered from the previous factorisation if the permutations are localised. This is experimentally demonstrated on 2D SLAM datasets. A method is then provided to identify when such recovery is advantageous over the complete re-computation of the Cholesky factor. Furthermore, a hybrid algorithm combining factorisation recovery and re-computation of the Cholesky factor is proposed for dynamically evolving problems and tested on SLAM datasets. Steps where reordering occurs can be executed up to 67% faster with the proposed method.

Keywords Cholesky Factorisation · Factor Modification · Incremental Reordering · SLAM · Smoothing and Mapping · Sparse Matrices

1 Introduction

Unmanned Aerial Systems (UAS) have, over the last forty years, escalated from a few highly experimental units to wide spread adoption across industries, fields and

Sébastien Touchette
School of Electrical Engineering and Computer Science, University of Ottawa
E-mail: sebastien.touchette@uOttawa.ca

Wail Gueaieb
School of Electrical Engineering and Computer Science, University of Ottawa
E-mail: wgueaieb@eecs.uOttawa.ca

Eric Lanteigne
Department of Mechanical Engineering, University of Ottawa
E-mail: Eric.Lanteigne@uOttawa.ca

borders [1, 2, 31, 51]. Ground vehicles have been around even longer and, with the introduction of robotic vacuums and interactive toys, are already part of our daily lives. Regardless of their type or purpose, most unmanned vehicles are confronted by the same problems. In the literature, Simultaneous Localisation And Mapping (SLAM) is the effective and concurrent navigation and modelling of an environment by an autonomous system. It has been extensively studied and many algorithms are available to solve it but this issue remains an active research area as performance requirements increase [5, 18].

Inspired by graph-based solutions for SLAM [19], as well as the work done in the field of Structure from Motion, smoothing solutions to the SLAM problem have gained significant popularity in the last two decades. Exploiting the link between operations on graphs and their associated matrices, Dellaert and Kaess [17] presented $\sqrt{\text{SAM}}$ and demonstrated that Smoothing and Mapping (SAM) can be a viable alternative to current SLAM implementations for on-line applications. Kummerle *et al.* [36] proposed g^2o, a highly general and optimised solver for graph based problems that use an efficient front end to obtain better performance than $\sqrt{SAM}$ and its incremental version, iSAM [30]. In an effort to reduce memory footprint and computation time, Dellaert *et al.* used an approximate solution and iterative methods [16]. Konolige *et al.* [34] have proposed SPA, a method similar to iSAM but designed to take advantage of the high landmark to pose ratio. Smoothing and mapping algorithms, most based on iSAM, have also been proposed in the field of cooperative mapping [8–10, 26, 33] and submaps [39, 40]. Bridging smoothing and filtering, mixed method have been proposed by Williams *et al.* [50] and Indelman *et al.* [27] for high rate information fusion.

Dellaert and Kaess [17] have illustrated the effects of column ordering, which is of prime importance to least square filtering problems. This is especially true for SLAM since the size becomes increasingly large with time and execution often requires real-time performance. Most smoothing methods rely, to some extent, on periodic applications of COLAMD [12] to maintain a good column ordering; however, this requires the complete factorisation to be performed and limits the possible performance gains of the incremental solution. Typically, current methods are designed to avoid these traditionally expensive operations. In this regard, Kaess *et al.* have proposed iSAM2 [29], an incremental algorithm based on Bayes Trees [28] and QR decomposition which performs local reordering on nodes affected by loop closing. A similar solution using efficient incremental updates to Cholesky factorisation and block matrix formulation is proposed by Polok et al. [42]. While reordering at loop closing has significantly reduced the need for full reordering, these methods are suboptimal as they do not guarantee that changes between two consecutive orderings will be limited to the scope of the loop being closed.

In this article, it is proven that the permutation of two adjacent rows/columns in a matrix only affects the corresponding rows/columns in said matrix's Cholesky factor and formulas characterising how these changes occur are given. Based on this result, a factor recovery algorithm is proposed to recover the Cholesky factor of a permuted matrix based on the factor before permutation. This leads to a Hybrid Cholesky method combining factor recovery and re-computation to efficiently recover the Cholesky factor of the reordered system from the current factor when a full reordering is required in a SLAM setting.

First, the mathematical literature is reviewed(Section 2), followed by concepts and definitions (Section 3). The mathematical formulations and proofs related to

column permutation (Section 4) are then presented and used in the Factor Recovery method (Section 5). A metric to select between the presented method and full Cholesky decomposition for complete reordering is presented (Section 6) followed by a presentation of the Hybrid Cholesky decomposition (Section 7). Results comparing the regular and Hybrid Cholesky decomposition on SLAM datasets are presented and discussed (Section 8) and a summary is presented along with future work (Section 9).

2 Background

Consider an overdetermined system of linear equations of the form $\mathbf{Ax} = \mathbf{b}$ where $\mathbf{A}$ is sparse and of dimension $m(t) \times n(t)$. The least square solution of such systems is represented by the normal equation $\mathbf{A}^T\mathbf{Ax} = \mathbf{A}^T\mathbf{b}$ and can be efficiently obtained by calculating the Cholesky factorisation $\mathbf{LL}^T$ of the sparse symmetric positive definite matrix $\mathbf{C} = \mathbf{A}^T\mathbf{A}$. Classically, this has been done directly using dense matrices but quickly becomes computationally expensive, making filtering the standard for online problems. The use of sparse matrices and algorithms tailored to work with such data structures have provided a large performance gain across many fields, including SLAM.

Symbolic factorisation [21, 46] exploits the sparsity of the problem, which can yield large performance increases for the computation of Cholesky factors whose complexity goes from $O(n^3/2)$ down to

$$O\left(0.5 \sum_{i=0:n} (|\mathbf{L}_i| - 1)(|\mathbf{L}_i| + 2)\right), \tag{1}$$

where $|\mathbf{L}_i|$ is the number of non-zero entries in column i of $\mathbf{L}$ [38]. This can be done as a two step process, with the symbolic Cholesky factorisation predicting which entries of $\mathbf{L}$ will be non-zero before calculating their actual value. In recent years, this has been extended to super-nodal sparse Cholesky factorisation [7] and graphs [25] for parallel computation while recent articles have detailed GPU implementations [45, 53]. Out-of-core methods adapted to very large problems have also been proposed [44].

The performance of sparse matrix decomposition methods are highly dependent on the fill-ins and therefore rely heavily on fill-reducing ordering algorithms. The optimal ordering problem has been demonstrated to be NP-Complete [52] but many heuristics exist to obtain a good ordering. The most popular are based on the minimum degree algorithm [4, 12] or graph partitioning [20, 32, 37]. A review of such methods has been done by Agarwal and Olson [3] in the case of SLAM; comparisons on multiple datasets are available in publications relating to the respective methods and recent advances in graph partitioning are reviewed by Buluç et al. [6].

In the case of dynamic problems where some values of $\mathbf{A}$ change, multiple methods have been proposed to translate modifications on $\mathbf{C} = \mathbf{A}^T\mathbf{A}$ to modifications on $\mathbf{LL}^T$ in order to avoid completely recalculating the factorisation. Early methods to modify Cholesky factorisation in the dense case are reviewed by Gill *et al.* [22]. Another comprehensive review of modifications of dense Cholesky factorisations is presented by Golub and Van Loan [23]. The algorithm outlined by Gill *et al.* [22] for rank 1 modification was later adapted to sparse matrices by Davis and Hager [13], providing a method with complexity linear with the number of the non zero entries in

L. This was later extended to rank-k modification [14] and to row/column modification [15] for systems where the dimension of **A** increases with time. These methods are still the accepted standard in numerical computation software and have been the basis for the incremental Cholesky based SLAM proposed by Polok *et al.* [42]. Some computations may be saved, however, if it is desired to displace an existing row/column of **C** rather than the more general arbitrary row/column modification. This would prove advantageous for autonomous robot navigation that often have limited computational capabilities.

3 Concepts and Definitions

In this article, the notation is the same as that used by Davis and Hager [13]. Calligraphic letters (such as $\mathcal{L}$, $\mathcal{A}$ and $\mathcal{C}$) are used to represent sets. Matrices are in upper case bold, while vectors are identified by lower case bold letters. Scalars are represented by lower case letters. The non-zero pattern of a $n \times n$ matrix $\mathbf{L}$ is denoted by $\mathcal{L}$ where $\mathcal{L} = \{\mathcal{L}_1, \ldots, \mathcal{L}_n\}$ and $\mathcal{L}_j = \{i | l_{ij} \neq 0\}$ is the non-zero pattern of column j of $\mathbf{L}$. The notation $\mathcal{L}_j^\sharp$ represents the multiset of $\mathcal{L}_j$ such that $\mathcal{L}_j^\sharp = \{(i, m(i)) | i \in \mathcal{L}_j\}$ where $m(i)$ is the multiplicity of element i [13].

The SLAM notation used is similar to what can be found in other articles [17, 29, 30, 42]. $\mathbf{s}$ represents a vehicle pose and $\mathbf{s}_{0:k}$ represents a series of poses from time 0 to k. Similarly, $\mathbf{l}$ indicates landmark position, $\mathbf{u}$ represents the control input and $\mathbf{z}$ the observation. $\mathbf{s}_i = f_i(\mathbf{s}_{i-1}, \mathbf{u}_i)$ is the control function relating one pose to the next while $\mathbf{z}_i = h_i(\mathbf{s}_{z_i}, \mathbf{l}_{z_i})$ is the observation function relating a pose to a landmark. Their partial derivative with regards to $\mathbf{s}$ and $\mathbf{l}$ are

$$\frac{\partial}{\partial \mathbf{s}_{i-1}} f_i(\mathbf{s}_{i-1}, \mathbf{u}_i) = \mathbf{F}_i \qquad \frac{\partial}{\partial \mathbf{s}_i} f_i(\mathbf{s}_{i-1}, \mathbf{u}_i) = \mathbf{G}_i = \mathbf{I}$$

$$\frac{\partial}{\partial \mathbf{s}_{z_i}} h_i(\mathbf{s}_{z_i}, \mathbf{l}_{z_i}) = \mathbf{H}_i \qquad \frac{\partial}{\partial \mathbf{l}_{z_i}} h_i(\mathbf{s}_{z_i}, \mathbf{l}_{z_i}) = \mathbf{J}_i$$

Where $\mathbf{I}$ is the identity matrix.

3.1 SLAM - Basics

Mathematically, the SLAM problem consists of calculating the probability distribution of the vehicle attitude ($\mathbf{s}$) and landmark positions ($\mathbf{l}$) at time k given the initial attitude $\mathbf{s}_0$, control inputs ($\mathbf{u}_{1:k}$) and n_z observations ($\mathbf{z}_{1:n_z}$) with their corresponding data associations ($\mathbf{n}_{1:n_z}$) . Let the map be the set of all landmarks at time k

$$\mathcal{M} = \{\mathbf{l}_1, \mathbf{l}_2, \ldots\}$$

The SLAM problem can then be stated as finding the probability of the possible attitude and map

$$p(\mathbf{s}_k, \mathcal{M} | \mathbf{z}_{1:n_z}, \mathbf{u}_{1:k}, \mathbf{n}_{1:n_z}, \mathbf{s}_0) \tag{2}$$

Given that the control input at time k, designated by $\mathbf{u}_k$ relates the state at $k-1$ to the state at time k

$$p(\mathbf{s}_k | \mathbf{s}_{k-1}, \mathbf{u}_k) \tag{3}$$

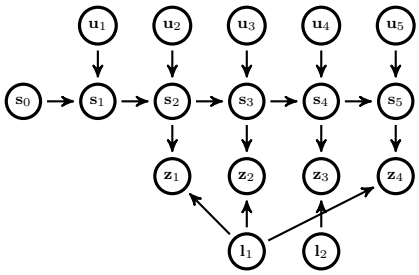

Fig. 1 Example of a Bayes Network representation of a small SLAM problem

that an arbitrary observation $\mathbf{z}_i$ obtained at time k depends on the map, data association and position at that time

$$p(\mathbf{z}_i|\mathbf{s}_{z_i}, \mathbf{n}_{z_i}, \mathcal{M})$$

and that the data association probability is given by

$$p(\mathbf{n}_{z_i}|\mathbf{s}_{z_i}, \mathbf{z}_i, \mathcal{M})$$

In the case where the data association is provided and considered exact, the observation probability is then

$$p(\mathbf{z}_i|\mathbf{s}_{z_i}, \mathbf{l}_{z_i}) \tag{4}$$

Through application of Bayes' rule and the theorem of total probability, (2) can be restated as finding the path $\mathbf{s}_{1:k}$ and map $\mathcal{M}$ probability

$$p(\mathbf{s}_{1:k}, \mathcal{M}|\mathbf{z}_{1:n_z}, \mathbf{u}_{1:k}, \mathbf{s}_0) \tag{5}$$

In the literature, this is referred to as full SLAM and used by most smoothing and mapping algorithms [17, 30, 36, 42]. Often it is desired to find the particular path and map that maximize (5). To calculate this efficiently, (3) and (4) can be used to factorize (5) as a product of simpler probability densities. The resulting equation is

$$\underset{\mathbf{s}_{1:k},\mathcal{M}}{\arg\max} \left(p(\mathbf{s}_0) \prod_{i=1}^{k} p(\mathbf{s}_i|\mathbf{s}_{i-1}, \mathbf{u}_i) \prod_{j=1}^{n_z} p(\mathbf{z}_j|\mathbf{s}_{z_j}, \mathbf{l}_{z_j}) \right) \tag{6}$$

where the constant factor $p(\mathbf{s}_0)$ can be normalised out. Regardless of the formulation, the SLAM problem can be represented by a graphical model as a collection of nodes related by constraints (observation and control). In Figure 1, the graphical model representation for a problem with 2 landmarks, 4 observations, 5 control inputs and 6 position nodes is depicted as a Bayesian Network, which can be obtained directly from (3) and (4) to get the relations between variables. The same SLAM problem is depicted in the left side of Figure 2 and Figure 3 as a factor graph and Markov Random Field (MRF) respectively using (6).

3.2 SLAM - Graph and Matrices

In order to solve (5) directly, the Gaussian assumption must be used for all probability distributions and the negative of the natural logarithm is taken to transform the probability maximisation to a minimisation problem and obtain

$$\underset{\mathbf{s}_{1:k},\mathcal{M}}{\arg\min} \left(\sum_{i=1}^{k} \|f_i(\mathbf{s}_{i-1}, \mathbf{u}_i) - \mathbf{s}_i\|^2_{\mathbf{\Sigma}_{u_i}} + \sum_{i=1}^{n_z} \|h_i(\mathbf{s}_{z_i}, \mathbf{l}_{z_i}) - \mathbf{z}_i\|^2_{\mathbf{\Sigma}_{z_i}} - b \right) \tag{7}$$

where $\mathbf{\Sigma}_{u_i}$ and $\mathbf{\Sigma}_{z_i}$ are the covariance matrices of control and observation i respectively. $\|\mathbf{a}\|^2_{\mathbf{\Sigma}} = \mathbf{a}^T\mathbf{\Sigma}^{-1}\mathbf{a}$ represents the Mahalanobis norm given the covariance matrix $\mathbf{\Sigma}$, and b represents the constant terms

$$b = \sum_{i=1}^{k} \ln \frac{1}{\sqrt{2\pi\mathbf{\Sigma}_{u_i}}} + \sum_{i=1}^{n_z} \frac{1}{\sqrt{2\pi\mathbf{\Sigma}_{z_i}}}$$

which are ignored since they do not affect the position of the minimum. Linearising (7) and including the $\mathbf{\Sigma}$ matrices inside the norms using

$$\|\mathbf{a}\|^2_{\mathbf{\Sigma}} = \mathbf{a}^T\mathbf{\Sigma}^{-1}\mathbf{a} = \|\mathbf{\Sigma}^{-T/2}\mathbf{a}\|^2_2$$

(7) yields

$$\underset{\mathbf{s}_{1:k},\mathcal{M}}{\arg\min} \left(\sum_{i=1}^{k} \|\hat{\mathbf{F}}_i\delta s_{i-1} - \hat{\mathbf{G}}\delta s_i - \mathbf{\Sigma}^{-T/2}(\mathbf{s}_i - f_i(\mathbf{s}_{i-1}, \mathbf{u}_i)\|^2_2 + \sum_{i=1}^{n_z} \|\hat{\mathbf{H}}_i\delta\mathbf{s}_{z_i} + \hat{\mathbf{J}}_i\delta\mathbf{l}_{z_i} - \mathbf{\Sigma}^{-T/2}(\mathbf{z}_i h_i - (\mathbf{s}_{z_i}, \mathbf{l}_{z_i}))\|^2_2 \right) \tag{8}$$

where $\hat{\mathbf{G}} = -\mathbf{\Sigma}_i^{-T/2}\mathbf{I}$ and $\mathbf{I}$ is an identity matrix of appropriate size. $\hat{\mathbf{F}}_i = \mathbf{\Sigma}_i^{-T/2}\mathbf{F}$, $\hat{\mathbf{H}}_i = \mathbf{\Sigma}_i^{-T/2}\mathbf{H}$, $\hat{\mathbf{J}}_i = \mathbf{\Sigma}_i^{-T/2}\mathbf{J}$, are the new variables obtained once the covariance has been distributed. Posing $\mathbf{a}_i = \mathbf{\Sigma}^{-T/2}(\mathbf{s}_i - f_i(\mathbf{s}_{i-1}, \mathbf{u}_i)$, $\mathbf{c}_i = \mathbf{\Sigma}^{-T/2}(\mathbf{z}_i - h_i(\mathbf{s}_{z_i}, \mathbf{l}_{z_i}))$ and expressing (8) in matrix form, the following over determined system is obtained

$$\mathbf{A}\begin{bmatrix}\delta\mathbf{s}_{0:k}\\ \delta\mathbf{l}_{1:n_l}\end{bmatrix} = \begin{bmatrix}\mathbf{a}_{0:k}\\ \mathbf{c}_{1:n_z}\end{bmatrix}$$

where the matrix $\mathbf{A}$ is a block matrix of Jacobians. This equation is closely related to the factor graph as seen in Figure 2. The least square solution can be obtained by solving the normal equation

$$\mathbf{A}^T\mathbf{A}\begin{bmatrix}\delta\mathbf{s}_{0:k}\\ \delta\mathbf{l}_{1:n_l}\end{bmatrix} = \mathbf{A}^T\begin{bmatrix}\mathbf{a}_{0:k}\\ \mathbf{c}_{1:n_z}\end{bmatrix} \tag{9}$$

which can equivalently be obtained by evaluating the norm in (8), taking the derivative and solving by equating to 0. This problem is related to the MRF representation as seen in Figure 3 where the Gramian matrix $\mathbf{A}^T\mathbf{A}$ is the adjacency matrix of the MRF. (9) can be efficiently solved using the Cholesky decomposition

$$\mathbf{L}\mathbf{L}^T\begin{bmatrix}\delta\mathbf{s}_{0:k}\\ \delta\mathbf{l}_{1:n_l}\end{bmatrix} = \mathbf{A}^T\begin{bmatrix}\mathbf{a}_{0:k}\\ \mathbf{c}_{1:n_z}\end{bmatrix}$$

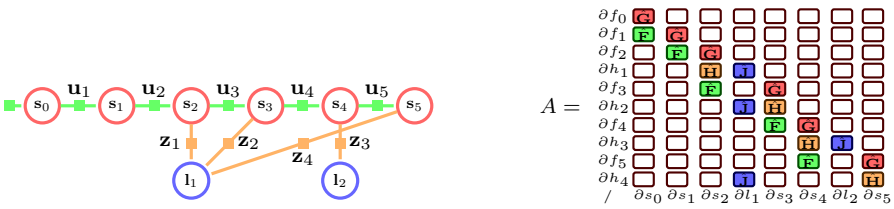

Fig. 2 Factor Graph representation of a small SLAM problem with the associated matrix representation. There is one variable node (circle) and matrix column associated with each unknown (pose or landmark). There is one factor node (square) and matrix row associated with each measurement. Note that u_k and z_k do not represent the functions of the factor nodes but the measurement they originated from.

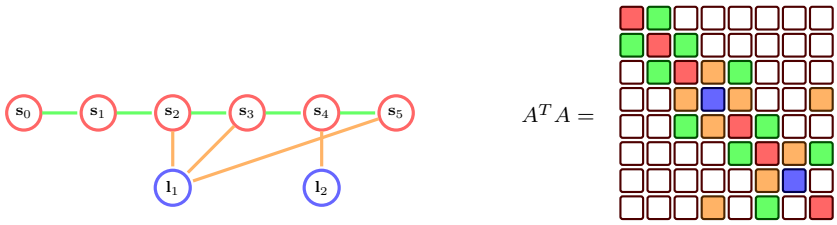

Fig. 3 MRF representation of a small SLAM problem with the associated matrix representation. There is a variable node associated with each row/column of the matrix and diagonal entry. A symmetric pair of off diagonal entry is related to each edge.

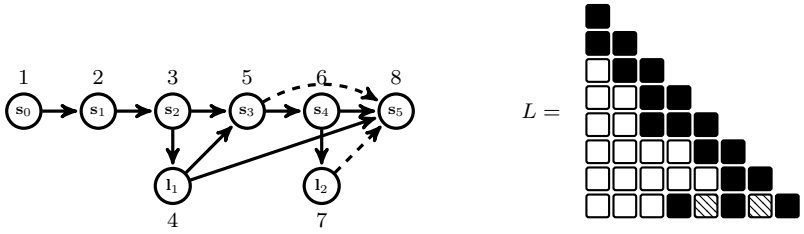

Fig. 4 MRF node elimination and associated Cholesky factor. The dark edges and dark cells correspond to the dependence between variables while the dotted edges and hatched cells corresponds to the dependencies added during variable elimination

followed by forward and backward substitution. The structure of the factor $\mathbf{L}$ for a sample matrix can be seen in Figure 4 where the fill-ins of the $\mathbf{L}$ matrix correspond to edges added when conducting Bayesian elimination on the MRF corresponding to $\mathbf{A}^T\mathbf{A}$.

3.3 Elimination Tree

The elimination tree is a structure illustrating the dependencies between columns of $\mathbf{L}$ in sparse Cholesky factorisation. Elimination trees have a wide array of uses and many interesting properties explained thoroughly by Liu [38]. Knowledge necessary

for the reading of this paper is summarised here for convenience using the notation presented by Davis and Hager [13] whenever possible. The function $\pi(j)$, called the parent function, represents the lowest index column that depends on column j and is defined as

$$\pi(j) = min(\mathcal{L}_j \setminus \{j\}).$$

The children multifunction $\pi^{-1}(k)$ represents the set of all children of k or alternatively, the set of columns on which k depends and is defined as

$$\pi^{-1}(k) = \{j | \pi(j) = k\}.$$

The set of nodes between node j and the root is defined as the sequence of parents, or path

$$\mathcal{P}(j) = \{\pi(j), \pi(\pi(j)), \dots\}.$$

3.4 Symbolic Factorisation

The symbolic Cholesky factorisation using multi-sets is defined with the same notation as Davis and Hager [13]. When presented such that operations are on $\mathcal{C}$ like the algorithm used by Liu [38], the following is obtained:

Algorithm 1 Symbolic Cholesky Factorisation

1: $\pi(j) \leftarrow 0 \; \forall \; j \in [1, n]$
2: $\mathcal{L}_j^\sharp \leftarrow \{(i, 1) | i \in \mathcal{C}_j\}$
3: **for** $j \leftarrow 1 : n$ **do**
4: $\quad \mathcal{L}_j^\sharp \leftarrow \mathcal{L}_j^\sharp + \left(\sum_{i \in \pi^{-1}(j)} \mathcal{L}_i \setminus \{i\} \right)$
5: $\quad \pi(j) \leftarrow \min(\mathcal{L}_j \setminus \{j\})$
6: **end for**

This algorithm performs the right looking decomposition of the sparse matrix $\mathcal{C}$ while keeping count of the number of children of $\mathcal{L}_j$ that contribute to each of the non-zeros entries. Note that only the lower triangular part of $\mathcal{C}$ is taken into account.

3.5 COLAMD

SLAM++ and many other solvers in the literature use COLAMD (COLumn Approximate Minimum Degree) to obtain a fill-reducing ordering and increase the efficiency of matrix factorisation. COLAMD is based on Approximate Minimum Degree (AMD) [4] which is itself based on minimum degree algorithms [21, 47]. These algorithms consists of eliminating nodes that have the lowest degree (number of edges) first. In its simplest form, this can be done by maintaining a list of nodes sorted by degree and updating said list while removing nodes by Bayesian elimination. Typically, node selection is done at random amongst the nodes of same degree. AMD forgoes the need for book keeping and sorting by calculating an approximate degree for the nodes while COLAMD allows for the calculations to take place directly on matrix $\mathbf{A}$ instead of on the symmetric matrix $\mathbf{A}^T\mathbf{A}$. For more details on ordering algorithms, the reader is referred to [11].

3.6 SLAM++

SLAM++ [42] is a Cholesky based non-linear least square solver which is optimised for block matrices, making it particularly well suited for the structure of SLAM problems. Although it supports non-linear optimisation methods, the linear version will be used in this article as a basis for implementing the hybrid Cholesky method and obtaining simulation results. Algorithm 2 is a simplified illustration of how SLAM++ works.

Algorithm 2 SLAM++
Input: $\mathbf{A}$ Measurement Jacobians
Input: $\mathbf{b}$ Measurement Error
Input: $\mathbf{L}$ Cholesky factor of $\mathbf{A}^T\mathbf{A}$
for all $i \in$ new measurements **do**
  add i to the graph
  calculate derivatives $\mathbf{G}_i$, $\mathbf{F}_i$, $\mathbf{H}_i$ and $\mathbf{J}_i$
  add derivatives to new row of $\mathbf{A}$
  add measurement error to new row of $\mathbf{b}$
  if non-zero density of $\mathbf{L} > 0.02$ **then**
    $\mathbf{P} \leftarrow COLAMD(\mathbf{A})$
    $\mathbf{L} \leftarrow \text{cholesky}(\mathbf{P}^T\mathbf{A}^T\mathbf{A}\mathbf{P})$
  else
    incremental_cholesky_update$(\mathbf{L}, \mathbf{A}, \mathbf{P}, \mathbf{b})$
  end if
  forward_backward_substitute$(\mathbf{L}, \mathbf{P}^T\mathbf{A}^T\mathbf{b}, \mathbf{s})$
end for

For more information regarding the optimised block matrix Cholesky factorisation, authors can refer to [41] while the incremental update scheme of SLAM++ is described in more detail in [42].

4 Swapping Adjacent Columns

In this section, the effect on $\mathbf{L}$ of moving a given row/column of the symmetric matrix $\mathbf{C}$ to another position will be analysed for different base cases. From these building blocks, arbitrary changes in the ordering of the row/column of $\mathbf{C}$ can be reflected on $\mathbf{L}$. In an effort to be consistent, the index notation of rows and columns will refer to their position in the original matrix $\mathbf{C}$ regardless of their position after the permutation is applied.

Proposition 1 shows that in the case where a given row/column of $\mathbf{C}$ exchanges position with an adjacent row/column, the sequential application of the row/column deletion and addition equations presented by Davis and Hager [15] and associated rank 1 updates [22] simplify such that only the two row/columns involved are modified. An expression on calculating the modified row/columns of $\mathbf{L}$ is given.

Proposition 1 (Row/Column Permutation) *Let $\mathbf{C}_j$ and $\mathbf{C}_{j+1}$ be two adjacent rows/columns in $\mathbf{C}$, $\mathbf{P} = [\mathbf{e}_1, \ldots, \mathbf{e}_{j+1}, \mathbf{e}_j, \ldots, \mathbf{e}_n]$ a permutation matrix such that $\bar{\mathbf{C}} = \mathbf{P}^T\mathbf{C}\mathbf{P}$ is $\mathbf{C}$ where rows and columns $\mathbf{C}_j$ and $\mathbf{C}_{j+1}$ have exchanged positions. Let*

$\mathbf{L}\mathbf{L}^T$ and $\bar{\mathbf{L}}\bar{\mathbf{L}}^T$ be the Cholesky factors of $\mathbf{C}$ and $\bar{\mathbf{C}}$ respectively, where the overhead bar indicates terms affected by the permutation. If the matrices are defined as:

$$\mathbf{C} = \begin{bmatrix} \mathbf{C}_{11} & \mathbf{c}_{12} & \mathbf{c}_{13} & \mathbf{C}_{14} \\ \mathbf{c}_{12}^T & c_{22} & c_{23} & \mathbf{c}_{42}^T \\ \mathbf{c}_{13}^T & c_{32} & c_{33} & \mathbf{c}_{43}^T \\ \mathbf{C}_{41} & \mathbf{c}_{42} & \mathbf{c}_{43} & \mathbf{C}_{44} \end{bmatrix} \qquad \mathbf{L} = \begin{bmatrix} \mathbf{L}_{11} & & & \\ \mathbf{l}_{12}^T & l_{22} & & \\ \mathbf{l}_{13}^T & l_{32} & l_{33} & \\ \mathbf{L}_{41} & \mathbf{l}_{42} & \mathbf{l}_{43} & \mathbf{L}_{44} \end{bmatrix}$$

$$\bar{\mathbf{C}} = \begin{bmatrix} \mathbf{C}_{11} & \mathbf{c}_{13} & \mathbf{c}_{12} & \mathbf{C}_{14} \\ \mathbf{c}_{13}^T & c_{33} & c_{32} & \mathbf{c}_{43}^T \\ \mathbf{c}_{12}^T & c_{23} & c_{22} & \mathbf{c}_{42}^T \\ \mathbf{C}_{41} & \mathbf{c}_{43} & \mathbf{c}_{42} & \mathbf{C}_{44} \end{bmatrix} \qquad \bar{\mathbf{L}} = \begin{bmatrix} \mathbf{L}_{11} & & & \\ \mathbf{l}_{13}^T & \bar{l}_{33} & & \\ \bar{\mathbf{l}}_{12}^T & \bar{l}_{32} & \bar{l}_{22} & \\ \mathbf{L}_{41} & \bar{\mathbf{l}}_{43} & \bar{\mathbf{l}}_{42} & \bar{\mathbf{L}}_{44} \end{bmatrix}$$

The new Cholesky factor $\bar{\mathbf{L}}$ can be recovered from $\mathbf{L}$ as follows:

$$\begin{aligned} \bar{\mathbf{l}}_{12} &= \mathbf{l}_{12} & \bar{l}_{33} &= \sqrt{l_{33}^2 + l_{32}^2} \\ \bar{l}_{22} &= l_{33}\frac{l_{22}}{\bar{l}_{33}} & \bar{l}_{32} &= l_{32}\frac{l_{22}}{\bar{l}_{33}} \\ \bar{\mathbf{l}}_{43} &= \frac{\mathbf{l}_{42}l_{32} + \mathbf{l}_{43}l_{33}}{\bar{l}_{33}} & \bar{\mathbf{l}}_{42} &= \frac{\mathbf{l}_{42}l_{33} - \mathbf{l}_{43}l_{32}}{\bar{l}_{33}} \\ \bar{\mathbf{L}}_{44} &= \mathbf{L}_{44} \end{aligned}$$

where all other entries are the same in both factors.

Proof The symmetric permutation $\mathbf{P}^T\mathbf{C}\mathbf{P}$ can be decomposed in a row/column deletion followed by a row/column addition on $\mathbf{C}$. The deletion operation [15] applied on row/column 2 affects the Cholesky factor such that the second row and column $\mathbf{l}_{12}^T$, and $[l_{22}\ l_{32}\ \mathbf{l}_{42}]^T$ are removed and the following columns are modified by a rank 1 update. We now have

$$\bar{\mathbf{C}} = \begin{bmatrix} \mathbf{C}_{11} & \mathbf{c}_{13} & \mathbf{C}_{14} \\ \mathbf{c}_{13}^T & c_{33} & \mathbf{c}_{43}^T \\ \mathbf{C}_{41} & \mathbf{c}_{43} & \mathbf{C}_{44} \end{bmatrix} \qquad \bar{\mathbf{L}} = \begin{bmatrix} \mathbf{L}_{11} & & \\ \mathbf{l}_{13}^T & \bar{l}_{33} & \\ \mathbf{L}_{41} & \bar{\mathbf{l}}_{43} & \hat{\mathbf{L}}_{44} \end{bmatrix}$$

The rank 1 update operation [22] affecting the blocks after column 2 is

$$\begin{bmatrix} \bar{l}_{33} & \\ \bar{\mathbf{l}}_{43} & \hat{\mathbf{L}}_{44} \end{bmatrix} \begin{bmatrix} \bar{l}_{33} & \bar{\mathbf{l}}_{43}^T \\ & \hat{\mathbf{L}}_{44}^T \end{bmatrix} = \begin{bmatrix} l_{33} & \\ \mathbf{l}_{43} & \mathbf{L}_{44} \end{bmatrix} \begin{bmatrix} l_{33} & \mathbf{l}_{43}^T \\ & \mathbf{L}_{44}^T \end{bmatrix} + \begin{bmatrix} l_{32} \\ \mathbf{l}_{42} \end{bmatrix} \begin{bmatrix} l_{32} & \mathbf{l}_{42}^T \end{bmatrix} \tag{10}$$

where $\hat{\mathbf{L}}_{44}$ corresponds to an intermediate result. By by evaluating the two upper blocks of (10) and solving the equations obtained for $\bar{l}_{33}$ and $\bar{\mathbf{l}}_{43}$, it can be shown that

$$\bar{l}_{33} = \sqrt{l_{33}^2 + l_{32}^2} \qquad \bar{\mathbf{l}}_{43} = \frac{\mathbf{l}_{42}l_{32} + \mathbf{l}_{43}l_{33}}{\bar{l}_{33}}$$

Applying the row/column addition operation [15] to put the removed row/column in its new position, we obtain

$$\bar{\mathbf{C}} = \begin{bmatrix} \mathbf{C}_{11} & \mathbf{c}_{13} & \mathbf{c}_{12} & \mathbf{C}_{14} \\ \mathbf{c}_{13}^T & c_{33} & c_{32} & \mathbf{c}_{43}^T \\ \mathbf{c}_{12}^T & c_{23} & c_{22} & \mathbf{c}_{42}^T \\ \mathbf{C}_{41} & \mathbf{c}_{43} & \mathbf{c}_{42} & \mathbf{C}_{44} \end{bmatrix} \qquad \bar{\mathbf{L}} = \begin{bmatrix} \mathbf{L}_{11} & & & \\ \mathbf{l}_{13}^T & \bar{l}_{33} & & \\ \bar{\mathbf{l}}_{12}^T & \bar{l}_{32} & \bar{l}_{22} & \\ \mathbf{L}_{41} & \bar{\mathbf{l}}_{43} & \bar{\mathbf{l}}_{42} & \bar{\mathbf{L}}_{44} \end{bmatrix}$$

where, from [15],

$$\begin{bmatrix}\mathbf{c}_{12}\\ c_{32}\end{bmatrix} = \begin{bmatrix}\mathbf{L}_{11} & \\ \mathbf{l}_{13} & l_{33}\end{bmatrix}\begin{bmatrix}\bar{\mathbf{l}}_{12}\\ \bar{l}_{32}\end{bmatrix} \tag{11}$$

$$\bar{l}_{22} = c_{22} - \begin{bmatrix}\mathbf{l}_{12} & l_{32}\end{bmatrix}\begin{bmatrix}\mathbf{l}_{12}\\ l_{32}\end{bmatrix} \tag{12}$$

$$\mathbf{l}_{42} = \left(\mathbf{c}_{42} - \begin{bmatrix}\mathbf{L}_{41} & \bar{\mathbf{l}}_{43}\end{bmatrix}\begin{bmatrix}\mathbf{l}_{12}\bar{l}_{32}\end{bmatrix}\right)/\bar{l}_{22} \tag{13}$$

$$\bar{\mathbf{L}}_{44}\bar{\mathbf{L}}_{44} = \hat{\mathbf{L}}_{44}\hat{\mathbf{L}}_{44} - \bar{\mathbf{l}}_{42}\bar{\mathbf{l}}_{42} \tag{14}$$

Solving (11) for l_{12} gives

$$\bar{\mathbf{l}}_{12} = \mathbf{l}_{12}.$$

l_{22} and $\mathbf{l}_{42}$ are obtained directly by simplifying (12) and (13) respectively and thus

$$\bar{l}_{22} = l_{33}\frac{l_{22}}{\bar{l}_{33}} \qquad\qquad \bar{\mathbf{l}}_{42} = \frac{\mathbf{l}_{42}l_{33} - \mathbf{l}_{43}l_{32}}{\bar{l}_{33}}$$

Substituting the equation for $\hat{\mathbf{L}}_{44}\hat{\mathbf{L}}_{44}^T$ of (10) in (14) leads to

$$\bar{\mathbf{L}}_{44}\bar{\mathbf{L}}_{44}^T = \mathbf{L}_{44}\mathbf{L}_{44}^T + \mathbf{l}_{43}\mathbf{l}_{43}^T + \mathbf{l}_{42}\mathbf{l}_{42}^T - \bar{\mathbf{l}}_{43}\bar{\mathbf{l}}_{43}^T - \bar{\mathbf{l}}_{42}\bar{\mathbf{l}}_{42}^T,$$

from which one can obtain

$$\bar{\mathbf{L}}_{44}\bar{\mathbf{L}}_{44}^T = \mathbf{L}_{44}\mathbf{L}_{44}^T,$$

thus $\mathbf{L}_{44}$ does not change. □

While Proposition 1 is valid for both dense and sparse matrices, additional savings can be obtained working exclusively on sparse matrices. Proposition 2 demonstrates that, due to basic properties of the elimination tree and results derived in [13], a row/column of $\mathbf{C}$ can be moved without causing numerical changes in $\mathbf{L}$ (rows/columns must still be exchanged accordingly) as long as it remains between its parent and children in the elimination tree.

Proposition 2 (Sparse independent row/col permutation) *Let $\mathbf{C}_j$ be the index of a row/column in sparse matrix $\mathbf{C}$ with non-zero structure $\mathcal{C}$, $\mathbf{L}$ the Cholesky factor of $\mathbf{C}$ with non-zero structure $\mathcal{L}$, π an elimination tree on $\mathbf{L}$,*

$$\mathbf{P} = [\mathbf{e}_1, \ldots, \mathbf{e}_{j-1}, \mathbf{e}_{j+1}, \ldots \mathbf{e}_{k-1}, \mathbf{e}_j, \mathbf{e}_k, \ldots \mathbf{e}_n]$$

be a permutation matrix such that $\bar{\mathbf{C}} = \mathbf{P}^T\mathbf{C}\mathbf{P}$ is matrix $\mathbf{C}$ where column j has moved to position $k|k > j$ and $\bar{\mathbf{L}}$ the Cholesky factor of $\bar{\mathbf{C}}$.

if $\max\{\pi^{-1}(j)\} < k < \pi(j)$ then $\bar{\mathbf{L}} = \mathbf{P}^T\mathbf{L}\mathbf{P}$ and $\bar{\pi} = \pi|\pi(\pi^{-1}(j)) = k$. That is, there is no numerical or structural change and the columns are simply permuted with the appropriate index updated in the elimination tree.

Proof From [13] it is known that a change in $\mathcal{L}_j$ will only affect nodes in $\mathcal{P}(j)$ and, in Proposition 1, it was demonstrated that permuting two adjacent rows/columns of $\mathbf{C}$ will only modify the corresponding rows/columns of $\mathbf{L}$. Thus, change will not occur if $\pi(j) \neq j+1$ or, equivalently, $l_{j+1,j} = 0$ since in such case $j+1 \notin \mathcal{P}(j)$. Extending this by induction to displacement from j to $k|k > j$, change will not occur if $\pi(j) < k$ or, equivalently, $l_{x,j} = 0 \; \forall \; x \in]j;k]$ □

A similar result has also been demonstrated by Liu [38] while discussing topological orderings. When a node is moved past its adjacent parent or child, Proposition 3 uses the basic properties of the elimination tree and multiset representation of sparse Cholesky factors [13] to show how the structure $\mathcal{L}$ and the elimination tree π are affected.

Proposition 3 (Sparse dependent row/col permutation) *Let j, $j+1$ be the index of adjacent rows/columns in sparse matrix $\mathbf{C}$ with non-zero structure $\mathcal{C}$, $\mathbf{L}$ the Cholesky factor of $\mathbf{C}$ with non-zero structure $\mathcal{L}$, π an elimination tree on $\mathbf{L}$ and*

$$\mathbf{P} = [\mathbf{e}_1, \ldots, \mathbf{e}_{j+1}, \mathbf{e}_j, \ldots, \mathbf{e}_n]$$

be a permutation matrix such that $\bar{\mathbf{C}} = \mathbf{P}^T\mathbf{C}\mathbf{P}$ is matrix $\mathbf{C}$ where column j and $j+1$ are exchanged and $\bar{\mathbf{L}}$ is the Cholesky factor of $\bar{\mathbf{C}}$ with non-zero structure $\bar{\mathcal{L}}$ and elimination tree $\bar{\pi}$.

If $\pi(j) = j+1$, then the elimination tree can be updated by

$$\bar{\pi}(l) = \begin{cases} j+1 & l \in \{\pi^{-1}(j)\} \setminus \mathcal{U}_c \\ j & l \in \{\pi^{-1}(j+1)\} \setminus \mathcal{U}_c \\ \pi(l) & \textit{otherwise} \end{cases} \tag{15}$$

and the structure of the new Cholesky factor can be found by

$$\bar{\mathcal{L}}^\sharp_{j+1} = \mathcal{L}^\sharp_{j+1} - \mathcal{L}_j + \sum_{i \in \mathcal{U}_c} \mathcal{L}_i \tag{16}$$

$$\bar{\mathcal{L}}^\sharp_{j} = \mathcal{L}^\sharp_{j} + \bar{\mathcal{L}}_{j+1} - \sum_{i \in \mathcal{U}_c} \mathcal{L}_i \tag{17}$$

where elements of zero multiplicity are removed and

$$\mathcal{U}_c = \{u \in \pi^{-1}(j) | \mathcal{L}_u(j+1) \neq 0\} \tag{18}$$

Proof (15) and (18) can be obtained from Algorithm 1 where it can be seen that if $\pi(j) = j+1$, eliminating $j+1$ before j will change $\pi(l)$ from j to $j+1$ for children $l \in \pi^{-1}(j)$ that are also related to $j+1$ since $\mathcal{L}_l(j+1)$ will become the lowest index off diagonal nonzero entry. Note that the index update required by the position change is included in (15).

To easily compute $\bar{\mathcal{L}}_{j+1}$, the notion of symbolic factorisation using multisets must be used to keep track of how many children have contributed to same non-zero elements to $\mathcal{L}_{j+1}$ [13]. Since $\mathcal{L}_j$ will no longer be a child of $\mathcal{L}_{j+1}$, its contribution must be subtracted while the contribution of the children inherited from $\mathcal{L}_j$ must be taken into account. This is done by (16) and (17) respectively. □

5 Factor Recovery

In typical evolving least square problems, the graph changes every time new information is added, requiring partial re-computation of the Cholesky factors. To simplify the discussion, the following block matrix terminology will be used regardless of local or global reordering:

$$\begin{bmatrix} \mathbf{C}_{11} & \mathbf{C}_{12} \\ \mathbf{C}_{12}^T & \bar{\mathbf{C}}_{22} \end{bmatrix} = \mathbf{P}^T \begin{bmatrix} \mathbf{C}_{11} & \mathbf{C}_{12} \\ \mathbf{C}_{12}^T & \mathbf{C}_{22} \end{bmatrix} \mathbf{P} + \begin{bmatrix} 0 & 0 \\ 0 & \mathbf{W}_{l\times l} \end{bmatrix}$$
$$\begin{bmatrix} \mathbf{C}_{11} & \mathbf{C}_{12} \\ \mathbf{C}_{12}^T & \bar{\mathbf{C}}_{22} \end{bmatrix} = \begin{bmatrix} \mathbf{L}_{11} & 0 \\ \mathbf{L}_{12}^T & \bar{\mathbf{L}}_{22} \end{bmatrix} \begin{bmatrix} \mathbf{L}_{11} & \mathbf{L}_{12} \\ 0 & \bar{\mathbf{L}}_{22} \end{bmatrix}$$

In sequential systems such as SLAM, it is advantageous to constrain the reordering such that the last node has the highest index, limiting the scope of the frequent incremental changes to the top of the elimination tree or, equivalently, the lower right triangular portion $\bar{\mathbf{L}}_{22}$ [29, 42]. In existing methods, most reordering occurs when closing loops, in which case the rows/columns of $\mathbf{C}$ affected by the loop closing, $[\mathbf{C}_{12}^T\ \mathbf{C}_{22}]$ and $[\mathbf{C}_{12}^T\ \mathbf{C}_{22}^T]^T$ are reordered. A new partial factor $\bar{\mathbf{L}}_{22}$ is then obtained by applying resumed Cholesky on $\bar{\mathbf{C}}_{22}$. The ordering, however, is not guaranteed to be as good as the ordering on the complete graph. When a full reordering is desired, the proposed method can be used to reorder $[\mathbf{L}_{11}^T\ \mathbf{L}_{12}]^T$, while the original update mechanism is used to compute $\bar{\mathbf{L}}_{22}$, which has to be done regardless due to loop closing. Without loss of generality, it will be assumed in the following description that reordering and loop closing events always coincide.

First, a constrained ordering heuristic is applied to the full system and a new reordering vector $\bar{\mathbf{p}}$ is obtained. Without loss of generality, CCOLAMD [12] is used on $\mathbf{C}$ to obtain the ordering. From $\bar{\mathbf{p}}$ and the previous ordering $\mathbf{p}$, a relative ordering vector $\hat{\mathbf{p}}$ is obtained. Bubble Sort is applied on $\hat{\mathbf{p}}$, with the changes duplicated on $\mathbf{L}$ using the method described in Proposition 1 to calculate new values and that of Proposition 2 and 3 to maintain the nonzero structure, multiplicity and elimination tree. The sorting halts when the nodes constituting the columns $[\mathbf{L}_{11}^T\ \mathbf{L}_{12}]^T$ are in order. The remaining columns involved in the loop closing are computed by resumed Cholesky. An example is presented in Figure 5 and the method is summarised in the following algorithm:

Algorithm 3 Factor Recovery Algorithm
Input: $\mathcal{L}$ is the previous Cholesky factor
Input: n_s is the number of columns in the system
Input: i_f is the index of the first column of reordered $\mathbf{C}_{22}$
Input: $\hat{\mathbf{p}}$ is a relative permutation vector of length n_s

```
i_l ← 0
while i_l < i_f do
    nn ← n_s − 1
    for i ← n_s − 1 : −1 : i_l do
        if p[i] < p[i − 1] then
            swap(p[i], p[i − 1])
            if π[i − 1] = i then
                use Proposition 3
                updates nonzero values as in Proposition 1
```

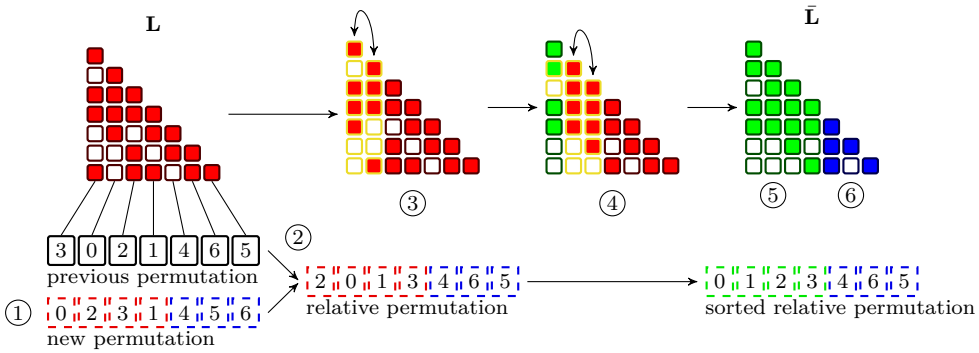

Fig. 5 Graphical example of the Factor Recovery Algorithm. (1) A new ordering is obtained. (2) Relative ordering is obtained from previous and new ordering. (3) and (4) Perform Bubble Sort and swap columns when necessary. (5) All non-loop node are ordered. (6) Perform resumed Cholesky on remaining columns.

else
  use Proposition 2
end if
$nn \leftarrow i$
end if
end for
$i_l \leftarrow nn$
end while
resumed Cholesky from n_f to $n_s - 1$

The use of the relatively inefficient Bubble Sort algorithm is justified in this case because the swapping operation is only defined for adjacent columns. A more efficient sorting algorithm would reduce the number of comparisons, but still require $O(n^2)$ adjacent column swaps. The extension of Proposition 1 to arbitrary column displacement is left as future work.

6 Threshold Selection

The part of Algorithm 3 responsible for reordering the columns of $[\mathbf{L}_{11}^T \mathbf{L}_{12}]^T$ has a worst case complexity of

$$O\left(\sum_{i=0}^{i_f} i|\hat{\mathcal{L}}_{i-1} \cup \hat{\mathcal{L}}_i|\right).$$

In the case where the permutations are localised, however, the permutation vector is close to a sorted array and the performance approaches

$$O\left(\sum_{i=0}^{i_f} |\hat{\mathcal{L}}_{i-1} \cup \hat{\mathcal{L}}_i|\right), \tag{19}$$

where $\hat{\mathcal{L}}$ is the evolving non-zero structure of the matrix. The cost in (19) is linear in the number of non-zeros compared to performing Cholesky on the same submatrix

Table 1 Datasets Characteristics

Dataset	Size	Loop Closings	Total Reordering	Reordered Using Factor Recovery
10k	64311	1431	32	6
City10k	20687	10688	13	3
CityTrees10k	14442	4343	13	0
CSAIL	1172	128	11	8
FR079	1217	229	8	8
FRH	2820	1505	13	13
Intel	1835	895	19	9
Killian	3995	2055	11	8
Victoria Park	10608	3489	14	6

Table 2 Datasets Used

Dataset	Author	Source
10k	G. Grisetti et al	SLAM++ [43]
City10k	M. Kaess et al	SLAM++ [43]
CityTrees10k	M. Kaess et al.	SLAM++ [43]
CSAIL	C. Stachniss	SLAM benchmarking [35]
FR079	C. Stachniss	SLAM benchmarking [35]
FRH	B, Steder et al.	SLAM benchmarking [35]
Intel	D. Hähnel Freiburg	SLAM++ [43]
Killian	M. Bosse and J. Leonard	SLAM++ [43]
Victoria Park	Jose Guivant	SLAM++ [43]

which can be seen in (1) to be quadratic. The efficiency of the Algorithm 3 compared to Cholesky is highly dependent on the assumption that few nodes need to be swapped. It is thus desired to obtain an estimation of the work required by both methods and a threshold to decide when one should be used over the other, similar to what is used by Supernodal Cholesky factorisation [7].

To obtain the threshold, a set of nine popular 2D SLAM datasets have been solved with a simplified version of the SLAM++ [41] algorithm. The source and authors of the datasets can be seen in Table 2. The datasets are stored in linearised form as sparse matrices of dense blocks, where each block represents the Jacobian or Hessian matrices of individual states and measurements. Permutations and swapping are carried out using block rows/columns, where the swapping of two block columns consist of elementary swap operations executed on individual columns of each block independently. When a full reordering is triggered based on SLAM++'s criterion (density of $\mathbf{L} \geq 2\%$, the new Cholesky factor is calculated both by regular Cholesky decomposition and recovered using Algorithm 3. The execution time, the overhead time and an estimation of the amount of work required are logged. Assuming the constant cost can be neglected in both cases, the cost (u) of the Cholesky decomposition and the proposed method are, respectively,

$$\begin{aligned} u_{Cholesky} &= \sum_{i=1}^{n} |\bar{\mathcal{L}}_i|^2 \\ u_{Recovery} &= \sum_{i \in \mathcal{S}} |\hat{\mathcal{L}}_i \cup \hat{\mathcal{L}}_{i-1}| \end{aligned} \tag{20}$$

Table 3 Operating Regions

Region	Fastest Method	Method Used
Q1 ($t_{ratio} > 1$, $\hat{u}_{ratio} > \alpha$)	Cholesky	Cholesky
Q2 ($t_{ratio} > 1$, $\hat{u}_{ratio} < \alpha$)	Cholesky	Hybrid
Q3 ($t_{ratio} < 1$, $\hat{u}_{ratio} < \alpha$)	Hybrid	Hybrid
Q4 ($t_{ratio} < 1$, $\hat{u}_{ratio} > \alpha$)	Hybrid	Cholesky

where $\mathcal{S}$ is the set of all the swaps that were done by the Bubble Sort during the proposed method's execution. To limit the overhead time of the method, Cholesky score is calculated from current factor $\mathcal{L}$ instead of the new factor $\bar{\mathcal{L}}$. Since it is assumed the current ordering is replaced by a better one, this is an upper bound on the number of operations to be done and requires less computation as $\mathcal{L}$ is already available. In the case of Cholesky factor recovery, the cost is estimated by multiplying the length of the permutation vector with a partial Kendrall-Tau distance (PKT) between the new and current permutation vectors. $PKT(k)$ is defined as the number of inversions required to obtain the k first terms of the current permutation vector from the previous one.

It can be seen from Algorithm 3 that the Factor Recovery method competes with Cholesky for the calculations of $[\mathbf{L}_{11}^T\mathbf{L}_{12}]^T$ only since $\mathbf{L}_{22}$ needs to be updated for loop closing. As such, in (20) $\mathcal{S}$ is the set of index swapped such that $[\mathbf{L}_{11}^T\mathbf{L}_{12}]^T$ is reordered. The cost of Cholesky for $\mathbf{L}_{22}$ will be subtracted from the total cost since this has to be executed in both cases. The cost of the two methods are estimated by

$$\hat{u}_{Cholesky} = \sum_{i=1}^{n} |\mathcal{L}_i|^2 - \sum_{i=k}^{n} |\mathcal{L}_i|^2 \tag{21}$$

$$\hat{u}_{Recovery} = \mathrm{PKT}(\bar{\mathbf{p}}, \mathbf{p}, k-1)n \tag{22}$$

where k is the index of the leftmost column involved in the loop closing. In order to have a threshold value that is independent of the matrix size and complexity, the execution time and estimated cost recorded for the proposed method are normalised by the execution time and estimated cost of the regular Cholesky factorisation respectively and the following is obtained

$$t_{ratio} = \frac{t_{Recovery}}{t_{Cholesky}} \qquad \hat{u}_{ratio} = \frac{\hat{u}_{Recovery}}{\hat{u}_{Cholesky}} \tag{23}$$

In Figure 6, the execution time ratio with regards to the estimated cost ratio is shown for each of the datasets. The sample points below $t_{ratio} = 1$ have a lower runtime with the proposed method than with Cholesky. For the best sample obtained, the execution time of the proposed method is smaller by a factor greater than 75. In order to select between both methods, a threshold value α on $\hat{u}_{ratio}$ must be chosen using $t_{ratio} = 1$. The resulting configuration divides the points in 4 regions explained in Table 3.

The threshold value used to choose between both method is shown in Figure 7 and is obtained by minimising the points in regions Q2 and Q4. It is thus advantageous to use Algorithm 3 if

$$\frac{\hat{u}_{Recovery}}{\hat{u}_{Cholesky}} < 5.21 \tag{24}$$

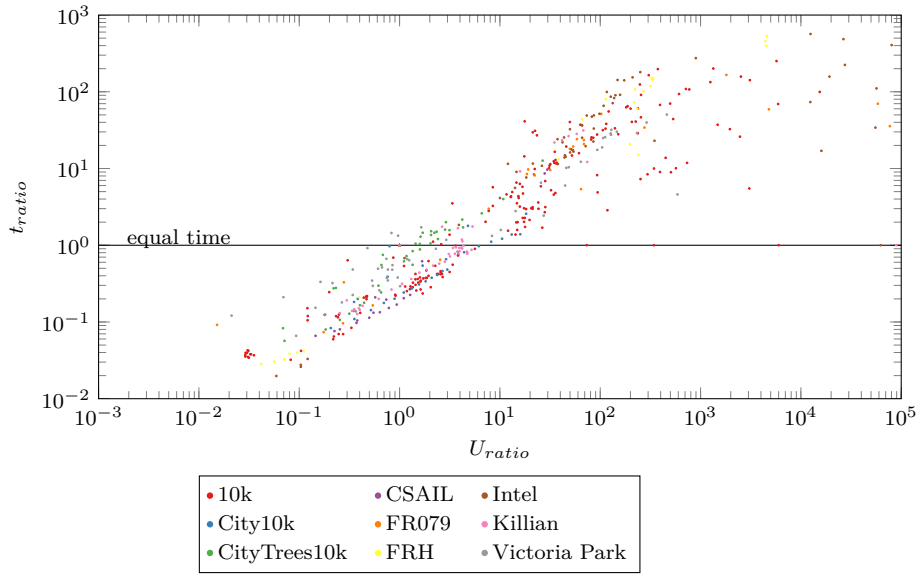

Fig. 6 Ratio of Cholesky execution time over the presented method execution time with regards to the estimated cost of the proposed method over the cost of Cholesky for 9 SLAM datasets

7 Hybrid Method

Let $\mathbf{L}$ be the previous Cholesky factor of a matrix $\mathbf{C}$, $\bar{\mathbf{C}}$ be a new matrix such that

$$\bar{\mathbf{C}} = \mathbf{P}^T\mathbf{C}\mathbf{P} + \begin{bmatrix} 0 & 0 \\ 0 & \mathbf{W}_{l\times l} \end{bmatrix}$$

where l is the lowest index of values that changed from $\mathbf{C}$ to $\bar{\mathbf{C}}$ and $\mathbf{P}$ the permutation matrix associated with a new ordering $\mathbf{p}$. The proposed Hybrid Cholesky factorisation method selects between regular Cholesky factorisation and the Factor Recovery algorithm combined with resumed Cholesky. First, the cost of complete Cholesky factorisation is found by evaluating (21) from 0 to $n-1$. (22) is evaluated between 0 and $l-1$ to estimate the work that can be done using Factor Recovery and the cost of the Resumed Cholesky is found by evaluating (21) from l to $n-1$. A ratio is calculated as in (23) by

$$u_{ratio} = \frac{\hat{u}_{Recovery}|_0^{l-1}}{\hat{u}_{Cholesky}|_0^{n-1} - \hat{u}_{Cholesky}|_l^{n-1}}$$

The obtained ratio is then compared to a threshold obtained experimentally on similar data or adaptively computed as $\mathbf{C}$ is modified (future work). If lower, Factor Recovery (Algorithm 3) is used with Resumed Cholesky otherwise, regular Cholesky is used. The resulting algorithm is as follows:

Algorithm 4 Hybrid Cholesky Algorithm
Input: l index of first value affected by $\mathbf{W}$
Input: k a given threshold

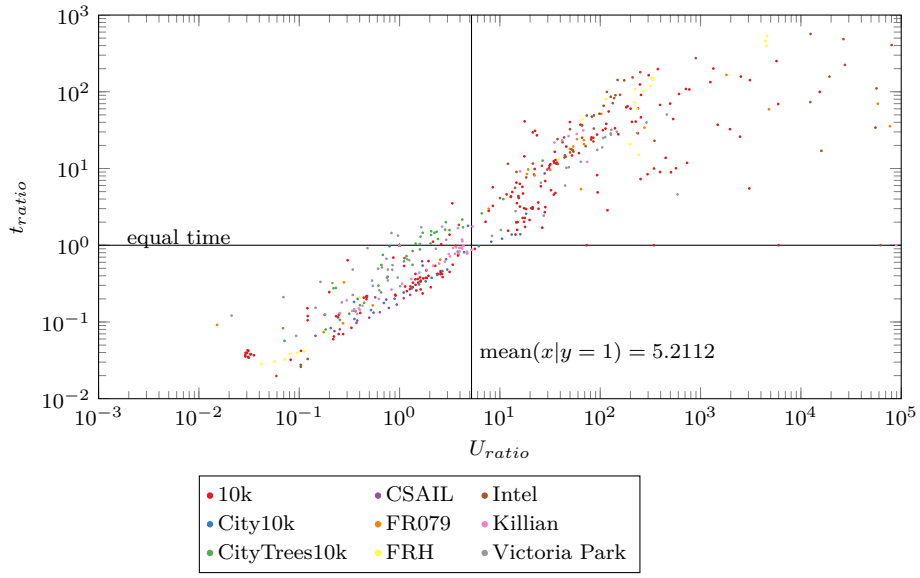

Fig. 7 Threshold selection based on Quadrant 2 and 4 point minimisation

Input: $\bar{\mathbf{C}}$ new system matrix
Input: n size of $\bar{\mathbf{C}}$
Input: $\mathbf{L}$ previous Cholesky factor
Output: $\bar{\mathbf{L}}$ new Cholesky factor
get $\hat{u}_{Recovery}|_0^{l-1}$ from (22)
get $\hat{u}_{Cholesky}|_0^{n-1}$ from (21)
get $\hat{u}_{Cholesky}|_l^{n-1}$ from (21)
if $\frac{U_{Recovery}|_0^{l-1}}{(U_{Cholesky}|_0^{n-1} - U_{Cholesky}|_l^{n-1})} < k$ **then**
    $\bar{\mathbf{L}} \leftarrow \mathbf{L}$
    Update columns 0 to $l-1$ of $\bar{\mathbf{L}}$ using Algorithm 3
    Update columns l to $n-1$ of $\bar{\mathbf{L}}$ using resumed Cholesky
else
    get $\bar{\mathbf{L}}$ by regular Cholesky decomposition
end if

8 Experimental Data and Discussion

This section will discuss the experimental results in two parts. First, the results concerning the threshold selection presented in Section 6 are presented and anomalies are discussed. The second part discusses the performance of the Hybrid method presented in Section 7.

8.1 Threshold Selection

Below the threshold expressed in (24), the execution time of Algorithm 3 will, on average, be lower than Cholesky; however, there are a few cases where it is twice as much. This undesired effect stems from two main causes. Firstly, the cost functions (21) and (22) are, for performance purposes, approximations of the actual cost of the corresponding algorithms. In some cases, this can lead to an underestimation (or overestimation) of the computational cost, producing a horizontal shift in the graph, which may causes points in Figure 7 to be in a different quadrant. This could be solved by using more accurate cost functions but the accuracy of the functions has to be carefully weighted against their complexity, which affects the overhead of the Hybrid Cholesky method. More research is needed to formally select scoring functions that appropriately balances accuracy and speed.

In the event the cost functions are exact ($t_{Cholesky} \propto \hat{u}_{Cholesky}$, $t_{Recovery} \propto \hat{u}_{Recovery}$) then the current method of selecting a discrete threshold to choose between Factor Recovery and traditional Cholesky decomposition is the optimal choice. Furthermore, if $\hat{u}_{Cholesky}$ and $\hat{u}_{Recovery}$ are obtained with the same performance metric, a simple comparison can be used to select the most efficient algorithm. Such assumption does not hold in the case where the cost functions are possibly biased approximations, and lead to a second possible error cause. In this article, the presence of approximation errors in the cost functions is mitigated by considering only SLAM structured problems when calculating the threshold. Using a function of multiple parameters instead of a fixed ratio as in (24) could help compensate for the approximation error and allow for a more general solution without requiring the evaluation of exact cost functions. To do so, the parameters affecting the error in the cost functions would need to be identified and their effect modelled.

The two current methods available to increase the accuracy come with an increased overhead cost of Hybrid Cholesky and introduce tuning issues to obtain a proper balance of performance and speed. In future research, the information to be computed for both Factor Recovery and Cholesky decomposition (such as the elimination tree) as well as quantities that can be incrementally calculated (such as Cholesky factor and adjacency matrix density) will be leveraged to select the most efficient algorithm while minimising unnecessary computations.

8.2 Hybrid Cholesky Decomposition

In order to obtain experimental results, the original SLAM++ implementation is modified such that, upon full reordering, Algorithm 4 is applied. Since it is desired to compare only the steps where reordering and re-factorisations are due to density, the corrected data is given as input to the solver to limit the number of complete re-factorisation due to changes in the linearisation point. By default, SLAM++'s variable reordering routine is triggered when the density of $\mathbf{L}$ is greater than 2%, which is left unchanged. The datasets used are the same as in Section 6. The characteristics of each dataset can be seen in Table 1 and an example of the final path obtained for the FR079 dataset can be seen in Figure 8.

Fig. 8 Map produced by the FR079 dataset using both methods. The results with and without Hybrid Cholesky are identical

A performance gain metric g is defined as

$$g = \frac{\sum\limits_{k\in\mathcal{V}} t_{Cholesky} - \sum\limits_{k\in\mathcal{V}} t_{Hybrid}}{\sum\limits_{k\in\mathcal{V}} t_{Cholesky}} \times 100\%$$

where t_{Hybrid} is the time taken by Algorithm 4 and $\mathcal{V}$ is the set of all steps where complete reordering was required. Figure 9 shows the performance gain g obtained by using the proposed Hybrid Cholesky method of Algorithm 4 instead of Cholesky for the reordering steps of each dataset. For consistency, the value of g is averaged over 30 experiments and a computer core is dedicated to the program's execution with no other processes running.

It can be seen that on most cases, the Cholesky factor can be recovered more efficiently by using the Hybrid Cholesky method. When the computation overhead is included, the reduction of the execution time reaches 67% in the best case, while in the worst case, the cost is not significantly higher (-6.56%) than Cholesky. In the cases where g is negative, the arithmetic part of the Hybrid Cholesky method still saves time (between 0.83% and 2.28%) as expected but not enough to make up for the overhead required to select between full Cholesky and Factor Recovery. In Table 1, the number of times reordering occurred is shown and it can be seen that for the datasets displaying a negative performance gain, the ratio of Factor Recovery uses to total number of reordering is much smaller than other methods. This explains why the overhead of the method, which is executed regardless of which method is chosen, becomes significant compared to the savings.

The Hybrid method presented yield some significant time savings on reordering steps but the savings do not translate to significant reductions in the total execution time since, as witnessed by the low number of reordering operations done by SLAM++ (Table 1), current methods are designed to avoid these traditionally expensive operations. The Hybrid Cholesky method's performance for recovering Cholesky Factor after reordering offers the foundations for methods that use variable reordering more liberally and could be combined with online graph clustering algorithms such as Fennel [48] or xDGP [49] for a truly incremental reordering phase.

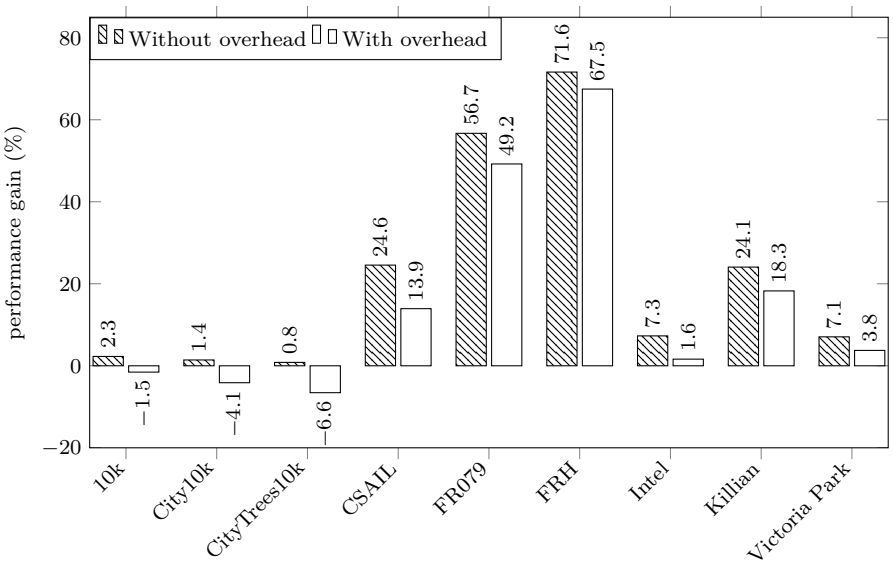

Fig. 9 The performance gain between the proposed method and regular Cholesky for full reordering on each dataset

The results obtained are also highly dependent on the variable ordering algorithm used since the cost of Algorithm 3 is proportional to the PKT distance between the current and new orderings. The present results are obtained using COLAMD [12] as a variable reordering strategy because it is the most popular for SLAM problems. Due to COLAMD's heuristic nature, consecutive calls may produce very different orderings (high PKT distance) and negatively impact the Hybrid Cholesky method's efficiency. The results would greatly benefit from fill-reducing ordering that also consider a global view of the graph [24, 32].

The increase in performance obtained by the Hybrid Method does not come at the expense of precision since the method is theoretically exact. Analysing the matrix norms however, show that the matrices are slightly different, which can be explained by the accumulation of small rounding errors. These are close to the machine epsilon for any given value and do not affect the algorithm. This can also be observed from the example in Figure 8 where the two paths overlap perfectly.

9 Conclusion

The Hybrid Cholesky method presented in this article provides an efficient alternative to the full Cholesky decomposition when it is desired to obtain a full reordering of the matrix $\mathbf{C}$ and when the ordering changes are small as is often the case in SLAM. In the best cases tested, recovering the Cholesky factors with the proposed method was over 75 times faster than recomputing the Cholesky decomposition, thus illustrating that significant improvement can be gained if the permutations are localised. A threshold was also provided to select between Cholesky and the factor recovery method presented.

The Hybrid Cholesky method was added to the SLAM++ software and results are obtained for nine popular 2D SLAM datasets. It has been found that using the proposed Hybrid method to calculate Cholesky factors during full reordering steps can be significantly faster than regular Cholesky decomposition.

Future research could explore calculating cost functions efficiently to increase the precision of the threshold obtained and reduce the cost of the overhead. The generalisation of adjacent columns swapping to displacement of arbitrary length in the elimination tree is also desired and careful analysis of the sorting efficiency and displacement cost would allow more efficient sorting algorithms to be used in the proposed Hybrid Cholesky decomposition. The use of an ordering algorithm taking into account geographical proximity of nodes is expected to affect the hybrid method favourably. To investigate this avenue, elimination tree rotations will be performed on COLAMD orderings to obtain an equivalent ordering that also minimises the PKT distance, which will be compared with orderings obtained from METIS [32].

References

1. Adams, J.A., Cooper, J.L., Goodrich, M.A., Humphrey, C., Quigley, M., Buss, B.G., Morse, B.S.: Camera-equipped mini UAVs for wilderness search support: Task analysis and lessons from field trials. Journal of Field Robotics **25**(1-2) (2007)
2. Adams, S.M., Friedland, C.J.: A survey of unmanned aerial vehicle (UAV) usage for imagery collection in disaster research and management. In: 9th International Workshop on Remote Sensing for Disaster Response (2011)
3. Agarwal, P., Olson, E.: Variable reordering strategies for SLAM. In: IEEE/RSJ International Conference on Intelligent Robots and Systems, pp. 3844–3850 (2012)
4. Amestoy, P.R., Davis, T.A., Duff, I.S.: An approximate minimum degree ordering algorithm. SIAM Journal on Matrix Analysis and Applications **17**(4), 886–905 (1996)
5. Bailey, T., Durrant-Whyte, H.: Simultaneous localization and mapping (SLAM): Part II. IEEE Robotics & Automation Magazine **13**(3), 108–117 (2006)
6. Buluç, A., Meyerhenke, H., Safro, I., Sanders, P., Schulz, C.: Recent advances in graph partitioning. Preprint (2013)
7. Chen, Y., Davis, T.A., Hager, W.W., Rajamanickam, S.: Algorithm 887: CHOLMOD, supernodal sparse Cholesky factorization and update/downdate. ACM Transactions on Mathematical Software **35**(3), 22 (2008)
8. Cunningham, A., Indelman, V., Dellaert, F.: DDF-SAM 2.0: Consistent distributed smoothing and mapping. In: IEEE International Conference on Robotics and Automation, pp. 5220–5227 (2013)
9. Cunningham, A., Paluri, M., Dellaert, F.: DDF-SAM: Fully distributed SLAM using constrained factor graphs. In: IEEE/RSJ International Conference on Intelligent Robots and Systems, pp. 3025–3030 (2010)
10. Cunningham, A., Wurm, K.M., Burgard, W., Dellaert, F.: Fully distributed scalable smoothing and mapping with robust multi-robot data association. In: IEEE International Conference on Robotics and Automation, pp. 1093–1100 (2012)
11. Davis, T.A.: Direct methods for sparse linear systems, vol. 2. Siam (2006)
12. Davis, T.A., Gilbert, J.R., Larimore, S.I., Ng, E.G.: Algorithm 836: COLAMD, a column approximate minimum degree ordering algorithm. ACM Transactions on Mathematical Software **30**(3), 377–380 (2004)
13. Davis, T.A., Hager, W.W.: Modifying a sparse Cholesky factorization. SIAM Journal on Matrix Analysis and Applications **20**(3), 606–627 (1999)
14. Davis, T.A., Hager, W.W.: Multiple-rank modifications of a sparse Cholesky factorization. SIAM Journal on Matrix Analysis and Applications **22**(4), 997–1013 (2001)
15. Davis, T.A., Hager, W.W.: Row modifications of a sparse Cholesky factorization. SIAM Journal on Matrix Analysis and Applications **26**(3), 621–639 (2005)
16. Dellaert, F., Carlson, J., Ila, V., Ni, K., Thorpe, C.E.: Subgraph-preconditioned conjugate gradients for large scale SLAM. In: IEEE/RSJ International Conference on Intelligent Robots and Systems, pp. 2566–2571 (2010)

17. Dellaert, F., Kaess, M.: Square root SAM: Simultaneous localization and mapping via square root information smoothing. The International Journal of Robotics Research **25**(12), 1181–1203 (2006)
18. Durrant-Whyte, H., Bailey, T.: Simultaneous localization and mapping: Part I. Robotics & Automation Magazine **13**(2), 99–110 (2006)
19. Folkesson, J., Christensen, H.: Graphical SLAM - a self-correcting map. In: IEEE International Conference on Robotics and Automation, vol. 1, pp. 383–390 (2004)
20. George, A.: Nested dissection of a regular finite element mesh. SIAM Journal on Numerical Analysis **10**(2), 345–363 (1973)
21. George, A., Liu, J.W.: An optimal agorithm for symbolic factorization of symmetric matrices. SIAM Journal on Computing **9**(3), 583–593 (1980)
22. Gill, P.E., Golub, G.H., Murray, W., Saunders, M.A.: Methods for modifying matrix factorizations. Mathematics of Computation **28**(126), 505–535 (1974)
23. Golub, G.H., Van Loan, C.F.: Matrix computations, vol. 3. JHU Press (2012)
24. Grigori, L., Boman, E.G., Donfack, S., Davis, T.A.: Hypergraph-based unsymmetric nested dissection ordering for sparse LU factorization. SIAM Journal on Scientific Computing **32**(6), 3426–3446 (2010)
25. Hogg, J.D., Reid, J.K., Scott, J.A.: Design of a multicore sparse Cholesky factorization using DAGs. SIAM Journal on Scientific Computing **32**(6), 3627–3649 (2010)
26. Huang, G., Truax, R., Kaess, M., Leonard, J.J.: Unscented iSAM: A consistent incremental solution to cooperative localization and target tracking. In: IEEE European Conference on Mobile Robots, pp. 248–254 (2013)
27. Indelman, V., Williams, S., Kaess, M., Dellaert, F.: Information fusion in navigation systems via factor graph based incremental smoothing. Robotics and Autonomous Systems **61**(8), 721–738 (2013)
28. Kaess, M., Ila, V., Roberts, R., Dellaert, F.: The bayes tree: Enabling incremental reordering and fluid relinearization for online mapping. Tech. rep., DTIC Document (2010)
29. Kaess, M., Johannsson, H., Roberts, R., Ila, V., Leonard, J.J., Dellaert, F.: isam2: Incremental smoothing and mapping using the bayes tree. The International Journal of Robotics Research **31**(2), 216–235 (2012)
30. Kaess, M., Ranganathan, A., Dellaert, F.: iSAM: Incremental smoothing and mapping. IEEE Transactions on Robotics **24**(6), 1365–1378 (2008)
31. Kang, S., Lee, W., Nam, M., Tsubouchi, T., Yuta, S.: Wheeled blimp: Hybrid structured airship with passive wheel mechanism for tele-guidance applications. In: IEEE International Conference on Intelligent Robots and Systems, vol. 4, pp. 3552–3557 (2003)
32. Karypis, G., Kumar, V.: Metis-unstructured graph partitioning and sparse matrix ordering system, version 2.0 (1995)
33. Kim, B., Kaess, M., Fletcher, L., Leonard, J., Bachrach, A., Roy, N., Teller, S.: Multiple relative pose graphs for robust cooperative mapping. In: IEEE International Conference on Robotics and Automation, pp. 3185–3192 (2010)
34. Konolige, K., Grisetti, G., Kummerle, R., Burgard, W., Limketkai, B., Vincent, R.: Efficient sparse pose adjustment for 2D mapping. In: IEEE/RSJ International Conference on Intelligent Robots and Systems, pp. 22–29 (2010)
35. Kümmerle, R., Steder, B., Dornhege, C., Ruhnke, M., Grisetti, G., Stachniss, C., Kleiner, A.: Slam benchmarking (2015). URL http://kaspar.informatik.uni-freiburg.de/ slamEvaluation/index.php
36. Küummerle, R., Grisetti, G., Strasdat, H., Konolige, K., Burgard, W.: g 2 o: A general framework for graph optimization. In: IEEE International Conference on Robotics and Automation, pp. 3607–3613 (2011)
37. LaSalle, D., Karypis, G.: Multi-threaded graph partitioning. In: IEEE International Symposium on Parallel & Distributed Processing, pp. 225–236 (2013)
38. Liu, J.W.: The role of elimination trees in sparse factorization. SIAM Journal on Matrix Analysis and Applications **11**(1), 134–172 (1990)
39. Ni, K., Dellaert, F.: Multi-level submap based SLAM using nested dissection. In: IEEE International Conference on Intelligent Robots and Systems, pp. 2558–2565 (2010)
40. Ni, K., Steedly, D., Dellaert, F.: Tectonic SAM: Exact, out-of-core, submap-based SLAM. In: IEEE International Conference on Robotics and Automation, pp. 1678–1685 (2007)
41. Polok, L., Solony, M., Ila, V., Smrz, P., Zemcik, P.: Efficient implementation for block matrix operations for nonlinear least squares problems in robotic applications. In: IEEE International Conference on Robotics and Automation, pp. 2263–2269 (2013)

42. Polok, L., Solony, M., Ila, V., Smrz, P., Zemcik, P.: Incremental Cholesky factorization for least squares problems in robotics. In: Intelligent Autonomous Vehicles, vol. 8, pp. 172–178 (2013)
43. Polok, L., Viorela, I.: Slam++ (2015). URL http://sourceforge.net/projects/slam-plus-plus/
44. Reid, J.K., Scott, J.A.: An out-of-core sparse Cholesky solver. ACM Transactions on Mathematical Software **36**(2), 9 (2009)
45. Rennich, S.C., Stosic, D., Davis, T.A.: Accelerating sparse Cholesky factorization on GPUs. In: Proceedings of the Fourth Workshop on Irregular Applications: Architectures and Algorithms, pp. 9–16 (2014)
46. Sherman, A.H.: On the efficient solution of sparse systems of linear and nonlinear equations. Ph.D. thesis, Yale. (1975)
47. Tinney, W.F., Walker, J.: Direct solutions of sparse network equations by optimally ordered triangular factorization. Proceedings of the IEEE **55**(11), 1801–1809 (1967)
48. Tsourakakis, C., Gkantsidis, C., Radunovic, B., Vojnovic, M.: FENNEL: Streaming graph partitioning for massive scale graphs. In: Proceedings of the 7th ACM international conference on Web search and data mining, pp. 333–342. ACM (2014)
49. Vaquero, L., Cuadrado, F., Logothetis, D., Martella, C.: xDGP: A dynamic graph processing system with adaptive partitioning. ACM Computing Research Repository (2013)
50. Williams, S., Indelman, V., Kaess, M., Roberts, R., Leonard, J.J., Dellaert, F.: Concurrent filtering and smoothing. In: IEEE International Conference on Information Fusion, pp. 1300–1307 (2012)
51. Wilson, J.: A new era for airships. Aerospace America **42**(5), 27–31 (2004)
52. Yannakakis, M.: Computing the mininimum fill-in is NP-complete. SIAM Journal on Algebraic and Discrete Methods **2**(1) (1981)
53. Zou, D., Dou, Y.: Implementation of parallel sparse Cholesky factorization on GPU. In: International Conference on Computer Science and Network Technology, pp. 2228–2232 (2012)