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

C. Cadena, J. Neira

Dept. Informática e Ingeniería de Sistemas
Instituto de Investigación en Ingeniería de Aragón (I3A)
Universidad de Zaragoza
María de Luna 1, E-50018 Zaragoza, Spain

Abstract

In this paper[1] we describe the Combined Kalman-Information Filter SLAM algorithm (CF SLAM), a judicious combination of Extended Kalman (EKF) and Extended Information Filters (EIF) that can be used to execute highly efficient SLAM in large environments. CF SLAM is always more efficient than any other EKF or EIF algorithm: filter updates can be executed in as low as $O(\log n)$ as compared with $O(n^2)$ for Map Joining SLAM, $O(n)$ for Divide and Conquer (D&C) SLAM, and the Sparse Local Submap Joining Filter (SLSJF). In the worst cases, updates are executed in $O(n)$ for CF SLAM as compared with $O(n^2)$ for all others. We also study an often overlooked problem in computationally efficient SLAM algorithms: data association. In situations in which only uncertain geometrical information is available for data association, CF SLAM is as efficient as D&C SLAM, and much more efficient than Map Joining SLAM or SLSJF. If alternative information is available for data association, such as texture in visual SLAM, CF SLAM outperforms all other algorithms. In large scale situations, both algorithms based on Extended Information filters, CF SLAM and SLSJF, avoid computing the full covariance matrix and thus require less memory, but still CF SLAM is the most computationally efficient. Both simulations and experiments with the Victoria Park dataset, the DLR dataset, and an experiment using visual stereo are used to illustrate the algorithms' advantages, also with respect to non filtering alternatives such as iSAM, the Treemap and Tectonic SAM.

Key words: SLAM; Extended Kalman Filter; Extended Information Filter; Data Association

1. Introduction

In recent years, researches have devoted much effort to develop computational efficient algorithms for SLAM. The goal is being able to map large scale environments in real time [1]. Given a map of n features, the classical EKF SLAM algorithm is known to have a cost of $O(n^2)$ per update step. One important contribution has been the idea of splitting the full map into local

[1]This research has been funded by the Dirección General de Investigación of Spain under the projects DPI2009-13710 and DPI2009-07130. Preliminary versions of this work were presented in the Proceedings of the IEEE/RSJ International Conference on Intelligent RObots and Systems 2009, St. Louis, MO, USA, October 2009, and in the European Conference on Mobile Robotics 2009, Mlini/Dubrovnik, Croatia, September 2009.

Email addresses: `ccadena@unizar.es` (C. Cadena), `jneira@unizar.es` (J. Neira) maps and then putting the pieces back together in some way. Decoupled Stochastic Mapping [21], Constant Time SLAM [22] and the ATLAS system [4] are local mapping solutions close to constant time, although through approximations that reduce precision. Map Joining [32] and the Constrained Local Submap Filter [37] are exact solutions (except for linearizations) that require periodical $O(n^2)$ updates. Exact solutions also include Treemap [13], incremental Smoothing and Mapping (iSAM) [20], Divide and Conquer (D&C) SLAM [30], Tectonic SAM [28] and Sparse Local Submap Joining (SLSJF) SLAM [18]. Two recent algorithms have provided important reductions in computational cost: D&C SLAM has an amortized cost $O(n)$ per step, and SLSJF SLAM reports a cost $O(n^{1.5})$ per step in the worst cases. The Treemap has a cost $O(\log n)$, although with topological restrictions on the environment, and a rather complex implementation.

The SLAM problem has also been addressed using particle filters [33]. The particle filter keeps a number k of possible locations of the robot, and computes an alternative map for each. The most efficient SLAM algorithms based on particle filters factorize the problem so that only the vehicle location is represented with a set of particles, and have a cost $O(nk)$ per step [9, 16, 24, 36]. The consistency and robustness of these algorithms depends on the number k of particles in the filter. In order not to underestimate the uncertainty, this number must grow with the size of the environment that is expected to map.

When the application requires mainly to have a very accurate localization of the robot, and a detailed map is of secondary importance, the use of Pose based SLAM [10, 29, 15], or Topological Maps [8], can be interesting. These algorithms keep a state vector containing all or some the poses of the robot, the map must be calculated a posteriori.

In this paper we describe the Combined Filter SLAM (CF SLAM) algorithm, a highly efficient filtering algorithm; in the best cases it can reduce the computational cost from $O(n)$ down to $O(\log n)$ per step; the total computational cost can be reduced from $O(n^2)$ down to $O(n \log n)$; in the worst cases the cost is reduced from $O(n^2)$ down to $O(n)$ per step; and in the total computational cost from $O(n^3)$ down to $O(n^2)$ [6]. The CF SLAM algorithm is a judicious combination of Extended Kalman and Extended Information Filters, combined with a divide and conquer local mapping strategy and using the sparse Cholesky decomposition with a minimum degree preordering. Being a local mapping algorithm, it provides more consistent results, compared with Treemap, reported to have the same $O(\log n)$ cost but computing an absolute map [17]. CF SLAM is also conceptually simple and rather easy to implement. We also show that CF SLAM can compute the data association and remain much more efficient than any other EKF and EIF based SLAM algorithms [7]. In this paper we study in detail the computational complexity of methods based on filtering to show the advantages of CF SLAM. We also compare our results using benchmark datasets, like the Victoria Park and DLR datasets, with other state of the art algorithms such as the Treemap, iSAM and Tectonic SAM.

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 3 contains a description of our algorithm and a study of its computational cost and consistency properties. We discuss the main factors that may influence the computational cost in section 4. In section 5 we describe the process to solve the data association problem for both geometrical information only and appearance-only information in the CF SLAM algorithm. In section 6 we test the algorithm using four experiments: one simulated environment, the Victoria Park dataset, the DLR dataset and an experiment done with a stereo camera-in-hand. In the final section we summarize the results and draw the fundamental conclusions of this work.

	EKF-SLAM		EIF-SLAM	
Jacobians	$F_t = \frac{\partial g(u_t,\mu_{t-1})}{\partial \mu_{t-1}}\Big\vert_{\hat{\mu}_{t-1}}$	$O(1)$	$G_t = \frac{\partial g(u_t,\mu_{t-1})}{\partial u_t}\Big\vert_{\hat{u}_t}$ $O(1)$ $H_t = \frac{\partial h(\mu_{t\|t-1})}{\partial \mu_{t\|t-1}}\Big\vert_{\hat{\mu}_{t\|t-1}}$	$O(r)$
			$\mu_{t-1} = \mathbf{\Omega}_{t-1} \backslash \xi_{t-1}$	$O(nr^2)$-$O(n^2r)$
Prediction	$\mu_{t\|t-1} = g(u_t, \mu_{t-1})$	$O(1)$	$\Phi = F_t^{-T} \mathbf{\Omega}_{t-1} F_t^{-1}$	$O(1)$
	$\mathbf{\Sigma}_{t\|t-1} = F_t \mathbf{\Sigma}_{t-1} F_t^T + G_t R_{t-1} G_t^T$	$O(n)$	$\mathbf{\Omega}_{t\|t-1} = \Phi - \Phi G_t (R_{t-1} + G_t^T \Phi G_t)^{-1} G_t^T \Phi$	$O(n)$
			$\xi_{t\|t-1} = \mathbf{\Omega}_{t\|t-1} g(u_t, \mu_{t-1})$	$O(nr)$
Innovation	$\nu_t = z_t - h(\mu_{t\|t-1})$	$O(r)$	$\nu_t = z_t - h(\mu_{t\|t-1})$	$O(r)$
	$S_t = H_t \mathbf{\Sigma}_{t\|t-1} H_t^T + Q_t$	$O(r^3)$	$S_t = H_t(\mathbf{\Omega}_{t\|t-1} \backslash H_t^T) + Q_t$	$O(nr^2)$
Test χ^2	$D^2 = \nu_t^T S_t^{-1} \nu_t$	$O(r^3)$	$D^2 = \nu_t^T S_t^{-1} \nu_t$	$O(r^3)$
	$K_t = \mathbf{\Sigma}_{t\|t-1} H_t^T / S_t$	$O(nr^2)$		
Update	$\mathbf{\Sigma}_t = (I - K_t H_t) \mathbf{\Sigma}_{t\|t-1}$	$O(n^2r)$	$\mathbf{\Omega}_t = \mathbf{\Omega}_{t\|t-1} + H_t^T Q_t^{-1} H_t$	$O(r)$
	$\mu_t = \mu_{t\|t-1} + K_t \nu_t$	$O(n)$	$\xi_t = \xi_{t\|t-1} + H_t^T Q_t^{-1} (\nu_t + H_t \mu_{t\|t-1})$	$O(r)$
Cost per step	$O(n^2)$		$O(n)$ to $O(n^2)$	

Table 1: Formulations of the computational cost of each of the operations carried out using the Extended Kalman Filter (left) and the Extended Information Filter (right) in the SLAM problem. Variable n is the size of the final state μ_t or information vector ξ_t, and r is the size of the measurement vector z_t, **constant** in EKF-SLAM and EIF-SLAM. The test χ^2 is only required for data association.

2. The Extended Kalman Filter, The Extended Information Filter and Map Joining Filters

In this section we summarize the basic concepts about the basic Kalman Filter and the basic Information Filter, as well as about Map Joining techniques and the state of art SLAM algorithms that use them.

2.1. The Extended Kalman Filter

The Extended Kalman Filter (EKF) is one of the main paradigms in SLAM [34, 35]. 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 1 (left). During exploratory trajectories, and using a sensor of limited range thus providing a constant number of r features per observation, the size of the map (n) grows linearly. Given that each EKF update step is $O(n^2)$, the *total* cost of carrying out EKF SLAM is known to be $O(n^3)$.

2.2. 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 1, right. The Information filter is an algebraic equivalent to the Kalman filter, because the following equivalences hold [34]:

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

For this reason, KF and IF are considered dual filters [25]. 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 [34]. 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

	Join with EKF		Join with EIF	
Jacobians	$G = \frac{\partial g(\mu)}{\partial \mu}\Big\|_{\hat{\mu}}$	$O(n_2)$	$H = \frac{\partial h(\mu^-)}{\partial \mu^-}\Big\|_{\hat{\mu}^-}$	$O(n_2)$
Initialization			$\mu^- = g([\mu_1;\mu_2])$	$O(n_2)$
	$\mu^- = \begin{bmatrix} \mu_1 \\ \mu_2 \end{bmatrix}$	$O(n)$	$\xi^- = \begin{bmatrix} \xi_1 \\ 0 \end{bmatrix}$	$O(n)$
	$\mathbf{\Sigma}^- = \begin{bmatrix} \mathbf{\Sigma}_1 & 0 \\ 0 & \mathbf{\Sigma}_2 \end{bmatrix}$	$O(n^2)$	$\mathbf{\Omega}^- = \begin{bmatrix} \mathbf{\Omega}_1 & 0 \\ 0 & 0 \end{bmatrix}$	$O(np)$
Innovation	$\nu = -h(\mu^-)$	$O(s)$	$\nu = \mu_2 - h(\mu^-)$	$O(s)$
	$S = H\mathbf{\Sigma}^- H^T$	$O(s^2)$	$Q^{-1} = \mathbf{\Omega}_2$	given
Update	$K = \mathbf{\Sigma}^- H^T / S$	$O(n^2)$		
	$\mathbf{\Sigma}^+ = (I - KH)\mathbf{\Sigma}^-$	$O(n^2 s)$	$\mathbf{\Omega} = \mathbf{\Omega}^- + H^T \mathbf{\Omega}_2 H$	$O(n_2 p)$
	$\mu^+ = \mu^- + K\nu$	$O(ns)$	$\xi = \xi^- + H^T \mathbf{\Omega}_2(\nu + H\mu^-)$	$O(n_2 p)$
	$\mu = g(\mu^+)$	$O(n_2)$	$\mu = \mathbf{\Omega} \backslash \xi$	$O(np^2)$ to $O(n^2 p)$
	$\mathbf{\Sigma} = G\mathbf{\Sigma}^+ G^T$	$O(n_2^2)$		
Cost per join	$O(n^2 s)$		$O(n)$ to $O(n^2)$	

Table 2: Formulations of the 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) in the best case. Variables n_1, n_2 and n are the size of the first, second and final state or information vector respectively, s is the size of the overlap between both submaps, and p is the size of the local map, **constant** with respect to the size of the map.

computationally more expensive than the EKF: computing the state μ_{t-1} is $O(n^3)$ because of the inversion of the the information matrix $\mathbf{\Omega}_{t-1}$. In the SLAM context, the information matrix has special structural properties, thus state vector recovery can be carried out by solving a equation system of size n very efficiently through the Cholesky decomposition (we shall discuss this more in section 4). An important insight into reducing the computational cost of EIF SLAM was to observe that the information matrix $\mathbf{\Omega}$ is *approximately* sparse [34], and can be easily sparsified. This Sparse Extended Information Filter (SEIF) allows a computational cost of $O(n)$ (pure exploration) up to $O(n^2)$ (repeated traversal), although because of sparsification SEIF SLAM is not an exact algorithm.

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 [10]. In this Exactly Sparse Delayed-State Filter (ESDF) SLAM, the reduction in the computational cost is the same as in SEIF. Additionally, since no approximations due to sparsification take place, the results are more precise [11]. When the state or the 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.

The total cost of ESDF is known to be range from $O(n^2)$ (pure exploration) to $O(n^3)$ (repeated traversal), as compared with the always cubic cost of EKF SLAM. Note however that the information vector is ever increasing. Even during revisiting, every new vehicle location is incorporated in the state vector, thus increasing n. The term r refers to the field of view of the sensor, we note again that in these cases is constant with respect to n.

2.3. *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 maps of constant size p 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 [32], 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 2, left. A specialized version of the Extended 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 because EKF allows the consideration of 0 covariance measurements. In this case, h computes the discrepancy of features from both maps in the same reference. The update step includes a computation using the function g to first delete duplicate features appearing in both maps, and then 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, map joining operations are $O(n^2)$ on the final size of the map, and although it results in great computational savings (it may slash the cost by a large constant), Map Joining SLAM is still $O(n^2)$ per step, just as EKF SLAM is.

2.4. *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 [18]. Its application is not as straightforward as the Map Joining with EKF for two reasons. First, in Map Joining with EKF, 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 $(\mu_1, \mathbf{\Sigma}_1)$ and $(\mu_2, \mathbf{\Sigma}_2)$ to join, the resulting map $(\xi, \mathbf{\Omega})$ 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 ($z_t = \mu_2$, $Q_t = \mathbf{\Omega}_2^{-1}$), and the final update step computes the information state ξ and information matrix $\mathbf{\Omega}$ using the standard EIF equations. Note in table 2 that g has the same functionality as in the previous section, and function h transforms the features revisited from the first map to the reference of the second map.

An important observation made in [18] 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 map 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, such as was pointed out by [18]. In the section 4.1 we shall see that the cost of this computation can be proportional to n during exploration, and up to $O(n^2)$ in the worst case.

2.5. *Divide and Conquer with EKF*

In the Divide and Conquer (D&C) SLAM algorithm [30] 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 $O(n^2)$ updates are not carried out sequentially, but become more distant as the map grows. An $O(n^2)$ 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 $O(n^2)$, as compared to the total cost of EKF SLAM, $O(n^3)$.

3. Our proposal: Combined Filter SLAM

The algorithm proposed here, Combined Filter SLAM, has three main highlights (see algorithm 1):

1. *Local mapping is carried out using standard online EKF SLAM to build a sequence of maps of constant size p.* Each local map $(\mu_i, \mathbf{\Sigma}_i)$ is also stored in information form $(\xi_i, \mathbf{\Omega}_i)$; both the information vector and the information matrix are computed in $O(p^3)$, 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 [26], and small local maps remain consistent.
2. *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 resulting information matrix and the join can be carried out in as low as linear time with the final size of the map. The covariance matrix is not computed after joins at the lower level.
3. *In contrast with the 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 as low as $O(n \log n)$, as compared with the total cost of SLSJF, $O(n^2)$. Additionally, the computation per step can be amortized to $O(\log n)$, as compared with $O(n)$ for SLSJF. This makes CF SLAM, the most efficient SLAM filtering algorithm.

Algorithm 1 The CF SLAM Algorithm

```
maps ← {}
while data from sensor do
  map ← ekf_slam
  if isempty(maps) then
    maps ← {map}
  else
    while size(map) ≥ size(maps{last})
    or global map is needed do
      map ← eif_map_join(maps{last}, map)
      maps{last} ← {}
    end while
    maps ← {maps, map}
  end if
end while
```

3.1. *Computational complexity*

We study initially the case of pure exploration, in which the robot is always observing new territory, and thus the sensor measurements have a constant overlap with the map already built. This case is very interesting because is the first situation that any SLAM algorithm will face, and it is also the case where the size of the map increases, and thus also the size of the problem. We discuss several other cases in section 4.

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 $O(p^3)$ each, which are joined into $l/2$ maps of size $2p$, at cost $O(2p)$ each. These in turn are joined into $l/4$ maps of size $4p$, at cost $O(4p)$ 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 $O(n)$ (other types of trajectories are considered in section 4.1). 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.

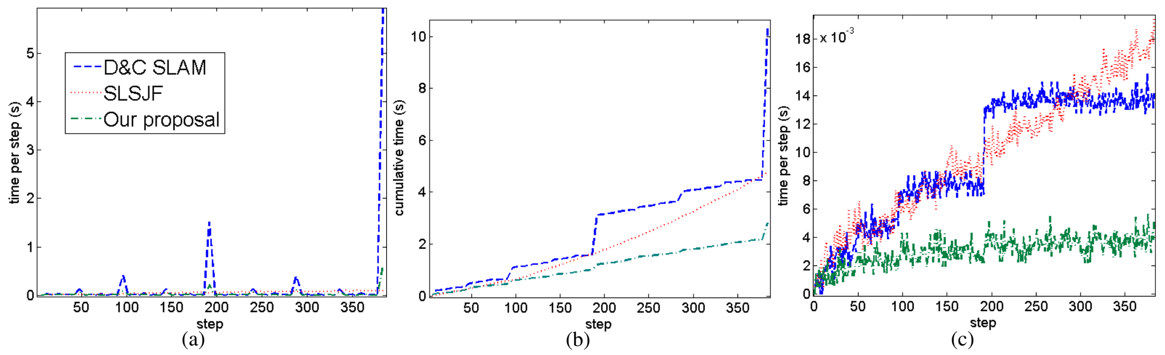
The *total* computational complexity of CF SLAM in this case is:

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

Therefore, CF SLAM offers a reduction in the total computational cost to $O(n \log n)$, 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 $2t$. This allows us to amortize the cost $O(t)$ at this step by dividing it up between steps $t+1$ to $2t$ in equal $O(1)$ computations for each step. In this way, our amortized algorithm becomes $O(\log n)$ *per step*. In the worst cases the cost can grow up to $O(n)$, but the cost of D&C SLAM and SLSJF SLAM will grow too (see Section 4).

3.2. *CF SLAM vs. other filtering 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 [30], and SLSJF [18] (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. Fig. 1 shows the resulting execution costs of the three algorithms. We can see that the total



(a) (b) (c)

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

costs of D&C SLAM and the SLSJF tend to be equal (Fig. 1, b). This is expected, since both have a total cost of $O(n^2)$. We can also see that the total cost of our algorithm increases more slowly, it is expected to be $O(n \log n)$. We can finally see that the amortized cost of our algorithm exhibits an $O(\log n)$ behavior, always smaller than the cost of the other two algorithms (Fig. 1, c).

3.3. 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:

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

where D^2 is the Mahalanobis distance or Normalized Estimation Error Squared (NEES) [2], $n = dim(\mu)$ 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, or 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 on one global map, (e.g. Treemap) [17]. We tested consistency of SLSJF, D&C SLAM and CF SLAM on the simulated experiments with 100 MonteCarlo runs. 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 the 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. In contrast, in SLSJF map joining is sequential, thus errors increase faster in the global map. Recently, a more consistent filter based on SLSJF, I-SLSJF, has been proposed [19]. This filter, in addition to being more computationally expensive than the original SLSJF, requires setting a threshold empirically to decide when it is necessary to solve the least squares problem and recompute the state vector from all the local maps stored.

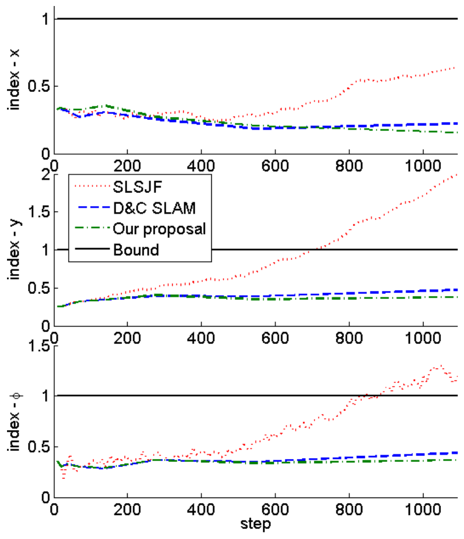


Figure 2: Mean consistency index CI in x, y, and ϕ for SLSJF, D&C SLAM and our proposal.

4. Factors that have influence in the computational cost

4.1. Vehicle trajectory

Given that local mapping is a constant time operation, we concentrate on the computational cost of map joining in CF SLAM. 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. The cost of this operation depends on the sparsity pattern of the information matrix and the density of non-zero elements [14, 18]. This in turn depends on the environment, on the sensor used and 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 shows the trajectory studied (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 compute a fit to equation $y = ax^b$ for the state recovery and for map joining costs. The independent variable is the dimension of the state vector resulting from each map joining. The dependent variable is the cost of the operation: `sr` for the state recovery $\mu = \mathbf{\Omega}\backslash\xi$, and `jEIF` is the cost of all operations involved in map joining with EIF, including `sr`. The results of the fit can be seen in Table 3. The sum of squared errors (SSE), the coefficient of correlation squared (R^2) and root mean squared errors (RMSE) are reported.

The most important results are:

- In pure *exploratory trajectories*, fig. 3 (top), the fit suggests that the exponent b can be considered equal to 1 in both state recovery and map joining operations, see Table 3 (top). 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 $O(\log n)$.

- We have also studied *lawn mowing*, fig. 3 (upper middle). This type of trajectory is frequent in underwater mosaicing applications [10]. In this case, the cost can increase to $O(n^{1.3})$ (see the exponent from the fit, Table 3).
- In another type of trajectory, *outward spiral*, fig. 3 (lower middle), the cost can increase to $O(n^2)$. Outward spirals are frequent in airborne mapping applications [5].
- In the worst case, *repeated traversal* (in this case a loop), fig. 3 (bottom), the cost of map joining is linear most of the time, except in the full overlap, when the operation is quadratic with the dimension of the state vector.

Three things are important to note:

1. 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 $O(n^2)$, CF SLAM is $O(n)$ per step.
2. 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 $O(n^2)$ per step, so CF SLAM will *always* be better.
3. Worst case situation will probably not very frequent, once a map of the environment of interest is avaiable, the robot can switch to localization using an *a priori* map, a much less expensive task computationally.

Table 4 summarizes the computational costs of all algorithms in the best and worst cases.

Trajectory		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 3: 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).

4.2. 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

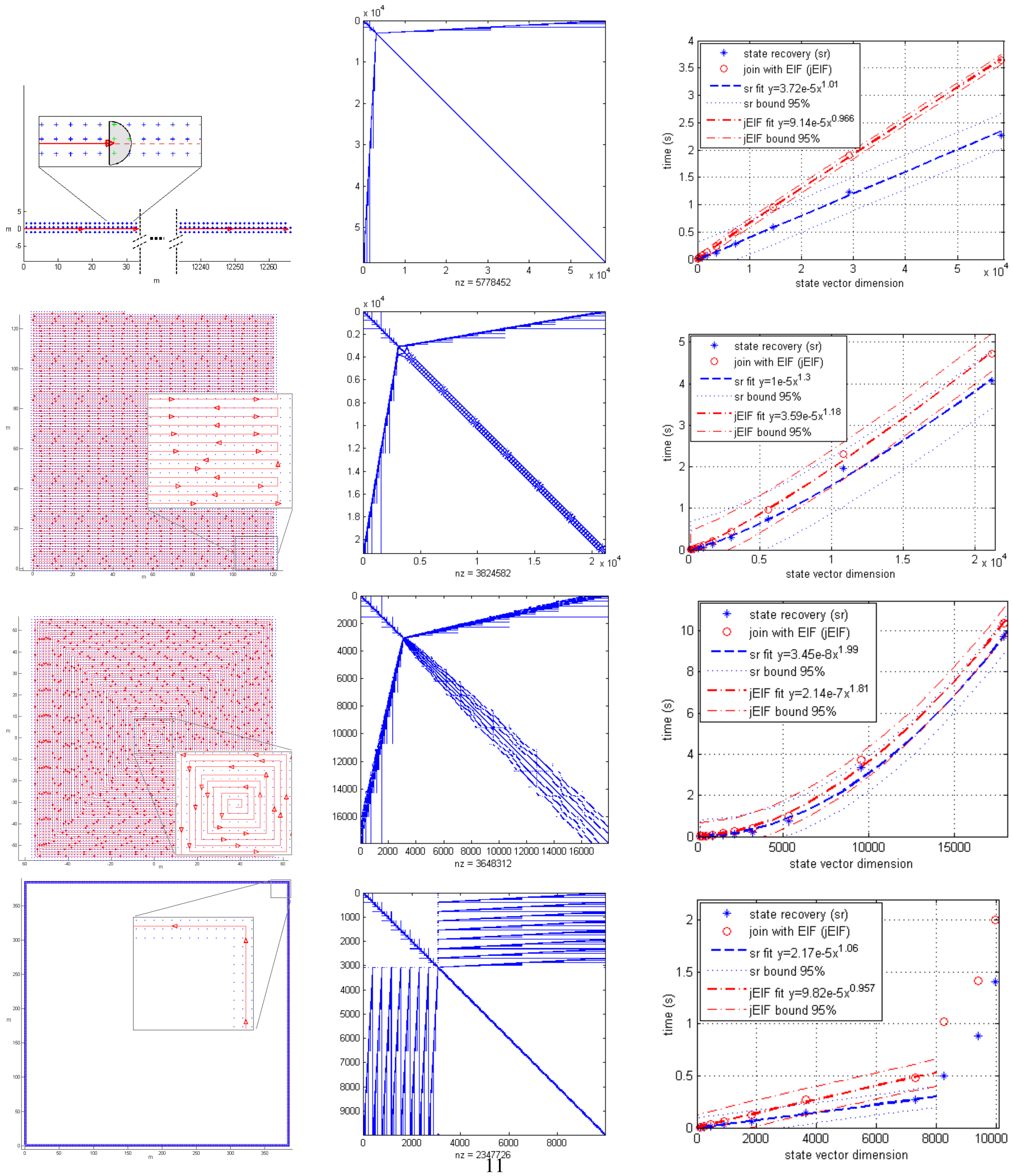


Figure 3: Computational cost of state recovery in four cases: exploration with 27651 features (top), lawn mowing with 9056 features (upper middle), out-ward spiral with 7423 features (lower middle), several loops with 3456 features (bottom), all from 1024 local maps. From left to right: ground truth environment 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. In order to concentrate in studying computational costs these simulations were carried out with noise equal to zero.

	Cost per step		Total cost	
	Best	Worst	Best	Worst
SLSJF	$O(n)$	$O(n^2)$	$O(n^2)$	$O(n^3)$
D&C SLAM	$O(n)$ amort.	$O(n^2)$ amort.	$O(n^2)$	$O(n^3)$
CF SLAM	$O(\log n)$ amort.	$O(n)$ amort.	$O(n \log n)$	$O(n^2)$

Table 4: Computational costs for all filtering algorithms in the best case (pure exploration) and in the worst case (repeated traversal). Note that the CF SLAM is the most efficient always.

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 in $\mathbf{\Omega}$ 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 3). The density is much lower, 1 in 1070, but the state vector is much larger because we have many more poses. In fig. 4 the number of features is different because the memory 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.

5. The Data Association problem

The problem of data association is often ignored when evaluating the efficiency and effectiveness of a SLAM algorithm. Here we show that the CF SLAM algorithm remains in the worst of case as computationally effective as D&C SLAM, the fastest to our knowledge that maintains the full covariance matrix and thus allows data association based on stochastic geometry. In the order of thousands of features, CF SLAM will outperform D&C SLAM because of reduced memory requirements.

Data association can be usually solved in two ways. If a covariance matrix is available, we can carry out statistical tests to find possible matches based on stochastic geometry. If additional information is available, such as texture or appearance in vision sensors, we can obtain matchings based not on location but on appearance.

In the following we describe two data association algorithms, one belonging to each category, that can be used in CF SLAM.

5.1. Data association using geometrical information

In some situations, only geometrical information, the uncertain location of features relative to the sensor location, is available for data association. Such is the case of 2 points or straight walls obtained with a laser sensor.

In CF SLAM, data association inside a local map is carried out using JCBB [26] because the corresponding covariance matrix is available. The data association that remains to be solved is the identification of features that appear in two consecutive maps, lets call them $M_1 = \{\mu_1, \xi_1, \mathbf{\Omega}_1\}$ and $M_2 = \{\mu_2, \xi_2, \mathbf{\Omega}_2\}$, either local maps at the lower level, or maps resulting from previous joins in the D&C map joining process. When the two maps to be joined are local maps, their corresponding covariances are available and data association can be done with the JCBB also.

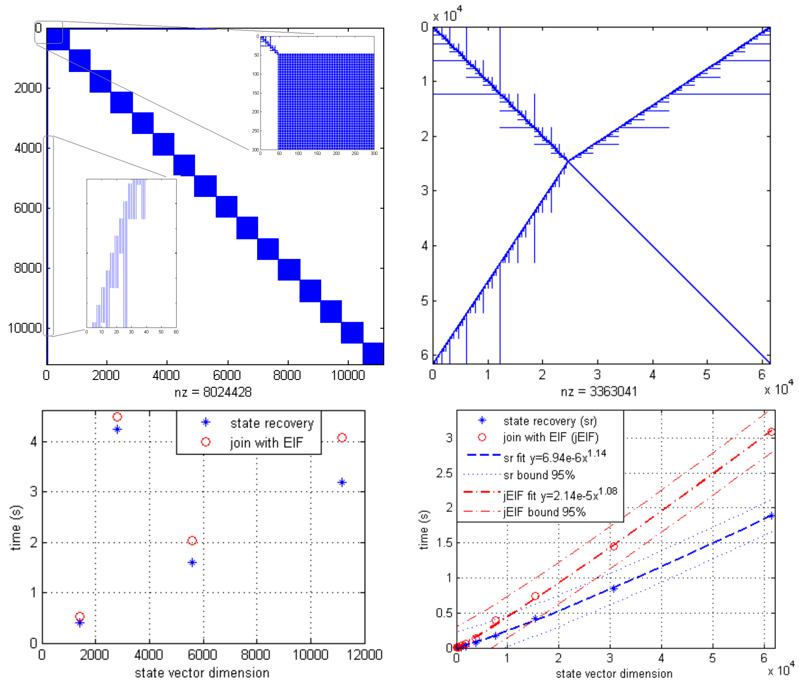


Figure 4: Results of different sizes of local maps. (Top) The sparse matrix information and (bottom) execution time of state recovery and map joining, both in exploration trajectory. On the left the local map size is 350 features, note the time needed for a final map of 5564 features from 16 local maps.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

If not, we first determine the overlap between the two maps, features that can potentially have pairings, and then we recover the covariance matrix for those features only. We proceed as follows:

1. *Identify the overlap, a set of potential matches*
   This is denominated individual compatibility, IC. Individually compatible features are obtained by tessellating the environment space, as was proposed in [30], but in our case we represent the grid in polar coordinates, see Fig. 5. For each feature in the second local map M_2, we assign an angular window of constant width in angle and height proportional to its distance from the origin, so that more distant features will have a larger region of uncertainty. The features in the first local map M_1 are referenced on M_2 through the last vehicle pose in M_1, μ_{x1}, which is the origin of map M_2. The uncertainty of μ_{x1} is recovered using equation 3 (below) and propagated on the these features, transformed to polar coordinates and embedded to angular windows. Features that intersect are considered individually compatible, giving IC.
2. *Partial recovery of covariances*
   For intermediate maps that are not at the lower local level CF SLAM does not compute covariances. As shown in [18], we can recover some columns of the covariance matrix by solving the sparse linear equation 3. The columns that we require are given by IC. We form a column selection matrix E_{IC} to obtain the columns that are given by IC. If column

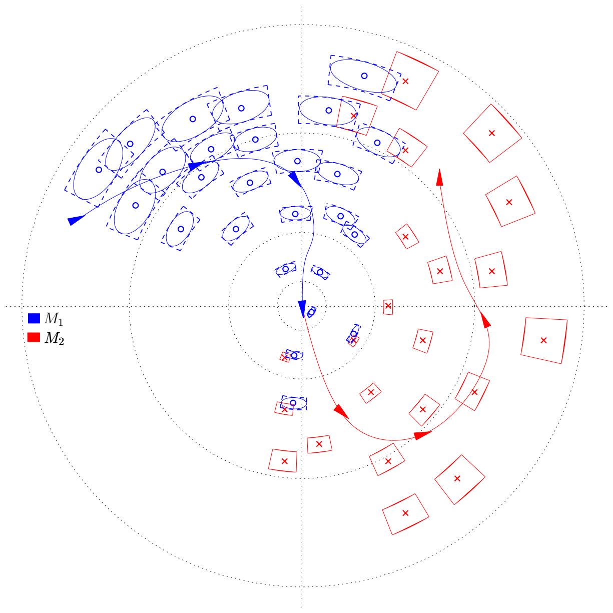


Figure 5: Computing the individual compatibility matrix for two local maps using polar coordinates. The angular windows for the features in the M_2 have constant width in angle and height proportional to the distance to the origin. Blue ellipses represent the uncertainties of the predicted features of the first local map with respect to the base reference of the second. The ellipses are approximated by bounding windows.

i of the covariance matrix is required, we include this column vector in E_{IC}:

$$e_i = [\overbrace{0, \cdots, 0, 1}^{i}, 0, \cdots, 0]^T$$

The sparse linear system to be solved is as follows:

$$\mathbf{\Omega\Sigma}_{IC} = E_{IC} \tag{3}$$

This partial recovery of the covariance matrix allows to use robust joint compatibility tests for data association. The efficiency of solving equation 3 is the same as for the recovery of the state vector: in the best case, in exploration trajectories, the order is $O(n)$, amortized $O(\log n)$. In the worst case, repeated traversal, the overlap will be the full map, and the cost will be $O(n^3)$, amortized $O(n^2)$. Here we reuse the Cholesky decomposition used for the state vector recovery.

3. *Prediction and Observation*
   At this point the features of M_1 that have potential matches according to IC, and their covariances, are transformed to be referenced on M_2.
4. *Randomized Joint Compatibility*
   We use the RJC algorithm of [30], a combination of JCBB and RANSAC that allows to carry out robust data association very efficiently, without traversing the whole solution space.

Algorithm 2 Data association for the Combined Filter using geometrical information only

```
Require: Two maps: $\langle M_1 = \{\mu_1, \xi_1, \mathbf{\Omega}_1\}, M_2 = \{\mu_2, \xi_2, \mathbf{\Omega}_2\}\rangle$
Ensure: Hypothesis $\mathcal{H}$
  Find the set of potential matches $IC \leftarrow (\mu_1, \mu_2)$
  if covariance matrices are available then
    extract covariances
    $(\mathbf{\Sigma}_{1i}, \mathbf{\Sigma}_{2j}) \leftarrow select(\mathbf{\Sigma}_1, \mathbf{\Sigma}_2, IC)$
  else
    recover partial covariances
    $(\mathbf{\Sigma}_{1i}, \mathbf{\Sigma}_{2j}) \leftarrow recoveryP(\mathbf{\Omega}_1, \mathbf{\Omega}_2, IC)$ eq: 3
  end if
  $predictions = (h, H\mathbf{\Sigma}H) \leftarrow predict\_map(\mu_{1i}, \mathbf{\Sigma}_{1i})$
  $observations = (z, R) \leftarrow (\mu_{2j}, \mathbf{\Sigma}_{2j})$
  $\mathcal{H} \leftarrow RJC(predictions, observations, IC)$
```

In the experimental section we will show that using this algorithm, CF SLAM is computationally as efficient as D&C SLAM but requires less memory because the full covariance matrix is not computed.

5.2. Data association using appearance information

In some cases, features in local maps can have associated appearance information, such as texture coming from vision. In these cases, appearance can be coded using a descriptor vector d, then $M = \{\mu, \xi, \mathbf{\Omega}, d\}$, for example SIFT [23] or SURF [3]. In these cases, we proceed as follows in CF SLAM:

1. *Obtain a set of potential matches*
   We find the best possible matches between the descriptors in M_1 and M_2 by searching for the nearest neighbor in the descriptor space. In this way we obtain the individual compatibility matrix, IC.
2. *Obtain a pairwise hypothesis using RANSAC*
   For each pair of minimum local maps, map i belonging to M_1 and map j belonging to M_2, that have a minimum number of matches (5 in our case) in IC, we use RANSAC [12] to find the subset $\mathcal{H}_{ij}$ of matches that corresponds to the best rigid-body transform between the two local maps. In Fig. 6, the transformations between the pairs $(i = 1, j = 3)$, $(i = 2, j = 3)$ and $(i = 3, j = 2)$ are not evaluated because they do not have sufficient matches.
3. *Obtain the final hypothesis*
   In most cases, the final hypothesis $\mathcal{H}$ is simply the result of joining all $\mathcal{H}_{ij}$. When there is ambiguity (one local map matched with two or more other local maps), we prefer the hypothesis for pairs of maps that have a smallest relative distance, because they have smaller relative errors. In fig 6 there is ambiguity between $\mathcal{H}_{41}$ and $\mathcal{H}_{42}$. In this case we accept hypothesis $\mathcal{H}_{41}$.

6. Simulations and Experiments

To compare the performance of the Combined Filter with other popular EKF and EIF based algorithms we use simulations, publicly available datasets, and our own stereo visual SLAM

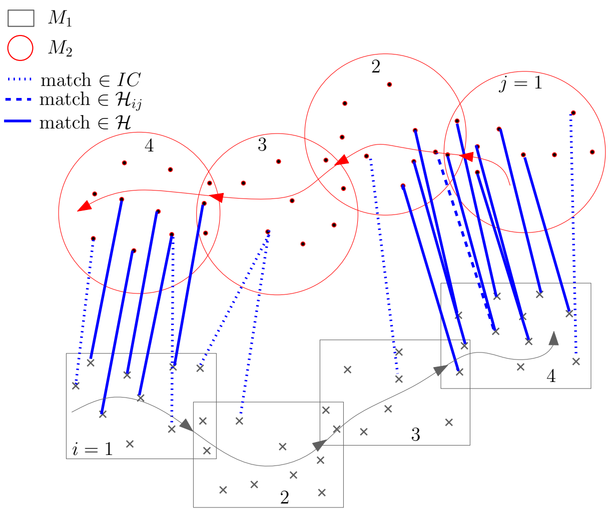


Figure 6: Computing the data association hypothesis from two local maps. The rectangles and circles show the minimum local maps of M_1 and M_2 respectively. All lines are potential matches, and form IC. The lines that are not dotted belong to a hypothesis between a pair of the minimum local maps, $\mathcal{H}_{ij}$. The solid lines represent the final hypothesis $\mathcal{H}$.

Algorithm 3 Data association using appearance information

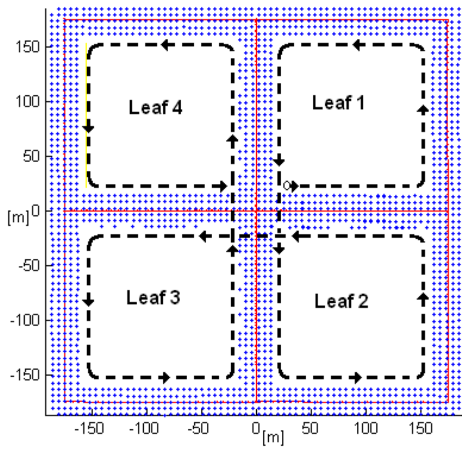
```
Require: Two maps: $\langle M_1 = \{\mu_1, \xi_1, \mathbf{\Omega}_1, d_1\}, M_2 = \{\mu_2, \xi_2, \mathbf{\Omega}_2, d_2\}\rangle$
Ensure: Hypothesis $\mathcal{H}$
  Find set of potential matches $IC \leftarrow (d_1, d_2)$
  for each pair minimum local maps $i, j$ in $IC$ do
    $\mathcal{H}_{ij} \leftarrow RANSAC(\mu_{1i}, \mu_{2j})$
  end for
  $\mathcal{H} \leftarrow select(\mathcal{H}_{ij})$
```

experiment[2]. All algorithms were implemented in MATLAB and executed on 2.4 GHz Intel Core 2 CPU 6600 and 3GB of RAM.

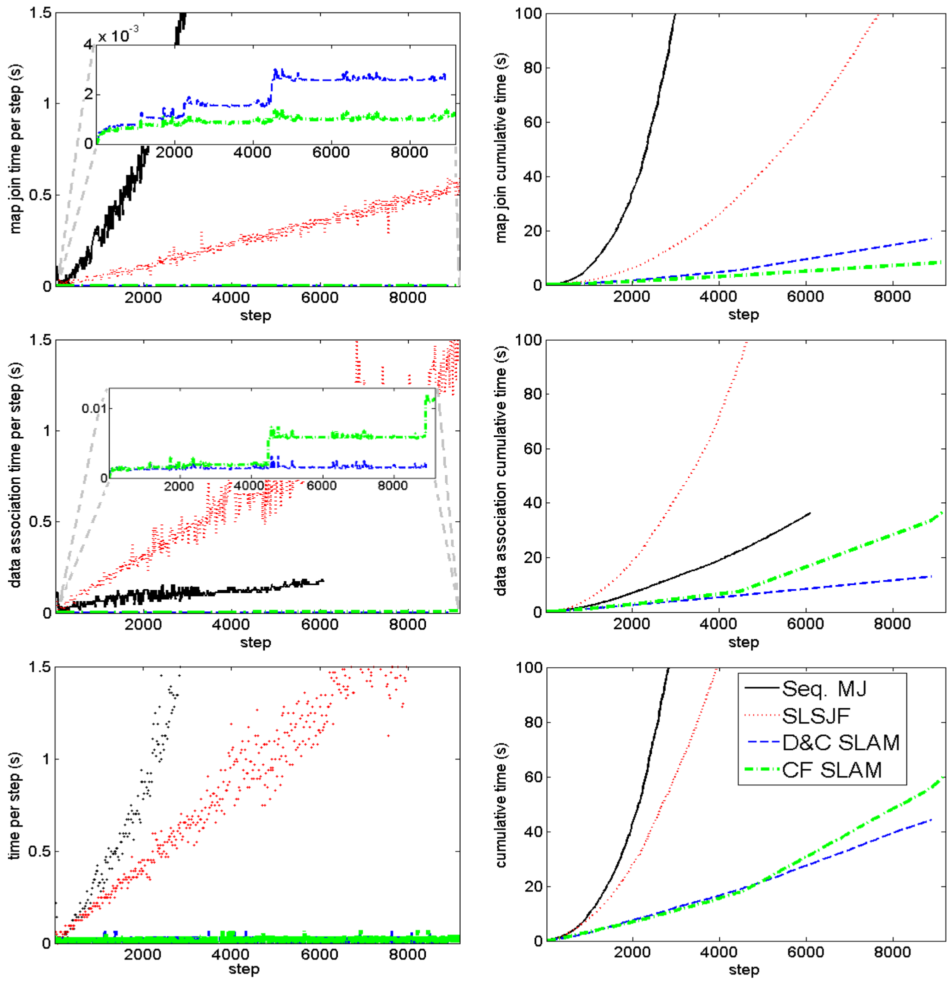
6.1. Simulated experiment

We simulated a robot moving in a 4-leaf clover trajectory, fig. 7(a). The robot is equipped with a range and bearing sensor. The features are uniformly distributed with a separation between them of $6m$. Data association was determined based only on geometric information using algorithm 2. Fig. 7(b) (left) shows the computation cost per step of Map Joining SLAM and SLSJF vs. the amortized cost for the D&C SLAM and CF SLAM: top, cost of map updates, center: cost of data association, bottom: total cost including local map building. We can see the sublinear cost of CF SLAM in the map updates as expected. Fig. 7(b) (right) shows the cumulative costs of map updates (top), data association (center) and total cost (bottom). The algorithms based on EKFs did not solve the problem completely because they exceeded the available memory before the end of the experiment.

[2]Videos are available at `http://webdiis.unizar.es/~ccadena/research.html`



(a)



(b)

Figure 7: Simulated experiment of a 4-leaf clover trajectory (a). In (b) the computational costs. Left: map update time per step (top), data association time per step (center), total time per step (bottom). Right: cumulative times for all algorithms.

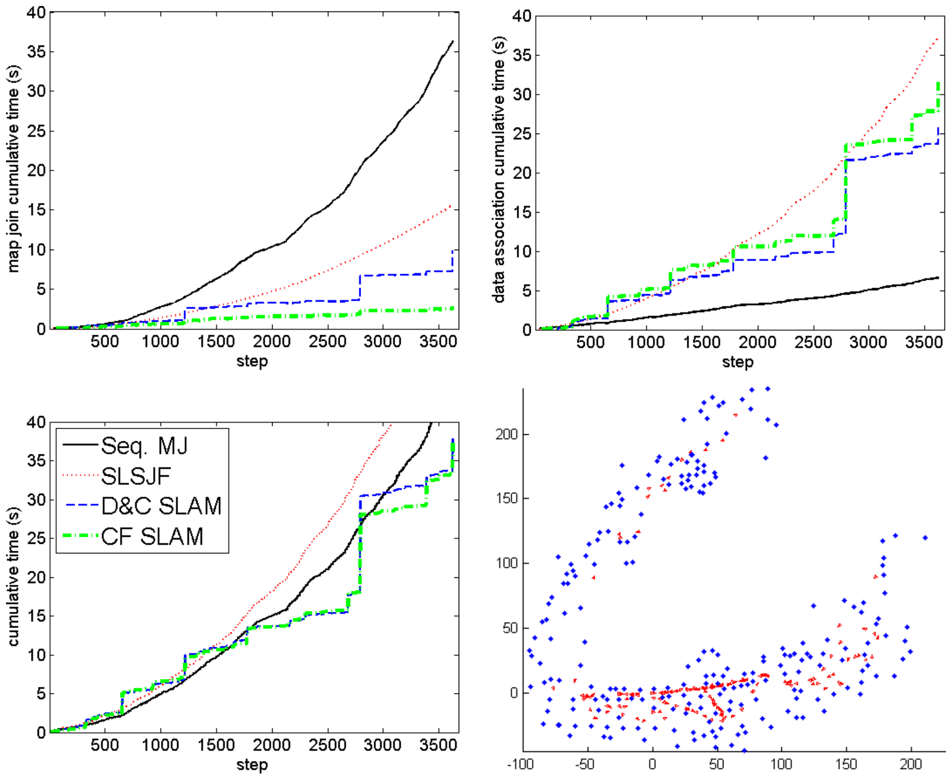


Figure 8: Results using the Victoria Park dataset. Top: cumulative cost for map updates (left) and data association (right). Bottom left, cumulative total cost including local map building, map joining and data association. Bottom right, the final map with CF SLAM.

6.2. The Victoria Park dataset

The figure 8 shows the resulting map obtained by CF SLAM on the Victoria Park dataset. All algorithms solve this dataset correctly. The trajectory of the vehicle explores and revisits frequently, so the uncertainty does not grow much and errors are kept small. The data association was determined with algorithm 2. This dataset is interesting to compare CF SLAM and SLSJF. Both algorithms require the recovery of the covariance submatrix for the overlap between the maps to be joined. There are some areas where the overlap is almost complete, thus requiring the recovery of almost the full covariance matrix. The cumulative computational costs are in fig. 8. We can see that CF SLAM is the most efficient for map updates, but Map Joining SLAM is most efficient for data association. In total, both algorithms that use the D&C strategy tend to be most efficient.

Different setups of Victoria Park are used in the literature, considering a different number of odometry steps and also different noise models for the sensor data. We sample one in every two odometry steps, processing a total of 3615. The iSAM algorithm has also used this dataset for its tests, sampling at every odometry step. In [20], it is implemented in OCaml on a 2 GHz Pentium M laptop computer. The authors report a cumulative time of 270s for the iSAM including the cost of data association, 159s with known data association. Comparatively, CF SLAM takes

38s for solving all the dataset: 3s for building local maps, 2.98s for map joins, and 32s for data association. Although our sampling frequency is one half, this only affects the speed of computation of the local maps, potentially doubling 3s of the 38s for a total of 41s, still much lower. The work in [27] recently reports results of Tectonic SAM (TSAM 2), a batch algorithm of smoothing and mapping with submapping, using also Victoria Park. They report a total time of 4.5s on an Macbook Pro with 2.8GHz CPU, without computing data association, or suggesting how this can be done in batch algorithms.

6.3. The DLR dataset

In the DLR dataset, the robot is equipped with a camera, and carries out a trajectory almost all indoors. Features are white cardboard circles placed on the ground. This dataset has 560 features and 3297 odometry steps. The path consists of a large loop with several smaller loops in the way. Position errors grow enough so that sequential algorithms (Map Joining SLAM and SLSJF) become weak and fail in the data association to close the loop (see fig. 9, bottom right). The D&C algorithms, D&C SLAM and CF SLAM have better consistency properties, and both solve the data association for the loop closing in this dataset (see fig. 9, left). The cumulative computational costs are show in fig. 9, right. In this mostly exploratory dataset, both D&C SLAM algorithms are clearly superior than both sequential algorithms. During loop closing, the cost of data association for both D&C algorithms is higher than that of both sequential algorithms, for the good reason that data association is computed correctly and the loop can be closed.

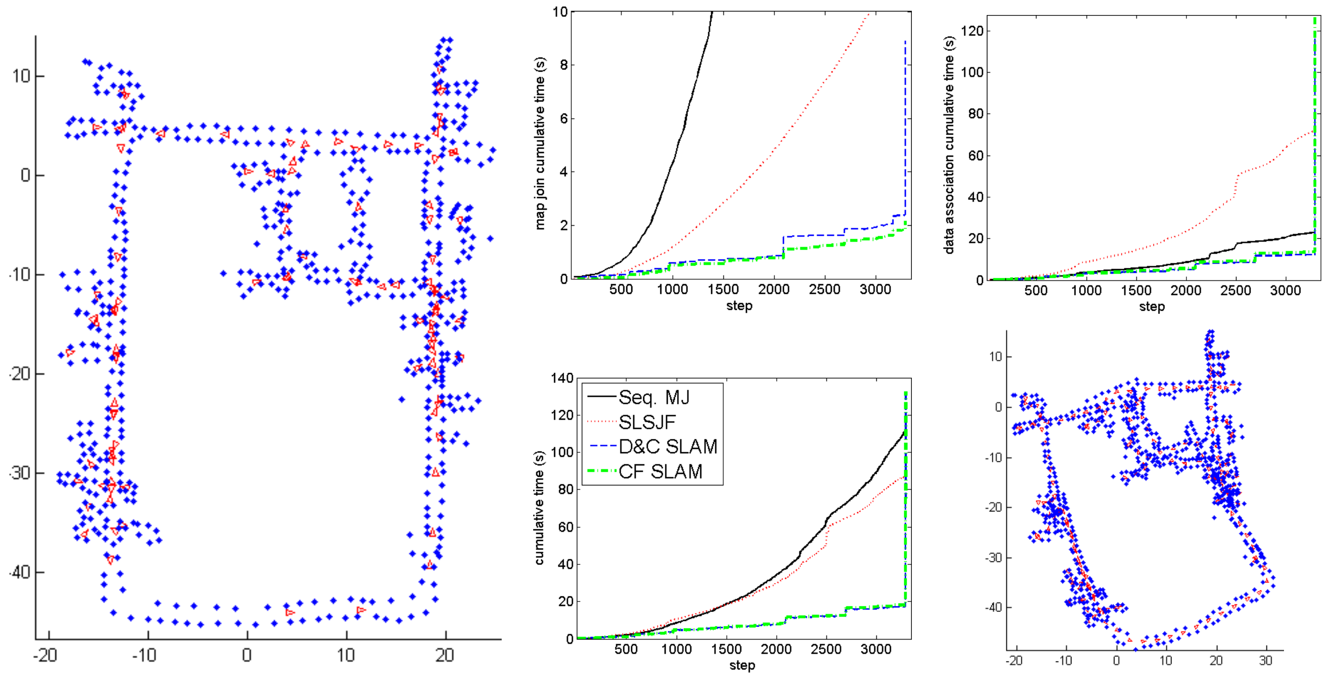


Figure 9: Results using the DLR-Spatial-Cognition dataset with D&C SLAM and CF SLAM (left), and with Map Joining SLAM and SLSJF (bottom right). Cumulative cost per map joining (center top), data association (top right) and total cost including local map building (center bottom).

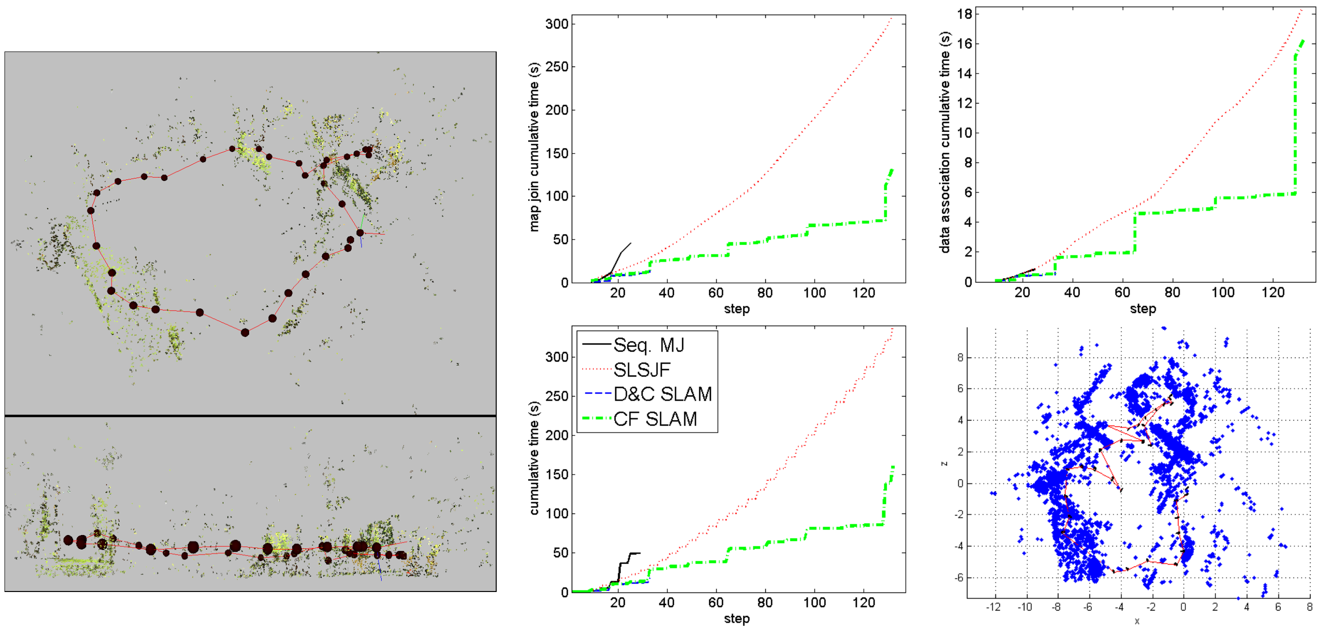


Figure 10: Final 3 map using 132 shoots of dense stereo data from a Triclops Pointgrey camera with the CF SLAM (left). We show the cumulative cost per map joining (center top), data association (top right) and total cost including build local maps, map joining and data association (center bottom). Map Joining SLAM and D&C SLAM exceed available memory capacity in shoots 24 and 32, respectively. The incorrect final map with SLSJF (bottom right).

In [13] a total execution time of 2.95s was reported with the Treemap for this dataset, without computing data association, implemented in C++ on an Intel Xeon, 2.67 GHz. Our algorithm implemented in MATLAB spent 4.84s, not counting the time devoted to data association. 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, it being an absolute map algorithm [17].

6.4. The visual stereo SLAM experiment

Finally, the CF SLAM was tested in a 3D environment with a high density of features. The sensor is a Triclops camera carried in hand. The path consists of a loop inside the Rose Building at the University of Sydney. We obtain the 3D position of points from the computation of the dense stereo point cloud that corresponds to each SIFT feature. The experiment consists of 132 shoots, with a total of 6064 features. Fig. 10 (left) shows the map obtained with the CF SLAM. The SLSJF obtains an incorrect map, fig. 10, bottom right. SLSJF is a sequential map joining algorithm, thus it is expected to provide less consistent results than D&C algorithms. In this experiments, this results in incorrect loop closure.

Both Map Joining SLAM and the D&C SLAM exceeded available memory in MATLAB. The data association is obtained with algorithm 3. Cumulative computational costs are shown in fig. 10: for map updates (top center), for data association (top right), and total cumulative cost (bottom center). CF SLAM clearly outperforms all the other algorithms.

7. Discussion and conclusions

In this paper we have proposed the Combined Filter, an algorithm that is always more efficient than any other filtering algorithm for SLAM. It can execute in as low as $O(\log n)$ per step. CF SLAM 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. 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 many SLAM algorithms, the computational efficiency will depend on this.

An important property of a SLAM algorithm is whether it is *on-line*, or provides the full vehicle and map estimation in every step, *delayed*, or providing a suboptimal estimation of the map and vehicle states at every step, but requiring additional computation in case the mission requires to have the best estimation, and finally *off-line*, or batch, carrying out the full computation only in case the vehicle and map states are required at a certain step. EKF and EIF SLAM, Map Joining, SLSJF, Treemap, and iSAM are on-line; D&C SLAM and CF SLAM, the algorithm that we present here, are delayed, and finally algorithms like Tectonic SAM are off-line. The selection of an appropriate algorithm for a specific mission must take this into account, it can an overkill to use an on-line algorithm in applications where the map is only required at the end of the mission or very infrequently during the mission.

CF SLAM provides the robot with a local map with all the information needed for local navigation tasks for a very low computational cost. This allows using processor time for other important tasks, such as image processing and data association. This can be essential in situations of limited computational power such as space exploratory rovers. If at a certain moment the robot needs all the environment information with respect to a global reference, for example to make decisions on global navigation like returning home, CF SLAM can provide the full map carrying out a single additional computation step for a cost as low as $O(n)$. In these situations, the robot will have a precise map to switch from SLAM to navigation using an *a priori* map. We think that such orders are usually less frequent during a mission in many applications, specially in large scale. For applications requiring global knowledge of the map and vehicle position at every step CF SLAM may be less adequate than sequential algorithms like Map Joining SLAM, SLSJF or iSAM, given the accumulated delay in computing the global map that CF SLAM incurs in. Finally, in applications where off-line processing is acceptable, we have shown that CF SLAM provides a solution in less time than any other algorithm discussed in this paper. Being a local mapping algorithm, the solution provided will have good consistency properties. Algorithms like Tectonic SAM that relinearize the full problem might provide even more consistent solutions. This is an interesting issue to be investigated.

The frequently overlooked problem of data association has also been addressed in this paper. We have shown that we can provide data association based on stochastic geometry that makes the CF SLAM algorithm as efficient as the most efficient algorithm that computes covariance matrices, with far less memory requirements. If appearance information is available for data association, then CF SLAM outperforms all algorithms discussed in this paper.

From our experiments it is clear that the greatest computational weight can lie in data association. Our future work includes the development of more robust and more efficient data association techniques to use in CF SLAM. Among the most promising appearance-based techniques, we will consider the 'bag-of-words' methods [8] and Conditional Random Fields [31].

References

[1] T. Bailey and H. Durrant-Whyte. Simultaneous Localization And Mapping (SLAM): Part II. *IEEE Robotics & Automation Magazine*, 13(3):108–117, 2006.

[2] Y. Bar-Shalom, X. R. Li, and T. Kirubarajan. *Estimation with Applications to Tracking and Navigation*. John Willey and Sons, New York, 2001.

[3] H. Bay, T. Tuytelaars, and L. Van Gool. SURF: Speeded Up Robust Features. In *Proceedings of the 9th European Conference on Computer Vision*, volume 3951, pages 404–417. Springer LNCS, 2006.

[4] M. Bosse, P. Newman, J. Leonard, and S. Teller. SLAM in large-scale cyclic environments using the ATLAS framework. *Int. J. Robotics Research*, 23(12):1113–1139, December 2004.

[5] M. Bryson and S. Sukkarieh. Observability Analysis and Active Control for Airborne SLAM. *IEEE Transactions on Aerospace Electronic Systems*, 44:261–280, January 2008.

[6] C. Cadena and J. Neira. SLAM in O(log n) with the Combined Kalman - Information Filter. In *Proc. IEEE/RJS Int. Conference on Intelligent Robots and Systems*, 2009.

[7] C. Cadena, F. Ramos, and J. Neira. Efficient large scale SLAM (including data association) using the Combined Filter. In *European Conference on Mobile Robotics, ECMR*, 2009.

[8] M. Cummins and P. Newman. FAB-MAP: Probabilistic Localization and Mapping in the Space of Appearance. *The International Journal of Robotics Research*, 27(6):647–665, 2008.

[9] A.I. Eliazar and R. Parr. DP-SLAM 2.0. In *Robotics and Automation, 2004. Proceedings. ICRA '04. 2004 IEEE International Conference on*, volume 2, pages 1314–1320 Vol.2, 26-May 1, 2004.

[10] R. Eustice, H. Singh, and J. J. Leonard. Exactly Sparse Delayed-State Filters for View-based SLAM. *IEEE Trans. Robotics*, 22(6):1100–1114, Dec 2006.

[11] R. Eustice, M. Walter, and J. Leonard. Sparse extended information filters: Insights into sparsification. In *Proc. IEEE/RJS Int. Conference on Intelligent Robots and Systems*, Edmonton, Alberta, Canada, August 2005.

[12] M. A. Fischler and R. C. Bolles. Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. *Commun. ACM*, 24(6):381–395, 1981.

[13] U. Frese. Treemap: An O(log n) algorithm for indoor Simultaneous Localization And Mapping. *Autonomous Robots*, 21(2):103–122, September 2006.

[14] J. R. Gilbert, C. Moler, and R. Schreiber. Sparse Matrices in MATLAB: Design and implementation. *SIAM Journal on Matrix Analysis and Applications*, 13:333–356, 1992.

[15] G. Grisetti, D. L. Rizzini, C. Stachniss, E. Olson, and W. Burgard. Online Constraint Network Optimization for Efficient Maximum Likelihood Mapping. In *ICRA*, San Diego, USA, October 2008.

[16] G. Grisetti, G.D. Tipaldi, C. Stachniss, W. Burgard, and D. Nardi. Fast and accurate SLAM with Rao-Blackwellized particle filters. *Robotics and Autonomous Systems*, 55(1):30–38, 2007.

[17] S. Huang and G. Dissanayake. Convergence and Consistency Analysis for Extended Kalman Filter Based SLAM. *IEEE Trans. Robotics*, 23(5):1036–1049, October 2007.

[18] S. Huang, Z. Wang, and G. Dissanayake. Sparse local submap Joining Filters for building large-scale maps. *IEEE Trans. Robotics*, 24:1121–1130, 2008.

[19] S. Huang, Z. Wang, G. Dissanayake, and U. Frese. Iterated SLSJF: A Sparse Local Submap Joining Algorithm with Improved Consistency. In *Proceedings of the Australasian Conference on Robotics and Automation*, 2008.

[20] M. Kaess, A. Ranganathan, and F. Dellaert. iSAM: Incremental Smoothing and Mapping. *IEEE Trans. on Robotics, TRO*, 24(6):1365–1378, Dec 2008.

[21] J. Leonard and H.S.J. Feder. Decoupled Stochastic Mapping. *IEEE Journal of Oceanic Engineering*, 26(4):561–571, 2001.

[22] J. Leonard and P. Newman. Consistent, Convergent and Constant-Time SLAM. In *Int. Joint Conf. Artificial Intelligence*, Acapulco, Mexico, August 2003.

[23] D. Lowe. Distinctive Image Features from Scale-Invariant Keypoints. *International Journal of Computer Vision*, 60(2):91–110, 2004.

[24] M. Montemerlo, S. Thrun, D. Koller, and B.Wegbreit. FastSLAM 2.0: An Improved Particle Filtering Algorithm for Simultaneous Localization and Mapping that Provably Converges. In *Int. Joint Conf. Artificial Intelligence*, 2003.

[25] A.G.O. Mutambara and M.S.Y. Al-Haik. State and Information space estimation: A comparison. *American Control Conference, 1997. Proceedings of the 1997*, 4:2374–2375 vol.4, Jun 1997.

[26] J. Neira and J. D. Tardós. Data Association in Stochastic Mapping Using the Joint Compatibility Test. *IEEE Trans. Robotics and Automation*, 17(6):890–897, 2001.

[27] K. Ni and F. Dellaert. Multi-Level Submap Based SLAM Using Nested Dissection. In *Proc. IEEE/RJS Int. Conference on Intelligent Robots and Systems*, 2010.

[28] 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.

[29] E. Olson, J. Leonard, and S. Teller. Fast iterative optimization of pose graphs with poor initial estimates. In *Proc. IEEE Int. Conf. Robotics and Automation*, pages 2262–2269, 2006.

[30] L. M. Paz, J. D. Tardós, and J. Neira. Divide and Conquer: EKF SLAM in O(n). *IEEE Trans. Robotics*, 24(5):1107–1120, October 2008.

[31] F. Ramos, D. Fox, and H. Durrant-Whyte. CRF-Matching: Conditional Random Fields for Feature-Based Scan Matching. In *Robotics: Science and Systems (RSS)*, 2007.

[32] J. D. Tardós, J. Neira, P. Newman, and J. Leonard. Robust Mapping and Localization in Indoor Environments using Sonar Data. *Int. J. Robotics Research*, 21(4):311–330, 2002.

[33] S. Thrun. Particle Filters in Robotics. In *Proceedings of Uncertainty in AI (UAI)*, 2002.

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

[35] S. Thrun and J. Leonard. Simultaneous Localization and Mapping. In B. Siciliano and O. Khatib, editors, *Springer Handbook of Robotics*, Springer Handbooks, chapter 37, pages 871–889. Springer, 2008.

[36] D. Törnqvist, T. B. Schön, R. Karlsson, and F. Gustafsson. Particle Filter SLAM with High Dimensional Vehicle Model. *J. Intell. Robotics Syst.*, 55(4-5):249–266, 2009.

[37] 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*, volume 1, pages 406–411, Washington DC, 2002.