Divide and Conquer: EKF SLAM in $O(n)$

Lina M. Paz, *Student Member, IEEE*, Juan D. Tardós, *Member, IEEE*, and José Neira, *Member, IEEE*

***Abstract*—In this paper we show that *all* processes associated to the move-sense-update cycle of EKF SLAM can be carried out in time *linear* with the number of map features. We describe Divide and Conquer SLAM, an EKF SLAM algorithm in which the computational complexity per step is reduced from $O(n^2)$ to $O(n)$; the total cost of SLAM is reduced from $O(n^3)$ to $O(n^2)$. Unlike many current large scale EKF SLAM techniques, this algorithm computes a solution without relying on approximations or simplifications (other than linearizations) to reduce computational complexity. Also, estimates and covariances are available when needed by data association without any further computation. Furthermore, as the method works most of the time in local maps, where angular errors remain small, the effect of linearization errors is limited. The resulting vehicle and map estimates are more precise than those obtained with standard EKF SLAM: the errors with respect to the true value are smaller, and the computed state covariance is consistent with the real error in the estimation. Both simulated experiments and the Victoria Park dataset are used to provide evidence of the advantages of this algorithm.**

***Index Terms*—SLAM, Linear time, Computational Complexity, Precision, Consistency.**

I. Introduction

THE Simultaneous Localization and Mapping (SLAM) problem deals with the construction of a model of the environment being traversed with an onboard sensor, while at the same time maintaining an estimation of the sensor location within the model [3], [4]. Solving SLAM is central to the effort of conferring real autonomy to robots and vehicles, but also opens possibilities in applications where a sensor moves with six degrees of freedom, such as augmented reality. SLAM has been the subject of much attention since the seminal work in the late 80s [5], [6], [7], [8].

The most popular solution to SLAM considers it a stochastic process in which the Extended Kalman Filter (EKF) is used to compute an estimation of a state vector $\mathbf{x}$ representing the sensor and environment feature locations, together with the covariance matrix $\mathbf{P}$ representing the error in the estimation. Most processes associated to the move-sense-update cycle of EKF SLAM are linear in the number of map features n: vehicle prediction and inclusion of new features [9], [10]. The exception is the update of the covariance matrix $\mathbf{P}$ of the stochastic state vector that represents the vehicle and map states, which is $O(n^2)$. The EKF solution to SLAM has been used successfully in small scale environments, however the $O(n^2)$ computational complexity limits the use EKF SLAM in large environments. This has been a subject of much interest in research. Postponement [11], the Compressed EKF filter [10], the Sparse Weight Kalman Filter [12] and Map Joining SLAM [13] are alternatives that work on local areas of the stochastic map and are essentially constant time most of the time, although they require periodical $O(n^2)$ updates. Given a certain environment and sensor characteristics, an optimal local map size can be derived to minimize the total computational cost [14]. Recently, researchers have pointed out the approximate sparseness of the Information matrix $\mathbf{Y}$, the inverse of the full covariance matrix $\mathbf{P}$. This suggests using the Extended Information Filter, the dual of the Extended Kalman Filter, for SLAM updates. The Sparse Extended Information Filter (SEIF) algorithm [15] approximates the Information matrix by a sparse form that allows $O(1)$ updates on the information vector. Nonetheless, data association becomes more difficult when the state and covariance matrix are not available, and the approximation can yield overconfident estimations of the state [16]. This overconfidence is overcome by the Exactly Sparse Extended Information Filter (ESEIF) [17] with a strategy that produces an exactly sparse Information matrix with no introduction of inaccuracies through sparsification, but discarding odometry information.

The Thin Junction Tree Filter algorithm [18] works on the Gaussian graphical model represented by the Information matrix, and achieves high scalability by working on an *approximation*, where weak links are broken. The Treemap algorithm [19] is a closely related technique: in cases where there are many overlapping features, an optional optimization uses a weak link breakage policy. Recently, the Exactly Sparse Delayed State Filter [20] and Square Root SAM [21] provided the insight that the full SLAM problem, the complete vehicle trajectory plus the map, is sparse in information form (although ever increasing) allowing the use of sparse linear algebra techniques. The Tectonic SAM algorithm [22] provides a local mapping version to reduce the computational cost. However, the method remains a batch algorithm and covariance is not available to solve data association. The Fast Incremental Smoothing and Mapping (iSAM) algorithm [23] addresses the data association problem by recovering the exact covariance using QR-factorization on the Information matrix. The authors report that iSAM is real time for the Victoria Park dataset (although EKF SLAM is also real time for this dataset).

A second important limitation of standard EKF SLAM is the effect that linearizations have in the precision and consistency of the final vehicle and feature estimates. Lin-

This research has been funded in part by the European Union under project RAWSEEDS FP6-IST-045144 and the Dirección General de Investigación of Spain under project SLAM6DOF DPI2006-13578.

L.M. Paz, J.D. Tardós and J. Neira are with Instituto de Investigación en Ingeniería de Aragón (I3A), Universidad de Zaragoza, María de Luna 1, E-50018, Zaragoza, Spain, e-mail: {linapaz, tardos, jneira}@unizar.es

Preliminary versions of this work were presented at the 2007 IEEE Int. Conf. on Robotics and Automation [1] and at the 2007 Robotics: Science and Systems conference [2].

earizations introduce errors in the estimation process that reduce precision and can render the result inconsistent, in the sense that the computed state covariance does not represent the real error in the estimation [24], [25], [26]. Among other things, this complicates data association, which is based on contrasting predicted feature locations with observations made by the sensor. Thus, important processes in SLAM like loop closing are complicated. All algorithms for EKF SLAM based on efficiently computing an approximation of the EKF solution will inevitably suffer from this consistency problem [18], [19]. The Unscented Kalman Filter [27] avoids linearizations via a parametrization of means and covariances through selected points to which the nonlinear transformation is applied. Unscented SLAM has been shown to have improved consistency properties [28]. These solutions however ignore the computational complexity problem.

In this paper we describe Divide and Conquer SLAM (D&C SLAM), an EKF SLAM algorithm that overcomes these two fundamental limitations:

1) The computational cost per step is reduced from $O(n^2)$ to $O(n)$; the total cost of SLAM is reduced from $O(n^3)$ to $O(n^2)$;
2) the resulting vehicle and map estimates are more precise than with standard EKF SLAM and the computed state covariance more adequately represents the real error in the estimation.

Unlike many current large scale EKF SLAM techniques, this algorithm computes a solution without relying on approximations or simplifications (other than linearizations) to reduce computational complexity. Also, estimates and covariances are available when needed by data association without any further computation. The Victoria Park dataset is used to test this algorithm. It produces better results than other state of the art SLAM methods [29], [23] for this dataset.

In a recent paper [30] the authors prove that convergence properties known to hold for linear EKF SLAM [31] hold for nonlinear EKF SLAM only when Jacobians (and thus linearizations) are computed around the ground truth solution, which is in general unfeasible. It is also proven that the use of Jacobians computed around the estimated value may result in overconfident estimates. In particular, when the robot orientation uncertainty is large, the extent of inconsistency is significant. Likewise, when the robot orientation uncertainty is small, the extent of inconsistency is insignificant. The authors point out that since the robot orientation error is the main source of inconsistency, algorithms that use local submapping, where the robot orientation error remains small, have the potential to improve consistency. Here we confirm the consistency improvement offered by one of such local submapping algorithms.

This paper is organized as follows: in section II we briefly review the standard EKF SLAM algorithm and its computational properties. We also discuss other recent alternative algorithms, based on local mapping, with reduced computational cost. In section III we describe the D&C SLAM algorithm, and study its computational cost as well as its consistency properties in comparison with EKF SLAM and Map Joining SLAM. In section IV we compare the computational cost, precision and consistency of EKF SLAM and D&C SLAM using simulated experiments. In section V we analyze the computational cost of continuous data association in EKF SLAM, and describe the Randomized Joint Compatibility (**RJC**) algorithm for carrying out data association in D&C SLAM also in linear time. In section VI we use the Victoria Park dataset to carry out an experimental comparison between EKF SLAM and D&C SLAM. Finally in section VII we draw the main conclusions of this work.

Algorithm 1 : $\mathbf{m}$ = `ekf_slam(steps)`

$$\begin{aligned}
\mathbf{z}_0, \mathbf{R}_0 &= get_measurements \\
\hat{\mathbf{x}}_0, \mathbf{P}_0 &= new_map(\mathbf{z}_0, \mathbf{R}_0)
\end{aligned}$$

for $k = 1$ to `steps` **do**

$$\begin{aligned}
\hat{\mathbf{x}}_{R_k}^{R_{k-1}}, \mathbf{Q}_k &= get_odometry \\
\hat{\mathbf{x}}_{k|k-1}, \mathbf{F}_k, \mathbf{G}_k &= prediction(\hat{\mathbf{x}}_{k-1}, \hat{\mathbf{x}}_{R_k}^{R_{k-1}}) \\
\mathbf{P}_{k|k-1} &= \mathbf{F}_k \mathbf{P}_{k-1} \mathbf{F}_k^T + \mathbf{G}_k \mathbf{Q}_k \mathbf{G}_k^T \qquad (1) \\
\\
\mathbf{z}_k, \mathbf{R}_k &= get_measurements \\
\mathcal{H}_k, \mathbf{H}_k, \mathbf{z}_{\mathcal{H}_k}, \mathbf{R}_{\mathcal{H}_k} &= data_association \\
&\quad (\hat{\mathbf{x}}_{k|k-1}, \mathbf{P}_{k|k-1}, \mathbf{z}_k, \mathbf{R}_k) \\
\\
\mathbf{S}_k &= \mathbf{H}_k \mathbf{P}_{k|k-1} \mathbf{H}_k^T + \mathbf{R}_{\mathcal{H}_k} \qquad (2) \\
\mathbf{K}_k &= \mathbf{P}_{k|k-1} \mathbf{H}_k^T / \mathbf{S}_k \qquad (3) \\
\mathbf{P}_k &= (\mathbf{I} - \mathbf{K}_k \mathbf{H}_k) \mathbf{P}_{k|k-1} \qquad (4) \\
\nu_k &= \mathbf{z}_{\mathcal{H}_k} - \mathbf{h}_{\mathcal{H}_k}(\hat{\mathbf{x}}_{k|k-1}) \qquad (5) \\
\hat{\mathbf{x}}_k &= \hat{\mathbf{x}}_{k|k-1} + \mathbf{K}_k \nu_k \qquad (6) \\
\hat{\mathbf{x}}_k, \mathbf{P}_k &= add_feat(\hat{\mathbf{x}}, \mathbf{P}_k, \mathbf{z}_k, \mathbf{R}_k, \mathcal{H}_k)
\end{aligned}$$

end for

`return` $\mathbf{m} = (\mathbf{x}_k, \mathbf{P}_k)$

II. The EKF SLAM algorithm

The EKF SLAM algorithm (see alg. 1) has been widely used for mapping and localization. Several authors have described the computational complexity of this algorithm [9], [10]. With the purpose of comparing EKF SLAM with the proposed D&C SLAM algorithm, in this section we briefly analyze the computational complexity of EKF SLAM.

A. Computational complexity of EKF SLAM

For simplicity, assume that in the environment being mapped, features are distributed more or less uniformly. If the vehicle is equipped with a sensor of limited range and bearing, the amount of measurements obtained at any location will be more or less constant. Assume that at some step k:

- the map contains n features,
- the sensor provides m measurements,
- r of which correspond to re-observed features and
- $s = m - r$ which correspond to new features.

This corresponds to an exploratory trajectory, where the size of the map is proportional to the number of steps that have been carried out.

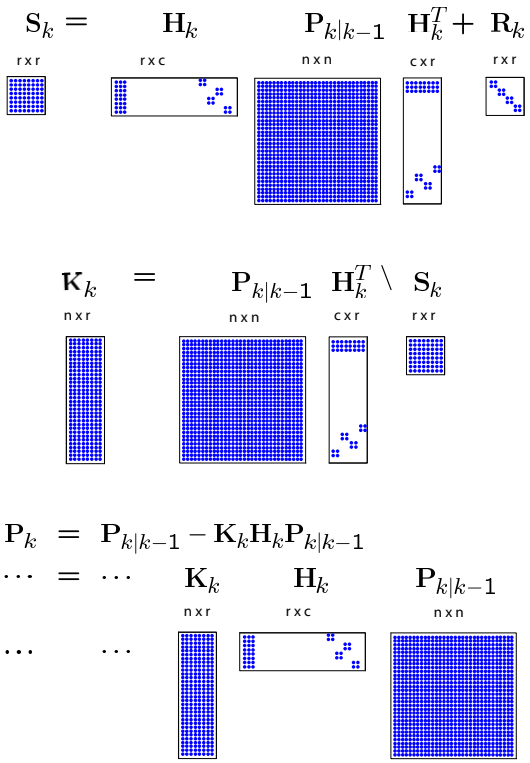

Fig. 1. Computation of the innovation covariance $\mathbf{S}_k$ matrix (top): given that the effective size of the Jacobian matrix $\mathbf{H}_k$ is $r \times c$, the computation requires $O(n)$ operations ($rcn + r^2c$ multiplications and $rcn + r^2c + r^2$ sums). In a similar way, the computations of the Kalman gain $\mathbf{K}_k$ matrix (middle) requires $O(n)$ operations, and the covariance Matrix $\mathbf{P}_k$ (bottom) requires $O(n^2)$ operations.

1) Computational cost per step: The computational complexity of carrying out the move-sense-update cycle of algorithm `ekf_slam` at step k involves the following:

- computing the *predicted map* $\hat{\mathbf{x}}_{k|k-1}, \mathbf{P}_{k|k-1}$, which requires obtaining also the corresponding Jacobians $\mathbf{F}_k, \mathbf{G}_k$
- solving data association (the complexity of data association is analyzed in section V).
- computing the *updated map* $\hat{\mathbf{x}}_k, \mathbf{P}_k$, which requires the computation of the corresponding Jacobian $\mathbf{H}_k$, the Kalman gain matrix $\mathbf{K}_k$, as well as the innovation ν_k, and its covariance $\mathbf{S}_k$.

The fundamental fact regarding computational complexity in standard EKF SLAM is that, given a sensor of limited range and bearing, Jacobians matrices are *sparse* [9], [10], [21]: their computation is $O(1)$. But more importantly, since they take part in the computation of both the predicted and updated map, the computational cost of eqs. (1) to (6) can also be reduced. Consider as an example the innovation covariance matrix $\mathbf{S}_k$ in eq. (2). Without considering sparseness, the computation of this $r \times r$ matrix would require $rn^2 + r^2n$ multiplications and $rn^2 + r^2n + r^2$ sums, that is, $O(n^2)$ operations. But given that matrix $\mathbf{H}_k$ is sparse, with an effective size of $r \times c$, the computation requires $rcn + r^2c$ multiplications and $rcn + r^2c + r^2$ sums, that is, $O(n)$ operations (see fig. 1 top). Similar analysis leads to the conclusion that the cost of computing both the predicted covariance $\mathbf{P}_{k|k-1}$ and the Kalman gain matrix $\mathbf{K}_k$ is $O(n)$, and that the greatest cost in an EKF SLAM update is the computation of the covariance matrix $\mathbf{P}_k$, which is $O(n^2)$. Thus, the computational cost per step of EKF SLAM is quadratic on the size of the map:

$$C_{EKF,k} = O(n^2) \tag{7}$$

2) Total computational cost: Considering the assumptions above, during an exploratory trajectory a constant number of s new features are added to the map at each step. Thus to map an environment of size n, n/s steps are required, and the total cost of EKF SLAM will be:

$$C_{EKF} = \sum_{k=1}^{n/s} O((k\,s)^2) = \sum_{k=1}^{n/s} O(k^2)$$

The square power summation is known to be:

$$\sum_{i=1}^{j} i^2 = \frac{j(j+1)(2j+1)}{6}$$

Thus, the total cost of EKF SLAM is cubic:

$$\begin{aligned} C_{EKF} &= O\left(\frac{(n/s)(n/s+1)(2n/s+1)}{6}\right) \\ &= O\left(\frac{n^3}{s^3}\right) \\ &= O(n^3) \end{aligned} \tag{8}$$

In general, the rate at which a map grows depends on the trajectory that the vehicle follows and on the density of features in the environment. During exploration, the number of features in the current map will grow linearly, but when the vehicle returns retracing its steps, the map size will remain more or less constant. In these cases the cost of updating the map is still $O(n^2)$ with the size of the map, but since n does not increase, the total cost may be less than cubic.

B. Local Mapping Algorithms

Local mapping techniques have been proposed as computationally efficient alternatives to EKF SLAM. Instead of working on a full global map all the time as EKF SLAM does, a sequence of local maps of limited size is produced. When the size of the current local map reaches some limit, the map is closed and stored, and a new local map is created. This allows us to maintain the computational cost constant most of the time, while working on the current local map. Algorithms based on this idea include suboptimal or approximate solutions such as Decoupled Stochastic Mapping (DSM) [32], Constant Time SLAM (CTS) [33], the ATLAS framework [34] and Hierarchical SLAM [35]. These algorithms sacrifice precision in the resulting estimation of the map in order to maintain the computational cost linear in the size of the map at worst.

There are alternative solutions that do not carry out approximations, such as Map Joining SLAM [13] and the Constrained

Algorithm 2 : $\mathbf{m}_{i...k}$ = `join` $(\mathbf{m}_{i...j}, \mathbf{m}_{j...k})$

$$
\begin{aligned}
\hat{\mathbf{x}}^-_{i...k} &= (\hat{\mathbf{x}}_{i...j}, \hat{\mathbf{x}}_{j...k})^T \\
\mathbf{P}^-_{i...k} &= blkdiag(\mathbf{P}_{i...j}, \mathbf{P}_{j...k}) \\
\mathcal{H}, \mathbf{H}_{\mathcal{H}} &= data\_assoc(\hat{\mathbf{x}}^-_{i...k}, \mathbf{P}^-_{i...k}) \\
\\
\mathbf{S}_{\mathcal{H}} &= \mathbf{H}_{\mathcal{H}} \mathbf{P}^-_{i...k} \mathbf{H}^T_{\mathcal{H}} \\
\mathbf{K}_{\mathcal{H}} &= \mathbf{P}^-_{i...k} \mathbf{H}^T_{\mathcal{H}} / \mathbf{S}_{\mathcal{H}} \\
\hat{\mathbf{x}}^+_{i...k} &= \hat{\mathbf{x}}^-_{i...k} - \mathbf{K}_{\mathcal{H}} \mathbf{H}_{\mathcal{H}} \hat{\mathbf{x}}^-_{i...k} \\
\mathbf{P}^+_{i...k} &= (\mathbf{I} - \mathbf{K}_{\mathcal{H}} \mathbf{H}_{\mathcal{H}}) \mathbf{P}^-_{i...k} \\
\\
\hat{\mathbf{x}}_{i...k} &= transform(\hat{\mathbf{x}}^+_{i...k}) \\
\mathbf{P}_{i...k} &= \frac{\partial \hat{\mathbf{x}}_{i...k}}{\partial \hat{\mathbf{x}}^+_{i...k}} \mathbf{P}^+_{i...k} \left(\frac{\partial \hat{\mathbf{x}}_{i...k}}{\partial \hat{\mathbf{x}}^+_{i...k}} \right)^T
\end{aligned}
\tag{9}
$$

`return` $\mathbf{m}_{i...k} = (\mathbf{x}_{i...k}, \mathbf{P}_{i...k})$

Algorithm 3 : $\mathbf{m}$ = `map_joining_slam`

`sequential implementation of map joining.`

```
m = ekf_slam(steps)
{
Main loop
}
repeat
   m_k = ekf_slam(steps)
   m = join(m, m_k)
until end_of_map

return (m)
```

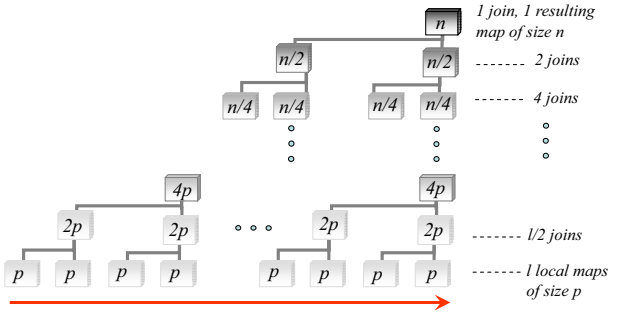

Fig. 2. Hierarchy of maps that are created and joined in D&C SLAM. The lower level is the sequence of l local maps of size p computed with standard EKF SLAM as the arrow suggests. The intermediate levels represent intermediate joins during the process. The top level represents the final map of size n resulting from the join of two local maps os size $n/2$.

Local Submap Filter [36]. Given that we are interested in algorithms that do not sacrifice precision in order to limit computational cost, we concentrate on these two. Map Joining SLAM (and similarly, the Constrained Local Submap Filter) is an EKF-based algorithm in which a sequence of independent local maps of a limited size p is produced using the standard EKF SLAM algorithm. Map independence is guaranteed by construction: once the current local map is closed, a new local map is initialized with zero covariance using the current vehicle pose as the base reference for the new map. These local maps are later fused using a map joining procedure to produce a single final stochastic map. Algorithm 2 details the map joining procedure. For two consecutive local maps $\mathbf{m}_{i...j}$ and $\mathbf{m}_{j...k}$, computed in steps $i \dots j$ and $j \dots k$ respectively, map joining computes the resulting map $\mathbf{m}_{i...k}$ for all steps $i \dots k$ in the following way:

- Both maps are simply stacked together, with the features of each in each local base reference. Thanks to map independence, the cross-covariance between both maps is zero.
- Data association is carried out to find common features between both maps.
- Using a modified version of the EKF update equations the map is optimized by fusing common features.
- All features are transformed to the base reference of the first map.

A more detailed explanation of this procedure and its notation can be found in the Appendix. It is worth noting that Map Joining SLAM never revisits a prior local map. Instead, a new local map is created, which will be joined later with the previous map to obtain a global map which includes all available information.

An analysis similar to that of fig 1 shows that for local maps of limited size, the computational cost of Map Joining SLAM is $O(n^2)$ on the size of the resulting global map, again being the most expensive operation the update of the covariance matrix $\mathbf{P}$. However, map joining takes place only when a given local map has reached its limit size. In all other steps, only a local map is being updated, with a computational cost of $O(1)$.

Algorithm 3 carries out Map Joining SLAM: `ekf_slam` is used to compute a local map $\mathbf{m}_k$ of a given limit size (or for a given limit number of steps). This local map is joined to a global map $\mathbf{m}$ by means of the `join` function in a sequential fashion. The process continues until the environment is completely covered.

III. The Divide and Conquer Algorithm

Instead of joining each new local map to a global map sequentially, as Map Joining SLAM does, the algorithm proposed in this paper, Divide and Conquer SLAM, carries out map joining in a hierarchical fashion, as depicted in fig. 2. The lower nodes of the hierarchy represent the sequence of l local maps of minimal 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 size. D&C is implemented using algorithm 4, which uses a stack to save intermediate maps. Whenever two maps of around the same size are at the top of the stack, they are replaced in the stack by their join. This allows a sequential execution of D&C SLAM.

A. Total computational complexity of D&C SLAM

In D&C SLAM, the process of building a map of size n produces $l = n/p$ maps of size p (not considering overlap), at

cost $O(p^3)$ each (see eq. (7)), which are joined into $l/2$ maps of size $2p$, at cost $O((2p)^2)$ each. These in turn are joined into $l/4$ maps of size $4p$, at cost $O((4p)^2)$ 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 $O(n^2)$. Map joining SLAM and D&C SLAM carry out the same number of map joining operations. The fundamental difference is that in D&C SLAM the size of the maps involved in map joining increases at a slower rate than in Map Joining SLAM. As shown next, this allows the total cost to remain quadratic with n.

The total computational complexity of D&C SLAM is:

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

Note that the sum represents all costs associated to map joining. This corresponds to the sum of a geometric progression of the type:

$$\sum_{i=1}^{k} r^i = \frac{r - r^{k+1}}{1 - r}$$

Thus, in this case:

$$\begin{aligned} C_{DC} &= O\left(p^2 n + p\,n \frac{2^{\log_2 n/p+1} - 2}{2 - 1}\right) \\ &= O\left(p^2 n + p\,n(2\,n/p - 2)\right) \\ &= O\left(p^2 n + 2n^2 - 2pn\right) \\ &= O(n^2) \end{aligned} \tag{10}$$

This means that D&C SLAM performs SLAM with a total cost quadratic with the size of the environment, as compared with the cubic cost of standard EKF SLAM. This corresponds to the normal exploration operation. In the worst case scenario, when the overlap between the maps to be joined is full (i.e., if the robot traverses the whole map for a second time), the cost of map joining will be cubic, the same as EKF SLAM.

B. Computational complexity of D&C SLAM per step

In D&C SLAM, in steps that are a power of 2, when $k = 2^t$, $i = 1 \ldots t$ joins will take place, at a cost $O(2^2), O(4^2) \ldots O(k^2)$ respectively. An analysis similar to that of eq. (10) shows that the total cost of such steps is $O(k^2)$, of the same order as a standard EKF SLAM step. However, in D&C SLAM the map to be generated at step k will not be required for joining until step $2\,k$. This allows us to amortize the cost $O(k^2)$ at this step by dividing it up between steps $k+1$ to $2\,k$ in equal $O(k)$ computations for each step. In this way, amortized D&C SLAM becomes a *linear time* SLAM algorithm.

Algorithm 4 : `dc_slam` sequential implementation using a stack.

```
stack = new()
m_0 = ekf_slam()
stack = push(m_0, stack) {
Main loop: postorder traversing of the map tree.
}
repeat
  m_k = ekf_slam()
  while ¬ empty(stack) and then
  size(m_k) ≥ size(top(stack)) do
    m = top(stack)
    stack = pop(stack)
    m_k = join(m, m_k)
  end while
  stack = push(m_k, stack)
until end_of_map {
Wrap up: join all maps in stack for full map recovery.
}
while ¬ empty(stack) do
  m = top(stack)
  stack = pop(stack)
  m_k = join(m, m_k)
end while
return (m_k)
```

An amortized version of D&C SLAM can be implemented using two concurrent threads: one high priority thread executes `ekf_slam` (alg. 1) to update the current local map, and the other low priority thread executes `dc_slam` (alg. 4). 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 D&C SLAM.

As we will see in the Monte Carlo simulations and the experiments, the amortized cost of D&C SLAM is always lower than that of EKF SLAM. D&C SLAM is an anytime algorithm, if at any moment during the map building process the full map is required for another task, it can be computed in a single $O(n^2)$ step.

IV. Simulated Experiments

We use four simulated scenarios (fig. 3) to illustrate the properties of the algorithms discussed in this paper. In an environment which consists of equally spaced point features, a vehicle equipped with a range and bearing sensor carries out four different trajectories: straight line, a square loop, lawn mowing and outward spiral (first, second, third and fourth column, respectively). The simulated experiments were carried out with known data association for the evaluated algorithms, in order to discard mismatching effects in the resulting performance of the estimators. The first row shows the environment and each of the trajectories. The second and third rows show the execution time per step and the total execution time, respectively. It can be seen that the total cost of D&C SLAM quickly separates from the total cost of EKF

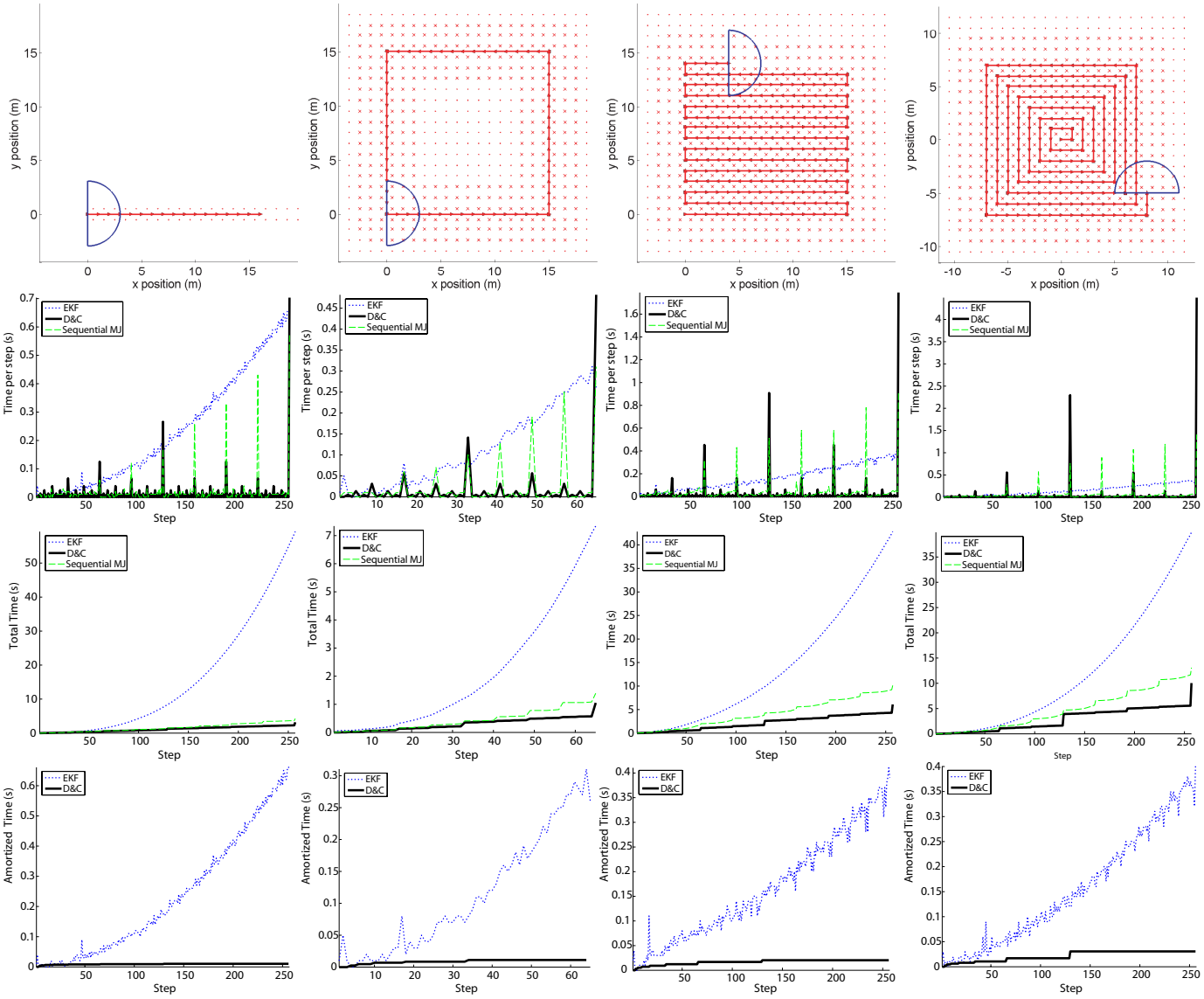

Fig. 3. Four simulated experiments for comparing the EKF, Sequential Map Joining and D&C SLAM algorithms: (first column) straight forward trajectory; (second column) loop closing; (third column) lawn mowing; (fourth column) outward spiral. Ground truth environment and trajectory (top row); execution time per step of EKF vs. Map Joining vs. D&C SLAM (second row); total execution time (third row); execution time per step of EKF SLAM vs. amortized execution time per step of D&C SLAM (bottom row).

SLAM, and also from Map Joining SLAM. The reason is that the computational cost per step of D&C SLAM is lower than that of EKF SLAM most of the time. EKF SLAM works with a map of non-decreasing size, while D&C SLAM works on local maps of small size most of the time. Map Joining SLAM is computationally equivalent to D&C SLAM when working on local maps. In some steps (in the simulation those which are a multiple of 2), the computational cost of D&C is higher. In those steps, one or more map joining operations take place (in those that are a power of 2, 2^t, t map joining operations take place). The accompanying AVI videos `dcslam_xvid_loop.avi`, `dcslam_xvid_lawn.avi` and `dcslam_xvid_spiral.avi` (available at `http://ieeexplore.ieee.org`) show the execution of both EKF SLAM and D&C SLAM for the same sample data. The frames have been time stamped so that the actual running times of the algorithms in our `Matlab` implementation are shown.

Fig. 3 (bottom row) shows the amortized cost per step for the four simulated experiments. We can see that the amortized cost of D&C SLAM is always lower than that of EKF SLAM.

A. Consistency and precision in Divide and Conquer SLAM

Apart from computational complexity, another important aspect of the solution computed by the EKF has gained attention recently: map consistency and precision. When the ground truth solution $\mathbf{x}$ for the state variables is available, a statistical test for filter consistency can be carried out on the estimation $(\hat{\mathbf{x}}, \mathbf{P})$, using the Normalized Estimation Error Squared (NEES), defined as:

$$D^2 = (\mathbf{x} - \hat{\mathbf{x}})^T \, \mathbf{P}^{-1} \, (\mathbf{x} - \hat{\mathbf{x}}) \tag{11}$$

Consistency is checked using a chi-squared test:

$$D^2 \leq \chi^2_{r,1-\alpha} \tag{12}$$

where $r = dim(\mathbf{x})$ and α is the desired significance level (we consider the usual $\alpha = 0.05$). If we define the consistency index of a given estimation $(\hat{\mathbf{x}}, \mathbf{P})$ with respect to its true value $\mathbf{x}$ as:

$$CI = \frac{D^2}{\chi^2_{r,1-\alpha}} \tag{13}$$

when $CI < 1$, the estimation is consistent with ground truth, and when $CI > 1$, the estimation is inconsistent (optimistic) with respect to ground truth. Thus CI measures how precise the computed covariance is with respect to the real error, while precision can be simply computed as the root of the squared difference with ground truth.

We tested consistency of both standard EKF and D&C SLAM algorithms by carrying out 100 Monte Carlo runs on the simulated experiments. Simulation allows us to have ground truth available. Additionally, Monte Carlo runs allow to obtain statistically significant evidence about the consistency of the algorithms being compared.

Figure 4 (top) shows the evolution of the mean consistency index of the vehicle position (left) and orientation (right) during all steps of the straight forward trajectory simulation. We can see that the D&C estimate on vehicle location is always more consistent than the standard EKF estimate; in less than 100 steps EKF falls out of consistency while D&C remains consistent. Figure 4 (bottom) shows the evolution of the root mean square error on the vehicle position and orientation. The 2σ bounds for the *theoretical* uncertainty are computed by running the simulated experiment without measurement and robot noise, so that linearizations take place in the true state values, and thus introduce no errors. The computed uncertainty of both standard EKF and D&C SLAM are also drawn. We can see how the RMS error increases more slowly in the case of D&C SLAM. We can also see the fast rate at which the uncertainty computed by standard EKF SLAM falls below its theoretical value.

Monte carlo runs show that *Divide and Conquer* SLAM is less subject to linearization errors than EKF SLAM. Fig. 5 shows a typical situation: the two algorithms run on exactly the same data of a loop closure (the accompanying video `dcslam_xvid_loop.avi` shows the execution of the two algorithms for the same data). Because of less accumulated error and thus better linearizations, the final result is much more precise for D&C SLAM (data association is known and used in both algorithms).

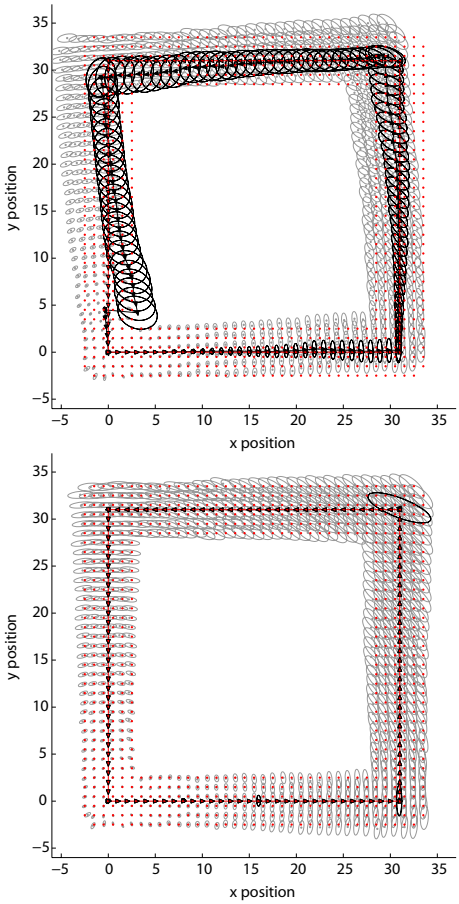

Fig. 5. A typical result when running EKF SLAM (top) and D&C SLAM (bottom) on the same data. The vehicle and map features tend to accumulate more error during exploration with EKF SLAM. Even if data association is known, the final result after closing a loop is less precise. The absolute vehicle location estimates are shown when available from each algorithm. Ground truth environment and trajectory are shown in red.

V. Data Association for Divide and Conquer SLAM

A. Data association for standard EKF SLAM

The data association problem in continuous EKF SLAM consists in producing a hypothesis $\mathcal{H} = [j_1 j_2 \cdots j_i \cdots j_m]$ where correspondences are established between each of the $i = 1 \ldots m$ sensor measurements and one (or none) of the $j = 1 \ldots n$ map features. The space of measurement-feature correspondences can be represented by an *interpretation tree* of m levels [37]. Each node of the tree at level i has $n + 1$ branches, corresponding to the n alternative feature pairings for measurement i, and an extra node (star-branch) to account for the possibility of the measurement being spurious or a new feature. The size of this correspondence space, (i.e. the number of alternative hypotheses) in which data association must be solved is exponential with the number of measurements: $(n + 1)^m$.

Fortunately, the availability of a stochastic model for both the map and the measurements allows us to check each measurement-feature correspondence for *individual compatibility* by predicting the location of the map features relative to the sensor reference, and determine compatibility using a hypothesis test on the innovation and covariance of each possible pairing.

The discrepancy between the observation i and the predicted observation of map feature j is measured using the innovation term of eq. (5) and covariance (2). The measurement can be considered corresponding to the feature if the Mahalanobis

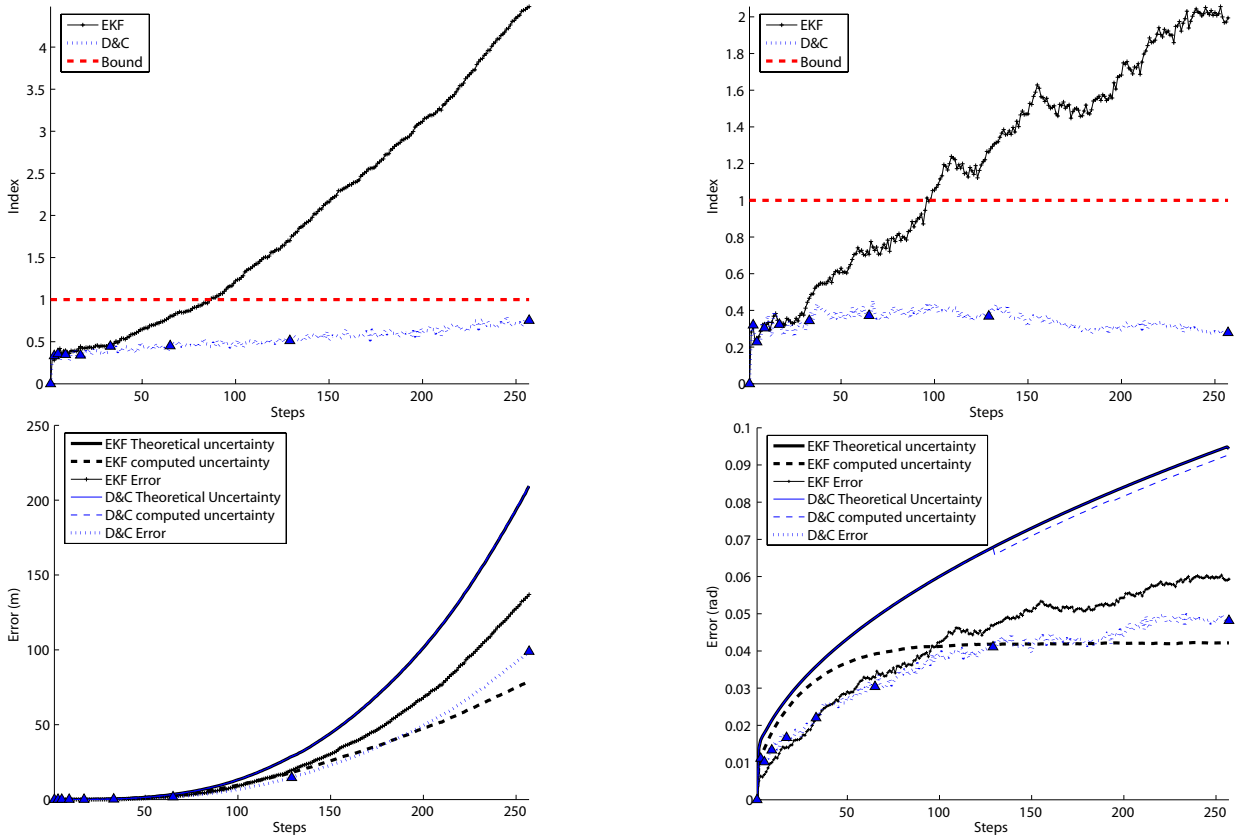

Fig. 4. Mean consistency index CI of eq. 13 (top) and Root Mean Squared Error (bottom) for the robot x-y position (left) and orientation (right) for standard EKF and D&C SLAM. The root mean square error is always smaller for D&C SLAM; also the computation of the variance of the error is more precise, and thus the estimation always remains consistent. The EKF and D&C theoretical uncertainties coincide over all steps. Also, D&C computed uncertainty superimposes the latter two.

distance $D^2_{k,ij}$ satisfies [38]:

$$D^2_{k,ij} = \nu^T_{k,ij}\mathbf{S}^{-1}_{k,ij}\nu_{k,ij} < \chi^2_{d,1-\alpha} \tag{14}$$

where $d = \dim(\mathbf{h}_{k,j})$ and $1-\alpha$ is the desired confidence level, usually 95%.

In standard EKF SLAM, and for a sensor of limited range and bearing, the number of measurements m is constant and thus individual compatibility is $O(nm) = O(n)$, linear on the size of the map. This cost can be easily reduced to $O(m)$, a constant, by a simple tessellation or grid of the map computed during map building, which allows us to determine candidates for a measurement in constant time simply by checking the grid element and nearby grid elements in which its predicted location falls. In 2D problems, the cost of computing and updating the tessellation would be constant per step (a constant amount of new features are included in the map per step), while the space required would be proportional to the total area covered by the map and the resolution of the tessellation.

In cases where clutter or vehicle error are high, there may be more than one possible correspondence for each measurement. More elaborate algorithms are required to disambiguate in these cases. Nevertheless, the overlap between the measurements and the map is the size of the sensor range plus the vehicle uncertainty, and thus more or less constant. In local mapping, after individual compatibility is sorted out, we use the **Joint Compatibility Branch and Bound (JCBB)** algorithm [39] (see alg. 5) to disambiguate between the possible associations for the m measurements. It has been shown that algorithms such as JCBB, based on a probabilistic model (feature estimates and covariances), can greatly increase the robustness of data association, even when other measurement properties are available. For instance, in the case of monocular SLAM, the combined use of texture-based matching and stochastic compatibility tests has proved critical to reject outliers, especially from repetitive patterns and dynamic objects [40]. **JCBB** performs branch and bound search on the interpretation tree looking for *jointly compatible* correspondences, but only in the overlap determined by individual compatibility. Given that this is a region of the map of constant size, each measurement will have a more or less constant number of feature candidates, say c, and thus the solution space is constant: $(c+1)^m$. In this way, **JCBB** will execute in constant time.

B. Data association for Divide and Conquer SLAM

Data association in D&C SLAM is a very particular problem because it involves finding correspondences between two local maps of similar size whenever joining is to take place. For instance, before obtaining a final map of size n, the data association problem has to be solved between two maps of size $n/2$ and so computing *individual compatibility* would be $O(n^2)$ instead of $O(n)$. Fortunately, as in the case of

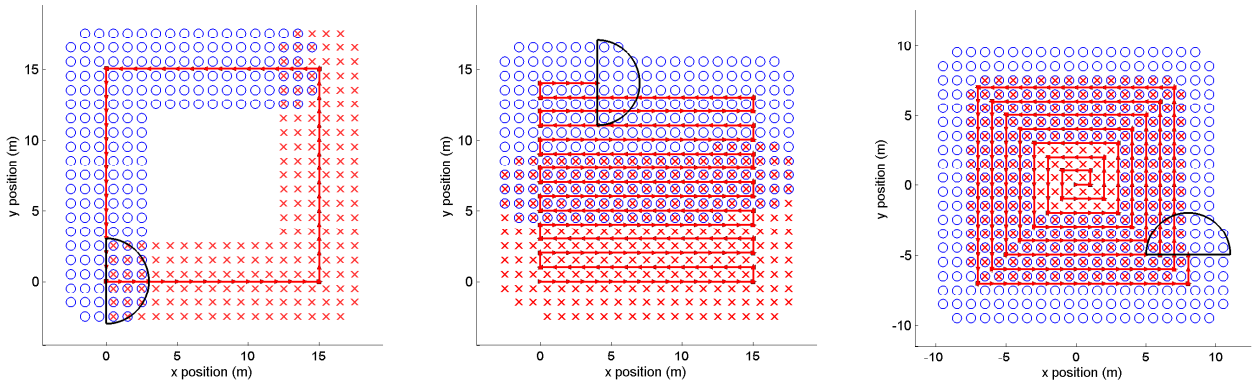

Fig. 7. Size of the overlap between two final local maps, i.e. common features in both maps: crosses correspond to the first map and circles to the second map. Square loop trajectory (left), lawn mowers trajectory (middle), and outward spiral (right).

Algorithm 5 Joint Compatibility Branch and Bound

```
Continuous_JCBB (E_{1...m}, F_{1...n}):
Best = [ ]
JCBB ([ ], 1)
return Best

JCBB (H, i): {find pairings for observation E_i}
if i > m then {leaf node?}
   if pairings(H) > pairings(Best) then
      Best ← H
   end if
else
   for j = 1 to n do
      if individual_compatibility(i, j) and then
      joint_compatibility(H, i, j) then
         JCBB([H j], i + 1) {pairing (E_i, F_j) accepted}
      end if
   end for
   if pairings(H)+m−i > pairings(Best) then {can do better?}
      JCBB([H 0], i + 1) {star node, E_i not paired}
   end if
end if
```

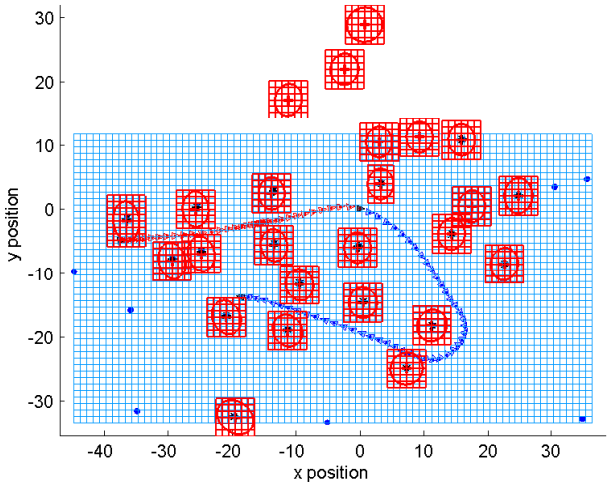

Fig. 6. Tessellation to compute individual compatibility between two local maps of similar size. The second (blue) map is tessellated using a grid. Red ellipses represent the uncertainties of the predicted features of the first local map with respect to the base reference of the second. The ellipses are approximated by windows, and in this way possible candidates (asterisks) for each red feature can be found in constant time. The robot trajectory is also shown for each local map with the corresponding color.

individual compatibility for standard EKF SLAM, finding potential matches for one feature in another map can be done in constant time using a simple tessellation or grid in the map where the search is done. Consider the example in fig. 6. The red trajectory and features correspond to the first local map built, and the blue trajectory and features correspond to the second local map. Individual compatibility may be done in a way similar to standard EKF SLAM: we predict the location of features in the first (red) map relative to the base reference of the second (blue) map, and check for possible correspondences with blue features. If the blue map is tessellated, we can find potential matches for a red feature in constant time, and for the whole red map in linear time. The cost of computing and updating the tessellation is constant per step, while the space required is proportional to the total area covered by each map. Alternative solutions, such as 2D priority kd-trees [41], can be used to make the storage space required be $O(n \log n)$ on the number of map features, instead of dependent on the covered area as the tessellation is. There is however a higher cost involved in finding a potential match per feature, $O(\log n)$, and an update cost of $O(\log n)$ per feature to be included.

A second issue of importance is the size of the region of overlap between two local maps. While in standard EKF SLAM this region is constant, and thus data association algorithms like **JCBB** will execute in constant time, in D&C SLAM the size of the overlap is not always constant. It basically depends on the environment and type of trajectory. Consider the simulated examples of fig. 7 where two $n/2$ maps are shown. In the first case, the square loop, the region of overlap between two maps will be of constant size, basically dependent on the sensor range. In the case of the lawn mowers trajectory, the overlap will be proportional to the length of the trajectory before the vehicle turns back, basically the square root of n. In the third case, the outward spiral, the region of overlap between the inner map and the encircling map is proportional to the square root of n as well. In some cases, like traversing a loop for a second time, the size of the overlap is the entire second local map.

In order to limit the computational cost of data association between local maps in D&C SLAM, we use a *randomized joint*

Algorithm 6 : RJC

$\mathbf{P}_{fail} = 0.01$, $\mathbf{P}_{good} = 0.8$, $b = 4$
$i = 1$, $Best$ = [], $\mathcal{H}$ = []
while $(i \leq t)$ **do**
  $\mathbf{m}_2^*$ = random_select($\mathbf{m}_2$, b)
  $\mathcal{H}$ = JCBB*($\mathcal{H}$,1,$\mathbf{m}_1$, $\mathbf{m}_2^*$)
  if pairings($\mathcal{H}$) > pairings($Best$) **then**
    $Best$ = $\mathcal{H}$
  end if
  $\mathbf{P}_{good}$ = $max(\mathbf{P}_{good}, pairings(Best)/m)$
  t = $\log \mathbf{P}_{fail} / \log(1 - \mathbf{P}_{good}^b)$
  i = $i + 1$
end while

{
JCBB: testing the joint compatibility for b pairings.*
}
`procedure JCBB*(`$\mathcal{H}$`, i,` $\mathbf{m}_1$`,` $\mathbf{m}_2^*$`)`
if pairings($\mathcal{H}$)==b **then**
  $\mathcal{H}$ = NN($\mathcal{H}$, $i + 1$, $\mathbf{m}_1$, $\mathbf{m}_2^*$)
else
  for $j = 1$ to $length(\mathbf{m}_1)$ **do**
    if `individually_compatible(i, j)` $\wedge$ `jointly_compatible(`$[\mathcal{H}\ j]$`)` **then**
      `JCBB*(`$[\mathcal{H}\ j], i + 1, \mathbf{m}_1, \mathbf{m}_2^*$`)`
    end if
  end for
end if

compatibility algorithm. Our **RJC** approach (see algorithm 6) is a variant of the linear **RS** algorithm [42] used for global localization. Consider two consecutive maps $\mathbf{m}_1$ and $\mathbf{m}_2$, of size n_1 and n_2 respectively, to be joined. First, the overlap between the two maps is identified using individual compatibility. Second, instead of performing branch and bound interpretation tree search in the whole overlap as in **JCBB**, we randomly select b features in the overlapped area of the second map and use **JCBB***. This algorithm is a version of **JCBB** where all b features are expected to be found in the second map (no star branch). This produces a hypothesis $\mathcal{H}$ of b jointly compatible features in the first map. Associations for the remaining features in the overlap are obtained using the simple nearest neighbor rule given hypothesis $\mathcal{H}$, that is, finding pairings that are compatible with the first b features. In the spirit of adaptive **RANSAC** [43], we repeat this process t times, so that the probability of missing a correct association is limited to P_{fail}.

The RJC algorithm successfully detects the overlap between two local maps in either continuous data association or loop closing. The only requirement is that the stochastic maps remain consistent, a condition which is enforced by the D&C algorithm.

Since **JCBB*** is executed using a fixed number of features, its cost remains constant. Finding the nearest neighbor for each remaining feature among the ones that are individually compatible with it, a constant number, will be constant. The cost of each try is thus $O(n)$. The number of tries depends on b, the number of features randomly selected, on the probability that a selected feature in the overlap can be actually found in the first map P_{good}, and on the acceptable probability of failure in this probabilistic algorithm, P_{fail}. It does not depend on the size of either map. In this way, we can maintain data association in D&C SLAM linear with the size of the joined map.

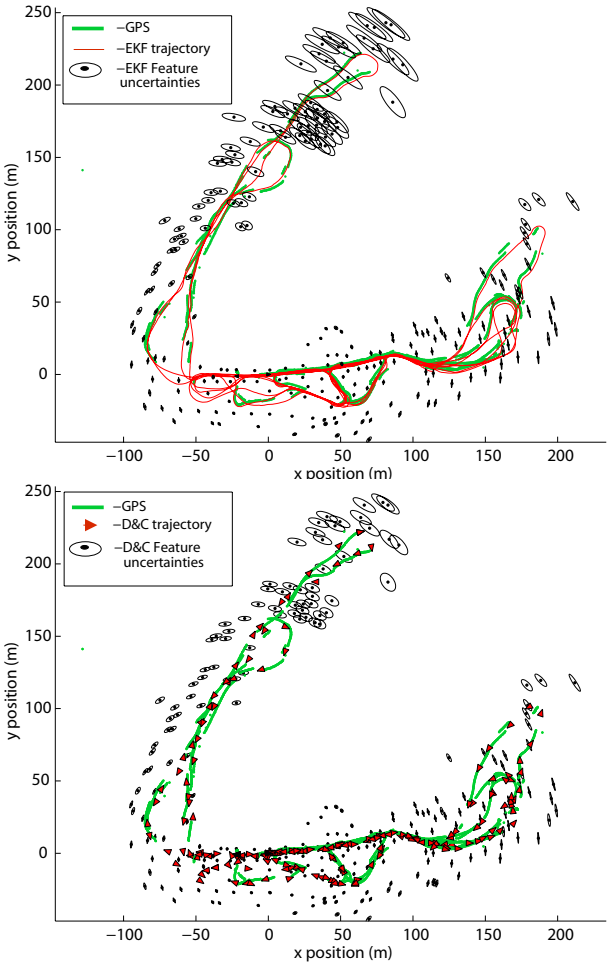

Fig. 8. Map for Victoria Park dataset according to the EKF SLAM (top) and to D&C SLAM (bottom). The results are essentially equivalent; there are some minor differences due to missed associations in the case of EKF. The estimated position along the whole trajectory is shown as a red line for EKF SLAM, and the vehicle locations are drawn as red triangles when available in D&C SLAM. Green points are GPS readings in both cases, and are not used in either case.

VI. Experiments

We have used the well known Victoria Park data set to validate the algorithms D&C SLAM and **RJC**. This experiment is particulary adequate for testing SLAM due to its large scale and the significant level of spurious measurements. The experiment also provides critical loops in absence of reliable features.

For **RJC**, we chose $b = 4$ as the number of map features to be randomly selected as seed for hypothesis generation. Two features are sufficient in theory to fix the relative location between the maps, but we have found 4 to adequately disambiguate. The probability that a selected feature in the overlap is not spurious, $\mathbf{P}_{good}$ is set to 0.8, and the probability of not finding a good solution when one exists, $\mathbf{P}_{fail}$ is set to 0.01. These parameters make the data association algorithm carry out 9 random tries.

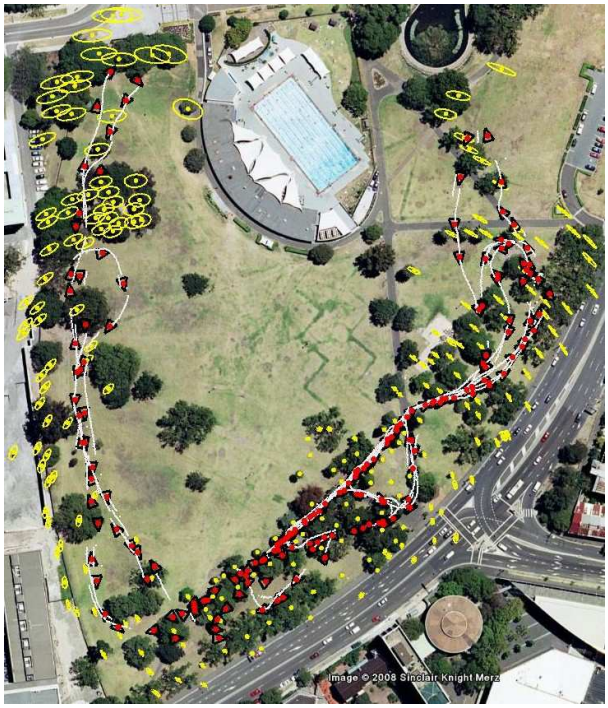

Fig. 9. The results were projected on Google Earth in order to compare the precision obtained.

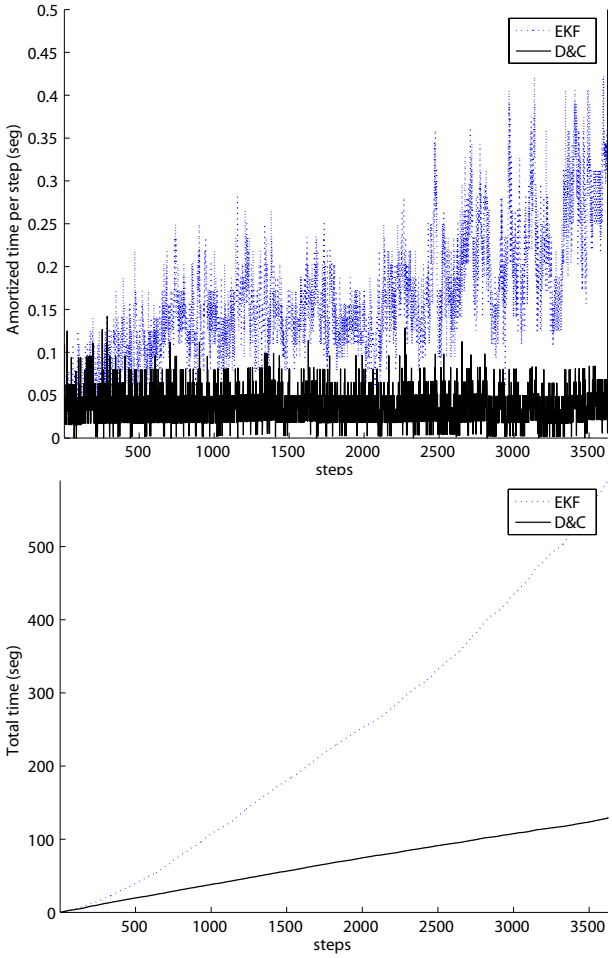

Fig. 10. Time per step of EKF SLAM vs. amortized time per step of D&C SLAM (top); accumulated time of EKF SLAM vs. D&C SLAM (bottom).

Figure 8 shows the resulting maps from standard EKF SLAM vs. D&C SLAM. Each algorithm solves data association on its own. This allows seeing when the estimator falls out of consistency precisely because data association starts to fail; there are some minor differences due to missed associations in the case of EKF. Figure 10, top, shows the amortized cost of D&C SLAM. We can see that in this experiment an EKF step can take 0.5 seconds, while the amortized D&C SLAM step will take around 0.05 seconds. The main source of the noise visible in the EKF timing values is the variable number of observations gathered when the vehicle traverses the environment.

In real experiments like Victoria Park it is not generally possible to predict at which step the size of the current local map will reach its limit size. This depends on the trajectory that the vehicle follows and on the density of features in the environment. Some features can be initialized in the map and later removed if they are not re-observed. In the Victoria Park data set, the vehicle sometimes carries out exploratory trajectories and sometimes it revisits previously mapped regions of the park. When two local maps are joined, the size of the resulting map will depend on the overlap. For these reasons, the total map size does not increase linearly with the number of steps. This makes the total cost of standard EKF SLAM to not be cubic with the number of steps (fig. 10, bottom). For the same reasons the total cost of D&C SLAM seems to grow linearly, instead of quadratically, with the step number. In any case, the benefits of using D&C SLAM can be clearly seen. In this experiment, the total cost of D&C SLAM is one fifth of the total cost of standard EKF, (130.24s on a 2.8 GHz Pentium IV, compared to 590.48s for EKF SLAM). This result is also comparatively better than the reported by iSAM algorithm [23] (464s on a 2 GHz Pentium M) and comparable to FastSLAM 2.0 [29] (140s on a 1 GHz Pentium IV). Plotting the results of D&C SLAM on ©Google Earth (Fig. 9) reveals a superior performance when comparing with the precision obtained by the mentioned algorithms [23], [29]. The accompanying video `dcslam_xvid_victoria.avi` shows the comparative running times of both standard EKF SLAM and D&C SLAM.

VII. Conclusions

In this paper we have shown that EKF SLAM can be carried out in time *linear* with map size. We describe an EKF SLAM variant: *Divide and Conquer* SLAM, an algorithm that can be easily implemented. In contrast with many current efficient SLAM algorithms, all information required for data association is available when needed with no further processing. D&C SLAM computes the EKF SLAM solution, both the state *and* its covariance, with no approximations, and with the additional advantage of providing always a more precise and consistent vehicle and map estimate. We also provide **RJC**, a data association algorithm that also executes in linear time per step.

We hope to have shown that D&C SLAM is the algorithm to use in all applications in which the Extended Kalman Filter solution is to be used. We also believe that the D&C map

hierarchical splitting strategy can also be incorporated in other algorithms based on local submaps and similar strategies. This idea is part of our future work.

APPENDIX: MAP JOINING 2.0

This appendix describes the map joining process used in D&C SLAM, an improved version with respect to the original map joining 1.0 in [13]. The general idea is the following: in a sequential move-sense-update cycle, a local map is initialized at some moment i using the current vehicle location R_i as base reference, and thus the initial vehicle location in the map is $\mathbf{x}_{R_iR_i} = 0$ an also the initial vehicle uncertainty $\mathbf{P}_{R_i} = 0$. Standard EKF SLAM is carried out in a this move-sense-update fashion, until the map reaches a certain size of n features $F_1 \dots F_n$ at step j. In this moment the state vector $\hat{\mathbf{x}}_{i\dots j}$ will be:

$$\hat{\mathbf{x}}_{i\dots j} = \begin{bmatrix} \hat{\mathbf{x}}_{R_iR_j} \\ \hat{\mathbf{x}}_{R_iF_1} \\ \vdots \\ \hat{\mathbf{x}}_{R_iF_n} \end{bmatrix}$$

with corresponding covariance matrix $\mathbf{P}_{i\dots j}$. This map is then closed, and a new local map $\mathbf{m}_{j\dots k} = (\hat{\mathbf{x}}_{j\dots k}, \mathbf{P}_{j\dots k})$ is initialized in the same way (for simplicity, assume the sensor measurements at step j are used to update the first map, and the vehicle motion from R_j to R_{j+1} is carried out in the second map). This results in having the last vehicle location in the first map, R_j, be the base reference of the second map, which allows maps to be joined into a full map in a three step process of (1) joining; (2) update; and (3) transformation, as it is explained next.

A. The Map Joining step

Consider two sequential local maps $\mathbf{m}_{i\dots j} = (\hat{\mathbf{x}}_{i\dots j}, \mathbf{P}_{i\dots j})$, $\mathbf{m}_{j\dots k} = (\hat{\mathbf{x}}_{j\dots k}, \mathbf{P}_{j\dots k})$, with n features $F_1 \dots F_n$ and m features $G_1 \dots G_m$ each:

$$\hat{\mathbf{x}}_{i\dots j} = \begin{bmatrix} \mathbf{x}_{R_iR_j} \\ \mathbf{x}_{R_iF_1} \\ \vdots \\ \mathbf{x}_{R_iF_n} \end{bmatrix}; \hat{\mathbf{x}}_{j\dots k} = \begin{bmatrix} \mathbf{x}_{R_jR_k} \\ \mathbf{x}_{R_jG_1} \\ \vdots \\ \mathbf{x}_{R_jG_m} \end{bmatrix} \tag{15}$$

The *joining step* allows us to obtain a stochastic map $\mathbf{m}^-_{i\dots k} = \left(\hat{\mathbf{x}}^-_{i\dots k}, \mathbf{P}^-_{i\dots k}\right)$ in the following simple way:

$$\hat{\mathbf{x}}^-_{i\dots k} = \begin{bmatrix} \hat{\mathbf{x}}_{i\dots j} \\ \hat{\mathbf{x}}_{j\dots k} \end{bmatrix}; \hat{\mathbf{P}}^-_{i\dots k} = \begin{bmatrix} \mathbf{P}_{i..j} & \mathbf{0} \\ \mathbf{0} & \mathbf{P}_{j..k} \end{bmatrix} \tag{16}$$

Note that the elements in the second map are kept in their own reference R_j instead of being referenced to reference frame R_i as in map joining 1.0. This has the effect of *delaying* the linearization process of converting all features to base reference R_i until the update step has taken place, and thus an improved estimation is used for this linearization. This is the fundamental difference between map joining 1.0 and map joining 2.0.

B. The update step

Data association is carried out to determine correspondences between features coming from the first and second map. This allows us to refine the vehicle and environment feature locations by the EKF update step on the state vector. Let $\mathcal{H}$ be a hypothesis that pairs r features $F_{f_1} \dots F_{f_r}$ coming from local map $\mathbf{m}_{i\dots j}$ with features $G_{g_1} \dots G_{g_r}$ coming from map $\mathbf{m}_{j\dots k}$. A modified ideal measurement equation for r re-observed features expresses this coincidence:

$$\mathbf{h}_{\mathcal{H}}(\hat{\mathbf{x}}^-_{i\dots k}) = \begin{bmatrix} \mathbf{h}_{f_1,g_1} \\ \vdots \\ \mathbf{h}_{f_r,g_r} \end{bmatrix} = \mathbf{0}$$

where for each pairing:

$$\mathbf{h}_{f_r,g_r} = \mathbf{x}_{R_iF_{f_r}} - \mathbf{x}_{R_iR_j} \oplus \mathbf{x}_{R_jG_{g_r}}.$$

Linearization yields:

$$\mathbf{h}_{\mathcal{H}}(\hat{\mathbf{x}}^-_{i\dots k}) \simeq \mathbf{h}_{\mathcal{H}}(\hat{\mathbf{x}}^-_{i\dots k}) + \mathbf{H}_{\mathcal{H}}(\mathbf{x}^-_{i\dots k} - \hat{\mathbf{x}}^-_{i\dots k})$$

$$\mathbf{H}_{\mathcal{H}} = \frac{\partial \mathbf{h}_{\mathcal{H}}}{\partial \mathbf{x}^-_{i\dots k}}\Big|_{(\hat{\mathbf{x}}^-_{i\dots k})} = \begin{bmatrix} \frac{\partial \mathbf{h}_{f_1g_1}}{\partial \mathbf{x}_{R_iR_j}} & \mathbf{0} & \cdots & \mathbf{I} & \mathbf{0} & \frac{\partial \mathbf{h}_{f_1g_1}}{\partial \mathbf{x}_{R_jG_{g_1}}} & \cdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \frac{\partial \mathbf{h}_{f_rg_r}}{\partial \mathbf{x}_{R_iR_j}} & \mathbf{0} & \mathbf{I} & \cdots & \mathbf{0} & \cdots & \frac{\partial \mathbf{h}_{f_rg_r}}{\partial \mathbf{x}_{R_jG_{g_r}}} \end{bmatrix} \tag{17}$$

The update step allows us to obtain a new estimate $\mathbf{m}^+_{i\dots k} = \left(\hat{\mathbf{x}}^+_{i\dots k}, \mathbf{P}^+_{i\dots k}\right)$ by applying modified EKF update equations:

$$\begin{aligned} \hat{\mathbf{x}}^+_{i\dots k} &= \hat{\mathbf{x}}^-_{i\dots k} - \mathbf{K}\mathbf{h}_{\mathcal{H}}(\hat{\mathbf{x}}^-_{i\dots k}) \\ \mathbf{P}^+_{i\dots k} &= (\mathbf{I} - \mathbf{K}\mathbf{H}_{\mathcal{H}})\mathbf{P}^-_{i\dots k} \\ \mathbf{K} &= \mathbf{P}^-_{i\dots k}\mathbf{H}^T_{\mathcal{H}}\left(\mathbf{H}_{\mathcal{H}}\mathbf{P}^-_{i\dots k}\mathbf{H}^T_{\mathcal{H}}\right)^{-1} \end{aligned}$$

C. The transformation step

A final step is carried out to transform all the elements of $\hat{\mathbf{x}}^+_{i\dots k}$ to the same base reference R_i and obtain the final joined map $\mathbf{m}_{i\dots k} = (\hat{\mathbf{x}}_{i\dots k}, \mathbf{P}_{i\dots k})$:

$$\begin{aligned} \hat{\mathbf{x}}_{i\dots k} &= \begin{bmatrix} \hat{\mathbf{x}}_{R_iR_k} \\ \hat{\mathbf{x}}_{R_iF_1} \\ \vdots \\ \hat{\mathbf{x}}_{R_iF_n} \\ \hat{\mathbf{x}}_{R_iG_1} \\ \vdots \\ \hat{\mathbf{x}}_{R_iG_m} \end{bmatrix} = \begin{bmatrix} \hat{\mathbf{x}}^+_{R_iR_j} \oplus \hat{\mathbf{x}}^+_{R_jR_k} \\ \hat{\mathbf{x}}^+_{R_iF_1} \\ \vdots \\ \hat{\mathbf{x}}^+_{R_iF_n} \\ \hat{\mathbf{x}}^+_{R_iR_j} \oplus \hat{\mathbf{x}}^+_{R_jG_1} \\ \vdots \\ \hat{\mathbf{x}}^+_{R_iR_j} \oplus \hat{\mathbf{x}}^+_{R_jG_m} \end{bmatrix} \\ \mathbf{P}_{i\dots k} &= \frac{\partial \hat{\mathbf{x}}_{i\dots k}}{\partial \hat{\mathbf{x}}^+_{i\dots k}} \mathbf{P}^+_{i\dots k} \left(\frac{\partial \hat{\mathbf{x}}_{i\dots k}}{\partial \hat{\mathbf{x}}^+_{i\dots k}}\right)^T \\ \frac{\partial \hat{\mathbf{x}}_{i\dots k}}{\partial \hat{\mathbf{x}}^+_{i\dots k}} &= \begin{bmatrix} \frac{\partial \mathbf{x}_{R_iR_k}}{\partial \mathbf{x}_{R_iR_j}} & \mathbf{0} & \frac{\partial \mathbf{x}_{R_iR_k}}{\partial \mathbf{x}_{R_jR_k}} & \mathbf{0} \\ \mathbf{0} & \mathbf{I} & \mathbf{0} & \mathbf{0} \\ \frac{\partial \mathbf{x}_{R_iE}}{\partial \mathbf{x}_{R_iR_j}} & \mathbf{0} & \mathbf{0} & \frac{\partial \mathbf{x}_{R_iE}}{\partial \mathbf{x}_{R_jE}} \end{bmatrix} \end{aligned} \tag{18}$$

Note again that this linearization is carried out once the map has been refined in the previous update step, thus using a better estimate.

REFERENCES

[1] L. M. Paz, P. Jensfelt, J. D. Tardós, and J. Neira, "EKF SLAM updates in *O(n)* with Divide and Conquer SLAM," in *Proc. IEEE Int. Conf. Robotics and Automation*, Rome, Italy, April 2007.

[2] L. M. Paz, J.Guivant, J. D. Tardós, and J. Neira, "Data Association in *O(n)* for Divide and Conquer SLAM," in *Proc. Robotics: Science and Systems*, Atlanta, GA, USA, June 2007.

[3] H. Durrant-Whyte and T. Bailey, "Simultaneous Localization and Mapping: Part I," *IEEE Robotics & Automation Magazine*, vol. 13, no. 2, pp. 99–110, 2006.

[4] 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.

[5] R. Chatila and J. Laumond, "Position referencing and consistent world modeling for mobile robots," in *Proc. IEEE Int. Conf. Robotics and Automation*, vol. 2, 1985.

[6] R. C. Smith and P. Cheeseman, "On the Representation and Estimation of Spatial Uncertainty," *Int. J. Robotics Research*, vol. 5, no. 4, pp. 56–68, 1986.

[7] R. Smith, M. Self, and P. Cheeseman, "A Stochastic Map for Uncertain Spatial Relationships," in *Robotics Research, The Fourth Int. Symposium*, O. Faugeras and G. Giralt, Eds. The MIT Press, 1988, pp. 467–474.

[8] J. Leonard and H. Durrant-Whyte, "Simultaneous Map Building and Localization for an Autonomous Mobile Robot," in *Proc. IEEE/RJS Int. Conference on Intelligent Robots and Systems*, Osaka, Japan, 1991, pp. 1442–1447.

[9] J. A. Castellanos and J. D. Tardós, *Mobile Robot Localization and Map Building: A Multisensor Fusion Approach*. Boston, Mass.: Kluwer Academic Publishers, 1999.

[10] J. E. Guivant and E. M. Nebot, "Optimization of the Simultaneous Localization and Map-Building Algorithm for Real-Time Implementation," *IEEE Trans. Robotics and Automation*, vol. 17, no. 3, pp. 242–257, 2001.

[11] J. Knight, A. Davison, and I. Reid, "Towards Constant Time SLAM using Postponement," in *Proc. IEEE/RJS Int. Conference on Intelligent Robots and Systems*, Maui, Hawaii, 2001, pp. 406–412.

[12] S. J. Julier, "A Sparse Weight Kalman Filter Approach to Simultaneous Localisation and Map Building," in *Proc. IEEE/RJS Int. Conference on Intelligent Robots and Systems*, vol. 1, Hawaii, October 2001, pp. 1251–1256.

[13] 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.

[14] L. M. Paz and J. Neira, "Optimal Local map size for EKF-based SLAM," in *Proc. IEEE/RJS Int. Conference on Intelligent Robots and Systems*, Beijing, China., October 2006.

[15] S. Thrun, Y. Liu, D. Koller, A. Y. Ng, Z. Ghahramani, and H. Durrant-Whyte, "Simultaneous Localization and Mapping with Sparse Extended Information Filters," *Int. J. Robotics Research*, vol. 23, no. 7-8, pp. 693–716, 2004.

[16] 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.

[17] M. Walter, R. Eustice, and J. Leonard, "A Provably Consistent Method for Imposing Sparsity in Feature-based SLAM Information Filters," in *Proc. Int. Symp. Robotics Research*, 2004.

[18] M. A. Paskin, "Thin Junction Tree Filters for Simultaneous Localization and Mapping," in *Proc. Int. Joint Conf. Artificial Intelligence*, San Francisco, CA., 2003, pp. 1157–1164.

[19] 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.

[20] 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.

[21] F. Dellaert and M. Kaess, "Square Root SAM: Simultaneous Localization and Mapping via Square Root Information Smoothing," *Int. J. Robotics Research*, vol. 25, no. 12, December 2006.

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

[23] M. Kaess, A. Ranganathan, and F. Dellaert, "iSAM: Fast Incremental Smoothing and Mapping with Efficient Data Association," in *IEEE Intl. Conf. on Robotics and Automation, ICRA*, Rome, Italy, Apr 2007.

[24] S. J. Julier and J. K. Uhlmann, "A Counter Example to the Theory of Simultaneous Localization and Map Building," in *Proc. IEEE Int. Conf. Robotics and Automation*, Seoul, Korea, 2001, pp. 4238–4243.

[25] J. A. Castellanos, J. Neira, and J. D. Tardós, "Limits to the Consistency of EKF-based SLAM," in *5th IFAC Symposium on Intelligent Autonomous Vehicles*, Lisbon, Portugal, 2004.

[26] T. Bailey, J. Nieto, J. Guivant, M. Stevens, and E. Nebot, "Consistency of the EKF-SLAM Algorithm," in *Proc. IEEE/RJS Int. Conference on Intelligent Robots and Systems*, 2006.

[27] S. Julier and J. Uhlmann, "A new extension of the Kalman Filter to nonlinear systems," in *International Symposium on Aerospace/Defense Sensing, Simulate and Controls*, Orlando, FL, 1997.

[28] R. Martinez-Cantin and J. A. Castellanos, "Unscented SLAM for large-scale outdoor environments," in *Proc. IEEE/RJS Int. Conference on Intelligent Robots and Systems*, Edmonton, Alberta, Canada, 2005, pp. pp. 328–333.

[29] 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 *Proc. Int. Joint Conf. Artificial Intelligence*, 2003.

[30] 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.

[31] M. W. M. G. Dissanayake, P. Newman, S. Clark, H. F. Durrant-Whyte, and M. Csorba, "A solution to the simultaneous localization and map building (SLAM) problem," *IEEE Trans. Robotics and Automation*, vol. 17, no. 3, pp. 229–241, 2001.

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

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

[34] 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.

[35] 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.

[36] S. B. Williams, "Efficient Solutions to Autonomous Mapping and Navigation Problems," Ph.D. dissertation, Australian Centre for Field Robotics, University of Sydney, September 2001, available at http://www.acfr.usyd.edu.au/.

[37] W. E. L. Grimson, *Object Recognition by Computer: The Role of Geometric Constraints*. Cambridge, Mass.: The MIT Press, 1990.

[38] Y. Bar-Shalom and T. E. Fortmann, *Tracking and Data Association*. Academic Press In., 1988.

[39] J. Neira and J. D. Tardós, "Data Association in Stochastic Mapping Using the Joint Compatibility Test," *IEEE Trans. Robotics and Automation*, vol. 17, no. 6, pp. 890–897, 2001.

[40] L. Clemente, A. J. Davison, I. D. Reid, J. Neira, and J. D. Tardós, "Mapping Large Loops with a Single Hand-Held Camera," in *Proc. Robotics: Science and Systems*, Atlanta, GA, USA, June 2007.

[41] J. Uhlmann, "Introduction to the Algorithmics of Data Association in Multiple-Target Tracking," in *Handbook of Multisensor Data Fusion*. Boca Raton, FL: CRC Press, 2001.

[42] J. Neira, J. D. Tardós, and J. A. Castellanos, "Linear time vehicle relocation in SLAM," in *Proc. IEEE Int. Conf. Robotics and Automation*, Taipei, Taiwan, September 2003, pp. 427–433.

[43] R. Hartley and A. Zisserman, *Multiple View Geometry in Computer Vision*. Cambridge, U. K.: Cambridge University Press, 2000.

Lina M. Paz was born in Cali, Colombia, in 1980. She received the M.S. degree in Electronic Engineering from the Universidad del Valle, Cali, Colombia, in 2003. Currently, she is a Ph.D. candidate at the Department of Computer Science and Systems Engineering, University of Zaragoza, Zaragoza, Spain, since 2004, in computer science.

She carries out research in mobile robotics, computer vision for environment modeling and SLAM.

Juan D. Tardós was born in Huesca, Spain, in 1961. He earned the M.S. and Ph.D. degrees in electrical engineering from the University of Zaragoza, Spain, in 1985 and 1991, respectively.

He is full professor with the Departamento de Informática e Ingeniería de Sistemas, University of Zaragoza, where he is in charge of courses in robotics, computer vision, and artificial intelligence. His current research interests include SLAM, perception and mobile robotics.

José Neira was born in Bogotá, Colombia, in 1963. He received the M.S. degree from the Universidad de los Andes, Bogot, Colombia, in 1986, and the Ph.D. degree from the University of Zaragoza, Spain, in 1993, both in Computer Science.

He is an Associate Professor with the Department of Computer Science and Systems Engineering, University of Zaragoza, where he teaches courses in compiler theory, computer vision, and mobile robotics. His current research interests include autonomous robots, data association, and environment modeling.