SLAM in $\mathcal{O}(\log n)$ with the Combined Kalman -Information Filter

César Cadena and José Neira

***Abstract*— In this paper we show that SLAM can be executed in as low as $\mathcal{O}(\log n)$ per step. Our algorithm, the Combined Filter SLAM, uses a combination of Extended Kalman and Extended Information filters in such a way that the *total* cost of building a map can be reduced to $\mathcal{O}(n \log n)$, as compared with $\mathcal{O}(n^3)$ for standard EKF SLAM, and $\mathcal{O}(n^2)$ for Divide and Conquer (D&C) SLAM and the Sparse Local Submap Joining Filter (SLSJF). We discuss the computational improvements that have been proposed for Kalman and Information filters, discuss the advantages and limitations of each, and how a judicious combination results in the possibility of reducing the computational cost per step to $\mathcal{O}(\log n)$. We use simulations and real datasets to show the advantages of the proposed algorithm.**

I. Introduction

In recent years, researches have devoted much effort to develop algorithms to improve the computational efficiency of SLAM. The goal is being able to map large scale environments in real time [1]. One important contribution has been the idea of splitting the full map into local maps and then put the pieces back together in some way. Decoupled Stochastic Mapping [2], Constant Time SLAM [3] and the ATLAS system [4] are local mapping solutions close to constant time, although through approximations that reduce precision. Map Joining [5] and the Constrained Local Submap Filter [6] are exact solutions (except for linearizations). Exact solutions also include Treemap [7], Divide and Conquer (D&C) SLAM [8], Tectonic SAM [9] and Sparse Local Submap Joining (SLSJF) SLAM [10]. Given a map of n features, the classical EKF SLAM algorithm is known to have a cost of $\mathcal{O}(n^2)$ per step. Two recent algorithms have provided important reductions in computational cost: D&C SLAM has an amortized cost $\mathcal{O}(n)$ per step, SLSJF SLAM reports a cost $\mathcal{O}(n^{1.5})$ per step in the worst cases. The Treemap has a cost $\mathcal{O}(\log n)$, although with topological restrictions on the environment, and has a rather complex implementation.

In this paper we describe the Combined Filter SLAM (CF SLAM) algorithm, that can reduce the computational cost per step down to $\mathcal{O}(\log n)$; the total computational cost can be reduced down to $\mathcal{O}(n \log n)$. The CF SLAM algorithm is a judicious combination of Extended Kalman and Extended Information filters, combined with a divide and conquer local mapping strategy. Being a local mapping algorithm, it can provide more consistent results, compared with Treemap, reported to have the same $\mathcal{O}(\log n)$ cost, but computing an absolute map [11]. CF SLAM is also conceptually simple and rather easy to implement.

This paper is organized as follows: the next section contains a detailed description of the improvements that have been reported on the use of Kalman and Information filters for SLAM, leading to the algorithm that we propose. Section III contains a description of our algorithm and a study of its computational cost and consistency properties. We study the main factors that may influence the computational cost in section IV. In section V we use the DLR circles dataset to test the algorithm with real data, and discuss results compared with the Treemap algorithm. In the final section we summarize the results and sketch the challenges ahead.

César Cadena and José Neira are with the Departamento de Informática e Ingenieria de Sistemas, Centro Politécnico Superior, Universidad de Zaragoza, Zaragoza, Spain {ccadena, jneira}@unizar.es

This research has been funded in part by the Dirección General de Investigación of Spain under project DPI2006-13578 and by the European Union under project RAWSEEDS FP6-IST-045144.

This work has been submitted to 2009 IEEE/RSJ International Conference on Intelligent RObots and Systems IROS-09.

II. Extended Kalman filters, Extended Information filters and Map Joining filters

A. The Extended Kalman Filter

The Extended Kalman Filter (EKF) is one of the main paradigms in SLAM [12], [13]. In EKF SLAM, a map $(\mu, \mathbf{\Sigma})$ includes the state μ to be estimated, which contains the current vehicle location and the location of a set of environment features. The covariance of μ, represented by $\mathbf{\Sigma}$, gives an idea of the precision in the estimation, 0 meaning total precision. EKF SLAM is an iterative prediction-sense-update process whose formulation we believe is widely known and is thus summarized in Table I (left). In exploratory trajectories, and using a sensor of limited range, the size of the map grows linearly. Given that each step is $\mathcal{O}(n^2)$, the *total* cost of carrying out EKF SLAM is known to be $\mathcal{O}(n^3)$.

B. The Extended Information Filter

In Extended Information Filter (EIF) SLAM, a map $(\xi, \mathbf{\Omega})$ consists of the information state ξ to be estimated, and the information matrix $\mathbf{\Omega}$, which gives an idea of the information known about the estimation (0 meaning no information). EIF SLAM is also an iterative prediction-sense-update process (its formulation is also summarized in Table I, right). The Information filter is an algebraic equivalent to the Kalman filter, because the following equivalences hold [12]:

$$\mathbf{\Omega} = \mathbf{\Sigma}^{-1} \qquad and \qquad \xi = \mathbf{\Sigma}^{-1}\mu \tag{1}$$

For this reason, KF and IF are considered dual filters [14]. Unfortunately, in the nonlinear case the filters are not completely dual, since both the transition function g and the measurement function h require the state as input

	EKF-SLAM		EIF-SLAM	
Jacobians	$F_t = \frac{\partial g(u_t,\mu_{t-1})}{\partial \mu_{t-1}}\Big\vert_{\hat{\mu}_{t-1}}$ $\mathcal{O}(c)$ $G_t = \frac{\partial g(u_t,\mu_{t-1})}{\partial u_t}\Big\vert_{\hat{u}_t}$		$\mathcal{O}(c)$ $H_t = \frac{\partial h(\mu_{t\vert t-1})}{\partial \mu_{t\vert t-1}}\Big\vert_{\hat{\mu}_{t\vert t-1}}$	$\mathcal{O}(r)$
Prediction	$\mu_{t\vert t-1} = g(u_t, \mu_{t-1})$	$\mathcal{O}(c)$	$\mu_{t-1} = \mathbf{\Omega}_{t-1} \backslash \xi_{t-1}$	$\mathcal{O}(n)$
	$\mathbf{\Sigma}_{t\vert t-1} = F_t \mathbf{\Sigma}_{t-1} F_t^T + G_t R_{t-1} G_t^T$	$\mathcal{O}(n)$	$\xi_{t\vert t-1} = \mathbf{\Omega}_t g(u_t, \mu_{t-1})$	$\mathcal{O}(n)$
			$\Phi = F_t^{-T} \mathbf{\Omega}_{t-1} F_t^{-1}$	$\mathcal{O}(c)$
			$\mathbf{\Omega}_{t\vert t-1} = \Phi - \Phi G_t (R_{t-1} + G_t^T \Phi G_t)^{-1} G_t^T \Phi$	$\mathcal{O}(n)$
Innovation	$\nu_t = z_t - h(\mu_{t\vert t-1})$	$\mathcal{O}(r)$	$\nu_t = z_t - h(\mu_{t\vert t-1})$	$\mathcal{O}(r)$
	$S_t = H_t \mathbf{\Sigma}_{t\vert t-1} H_t^T + Q_t$	$\mathcal{O}(n)$	$S_t = H_t \mathbf{\Omega}_{t\vert t-1}^{-1} H_t^T + Q_t$	$\mathcal{O}(n^3)$
Test χ^2	$D^2 = \nu_t^T S_t^{-1} \nu_t$	$\mathcal{O}(r^3)$	$D^2 = \nu_t^T S_t^{-1} \nu_t$	$\mathcal{O}(r^3)$
	$K_t = \mathbf{\Sigma}_{t\vert t-1} H_t^T / S_t$	$\mathcal{O}(n)$		
Update	$\mathbf{\Sigma}_t = (I - K_t H_t) \mathbf{\Sigma}_{t\vert t-1}$	$\mathcal{O}(n^2)$	$\mathbf{\Omega}_t = \mathbf{\Omega}_{t\vert t-1} + H_t^T Q_t^{-1} H_t$	$\mathcal{O}(n)$
	$\mu_t = \mu_{t\vert t-1} + K_t \nu_t$	$\mathcal{O}(n)$	$\xi_t = \xi_{t\vert t-1} + H_t^T Q_t^{-1} (\nu_t + H_t \mu_{t\vert t-1})$	$\mathcal{O}(n)$
Cost per step	$\mathcal{O}(n^2)$		without Data Asociation (S_t not required) $\mathcal{O}(n)$	
Total cost during exploration	$\mathcal{O}(n^3)$		$\mathcal{O}(n^2)$	

TABLE I

FORMULATIONS AND OPTIMIZED (BEST REPORTED TO DATE) COMPUTATIONAL COST OF EACH OF THE OPERATIONS CARRIED OUT USING THE EXTENDED KALMAN FILTER (LEFT) AND THE EXTENDED INFORMATION FILTER (RIGHT). VARIABLE n IS THE SIZE OF THE FINAL STATE μ_t OR INFORMATION VECTOR ξ_t, r IS THE SIZE OF THE MEASUREMENT VECTOR z_t AND c IS A CONSTANT.

[12]. For this reason, the initial required computation during the prediction step is to derive the state variables μ_{t-1}. In the general case, the EIF is considered computationally more expensive than the EKF: computing the state μ_{t-1} without optimizations is $\mathcal{O}(n^3)$ because of the inversion of the information matrix $\mathbf{\Omega}_{t-1}$ (only optimized costs are given in the table).

An important insight into reducing the computational cost of EIF SLAM was to observe that the information matrix $\mathbf{\Omega}$ is *approximately* sparse [12], and can be easily sparsified. This Sparse Extended Information Filter (SEIF) allows reducing the computational cost of most EIF operations to $\mathcal{O}(n)$. The computation of the innovation covariance S_t is the exception, it still is $\mathcal{O}(n^3)$. Its computation is not strictly necessary, except to carry out data association. Therefore, in SLAM problems where data association is known, or can be decided by other means (the use of textured points in vision could be an example), SEIF SLAM is $\mathcal{O}(n)$ per step, and the total cost $\mathcal{O}(n^2)$.

Another important observation regarding EIF SLAM is that if all vehicle locations are incorporated into the information vector, instead of the current one only, the information matrix becomes *exactly* sparse [15]. In this Exactly Sparse Delayed-State Filter (ESDF) SLAM, the reduction in the computational cost due to this fact is the same as in SEIF. Additionally, since no approximations due to sparsification take place, the results are more precise [16]. When the state or information vector contain only the current vehicle location, we have an *online* SLAM problem; if it contains all vehicle locations along the trajectory, we have *full* SLAM.

Again, during exploration, the total cost of ESDF is known to be $\mathcal{O}(n^2)$, as compared with the cubic cost of EKF SLAM. Note however that the information vector is ever increasing, even during revisiting every new vehicle location is incorporated.

C. Map Joining SLAM with EKF

Local mapping (or submapping) algorithms were the next contribution to the reduction of the computational cost of SLAM. In these algorithms, sequences of local map of constant size are sequentially built (in constant time because of the size limitation) and then put together into a global map in different ways.

One of such solutions, Map Joining SLAM [5], works in the following way: given two consecutive local maps $(\mu_1, \mathbf{\Sigma}_1)$ and $(\mu_2, \mathbf{\Sigma}_2)$, the map $(\mu, \mathbf{\Sigma})$ resulting from joining their information together is computed in three initialization-innovation-update steps, summarized in Table II, left. A specialized version of the Extendad Kalman filter is used, where the full state vectors and covariance matrices are simply stacked in the predicted map; correspondences can then be established between features from both maps through a prediction function h, equivalent to considering a perfect measurement $z = 0, Q = 0$. Notice that this is possible, since the EKF allows the consideration of 0 covariance measurements. The update step includes a computation using the transformation function g to transform all map features and vehicle locations to a common base reference, usually the starting vehicle location in the first map.

Map joining SLAM is constant time most of the time, when working with local maps. However, joining operations are $\mathcal{O}(n^2)$ on the final size of the map, and although it results in great computational savings (it may slash the cost by a large constant), MJ SLAM is still $\mathcal{O}(n^2)$ per step, just as EKF SLAM is.

D. Map Joining SLAM with EIF

The Extended Information filter can also be used to carry out the map joining operations, as reported in the Sparse Local Submap Joining filter (SLSJF) SLAM [10]. Its application is not as straightforward as the Map Joining with EKF for two reasons. First, in Map joining with EKF

	Join with EKF		**Join with EIF**	
Jacobians	$G = \frac{\partial g(\mu^+)}{\partial \mu^+}\Big\vert_{\hat{\mu}^+}$	$\mathcal{O}(n_2)$	$H = \frac{\partial h(\mu^-)}{\partial \mu^-}\Big\vert_{\hat{\mu}^-}$	$\mathcal{O}(n_2)$
Initialization	$\mu^- = \begin{bmatrix} \mu_1 \\ \mu_2 \end{bmatrix}$	$\mathcal{O}(n)$	$\mu^- = g(\mu_1, \mu_2)$	$\mathcal{O}(n_2)$
	$\boldsymbol{\Sigma}^- = \begin{bmatrix} \boldsymbol{\Sigma}_1 & 0 \\ 0 & \boldsymbol{\Sigma}_2 \end{bmatrix}$	$\mathcal{O}(n^2)$	$\xi^- = \begin{bmatrix} \xi_1 \\ 0 \end{bmatrix}$	$\mathcal{O}(n)$
			$\boldsymbol{\Omega}^- = \begin{bmatrix} \boldsymbol{\Omega}_1 & 0 \\ 0 & 0 \end{bmatrix}$	$\mathcal{O}(n)$
Innovation	$\nu = -h(\mu^-)$	$O(r)$	$\nu = \mu_2 - h(\mu^-)$	$O(r)$
	$S = H\boldsymbol{\Sigma}^- H^T$	$\mathcal{O}(n)$	$Q^{-1} = \boldsymbol{\Omega}_2$	given
Update	$K = \boldsymbol{\Sigma}^- H^T / S$	$\mathcal{O}(n)$		
	$\boldsymbol{\Sigma}^+ = (I - KH)\boldsymbol{\Sigma}^-$	$\mathcal{O}(n^2)$	$\boldsymbol{\Omega} = \boldsymbol{\Omega}^- + H^T \boldsymbol{\Omega}_2 H$	$\mathcal{O}(n_2)$
	$\mu^+ = x^- + K\nu$	$\mathcal{O}(n)$	$\xi = \xi^- + H^T \boldsymbol{\Omega}_2 (\nu + H\mu^-)$	$\mathcal{O}(n_2)$
	$\mu = g(\mu^+)$	$\mathcal{O}(n_2)$	$\mu = \boldsymbol{\Omega} \backslash \xi$	$\mathcal{O}(n)$ to $\mathcal{O}(n^2)$
	$\boldsymbol{\Sigma} = G\boldsymbol{\Sigma}^+ G^T$	$\mathcal{O}(n_2^2)$		
Cost per join	$\mathcal{O}(n^2)$		$\mathcal{O}(n)$ to $\mathcal{O}(n^2)$	

TABLE II

FORMULATIONS AND OPTIMIZED (BEST REPORTED TO DATE) COMPUTATIONAL COST OF EACH OF THE OPERATIONS CARRIED OUT FOR MAP JOINING USING THE EXTENDED KALMAN FILTER (LEFT) AND USING THE EXTENDED INFORMATION FILTER (RIGHT). VARIABLES n_1, n_2 AND n ARE THE SIZE OF THE FIRST, SECOND AND FINAL STATE OR INFORMATION VECTOR RESPECTIVELY, AND r IS THE SIZE OF THE MEASUREMENT VECTOR z_t.

correspondences are established by considering a perfect measurement $z = 0, Q = 0$. In the information form, 0 covariance measurements are not allowed since Q^{-1} is required in the formulation. For this reason, in SLSJF SLAM, having two consecutive local maps $(\mu_1, \boldsymbol{\Sigma}_1)$ and $(\mu_2, \boldsymbol{\Sigma}_2)$ to join, the resulting map $(\xi, \boldsymbol{\Omega})$ is predicted in the information form with the information of the first map, and an initial 0 (no information) from the second map. The innovation is computed considering the second map as a set of measurements for the full map ($z_t = \mu_2$, $Q_t = \boldsymbol{\Omega}_2^{-1}$), and the final update step computes the information state ξ and information matrix $\boldsymbol{\Omega}$ using the standard EIF equations.

An important observation made in [10] is that the information matrix resulting from the map joining operation using EIF is *exactly* sparse if the vehicle locations coming from each local map are maintained in the final information state. This is a situation very similar to the full SLAM problem, except that not all vehicle locations remain, only a fraction corresponding to the final vehicle locations in each local map. There is an additional final computation of the final state μ, to make it available for potential future joining operations. This state recovery can be done with a preordering of minimum degree of the information matrix and the sparse Cholesky factorization to solve the linear system. In the section IV-A we shall see that the cost of this computation can be proportional to n during exploration, and up to $\mathcal{O}(n^2)$ in the worst case.

E. Divide and Conquer with EKF

In the Divide and Conquer (D&C) SLAM algorithm [8] it was pointed out that SLAM can be carried out in *linear* time per step if map joining operations are carried out in a hierarchical binary tree fashion, instead of a sequential fashion. The leafs of the binary tree represent the sequence of l local maps of constant size p, computed with standard EKF SLAM. These maps are joined pairwise to compute $l/2$ local maps of double their size ($2p$), which will in turn be joined pairwise into $l/4$ local maps of size $4p$, until finally two local maps of size $n/2$ will be joined into one full map of size n, the final map. The $\mathcal{O}(n^2)$ updates are not carried out sequentially, but become more distant as the map grows. An $\mathcal{O}(n^2)$ computation can then be amortized in the next n steps, making the amortized version of the algorithm linear with the size of the map. It can also be shown that the total cost of D&C SLAM is $\mathcal{O}(n^2)$, as compared to the total cost of EKF SLAM, $\mathcal{O}(n^3)$.

III. OUR PROPOSAL: COMBINED FILTER SLAM

The algorithm proposed here, Combined Filter SLAM, has three main highlights:

a) Local mapping is carried out using standard online EKF SLAM to build a sequence of maps of constant size p: each local map $(\mu_i, \boldsymbol{\Sigma}_i)$ is also stored in information form $(\xi_i, \boldsymbol{\Omega}_i)$; both the information vector and the information matrix are computed in $\mathcal{O}(p^3)$, constant with respect to the total map size n. Each local map only keeps the final pose of the robot. EKF SLAM in local maps allows robust data association, e.g. with JCBB, and local maps remain consistent.

b) Map Joining is carried out using EIF, keeping vehicle locations from each local map in the final map: this allows to exploit the exact sparse structure of the information matrix and the join can be carried out in linear time with the final size of the map.

c) In contrast with sequential map joining strategy followed by SLSJF, the D&C strategy is followed to decide when map joining takes place: we will see that this will result in a total computation cost of $\mathcal{O}(n \log n)$, as compared with the total cost of SLSJF, $\mathcal{O}(n^2)$. Additionally, the computation per step can be amortized to $\mathcal{O}(\log n)$, as compared with $\mathcal{O}(n)$ for SLSJF.

A. Total computational complexity

In exploratory trajectories, the process of building a map of size n using the proposed CF SLAM follows the divide and

conquer strategy: $l = n/p$ maps of size p are produced (not considering overlap), at cost $\mathcal{O}(p^3)$ each, which are joined into $l/2$ maps of size $2p$, at cost $\mathcal{O}(2p)$ each. These in turn are joined into $l/4$ maps of size $4p$, at cost $\mathcal{O}(4p)$ each. This process continues until two local maps of size $n/2$ are joined into one local map of size n, at a cost of $\mathcal{O}(n)$, see section IV-A. SLSJF SLAM and our algorithm carry out the same number of map joining operations. The fundamental difference is that in our case the size of the maps involved in map joining increases at a slower rate than in SLSJF SLAM.

The total computational complexity of CF SLAM in this case is:

$$\begin{aligned} C &= O\left(p^3 l + \sum_{i=1}^{\log_2 l} \frac{l}{2^i}(2^i\, p)\right) \\ &= O\left(p^3 n/p + \sum_{i=1}^{\log_2 n/p} \frac{n/p}{2^i}(2^i\, p)\right) \\ &= O\left(p^2 n + \sum_{i=1}^{\log_2 n/p} n\right) \\ &= O\left(n + n\log n/p\right) \\ &= O\left(n + n\log n\right) \\ &= O\left(n\log n\right) \end{aligned}$$

Therefore, CF SLAM offers a reduction in the total computational cost to $O\left(n\log n\right)$, as compared with the total cost $O\left(n^3\right)$ for EKF SLAM, and $O\left(n^2\right)$ for D&C SLAM and SLSJF SLAM. Furthermore, as in D&C SLAM, the map to be generated at step t will not be required for joining until step $2\,t$. This allows us to amortize the cost $\mathcal{O}(t)$ at this step by dividing it up between steps $t+1$ to $2\,t$ in equal $\mathcal{O}(1)$ computations for each step. In this way, our amortized algorithm becomes $\mathcal{O}(\log n)$ per step.

B. CF SLAM vs. other algorithms

To illustrate the computational efficiency of the algorithm proposed in this paper, a simulated experiment was conducted using a simple Matlab implementation of CF SLAM, D&C SLAM [8], and SLSJF [10] (with reordering using *symmmd* instead of the heuristic criteria proposed there). The simulated environment contains equally spaced point features 1.33m apart, a robot equipped with a range and bearing sensor with a field of view of 180 degrees and a range of 2m. Local maps are built containing $p = 30$ features each. All tests are done over 100 MonteCarlo runs. Henceforth these parameters are maintained in all simulations. Fig. 1 show the resulting execution costs of the three algorithms. We can see that the total costs of D&C SLAM and the SLSJF tend to be equal (fig. 1, middle). This is expected, since both have a total cost of $\mathcal{O}(n^2)$. We can also see that the total cost of our algorithm increases more slowly, it is expected to be $\mathcal{O}(n\log n)$. We can finally see that the amortized cost of our algorithm exhibits an $\mathcal{O}(\log n)$ behavior, always smaller than the cost of the other two algorithms (fig. 1, bottom).

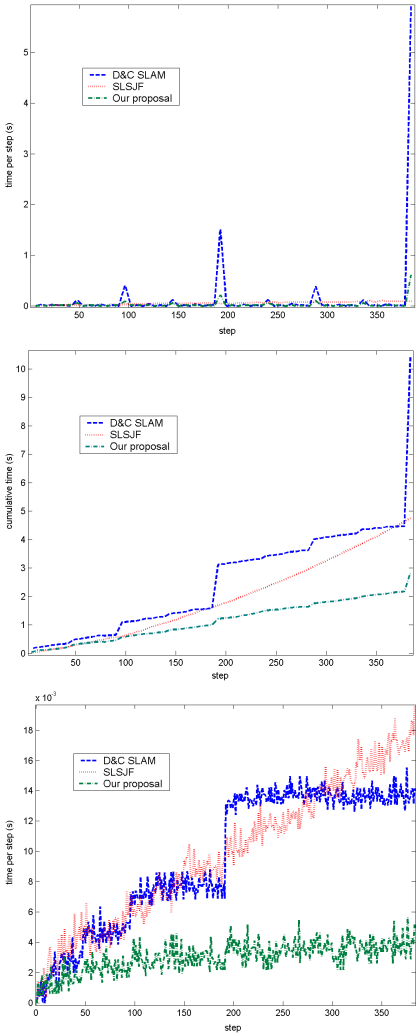

Fig. 1. Computational time in the simulated execution for D&C SLAM (blue), SLSJF SLAM (red) and our algorithm, CF SLAM (green): (top) Time per step; (middle) Total time of execution; (bottom) Time per step for SLSJF vs. amortized time per step for D&C SLAM and our algorithm, CF SLAM. The final map contains 1093 features from 64 local maps.

C. Consistency

When the ground truth is available as in this simulated experiment, we can carry out a statistical test on the estimation $(\mu, \mathbf{\Sigma})$ for filter consistency. We define the consistency index, CI, as:

$$CI = \frac{D^2}{\chi^2_{n,1-\alpha}} \tag{2}$$

where D^2 is the Mahalanobis distance, $n = dim(\mu)$ and $(1-\alpha)$ is the confidence level (95% typically). When $CI < 1$ the estimation is consistent with the ground truth, otherwise the estimation is optimistic, inconsistent.

It is known that local map-based strategies, (e.g. SLSJF and D&C SLAM) improve the consistency of SLAM by including less linearization errors than strategies based in one global map, (e.g. Treemap) [11]. We tested consistency

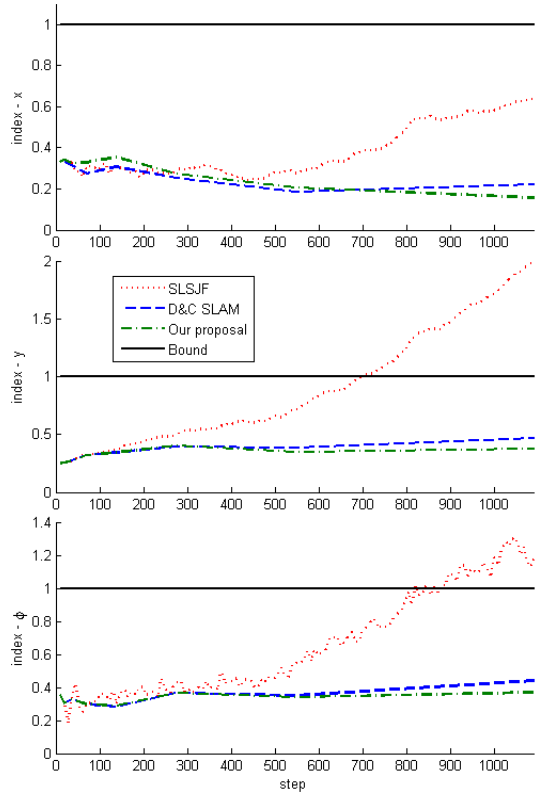

Fig. 2. Mean consistency index CI in x, y, and ϕ for SLSJF, D&C SLAM and our proposal.

of SLSJF, D&C SLAM and CF SLAM proposal on the simulated experiments. Fig. 2 shows the evolution of the mean consistency index of the vehicle position in x (top) and y (middle), and orientation ϕ (bottom), during the steps of the trajectory. We can see that the performance of th index for D&C SLAM and for our proposal a very similar and clearly better than for SLSJF. This is due to that both D&C SLAM and our proposal follow a tree structure to carry out the map joining process. However, in SLSJF map joining is sequential, thus errors increase faster in the global map.

IV. Factors that have influence in the computational cost

A. Vehicle trajectory

Given that local mapping is a constant time operation, we concentrate on the computational cost of map joining in information form. The state recovery is the most computationally expensive operation in this case. State recovery is carried out using the Cholesky factorization, with a previous minimum degree ordering of the information matrix. This cost depends on the sparsity pattern of the information matrix and the density of non-zero elements [17], [10]. This in turn depends on the environment, the sensor used and the more importantly, on the trajectory of the robot.

We have used the simulated experiments to study the effect of the trajectory of the vehicle in the computational cost of the map joining operation. Fig. 3 (top) shows an exploratory trajectory (left), the sparsificacion pattern of the information matrix of the final map (middle), and the mean cost for the 100 Monte Carlo runs of state recovery and joining between maps versus the dimension of the state vector after joining (right).

To determine the order of the computational cost, we have made a fit to the equation $y = ax^b$ for the state recovery and for map joining costs. The independent variable x is the dimension of the state vector resulting from each map joining. The dependent variable y is the cost of the operation: for the state recovery (`sr`), the cost is the operation $\mu = \mathbf{\Omega}\backslash\xi$, and for the map joining with EIF (`jEIF`), it is the cost of all operations involved, including the state recovery. The results of the fits can be seen in Table III, top. The sum of squared errors (SSE), the coefficient of correlation squared (R^2) and root mean squared errors (RMSE) suggest that the exponent b can be considered equal to 1 in both state recovery and map joining operations. Thus, as we said in the previous section, in the case of exploration, the cost of map joining has a linear behavior with respect to the dimension of the state vector, and can be amortized in CF SLAM to attain $\mathcal{O}(\log n)$.

We have also studied *lawn mowing*, Fig. 3 (upper middle). In this case, the cost can increase to $\mathcal{O}(n^{1.3})$ (see the exponent from the fit, Table III). In another type of trajectory, *outward spiral*, Fig. 3 (lower middle), the cost can increase to $\mathcal{O}(n^2)$. In the case of repeated traversal of a loop (Fig. 3, bottom), the cost of map joining is linear most of the time, except during loop closing. At this moment, the operation is quadratic with the dimension of the state vector.

Two things are important to note:

- Whatever the cost of the map joining operation, it can be amortized in CF SLAM. This means that in the worst case, when map joining is $\mathcal{O}(n^2)$, CF SLAM is $\mathcal{O}(n)$ per step.
- In these cases, the computational cost of D&C SLAM and SLSJF also increase in the same manner: in the worst case, both will be $\mathcal{O}(n^2)$ per step, so CF SLAM will *always* be better.

B. Local map size

The selection of the local map size p can also influence the computational cost of CF SLAM. Local maps of a large size p (for example $p = 350$) cannot be computed in real time, and also increase the density on non-zero elements in the information matrix (see Fig. 4, top left). If on the contrary the local map size is too small ($p = 4$), a large number of robot poses will appear in the state vector (Fig. 4, top right). Both situations may result in the cost of map joining not being linear anymore (Fig. 4, bottom).

In the first case, the density of non-zero elements is 1 in every 12, and thus map joining in the lower levels of the tree (the most frequent) are very expensive, more than 4s in the simulation. In the case of small local maps, the exponent from the fit increases to 1.14. (see Table III). The density is much lower, 1 in every 1070, but the state vector is much larger because we have many more poses. In the Fig. 4 the number of features is different because the memory

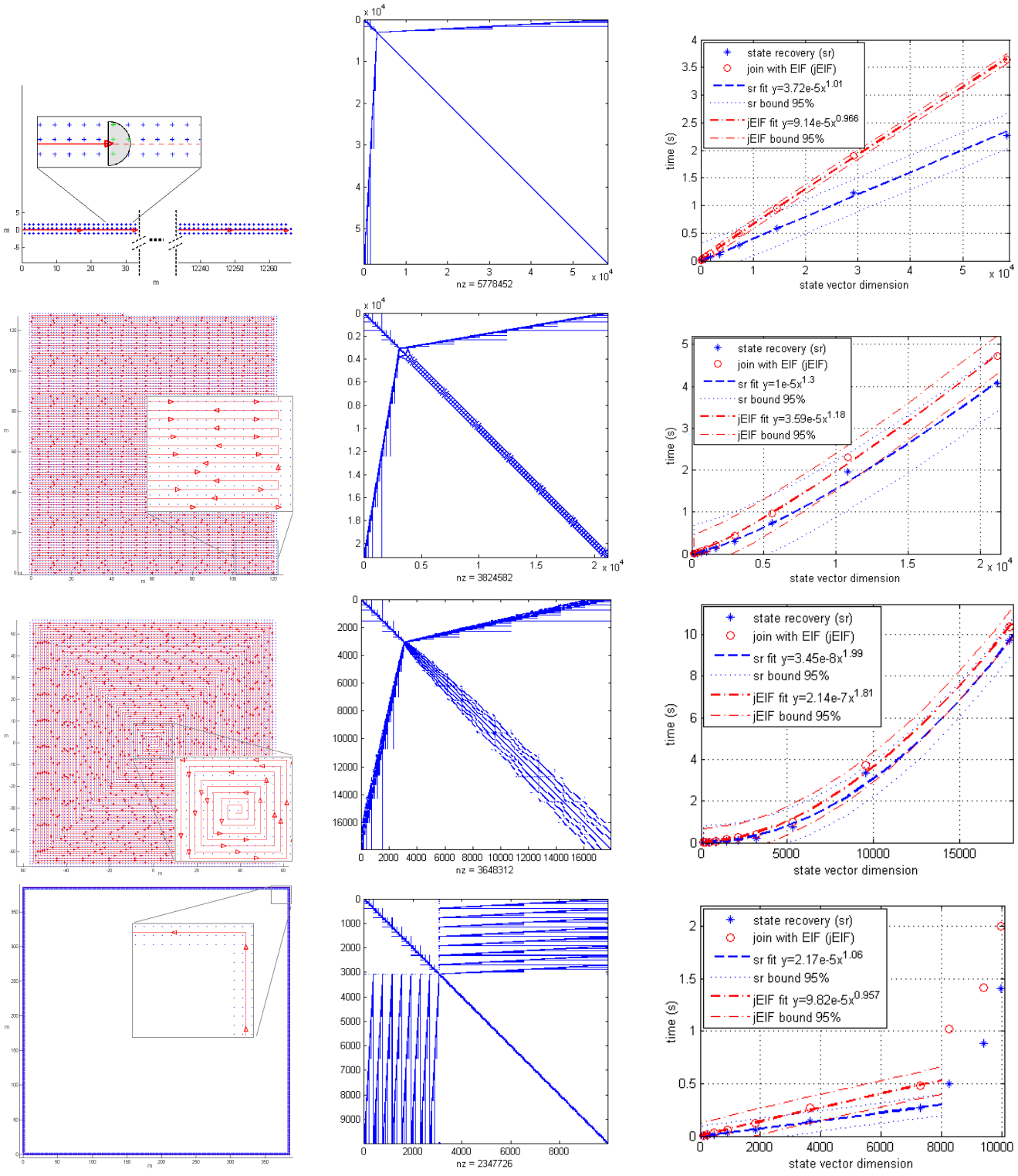

Fig. 3. Computational cost of state recovery in four cases: exploration with 27651 features (top), lawn mowing with 9056 features (upper middle), out-ward spiral with 7423 features (lower middle), several loops with 3456 features (bottom), all from 1024 local maps. From left to right: ground truth enviroment and trajectory, sparse information matrix obtained by our algorithm and execution time for to do joining and recovery state versus the state vector's dimension with their fit functions.

Trayectory		b	(95%)	SSE	R^2	RMSE
Exploration	sr	1.01	(0.956, 1.058)	0.1385	0.9976	0.1316
	jEIF	0.97	(0.955, 0.977)	0.0094	0.9999	0.0342
Lawn mowing	sr	1.30	(1.21, 1.39)	0.6849	0.9976	0.2926
	jEIF	1.18	(1.13, 1.237)	0.2900	0.9991	0.1904
Out-ward spiral	sr	1.99	(1.816, 2.157)	0.9726	0.9993	0.3497
	jEIF	1.81	(1.695, 1.92)	0.6629	0.9995	0.28797
Inside the loops	sr	1.06	(0.982, 1.139)	0.0072	0.9985	0.0379
	jEIF	0.96	(0.884, 1.03)	0.0119	0.9976	0.0487
Exploration with smallest local maps	sr	1.14	(1.097, 1.174)	0.1151	0.9990	0.1023
	jEIF	1.08	(1.032, 1.123)	0.1958	0.9980	0.1334

TABLE III

RESULTS OF THE FIT TO $y = ax^b$, BOTH THE COST OF STATE RECOVERY (`sr`) AND THE JOINING WITH EIF (`jEIF`). WE CAN SEE THE VALUE OF THE EXPONENT b WITH 95% CONFIDENCE BOUNDS, THE SUM OF SQUARED ERRORS (SSE), THE COEFFICIENT OF CORRELATION SQUARED (R^2) AND ROOT MEAN SQUARED ERRORS (RMSE) FOR THE SIMULATIONS OF FIG. 3 AND FIG. 4(RIGHT).

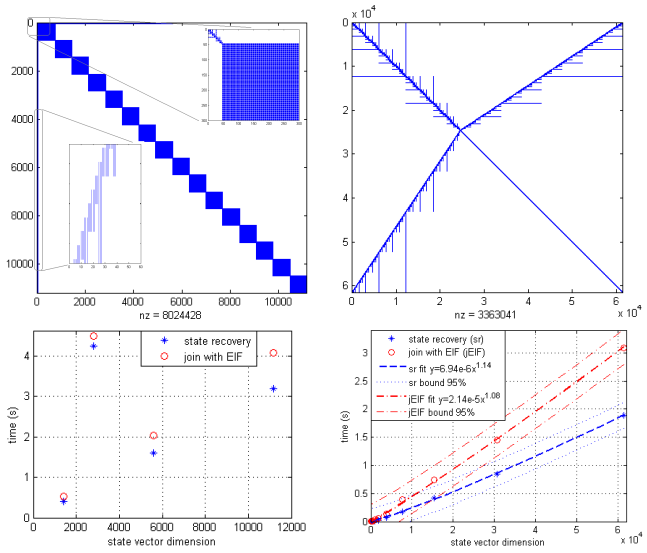

Fig. 4. Results of different sizes of local maps. (Top) The sparse matrix information and (bottom) execution time of state recovery and map joining, both in exploration trajectory. On the left the local map size is 350 features, note the time needed for a final map of 5564 features from 16 local maps. On the right the local map size is 4 features (field of view of the sensor) and the final map has 18431 features from 8192 local maps

requirements of the scenario on the left overflowing the capacity of MATLAB. In our experience, a good rule of thumb is that we should select p to keep the feature variable vs. pose variable ratio between 5 and 20, to 1.

V. EXPERIMENTS

The algorithm proposed here was also executed with the DLR circles dataset[1] and compared with the other two algorithms, see the accompanying video [2].

The complete experiment contains 3297 steps of odometry and 560 features; data association is provided. The size of each local map was fixed to 10 features, the final map was obtained from 249 local maps (so the feature/pose ratio is 6.66). Fig. 5 (a) shows the map resulting of the execution of CF SLAM (it was in fact very similar for D&C SLAM and SLSJF). In this experiment, the CF SLAM algorithm has a lower total computational cost compared with D&C SLAM and SLSJF (fig. 5, b), and its computation cost is less than that of SLSJF most of the time (fig. 5, c and d). In the amortized version, not yet implemented, the computational cost would be lower in all steps.

In [7] a total execution time of 2.95s was reported with the Treemap for this dataset, implemented in C++ on an Intel Xeon, 2.67 GHz. Our algorithm implemented in MATLAB spent 4.2s on 2.4 GHz Intel Core 2 CPU 6600. We believe that the CF SLAM algorithm is much more simple to implement and use. Furthermore, Treemap is expected to be less consistent in general, being an absolute map algorithm [11].

In real experiments, the step in which two local maps will have the same size and will thus be available for map joining is not known in advance. Amortization can be carried out by step number instead of by map size. As in [8], a simpler amortized version of our algorithm can be implemented in using two concurrent threads: one high priority thread executes `ekf_slam` to update the current local map, and the other low priority thread executes all pending map joinings. In this way, all otherwise idle time in the processor will be used for the more costly map joining operations, but high priority is given to local mapping, allowing for real time execution of our algorithm.

VI. DISCUSSION

In this paper we have proposed an algorithm that can carry out SLAM in $\mathcal{O}(\log n)$ per step. It brings together the advantages of different methods that have been proposed to optimize EKF and EIF SLAM. There is no loss of information, because the solution is computed without approximations, except for linearizations. We believe that it is conceptually simple and easy to implement. There are no restrictions on the topology of the environment and trajectory, although, as it is the case in all other SLAM algorithms, the efficiency will depend on this. It can be used with any geometric sensor and expanded without much difficulty to 6DOF.

One limitation of the approach is the unavailability of covariance matrices (except with a high computational cost) for data association. This is the case in any SLAM algorithm that makes use of the extended Information filter, including

[1] Available from the Workshop Inside Data Association in RSS'08 Conference, at http://www.sfbtr8.spatial-cognition.de/insidedataassociation/data.html

[2] A high resolution version of this video is available at http://webdiis.unizar.es/~ccadena/videos/dlr_dataset.avi

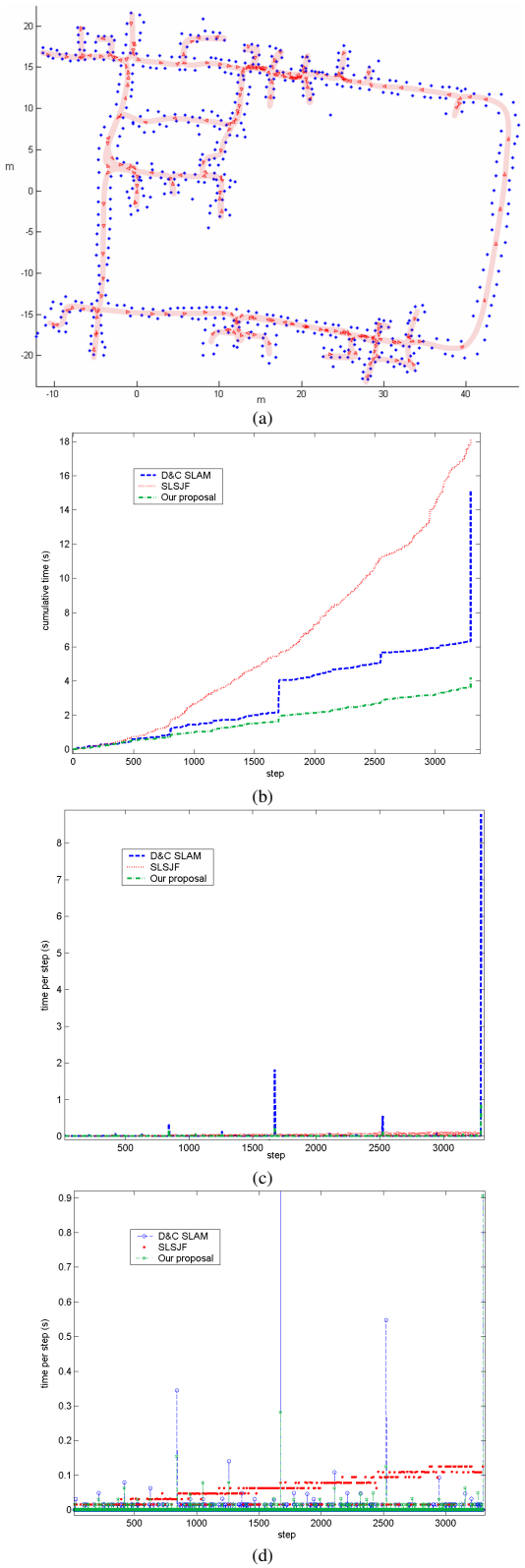

Fig. 5. Results for the DLR circles dataset: (a) final map obtained with our algorithm; (b) total cost of the three algorithms; (c) time per step for each algorithm; (d) zoom from (c)

SEIFs, ESEIFs, SLSJF. Also TreeMap does not have a data association algorithm. Several authors have proposed alternatives, such as exact covariance submatrix recovery [10], compute associations for local maps at the bottom level, where covariances are available [18], or using local maps to make local matching [19]. In some cases, for example if we have a vision sensor, additional information such as texture can be very useful for data association. But not general solution is known, so data association is an issue in which we must continue working on. In any case, we are getting closer to the ultimate goal of having a constant time SLAM algorithm.

REFERENCES

[1] T. Bailey and H. Durrant-Whyte, "Simultaneous localization and mapping (SLAM): Part II," *IEEE Robotics & Automation Magazine*, vol. 13, no. 3, pp. 108–117, 2006.

[2] J. Leonard and H. Feder, "Decoupled stochastic mapping," *IEEE Journal of Oceanic Engineering*, vol. 26, no. 4, pp. 561–571, 2001.

[3] J. Leonard and P. Newman, "Consistent, convergent and constant-time SLAM," in *Int. Joint Conf. Artificial Intelligence*, Acapulco, Mexico, August 2003.

[4] M. Bosse, P. M. Newman, J. J. Leonard, and S. Teller, "SLAM in large-scale cyclic environments using the atlas framework," *Int. J. Robotics Research*, vol. 23, no. 12, pp. 1113–1139, December 2004.

[5] J. D. Tardós, J. Neira, P. M. Newman, and J. J. Leonard, "Robust Mapping and Localization in Indoor Environments using Sonar Data," *Int. J. Robotics Research*, vol. 21, no. 4, pp. 311–330, 2002.

[6] S. B. Williams, G. Dissanayake, and H. Durrant-Whyte, "An efficient approach to the simultaneous localisation and mapping problem," in *Proc. IEEE Int. Conf. Robotics and Automation*, vol. 1, Washington DC, 2002, pp. 406–411.

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

[8] L. M. Paz, J. D. Tardós, and J. Neira, "Divide and conquer: Ekf slam in O(n)," *IEEE Trans. Robotics*, vol. 24, no. 5, pp. 1107–1120, October 2008.

[9] K. Ni, D. Steedly, and F. Dellaert, "Tectonic SAM: Exact, Out-of-Core, Submap-Based SLAM," in *2007 IEEE Int. Conf. on Robotics and Automation*, Rome, Italy, April 2007.

[10] S. Huang, Z. Wang, and G. Dissanayake, "Sparse local submap joining filters for building large-scale maps," *IEEE Trans. Robotics*, vol. 24, pp. 1121–1130, 2008.

[11] S. Huang and G. Dissanayake, "Convergence and Consistency Analysis for Extended Kalman Filter Based SLAM," *IEEE Trans. Robotics*, vol. 23, no. 5, pp. 1036–1049, October 2007.

[12] S. Thrun, W. Burgard, and D. Fox, *Probabilistic Robotics*. The MIT Press, September 2005.

[13] S. Thrun and J. Leonard, "Simultaneous Localization and Mapping," in *Springer Handbook of Robotics*, ser. Springer Handbooks, B. Siciliano and O. Khatib, Eds. Springer, 2008, ch. 37, pp. 871–889.

[14] A. Mutambara and M. Al-Haik, "State and information space estimation: a comparison," *American Control Conference, 1997. Proceedings of the 1997*, vol. 4, pp. 2374–2375 vol.4, Jun 1997.

[15] R. M. Eustice, H. Singh, and J. J. Leonard, "Exactly Sparse Delayed-State Filters for View-based SLAM," *IEEE Trans. Robotics*, vol. 22, no. 6, pp. 1100–1114, Dec 2006.

[16] R. Eustice, M. Walter, and J. Leonard, "Sparse extended information filters: Insights into sparsification," in *IEEE Int. Workshop on Intelligent Robots and Systems*, Edmonton, Alberta, Canada, August 2005.

[17] J. R. Gilbert, C. Moler, and R. Schreiber, "Sparse matrices in matlab: Design and implementation," *SIAM Journal on Matrix Analysis and Applications*, vol. 13, pp. 333–356, 1992.

[18] C. Estrada, J. Neira, and J. D. Tardós, "Hierarchical SLAM: real-time accurate mapping of large environments," *IEEE Trans. Robotics*, vol. 21, no. 4, pp. 588–596, August 2005.

[19] E. Olson, "Implicit data association from spectrally clustered local matches," in *Inside Data Association, held in RSS 2008*, Zurich, Switzerland, 2008.