Article

The International Journal of Robotics Research
2017, Vol. 36(2) 210–230

DOI: 10.1177/0278364917691110
journals.sagepub.com/home/ijr

SLAM++[1]-A highly efficient and temporally scalable incremental SLAM framework

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

Abstract

The most common way to deal with the uncertainty present in noisy sensorial perception and action is to model the problem with a probabilistic framework. Maximum likelihood estimation is a well-known estimation method used in many robotic and computer vision applications. Under Gaussian assumption, the maximum likelihood estimation converts to a nonlinear least squares problem. Efficient solutions to nonlinear least squares exist and they are based on iteratively solving sparse linear systems until convergence. In general, the existing solutions provide only an estimation of the mean state vector, the resulting covariance being computationally too expensive to recover. Nevertheless, in many simultaneous localization and mapping (SLAM) applications, knowing only the mean vector is not enough. Data association, obtaining reduced state representations, active decisions and next best view are only a few of the applications that require fast state covariance recovery. Furthermore, computer vision and robotic applications are in general performed online. In this case, the state is updated and recomputed every step and its size is continuously growing, therefore, the estimation process may become highly computationally demanding.

This paper introduces a general framework for incremental maximum likelihood estimation called SLAM++, which fully benefits from the incremental nature of the online applications, and provides efficient estimation of both the mean and the covariance of the estimate. Based on that, we propose a strategy for maintaining a sparse and scalable state representation for large scale mapping, which uses information theory measures to integrate only informative and non-redundant contributions to the state representation. SLAM++ differs from existing implementations by performing all the matrix operations by blocks. This led to extremely fast matrix manipulation and arithmetic operations used in nonlinear least squares. Even though this paper tests SLAM++ efficiency on SLAM problems, its applicability remains general.

Keywords

Nonlinear least squares, incremental covariance recovery, long-term SLAM, loop closure, compact state representation

1. 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 localization and mapping (SLAM) (Dellaert and Kaess, 2006; Kaess et al., 2008, 2011b; Kümmerle et al., 2011) or other mathematical equivalent problems such as bundle adjustment (BA) (Agarwal et al., 2009; Konolige, 2010) or structure from motion (SFM) (Beall et al., 2010), the estimation problem is solved by finding 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 has an elegant nonlinear least squares (NLS) solution.

A major challenge appears in online robotic applications, where the state changes at every step. For very large problems, updating the system and solving the NLS at every step can become very expensive. Efficient incremental NLS solutions have been developed, either by working directly on the matrix factorization of the linearized system (Kaess et al., 2008), by using graphical model-based data structures such as the Bayes tree (Kaess et al., 2010, 2011b) or by

[1]Australian National University, Canberra, Australia
[2]Faculty of Information Technology, Brno University of Technology, Brno, Czech Republic

Corresponding author:
Viorela Ila, 115 North Road, Canberra, Australian Capital Territory 2100, Australia.
Email: viorela.ila@anu.edu.au

exploiting the inherent sparse block structure (Polok et al., 2013b).

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

In online SLAM and SFM applications, the marginal covariances can be used to perform *data association* (Kaess and Dellaert, 2009; Neira and Tard os, 2001), to obtain a *reduced state representation* which will further allow efficiently handling large scale applications (Carlevaris-Bianco and Eustice, 2013; Huang et al., 2013; Ila et al., 2010; Johannsson et al., 2013; Kretzschmar et al., 2011), to perform *active sensing* (Davison and Murray, 2002; Haner and Heyden, 2012), to decide on the *next best actions* to take (Vidal–Calleja et al., 2006) or the *best reliable path* to follow (Valencia et al., 2013) to reduce the uncertainty of the estimate or simply to *provide feedback* about the error of the estimation.

A novel technique to obtain exact *marginal covariances* in an online NLS framework, where the system changes every step was proposed in Ila et al. (2015). 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 changes in the linearization point are often very small and can be ignored. The formulation and the implementation depart from the existing approaches by exploiting the sparsity and the block structure of the robotic problems. A new data-structure was introduced in our previous work, which enabled efficient block-matrix manipulation, block-wise arithmetic operations (Polok et al., 2013c), and block matrix factorization. Furthermore, a novel algorithm for incrementally solving NLS problems, with a new efficient incremental ordering scheme was proposed in Polok et al. (2013b) and extensively tested on well known datasets. The current work integrates the strategy for fast covariance recovery proposed in Ila et al. (2015) into the NLS solution introduced in Polok et al. (2013b).

Furthermore, based on the incremental solver with fast covariance recovery, this paper proposes an incremental algorithm which performs state-based data association and, at the same time, maintains a scalable representation of the state. This is achieved by computing two measures, the proximity in terms of sensor range of the current pose to any other previous poses the robot traveled, and the mutual information of each candidate link, which require good estimations of the marginal covariances. The proposed algorithm is an extension of the 2D filtering approach proposed by Ila et al. (2010) to the 3D MLE SLAM framework. Comparing to 2D filtering approach, several new challenges are met in compact 3D MLE SLAM. Careful handling of the thresholds in the rotation space, complex incremental covariance calculations, changes in the linearization point are among them.

This paper addresses all above mentioned challenges and proposes solutions which are implemented within the SLAM++ nonlinear least squares library. The name of the library was chosen to reflect the incremental nature of the solving. The early version of the library was introduced in Polok et al. (2013c) and proved to supersede the existing similar implementations. The current version differentiates from other similar libraries proposed in the SLAM community (g2o (Kümmerle et al., 2011), iSAM (Kaess et al., 2008) or iSAM2 (Kaess et al., 2011b)) in several ways.

1. It is based on a sparse, block data-structure which has been proven to be extremely efficient for problems with higher than two variable size (Polok et al., 2013a,c).
2. It integrates an efficient incremental, sparse, block Cholesky factorization (Polok et al., 2013b).
3. It integrates an incremental ordering which maintains a sparse matrix factorization, and in consequence allows for efficient solving (Polok et al., 2013b).
4. Allows for incremental covariance calculation which provides marginal covariances two orders of magnitude faster than previous implementations (Ila et al., 2015).
5. Integrates a compact pose SLAM algorithm where information theoretic measures are used to maintain a sparse, conservative representation of the MLE SLAM.
6. It comprises not only the C++ code but also the datasets and the scripts that can be used to reproduce the benchmarks presented in this paper, as well as scripts for automatic calculation of all involved thresholds in compact pose SLAM.

The above mentioned characteristics are validated in this paper through extensive tests of the time efficiency, accuracy and conservativeness of the estimation on several simulated and real datasets.

2. Related works

When using MLE in real applications such as online SLAM, 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, or to evaluate the mutual information required in active mapping or graph sparsification. The calculation of the covariance amounts to inverting the system 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. (2004) 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 Eustice et al. (2006), 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 can be used to obtain conservative approximations suitable for data association (Tipaldi et al., 2007).

An exact method for sparse covariance recovery was proposed in Kaess and Dellaert (2009). It is based on a recursive formula from Björck (1996) and Golub and Plemmons (1980), which calculates any covariance elements on demand from the other covariance elements and the elements of the Cholesky factor. It was implemented using a hash map, to provide for fast dependence tracking. The method, though, does not benefit from the incremental nature of the online problem.

In their paper, Prentice and Roy (2011) 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 Hager (1989). Specifically, it states the usefulness of these formulas for updating matrix inversion, after a small-rank modification, where the rank must be kept low enough in order for the update to be faster than simply calculating the inverse. In our latest work (Ila et al., 2015), we proposed an algorithm which confirms this conclusion, and also proves its usefulness in online SLAM applications.

When optimizing over the entire robot trajectory, the SLAM solution is more accurate but the computational complexity grows at every step and can become intractable for long runs. This can be alleviated by several techniques. One way is to optimize for a small window around the current pose, and only do expensive optimization when large loops are closed (Huang et al., 2011; Sibley et al., 2008). This technique assumes that the robot is running in open loop for long periods of time and that the loop closure detection is strictly based on appearance based sensors and has no information about the current estimates. Without prior information about the robot position and the topology of the map, appearance based methods can be easily tricked by perceptual aliasing (Ila et al., 2010). Other techniques consider only a subset of empirically selected key frames to perform global optimization, while the intermediate frames are referred to the optimized key frames (Klein and Murray, 2007). These techniques can reduce the run time, nevertheless they require more principled selection methods to automatically restrict the state estimation problem size to that of the area the robot operates in.

Recently, pose graph has received a lot of attention in the SLAM literature; when having a good estimation of the robot pose, the map can be retrieved by simply referring the relative measurements which can come from a large variety of sparse or dense sensors. Even if very efficient solutions to medium-size pose graph SLAM exist (Kaess et al., 2008; Kümmerle et al., 2011; Polok et al., 2013c), it can become inefficient for very long runs. The size of the graph grows in time and it is not bounded by the size of the environment. Apart from the node count, the sparsity of the graph also affects the performance. Therefore, current literature in SLAM proposes reduction strategies which involve both, node marginalization and edge sparsification. Eliminating a node through marginalization actually introduces more edges which densify the graph. Carlevaris–Bianco and Eustice (2013) initially proposed a technique which approximates the dense clique generated by the node removal by using the Chow-Liu tree (CLT) (Chow and Liu, 1968). Those approximations produce overconfident estimates, therefore, in their latter work, Carlevaris–Bianco and Eustice (2014b) proposed a sparse CLT approximations which result in conservative estimates. Conservative estimates are preferable in SLAM applications where data association is using the state estimate, or in applications where the map is used to plan a path to a goal point in the map. Carlevaris–Bianco and Eustice (2013) also pointed out the difference between composition and marginalization for the specific case where the pose to be removed is involved in a loop closure link. The strategy proposed in this paper automatically avoids this situation by always keeping the poses which are involved in potentially informative links.

Most of the existing graph pruning methods reduce the size of the graph in batch mode, after the graph was already built. Johannsson et al. (2013) proposes a technique to temporally scalable SLAM that decides on the fly which nodes are added to the graph. This is done by introducing the concept of active nodes and re-uses the already existing poses in the graph when the robot revisits previously mapped areas. Huang et al. (2013) considers the incremental nature of the SLAM problem and proposes a technique to keep consistent estimates by retaining all the information of the marginalized-out poses.

This paper shows how, based on the efficient covariance recovery strategy, a principled method that incrementally maintain a compact representation of a SLAM problem can be obtained. The system only keeps non-redundant poses and informative links. The result is a set of robot poses nicely distributed in the information space. This translates in highly scalable solutions and great speed-ups for large scale estimation problems, while the accuracy is marginally affected. The idea was previously introduced in a filtering framework (Ila et al., 2010) to maintain a compact representation of a 2D pose SLAM, and this paper extends

it to large scale MLE estimation for 3D SLAM, addressing all the corresponding challenges. The method can be easily extended beyond pose SLAM, to problems with different types of measurements, for example trinary factors present in structure-less BA (Indelman et al., 2012), or different types of variables, for example landmark SLAM, with the constrain that variables can be eliminated as long as measurement composition is possible.

3. Incremental estimation

In this paper, the estimation 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 $\mathbf{z} = [z_1 \dots z_m]$ 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 incrementally.

3.1. State estimation

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

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

where $h(\theta_{i_k}, \theta_{j_k})$ are the nonlinear measurement function and z_k are the measurements with normally distributed noise with covariance Σ_k. Finally, $\ominus$ is the inverse composition operator, e.g. for 3D pose SLAM, if we have two poses $p, q \in \mathbb{R}^6 \hookleftarrow \mathfrak{se}(3)$, and $\mathfrak{se}(3)$ is the Lie algebra of the special Euclidean group $SE(3)$, $p \ominus q = \log(P\,Q^{-1})$, with $P, Q \in SE(3)$ and $P = \exp(p)$ and $Q = \exp(q)$, defining the logarithm map as $\log : SE(3) \to \mathfrak{se}(3)$ and the exponential map as $\exp : \mathfrak{se}(3) \to SE(3)$.

Iterative methods, such as Gauss–Newton or Levenberg–Marquardt, are often used to solve the NLS in (2). This is usually addressed by solving a sequence of linear systems at every iteration. Linear approximations of the nonlinear residual functions around the current linearization point $\boldsymbol{\theta}^i$ are calculated

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

with $\mathbf{r}(\boldsymbol{\theta}) = [r_1, \dots, r_m]^\top$ being a vector gathering all nonlinear residuals of the type $r_k = h_k(\theta_{i_k}, \theta_{j_k}) \ominus z_k$ 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

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

where the matrix A and the vector $\mathbf{b}$ are defined as $A \triangleq D^{-1\backslash 2} J$ and $\mathbf{b} \triangleq -D^{-1\backslash 2}\mathbf{r}$, with D gathering all the Σ_k measurement covariances (Dellaert and Kaess, 2006). The correction $\boldsymbol{\delta} \triangleq \boldsymbol{\theta} \ominus \boldsymbol{\theta}^i$ towards the solution is obtained by solving the linear system

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

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 back substitutions 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 with positive diagonal entries. The forward and back substitutions 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 $\boldsymbol{\theta}^{i+1} = \boldsymbol{\theta}^i \oplus \boldsymbol{\delta}$, $\oplus$ being the vectorial composition operator, given two poses $p, q \in \mathbb{R}^6 \hookleftarrow \mathfrak{se}(3)$, $p \oplus q = \log(P\,Q)$, with $P, Q \in SE(3)$ and $P = \exp(p)$ and $Q = \exp(q)$. The nonlinear solver iterates until the norm of the correction becomes smaller than a tolerance or the maximum number of iterations is reached.

3.2. Incremental updates

For large online problems, updating and solving the entire system at every step becomes very expensive. Therefore, online estimations need to be approached incrementally. At every step, a new variable and the corresponding observations are integrated into the system. This translates to new elements added to the summand in (2) and consequently new rows added to the matrix A in (4). For example, in case of a new observation $h_k(\theta_i, \theta_j)$ involving two variables, the update of A becomes

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

This translates into additive updates of the system matrix Λ

The sparsity of A_u can be used to identify the blocks in Λ as well as segments in the right-hand side $\boldsymbol{\eta}$ that change with this update. Considering $A_k = [J_i^j\,\Sigma_k^{-1\backslash 2} \dots 0 \dots J_j^i \Sigma_k^{-1\backslash 2}]$ the part of A_u which actually affects the system, we obtain

$$\hat{\Lambda} = \begin{bmatrix} \Lambda_{00} & \Lambda_{10}^\top \\ \Lambda_{10} & \Lambda_{11} + \Omega \end{bmatrix},\ \hat{\boldsymbol{\eta}} = \begin{bmatrix} \boldsymbol{\eta}_0 \\ \boldsymbol{\eta}_1 + \boldsymbol{\omega} \end{bmatrix}, \quad (7)$$

where Λ_{00}, Λ_{10} and $\boldsymbol{\eta}_0$ are the parts of the systems which remain unchanged, while Λ_{11} and $\boldsymbol{\eta}_1$ increment with

$\Omega = A_k^\top A_k$ and $\boldsymbol{\omega} = -A_k^\top r_k$, respectively. Theoretically, the solution $\boldsymbol{\theta}$ changes every step and in consequence the system matrix Λ and the right-hand side. $\boldsymbol{\eta}$ change entirely. In practice, though, the changes in the state vector are very small, sometimes affecting only a small part of the vector, and in consequence the changes in the system can be isolated and treated accordingly. This is the key factor in updating and solving a nonlinear system incrementally. This allowed for fast algorithms to solve the incremental estimation problem (Kaess et al., 2008, 2011b; Polok et al., 2013b). If the changes in the linearization point are substantial, the system matrix needs to be fully recalculated by computing the Jacobians using the new linearization point, but this happens less frequently in an incremental estimation problem. The recently introduced data structure in Kaess et al. (2011b), the Bayes tree, offers the possibility to develop incremental algorithms where reordering and re-linearization are performed fluidly, without the need of periodic updates. In Polok et al. (2013b) we proposed an elegant and highly efficient approach which combines the efficiency of matrix implementation and considers the insights gained using the Bayes tree data structure.

3.3. Incremental solving

As mentioned above, when incrementally calculating the solution to an NLS which continuously updates with new variables and observations, two situations can be distinguished; the small changes in the linearization point can be ignored and only the parts affected by the update need to be recalculated, or, less often, the linearization point changes significantly and the system needs to be recalculated entirely.

In our previous work (Polok et al., 2013b), we have shown that, in the same way as in Λ, the parts of the factorized form R affected by the update, can also be identified. The updated $\hat{R}$ factor and the corresponding right-hand side. $\hat{\mathbf{d}}$ can be written as

$$\hat{R} = \begin{bmatrix} R_{00} & R_{01} \\ 0 & \hat{R}_{11} \end{bmatrix}, \ \hat{\mathbf{d}} = \begin{bmatrix} \mathbf{d_0} \\ \hat{\mathbf{d}_1} \end{bmatrix}. \tag{8}$$

From $\hat{\Lambda} = \hat{R}^\top \hat{R}$, (7) and (8) the updated part of the Cholesky factor and the right-hand side can be easily computed

$$\hat{R}_{11} = \text{chol}(R_{11}^\top R_{11} + \Omega). \tag{9}$$

$$\hat{\mathbf{d}}_1 = \hat{R}_{11}^\top \backslash (\hat{\boldsymbol{\eta}}_1 - R_{01}^\top \mathbf{d_0}). \tag{10}$$

In order to update the R factor and the right-hand side. vector $\mathbf{d}$, it is possible to use (9) and (10), respectively. Nevertheless, without a proper ordering, R will quickly become dense, slowing down the computation. It is well known that Λ can be reordered to reduce the fill-in. This has one major disadvantage that the factor R changes completely with the new ordering, impeding the incremental factorization. The solution is to only calculate a new ordering for the parts of R which are being affected by the update. Polok et al. (2013b) shows how an efficient incremental ordering can be obtained by considering a partial ordering on a sub matrix of $\hat{\Lambda}$, which is slightly larger than $\hat{\Lambda}_{11} = \Lambda_{11} + \Omega$ and which satisfies the conditions of being square and not having any nonzero elements above or left from it. This guarantees that the ordering heuristics such as approximate minimum degree (AMD) will have information about the nonzero entries in $\hat{\Lambda}_{10} = \hat{\Lambda}_{01}^\top$, which would otherwise cause unwanted fill-in. A similar fluid reordering approached was introduced in Kaess et al. (2011a), and was obtained by applying partial elimination on a Bayes tree data-structure which is a graph representation of the factorized matrix Λ. In contrast, our proposed technique operates directly on the sparse block-matrix avoiding matrix-graph conversions.

Once the new ordering is calculated, a resumed factorization can be performed. The column, left-looking Cholesky calculates one column of the factor at a time, while only reading the values left to it. This algorithm can be used to "resume" the factorization of the right part of R while only using the reordered part of Λ and the unchanged part of the factor, R_{00}. The advantage of this approach is the overall simplicity of the incremental updates to the factor, while also saving substantial time by avoiding recalculation of R_{00}.

Back substitution is used to obtain the correction $\boldsymbol{\delta}$ and further the solution $\boldsymbol{\theta}$ from the updated $\hat{R}$ and the right-hand side. $\hat{\mathbf{d}}$. Maintaining a system representation updated with the new observations and variables every step and solved incrementally without affecting the quality of the estimation can highly increase the efficiency of the online MLE. Nevertheless, in many applications the uncertainty of the estimation is required. The following section describes how the required elements of the covariance matrix can also be calculated incrementally.

3.4. Covariance recovery

The covariance matrix of a system is given by the inverse of the system matrix $\Sigma = \Lambda^{-1}$. For large systems, such inversion is prohibitive, since the result is a dense matrix. Nevertheless, most of the applications require only a few elements of the covariance matrix, eliminating the need for recovering the whole 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 Björck (1996) and Golub and Plemmons (1980), it was shown how specific elements of the covariance matrix can be recursively calculated from the R factor and Kaess and Dellaert (2009) shows a practical implementation of this formula. For computation of multiple elements of the covariance matrix, such as the block diagonal, the recursive computation becomes efficient only if all the intermediate results are stored.

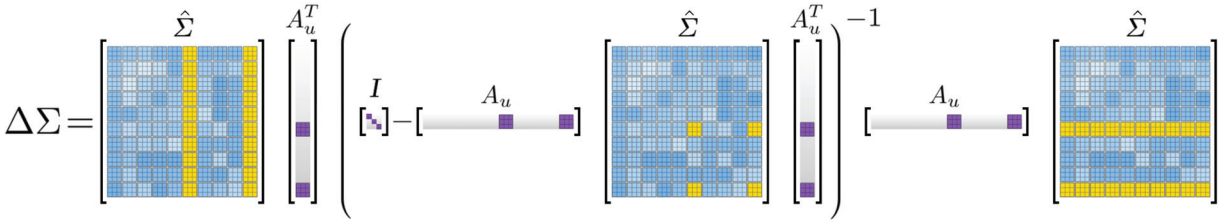

Fig. 1. Sparsity pattern of the matrices involved in calculating the increment on Σ (best viewed in color).

In subsection 3.2, we mentioned that most of the algorithmic speed-ups can be applied in case the linearization point is kept unchanged or changes partially. Then the effect of every new measurement can easily be integrated into the current system matrix Λ by a simple addition (see (7)). The matters get complicated when it is required to update its inverse

$$\hat{\Sigma} = (\Lambda + A_u^\top A_u)^{-1}. \tag{11}$$

By applying the Woodbury formula, it can be shown that in contrast to the information matrix which is additive, the covariance is subtractive

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

Here, $S \triangleq I + A_u \Sigma A_u^\top$ is a square invertible matrix with the size equal to the rank of the update A_u. This rank is usually much smaller than that of Λ and thus the cost of calculating this inverse is negligible compared to the full inverse in $\Sigma = \Lambda^{-1}$. Similarly to (11), one can downdate $\hat{\Lambda}$ to obtain Σ

$$\Sigma = (\hat{\Lambda} - A_u^\top A_u)^{-1} \tag{13}$$

and by applying the Woodbury formula the increment can now be calculated in terms of the new covariance $\hat{\Sigma}$

$$\Delta\Sigma = \hat{\Sigma} A_u^\top (I - A_u \hat{\Sigma} A_u^\top)^{-1} A_u \hat{\Sigma}. \tag{14}$$

Defining $U \triangleq I - A_u \hat{\Sigma} A_u^\top$, which is a matrix related to S, (14) becomes

$$\Delta\Sigma = \hat{B} U^{-1} \hat{B}^\top \text{ , with } \hat{B} = \hat{\Sigma} A_u^\top, \tag{15}$$

where looking at the sparsity pattern in the $\hat{\Sigma} A_u^\top$ product, it becomes apparent that only the block columns of $\hat{\Sigma}$, corresponding to the nonzero blocks in A_u or the variables $\mathbf{v}$ being updated, are required. The sparsity pattern of all the matrices involved in the calculation of the increment is shown in Figure 1.

We call this $\hat{\Sigma}_\mathbf{v}$ and can thus equivalently write $\hat{B} = \hat{\Sigma}_\mathbf{v} A_u^\top$, with $\hat{\Sigma}_\mathbf{v}$ obtained by 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 (5). Here, $I_\mathbf{v}$ is a square matrix with identity diagonal block on columns corresponding to the variables $\mathbf{v}$ and zeros elsewhere.

Even though it sounds counterintuitive to compute an increment $\Delta\Sigma$ form the already (albeit partially) incremented value $\hat{\Sigma}_\mathbf{v}$, it allows us to update the covariance at any step from A_u and $\hat{\Lambda}$ or $\hat{R}$, instead of having to store the old Λ or R in addition to the old Σ. In this way, it is not mandatory to update Σ at each step: when performing an update to Σ over several steps, A_u will simply contain all the measurements since Σ was last calculated. In this way, the covariance can be calculated incrementally, on demand whenever it is needed. Based on whether or not the linearization point changed or the number of variables being updated gets very large, the algorithm for calculating the covariance incrementally has two branches: a) calculates sparse elements of the covariance matrix using the recursive formula as introduced in Golub and Plemmons (1980), and b) updates sparse elements of the covariance using the covariance downdate in (15). The detailed algorithm can be found in Ila et al. (2015).

3.5. *The sparsity and the block structure*

The problems in robotics are in general *sparse*, which means that the associated system matrix is primarily populated with zeros. Many efforts have been recently made to develop efficient implementations to store and manipulate sparse matrices. CSparse (Davis, 2006a), developed by Tim Davis (Davis, 2006b) is one of the most popular sparse linear algebra libraries. It is highly optimized in terms of run time and memory storage and it is also very easy to use. CSparse stores the sparse matrices in compressed sparse column format (CSC) which considerably reduces the memory requirements and is suitable for matrix operations.

Furthermore, in many estimation problems, the random variables have more than one degree of freedom (DOF). For example, in 3*D*-SLAM the poses $\theta_i \in \mathbb{R}^6 \hookleftarrow \mathfrak{se}(3)$ have 6 DOF and the landmarks $l_j \in \mathbb{R}^3$ have 3 DOF. 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 system matrix is $N \times N$, where N is a sum of the products of the number of variables of each type and their corresponding DOF (Blanco, 2010).

The block structure and the sparsity of the matrices can bring important advantages in terms of storage and matrix manipulation. Some of the existing implementations rely on sparse block structure schemes. In g2o (Kümmerle et al., 2011), matrices are represented as a vector column of blocks where each block is row-indexed associative

array of matrix blocks. This is similar to sSBA (Konolige et al., 2010), with the exception that g2o blocks can take any size. Notably, neither iSAM (Kaess et al., 2008) nor iSAM2 (Kaess et al., 2011b) employ any sparse block matrix representation at all. iSAM uses a modification of (element-wise) sparse compressed column format optimized for incremental element addition. iSAM2 is based on a graph data structure where each node is a variable which can have different size depending on its type. Google's Ceres solver (Agarwal and Mierle, 2012) has its own block matrix storage quite similar to g2o, with the difference that the blocks are stored in an array of matrix element values rather than each block separately. However, Ceres implements almost no operations on their block matrices and so those are merely in a role of intermediate storage before converting to compressed sparse column and passing the system to a linear solver such as Cholmod. In the existing schemes, the block structure is maintained until the point of solving the linear system. Here is where CSparse (Davis, 2006b) or CHOLMOD (Davis and Hager, 1997) libraries are used to perform the element-wise matrix factorization. Once it has been compressed, it becomes impractical and inefficient to change a matrix structurally or numerically and therefore these implementations need to convert their block matrices to CSC at each linear solving step.

This motivated us to find efficient solutions for arithmetic operations on sparse block matrices, especially the matrix factorization, as well as solutions to sparse block matrix modification and storage. Our recent work maximally exploits the sparse-block structure of the problem. On one hand, the block matrix manipulation is highly optimized, facilitating convenient structural and numerical matrix changes while also performing arithmetic operations efficiently. On the other hand, the block structure is maintained in all the operations including the matrix factorization, variable ordering and covariance recovery, eliminating the cost of converting between sparse element-wise and sparse blockwise representation. Correct manipulation of the *block matrices* enabled very efficient NLS and incremental NLS solutions (Polok et al., 2013b,c) implemented in SLAM++ library, which outperformed other similar state-of-the-art implementations, without affecting the precision in any way. What is interesting about the block matrix format employed in SLAM++ is the use of template meta-programming for further acceleration using loop unrolling and SIMD instruction sets such as SSE. This is a novel feature in the context of sparse block matrix work which sets our work apart and yields a considerable performance advantage. In this paper, we will show that, based on the previously proposed block-based data structure in Polok et al. (2013c), we can also efficiently recover the marginal covariance matrices incrementally to be used in a state-based loop closure detection and compact representation of the SLAM problem.

4. Information based compact 3D pose SLAM

Pose SLAM is a variant of SLAM where only the robot trajectory is estimated and the sensor measurements are only used to produce relative constraints between the robot poses. In this case the state vector $\boldsymbol{\theta} = [\theta_1, \theta_2 \dots \theta_n]$ gathers only the variables corresponding to the robot poses. To reduce the computational cost of the Pose SLAM and to facilitate its application to very large scale problems, we previously introduced an approach that only takes into account highly informative loop closure links and non-redundant poses in an information filtering framework (Ila et al., 2010). A more compact representation of the SLAM problem reduces the memory requirements to store the entire state of the robot as well as it is more computationally efficient, since the systems to be solved are small. Note that maintaining the sparsity of the problem is an important factor when generating efficient solutions. In all existing approaches, maintaining the sparsity comes at a price of introducing some approximations but in general those approximations try to minimize the loss of information in the system.

Filtering is well known to produce less accurate solutions for the SLAM problem, therefore, in this paper, we extend the approach introduced in Ila et al. (2010) to maximum likelihood estimation. The existing strategies in this direction focus on how to select the measurements and the variables to be removed from the state representation, and on how the removing process unfolds.

Mutual information of the laser scans is used in Kretzschmar and Stachniss (2012) to decide which measurements and nodes should be removed from the pose graph representation of the SLAM. The problem of removing a variables and measurements form the SLAM state representation is from a graph is slightly different from the problem of deciding on the fly whether or not a variable or a measurement is added to the state. The former involves variable marginalization which in turn produce a dense rather than sparse state representation. Therefore, the existing approaches resort to local approximations such as Chow-Liu trees to obtain a sparse graph. Several aspects on how to apply graph sparsification by marginalization in the context of MLE SLAM are discussed in Carlevaris–Bianco and Eustice (2014a). Johannsson et al. (2013), on the other hand, introduced an incremental strategy which avoids marginalization and which is similar to the one proposed in Ila et al. (2010) and its extension to MLE-SLAM proposed in this paper, with the difference that their method requires relocalization when revisiting parts of the map and the fact that it is applied in the context of visual SLAM.

The strategy introduced in this paper is incremental and therefore suitable for online SLAM, has a general application to any type of pose SLAM problems, can be easily extended to landmark SLAM or SFM, it is complete in the sense that provides not only the way to calculate the

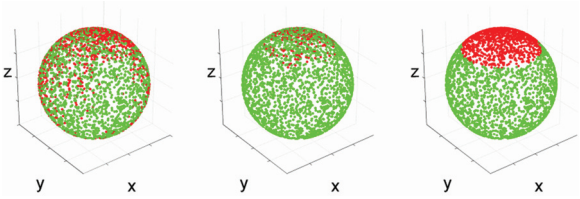

Fig. 2. Rotation threshold applied to 5000 normally distributed random rotations. Each point is a rotated view direction vector; roll is however not visible. The $z+$ axis is the forward direction. Distance threshold is applied to: left) the elements of the axis-angle vector, middle) the rotation angle (note the dependence on roll) and right) the angular change of view direction. The angular threshold is $\frac{\pi}{4}$ in all cases.

distance and information measures but also the corresponding methods to obtain the required thresholds. The compact representation is achieved by computing two measures, the proximity in terms of sensor range of current pose to any other previous poses and the information gain for each candidate link. Apart from maintaining a compact representation of the state, the calculation of the proximity of two poses in terms of sensor range can provide a more efficient loop-closure detection than the actual registration of all the sensor readings or even the appearance-based techniques.

4.1. Distance measure

A measure of proximity of two poses is the relative displacement calculated from the current estimation (Ila et al., 2010). In the following formulations, the vectors corresponding to a single state variable, θ_i, are denoted in lower case to be consistent with the previous notation. The scalar elements of those vectors will be marked with corresponding subscripts. In an online application, the displacement between the current pose, θ_n, to any other previous pose in the trajectory, θ_i, can be estimated as a Gaussian with parameters

$$\mu_d = D(\mu_i, \mu_n) \tag{16}$$

$$\Sigma_d = [J_i\, J_n] \begin{bmatrix} \Sigma_{ii} & \Sigma_{in} \\ \Sigma_{in}^\top & \Sigma_{nn} \end{bmatrix} [J_i\, J_n]^\top, \tag{17}$$

where $D(\cdot)$ calculates the relative displacement between the mean estimates of the two poses, Σ_{ii} and Σ_{nn} are the marginal covariances and Σ_{in} is the cross correlation between the i^{th} and the current (n^{th}) pose.

In an online application, at each step, one can test the proximity of the current pose with any of the previously visited poses to determine if there is a possibility of the sensor range overlap. This can be obtained by calculating the probability of a pose θ_i being closer than v to the pose θ_n along each dimension, v being the sensor range. We marginalize the distribution on the displacement for each of its dimensions, r to get a one-dimensional Gaussian distribution $\mathcal{N}(\mu_r, \sigma_r^2)$ that allows to compute the probability

$$\begin{aligned} p_r &= \int_{-v_r}^{+v_r} \mathcal{N}(\mu_r, \sigma_r^2) \\ &= \frac{1}{2}\left(\mathrm{erf}\left(\frac{v_r - \mu_r}{\sigma_r\sqrt{2}}\right) - \mathrm{erf}\left(\frac{-v_r - \mu_r}{\sigma_r\sqrt{2}}\right)\right). \end{aligned} \tag{18}$$

If, for all dimensions, p_r is above a given threshold, s, then the pose θ_i is considered close enough to the current robot pose, θ_n. We include θ_n in the state only if no other poses in the representation are close to it.

The thresholds v are derived from the sensor characteristics: the field of view for cameras, the maximum distance for the laser scan alignment, etc. In general, it is simpler to define a threshold for each dimension separately than to define a single threshold for a measure integrating the distances along all dimensions (e.g. a weighted norm). Therefore, separate thresholds for translation and rotation are defined.

The translation component of the v threshold can be easily determined from the sensor range. For the rotational component, however, there are several ways to define the threshold. Given the rotation component $q_i = \mu_i(4:6)$ and $q_j = \mu_j(4:6)$ then $q_d = \mu_d(4:6)$ is the relative rotation of the two poses, represented by an axis angle vector, obtained by taking unit length axis of rotation and multiplying it by the rotation angle in radians. The easiest is to calculate the probability of each element in q_d to be below a threshold v_a but this is incorrect due to the strong correlation between the elements of the rotational components. This can be seen in Figure 2, left.

A correct way is to compute the probability of the magnitude of the relative rotation $|q_d|$ to be smaller than a threshold v_a, 2, middle. In this particular application though, we do not want just to limit the relative rotation but to see if the fields of view of the sensor overlap. Therefore, in the case of using cameras, a more permissive threshold can be considered, a threshold invariant to roll. We can compute the probability of the angle of the relative *view direction rotation* to be smaller than a threshold v_a. This threshold is shown in Figure 2, right.

The last one, although being desirable, involves calculation of nontrivial Jacobians. Since we deal with uncertainty in the estimate, the probability depends on the marginal covariances. These covariances need to be transformed from covariances on the state space, $\mathbb{R}^6 \times \mathbb{R}^6$, where the 6 DOF are $[x, y, z]$ and a $\mathbb{R}^3$ axis angle rotation, to the threshold space, $\mathbb{R}^4 \times \mathbb{R}^4$, where the 4 DOF are $[dx, dy, dz]$ and a (scalar) angle of the relative view direction rotation.

To calculate the view direction vectors, one can convert the rotations from axis angle representation to rotation matrices, where the columns of the matrix give the directions of the principal axes of the coordinate frame, $Q_i = \mathrm{ROT}(q_i)$ and $Q_j = \mathrm{ROT}(q_j)$. Assuming that the view direction coincides with the 'z+' axis, the view directions are $d_i = Q_i(1:3, 3)$ and $d_j = Q_j(1:3, 3)$. Finally, the view

direction angle is $\alpha = \arccos(d_i \cdot d_j)$. This can also be calculated directly from the relative rotation as

$$\begin{aligned}\alpha &= \arccos(\text{ROT}(r_d)(1:3,3)\cdot[0\ 0\ 1]^\top) \\ &= \arccos(\text{ROT}(r_d)(3,3)).\end{aligned} \tag{19}$$

By expanding Rodriguez' rotation formula, we can see that

$$\text{ROT}(r_d)(3,3) = \cos(\|r_d\|) + (1-\cos(\|r_d\|))\frac{r_d(3)^2}{\|r_d\|^2}. \tag{20}$$

This finally gives us the lower dimension (4 DOF) distance $\hat{\mu}_d = [\mu_d(1:3), \text{ROT}(\mu_d(4:6))(3,3)]$. The Jacobian of this transformation is $J_{tr} = \frac{\partial \hat{\mu}_d}{\partial \mu_d}$, which is a 4×6 matrix. The covariance is then transformed with $\hat{\Sigma}_d = J_{tr}\Sigma_d J_{tr}^\top$ and the result is a 4×4 matrix. Distance threshold can be applied in this space in order to determine whether the poses have overlapped field of view and in consequence loop closure links can be obtained.

4.2. Information measure

The mutual information quantifies the Entropy reduction in the system after the integration of an observation. For Gaussian distributions, it is given by the logarithm of the ratio of determinants of prior and posterior state covariances (Dissanayake et al., 2002; Ila et al., 2010; Sim, 2005). In Ila et al. (2010) has been shown that by algebraically manipulating this ratio of determinants, one can easily obtain the mutual information from the uncertainty of the observation, Σ_k and $S = I + A_u\,\Sigma A_u^\top$ the innovation matrix

$$\mathcal{I} = \frac{1}{2}\ln\frac{|\Lambda + A_u^\top A_u|}{|\Lambda|} = \frac{1}{2}\ln|\Sigma_k^{-1}|\cdot|S|. \tag{21}$$

In case of pose SLAM, an observation is given by a relative transformation between two poses. Therefore, the estimated mutual information can be written in terms of the uncertainty of the observation, Σ_k and the uncertainty of the edge Σ_d

$$\mathcal{I} = \frac{1}{2}\ln|\Sigma_k^{-1}|\cdot|\Sigma_k + \Sigma_d|, \tag{22}$$

where Σ_d is calculated as in (17). In this way, the mutual information of a potential observation can be estimated by specifying the marginal covariances of the poses involved in the observation and using an initial guess for the uncertainty of the observation. After sensor registration, the exact uncertainty of the observation is known and the mutual information of the link can be evaluated precisely.

4.3. Compact 3D pose SLAM – The algorithm

This section describes the algorithm to obtain a compact representation of the 3D pose SLAM problem in the context of maximum likelihood estimation. The algorithm is meant for online applications; therefore, it involves incremental estimation strategies. The efficient incremental solving summarized in sub section 3.3 and detailed in Polok et al. (2013b) is used together with incremental covariance recovery summarized in sub section 3.4 and detailed in Ila et al. (2015). The two measures introduced above, the estimated relative transformation between two poses and the mutual information are used to select only those poses which are relevant and those edges that are informative. The result is an incremental algorithm for compact pose SLAM, which automatically maintains a sparse representation of the state in the information space. Its efficiency comes from its sparsity, the incremental computations and the fact that all the matrix computations are done by blocks.

Algorithm 1 Incremental compact SLAM estimation.

Require: thresholds: v, s, g_{pose}, and g_{loop}
Require: expected sensor covariance: $\bar{\Sigma}_y$
Require: initial state: $\boldsymbol{\theta}_0$, Σ_0
$(\boldsymbol{\theta}, S) = \text{INITSYSTEM}(\boldsymbol{\theta}_0, \Sigma_0)$
keepPose = TRUE
$n = 1$
$I_{n-1} = \text{GETNEXTDATA}$
while $I_n = \text{GETNEXTDATA}$ **do**
  $(\mu_u, \Sigma_u) = \text{REGISTRATION}(I_n, I_{n-1})$
  if *keepPose* **then**
    $(\mu_e, \Sigma_e) = (\mu_u, \Sigma_u)$
    $(\boldsymbol{\theta}, S, \Sigma_M) = \text{INCUP}(\boldsymbol{\theta}, S, \Sigma_M, (\mu_e, \Sigma_e))$
  else
    $(\mu_e, \Sigma_e) = \text{CONCATENATEPOSE}((\mu_e, \Sigma_e), (\mu_u, \Sigma_u))$
    $(\boldsymbol{\theta}, S, \Sigma_M) = \text{REPLUP}(\boldsymbol{\theta}, S, \Sigma_M, (\mu_e, \Sigma_e))$
  end if
  $C = \text{SEARCHLOOPCLOSURE}(\boldsymbol{\theta}, \Sigma_M, v, s)$
  $\mathcal{I} = \text{MINFO}(C, \boldsymbol{\theta}, \Sigma_M, \bar{\Sigma}_y)$
  loopClosed = FALSE
  while $C \neq \varnothing$ **do**
    $i = \text{ARGMAX}(\mathcal{I})$
    if $C(i) < n-1$ **and** $\mathcal{I}(i) > g_{\text{loop}}$ **then**
      $(\mu_y, \Sigma_y) = \text{REGISTRATION}(I_n, I_{C(i)})$
      if $\text{NOTVOID}(\mu_y)$ **then**
        $\mathcal{I}(i) = \text{MINFO}(C(i), \boldsymbol{\theta}_{C(i),n}, \Sigma_{M_{C(i),n}}, \Sigma_y)$
        if $\mathcal{I}(i) > g_{\text{loop}}$ **then**
          $(\boldsymbol{\theta}, S, \Sigma_M) = \text{INCUP}(\boldsymbol{\theta}, S, \Sigma_M, (\mu_e, \Sigma_e))$
          $\mathcal{I} = \text{MINFO}(C, \boldsymbol{\theta}, \Sigma_M, \bar{\Sigma}_y)$
          loopClosed = TRUE
        end if
      end if
    end if
    $(C, \mathcal{I}) = (C, \mathcal{I}) \setminus \{i\}$
  end while
  keepPose = (*loopClosed* **or** $\text{MIN}(\mathcal{I}) > g_{\text{pose}}$)
  $n = n + 1$
end while

The algorithm is detailed in 1. It requires four parameters, the sensor range, v, the probability to accept a pose as having overlapping field of view with the current one, s, and the minimum information gain to add poses, g_{pose}, and to close loops, g_{loop}.

The algorithm starts with an initial state and loops while there are measurements (edges) to be integrated into the system. The measurements are relative transformations between the robot poses obtained by registering either images or laser scans (line 6). Registration function can also account for other sensors such as odometry or IMU. In case that the previous pose was not integrated into the system, the new relative transformation is concatenated to it at line 11. The concatenation is done in both mean and covariance space. The mean is obtained by pose composition and the covariance of the composed poses can be obtained as in Smith and Cheeseman (1986) and requires the Jacobians of the pose composition function, or as in Barfoot and Furgale (2014) to handle the uncertainty of the $SE(3)$ pose composition. Correct computation of the composed measurement covariance is important to yield conservative estimates compared to the full solution, otherwise the resulting estimates could end up being overconfident (low covariance) and potential loop closures would be lost, or conversely not confident enough (high covariance) which would deteriorate performance by performing too many unnecessary sensor registration attempts. The method used in our approach leads to a conservative estimate, as demonstrated is 5.5. The resulting transformation updates the system either by incrementing the state (line 9) in case that the pose is to be kept in the system, or by replacing the previous pose in case that it is deemed redundant (line 12).

At line 14, the algorithm searches for potential loop-closures by applying the distance test introduced in sub section 4.1. The search returns C, a set of candidate pose ids. Before proceeding with the sensor registration, the relevance of the possible links that can be established with the candidates C is determined by calculating the mutual information as in (22), at line 15. In here, an estimate of the sensor noise, $\bar{\Sigma}_y$, is used – the actual value is only available after the registration (and then the mutual information is updated, line 22). Nevertheless, given known sensor characteristics, in practice, using an estimated value offers good means to select the relevant links.

Starting with the most informative candidate, the algorithm performs sensor registration to obtain the link that will update the system. Observe that the mutual information of each of the remaining candidates is recalculated after the update (line 25), and eventually only a few or no further candidates are registered. This is due to the fact that informative links substantially change the entropy of the system, and the other candidates become irrelevant.

At line 32, the algorithm decides whether or not the current pose is added to the system. A pose is added to the system if it is part of an informative link or if the possible links it can establish are informative. In this way the algorithm manages to maintain a set of poses and links uniformly distributed in the information space. Tests on real and simulated datasets described in the next section will show how this strategy considerably reduces the computational costs while maintaining a good accuracy of the estimate. Note that only poses that do not involve possible informative loop closure are removed, therefore marginalizing out those poses can be done by simply concatenating the edges that involve the removed pose. This has the advantage that marginalization is guaranteed to not introduce extra-edges in the graph and therefore, there is no requirement for complementary strategies to sparsify the resulting subgraph as in Carlevaris–Bianco and Eustice (2013).

The main difference between the algorithm proposed in this paper and the one in Ila et al. (2010) is the fact that in here the state pruning is performed in a MLE framework, whereas in Ila et al. (2010) it was integrated in a filtering approach. Filtering is, in general, much simpler than the current incremental approach, mainly because the linearization point stays fixed. Nevertheless, this is also an important source of errors in the estimation. The current strategy keeps this error low by updating the linearization point when needed. Another difference between the two approaches is that the latter is applicable to any variable dimensions (2D or 3D SLAM) as well as it is easy to extend to landmark SLAM or even structure from motion problems.

5. Experimental validation

This section evaluates both the state-based loop closure detection and the online state reduction strategy proposed in section 4. The incremental SLAM implementation integrates the incremental solving with fluid reordering and resumed Cholesky factorization proposed in Polok et al. (2013b) and the incremental covariance recovery introduced in Ila et al. (2015) together with the information-based pose and links selection proposed in this paper. The code and the help bash scripts associated to the experiments can be found in Extensions 2 to 4 associated to this paper. The included Readme files explain how to perform the compilation and the execution. Extension 2 comprises SLAM++ (https://sf.net/p/slam-plus-plus/) nonlinear least squares library implemented in C++, and also the proposed algorithm. The library performs highly efficient operations on sparse block matrices which have proven to outperform most of the existing state of the art incremental SLAM software (Ila et al., 2015; Polok et al., 2013c). Finally, Extension 1 contains a short video demonstrating the proposed technique.

The tests were performed on a computer with Intel Core *i*5 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.

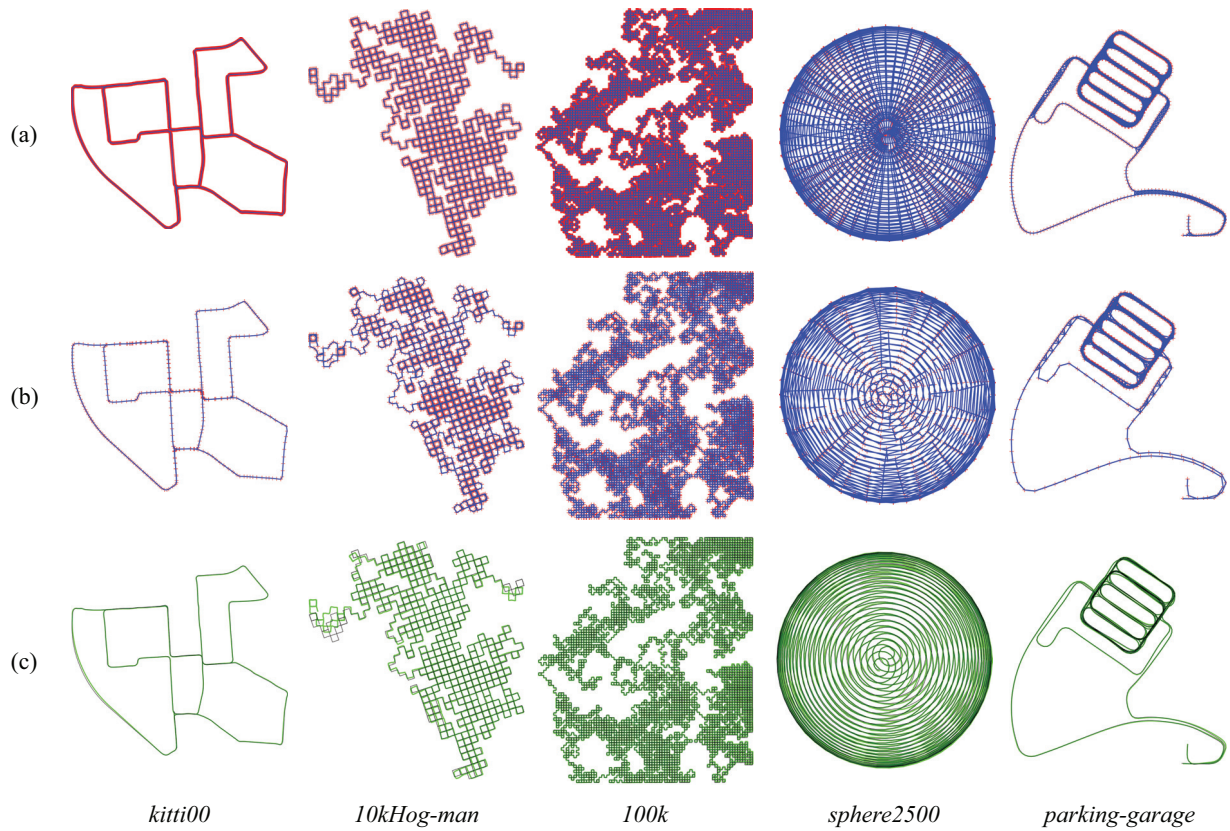

Fig. 3. The datasets. The tiny red crosses mark the robot poses over the blue estimated trajectory in (a) full state APAL-SLAM versus (b) compact FPFL-SLAM and (c) in green the recovered full state compared with the ground truth in black.

5.1. Datasets

The tests were performed on several simulated and real datasets which can be seen in Figures 3 and 7 and can be found in Extension 5 in a graph file format, which is a popular format in SLAM community. The following subsections describe the datasets and the evaluations in more detail.

5.1.1. Simulated datasets. Several simulated datasets are used to test the methods proposed in this paper. We used four simulated datasets with available ground truth, one generated by our code, called *ellipse3D* and three other datasets which are publicly available, *sphere2500* dataset (Kaess et al., 2008), *10kHog-man* and *100k* (Grisetti et al., 2007). While the first two, *ellipse3D* and *sphere2500* are 3D datasets, the last two are 2D pose SLAM datasets.

The *ellipse3D* dataset was generated from a ground truth trajectory in form of two concentric 3D ellipses, the first one with semi-axes of 10 m and 6 m and the second with semi-axes of 20 m and 6 m, respectively. The length of the total trajectory is of 170 robot poses over 72.29 m. The relative transformations between the robot positions are measured with a sensor with 5% error in translation and 5% error in orientation. The sensor is able to establish a link between any two poses closer than 3 m and 0.2 rad in orientation.

5.1.2. Real datasets. Two real datasets were also used to validate both the state-based loop closure detection as well as the algorithm to maintain a compact representation of the SLAM problem. We used the *kitti* dataset with available GPS ground truth (Geiger et al., 2013). This dataset contains several trajectories from which we selected the longest one, *kitti00* (4541 poses). The stereo images are processed by a front-end connected to the SLAM++ nonlinear optimizer. In an on-line processing scenario, the connection is be bidirectional; the front-end provides relative measurements between camera poses to the compact pose SLAM (Algorithm 1, line 6), and the compact pose SLAM provides the indices of pairs of images that are matching candidates. The stereo processing pipeline follows a standard stereo processing algorithm, including feature detection and matching, triangulation and 3D point cloud alignment followed by least square reprojection error refinement.

However, the *kitti00* dataset is used to validate the state-based loop closure strategy proposed in this paper which is compared with the appearance based methods. The tests analyze the threshold sensitivity of both methods, therefore repeated runs with different thresholds need to be performed. For that, we generated a file containing all possible loop closures, by exhaustively matching all-to-all images, and computing the relative transformations within a RANSAC approach. Only the transformations with the

inlier ratio greater than 0.35 are kept (using a higher threshold leads to having only highly relevant loops, which would make the work of the loop-closing algorithm a simple one). In this way all the tests in subsection 5.4 were done on the same dataset.

The uncertainty of each relative pose measurement in the *kitti00* dataset is estimated using Monte Carlo approach: each feature point used in the calculation of the relative transformations is corrupted using zero mean Gaussian error with variance equal to 1 px and samples are drawn from that distribution. Each sample is propagated through the triangulation and relative camera pose estimation to find the measurement covariance. The images can be used to detect loop closures based on appearance, therefore this dataset is used to compare the loop closure strategy based on state estimation to an appearance-based one.

A second 3D dataset called *parking-garage* was also used in our tests (Kümmerle et al., 2011). This dataset is a 3D pose graph of a multi-level parking garage and has about 1661 robot poses and 6275 edges. Being a graph file dataset, this dataset assumes loops are already detected, therefore it will be used only in testing the compact SLAM representation. Since this dataset does not provide ground truth, it was processed by one hundred iterations of a batch solver and the results were used as a de-facto ground truth.

5.2. Compact pose SLAM evaluation

Maintaining a compact state representation in online SLAM can lead to great computational savings. Three strategies were tested: a) all the poses and all the possible loop closures are integrated into the system (denoted as all poses, all loops (APAL)), b) all the poses but only informative loop closure links are integrated into the system (all poses, few loops (APFL)) and c) relevant poses and informative loops are integrated into the system (few poses, few loops (FPFL)). The selection of one or another strategy can be easily implemented by appropriately setting the distance and information thresholds (v, s, g_{pose} and g_{loop}) in Algorithm 1. In particular, setting s to zero and both g_{pose} and g_{loop} negative infinity, respectively, leads to APAL strategy with all the loop closures detected by the distance test. Setting up a higher value for the probability threshold s reduces the number of loop closure candidates. Setting g_{loop} to a minimum mutual information a loop needs to have to be added to the system leads to APFL strategy, and setting g_{pose} to a minimum of information a link connecting a pose must have, leads to FPFL. Again, we offer an automated solution to select the adequate thresholds for the strategy to use. The three strategies were tested on above mentioned datasets. Execution time and translational and rotational errors are provided in Tables 3 and 4, respectively.

To select suitable values of thresholds v, s and g_{pose} and g_{loop}, a representative part of each dataset is used for setting the thresholds. The value of 60% (by the number of vertices) was used with all the datasets to make sure that a major loop closure is included in this sample, except for the *100k* dataset which is highly repetitive and only 20% sample was used. On this sample, poses and measurements are incrementally added into the system and, at each step, σ_d^2 and μ_d are recorded for each loop closure with the current pose. At the end, it is possible to find such a sensor range v so that all loops would have probability above a certain threshold s, using (18). The choice of s is arbitrary and we chose the value of 0.1 to stay in the region where the number of proposed candidates is stable, with some space for increasing this value if required. Conversely, starting with a known sensor range (e.g. from the physical characteristics of the given sensor and the capabilities of the corresponding sensor registration algorithm), it is possible to find such value of s that no loop closures are lost.

For the APAL scenario, g_{pose} and g_{loop} are simply set to negative infinity in order to accept all the poses and loops as having sufficiently high mutual information. A run with this configuration is then performed while recording the mutual information of all the ground truth loop closures edges as well as the mutual information of the edges linking every new pose (recorded in Algorithm 1, lines 23 and 32).

To generate the APFL configuration, g_{loop} is then set to $e^{(1.36ln(l_{90}+1))} - 1$ where l_{90} is a 90-percentile of all the recorded mutual information of the loop edges. It is possible to run with this configuration and verify that no important loops are lost and at the same time enough loops are being discarded. If that is not the case, it is possible to manually adjust the threshold before selecting the g_{pose} threshold. To this end, the Extension 4 contains a simple script which takes the initial g_{pose} as an input and runs several tests using the multiples of this value, distributed in the range $[g_{\text{pose}}\frac{1}{10}, 10g_{\text{pose}}]$ in such a way that the ratio of the adjacent thresholds is a constant. Then, it is just a matter of choosing the preferred trade-of between speed and precision and using the corresponding threshold.

To obtain FPFL we fix g_{loop} at the chosen value and set g_{pose} to $e^{(1.7\ln(p_{90}+1))} - 1$ where p_{90} is 90-percentile of all the recorded pose mutual information. Another run with this configuration is performed to make sure that the pose graph sparsity is as expected. Again, if the result is not satisfactory, it is possible to execute several runs with scaled values of g_{pose} and to choose a suitable value for the threshold. Note that setting the thresholds happens only on the *sample* rather than on the entire dataset. Also note that in most cases the thresholds proposed by the above-mentioned heuristic do not require further fine-tuning.

The above-mentioned heuristics were developed by first running exhaustive tests on all the datasets, then manually choosing the preferred thresholds and finally finding a function which would yield values close to the manually chosen ones. The thresholds used in our tests are listed in Table 2. Out of all the thresholds, only two had to be manually modified. Specifically, in *parking-garage* the g_{loop} was reduced in order to allow more loop closures and decrease the error, and in *sphere2500*, g_{pose} was increased in order

to obtain a more compact representation. The suggested thresholds would still work in both cases, except that in the former case of *parking-garage* the solution would be less precise (accepted only 115 out of 4615 loops rather than 964 with the decreased threshold) and in the latter case of *sphere2500*, it would be less compact (accepted 2500 poses rather than only 959 with the increased threshold).

5.3. Error evaluation

In order to evaluate the compact pose SLAM algorithm, we want to compare the rotational and translational errors of the final estimate for all the three cases mentioned above. The problem when evaluating the accuracy is the fact that the size of the state varies in all cases. Several types of errors are proposed in the literature for evaluating the SLAM problem. Relative pose error (RPE) was used in Kummerle et al. (2009) and Sturm et al. (2012) and was shown to be useful in the evaluation of the graph based SLAM. A more intuitive way is to compare the absolute trajectory error (ATE) after registering the two configurations: the ground truth and the estimated graph (Sturm et al., 2012). For calculating the relative pose error or absolute trajectory error, we first need to find a way to recover the poses corresponding to the poses in the initial graph which are missing in the compacted representation.

One way of doing this is by applying linear interpolation to each edge in the compact representation, while using the contribution of the corresponding edges in the initial representation as weights. The cumulative weights can be calculated given the measurements $z_{\cdot,\cdot}$ as follows

$$w_u = \frac{\sum_{k=i}^{i+u-1} \left\| z_{k,k+1} \right\|_2}{\sum_{l=i}^{i+q-1} \left\| z_{l,l+1} \right\|_2}, \tag{23}$$

where q is the length of the path in the graph between variables ${}^I\theta_i$ and ${}^I\theta_{i+q}$ in the initial (I) state, where ${}^I\theta_i$ corresponds to ${}^C\theta_j$ in the compacted (C) graph and similarly ${}^I\theta_{i+q}$ corresponds to ${}^C\theta_{j+1}$, and $0 \leq m \leq q$ is index of a pose in this path.

The poses ${}^I\theta_i$ through ${}^I\theta_{i+q}$ can now be interpolated as

$${}^I\theta_{i+u} = {}^C\theta_j \oplus w_m \cdot ({}^C\theta_{j+1} \ominus {}^C\theta_j). \tag{24}$$

Another way is to apply the weighting in the error space. For that we calculate the relative displacement (error) $d_{j,j+1}$ between the initial and optimised estimation of each odometric edge in the compacted graph

$$d_{j,j+1} = h({}^C\theta_j, {}^C\theta_{j+1}) \ominus (z_{i,i+1} \oplus \ldots \oplus z_{i+q-1,i+q}) \tag{25}$$

and the poses ${}^I\theta_i$ through ${}^I\theta_{i+q}$ can be approximated as

$${}^I\theta_{i+m} = {}^C\theta_j \oplus (z_{i,i+1} \oplus \ldots \oplus z_{i+m-1,i+m}) \oplus w_m \cdot d_{j,j+1}. \tag{26}$$

Note that in both (24) and (26), ${}^I\theta_i = {}^C\theta_j$ and ${}^I\theta_{i+q} = {}^C\theta_{j+1}$ holds. Figure 4 shows that the interpolated trajectory using (26) represented by the green squares nicely follows the ground truth represented by the black dots. In the captions, this strategy corresponds to v3. On the other hand, the violet circles representing the interpolated trajectory using (24) and denoted v1, stay on the segments defined by the compact representation (big red crosses). Figure 4 also shows interpolation when the weights in (24) are uniform in small dark red crosses, denoted v0. Note how the violet circles concentrate near the curve, while the red crosses do not.

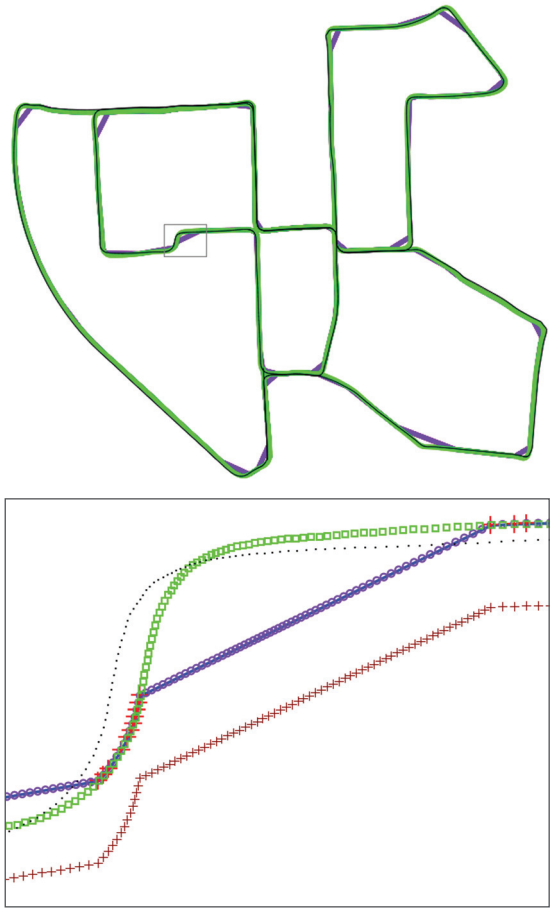

Fig. 4. *Kitti00* trajectory compacted to only 355 poses out of 4541. v0 and v1 in violet, v2 in green, ground truth in black (top), a detailed view of the trajectory, optimized poses marked by large red crosses, v0 small dark red crosses (shifted vertically to not overlap with v1), v1 violet circles, v2 green rectangles, ground truth in black (bottom).

Now that we have recovered the full state, we can rigidly align it to the ground truth trajectory e.g. using the Kabsch algorithm (Kabsch, 1976) and apply the error metrics as described in (2) and (5) in Sturm et al. (2012). These are the absolute trajectory error (ATE) – the error between the corresponding poses in the estimated and the ground truth trajectory, the relative pose error (RPE) – an error between the corresponding relative transformations between the consecutive poses in the estimated and the ground truth trajectory and finally the relative pose error all to all (RPE all-all) – an error between the corresponding relative transformations

Table 1. Error evaluation for the *kitti00* dataset.

	RMSE Error		
Alg.	ATE	RPE	RPE all-all
v0	3.853m, 3.392°	0.076 m, 0.459°	11.782 m, 3.374°
v1	3.818m, 3.598°	0.064 m, 0.501°	12.363 m, 3.581°
v2	**3.093**m, **2.250**°	**0.029**m, **0.119**°	**7.297**m, **2.223**°

Table 2. Compact SLAM thresholds. Note that the $\{X\}^Y$ notation in the v column means merely Y repetitions of X, and was introduced to save space.*The thresholds marked by asterisk were manually modified, as described at the end of 5.2.

Dataset	v	s	g_{pose}	g_{loop}
ellipse3D	$\{2.5\}^3, 0.4$	$\frac{1}{8}$	5.74	5.50
ellipseN	$\{1.1\}^3, 0.3$	$\frac{1}{10}$	5.70	5.11
kitti00	$\{25.5\}^3, 1.1$	$\frac{1}{10}$	8.51	5.13
parking-garage	$\{95.0\}^3, 1.1$	$\frac{1}{10}$	5.77	2.45*
sphere2500	$\{2.9\}^3, 0.1$	$\frac{1}{10}$	4.33*	9.14
10kHog-man	$\{8.9\}^2, 6.3$	$\frac{1}{10}$	2.36	1.94
100k	$\{27.6\}^2, 6.2$	$\frac{1}{10}$	2.36	4.08

between all the possible pairs of poses in the estimated and the ground truth trajectory. The results for the trajectory in Figure 4 can be found in Table 1. The v2 strategy always leads to the lowest error, therefore it will be further used for the rest of the evaluations. The translational and rotational components of these errors are reported separately in Table 4.

5.4. State-based loop closure detection

The Algorithm 1 integrates a loop closure search scheme based on the estimated state at each step. This is done at line 14 and it is based on the distance test described in subsection 4.1.

Alternatively, in case that the robot is equipped with an image sensor, appearance can be used to detect loop closures. *FAB-MAP2* is an appearance based method which classifies the place the robot is currently seeing as new or already seen before from a different pose (Cummins and Newman, 2010). If a current place is categorized as seen before, the online SLAM algorithm attempts to close the loop by matching the similar views. *FAB-MAP2* uses visual words to represent the appearance, which were obtained a priori from a training set. *FAB-MAP2* is a vision-based technique suitable for closing very large loops under good lighting condition in non-repetitive environments. On the other hand, the proposed state-based loop closure detection algorithm works for any exteroceptive sensors (lasers, sonars, etc.) which can be registered to obtain relative transformations and is independent of the environment, although it requires relatively small loops and good estimation in order to be highly efficient. The probability thresholds need to be set in concordance with the size of the loops and the errors in the estimation.

In the case of the *distance test*, one needs to provide a trusted sensor range threshold v_r for each measurement dimension and a probability threshold $s \in [0, 1]$. If s is set too low, more loop closure candidate are generated by the test, and this is not desired. If the threshold is set too high, some loop closures might be lost. Our tests show that, once the sensor range threshold v is set correctly, s drastically affects the number of candidates only in the very close vicinity of 0 and 1, being relatively conservative, otherwise. This characteristic favors the automatic selection of the threshold, and therefore allowed us to actually implement it in our code. This has a great benefit in real robotic applications, a robot can sample a small part of the environment and based on that, automatically decide on which thresholds to use for the rest of the long-run mission. To our best knowledge, this is the only existing loop closure detection strategy that allows a high level of automation of the process.

We further evaluated the dependence of the two loop closing methods on their respective thresholds and the effect it has on the precision. We also evaluated the number of detected valid loops. In order to do that, we processed each pair of images by standard sparse relative pose estimation procedure and applied a threshold of 0.35 on the matched keypoint inlier ratio. Figure 5 (top), shows the dependence of the number of detected loops and the corresponding solution error on the loop closure information gain threshold. Note that the horizontal axis is a factor of loop gain, with value 32 being equal to l_{90} (the 90% percentile of loop gains in the sample of the dataset) and the loop gain changes 100-fold down to value 0 or up to value 64, respectively, with the ratio between the consecutive gains being constant. This is because the information gain is a logarithmic quantity. It would also be possible to use a logarithmic plot, but then the vertical axis would be in the middle. Note that the number of detected loops changes smoothly, in two intervals: in $[10, 35]$ the relatively short loops with low information gain are being culled, whereas in $[48, 60]$ the long loops with high information gain are being culled. The threshold we applied in our evaluations falls in the plateau in between those two intervals. The error varies slightly and in value 65 it would spike up since that is the point where all the loop closures are culled. Also note that the number of missed loops is zero on the left and it reaches the number of the loops on the right.

On Figure 5 (middle) there is a plot of the number of *FAB-MAP2* candidates and the solution error, depending on the probability threshold, using the Oxford dictionary. Only the left frames (out of the stereo pairs in the *kitti* dataset) were used for all the *FAB-MAP2* evaluations. Note that the number of loops is relatively constant and changes abruptly in the $[0.0, 0.1]$ interval and, more importantly, also in the $[0.9, 1.0]$ interval which is the typical working point. Note that *FAB-MAP2* misses many of the loops with good inlier

Table 3. Time performance in seconds. APAL and FPFL, both include state-based loop closure detection. SLAM++, g2o and iSAM columns show the solving time for given data association.

Dataset	Time of covariance calculation/nonlinear solving (s)			Time (s)		FPFL (%)	
	SLAM++	g2o	iSAM	APAL	FPFL	Loop	Vert.
ellipse3D	0.112/0.051	1.197/0.160	1.156/1.868	0.502	0.066	2.91	23.52
ellipseN	30.564/8.058	419.229/25.884	480.056/150.264	146.954	5.387	3.20	31.00
kitti00	33.185/77.873	679.592/80.148	733.914/688.287	203.045	6.163	1.63	7.81
sphere2500	29.589/85.870	5474.351/207.284	5965.062/281.754	655.016	19.558	16.32	38.36
parking-garage	11.717/13.589	212.725/20.302	243.867/147.649	102.989	30.602	20.86	64.29
10kHog-man	201.498/246.005	5765.850/552.462	5955.160/1431.893	3102.154	401.587	7.11	46.76
100k	4h53m/18h22m	–/22h03m	–/40h24m	200h28m	26h15m	4.26	25.30

Table 4. Error evaluation. †Note that the *parking-garage* dataset does not come with ground truth and a de-facto ground truth was obtained by batch solving until convergence.

Dataset	Mode	RMSE error					
		ATE		RPE		RPE all-all	
		Translation	Rotation	Translation	Rotation	Translation	Rotation
ellipse3D	APAL	0.173	3.666	0.061	1.643	0.521	3.121
	FPFL	0.338	5.815	0.100	3.156	0.839	5.778
ellipseN	APAL	0.138	1.599	0.039	1.102	0.338	1.535
	FPFL	0.235	4.820	0.095	2.737	0.841	4.033
kitti00	APAL	3.046	2.251	0.037	0.143	7.007	2.188
	FPFL	3.093	2.250	0.029	0.119	7.297	2.223
	RFPFL	12.017	3.702	0.028	0.118	17.448	3.623
sphere2500	APAL	0.203	1.397	0.166	1.582	0.905	1.391
	FPFL	0.457	2.440	0.244	2.179	1.580	2.397
parking-garage†	APAL	0.193	0.787	0.016	0.270	2.563	0.777
	FPFL	0.661	0.757	0.012	0.109	1.167	0.647
	RFPFL	6.197	9.761	0.247	1.688	9.910	7.264
10kHog-man	APAL	0.917	3.325	0.081	1.612	3.707	3.318
	FPFL	1.479	4.902	0.139	2.583	5.754	4.896
100k	APAL	0.913	0.534	0.009	0.270	1.644	0.522
	FPFL	1.205	0.814	0.020	0.473	2.559	0.802

ratio, even though the threshold applied is very low (it only accepts all the loops if zero threshold is specified). Also note that the error is worse than that of the distance-based approach.

Since the results on the Oxford dictionary were worse than ours, we decided to train a new dictionary specifically for the *kitti* dataset. We tried to match the procedure described in Cummins and Newman (2010). We used 2.5 million SURF features extracted from the frames of left camera sequences *kitti01* – *kitti21*, deliberately skipping *kitti00*. We used frames spaced approximately every 20 m (only 1951 out of all the frames). K-means clustering was trained on Intel Core i5-4590 with 8 GB RAM while Chow-Liu tree was computed on dual Intel Xeon E5-2665 (16 cores in total) with 64 GB RAM because of its high memory requirements. The calculation of the K-means took 1 day 06:00:30 and required 2.5 GB of memory, The subsequent calculation of Chow-Liu tree took 12:42:01 and required 25 GB or memory (using the compact method – the fast one would require 45 GB of RAM).

On Figure 5 (bottom) there is a plot of the number of *FAB-MAP2* candidates and the solution error, depending on the probability threshold, using the *kitti* dictionary. Note that the number of loops is still relatively constant and again changes abruptly at the borders of the plot. Note that using a custom dictionary did not help closing all the loops with good inlier ratio, even with low thresholds. The error improved somewhat, compared to the Oxford dictionary.

5.5. Conservativeness of the compact pose estimate

An important property of the compact SLAM algorithm is conservativeness of the computed pose estimates. If the algorithm produces an over-confident estimate, the robot poses could be imprecise by more than what their covariance suggests, which could lead to bad decisions in data association, loop closure detection, motion planning, etc. If the estimate is, on the other hand, over-conservative, the result would be equally difficult to use.

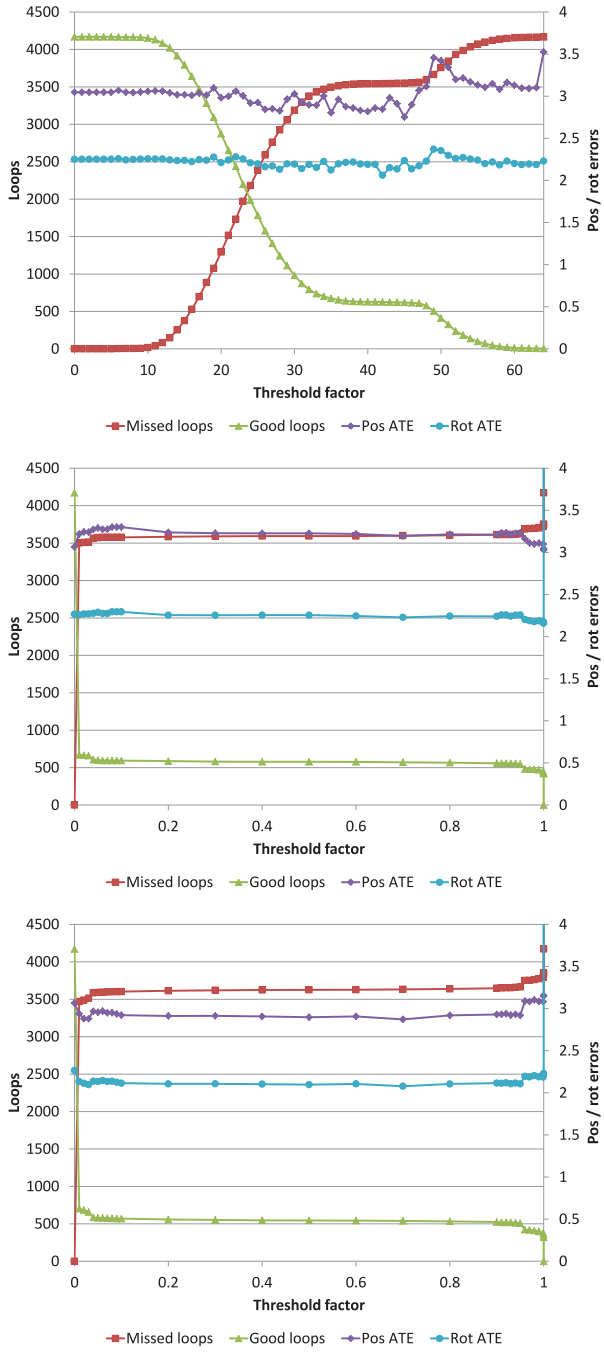

Fig. 5. Comparison of sensitivity of different loop closing methods on their respective thresholds, distance-based loop closing (top), *FAB-MAP2* on the Oxford dictionary provided by the authors (middle) and *FAB-MAP2* on the dictionary we trained on the *kitti* dataset (bottom).

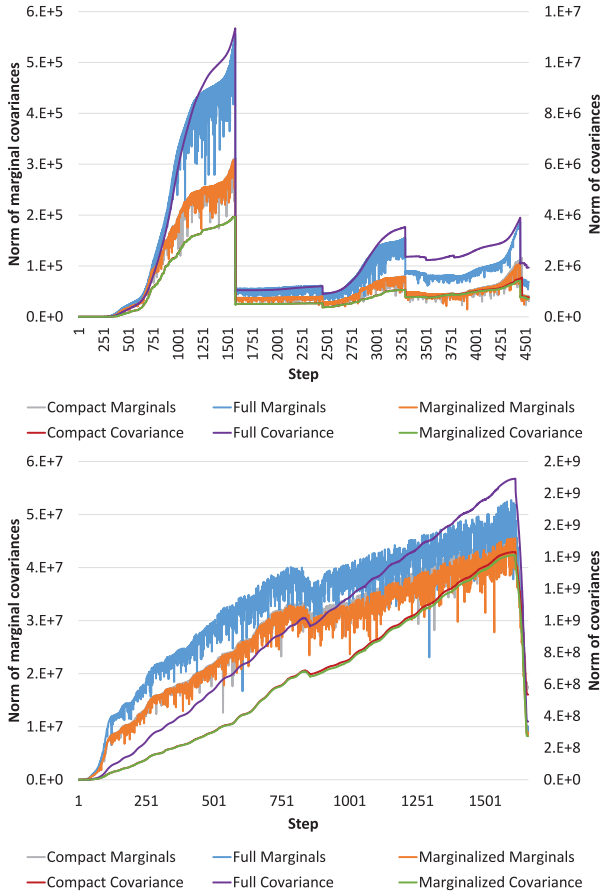

Fig. 6. Comparison of the norms of covariances of the incremental solutions of the *kitti00* (top) and *parking-garage* (bottom) datasets. Best viewed in color.

To evaluate the conservativeness of the estimate, the norms of marginal covariances of all the variables in the system are calculated, at each step. These indicate the uncertainty in the poses of the trajectory. Additionally, the norms of full covariances (i.e. both marginal and cross-covariances) are calculated at each step. These add more information about the correlation of the variables in the system. These two norms are calculated for three scenarios: the compact SLAM algorithm (FPFL), the SLAM algorithm including all the possible poses and loops (APAL) and also a variant of APAL where the redundant poses not present in the compact representation are marginalized out using the Schur Complement. This essentially compares the effects of *measurement composition* in the case of compact SLAM with the effects of *variable marginalization*.

Figure 6 plots the evolution of the covariance norms for the *kitti00* and *parking-garage* datasets. The highest norm corresponds to the APAL SLAM. This is followed by the FPFL and the marginalized system. This indicates that the compact SLAM is slightly more conservative than marginalization of the variables not present in the compact system. A similar result is obtained when comparing the norms of marginal covariances – again, the full system has the greatest norm of marginal covariances and the compact and marginalized systems have approximately the same norms. The slight difference in norm stems from the fact that the covariances of the composed measurements are calculated using an approximate function as described

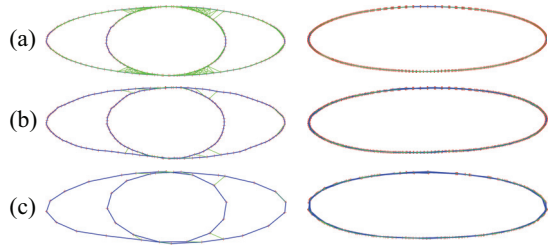

Fig. 7. *Ellipse* datasets; poses (red), and loop closures (green); (a) using all possible loop closures, (b) using only relevant loop closures and (c) compact pose SLAM with reduction of the number of poses.

in Smith and Cheeseman (1986). The same evaluation was performed on the other datasets as well, supporting the same conclusions, but were omitted from this paper to save space.

5.6. Performance and accuracy analysis

A compact representation of the SLAM problem translates into more efficient estimation both from the point of view of the memory occupied by the state as well as from the point of view of the execution time. To validate this we compare the cumulative time at the end of the processing of each of the datasets mentioned above. We first define the configuration of the thresholds used to obtain the APAL and FPFL solutions for each of the datasets.

For example, for the *ellipse3D* dataset, following the procedure described at the end of subsection 5.2, we set $v = [2.54\text{ m}, 2.54\text{ m}, 2.54\text{ m}, 0.36\ °]$, $s = 0.125$, $g_{\text{pose}} = -\infty$ and $g_{\text{loop}} = -\infty$ to obtain an estimation problem which includes all poses and all loops (APAL). We then increased the $g_{\text{loop}} = 5.50$ to obtain an estimation problem which contains all the poses but only the informative links (APFL). Similarly, increasing $g_{\text{pose}} = 5.74$ leads to a compact estimation problem, yet containing maximal amount of information. The resulting trajectories in all three cases are shown in Figure 7 left, and the timing and accuracy are reported in the Tables 3 and 4, respectively. Out of 377 possible loop closure links, the algorithm selected only 11 relevant ones, resulting in a factor of 7.6× reduction in the run time. Nevertheless, only 0.228% of the accuracy in position and 1.140% in rotation is lost when reducing the number of links to only informative.

Similar tests were performed with the rest of the datasets. This paper reports only the APAL and FPFL cases, being the most relevant in our comparisons. Figure 3 shows the solutions of several SLAM datasets processed by allowing all poses and all loop closures to be added to the state representation Figure (3(a)) and by selecting only informative links and non-redundant poses in an incremental processing Figure (3(b))). The corresponding timing results are shown in Figure 3. We can see that for all the datasets there is a considerable time reduction when performing compact SLAM. Figure 4 on the other hand, shows that the translational and rotational errors increase only slightly, in the case of the compact SLAM. To provide some perspective on the values of the errors, we also performed *random* selection of poses and loop closures on the *kitti00* and *parking-garage* datasets, in order to get a solution with the same sparsity as the one in the FPFL case. Those results are denoted RFPFL and it is visible that they are much worse in both cases.

Runtime evaluation is provided in Figure 3. The APAL and FPFL strategies are integrated into the SLAM++ library and compared against the solution of the SLAM++ (Polok et al., 2013b), g2o (Kümmerle et al., 2011), and iSAM (Kaess et al., 2008) solvers with neither loop detection nor compact representation. The time required to obtain the marginal covariances is also provided, except for the 100*k* dataset processed with g2o and iSAM where it takes several days. The plain nonlinear least squares solver and marginalisation times are used only as reference. Note that there is an important difference on how the incremental processing is performed in APAL and FPFL strategies. While in plain nonlinear solving (SLAM++, g2o and iSAM columns in Figure 3) the incremental updates occur *every new vertex*, in APAL and FPFL the updates happen *every new measurement* (see Algorithm 1, line 24). This is due to the fact that the mutual information of every measurement is calculated. In general, the number of edges in the system is much higher than the number of vertices. Nevertheless, due to highly efficient block matrix solvers and covariance recovery algorithms implemented into SLAM++ library, the APAL strategy has comparable runtime and has the benefit of providing state-based loop closure detection. At the same time, the FPFL strategy remains efficient by maintaining a compact representation of the state, but at the same time integrating the state-based loop closure detection technique. Figure 3 also reports percentage of loops and poses kept in the compact representation.

The compact pose SLAM was also tested on a multiple loops dataset. Similar to *ellipse3D* we created *ellipseN* dataset which loops N-times around an ellipse with semi-axes of 20 m and 6 m, respectively, see Figure 7 right. Figure 8 (top) shows the evolution of number of poses and loop closures over $N = 10$ loops around the ellipse. We can see that, while in APAL the number of loop-closures increases exponentially, the FPFL strategy maintains a linear trend with a low slope increase in number of both poses and loops. While looping 10 times took 146.954 s to run incrementally with the APAL strategy, it took only 5.387 s with the compact representation. Figure 8 (bottom) shows the cumulative time for APAL and FPFL strategies, and includes the solving and search for loop closures times. While APAL runs in polynomial time the FPFL runs in linear time.

We have also evaluated the memory requirements of the algorithm, again on the *ellipseN* dataset. Figure 9 shows the evolution of the number of nonzero elements in the information matrix and its factorization at each step. Note that

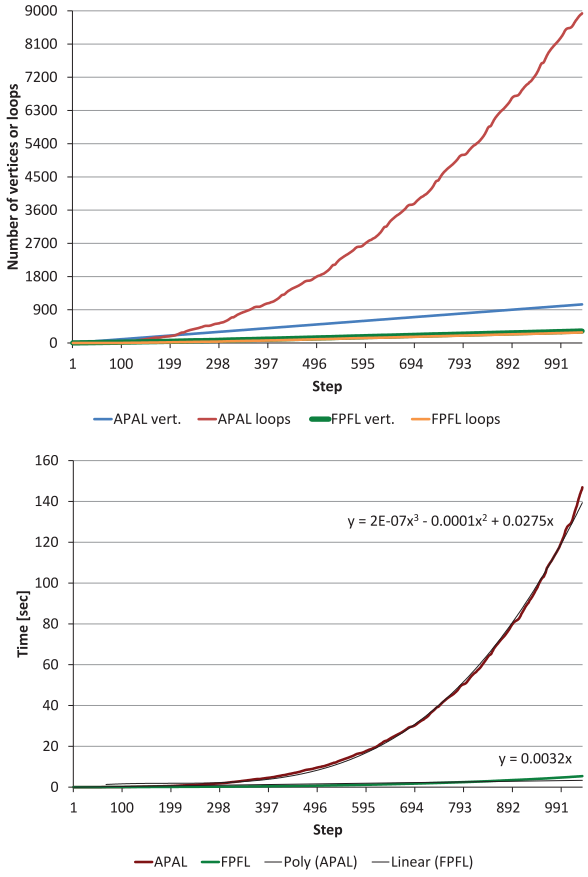

Fig. 8. Evolution of the number of poses and loop closures for both strategies APAL and FPFL (top) and cumulative time of distance based loop-closing (bottom) on the *ellipseN* dataset.

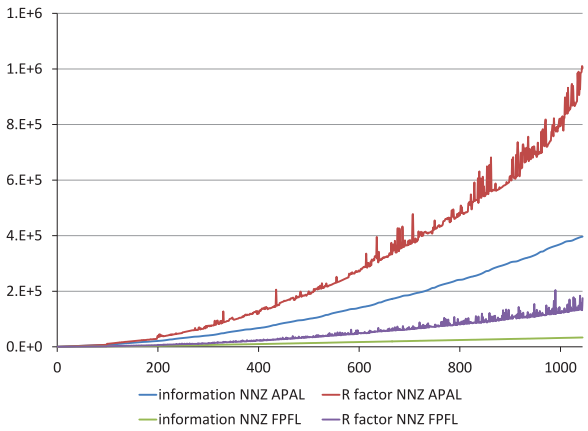

Fig. 9. Evolution of number of nonzero system matrix entries on the *ellipseN* dataset.

for all poses all loops, the size of both matrices grows exponentially while in the compact case the size of the information matrix follows a linear trend. The number of nonzeros in the R factor is slightly higher due to fill-in and is a bit noisy due to the incremental reordering strategy described in Polok et al. (2013b).

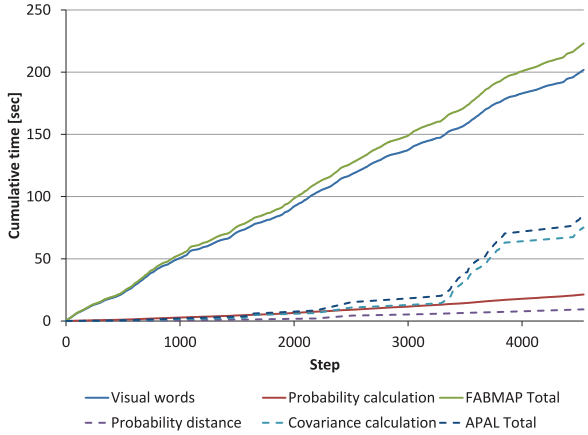

Fig. 10. Comparison of cumulative time of appearance-based data association on the *kitti00* dataset using *FAB-MAP2* and the Oxford dictionary provided by the authors (solid lines) and distance-based data association using the proposed method (dashed lines).

The distance loop closing strategy was compared to appearance-based loop-closing implemented in the *FAB-MAP2* library (Cummins and Newman, 2010). Figure 10 shows the execution time comparison of the two methods. For FAB-MAP2, the plot shows a sum of the time of transforming the feature descriptors into visual words and the match probability matrix calculation. We have used an incremental approach, as described by the authors, and at each step, one new column of the match probability matrix is added. Note that the time complexity is dominated by the visual words formation, which in turns depends on the number of features detected in the image and on the size of the vocabulary. For the distance based approach, we time the equivalent computation, consisting of finding the loop closure candidates and calculating the information gains for them.

For the *kitti00* dataset, our method significantly outperforms FAB-MAP2, even if including the time it takes to calculate the covariances. Note that this test was performed on the Oxford dictionary which was provided by the authors of FAB-MAP2. For the following tests, we have also trained our own dictionary on the *kitti* dataset, as described below. The loop closing run time with this vocabulary has the same complexity with higher constant factor. The final total time with this dictionary is 2107.71 s (out of that 2043.00 s visual words and 64.72 s probability calculation) and was omitted from Figure 10 otherwise the bottom part of the plot would be illegible.

6. Conclusions

This paper addressed both the efficiency and temporal scalability of the online SLAM. SLAM++ nonlinear least square solver based on efficient sparse block matrix operations has already proven its superiority over the existing solutions for incremental processing in SLAM (Polok et al., 2013b). At the same time, in our latest work (Ila et al.,

2015), we showed how the uncertainty of the estimate can be calculated incrementally in a very efficient manner.

This paper integrates all of the above mentioned characteristics into a complete SLAM algorithm which not only maintains a scalable representation of the state but also efficiently contributes to the data association process without a significant computational overhead. Information theory measures play an important role in the proposed technique, allowing for principled methods to select only informative links and non-redundant poses. The proposed system automatically limits the growth of the map representation when continuously operating in the same environment. The results significantly outperform the state of the art in processing speed while maintaining an accurate estimation. The proposed method for distance-based loop closure detection has the benefit of being applicable to modalities of sensors other than image, being fast at the same time.

In addition to that, we also discuss several methods for efficiently recovering an estimate of the full state from the compact representation. While here they are applied incrementally, it would also be possible to apply them in batch solving in order to convert the problem to a much smaller one by condensing all the vertices of order two. The full solution can then be efficiently reconstructed in linear time.

While the information theoretic measures provide a solid foundation for selecting informative links and non-redundant poses, there is also a question of robustness to outliers. Intuitively, outlier measurements are likely to have high mutual information, but at the same time they are undesirable. In the continuation of this work, we will aim to verify this claim and integrate robustness in the process of maintaining a compact and scalable representation of the SLAM problem. Furthermore, the proposed algorithm can easily be adapted to landmark SLAM and even to structure from motion allowing 3D mapping of large scale environments.

Funding

The author(s) disclosed receipt of the following financial support for the research, authorship, and/or publication of this article: Dr. Viorela Ila was supported by the ARC Centre of Excellence for Robotic Vision, project number CE140100016.

The authors from Brno University of Technology were supported by the European Union, 7^{th} Framework Programme grant 316564-IMPART and the IT4-Innovations Centre of Excellence project (CZ.1.05/1.1.00/02.0070), funded by the European Regional Development Fund and the national budget of the Czech Republic via the Research and Development for Innovations Operational Programme, as well as Czech Ministry of Education, Youth and Sports via the project Large Research, Development and Innovations Infrastructures (LM2011033).

Note

1. Not to be confused with "*SLAM++: Simultaneous Localization and Mapping at the Level of Objects*" proposed by Salas–Moreno et al. (2013). Each software was developed and named independently and simultaneously. Our incremental SLAM framework was first introduced at ICRA 2013 (May 6–10) during the interactive presentation of our seminal work in this direction (Polok et al., 2013c).

References

Agarwal S and Mierle K (2012) Ceres solver. Available at: http://code.google.com/p/ceres-solver/(accessed 20 August 2015).

Agarwal S, Snavely N, Simon I, et al. (2009) Building Rome in a day. In: *International conference on computer vision (ICCV)*, Kyoto, Japan, 29 September-2 October, pp. 72–79, IEEE.

Barfoot T and Furgale P (2014) Associating uncertainty with three-dimensional poses for use in estimation problems. *IEEE Transactions on Robotics* 30(3): 679–693.

Beall C, Lawrence B, Ila V, et al. (2010) 3D reconstruction of underwater structures. In: *IEEE/RSJ international conference on intelligent robots and systems (IROS)*, 18–22 October, pp. 4418–4423, IEEE.

Björck A (1996) *Numerical methods for least squares problems*. SIAM. ISBN: 978-0-89871-360-2.

Blanco JL (2010) A tutorial on SE(3) transformation parameterizations and on-manifold optimization. Technical report, University of Malaga, Spain.

Carlevaris–Bianco N and Eustice RM (2014a) Generic node removal for factor-graph SLAM. *IEEE Transactions on Robotics* 30(6): 1371–1385.

Carlevaris–Bianco N and Eustice RM (2013) Generic factor-based node marginalization and edge sparsification for pose-graph SLAM. In: *IEEE international conference on robotics and automation (ICRA)*, Karlsruhe, Germany, 6–10 May, pp. 5728–5735, IEEE.

Carlevaris–Bianco N and Eustice RM (2014b) Conservative edge sparsification for graph SLAM node removal. In: *IEEE international conference on robotics and automation (ICRA)*, Hong Kong, 31 May–7 June, pp. 854–860, IEEE.

Chow C and Liu C (1968) Approximating discrete probability distributions with dependence trees. *IEEE Transactions on Information Theory* 14(3): 462–467.

Cummins M and Newman P (2010) Appearance-only SLAM at large scale with FAB-MAP 2.0. *The International Journal of Robotics Research* 30(9): 1100–1123.

Davis T (2006a) Csparse. Available at: http://www.cise.ufl.edu/research/sparse/CSparse/ (accessed 20 August 2015).

Davis TA (2006b) *Direct methods for sparse linear systems (fundamentals of algorithms 2)*. SIAM. ISBN: 978-0-89871-613-9.

Davis TA and Hager WW (1997) Modifying a sparse cholesky factorization. *SIAM Journal on Matrix Analysis and Applications* 20(3): 606–627. SIAM.

Davison A and Murray D (2002) Simulataneous localization and map-building using active vision. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 24(7): 865–880.

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

Dissanayake G, Williams SB, Durrant–Whyte H, et al. (2002) Map management for efficient simultaneous localization and mapping (SLAM). *Autonomous Robots* 12(3): 267–286.

Eustice R, Singh H, Leonard J, et al. (2006) Visually mapping the RMS Titanic: Conservative covariance estimates for SLAM information filters. *The International Journal of Robotics Research* 25(12): 1223–1242.

Geiger A, Lenz P, Stiller C, et al. (2013) Vision meets robotics: The KITTI dataset. *The International Journal of Robotics Research* 32(11): 1231–1237.

Golub GH and Plemmons RJ (1980) Large-scale geodetic least-squares adjustment by dissection and orthogonal decomposition. *Linear Algebra and its Applications* 34: 3–28.

Grisetti G, Stachniss C, Grzonka S, et al. (2007) A tree parameterization for efficiently computing maximum likelihood maps using gradient descent. In: *Robotics: Science and Systems (RSS)*, Atlanta, Georgia, 27–30 June, pp. 27–30.

Hager WW (1989) Updating the inverse of a matrix. *SIAM Review* 31(2): 221–239.

Haner S and Heyden A (2012) Covariance propagation and next best view planning for 3D reconstruction. In: *European conference on computer vision (ECCV)*, Firenze, Italy, 7–13 October, pp.545–556, Berlin Heidelberg: Springer.

Huang G, Kaess M and Leonard J (2013) Consistent sparsification for graph optimization. In: *European conference on mobile robots (ECMR)*, Barcelona, Spain, 25–27 September, pp. 150–157, IEEE.

Huang GP, Mourikis A and Roumeliotis S (2011) An observability-constrained sliding window filter for SLAM. In: *IEEE/RSJ international conference on intelligent robots and systems (IROS)*, 25–30 September, pp.65–72, San Francisco, CA, USA: IEEE.

Ila V, Polok L, Šolony M, et al. (2015) Fast covariance recovery in incremental nonlinear least square solvers. In: *IEEE international conference on robotics and automation (ICRA)*, 6–10 May, pp.4636–4643, Karlsruhe, Germany: IEEE.

Ila V, Porta JM and Andrade-Cetto J (2010) Information-based compact pose SLAM. *IEEE Transactions on Robotics* 26(1): 78–93.

Indelman V, Roberts R, Beall C, et al. (2012) Incremental light bundle adjustment. In: *British machine vision conference (BMVC)*, 3–7 September, pp.134.1–134.11. Guildford, UK: BMVA Press.

Johannsson H, Kaess M, Fallon M, et al. (2013) Temporally scalable visual SLAM using a reduced pose graph. In: *IEEE international conference on robotics and automation (ICRA)*, Karlsruhe, Germany, 6–10 May, pp. 54–61, IEEE.

Kabsch W (1976) A solution for the best rotation to relate two sets of vectors. *Acta Crystallographica Section A: Crystal Physics, Diffraction, Theoretical and General Crystallography* 32(5): 922–923.

Kaess M and Dellaert F (2009) Covariance recovery from a square root information matrix for data association. *Robotics and Autonomous Systems* 57(12): 1198–1210, Elsevier.

Kaess M, Ila V, Roberts R, et al. (2010) The Bayes tree: An algorithmic foundation for probabilistic robot mapping. In: *International workshop on the algorithmic foundations of robotics*, NUS, Singapore, 13–15 December, pp. 150–157, Berlin Heidelberg: Springer.

Kaess M, Johannsson H, Roberts R, et al. (2011a) iSAM2: Incremental smoothing and mapping with fluid relinearization and incremental variable reordering. In: *IEEE international conference on robotics and automation (ICRA)*, Shanghai, China, 9–13 May, pp. 3281–3288, IEEE.

Kaess M, Johannsson H, Roberts R, et al. (2011b) iSAM2: Incremental smoothing and mapping using the Bayes tree. *The International Journal of Robotics Research* 31: 217–236.

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

Klein G and Murray D (2007) Parallel tracking and mapping for small AR workspaces. In: *IEEE and ACM international symposium on mixed and augmented reality (ISMAR)*, Nara, Japan, 13–16 November, pp.225–234, ACM.

Konolige K (2010) Sparse sparse bundle adjustment. In: *British machine vision conference (BMVC)*, Aberystwyth, Wales, 31 August–3 September, pp. 102.1–102.11

Konolige K, Grisetti G, Kümmerle R, et al. (2010) Efficient sparse pose adjustment for 2D mapping. In: *IEEE/RSJ international conference on intelligent robots and systems (IROS)*, Taipei, Taiwan. 18–22 October, pp. 22–29, IEEE.

Kretzschmar H and Stachniss C (2012) Information-theoretic compression of pose graphs for laser-based SLAM. *The International Journal of Robotics Research* 31(11): 1219–1230.

Kretzschmar H, Stachniss C and Grisetti G (2011) Efficient information-theoretic graph pruning for graph-based SLAM with laser range finders. In: *IEEE/RSJ international conference on intelligent robots and systems (IROS)*, 25–30 September, pp.865–871, San Francisco, CA, USA: IEEE.

Kümmerle R, Grisetti G, Strasdat H, et al. (2011) g2o: A general framework for graph optimization. In: *Proceedings of the IEEE international conference on robotics and automation (ICRA)*, Shanghai, China, 9–13 May, pp. 3607–3613, IEEE.

Kummerle R, Steder B, Dornhege C, et al. (2009) On measuring the accuracy of SLAM algorithms. *Autonomous Robots* 27(4): 387–407, US: Springer.

Neira J and Tard os J (2001) Data association in stochastic mapping using the joint compatibility test. *IEEE Transactions on Robotics and Automation* 17(6): 890–897.

Polok L, Ila V and Smrž P (2013a) Cache efficient implementation for block matrix operations. In: *Proceedings of the high performance computing symposium*, San Diego, CA, USA, 7–10 April, pp.698–706. ACM.

Polok L, Ila V, Šolony M, Smrž P, et al. (2013b) Incremental block Cholesky factorization for nonlinear least squares in robotics. In: *Robotics: Science and Systems (RSS)*. Berlin, Germany, 24–28 June, pp. 42.1–42.8

Polok L, Šolony M, Ila V, et al. (2013c) Efficient implementation for block matrix operations for nonlinear least squares problems in robotic applications. In: *IEEE international conference on robotics and automation (ICRA)*, 6–10 May, pp. 2263–2269, Karlsruhe, Germany: IEEE.

Prentice S and Roy N (2011) The belief roadmap: Efficient planning in linear POMDPs by factoring the covariance. In: *Proceedings of the international symposium of robotics research (ISRR)*, Flagstaff, AZ, USA, 28 August–1 September, pp.293–305. Heidelberg: Springer.

Salas–Moreno RF, Newcombe RA, Strasdat H, et al. (2013) SLAM++: Simultaneous localisation and mapping at the level of objects. In: *IEEE conference on computer vision and*

pattern recognition (CVPR), Portland, OR, USA, 25–27 June, pp.1352–1359, IEEE.

Sibley G, Matthies L and Sukhatme G (2008) A sliding window filter for incremental SLAM. In: Kragic D and Kyrki V (eds) *Unifying Perspectives in Computational and Robot Vision*, *Lecture Notes in Electrical Engineering*, volume 8. Springer, pp.103–112. US: Springer.

Sim R (2005) Stable exploration for bearings-only SLAM. In: *IEEE international conference on robotics and automation (ICRA)*, Barcelona, Spain, 18–22 April, pp.2422–2427, IEEE.

Smith RC and Cheeseman P (1986) On the representation and estimation of spatial uncertainly. *The International Journal of Robotics Research* 5(4): 56–68.

Sturm J, Engelhard N, Endres F, et al. (2012) A benchmark for the evaluation of RGB-D SLAM systems. In: *IEEE/RSJ international conference on intelligent robots and systems (IROS)*, Algarve, Portugal, 8–11 October, pp. 573–580, IEEE.

Thrun S, Liu Y, Koller D, et al. (2004) Simultaneous localization and mapping with sparse extended information filters. *The International Journal of Robotics Research* 23(7–8): 693–716.

Tipaldi GD, Grisetti G and Burgard W (2007) Approximate covariance estimation in graphical approaches to SLAM. In: *IEEE/RSJ international conference on intelligent robots and systems (IROS)*, San Diego, CA, USA, 29 October-2 November, pp.3460–3465. IEEE.

Valencia R, Morta M, Andrade–Cetto J, et al. (2013) Planning reliable paths with pose SLAM. *IEEE Transactions on Robotics* 29(4): 1050–1059.

Vidal–Calleja T, Davison A, Andrade-Cetto J, et al. (2006) Active control for single camera SLAM. In: *IEEE international conference on robotics and automation (ICRA)*, Orlando, FL, USA, 15–19 May, pp.1930–1936.

Appendix: Index to multimedia extensions

The multimedia extensions to this article are at: http://www.ijrr.org

Extensions	Media type	Description
1	Video	Incremental compact SLAM
2	Code	SLAM++ library
3	Scripts	To run compact SLAM
4	Scripts	To set thresholds
5	Data	All datasets