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 where 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 an exact 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 a by-product of reduced computation, the resulting vehicle and map estimates are more precise than standard EKF SLAM: the error with respect to the true value is 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 [1], [2]. 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 egomotion and augmented reality. SLAM has been the subject of much attention since the seminal work in the late 80s [3], [4], [5], [6].

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 $x$ representing the sensor and environment feature locations, together with the covariance matrix $P$ 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 [7], [8]. The exception is the update of the covariance matrix 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 [9], the Compressed EKF filter [8], and Map Joining SLAM [10] 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 [11]. Recently, researchers have pointed out the approximate sparseness of the Information matrix $Y$, the inverse of the full covariance matrix $P$. This suggests using the Extended Information Filter, the dual of the Extended Kalman Filter, for SLAM updates. The Sparse Extended Information Filter (SEIF) algorithm [12] 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 [13]. This overconfidence is overcome by the Exactly Sparse Extended Information Filter (ESEIF) [14] with a strategy that produces an exactly sparse Information matrix with no introduction of inaccuracies through sparsification.

The Thin Junction Tree Filter algorithm [15] 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 [16] is a closely related technique, which also uses a weak link breakage policy. Recently, the Exactly Sparse Delayed State Filter [17] and Square Root SAM [18] provided the insight that the full SLAM problem, the complete vehicle trajectory plus the map, is sparse in information form (although ever increasing). Sparse linear algebra techniques allow to compute the state, without the covariance, in time linear with the whole trajectory and map size. The Tectonic SAM algorithm [19] 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.

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. Linearizations 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 [20], [21], [22]. Among other things, this shuts down 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 crippled. All algorithms for EKF SLAM based on efficiently computing an approximation of the EKF solution [15], [16] will inevitably suffer from this consistency problem. The Unscented Kalman Filter [23] avoids linearization 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 [24]. 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 $O(n^1)$; 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 an exact 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. Empirical results show that, as a by-product of reduced computations, and without losing precision because of approximations, D&C SLAM has better consistency properties than standard EKF SLAM.

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 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 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.

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 [7], [8]. With the purpose of comparing EKF SLAM with the proposed D&C SLAM algorithm, in this section we briefly analyze its computational complexity.

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,

The computational cost per step involves the following operations:

- $r$ of which correspond to re-observed features,
- $s = m - r$ 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.

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 computation of the prediction $x_{k|k-1}^n, P_{k|k-1}$, which requires obtaining also the computation of the corresponding Jacobians $F_k, G_k$, and the update of the state $x_{k}, P_{k}$, which requires the computation of the corresponding Jacobian $H_k$, the Kalman gain matrix $K_k$, as well as the innovation $v_k$ and its covariance $S_k$ (the complexity of data association is analyzed in section V).

The fundamental fact regarding computational complexity in standard EKF SLAM is that, given a sensor of limited range and bearing, Jacobians matrices are sparse [7], [8], [18]. 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 $S_k$ in eq. (2). Without considering sparseness, the computation of this $r \times r$ matrix would require $r n^2 + r^2 n$ multiplications and $r n^2 + r^2 n + r^2$ sums, that is, $O(n^2)$ operations. But given that matrix $H_k$ is sparse, with an effective size of $r \times c$, the computation requires $r cn + r^2 c$ multiplications and $rcn + r^2 c + r^2$ sums, that is, $O(n)$ operations (see fig. 1 top). Similar analysis leads to the conclusion that the cost of computing

<table>
<thead>
<tr>
<th>Algorithm 1: $m = ekf_slam$(steps)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$z_0, R_0 = get_measurements$</td>
</tr>
<tr>
<td>$x_0, P_0 = new_map(z_0, R_0)$</td>
</tr>
<tr>
<td>for $k = 1$ to steps do</td>
</tr>
<tr>
<td>$x_{k</td>
</tr>
<tr>
<td>$x_{k</td>
</tr>
<tr>
<td>$P_{k</td>
</tr>
<tr>
<td>$z_k, R_k = get_measurements$</td>
</tr>
<tr>
<td>$H_k, H_k^c, z_{H_k}, R_{H_k} = data_association$</td>
</tr>
<tr>
<td>$(\bar{x}_{k</td>
</tr>
<tr>
<td>$S_k = H_k P_{k</td>
</tr>
<tr>
<td>$K_k = P_{k</td>
</tr>
<tr>
<td>$P_k = (I - K_k H_k) P_{k</td>
</tr>
<tr>
<td>$v_k = z_{H_k} - H_k \bar{x}_{k</td>
</tr>
<tr>
<td>$\bar{x}<em>k = x</em>{k</td>
</tr>
<tr>
<td>$x_{k}, P_{k} = add_feat(\bar{x}, P_k, z_k, R_k, H_k)$</td>
</tr>
</tbody>
</table>

for end

return $m = (x_k, P_k)$

- $r$ of which correspond to re-observed features,
- $s = m - r$ 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.
both the predicted covariance $P_{k|k-1}$ and the Kalman gain matrix $K_k$ is $O(n)$, and that the greatest cost in an EKF SLAM update is the computation of the covariance matrix $P_k$, which is $O(n^2)$. Thus, the computational cost per step of EKF SLAM is quadratic on the size of the map:

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

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:

$$\sum_{i=1}^{n} k^2 = (1/6)n(n+1)(2n+1)$$

Thus, the total cost if EKF SLAM is cubic:

$$C_{EKF} = O\left(\frac{(n/s)(n/s+1)(2n/s+1)}{6}\right) = O\left(\frac{n^3}{s^3}\right) = O(n^3) \quad (8)$$

### 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 size, the map is closed and stored, and a new local map is created. This allows 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), [25], Constant Time SLAM (CTS) [26], the ATLAS framework [27], and Hierarchical SLAM [28]. 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 [10] and the Constrained Local Submap Filter [29]. Given that we are interested in algorithms that do not sacrifice precision in order to limit computational cost, we concentrate on the latter. Map Joining SLAM (and similarly, the Constrained Local Submap Filter) is an EKF-based algorithm in which a sequence of local maps of a limited size $p$ is produced using the standard EKF SLAM algorithm. These maps are then joined using a map joining procedure to produce a single final stochastic map. Algorithm 2 details the map joining procedure. For two consecutive local maps $m_{i...j}$ and $m_{j...k}$, computed in steps $i...j$ and $j...k$ respectively, map joining computes the resulting map $m_{i...k}$ for all steps $i...k$ in the following way:

1) Both maps are simply stacked together, with the features of each in each local base reference.
2) Data association is carried out to find common features between both maps.
3) Using a modified version of the EKF update equations the map is optimized by fusing common features.
4) All features are transformed to the same base reference (of the first map).

A more detailed explanation of this procedure and its notation can be found in [30]. 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 $P$. 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 $m_k$ of a given limit size (or for a given limit number of steps). This local map is joined to a global map $m$ 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 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 1 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:

\[
C_{DC} = O \left( p^3 + \sum_{i=1}^{\log_2 t} \frac{1}{2i} (2^i p)^2 \right)
\]

\[
= O \left( p^3 \frac{n}{p} + \sum_{i=1}^{\log_2 \frac{n}{p}} \frac{n}{2i} (2^i p)^2 \right)
\]

\[
= O \left( p^2 n + \sum_{i=1}^{\log_2 \frac{n}{p}} \frac{n}{2i} (2^i)^2 \right)
\]

\[
= O \left( p^2 n + p \sum_{i=1}^{\log_2 \frac{n}{p}} 2^i \right)
\]

(note that the sum represents all costs associated to map joining, which is \( O(n^2) \)). The sum in the equation above is a geometric progression of the type:

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

Thus, in this case:

\[
C_{DC} = O \left( p^2 n + p \sum_{i=1}^{\log_2 \frac{n}{p}} 2^i \right)
\]

\[
= O \left( p^2 n + p \left( 2^{\log_2 n - 1} - 2 \right) \right)
\]

\[
= O \left( p^2 n + 2n^2 - 2pn \right)
\]

\[
= O(n^2)
\] (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.

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 \( O(2^i), O(2^2) \ldots O((k)^2) \) 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 of 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 \( 2k \). This allows us to amortize the cost \( O(k^2) \) at this step by dividing it up between steps \( k+1 \) to \( 2k \) in equal \( O(k) \) computations for each step. In this way, amortized D&C SLAM becomes a linear time SLAM algorithm.

An amortized version of D&C SLAM can be implemented in a simple way using two threads: one high priority thread executes \texttt{ekf\_slam} (alg. 1) to update the current local map, and the other low priority thread executes \texttt{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 forward, a square loop, lawn mowing and outward spiral (first, second, third and fourth column, respectively). The first row shows the environment and each of the trajectories. The second and third rows show the execution time per step and total execution time, respectively, for EKF SLAM (blue), Map Joining SLAM (red) and D&C SLAM (black) for the four trajectories. It can be seen that the total cost of D&C SLAM quickly separates from the total cost of EKF 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 videos \texttt{dcslam\_xvid\_loop.avi}, \texttt{dcslam\_xvid\_lawn.avi} and \texttt{dcslam\_xvid\_spiral.avi} 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 \texttt{Matlab} implementation are shown.

Fig. 3 (last 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 \( x \) for the state variables is available, a statistical test for filter consistency can be carried out on the estimation \( \hat{x} \) and \( P \), using the Normalized Estimation Error Squared (NEES), defined as:

\[
D^2 = (x - \hat{x})^T P^{-1} (x - \hat{x})
\] (11)

Consistency is checked using a chi-squared test:

\[
D^2 \leq \chi^2_{1,\alpha}
\] (12)

where \( r = \text{dim}(x) \) and \( \alpha \) is the desired significance level (usually 0.05). If we define the consistency index of a given estimation \( (\hat{x}, P) \) with respect to its true value \( x \) as:
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 100 Monte Carlo runs on the simulated experiments. Simulation allows to have ground truth available. Additionally, Monte Carlo runs allow to gather empirical evidence about the consistency properties of the algorithms being compared, while a single experiment, although valuable for the use of real data, allows to carry out only one run of the algorithms.

Figure 4 (top) shows the evolution of the mean consistency index of the vehicle x-y position (left) and angular error (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; EKF falls out of consistency while D&C remains consistent. Figure 4 (bottom) shows the evolution of the root mean square error of the vehicle x-y position and orientation errors. The 2\( \sigma \) bounds for the theoretical uncertainty are computed by running the simulated experiment with no measurement or robot noise, so that linearizations take place in the true 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 usually 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.

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 \( \mathcal{H} = [j_1j_2\ldots j_i\ldots j_m] \) 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 [31]. 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 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. 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 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 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 JCBB [32] to disambiguate between the possible associations for the \( m \) measurements. 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.
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.

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 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.

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 (features in the first map are crosses, features in the second are circles). In the top 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 (middle), 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 (bottom), the outward spiral, the region of overlap between the inner map and the encircling map is proportional to the final map size \( n \). 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 compatibility algorithm. Our RJC approach (see algorithm 5) is a variant of the linear RS algorithm [33] used for global localization. Consider two consecutive maps \( m_1 \) and \( m_2 \), 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 $H$ 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 $H$, that is, finding pairings that are compatible with the first $b$ features. In the spirit of adaptive RANSAC [34], we repeat this process $t$ times, so that the probability of missing a correct association is limited to $P_{fail}$.

Algorithm 5: RJC

\[ P_{fail} = 0.01, P_{good} = 0.8, b = 4 \]

\[ i = 1, \text{Best} = \emptyset, H = \emptyset \]

while \((i \leq t)\) do

\[ m^*_2 = \text{random}_\text{select}(m_2, b) \]

\[ H = \text{JCBB}^*(H, 1, m_1, m^*_2) \]

if \(\text{pairings}(H) > \text{pairings(Best)}\) then

\[ \text{Best} = H \]

end if

\[ P_{good} = \max(P_{good}, \text{pairings(Best)}/m) \]

\[ t = \log P_{fail}/\log(1 - P_{good}) \]

\[ i = i + 1 \]

end while

{\n\text\{ JCBB*: testing the joint compatibility for } b \text{ pairings. \}
\}

procedure JCBB* \((H, i, m_1, m^*_2)\)

if \(\text{pairings}(H) = b\) then

\[ H = \text{NN}(H, i + 1, m_1, m^*_2) \]

else

for \(j = 1\) to length\((m_1)\) do

\[ \text{individually}_\text{compatible}(i, j) \land \]

\[ \text{jointly}_\text{compatible}([H j]) \]

\[ \text{JCBB}^*([H j], i + 1, m_1, m^*_2) \]

end if

end for

end if

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.

VI. EXPERIMENTS

We have used the well known Victoria Park data set to validate the algorithms D&C SLAM and RJC. This experiment is particularly adequate for testing SLAM due 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, $P_{good}$ is set to 0.8, and the probability of not finding a good solution when one exists, $P_{fail}$ is set to 0.01. These parameters make the data association algorithm carry out 9 random tries.

Figure 8 shows the resulting maps from standard EKF SLAM algorithm (top); according to the D &C SLAM algorithm (bottom). The results are essentially equivalent; some missed associations may result in minor differences. 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.
SLAM.

In experiments like Victoria Park, and in general, it is not 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. In Victoria Park, the vehicle sometimes carries out exploratory trajectories, it some times returns to a previous location, and some times its revisits previously mapped regions of the park. Furthermore, when two local maps are joined, the size of the resulting map will depend on the overlap. No overlap means a total map size equal to the sum of the sizes of the involved local maps, and on the other extreme full overlap means no increase in the size of the full map.

For these reasons, the total map size not to increase linearly with the number of steps. This makes the total cost of standard EKF SLAM not be cubic with the number of steps (fig. 9, bottom). The total cost of D&C SLAM seems to grow linearly, instead of quadratically, with the step number, for the same reasons. In any case, the benefits of using D&C SLAM can be clearly seen. In the worst case scenario, when overlap between the maps to be joined is full (i.e., when traversing the same loop several times), the cost of map joining will be cubic. Thus, the cost per step of amortized D&C SLAM will be quadratic, the same as EKK 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, a simple algorithm to implement. 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 exact EKF SLAM solution, 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 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.

ACKNOWLEDGMENT

This research has been funded in part by the Dirección General de Investigación of Spain under project DPI2006-13578.

REFERENCES


![Fig. 9. Time per step of EKF SLAM vs. amortized time per step of D&C SLAM (left); accumulated time of EKF SLAM vs. D&C SLAM (right).](image_url)


