Sparse local submap joining filter for building large-scale maps

Shoudong Huang, Member, IEEE, Zhan Wang, and Gamini Dissanayake, Member, IEEE

May 27, 2008

Abstract

This paper presents a novel local submap joining algorithm for building large-scale feature based maps – Sparse Local Submap Joining Filter (SLSJF). The input to the filter is a sequence of local submaps – each local submap is represented in a coordinate frame decided by the robot pose at which the map is initiated. The local submap state vector consists of the positions of all the local features and the final robot pose within the submap. The output of the filter is a global map containing all the feature positions as well as all the robot end poses when building the local submaps.

The Extended Information Filter (EIF) is used in the SLSJF and the resulting information matrix is exactly sparse. The sparse structure together with a novel state vector and covariance submatrix recovery technique make the SLSJF computationally very efficient. The SLSJF is a canonical and efficient submap joining solution for large-scale Simultaneous Localization and Mapping (SLAM) problems as long as consistent local submaps can be generated by some reliable SLAM algorithm. The effectiveness and efficiency of the new algorithm is verified through computer simulations and experiments.

Index Terms

Simultaneous localization and mapping (SLAM), Extended Kalman Filter, Extended Information Filter, Map joining, Sparse matrix.

Corresponding Author – Shoudong Huang
Email: sdhuang@eng.uts.edu.au

Paper type — Regular paper
Sparse local submap joining filter for building large-scale maps

I. INTRODUCTION

In the recent years, many different Simultaneous Localization and Mapping (SLAM) algorithms have been proposed to reduce the computational effort required for solving large-scale SLAM problems (see e.g. [1] and the references therein). Especially, sparse representations for solving SLAM problems have attracted special attention since the development of the Sparse Extended Information Filter (SEIF) by Thrun et al. [2]. It has been shown that significant computational advantages can be achieved by exploiting the sparseness of the information matrix or techniques from sparse graph and sparse linear algebra ([3]-[6]). However, most of the methods based on sparse representation have focused on building a single large-scale map, resulting in the need to update a large map whenever a new observation is made.

Local submap joining [7][8] provides an efficient way to build large-scale maps. The idea of local submap joining is to first build a sequence of small sized local submaps (e.g. by traditional Extended Kalman Filter (EKF) SLAM [10]), and then combine the local submaps into a large-scale global map. In the map joining process of [7], the state of the local submap is first transferred into the global coordinate frame and added into the global map state, then the common features presented in both the local and global maps are identified and an EKF estimator is used to enforce the constraints. The resulting map covariance matrix is fully correlated and thus the map fusion process is computationally demanding. Notably a key advantage of local submap joining is that it significantly reduces the frequency of global map update [8].

In this paper, it is shown that local submap joining can be achieved through the use of a sparse information filter. The proposed map joining filter, Sparse Local Submap Joining Filter (SLSJF), combines the advantages of the local submap joining algorithms and the sparse representation of SLAM. It is demonstrated that the computational cost of the global map construction can be substantially reduced by exploiting the sparseness of the information matrix.

The paper is organized as follows. Section II describes the overall structure of the SLSJF and explains the reason why the local submap joining filter can be formulated such that it results in an exactly sparse information matrix. The SLSJF algorithm is stated in detail in Section III. Section IV provides simulation and experiment results. Section V discusses some properties of the SLSJF and some related work. Section VI concludes the paper.

II. THE OVERALL STRUCTURE OF SLSJF

This section describes the overall structure of the SLSJF and explains why the sparse representation can be achieved.

A. The input and output of SLSJF

The input to the SLSJF is a sequence of local submaps. Here it is assumed that a consistent local submap can be constructed by some SLAM algorithm and is expressed by

\[
(\hat{X}^L, P^L)
\]

where \(\hat{X}^L\) (here the superscript ‘L’ stands for the local map) is an estimate of the state vector

\[
X^L = (X^L_0, X^L_1, \ldots, X^L_n)
\]

and \(P^L\) is the associated covariance matrix. The state vector \(X^L\) contains the final robot pose \(X^L_k\) and all the local feature positions \(X^L_1, \ldots, X^L_n\) (as in traditional EKF SLAM). The coordinate system of a local submap is defined by the initial robot pose when the building of the local submap is started, i.e. the robot starts at the coordinate origin of the local submap.

It is assumed that the robot end pose in local map \(k\) is the same as the robot start pose in local map \(k+1\) (Fig. 1). This assumption is valid if the robot starts to build local map \(k+1\) as soon as it finishes local map \(k\).

The output of SLSJF is a global map. The global map state vector contains all the feature positions and all the robot end poses involved in each local map (all the filled features and robot poses in Fig. 1).

B. Why can local submap joining have sparse representation?

The reason why Sparse Local Submap Joining Filter (SLSJF) can be developed is that the information contained in each local map is the relative position information about some “nearby objects” – the features and the robot start/end poses involved in the local map.

By including all the objects involved (all the features and all the robot start/end poses) in the global map state vector, the local submap joining problem becomes a large-scale estimation problem with only “local” information (similar to SAM [3] and full SLAM [6]). When Extended Information Filter (EIF) is applied to the estimation problem, an off-diagonal element of the information matrix is non-zero only when the two related objects are within the same local map. Since the size of each local map is limited, any object will only link to its “nearby objects” no matter how many (overlapping) local maps are fused (Figure 1). This results in an exactly sparse information matrix without any approximation.

Since all the objects involved in the local maps are included in the global state vector, no marginalization is required in the map joining process and thus the information matrix will keep exactly sparse all the time. Because most of the robot poses are marginalized out during the local map building process,
the dimension of the global state vector is much less than that of SAM [3] and full SLAM [6].

![Diagram of SLSJF](image)

Fig. 1. The idea of SLSJF: Each object (e.g. the feature *) is only linked to its "nearby objects" (features and robot poses that share the same local map with it)

C. The overall structure of SLSJF

Similar to the sequential map joining in [7][8], SLSJF fuses the local submaps one by one. The overall structure of SLSJF is shown in Algorithm 1.

**Algorithm 1 Overall structure of SLSJF**

**Require:** A sequence of local maps: each local map contains a state vector estimate and a covariance matrix
1: Set local map 1 as the global map
2: For $k = 2 : p$ (p is the total number of local maps),
   fuse local map k into the global map
3: End

III. THE SLSJF ALGORITHM

This section states the EIF based SLSJF algorithm in detail including global map initialization and update, reordering of the global state vector, state vector and covariance submatrix recovery, and data association.

A. State vector of the global map

The state vector of the global map in SLSJF contains feature positions and robot end poses when building each of the local submaps. For convenience, the origin of the global map is chosen to be the same as the origin of local map 1 (Fig. 1).

After local maps 1 to $k$ are fused into the global map, the global state vector is denoted as $X^G(k)$ (here the superscript ‘G’ stands for the global map) and is given by

$$X^G(k) = (X^G_1, \ldots, X^G_{n_1}, X^G_{e1}, X^G_{n_1+1}, \ldots, X^G_{n_1+n_2}, X^G_{e2}, \ldots, X^G_{n_k+\cdots+n_k-1+1}, \ldots, X^G_{n_k+\cdots+n_k-1+n_k}, X^G_{ke})$$

where $X^G_1, \ldots, X^G_{n_1}$ are the global positions of the features in local map 1; $X^G_{n_1+1}, \ldots, X^G_{n_1+n_2}$ are the global positions of those features in local map 2 but not in local map 1; $X^G_{n_1+\cdots+n_k-1+1}, \ldots, X^G_{n_k+\cdots+n_k-1+n_k}$ are the global positions of the features in local map $k$ but not in local maps 1 to $k-1$. $X^G_{e} = (x^G_{e}, y^G_{e}, \phi^G_{e})$ (1 $\leq i \leq k$) is the global position of the robot end pose in local map $i$, which is also the robot start pose in local map $i+1$. Here the subscript ‘e’ stands for ‘end pose’.

The EIF is used in the SLSJF algorithm. Instead of the global state vector estimate $\hat{X}^G(k)$, the corresponding covariance matrix $P(k)$ used in EKF, an information vector $i(k)$ and an information matrix $I(k)$ are used in EIF to express the Gaussian distribution. The relationship between $\hat{X}^G(k)$, $P(k)$ and $i(k)$, $I(k)$ is ([6])

$$I(k)\hat{X}^G(k) = i(k), \quad P(k) = I(k)^{-1}. \quad (4)$$

In SLSJF, $I(k)$ is an exactly sparse matrix and can be stored and computed efficiently. The state vector estimate $\hat{X}^G(k)$ is recovered after fusing each local map by solving the sparse linear equation (the first equation in (4)). However, the whole dense matrix $P(k)$ is neither computed nor stored in SLSJF. For data association, only a small part of $P(k)$ is computed by solving sparse linear equations (Section III-C.3).

When fusing local map $k+1$ into the global map, the features that are in local map $k+1$ but have not been included in the global map yet, $X^G_{n_k+\cdots+n_k+1}, \ldots, X^G_{n_k+\cdots+n_k+n_k}$, together with the robot end pose of local map $k+1$, $X^G_{(k+1)e}$, are added into the global state vector. So the new state vector becomes

$$X^G(k+1) = (X^G_1, \ldots, X^G_{n_1}, X^G_{e1}, X^G_{n_1+1}, \ldots, X^G_{n_1+n_2}, X^G_{e2}, \ldots, X^G_{n_k+\cdots+n_k-1+1}, \ldots, X^G_{n_k+\cdots+n_k-1+n_k}, X^G_{ke}, X^G_{n_k+\cdots+n_k+1}, \ldots, X^G_{n_k+\cdots+n_k+n_k}, X^G_{(k+1)e}). \quad (5)$$

B. Steps of local submap fusion

The steps used in fusing local map $k+1$ into the global map are listed in Algorithm 2. The following subsections provide the essential details.

**Algorithm 2 Fuse local map $k+1$ into global map**

**Require:** global map and local map $k+1$
1: Data association
2: Initialize the new features and $X^G_{(k+1)e}$ in the global map
3: Update the global map
4: Reorder the global map state vector when necessary
5: Compute the Cholesky Factorization of $I(k+1)$
6: Recover the global map state estimate $\hat{X}^G(k+1)$
C. Data Association

Data association finds the features in local map \( k + 1 \) that are already included in the global map and their corresponding indices in the global state vector. This is a necessary step in SLAM algorithms using practical data. However, effective data association is a step that is often neglected in many of the sparse information filter based SLAM algorithms published in the literature. Algorithm 3 and the associated explanations describe how efficient data association can be achieved in SLSIF assuming that only the geometric relationships among features present in the global and local maps are available.

Algorithm 3 Data association between local map \( k + 1 \) and the global map

Require: global map and local map \( k + 1 \)

1: Determine the set of potentially overlapping local maps
2: Find the set of potentially matched features
3: Recover the covariance submatrix associated with \( X_{ke}^G \) and the potentially matched features

1) Determine the set of potentially overlapping local maps: Since the size of local map \( k + 1 \) is small, it is not necessary to compare the features in local map \( k + 1 \) with the global map features which are far away from the robot start pose of local map \( k + 1 \), which has an estimate \( \hat{X}_{ke} \).

The local maps that potentially overlap with local map \( k + 1 \) are determined by computing the distance between \( \hat{X}_{ke}^G \) (the estimated global position of the origin of local map \( k + 1 \)) and the estimated global position of the origin of local map \( i \) (1 \( \leq i \leq k \)) (it is \((0, 0, 0)\) for local map 1 and \( \hat{X}_{i-1}^G \) for local map \( i \), \( 2 \leq i \leq k \)). If the distance is larger than the sum of the two local map radiuses (the radius of a local map is defined as the maximal distance from the local map features to its origin) plus the possible estimation error, then local map \( i \) cannot overlap with local map \( k + 1 \). Fig. 2 illustrates the idea.

2) Find the set of potentially matched features: For each feature involved in the potentially overlapping local maps, if the distance from it to \( \hat{X}_{ke} \) is larger than the radius of local map \( k + 1 \) plus the maximal possible estimation error, it cannot match with any feature in local map \( k + 1 \).

Furthermore, the (estimated) relative positions from the potentially matched features to \( \hat{X}_{ke}^G \) are computed. They are compared with each feature in local map \( k + 1 \). For each potentially matched feature, if its Euclidean distance to the closest feature in local map \( k + 1 \) is larger than a threshold, then it cannot match with any local map feature and will be removed from the list of potentially matched features.

3) Recover the covariance submatrix associated with \( X_{ke}^G \) and the potentially matched features: The covariance submatrix can be obtained by first computing the corresponding columns of the whole covariance matrix \( P(k) \) and then deleting the unnecessary rows from the columns.

By (4), the required columns of the covariance matrix \( P(k) \) can be obtained by solving a number of sparse linear equations \([24]\). In fact, the \( l \)-th column of the covariance matrix \( P(k) \), \( P_l \), can be obtained by solving the sparse linear equation

\[
I(k)P_l = e_l
\]

where

\[
e_l = [0, \ldots, 0, 1, 0, \ldots, 0]^T.
\]

Similar to SAM [3] and Tectonic SAM [23], direct linear equation solver using Cholesky factorization is applied to solve these sparse linear equations. Since the Cholesky factorization of \( I(k) \), \( L_k \), is a triangular matrix satisfying \( L_k^T L_k = I(k) \), the sparse linear equations (6) can be solved efficiently by first solving \( L_k q = e_l \) and then solving \( L_k^T P_l = q \). Note that the Cholesky factorization \( L_k \) is already available from Step 5 of Algorithm 2 when fusing local map \( k \) into the global map, as described in Section III-G.

4) Use Nearest Neighbor method [10] or Joint Compatibility Test [11] to find the match: Before comparing the features in local map \( k + 1 \) with the potentially matched global map features, all the feature positions need to be transferred into the same coordinate system. One way is to transfer the global map features into the local map frame, another way is to transfer the local map features into the global map frame.

Since both the state estimates and the covariance matrices of the related features are available, any statistical data association algorithm (such as the simple Nearest Neighbor method [10] or the more robust Joint Compatibility Test with branch and bound technique [11]) can be applied to find the match.

D. Initialize the new features and \( X_{(k+1)ke}^G \) in the global map

After the data association, the features in local map \( k + 1 \) which are new to the global map are identified. In this step, the
estimated robot start pose $\hat{X}_{ke}^G$ and the local map estimate $\hat{X}_L^L$ are used to compute the initial values of the global pose of the new features and the robot end pose $X_{(k+1)}^G$. Then insert these to $X^G(k)$ to form a new state vector estimate $\hat{X}^G(k)$. The dimensions of $i(k), I(k)$ and $L_k$ are increased by adding zeros to form a new information vector $i(k)$, a new information matrix $I(k)$, and the corresponding Cholesky factorization $L_k$.

E. Update the global map

Suppose local map $k + 1$ is given by (1). Since the local map is a consistent estimate of the relative position from robot start pose to the local features and the robot end pose, it can be treated as an observation of the true relative position with a zero-mean Gaussian observation noise whose covariance matrix is $P_L$.

To state it clearly, suppose the data association result is $X_L^L \leftarrow X_{i1}^L, \ldots, X_{in}^L \leftarrow X_{in}^G$ (including both old and new features). Then the local map state estimate $\hat{X}_L^L$ can be regarded as an observation of the true relative position from $X^G(k)$ to $X^G_{i1}, \ldots, X^G_{in}, X^G_{(k+1)}$ etc. That is,

$$z_{map} = \hat{X}_L^L = H_{map}(X^G) + w_{map}$$

(8)

where $H_{map}(X^G)$ is the relative position given by

$$
\begin{bmatrix}
(x_{(k+1)e}^G - x_{ke}^G) \cos \phi_{ke}^G + (y_{(k+1)e}^G - y_{ke}^G) \sin \phi_{ke}^G \\
(y_{(k+1)e}^G - y_{ke}^G) \cos \phi_{ke}^G - (x_{(k+1)e}^G - x_{ke}^G) \sin \phi_{ke}^G \\
\vdots \\
(x_{(k+1)e}^G - x_{ke}^G) \cos \phi_{ke}^G + (y_{(k+1)e}^G - y_{ke}^G) \sin \phi_{ke}^G \\
(y_{(k+1)e}^G - y_{ke}^G) \cos \phi_{ke}^G - (x_{(k+1)e}^G - x_{ke}^G) \sin \phi_{ke}^G \\
\vdots \\
(x_{in}^G - x_{ke}^G) \cos \phi_{ke}^G + (y_{in}^G - y_{ke}^G) \sin \phi_{ke}^G \\
(y_{in}^G - y_{ke}^G) \cos \phi_{ke}^G - (x_{in}^G - x_{ke}^G) \sin \phi_{ke}^G
\end{bmatrix}
$$

and $w_{map}$ is the zero mean Gaussian “observation noise” whose covariance matrix is $P_L$.

The formula of using the “observation” $z_{map}$ to update the information vector and the information matrix are as follows:

$$I(k + 1) = I(k) + \nabla H_{map}^T (P_L)^{-1} \nabla H_{map}$$

$$i(k + 1) = i(k) + \nabla H_{map}^T (P_L)^{-1} z_{map} - H_{map}(X^G(k)) + \nabla H_{map} X^G(k)$$

(9)

where $\nabla H_{map}(X^G)$ is the Jacobian of the function $H_{map}$ with respect to $X^G(k)$ evaluated at $X^G(k)$.

Since $z_{map} = X_L^L$ only involves two robot poses $X_{ke}^G, X_{(k+1)e}^G$ and some local features (a small fraction of the total features in the global map), the matrix $\nabla H_{map}^T (P_L)^{-1} \nabla H_{map}$ in (9) and the information matrix $I(k + 1)$ are both exactly sparse matrices.

F. Reorder the global map state vector when necessary

The purpose of reordering the global state vector is to make the computation of Cholesky factorization (Section III-G), the state vector recovery (in Section III-H), and the covariance submatrix recovery (in Section III-C.3) more efficient. Different strategies can be applied in this reordering step. The strategy proposed here is a combination of the Approximately Minimal Degree (AMD) reordering [3] and the reordering based on distances [5].

Whether to reorder the global map state vector or not depends on where the features in local map $k + 1$ are located in the global state vector. If all of the features in local map $k + 1$ are present within the $n_0$ elements at the bottom of the global state vector, then no state vector reordering is performed. If one or more of the features in local map $k + 1$ are not located within the bottom $n_0$ elements in the global state vector (this happens only when closing a big loop), then the state vector will be reordered.

The reordering of the state vector follows the following process. The robot pose $X_{(k+1)e}^G$ and the features that are within distance $d_0$ to $X_{(k+1)e}^G$ are placed at the bottom of the state vector. Their order is determined based on the distance from each feature to $X_{(k+1)e}^G$. The smaller the distance, the closer to the bottom. All the other robot poses and features are placed in the upper part of the state vector, they are reordered based on AMD.

The major advantage of reordering by AMD is that the number of fill-ins in Cholesky factorization will be reduced. The major advantage of reordering the nearby features based on distances is that once the reordering is performed, another reordering will not happen in the next few local map fusion steps. This is because the robot cannot move very far away from local map $k + 1$ in the next few steps. No matter which direction it moves along, it cannot observe features that are not located in the bottom part of the state vector until it travels a certain distance.

Once the state vector is reordered, the corresponding information matrix $I(k + 1)$ and information vector $i(k + 1)$ are reordered accordingly. They are still denoted as $I(k + 1)$ and $i(k + 1)$. Note that the Cholesky factorization of the reordered $I(k + 1)$ cannot be easily obtained from $L_k$.

G. Compute the Cholesky factorization of $I(k + 1)$

The method used to compute the Cholesky factorization of $I(k + 1)$ depends on whether the global state vector was reordered in Section III-F or not.

Case (i). If the global state vector was not reordered in Section III-F, then the Cholesky factorization of $I(k)$ (available from Step 5 of Algorithm 2 when fusing local map $k$) is used to construct the Cholesky factorization of $I(k + 1)$ as follows.

By (9), the relation between $I(k + 1)$ and $I(k)$ is

$$I(k + 1) = I(k) + \begin{bmatrix} 0 & 0 \\ 0 & \Omega \end{bmatrix}$$

(10)

where the upper-left element in $\Omega$ is non-zero. Here $\Omega$ is determined by the term $\nabla H_{map}^T (P_L)^{-1} \nabla H_{map}$ in (9). Its

1The threshold $n_0$ needs to be properly chosen in order to make the SLSJF algorithm more efficient. A smaller $n_0$ will make the incremental Cholesky factorization step (Case (i) in Section III-G) more efficient but will also increase the total number of reordering and the complete Cholesky factorization operations (see Section III-G). As a rule of thumb, $n_0$ can be chosen to be around one tenth of the dimension of the current global state vector.

2The threshold $d_0$ is related to the parameter $n_0$. It also depends on the feature density of the environment and the guideline is that the dimension of the features and robot poses that are placed at the bottom part of the state vector is around $n_0$. 
dimension is less than \( n_0 \) since otherwise the state vector would have been reordered.

Let \( I(k) \) and its Cholesky factorization \( L_k \) (a lower triangular matrix) be partitioned according to (10) as

\[
I(k) = \begin{bmatrix} I_{11} & I_{12}^T \; I_{21} \; I_{22} \end{bmatrix}, \quad L_k = \begin{bmatrix} L_{11} & 0 \; L_{21} \; L_{22} \end{bmatrix}.
\]

According to (10) and (11), \( I(k+1) \) can be expressed by

\[
I(k+1) = \begin{bmatrix} I_{11} & I_{12}^T \; I_{21} \; I_{22} \end{bmatrix} = \begin{bmatrix} I_{11} & I_{12}^T \; I_{21} \; I_{22} + \Omega \end{bmatrix}.
\]

By Lemma 1 in the Appendix of [5], the Cholesky factorization of \( I(k+1) \) can be obtained by

\[
L_{k+1} = \begin{bmatrix} L_{11} & 0 \; L_{21} \; L_{22} \end{bmatrix}
\]

where \( L_{22}^{k+1} \) is the Cholesky factorization of the low dimensional matrix \( \Omega + L_{22} L_{22}^T = I_{22}^{k+1} - L_{21} L_{21}^T \).

Computing \( L_{k+1} \) by (13) is much more efficient than directly computing the Cholesky factorization of the high dimensional matrix \( I(k+1) \).

**Case (ii).** If the global state vector has been reordered in Section III-F, then the Cholesky factorization of \( I(k) \) cannot be used to construct the Cholesky factorization of \( I(k+1) \).

In this case, a direct Cholesky factorization of \( I(k+1) \) is performed to obtain \( L_{k+1} \).

Since the reordering only happens occasionally, Case (i) can be applied most of the time.

**H. State vector recovery**

Because the global map is maintained by an information vector and an information matrix, the global state estimate \( \hat{X}^G(k+1) \) is not directly available.

By (4), the state vector estimate \( \hat{X}^G(k+1) \) can be recovered by solving the sparse linear equation

\[
I(k+1) \hat{X}^G(k+1) = i(k+1).
\]

The computed Cholesky factorization \( L_{k+1} \) is used to solve the sparse linear equation. Since \( L_{k+1} L_{k+1}^T = I(k+1) \), the sparse linear equation (14) can be solved efficiently by solving \( L_{k+1} Y = i(k+1) \) and \( L_{k+1}^T \hat{X}^G(k+1) = Y \).

**IV. SIMULATION AND EXPERIMENT RESULTS**

In this section, simulation and experiment results are given to illustrate the accuracy and efficiency of SLSJF.

**A. Simulation results**

A simulation experiment with a large number of features was conducted to evaluate the proposed map joining algorithm. The \( 150 \times 150 m^2 \) environment contains 2500 features arranged in uniformly spaced rows and columns. The robot starts from the left bottom corner of the square and follows a random trajectory as shown in Fig. 3(a). A sensor with a field of view of 180 degrees and a range of 6 meters is simulated to generate relative range and bearing measurements between the robot and the features. There are 22384 robot poses in total and 138026 measurements are made from the robot poses. The robot observed 2316 features in total and most of them are observed a few times.

Five hundred small sized local submaps were built by EKF SLAM using the odometry and measurement information. Each local map contains around 10 features. Fig. 3(b) shows the global position of the robot trajectory and features involved in the local map 100. Fig. 3(c) shows the local map 100.

Fig. 3(a) shows the final map generated by EKF SLAM. Fig. 4(a) shows the global map generated by fusing all the 500 local maps using EKF sequential map joining [7][8]. Fig. 4(b) shows the global map generated by fusing all the 500 local maps using SLSJF. The data association in SLSJF is performed using Nearest Neighbor method [10]. It can be seen that the global feature position estimate using the three methods are all consistent since the feature positions fall within the 2σ covariance ellipses drawn around the estimates. Close examination shows that the final map uncertainty from the three methods are very similar.

Fig. 5(a) shows the errors and 2σ bounds of the estimates of robot end poses in each local map after fusing the local map. It is clear that the estimates are consistent. Comparing with the localization result from traditional EKF SLAM shown in Fig. 5(b) and the robot end poses estimate in EKF sequential map joining solution shown in Fig. 5(c), it can be seen that the robot pose estimate from the three different algorithms are almost identical.

Fig. 5(d) shows all the non-zero elements of the sparse information matrix obtained by SLSJF in black. Fig. 5(e) shows the CPU time \(^1\) required for the local map fusion using SLSJF and EKF sequential map joining. The total time for fusing all the 500 local submaps is 129 seconds for SLSJF and 7240 seconds for EKF sequential map joining (building the 500 local submaps takes 409 seconds; it takes traditional EKF SLAM more than 21 hours to finish the map). Table I presents the detailed processing time of the two map joining algorithms. In SLSJF, the major computation cost is “data association” which includes the time for covariance sub-matrix recovery. The “others” including reordering of state vector, Cholesky factorization and state vector recovery also take significant time. On the other hand, “global map update” uses most of the computation time in EKF sequential map joining.

Fig. 5(f) compares the CPU time of SLSJF with the proposed reordering strategy and that of SLSJF with the pure AMD reordering (for the proposed reordering, the parameters \( n_0 = 400 \) and \( d_0 = 15 \), for the pure AMD reordering, the reordering is performed after fusing every 5 local maps, the parameters are chosen such that both algorithms have their best performance). The performance of the two reordering algorithms are very similar, presumably due to the fact that the MATLAB implementation of AMD algorithm is very efficient.

**B. Experimental results**

SLSJF was also applied to the popular Victoria Park data set which was first used in [15]. Neither ground truth nor

\(^1\) All time measurements in this paper are performed on a laptop computer with Intel Core 2 Duo T7500 at 2.2GHz, 3GB of RAM and running Windows; all programs are written in MATLAB.
noise parameters are available for this data set. Published results for the vehicle trajectory and uncertainty estimates vary
[4][5][15][16], presumably due to different parameters used by
various researchers. The results in this section therefore only
demonstrate that SLSJF can be applied to this popular data set.

Fig. 6(a) shows the map obtained by traditional EKF SLAM.
The odometry and range-bearing observation data were used
to build 200 local maps by EKF SLAM. Fig. 6(b) shows the
global map obtained by joining the 200 local submaps

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Data Association</th>
<th>Update</th>
<th>Others</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>EKF map joining</td>
<td>11s</td>
<td>7223s</td>
<td>6s</td>
<td>7240s</td>
</tr>
<tr>
<td>SLSJF</td>
<td>76s</td>
<td>12s</td>
<td>41s</td>
<td>129s</td>
</tr>
</tbody>
</table>

TABLE I
PROCESSING TIME OF EKF SEQUENTIAL MAP JOINING AND SLSJF.
Fig. 4. Simulation results — global maps obtained by EKF sequential map joining and SLSJF.
using SLSJF. Data association in SLSJF is performed using Nearest Neighbor method [10]. Fig. 6(c) shows all the non-zero elements of the information matrix in black. The information matrix is not very sparse because the sensor range is relatively large (around 80m) as compared with the size of the environment (300m × 300m). Fig. 6(d) shows the CPU time required to fuse each of the 200 local maps. The total computation time for joining all the 200 maps by SLSJF is 22 seconds (the time used for building the 200 local maps is 63 seconds).

V. RELATED WORK AND DISCUSSIONS

In this section, some of the properties of SLSJF and some related work are discussed.

A. Different ways to achieve sparse representation

The sparse representations of SLAM recently proposed in the literature (e.g. [2][3][4][5][22]) make use of different state vectors and/or have different strategies for marginalizing out robot poses. In SAM [3], iSAM [16], Tectonic SAM [23]
and full-SLAM [6], all the robot poses are included in the state vector and no marginalization is needed. However, the dimension of the state vector is very high especially when the robot trajectory is long.

When all the previous robot poses are marginalized out as in traditional EIF SLAM, the information matrix becomes dense although it is approximately sparse [12]. The SEIF algorithm presented in [2] approximates some small entries of the information matrix by zeros using sparsification, but this leads to inconsistent estimates [4].

The ESEIF developed by [4] follows the EIF SLAM algorithm, but marginalizes out the robot pose and relocates the robot from time to time. In this way the information matrix is kept exactly sparse with sacrificing the robot location information once in a while.

In D-SLAM algorithm [5], the robot pose is not involved in the state vector. The observations made from one robot pose are first transferred into the relative position information among the observed features (the robot pose is marginalized out from the observations), then the relative position information is used to update the map. This process also results in some information loss.

The D-SLAM map joining algorithm [22] first builds local maps and then marginalizes out the robot start and end poses from the local map, the obtained relative position information among features are fused into the global map in a way similar to the D-SLAM algorithm. Most of the odometry information is maintained in the local maps but there is still some information loss due to the marginalization of robot start/end poses.

In SLSJF, the robot start and end poses from the local map are never marginalized but kept in the global state vector. Thus all the information is kept by adding one robot pose in the global state vector for each local map.

If each local map is treated as one integrated observation, then SLSJF has some similarity to iSAM [16]. The role of local maps in SLSJF is also similar to the “star nodes” in the Graphical SLAM [17]. However, in the Graphical SLAM, the poses are first added in the graph and then “star nodes” are made. While in SLSJF, most of the robot poses are marginalized out in the local map building steps. Those robot poses are never present in the global state vector.

It should be noted that in SLSJF, there are no dynamic objects involved in the global state vector, so the algorithm is a special case of EIF (no prediction step). It can be treated as a linearized least square solution with only one iteration.
in each map fusion step. In fact, at any map fusion step, the linearization error can be reduced further by recomputing the information matrix $I$ and the information vector $i$ as a sum of all the contributions in (9) using the new estimate as linearization point for the Jacobians.

**B. Computational complexity**

The map joining problem considered in this paper is similar to that studied in [7] and [8]. The computational complexity of the sequential map joining approach in [7] and the Constrained Local Submap Filter in [8] is quadratic for the global map update.

In SLSJF, the robot end poses of the local maps are included in the global state vector and the EIF implementation results in an exactly sparse information matrix. The sparseness of information matrix results in significant computational saving in the map joining process.

Although simulation results show that SLSJF is computationally very efficient, there are still a few steps in SLSJF preventing the computational complexity to be linear for worst case scenarios. For example, the number of fill-ins introduced in the Cholesky factorization depends on the environment and the robot trajectory and this influences the computation cost of the full Cholesky factorization and the solving of the sparse linear equations. Also, the computation cost of the proposed reordering is not well understood yet. In theory, SLSJF suffers the general $O(n^{1.5})$ worst case scenario of planar grids, as all sparse factorization based methods do [19]. This is similar to the treemap algorithm [13] and the SAM using nested dissection algorithm [14].

Very recently, it is shown in [18] that the average computational cost of map joining in each step can be reduced to $O(n)$ by “Divide and Conquer SLAM”. However, the algorithm is only compared to EKF SLAM instead of the more efficient EKF sequential map joining. Moreover, linear computation cost for data association is impossible unless some approximation is applied [18].

The SLSJF has some similarity to the Tectonic SAM algorithm [23]. Tectonic SAM is also an efficient submap based approach and the state vector reordering and Cholesky factorization are used in solving the least-square problem. The submap fusion in Tectonic SAM uses a divide-and-conquer approach, which we believe to be more efficient than the sequential map joining in SLSJF when data association is assumed. The major difference between Tectonic SAM and SLSJF is that in Tectonic SAM, all the poses involved in building the local maps are kept and the dimension of the global state vector is much higher than that of SLSJF.

In SLSJF, it is also assumed that the local maps are consistent and accurate enough. If this is not the case, SLSJF may produce wrong results due to the inconsistency or linearization errors. For example, when the orientation error in the robot pose is very large, the proposed data association algorithm will not work and a more complicated and robust data association algorithm is required (such as making multiple hypotheses on the robot heading and checking each of the hypotheses). This will also increase the number of potentially matched features and increase the computation time of covariance sub-matrix recovery.

Similar to [7][8], there is no requirement on the structure of the environments for SLSJF to be applicable. This is different from the treemap SLAM algorithm [13][20][21] where the environment has to be “topological suitable” and a hierarchical tree partitioning (HTP) subalgorithm is needed to find such a suitable partitioning.

When the environment has an efficient tree representation and the HTP algorithm can find such a representation, the treemap SLAM algorithm can be very fast [20]. However, as pointed out in [13][20], “large open halls or outdoor environments often do not have an efficient tree representation”. Thus SLSJF may be more suitable for these applications.

Another difference between SLSJF and the treemap SLAM algorithm is that the covariance submatrix recovery and data association was ignored in the treemap SLAM implementations [13] [20][21].

**C. Requirements on SLSJF**

One assumption made in SLSJF is that the local map only involves “nearby objects”. This guarantees that the global information matrix is exactly sparse no matter how many local maps are fused. When this assumption does not hold such as the case with vision sensors, SLSJF can still be applied but may not be that efficient.

**D. Exact covariance submatrix recovery**

The covariance submatrix recovery in SLSJF is exact. This is different from the approximate covariance submatrix recovery methods (e.g. [2] [24]) where only an approximate or upper bound of covariance submatrix is computed. As pointed out in [24], the upper bound can only be used in nearest neighbor data association [10] but cannot be used in the more robust joint compatibility test [11].

An algorithm for exact recovery of covariance submatrix was proposed in iSAM [16]. It has “$O(n)$ time complexity for band-diagonal matrices and matrices with only a constant number of entries far from the diagonal, but can be more expensive for general sparse matrices” [16]. The covariance sub-matrix recovery in SLSJF is similar. The major advantages of SLSJF over iSAM is that the dimension of the state vector in SLSJF is much lower than that of iSAM. Thus SLSJF may be more suitable for the situations where the robot trajectory is very long and/or the observation frequency is high, which is true for many sensors such as laser.

**E. Incremental Cholesky factorization for recovery**

The idea of using incremental Cholesky factorization to compute the Cholesky factorization is motivated by [5]. The main difference between the recovery method in this paper and that in [5] is that complete Cholesky factorization and direct method for linear equation solving are used in SLSJF, while approximate Cholesky factorization and Preconditioned Conjugate Gradient method are used in [5].
The incremental Cholesky factorization also has some similarity with the QR factorization update in [16]. The QR factorization update in [16] is based on “Givens rotations”, while the incremental Cholesky factorization process in SLSJF is based on the “block-partitioned form of Cholesky factorization”. The performance of these two approaches are expected to be similar.

F. Reordering of global state vector

The reordering of global state vector is motivated by [5] and [3][14][16] while the reordering of state vector in [5] is based on distances only and the reordering in [3][14][16] is based on AMD or Nested Dissection (ND) only. In SLSJF, the reordering of state vector aims to combine the advantages of AMD reordering (the number of fill-ins is reduced) and the advantages of reordering by distance (the efficient incremental Cholesky factorization procedure can be applied in most cases).

The idea behind the “reordering by distance” is to make sure that the robot will not observe any features that are not in the bottom part within short time period no matter in which direction the robot is moving (since robot cannot move very far away). However, this is not the best way of reordering for indoor environments where features in different rooms might actually be very close but cannot be seen simultaneously. For indoor environments, the knowledge on the structure of the environment (and the knowledge on the possible robot trajectory) can be exploited to place “the possibly to be re-observed features” near the bottom of the state vector.

G. Consistency

The SLSJF algorithm does not contain any approximations (such as sparsification [2]) that can lead to estimator inconsistency. However, as the case with all EKF/EIF based estimation algorithms, it is possible that inconsistencies occur in SLSJF due to errors introduced by the linearization process.

It was shown in [9] that using submaps can improve consistency in SLAM by reducing linearization errors [9]. However, in our experience, we did not observe significant consistency improvement of submap based approaches over EKF SLAM. What we have found is that EKF SLAM using batch update is much more consistent than EKF using sequential update. For submap based approaches, small inconsistency in local maps may result in large inconsistency in the global map. The consistency issue is a bit more complicated and needs further investigation.

H. Treating the local submap as a virtual observation

Many submap based SLAM algorithms have (either explicitly or implicitly) treated the local submap as a virtual observation, but most of them treat a local submap as “an observation made from the robot to all the features in the local submap”. In SLSJF, the local submap is treated as “an observation made from the robot to all the features in the local submap and a virtual robot located at the robot end pose”. This motivates the inclusion of all the robot end poses in the global state vector to achieve the exactly sparse information matrix.

I. Comparison with two level mapping algorithms

The output of SLSJF is one single global stochastic map. This approach is different from the two level mapping algorithms (e.g. Hierarchical SLAM [26], Atlas [27], Network Coupled Feature Maps [28]), where a set of submaps are maintained and the relationship among these maps is described at a higher level. Though promising due to their reduced computational cost, the two level mapping approaches require more work to completely resolve the question of how to treat the overlapping regions among local submaps. As pointed out in [26], all the two level mapping systems result in suboptimal solutions because the effect of the upper level update cannot be propagated back to the local level.

When the local maps are accurate enough and relatively separated, the two level mapping algorithms appear to be more powerful. But when the uncertainty of local submaps is not very small and the overlaps among local submaps are significant, sequential map joining may be more appropriate. This issue needs to be further explored.

VI. Conclusions

By adding robot end poses of the local maps into the global state vector and applying Extended Information Filter (EIF) as the estimator, a sparse extended information filter for local submap joining, SLSJF, is developed. There is no approximation involved in SLSJF apart from linearization processes. SLSJF contains two important steps for real application of EIF based algorithms — a covariance submatrix recovery step and a data association step. The sparse information matrix together with the novel state vector and covariance submatrix recovery procedure make the SLSJF algorithm computationally very efficient.

SLSJF achieves an exactly sparse information matrix with no information loss. The dimension of its state vector is significantly less than that of the full SLAM algorithm [6] where all the robot poses are included in the state vector. As it does not matter how the local submaps are built, SLSJF can also be applied to large-scale range-only or bearing-only SLAM problem — use a range-only or bearing-only SLAM algorithm to build local submaps, then fuse them together using SLSJF.

For the successful application of SLSJF for local submap joining, it is important that all the local submaps are consistent. This is because inconsistency of local submaps may result in wrong data association between local submaps and the global map. Thus it is essential to use a reliable SLAM algorithm to build the local submaps.

More work is required to determine the best reordering strategy which can be used in SLSJF; the optimal size of the local map and the optimal number of local maps for a given mapping problem, taking into account the computational costs of both local map joining and the local map building. The idea of SLSJF can be easily extended to the joining of 3D local maps but there are still some technical details that need to be resolved. Research along these directions is underway.
ACKNOWLEDGMENT

The authors would like to thank Dr. Udo Frese for helpful suggestions and the anonymous reviewers for the valuable comments. This work is supported by the ARC Centre of Excellence programme, funded by the Australian Research Council (ARC) and the New South Wales State Government.

REFERENCES