2015 IEEE International Conference on Robotics and Automation (ICRA)
Washington State Convention Center
Seattle, Washington, May 26-30, 2015

Fast Covariance Recovery in Incremental Nonlinear Least Square Solvers

Viorela Ila[1] , Lukas Polok[2], Marek Solony[2], Pavel Smrz[2] and Pavel Zemcik[2]

***Abstract*— Many estimation problems in robotics rely on efficiently solving nonlinear least squares (NLS). For example, it is well known that the simultaneous localisation and mapping (SLAM) problem can be formulated as a maximum likelihood estimation (MLE) and solved using NLS, yielding a mean state vector. However, for many applications recovering only the mean vector is not enough. Data association, active decisions, next best view, are only few of the applications that require fast state covariance recovery. The problem is not simple since, in general, the covariance is obtained by inverting the system matrix and the result is dense.**

The main contribution of this paper is a novel algorithm for fast incremental covariance update, complemented by a highly efficient implementation of the covariance recovery. This combination yields to two orders of magnitude reduction in computation time, compared to the other state of the art solutions. The proposed algorithm is applicable to any NLS solver implementation, and does not depend on incremental strategies described in our previous papers, which are not a subject of this paper.

I. Introduction

Probabilistic methods have been extensively applied in robotics and computer vision to handle noisy perception of the environment and the inherent uncertainty in the estimation. There are a variety of solutions to the estimation problems in today's literature. Filtering and maximum likelihood estimation (MLE) are among the most used in robotics. Since filtering easily becomes inconsistent when applied to nonlinear processes, MLE gained a prime role among the estimation solutions. In simultaneous localisation and mapping (SLAM) [1], [2], [3], [4] or other mathematical equivalent problems such as bundle adjustment (BA) [5], [6] or structure from motion (SFM) [7], the estimation problem finds the MLE of a set of variables (e.g. camera/robot poses and 3D points in the environment) given a set of observations. Assuming Gaussian noises and processes, the MLE have an elegant nonlinear least squares (NLS) solution.

A major challenge appears in online robotic applications, where the state changes every step. For very large problems, updating and solving the system every step may become very expensive. Efficient incremental NLS solutions have been developed, either by working directly on the matrix factorization of the linearised system [2], by using graphical model-based data structures such as the Bayes tree [4], or by exploiting the sparse block structure of the problems [8].

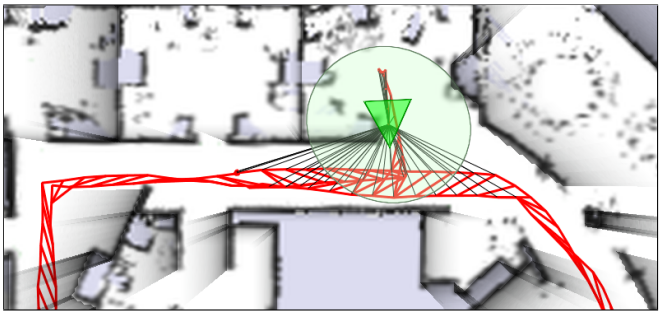


Fig. 1. Distance-based candidates for data association calculated using the marginal covariances (95% confidence interval shown in green).

The existing incremental NLS solutions provide fast and accurate estimations of the mean state vector, for example the mean position of the robot and features in the environment. However, in a real applications, the uncertainty of the estimation plays an important role. This is given by the covariance matrix, which generalizes the notion of variance to multiple dimensions. In particular, the marginal covariances, that encode the uncertainties between a subset of variables, are required in many applications.

Data association is the problem of associating current observations with previous ones, and it is the key to reduce the uncertainty in SLAM. Finding those associations becomes very expensive for large problems, nevertheless it can be simplified when the uncertainties of the estimates are known. Joint-compatibility tests in the case of landmark SLAM [9], [10] or estimation of possible relative displacement between poses in pose SLAM [11] are all based on recovering the marginal covariances. Figure 1 shows how the data association problem can be restricted to only a small set of sensor registration indicated by the grey links between the current pose of the robot and close poses already visited.

Information theoretic measures, such as mutual information, are also computed using the marginal covariances. This allows for principled ways to reduce the complexity of the SLAM problem by selecting only the informative measurements [11] or to plan reliable paths with the least probability of becoming lost [12]. In computer vision, the mutual information is used in online systems to compute the most appropriate actions for feature selection [13] or in *active vision* to guide efficient tracking and image processing. It is

*The research leading to these results has received funding from the EU, 7^{th} FP grant 316564-IMPART and the IT4Innovations Centre of Excellence, grant n. CZ.1.05/1.1.00/02.0070, supported Operational Programme Research and Development for Innovations funded by Structural Funds of the European Union and the state budget of the Czech Republic. Viorela Ila's work was supported by Australian Research Council under ARC Centres of Excellence program.

[1]V. Ila is with the Australian National University, Canberra, Australia. viorela.ila@anu.edu.au

[2]L. Polok, M. Solony, P. Smrz and P. Zemcik are with the Faculty of Information Technology, Brno University of Technology, Czech Republic. {ipolok, isolony, smrz, zemcik}@fit.vutbr.cz

also used in reducing the uncertainty in real-time monocular SLAM [14] and in *active matching* [15] of image feature. A problem related to active vision is the *next best view* for 3D reconstruction where the trace of the camera covariance matrix is used to select the images that will reduce the uncertainty in the reconstruction [16].

While easy to calculate in an information filter estimation framework [11] or even explicit in Kalman filtering, the marginal covariances are expensive to obtain in an NLS framework. Some of the applications, such as *graph sparsification* in the context of graph SLAM optimisation [17], [18], are based on approximations of the marginal covariances, in particular on local Chow-Liu trees [19]. In general, these approximations do not guarantee consistency, therefore recent methods opted for exact solutions to marginal covariance calculation [20], [21], such as the one introduced in [10].

This paper proposes a novel technique for obtaining the marginal covariances in an online NLS framework, where the system changes all the time. It is based on incremental updates of marginal covariances every time new variables and observations are integrated into the system, and on the fact that, in practice, when the linearisation point changes, the changes in the system are often so small that they can be ignored. The proposed methods are accompanied by a very efficient blockwise implementation. Extensive tests on large online NLS problems have proven that the new strategy significantly outperforms all the other state of the art exact solutions, without compromising the accuracy.

II. Incremental estimation

In this paper, the problem is formulated as a maximum likelihood estimation of a set of variables $\boldsymbol{\theta}$ given a set of observations $\mathbf{z}$. The SLAM example is considered, where the vector $\boldsymbol{\theta} = [\theta_1 \dots \theta_n]$ gathers the variables corresponding to the robot poses and the map, and the vector $\mathbf{z} = [z_1 \dots z_n]$ gathers the available observations. This estimation has to be done incrementally in an online application; every step a new variable and the associated measurements are integrated into the system and a new solution is calculated. In this section we briefly show how the MLE problem is formulated and solved.

A. State estimation

The goal is to obtain, at every step, the maximum likelihood estimate (MLE) of a set of variables in $\boldsymbol{\theta}$ given the available observations in $\mathbf{z}$:

$$\boldsymbol{\theta}^* = \underset{\boldsymbol{\theta}}{\operatorname{argmax}}\ P(\boldsymbol{\theta} \mid \mathbf{z}) = \underset{\boldsymbol{\theta}}{\operatorname{argmin}}\ \{-\log(P(\boldsymbol{\theta} \mid \mathbf{z})\}\ . \quad (1)$$

It is well known that, assuming Gaussian distributed processes and measurements, the MLE has an elegant and accurate solution based on solving a NLS problem:

$$\boldsymbol{\theta}^* = \underset{\boldsymbol{\theta}}{\operatorname{argmin}} \left\{ \frac{1}{2} \sum_{k=1}^{m} \|h_k(\theta_{i_k}, \theta_{j_k}) - z_k\|^2_{\Sigma_k} \right\}\ , \quad (2)$$

where $h(\theta_{i_k}, \theta_{j_k})$ is the nonlinear measurement function and z_k are the normally distributed measurements with the covariance Σ_k. Iterative methods, such as Gauss-Newton or Levenberg-Marquardt, are often used to solve the NLS in (2). This is usually addressed by solving the sequence of linear systems at every iteration. Linear approximations of the nonlinear residual functions around the current linearisation point $\boldsymbol{\theta}^i$ are calculated:

$$\tilde{\mathbf{r}}(\boldsymbol{\theta}^i) = \mathbf{r}(\boldsymbol{\theta}^i) + J(\boldsymbol{\theta}^i)(\boldsymbol{\theta} - \boldsymbol{\theta}^i)\ , \quad (3)$$

with $\mathbf{r}(\boldsymbol{\theta}) = [r_1, \dots, r_m]^\top$ being a vector gathering all nonlinear residuals of the type $r_k = h_k(\theta_{i_k}, \theta_{j_k}) - z_k$ and J being the Jacobian matrix which gathers the derivatives of the components of $\mathbf{r}(\boldsymbol{\theta})$. With this, the NLS in (2) is approximated by a linear one and solved by successive iterations:

$$\boldsymbol{\delta}^* = \underset{\boldsymbol{\delta}}{\operatorname{argmin}} \frac{1}{2} \|A\,\boldsymbol{\delta} - \mathbf{b}\|^2\ , \quad (4)$$

where the matrix A and the vector $\mathbf{b}$ are defined as $A \triangleq D^{-1/2}J$ and $\mathbf{b} \triangleq -D^{-1/2}\mathbf{r}$, with D gathering all the Σ_k measurement covariances [1] . The correction $\boldsymbol{\delta} \triangleq \boldsymbol{\theta} - \boldsymbol{\theta}^i$ towards the solution is obtained by solving the linear system:

$$A^\top A\,\boldsymbol{\delta} = A^\top \mathbf{b}\,, \quad \text{or} \quad \Lambda\boldsymbol{\delta} = \boldsymbol{\eta}\,, \quad (5)$$

with Λ, the square symmetric positive definite system matrix and $\boldsymbol{\eta}$ the right hand side. In the case of sparse problems such as SLAM, it is common to apply sparse matrix factorization, followed by backsubstitutions to obtain the solution of the linear system. The Cholesky factorization of the matrix Λ has the form $R^\top R = \Lambda$, where R is an upper triangular matrix. The forward and back-substitution on $R^\top \mathbf{d} = \boldsymbol{\eta}$ and $R\boldsymbol{\delta} = \mathbf{d}$ first recover $\mathbf{d}$, then the actual solution $\boldsymbol{\delta}$. After computing $\boldsymbol{\delta}$, the new linearisation point becomes $\boldsymbol{\theta}^{i+1} = \boldsymbol{\theta}^i + \boldsymbol{\delta}$. The nonlinear solver iterates until the norm of the correction becomes smaller than a tolerance.

B. Block structure

In many estimation problems, the random variables have more than one degree of freedom (DOF). For example, in a two-dimensional SLAM, every pose has 3 DOF and every landmark has 2 DOF; a $3D$-SLAM has 6 DOF variables. The associated system matrix can be interpreted as partitioned into sections corresponding to each variable, called *blocks*, which can be manipulated at once. If the number of variables is n, the size of the corresponding Λ matrix is $N \times N$, where N sums the products of the number of variables of each type and their DOF.

Correct manipulation of the *block matrices* enabled very efficient NLS and incremental NLS solutions [22], [8], which outperformed other similar state-of-the-art implementations, without affecting the precision in any way. In this paper, we will show that, based on the previously proposed block-based data structure in [22], we can also efficiently recover the marginal covariance matrices incrementally.

C. State augmentation and update

For large online problems, updating and solving at every step can become very expensive. In our previous work [8], we proposed an efficient algorithm and its implementation

to incrementally update and solve the NLS problem in (2). At every step, a new variable and the corresponding observations are integrated into the system. For each new observation, the matrix A is augmented with a block-row A_k. In general, just few variables are involved. Therefore, every new row of the matrix A is very sparse. For observations $h_k(\theta_i, \theta_j)$, involving two variables, the update of A becomes:

$$\hat{A} = \begin{bmatrix} A \\ A_k \end{bmatrix}, \text{with } A_k = \begin{bmatrix} 0 \dots J_i^j \Sigma_k^{-1/2} \dots 0 \dots \ J_j^i \Sigma_k^{-1/2} \end{bmatrix}. \quad (6)$$

This translates into additive updates of the system matrix Λ:

$$\hat{\Lambda} = \Lambda + A_u^\top A_u \, . \quad (7)$$

In [8] we showed that, based on this, one can determine which part of the R factorization changes. Furthermore, an incremental block Cholesky factorization was introduced, where the R factor and the r.h.s. vector $\mathbf{d}$ are efficiently maintained.

When updating the factor R, two situations can be distinguished; a) when the linearisation point does not change and b) when the linearisation point changes, the first being the one of interest in this paper. Theoretically, the linearisation point should change every iteration of the nonlinear system solver. In practice, though, the changes are so small that they can be ignored most of the time. This allowed for fast algorithms to solve the incremental SLAM problem [2], [4], [8]. When the changes in the linearisation point are substantial, the two matrices, Λ and R need to be recalculated, computing the Jacobians using the new linearisation point. Sometimes, these updates can be partial, only a few variables having important changes [4], and in this case, Λ and R can be updated partially as well [8], speeding up the update process.

III. Covariance recovery

When using MLE in real, online applications, the recovery of the uncertainty of the estimate, *the covariance*, can become a computational bottleneck. The covariance is needed, for example, to generate data association hypotheses, to evaluate the mutual information required in active mapping or graph sparsification, etc. The calculation of the covariance amounts to inverting the information matrix, $\Sigma = \Lambda^{-1}$, where the resulting matrix Σ is no longer sparse.

Several approximations for marginal covariance recovery have been proposed in the literature. Thrun et al. [23] suggested using conditional covariances, which are inversions of sub-blocks of Λ called the Markov blankets. The result is an overconfident approximation of the marginal covariances. Online, conservative approximations were proposed in [24], where at every step, the covariances corresponding to the new variables are computed by solving the augmented system with a set of basis vectors. The recovered covariance column is passed to a Kalman filter bank, which updates the rest of the covariance matrix. The filtering is reported to run in constant time, and the recovery speed is bounded by the linear solving. In the context of MLE, belief propagation over a spanning tree or loopy intersection propagation [25] can be used to obtain conservative approximations suitable for data association.

An exact method for sparse covariance recovery was proposed in [10]. It is based on a recursive formula [26], [27], which calculates any covariance elements on demand from other covariance elements and elements of the R factor. Therefore, a hash map is needed for fast dependence tracking. The method, though, does not benefit from the incremental nature of the online problem.

The authors of [28] proposed a covariance factorization for calculating linearized updates to the covariance matrix over arbitrary number of planning decision steps in a partially observable Markov decision process (POMDP). The method uses matrix inversion lemmas to efficiently calculate the updates. The idea of using factorizations for calculating inversion update is not new, though. A discussion of applications of the Sherman-Morrison and Woodbury formulas is presented in [29]. Specifically, it states the usefulness of these formula for updating the matrix inversion after small-rank modifications, where the rank is kept low enough to allow faster updates than actually calculating the inverse. In this section we propose a new update strategy which confirms this conclusion, but it has more practical application.

A. Recursive formula for covariance matrix calculation

Operating on dense matrices is unwanted, especially in the case of large size matrix such as Σ. Nevertheless, most of the applications require only a few elements of the covariance matrix, eliminating the need of recovering the full Σ. In general, the elements of interest are the block diagonal and the block column corresponding to the last pose. Some other applications only require a few block diagonal and off-diagonal block elements. In [10] was shown how specific elements from the covariance matrix can be efficiently calculated from the R factor by applying the recursive formula:

$$\Sigma_{ii} = \frac{1}{R_{ii}} \left[\frac{1}{R_{ii}} - \sum_{k=i+1, R_{ik} \neq 0}^{n} R_{ik} \Sigma_{ki} \right] , \quad (8)$$

$$\Sigma_{ij} = \frac{1}{R_{ii}} \left[- \sum_{k=i+1, R_{ik} \neq 0}^{j} R_{ik} \Sigma_{kj} - \sum_{k=j+1, R_{ik} \neq 0}^{n} R_{ik} \Sigma_{jk} \right] . \quad (9)$$

In case that R is sparse, the formulas above can be used to compute the elements of Σ at the positions of nonzero elements in R very efficiently [26]. To compute multiple elements of the covariance matrix, such as the whole block diagonal, these formulas becomes efficient unless all the intermediate results are stored. Figure 2[1] shows which elements need to be calculated for a specific block diagonal element.

B. Incremental update of the covariance matrix

In II-C, we mentioned that most of the algorithmic speedups can be applied in case the linearisation point is kept the same. Then, the contribution of every new measurement

[1] An insightful animation of the covariance recovery is available online at `http://slam-plus-plus.sourceforge.net/cov/`

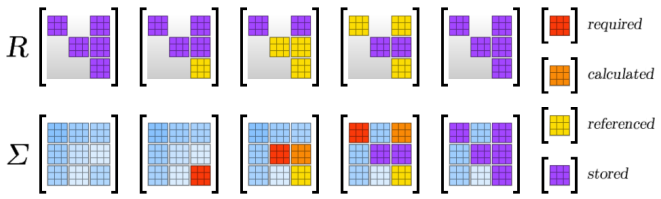


Fig. 2. Recovering the diagonal of the covariance matrix using the recursive formula.

can be easily integrated into the current system matrix Λ by a simple addition (see (7)), but things get complicated when the compute of the covariance is required:

$$\hat{\Sigma} \quad = \quad (\Lambda + A_u^\top A_u)^{-1} \,. \tag{10}$$

By applying the Woodbury formula, the above inverse can be written in terms of the previous covariance matrix:

$$\begin{aligned} \hat{\Sigma} &= \Lambda^{-1} - \Lambda^{-1} A_u^\top (I + A_u \Lambda^{-1} A_u^\top)^{-1} A_u \Lambda^{-1} \,, \\ \hat{\Sigma} &= \Sigma - \Sigma A_u^\top (I + A_u \Sigma A_u^\top)^{-1} A_u \Sigma \,. \end{aligned} \tag{11}$$

This shows that, in contrast to the information matrix which is additive, the covariance is subtractive:

$$\hat{\Sigma} = \Sigma + \Delta\Sigma, \quad \Delta\Sigma = -\Sigma A_u^\top (I + A_u \Sigma A_u^\top)^{-1} A_u \Sigma. \tag{12}$$

In SLAM, for example, this is easy to understand: a new measurement adds information to the system and reduces the uncertainty. It is important to mention that the size of the matrix to be inverted, $S \triangleq I + A_u \Sigma A_u^\top$, is very small compared to the system size. More precisely, the size of S, $(M_u \times M_u)$ with $(M_u \ll M)$, corresponds to the measurements involved in the update. For the simple case of a single measurement of a given DOF, $M_u = DOF$, regardless of the number of variables involved or their respective DOF. Furthermore, due to the fact that A_u is very sparse, the computation of S can be performed very efficiently. The complex update in (12) becomes a simple block vector multiplication:

$$\Delta\Sigma = -B\, S^{-1}\, B^\top \,, \quad \text{where} \quad B = \Sigma\, A_u^\top \,. \tag{13}$$

Due to the sparsity of the A_u, only few elements of the full Σ matrix are referenced, in particular only the block rows corresponding to the variables involved in the update. A simple example where the update involves two variables, is shown in Fig. 3[1]. Furthermore, the size of B is $(N \times M_u)$, but the product in (13) is a full matrix. Therefore the computation of the entire $\Delta\Sigma$ is prohibitive. We mentioned above that only some elements of the covariance are needed in the applications. For a single block, $\hat{\Sigma}_{ij}$, the update can be easily calculated as:

$$\Delta\Sigma_{ij} = -B_i\, S^{-1}\, B_j^\top \,, \tag{14}$$

where B_i and B_j are block rows of B of size of the update and the DOF of the variables i and j ($N_i \times M_u$ and $M_u \times N_j$, respectively). A similar formulation of the covariance update was used in [11] in the context of filtering SLAM. In there, the marginal covariance of the variables were used to facilitate data association and graph sparsification using information theory measures.

The storage of the dense matrix Σ must be avoided. Only the blocks requred by the application (for instance only the diagonal of Σ) are stored in a sparse block matrix. However, in order to compute the update in (13) or in (14), other elements of Σ are needed (the block columns, corresponding to the variables $\mathbf{v}$, involved in the update). Those are obtained by solving the system:

$$\Lambda\Sigma_{\mathbf{v}} = I_{\mathbf{v}} \quad \text{or} \quad R\,\Sigma_{\mathbf{v}} = R^{-\top} I_{\mathbf{v}} \,, \tag{15}$$

where $I_{\mathbf{v}}$ is a sparse block column matrix with only a few identity blocks at the positions corresponding to the variables involved in the update. The complexity of this calculation is directly proportional to the sum of DOF of the variables, involved in the update. For sparse R with n_{nz} nonzero elements, calculating a single column of $\Sigma_{\mathbf{v}}$ by forward and back substitution amounts to $O(2n_{nz})$.

C. Incremental downdate of the covariance matrix

Although very attractive, updating Σ as shown above sometimes becomes impractical to implement. In general, the covariances are calculated periodically, after the system was updated, which happens after one or several steps. In this case the Λ or R are not available anymore, as they were replaced by $\hat{\Lambda}$ and $\hat{R}$, respectively. Similarly to (10), one can downdate $\hat{\Lambda}$ to obtain Σ:

$$\Sigma \quad = \quad (\hat{\Lambda} - A_u^\top A_u)^{-1} \,. \tag{16}$$

Following the same scheme as in (11), $\Delta\Sigma$ can be now written in terms of $\hat{\Sigma}$:

$$\Delta\Sigma = \hat{\Sigma}\, A_u^\top (I - A_u\, \hat{\Sigma} A_u^\top)^{-1} A_u\, \hat{\Sigma} \,. \tag{17}$$

Defining $U \triangleq I - A_u\, \hat{\Sigma} A_u^\top$, which is a small size matrix similar to S, (17) becomes:

$$\Delta\Sigma = \hat{B} U^{-1} \hat{B}^\top \,, \quad \text{with} \quad \hat{B} = \hat{\Sigma}\, A_u^\top \,, \tag{18}$$

with $\hat{\Sigma}_{\mathbf{v}}$ obtained from solving $\hat{\Lambda}\hat{\Sigma}_{\mathbf{v}}^\top = I_{\mathbf{v}}$ or $\hat{R}\,\hat{\Sigma}_{\mathbf{v}} = \hat{R}^{-\top} I_{\mathbf{v}}$, much like in (15). This allows us to update the covariance at any step from the current $\hat{\Lambda}$ or $\hat{R}$ and a small A_u, instead of having to bookkeep the much larger Λ or R. Also, it is not mandatory to upate Σ at each step: to perform update to Σ over several steps, A_u will simply contain all the measurements since Σ was calculated.

IV. The Algorithm

This section proposes an efficient algorithm for online recovery of the marginal covariances. Based on whether or not the linearisation point changed, the algorithm has two branches: a) calculates sparse elements of the covariance matrix using the recursive formula (8) and (9), and b) updates sparse elements of the covariance using the covariance downdate in (18). The decision is outlined in Algorithm 1. Note that this algorithm also involves a simple *incremental* Gauss-Newton solver, but other nonlinear solvers, even batch solvers, are also suitable.

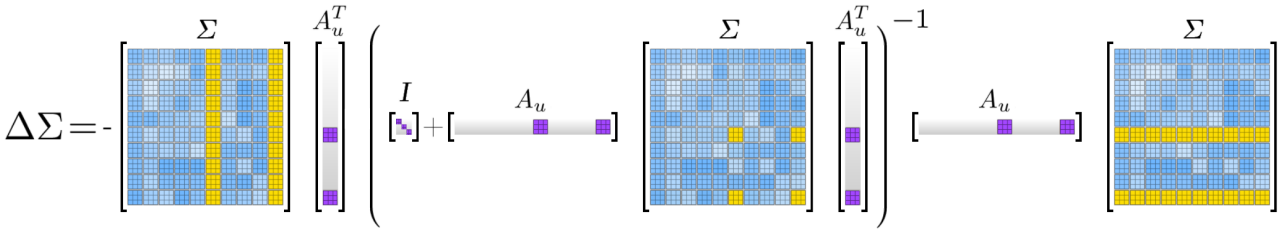


Fig. 3. Sparsity patterns involved in covariance update calculation (best viewed in color)

Algorithm 1 Covariance Recovery Algorithm Selection

1: **function** INCREMENTALGN($\boldsymbol{\theta}, \mathbf{v}, \mathbf{r}, \mathbf{z}_u, \Sigma_u, tol, it_{max}$)
2: $(\hat{\boldsymbol{\theta}}, \hat{\mathbf{r}}) = \text{UPDATE}(\boldsymbol{\theta}, \mathbf{v}, \mathbf{r}, \mathbf{z}_u, \Sigma_u)$
3: $(\hat{\Lambda}, \hat{\boldsymbol{\eta}}, A_u) = \text{LINEARSYSTEM}(\hat{\boldsymbol{\theta}}, \hat{\mathbf{r}})$
4: $changedLP = \text{FALSE}$
5: **for** $it = 0$ **to** it_{max} **do**
6: $\boldsymbol{\delta} = \text{SOLVE}(\hat{\Lambda}, \hat{\boldsymbol{\eta}})$
7: **if** $norm(\boldsymbol{\delta}) < tol$ **then**
8: **break**
9: **end if**
10: $\hat{\boldsymbol{\theta}} \leftarrow \hat{\boldsymbol{\theta}} \oplus \boldsymbol{\delta}$
11: $(\hat{\Lambda}, \hat{\boldsymbol{\eta}}) = \text{LINEARSYSTEM}(\hat{\boldsymbol{\theta}}, \hat{\mathbf{r}})$
12: $changedLP = \text{TRUE}$
13: **end for** ▷ incremental GN solver
14: $ordering = \text{AMD}(\hat{\Lambda})$
15: $\hat{R} = \text{CHOL}(\hat{\Lambda}, ordering)$
16: **if** $changedLP$ **then**
17: $\hat{\Sigma} = \text{CALCULATECOVARIANCE}(\hat{R}, ordering)$
18: **else**
19: $\hat{\Sigma} = \text{UPDATECOV}(\Sigma, \hat{R}, ordering, A_u, \mathbf{v})$
20: **end if**
21: **end function**

The two branches have different complexities. The first branch has complexity of $O(n_{nz}^2 N)$ in n_{nz}, the number of nonzeros of the $\hat{R}$ factor and N, the sum of DOF of all the vertices [26]. The second branch is dominated by the complexity of $O(n_{nz} N_u)$ where N_u is the sum of DOF of vertices that are being updated. As a result, the second branch is much faster if only a few vertices are changing. In case that most of the vertices are being updated (e.g. after a linearisation point change), the first branch becomes faster.

A. Sparse Blockwise Covariance Calculation

The proposed implementation of the recursive formula is slightly different from the other state of the art implementations [10] or [3]. These use a hash map or a similar structure for fast lookup of the elements of the covariance matrix that were already calculated in the course of evaluating (8), (9).

In contrast, the proposed algorithm calculates the covariance matrix column by column, right to left, calculating only the queried covariance elements and all the elements at the same place as the nonzero elements of the $\hat{R}$ factor. By the time the algorithm evaluates a specific element, it is guaranteed that all the references were already evaluated, eliminating the need for a hash map. A similar, but element-wise approach is described in [27]. The algorithm CALCULATECOVARIANCE was thus omitted to save space.

Algorithm 2 Incremental Covariance Update

1: **function** UPDATECOV(Σ, $\hat{R}$, $ordering$, A_u, $\mathbf{v}$)
Require: Σ is the covariance matrix to be updated
Require: $\hat{R}$ is Cholesky of $\hat{\Lambda}$ with fill-reducing ordering
Require: $ordering$ is fill-reducing ordering used in $\hat{R}$
Require: A_u is matrix of measurements since Σ
Require: $\mathbf{v}$ is list of vertices being updated
2: $I_{\mathbf{v}} = \text{EYE}(\text{SIZE}(\hat{R}))_{\mathbf{v}}$
3: $I_{\mathbf{v}} = \text{PERMUTE}(I_{\mathbf{v}}, ordering)$
4: $T = \hat{R} \backslash I_{\mathbf{v}}$
5: $T = \text{PERMUTE}(T, ordering^{-1})$
6: $Q = \left(T^\top\right)_{\mathbf{v}}$
7: $M = \text{DENSE}((A_u)_{\mathbf{v}})$
8: $U = \text{EYE}(\text{SIZE}(Q)) - MQM^\top$
9: $\hat{B} = T(1 : \text{ROWS}(\Sigma), :)\, M^\top$
10: $\hat{\Sigma} = \Sigma + \hat{B} U^{-1} \hat{B}^\top$
11: $ov = \text{COLUMNS}(\Sigma)$
12: $nv = \text{COLUMNS}(\hat{\Sigma}) - \text{COLUMNS}(\Sigma)$
13: $\hat{\Sigma}(:, ov : end) = T(:, \text{COLUMNS}(T) - nv : end)$
14: **return** $\hat{\Sigma}$
15: **end function**

Note that the resulting covariance matrix is sparse: the algorithm does not calculate more elements, than [10], [3].

Once finished, the proposed algorithm permutes the calculated covariance matrix to the natural order, so that the block columns and block rows of $\hat{\Sigma}$ correspond to the variables of the optimized system. The covariance matrix is symmetric, and only the upper-triangular part is stored.

B. Covariance Update

Updating the covariance incrementally is significantly faster in the second branch of the Algorithm 1. To calculate an update to the covariance matrix from the previous step, Algorithm 2 closely follows the calculation outlined in Section III. Note that reordering the system matrix ($\hat{\Lambda}$ or $\hat{R}$), e.g. as described in [8] does not impede the incremental update and that the algorithm is valid, but not efficient when the linearisation point changes.

The algorithm begins by evaluating T, the block columns of $\hat{\Sigma}$ corresponding to the vertices, $\mathbf{v}$, that are being updated (lines 2 to 5). This is illustrated in Fig. 3[1], where T comprises the highlighted columns of Σ on the left (or rows on the right, as Σ is symmetric). Note that the inverse fill-reducing ordering is applied so that the block rows of T correspond to the variables of the optimized system.

To calculate the small matrix U by directly following (17) would involve several sparse matrix products. In the

Dataset	iSAM	g2o	SLAM++	SLAM++ Total
Manhattan	206.58	180.42	**4.37**	13.88
10k	6712.03	5902.46	**179.69**	388.67
City10k	4585.15	3742.66	**55.87**	219.43
CityTrees10k	1009.91	938.97	**30.98**	60.41
Sphere	6051.73	5536.48	**24.64**	105.35
Intel	6.23	6.92	**0.54**	1.11
Killian	19.27	21.59	**1.43**	2.99
Victoria Park	310.57	293.09	**13.89**	37.11
Park Garage	237.13	216.28	**10.77**	27.08

TABLE I

TIME PERFORMANCE IN SECONDS OF THE COVARIANCE RECOVERY STRATEGIES ON MULTIPLE SLAM DATASETS.

proposed algorithm, dense calculations are used instead: the small portion of Σ used in the product is copied to a small dense matrix (Q on line 6, corresponding to the highlighted blocks of Σ in the center of figure 3[1]). Similarly, nonzero columns of A_u are copied to another dense matrix (M on line 7). The calculation of U on line 8 is then performed using only small dense matrices, enabling better cache coherency and acceleration using e.g. SSE instructions. The result is identical to the one obtained by the equivalent sparse calculation in (17).

Finally, the additive update of Σ to $\hat{\Sigma}$ is calculated on line 10. Note that it is not necessary to calculate full dense $\hat{B}U^{-1}\hat{B}^\top$. Instead, only the relevant blocks of Σ can be updated by using $\Delta\Sigma_{ij} = \hat{B}_i U^{-1} \hat{B}_j^\top$, in analogy to (14). In our implementation, this product is carried out in parallel. Some parts of $\hat{\Sigma}$ do not need to be updated, as they were already calculated using forward and backsubstitution (at lines 2 to 5). These are specifically the columns, corresponding to the vertices being updated. In Algorithm 2, this is used to *extend* $\hat{\Sigma}$ with covariances of the newly introduced vertices.

V. EXPERIMENTAL VALIDATION

In this work, we focused our efforts on testing the proposed algorithms on SLAM applications, but the applicability of the technique remains general. Many other applications from robotics such as active vision, planning in belief space etc. can benefit from the solution presented in this paper.

The computational efficiency and precision of the method and its implementation were tested and compared with similar state of the art implementations, in particular, iSAM [2] and g2o [3]. For iSAM v1.7 we used revision 10 and for g2o, svn revision 54. Both, iSAM and g2o use fairly similar implementation of the recursive formula (8), (9) together with a cache of already calculated covariances, based on STL hash map containers. Although highly efficient, these implementations do not handle incremental updates of the covariance. The proposed online covariance recovery will be included in SLAM ++[4], our block-efficient NLS framework, which is distributed under MIT open-source license. Other implementations can easily benefit from the proposed scheme. The only requirement on the solver is to be able to solve for dense columns of Σ and to have explicit A_u.

[4]http://sourceforge.net/p/slam-plus-plus/

The evaluation was performed on five simulated datasets; *Manhattan* [30], *10k* [31], *City10k* and *CityTrees10k* [32], *Sphere* [3] and four real datasets; *Intel* [33], *Killian Court* [34], *Victoria park* and *Parking Garage* [3] (see Tab. I). These are the datasets commonly used in evaluating NLS solutions to SLAM problems. The tests were performed on a computer with Intel Core $i5$ CPU 661 running at 3.33 GHz and 8 GB of RAM. This is a quad-core CPU without hyperthreading and with full SSE instruction set support. Each test was run ten times and the average time was calculated in order to avoid measurement errors, especially on smaller datasets.

A. Time evaluation

Table I shows the time performance of the incremental covariance recovery strategy in Algorithm 1 tested on the above-mentioned datasets and compared with the g2o and iSAM implementations of covariance recovery. The *block-diagonal* and the *last block-column* of the covariance matrix are calculated every step in all the cases. These are the only elements of the covariance matrix required for taking active decisions based on the current estimation and efficient search for data association in an online SLAM application [11]. In incremental mode, the covariance elements are calculated after each variable is added to the system (e.g. for the 10k dataset, it is calculated ten thousand times). The total time spent in solving the SLAM problem with covariance recovery is reported in the last column.

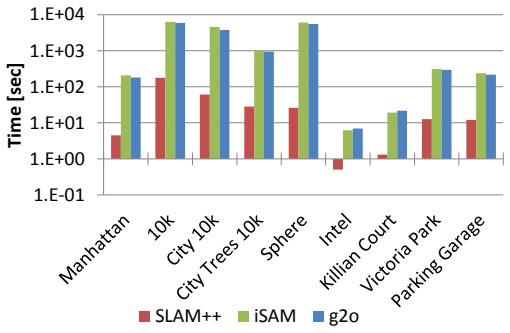
Figure 4 left reports the covariance recovery time on logarithmic scale while Fig.4 right shows the cumulative time of the incremental covariance computation on the *Intel* dataset during the execution of the algorithm. An approximate time complexity was estimated from these readings using least squares. The time complexity for SLAM ++ $O(n^{1.77})$ is superior to the ones of g2o $O(n^{2.31})$ and iSAM $O(n^{2.36})$.

The performance of our incremental NLS solver in [8] was also compared against GTSAM 2.3.1. However, the computation of the marginal covariances is not optimised for recovering all the block-diagonal elements in the current version of the GTSAM, therefore we excluded it from our comparisons. Nevertheless, we tested the available function for recovering the covariance of a single variable, the first variable (the most expensive one to calculate), against a similar function in SLAM ++, and this produced on *Manhattan*, 26.270 s GTSAM vs. 2.125 s SLAM ++, on *10k*, 261.880 s vs. 50.550 s, and on *Intel*, 1.429 s vs. 0.148 s.

In conclusion, the proposed implementation significantly outperforms all the existing implementations due to the proposed incremental covariance update algorithm and the blockwise implementation of the recursive formula.

B. Memory Usage Evaluation

Memory consumption of the above-mentioned implementations was also evaluated. The memory usage has been measured during two series of runs: with and without the marginal covariances computations. Figure 5 shows on the left the overall memory usage from experiments performed



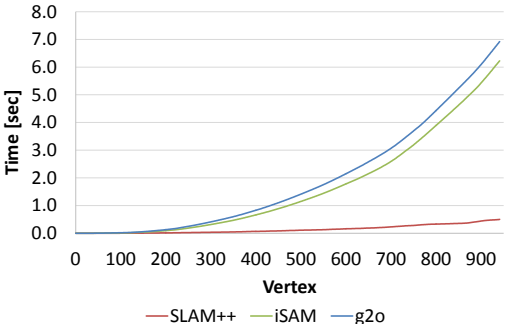
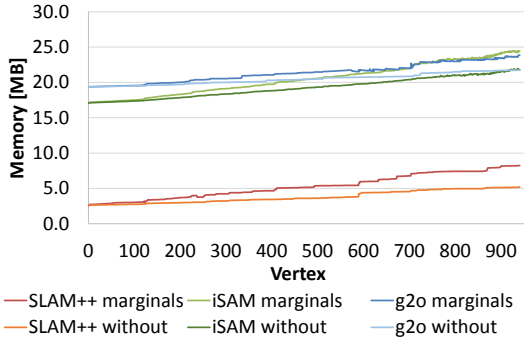


Fig. 4. Left: Logarithmic plot of time on standard datasets. Right: Marginal covariances computation cumulative times on *Intel* dataset.



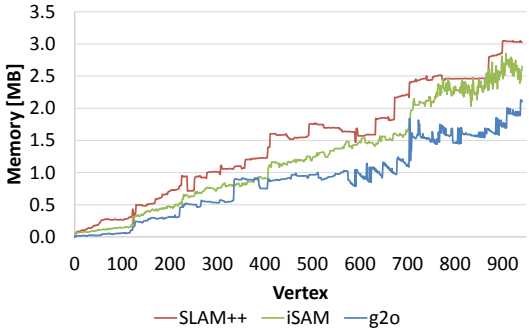


Fig. 5. Left: Overall memory usage with and without marginal covariances computation. Right: Memory allocation of marginals algorithm only.

on the *Intel* dataset and on the right the memory allocation of marginal covariances calculation only.

The overall memory usage plot shows that SLAM ++ uses the least memory, which is achieved thanks to efficient implementation of matrix storage. The evaluation of the memory used by marginal covariances computation algorithm is comparable to g2o and iSAM. SLAM ++ performs pooled memory allocation, which can be seen as steps in the plot. This is advantageous, compared to the noisy allocation patterns of g2o and iSAM, which probably lead to more system calls and thus higher execution time.

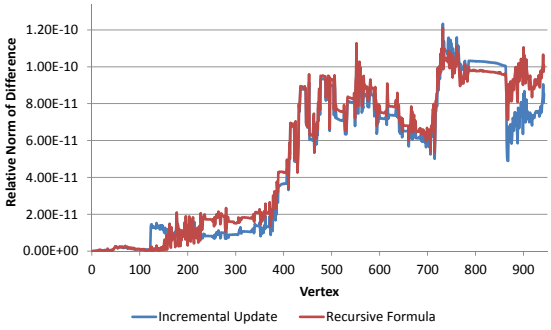


Fig. 6. Covariance Precision on Intel Dataset

C. Numerical Precision Evaluation

Since the proposed incremental update of the covariance is additive, it is likely that arithmetic errors in calculating the update will accumulate over consecutive steps, causing the solution to drift away from the correct values. Although no proof of numerical stability is offered here, we consider it is very important to show how the algorithm behaves in practice. A benchmark was performed on the *Intel* dataset, where the covariances were calculated using recursive formula, using the proposed method and using back and forward substitution to solve for a full inverse. The *Intel* dataset was chosen specifically because it contains just a handful of loop closures, causing the incremental covariance update to last for long periods, exceeding hundreds of steps.

Although being the slowest, backsubstitution was shown to be numerically *backward stable*. Therefore, the covariance calculated using back and forward substitution was used as a ground truth. The recursive formula in (8) and (9) is arguably less precise, as it reuses already calculated values of the covariance, potentially amplifying their error. The increment, $\Delta\Sigma$, is calculated from backsubstitution so it should be relatively precise, however the update is additive, allowing the error to slowly creep in. Figure 6 plots the relative norm of error of covariances, calculated using the recursive formula and using the incremental update. It can be seen that the errors are quite correlated, incremental update having mostly lower error. This is given by the fact that the covariance for incremental update is initialized using the recursive formula after a linearisation point change occurred. Generally, it shows that the covariance was calculated with error at 10^{th} decimal place and that using the incremental update slightly increased precision, rather than decreasing it.

VI. Conclusions and Future Work

In this paper, we introduced a novel method for incrementally updating the covariance in an NLS problem, which significantly speeds up the computation of the covariance matrices useful in a broad range of robotic applications. We

targeted problems which have a particular block structure, where the size of the blocks corresponds to the number of degrees of freedom of the variables. The advantage of the new scheme was demonstrated through an exhaustive comparison with the existing implementations on several available datasets. The tests show that the proposed scheme is not only about an order of magnitude faster, but also numerically stable. Error of the covariance calculated using the incremental update is, on average, lower than the error of the commonly used recursive formula.

In our multimedia attachment[1], we demonstrate the usefulness of the incremental covariances calculation in a data association context, where the number of expensive sensor registrations can be reduced by applying distance tests [11].

Even though the proposed algorithm proved to be significantly faster than other state of the art implementations, several improvements can still be applied. For example, when solving *bundle adjustment* type of problems, Schur complement is typically used to accelerate the matrix inversion. The incremental update can be adapted to work with the Schur complement. However, the recursive formula would require more attention. Finally, the block layout was designed with hardware acceleration in mind. Our SLAM ++ framework runs on embedded platforms, multi-core CPUs and steps were already taken to implement a GPU accelerated version.

REFERENCES

[1] F. Dellaert and M. Kaess, "Square Root SAM: Simultaneous localization and mapping via square root information smoothing," *Intl. J. of Robotics Research*, vol. 25, no. 12, pp. 1181–1203, Dec 2006.

[2] M. Kaess, A. Ranganathan, and F. Dellaert, "iSAM: Incremental smoothing and mapping," *IEEE Trans. Robotics*, vol. 24, no. 6, pp. 1365–1378, Dec 2008.

[3] R. Kümmerle, G. Grisetti, H. Strasdat, K. Konolige, and W. Burgard, "g2o: A general framework for graph optimization," in *Proc. of the IEEE Int. Conf. on Robotics and Automation (ICRA)*, Shanghai, China, May 2011.

[4] M. Kaess, H. Johannsson, R. Roberts, V. Ila, J. J. Leonard, and F. Dellaert, "iSAM2: Incremental smoothing and mapping using the Bayes tree," *Intl. J. of Robotics Research*, vol. 31, pp. 217–236, Feb. 2012.

[5] S. Agarwal, N. Snavely, I. Simon, S. M. Seitz, and R. Szeliski, "Building rome in a day," in *Twelfth IEEE International Conference on Computer Vision (ICCV 2009)*, Kyoto, Japan, 2009.

[6] K. Konolige, "Sparse sparse bundle adjustment," in *British Machine Vision Conference*, Aberystwyth, Wales, 08/2010 2010.

[7] C. Beall, B. Lawrence, V. Ila, and F. Dellaert, "3D Reconstruction of Underwater Structures," in *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, 2010.

[8] L. Polok, V. Ila, M. Solony, P. Smrz, and P. Zemcik, "Incremental block cholesky factorization for nonlinear least squares in robotics," in *Proceedings of the Robotics: Science and Systems 2013*, 2013.

[9] J. Neira and J. Tardós, "Data association in stochastic mapping using the joint compatibility test," *IEEE Trans. Robot. Automat.*, vol. 17, no. 6, pp. 890–897, December 2001.

[10] M. Kaess and F. Dellaert, "Covariance recovery from a square root information matrix for data association," *Robotics and Autonomous Systems*, 2009.

[11] V. Ila, J. M. Porta, and J. Andrade-Cetto, "Information-based compact Pose SLAM," *IEEE Trans. Robotics*, vol. 26, no. 1, pp. 78 – 93, 2010.

[12] R. Valencia, M. Morta, J. Andrade-Cetto, and J. Porta, "Planning reliable paths with pose slam," *IEEE Trans. Robotics*, vol. 29, no. 4, pp. 1050–1059, Aug 2013.

[13] A. Davison and D. Murray, "Simulataneous localization and map-building using active vision," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 24, no. 7, pp. 865–880, Jul 2002.

[14] T. Vidal-Calleja, A. Davison, J. Andrade-Cetto, and D. Murray, "Active control for single camera slam," in *Robotics and Automation, 2006. ICRA 2006. Proceedings 2006 IEEE International Conference on*, May 2006, pp. 1930–1936.

[15] A. Handa, M. Chli, H. Strasdat, and A. J. Davison, "Scalable active matching," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2010.

[16] S. Haner and A. Heyden, "Covariance propagation and next best view planning for 3d reconstruction," in *Eur. Conf. on Computer Vision (ECCV)*, Italy, October 2012, pp. 545–556.

[17] H. Kretzschmar, C. Stachniss, and G. Grisetti, "Efficient information-theoretic graph pruning for graph-based slam with laser range finders," in *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, 2011, pp. 865–871.

[18] N. Carlevaris-Bianco and R. M. Eustice, "Generic factor-based node marginalization and edge sparsification for pose-graph slam," in *Proceedings of the IEEE International Conference on Robotics and Automation*, Karlsruhe, May 2013, pp. 5728–5735.

[19] C. Chow and C. Liu, "Approximating discrete probability distributions with dependence trees," *Information Theory, IEEE Transactions on*, vol. 14, no. 3, pp. 462–467, May 1968.

[20] H. Johannsson, M. Kaess, M. Fallon, and J. Leonard, "Temporally scalable visual SLAM using a reduced pose graph," in *IEEE Intl. Conf. on Robotics and Automation, ICRA*, Karlsruhe, Germany, May 2013, best student paper finalist (one of five).

[21] G. Huang, M. Kaess, and J. Leonard, "Consistent sparsification for graph optimization," in *European Conference on Mobile Robots (ECMR)*, Barcelona, Spain, Sep 2013.

[22] L. Polok, M. Solony, V. Ila, P. Zemcik, and P. Smrz, "Efficient implementation for block matrix operations for nonlinear least squares problems in robotic applications," in *Proceedings of the IEEE International Conference on Robotics and Automation*. IEEE, 2013.

[23] S. Thrun, Y. Liu, D. Koller, A. Ng, Z. Ghahramani, and H. Durrant-Whyte, "Simultaneous localization and mapping with sparse extended information filters," *Intl. J. of Robotics Research*, vol. 23, no. 7-8, pp. 693–716, 2004.

[24] R. Eustice, H. Singh, J. Leonard, and M. Walter, "Visually mapping the RMS Titanic: Conservative covariance estimates for SLAM information filters," *Intl. J. of Robotics Research*, vol. 25, no. 12, pp. 1223–1242, Dec 2006.

[25] G. D. Tipaldi, G. Grisetti, and W. Burgard, "Approximate covariance estimation in graphical approaches to slam," in *Intelligent Robots and Systems, 2007. IROS 2007. IEEE/RSJ International Conference on*. IEEE, 2007, pp. 3460–3465.

[26] A. Björck, *Numerical methods for least squares problems*. SIAM, 1996.

[27] G. H. Golub and R. J. Plemmons, "Large-scale geodetic least-squares adjustment by dissection and orthogonal decomposition," *Linear Algebra and Its Applications*, vol. 34, pp. 3–28, 1980.

[28] S. Prentice and N. Roy, "The belief roadmap: Efficient planning in linear pomdps by factoring the covariance," in *Robotics Research*. Springer, 2011, pp. 293–305.

[29] W. W. Hager, "Updating the inverse of a matrix," *SIAM review*, vol. 31, no. 2, pp. 221–239, 1989.

[30] E. Olson, "Robust and efficient robot mapping," Ph.D. dissertation, Massachusetts Institute of Technology, 2008.

[31] 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)*, Jun 2007.

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

[33] A. Howard and N. Roy, "The robotics data set repository (Radish)," 2003. [Online]. Available: http://radish.sourceforge.net/

[34] 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, no. 12, pp. 1113–1139, Dec 2004.