CI-Graph: An efficient approach for Large Scale SLAM

Pedro Piniés, Lina M. Paz, Juan D. Tardós

***Abstract*— When solving the Simultaneous Localization and Mapping (SLAM) problem, submapping and graphical methods have shown to be valuable approaches that provide significant advantages over the standard EKF solution: they are faster and can produce more consistent estimates when using *local coordinates*. In this paper we present *CI-Graph*, a submapping method for SLAM that uses a graph structure to efficiently solve complex trajectories reducing the computational cost. Unlike other submapping SLAM approaches, we are able to transmit and share information through maps in the graph in a consistent manner by using *conditionally independent* submaps. In addition, the current submap always summarizes, without further computations, all information available making *CI-Graph* be an intrinsically "up to date" algorithm. Moreover, the technique is also efficient in memory requirements since it does not need to recover the full covariance matrix. To evaluate *CI-Graph* performance, the method has been tested using a synthetic Manhattan world and Victoria Park data set.**

I. INTRODUCTION

Essential tasks in mobile robotics strongly rely, not only on a precise estimation of the robot location, but also, on an accurate map estimate of the surrounding environment. Simultaneous Localization and Mapping algorithms (SLAM) confront both problems in a single estimation process. The first consistent solution proposed was based on the Extended Kalman Filter (EKF) [1], [2]. However, an standard implementation of the algorithm suffers from memory and time complexities of $O(n^2)$ per step, where n is the total number of features stored in the map. To reduce the computational cost, new algorithms take profit of the fact that SLAM is a sparse problem, i.e., from a given robot position only a limited number of features is visible. If all features were always visible then no algorithm could overcome the computational complexity of the EKF solution since the linearized system to be solved would be completely full.

Submapping strategies have become interesting approaches since they work in small regions of the environment reducing the computational cost of EKF and improving consistency. Under the assumption of white noise and if no information is shared between maps, submaps are statistically independent. This allows submaps to be consistently joined using Map Joining algorithm [3] or equivalent Constrained Local Submap Filter (CLSF) [4] with joining cost $O(n^2)$. More recently, Divide and Conquer SLAM [5] has shown to provide a more efficient strategy to join local maps with amortized linear cost in exploration, outperforming past sequential methods. Despite its high scalability, the main limitations of these techniques are their inability to share information between maps and a memory cost of $O(n^2)$.

There are submapping techniques that work on approximations trading off precision for complexity properties [6]. Some of these techniques combine submaps with a graph structure that represents adjacency relations between maps. In ATLAS [7] and CTS [8] for example, nodes of the graph correspond to submaps and links between nodes represent relative locations between adjacent submaps. However, in order to achieve high efficiency, they do not impose loop constraints to update the graph estimation. Hierarchical SLAM [9] outperforms these approaches by introducing an optimization step along the cycles of the graph. Nevertheless, it still remains as an approximate algorithm since optimized information is not transmitted to submaps.

In contrast to EKF-based approaches, there is a family of algorithms that considers the full SLAM problem in a Smoothing and Mapping (SAM) sense. Graph SLAM and Square Root SLAM [10], [11], report that the intrinsic structure of the problem can be modeled as a *sparse graph* (obtained from the sparse information matrix) when the state vector is augmented with the total trajectory. The main problem of these techniques is that they continuously grow with the number of robot poses.

Based on EKF, Graphical SLAM [12], builds a compressed graph of all robot and features poses as nodes. However, special cases as loop closings need particular manipulations. Treemap [13], is based on creating a *balanced binary tree* structure of the map. The technique uses a detailed graph granularity to construct the tree, where leaf nodes represent each map entity (current robot and feature locations) resulting in a very complete but complex graph algorithm.

In this work we are interested in methods that do not use any approximations. We consider the SLAM problem as a Gaussian Graph model that evolves over time. We propose *CI-Graph* SLAM based on [14], a submapping method that performs EKF updates efficiently reducing its quadratic cost. Unlike other non-approximated submapping approaches [3], [4], *CI-Graph* SLAM builds a spanning tree of *conditionally independent* submaps, that allows us to transmit information between submaps in a consistent manner. Compared to batch algorithms [10], [11], *CI-Graph* does not require to augment the state vector with the full trajectory. Instead, only robot poses corresponding to map transitions are considered. In *CI-Graph*, the nodes do not represent each element of the map but the CI-submaps. This results in a graph with high level abstraction of the map that allows a simpler implementation.

Section II is devoted to review conditionally independent submaps and describes their advantages. CI-Graph approach

Pedro Piniés, Lina M. Paz, Juan D. Tardós are with the Departamento de Informática e Ingenieria de Sistemas, Centro Politécnico Superior, Universidad de Zaragoza, Zaragoza, Spain {ppinies, linapaz, tardos}@unizar.es

is presented in section III. We evaluate the method using a synthetic Manhattan world and Victoria Park data set. Their results are described in section IV. In section V we discuss concerns related to the construction of the map spanning tree. Finally, we summarize the main advantages of our method in section VI and draw future lines of work.

II. CONDITIONALLY INDEPENDENT (CI) SUBMAPS

In submapping algorithms, instead of dealing with a single total map of an environment, the whole map is divided into groups of state vector entities (features and/or vehicle poses) that are processed separately. We call *absolute submap*, to a map that is expressed in a global coordinate frame while a *local submap* is a submap whose elements are represented with respect to a local reference frame. Most recent submapping techniques are based on building local maps of limited size that are statistically *independent* [3], [4], [15], [5]. This requirement imposes important constraints to the submaps structure. Valuable information present in a submap cannot be used to improve other submap estimates since, otherwise, the independence property could not be preserved. In addition, same environment features observed in different maps have independent estimations in each map.

Instead of using independent submaps, our *CI-Graph* SLAM approach is based on building *conditionally independent* CI-submaps. The previous technique was presented in [14] and allows CI-submaps to share submap components and information in a consistent manner. Using absolute submaps, the final map obtained is the same as with the classical EKF-SLAM algorithm. If local submaps are used better consistency properties than the EKF can be achieved. While techniques based on independent submaps preclude the use of inertial sensors or sensors that give absolute measurements such as GPS and compass, our method can easily use these devices by propagating the information through CI-submaps without approximations. At the same time, CI-submaps inherit the computational efficiency of submapping techniques that, taking into account a subgroup of the map elements, can work with covariance/information submatrices of limited size.

The technique presented in [14] is restricted to sequences of maps forming simple topologies such as single loops. Even though it has been successfully tested in large environments, the generalization to more complex topologies in which the CI property between maps still hold is not trivial. The purpose of this paper is to develop a new algorithm that extents the properties of the CI-submaps to more complicated trajectories. In order to facilitate the explanation, we will work for the rest of the paper with *absolute submaps*, although *local submaps* can be used as well with the technique.

A. Brief CI-submaps review

Figure 1 shows a Bayesian Network that represents the stochastic dependencies between a pair of CI-submaps, $\mathbf{x}_1$ and $\mathbf{x}_2$, that have been built sequentially.

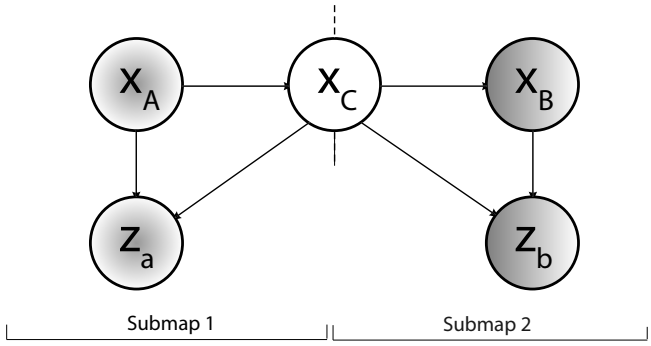

Fig. 1. Bayesian Network that describes the probabilistic dependencies between CI-submaps.

We define the state vectors of the submaps by:

$$\mathbf{x}_1 = \begin{bmatrix} \mathbf{x}_A \\ \mathbf{x}_C \end{bmatrix} \qquad \mathbf{x}_2 = \begin{bmatrix} \mathbf{x}_B \\ \mathbf{x}_C \end{bmatrix} \tag{1}$$

where $\mathbf{x}_A$ represents state components that only belong to the first map, $\mathbf{x}_B$ is for elements exclusively included in the second submap and $\mathbf{x}_C$ represents features and vehicle states that are shared in common between both. Notice that common elements $\mathbf{x}_C$ are replicated in $\mathbf{x}_1$ and $\mathbf{x}_2$. This division of the stochastic state variables can be done in SLAM because it is a sparse problem (features are locally observable).

In figure 1, we can also observe that the only connection between the set of nodes $(\mathbf{x}_A, \mathbf{z}_a)$ and $(\mathbf{x}_B, \mathbf{z}_b)$ is through node $\mathbf{x}_C$ that, according to Bayesian Network theory [16], means that both subgraphs are *d-separated* given $\mathbf{x}_C$. This in turn implies that the components of the submaps are *Conditionally Independent (CI)* when $\mathbf{x}_C$ is known:

$$\begin{aligned} p(\mathbf{x}_A|\mathbf{x}_B, \mathbf{x}_C, \mathbf{z}_a, \mathbf{z}_b) &= p(\mathbf{x}_A|\mathbf{x}_C, \mathbf{z}_a) \\ p(\mathbf{x}_B|\mathbf{x}_A, \mathbf{x}_C, \mathbf{z}_a, \mathbf{z}_b) &= p(\mathbf{x}_B|\mathbf{x}_C, \mathbf{z}_b) \end{aligned} \tag{2}$$

It is precisely by means of the common elements between maps and the CI Property that we can easily transmit information between map pairs *at any time* with no approximations. Suppose now that during the sequential creation of submaps $\mathbf{x}_1$ and $\mathbf{x}_2$, the first submap $\mathbf{x}_1$ was built using observations $\mathbf{z}_a$ whereas for $\mathbf{x}_2$, in addition to $\mathbf{z}_a$ measurements, new observations $\mathbf{z}_b$ were taken into account. Assuming Gaussian distributions for the map states we have:

$$p(\mathbf{x}_A, \mathbf{x}_C|\mathbf{z}_a) \sim \mathcal{N}\left(\begin{bmatrix} \hat{\mathbf{x}}_{A_a} \\ \hat{\mathbf{x}}_{C_a} \end{bmatrix}, \begin{bmatrix} P_{A_a} & P_{AC_a} \\ P_{CA_a} & P_{C_a} \end{bmatrix} \right) \tag{3}$$

$$p(\mathbf{x}_C, \mathbf{x}_B|\mathbf{z}_a, \mathbf{z}_b) \sim \mathcal{N}\left(\begin{bmatrix} \hat{\mathbf{x}}_{C_{ab}} \\ \hat{\mathbf{x}}_{B_{ab}} \end{bmatrix}, \begin{bmatrix} P_{C_{ab}} & P_{CB_{ab}} \\ P_{BC_{ab}} & P_{B_{ab}} \end{bmatrix} \right) \tag{4}$$

where the lowercase subindexes in the estimates $\mathbf{x}_{B_{ab}}$ and $\mathbf{x}_{C_{ab}}$ reveal that both sets of observations $\mathbf{z}_a$ and $\mathbf{z}_b$ have been used in the estimation process.

Notice that map $\mathbf{x}_1$ is 'out of date' with respect to map $\mathbf{x}_2$ since the influence of new observations $\mathbf{z}_b$ is not included in the estimate. In order to update submap $\mathbf{x}_1$ information from $\mathbf{x}_2$ has to be transmitted. The operation of transmitting

information is called *back-propagation* where 'back' means propagation from the updated to the out of date map. To update submap $\mathbf{x}_1$ we only need to recalculate the state vector and covariance matrix of those elements related to $\mathbf{x}_A$. The *back-propagation* equations are given by:

$$\begin{aligned} K &= P_{AC_a} P_{C_a}^{-1} \\ &= P_{AC_{ab}} P_{C_{ab}}^{-1} \quad (5) \\ P_{AC_{ab}} &= K P_{C_{ab}} \quad (6) \\ P_{A_{ab}} &= P_{A_a} + K(P_{CA_{ab}} - P_{CA_a}) \quad (7) \\ \hat{\mathbf{x}}_{A_{ab}} &= \hat{\mathbf{x}}_{A_a} + K(\hat{\mathbf{x}}_{C_{ab}} - \hat{\mathbf{x}}_{C_a}) \quad (8) \end{aligned}$$

Observe that to update the first submap we only need the mean and covariance of the common elements $\hat{\mathbf{x}}_{C_{ab}}$ and $P_{C_{ab}}$ from the second submap. We can also calculate the correlation between non-common elements of both maps $\mathbf{x}_A$ and $\mathbf{x}_B$ by:

$$P_{AB_{ab}} = K P_{CB_{ab}} \quad (9)$$

III. CI-GRAPH ALGORITHM DESCRIPTION

In order to work with complex topologies, the algorithm proposed is based on building an undirected graph of the CI-submaps. An undirected graph is defined as a pair $\mathcal{G} = (\mathcal{N}, \mathcal{E}_\mathcal{G})$ where $\mathcal{N}$ are the *nodes* of $\mathcal{G}$ and $\mathcal{E}_\mathcal{G}$ are its undirected *edges* [17]. In our graph, $\mathcal{N}$ is the set of CI-submaps $\mathbf{m}_i$ with $i = 1 \dots N$. An edge connecting two nodes is created either because the robot makes a transition between the corresponding submaps or because being the robot in a submap, it observes a feature that belongs to the other submap.

In addition, the algorithm builds a spanning tree $\mathcal{T}(\mathcal{N}, \mathcal{E}_\mathcal{T})$ of the graph $\mathcal{G}$, where $\mathcal{E}_\mathcal{T} \subset \mathcal{E}_\mathcal{G}$. A spanning tree $\mathcal{T}$ of a connected undirected graph $\mathcal{G}$ is defined as a subgraph of $\mathcal{G}$ which is a tree (it contains no cycles) and connects all the nodes. Our algorithm ensures that, by construction, any pair of submaps ($\mathbf{m}_i$, $\mathbf{m}_j$) that are adjacent in $\mathcal{T}$ have a conditionally independent structure as shown in figure 1, sharing some vehicle and feature states. Each edge in $\mathcal{E}_\mathcal{T}$ will be labeled with the corresponding shared states. Given any pair of submaps, $\mathbf{m}_i$ and $\mathbf{m}_j$, there is a unique path in $\mathcal{T}$ connecting them. This path allows us to transmit information from map to map without loosing the conditional independence property between submaps. In all graph figures of the paper, spanning tree edges $\mathcal{E}_\mathcal{T}$ will be depicted using a continuous line while the remaining edges of $\mathcal{G}$, i.e. $\mathcal{E}_\mathcal{G} \backslash \mathcal{E}_\mathcal{T}$, will be traced with a dashed line.

Two operational levels can be distinguished in the algorithm. Local operations that are only applied to the current submap $\mathbf{m}_i$, and graph operations that are performed through the graph involving at least two submaps. Most of the time, the operations carried out when the robot moves inside a CI-submap are local operations corresponding to standard EKF-SLAM equations. Graph operations are more sporadic and can be considered as the interface between CI-submaps. In the following subsections, the graph operations are explained in detail as presented in Algorithm 1.

Algorithm 1 : `CI-Graph SLAM`

$\mathbf{z}_0, \mathbf{R}_0 = getObservations$
$\mathbf{m}_0 = initMap(\mathbf{z}_0, \mathbf{R}_0)$
$[\mathcal{G}, \mathcal{T}] = initGraph(\mathbf{m}_0)$ $\{\mathcal{G}(\mathcal{N} = \mathbf{m}_0, \mathcal{E}_\mathcal{G} = \emptyset)\}$
$i = 0$ $\{i$ for current submap$\}$
for $k = 1$ to steps **do**
  $\mathbf{u}_{k-1}, \mathbf{Q}_{k-1} = getOdometry$
  $\mathbf{m}_i = ekfPrediction(\mathbf{m}_i, \mathbf{u}_{k-1}, \mathbf{Q}_{k-1})$
  $\mathbf{z}_k, \mathbf{R}_k = getObservations$
  $\mathcal{DA}_k = dataAssociation(\mathbf{m}_i, \mathbf{z}_k, \mathbf{R}_k)$
  if revisiting $\mathbf{m}_j$ **then**
    {*Subsection III-C*}
    for $\langle \mathbf{m}_k, \mathbf{m}_l \rangle$ in $path(\mathbf{m}_i, \mathbf{m}_j)$ **do**
      $backPropagation(\mathbf{m}_k, \mathbf{m}_l)$
      $copyRobot(\mathbf{m}_k, \mathbf{m}_l)$
    end for
    $addEdge(\langle \mathbf{m}_i, \mathbf{m}_j \rangle, \mathcal{E}_\mathcal{G} \backslash \mathcal{E}_\mathcal{T})$
    $i = j$ {*Map change*}
  else if newMap $\mathbf{m}_j$ **then**
    {*Subsection III-A*}
    $addNode(\mathbf{m}_j, \mathcal{N})$
    $addEdge(\langle \mathbf{m}_i, \mathbf{m}_j \rangle, \mathcal{E}_\mathcal{T})$
    $copyRobot(\mathbf{m}_i, \mathbf{m}_j)$
    $copyActiveFeat(\mathbf{m}_i, \mathbf{m}_j)$
    $i = j$ {*Map change*}
  end if
  if reobserved $\mathbf{f} \notin \mathbf{m}_i$ & $\mathbf{f} \in \mathbf{m}_j$ **then**
    {*Subsection III-B*}
    for $\langle \mathbf{m}_k, \mathbf{m}_l \rangle$ in path$(\mathbf{m}_j, \mathbf{m}_i)$ **do**
      $copyFeat(\mathbf{f}, \mathbf{m}_k, \mathbf{m}_l)$
    end for
    $addEdge(\langle \mathbf{m}_j, \mathbf{m}_i \rangle, \mathcal{E}_\mathcal{G} \backslash \mathcal{E}_\mathcal{T})$
  end if
  $\mathbf{m}_i = ekfUpdate(\mathbf{m}_i, \mathbf{z}_k, \mathbf{R}_k, \mathcal{DA}_k)$
  $\mathbf{m}_i = addNewFeatures(\mathbf{m}_i, \mathbf{z}_k, \mathbf{R}_k, \mathcal{DA}_k)$
end for
{*Subsection III-D*}
$updateAllMaps(\mathbf{m}_i, \mathcal{T})$ {*Updates* $\mathcal{T}$ *starting from* $\mathbf{m}_i$}

A. Starting a new submap

Suppose that robot is in submap $\mathbf{m}_i$ and we decide to start a new submap $\mathbf{m}_j$. The steps followed in the algorithm are:

- Add $\mathbf{m}_j$ to $\mathcal{N}$
- Add edge $\langle \mathbf{m}_i, \mathbf{m}_j \rangle$ to $\mathcal{E}_\mathcal{T}$
- Copy robot pose and last seen features from $\mathbf{m}_i$ to $\mathbf{m}_j$

In fact, the robot pose is copied twice in submap $\mathbf{m}_j$. The first copy will represent the current robot position which changes as the robot moves through the new map. The second copy will represent the initial position of the robot when it entered the map. This initial pose remains fixed as a common element with map $\mathbf{m}_i$.

An example can be seen in figure 2. At time k_2, submaps $\mathbf{m}_1$ and $\mathbf{m}_2$ have been already explored and a new submap is being created $\mathbf{m}_3$. Nodes $\mathbf{m}_1$ and $\mathbf{m}_2$ share in common a robot position R_{k_1} and a feature f_4. Submap 3 is initialized

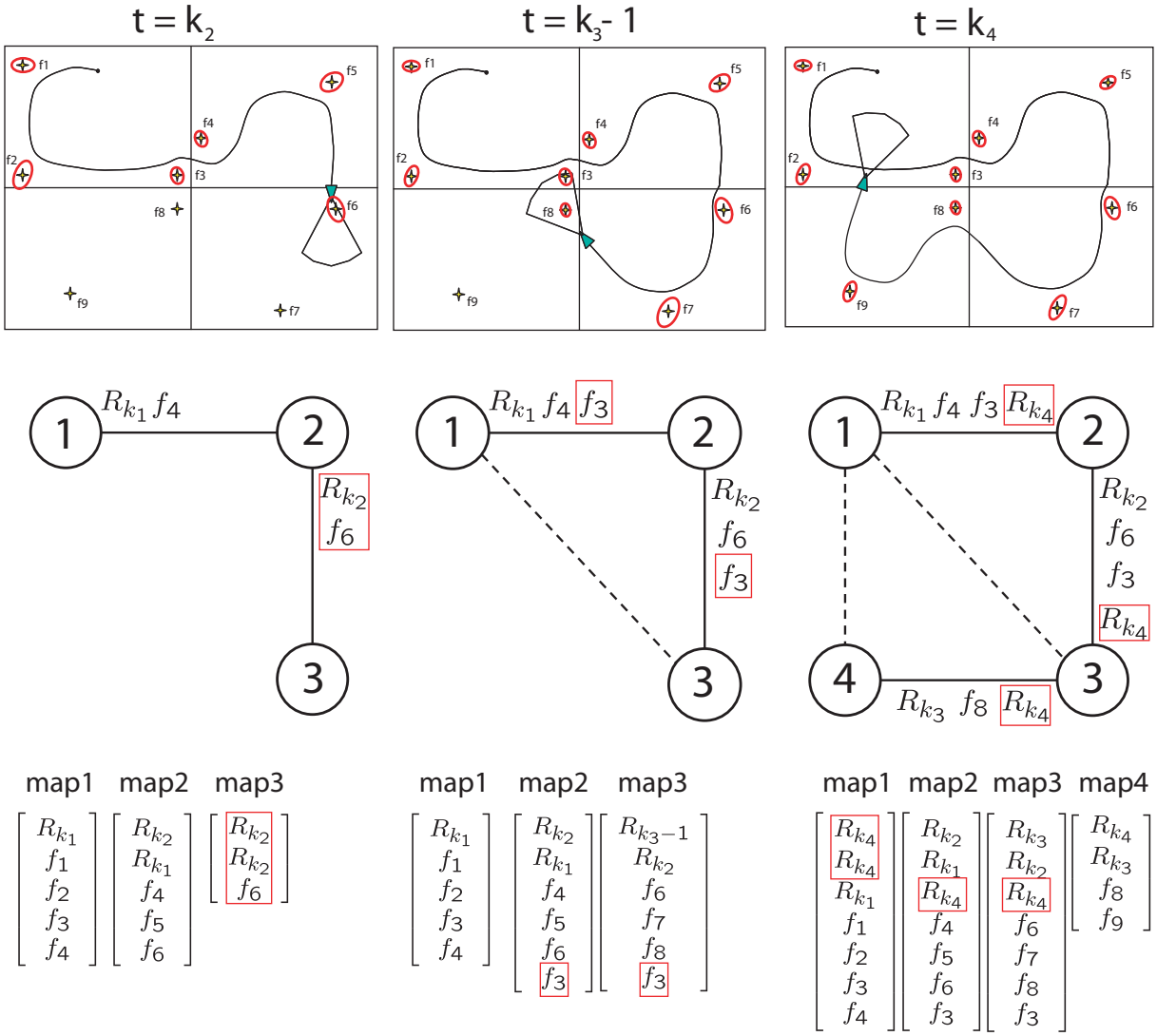

Fig. 2. Example using CI-Graph SLAM. The figure is divided in three rows that show information about the state of a simulated experiment at three different instants of time (columns). In the first row, the map of the simulated environment with the current robot position is shown. In the second row, the graph of relations between submaps will be created according to the state of the estimation. In the last row we will show the state vectors of the estimated submaps at different moments of time.

with robot R_{k_2} and feature f_6 from submap 2.

B. Re-observing a feature from a different map

This situation occurs when the robot is at submap $\mathbf{m}_i$ and observes *for the first time* a feature that is already included in a previous submap $\mathbf{m}_j$. The process followed is:

- Copy the feature from $\mathbf{m}_j$ to $\mathbf{m}_i$ along all nodes of the path in $\mathcal{T}$
- Add $\langle \mathbf{m}_j, \mathbf{m}_i \rangle$ to $\mathcal{E}_\mathcal{G} \backslash \mathcal{E}_\mathcal{T}$

If $\langle \mathbf{m}_k, \mathbf{m}_l \rangle \in \mathcal{T}$ represents an edge in the path, to copy the feature from $\mathbf{m}_k$ to $\mathbf{m}_l$, the feature is first updated with the information contained in $\mathbf{m}_l$ using *back-propagation* equations (5-8) and the correlations with the elements of $\mathbf{m}_l$ are calculated with equation (9).

Figure 2 at time $k_3 - 1$ shows an example of this case. Feature f_3 that belongs to submap $\mathbf{m}_1$ is measured by the robot when it is traversing submap $\mathbf{m}_3$. Since edge $\langle \mathbf{m}_1, \mathbf{m}_3 \rangle \notin \mathcal{T}$, f_3 is transmitted along the path $\langle \mathbf{m}_1, \mathbf{m}_2 \rangle$, $\langle \mathbf{m}_2, \mathbf{m}_3 \rangle$ that connects both nodes. Observe that the feature is replicated in all intermediate nodes. Finally, edge $\langle \mathbf{m}_1, \mathbf{m}_3 \rangle$ is included in $\mathcal{E}_\mathcal{G} \backslash \mathcal{E}_\mathcal{T}$.

C. Revisiting a previous submap

When the algorithm detects that the robot revisits an already traversed area $\mathbf{m}_j$, the transition from the current submap $\mathbf{m}_i$ to $\mathbf{m}_j$ is as follows:

- Update all nodes in the path from $\mathbf{m}_i$ to $\mathbf{m}_j$
- Copy the current robot pose along all nodes of the path
- Add $\langle \mathbf{m}_i, \mathbf{m}_j \rangle$ to $\mathcal{E}_\mathcal{G} \backslash \mathcal{E}_\mathcal{T}$

As in the previous subsection, to update submaps in the path we use the *back-propagation* equations (5-8) and to copy the current robot pose, correlations with submaps elements are calculated with equation (9).

Figure 2 at time k_4 shows an example of this operation. When the robot makes a transition between submaps $\mathbf{m}_4$ and $\mathbf{m}_1$, current robot position R_{k_4} is replicated along all nodes that are in the path, i.e., along $\mathbf{m}_3$, $\mathbf{m}_2$ and $\mathbf{m}_1$. Finally, edge $\langle \mathbf{m}_4, \mathbf{m}_1 \rangle$ is added to $\mathcal{E}_\mathcal{G} \backslash \mathcal{E}_\mathcal{T}$ and submap $\mathbf{m}_1$ becomes the current map.

D. Updating all maps from the current submap

Using the Graph operations just described, we can assure that the current submap is always updated with all available information. In addition, the CI property between submaps is preserved. An interesting property of the back-propagation equations is that they can be applied at any moment. They work correctly even if we back-propagate twice the same information: the terms inside the parentheses in equations (7,8) will be zero and the maps will remain unchanged. This allows us to schedule the back-propagation in moments with low CPU loads, or when graph operations are required. If the whole map has to be updated, the *back-propagation* equations are recursively applied starting from the current node and following the spanning tree $\mathcal{T}$.

IV. EXPERIMENTS AND RESULTS

CI-Graph SLAM has been tested using a simulated environment that emules a Manhattan World, as the one proposed in [11], with 2420 point features lying on the walls of a 11×11 matrix of building blocks. For this $2D$ example, the total space is divided in submaps using a grid cell. When the robot crosses the border between two cells for the first time a new submap is initialized. If the arriving cell has already been traversed we consider that a previous submap is revisited. The actual size of each submap, is not limited to the number of features content in a cell but to the number of features that are *observed* from it. For more general situations, $3D$ environments with complex topologies and different kind of sensors such as cameras, the decision to start a new submap can be based on the maximum number of features allowed in a a map, to limit computational complexity, or on a maximum value for the uncertainty of the robot position, to improve the consistency of the result.

In the Manhattan environment, the vehicle performs a randomly chosen trajectory of 1600 steps of $1m$. Fig. 3 top, shows a smaller 5×5 example to give the reader an idea of the experiment. Each time the vehicle reaches a block corner (circles), a control input is applied randomly. This kind of motions allows the robot to perform any trajectory: the robot can move from one map to its neighbor and performs any large loops. The motion model noises are assumed to be gaussian with respectively $\sigma_{xy} = 0.05m$ and $\sigma_\theta = 0.3deg$ standard deviations in position and orientation. As the robot moves, the graph of CI-submaps is created on the fly. Fig. 3 bottom, shows an example of the resulting spanning tree with nodes numerated in the order they were created.

In this simulated experiment, Monte Carlo runs are particularly suitable to evaluate the CI-Graph SLAM efficiency.

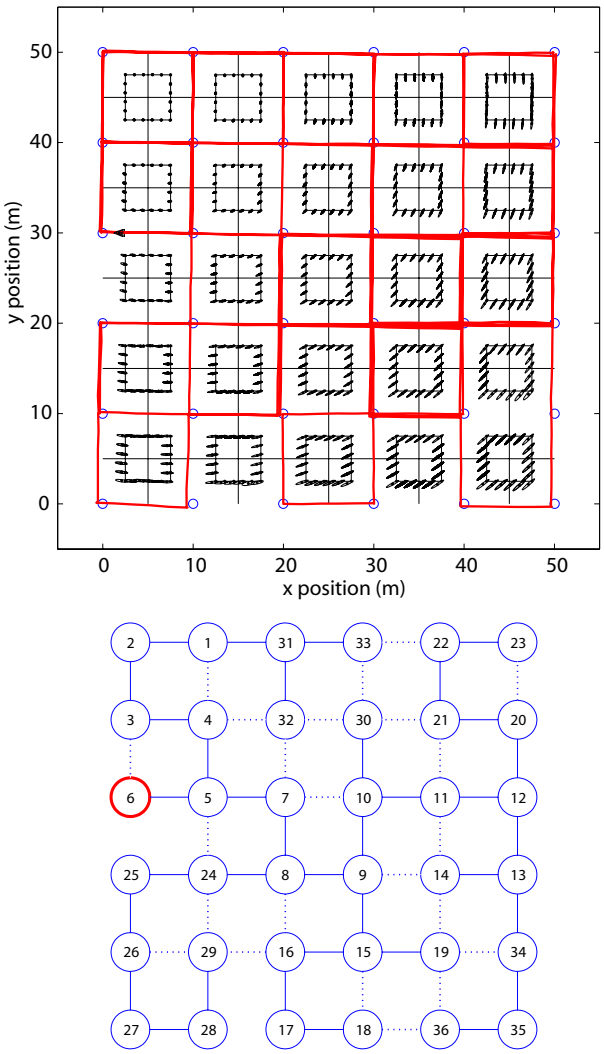

Fig. 3. CI-Graph SLAM execution on a Simulated environment of a 5×5 matrix of building blocks of a Manhattan World (top). The environment is divided up into 36 submaps (nodes) using a cell grid. The darker-red line represents the estimated trajectory. Ellipses also shows the estimated feature uncertainties. The final CI-Graph contains direct links (continuos lines) that forms the spanning tree between nodes (bottom). Indirect links are shown in dashed lines such that their represents mutual information seen between adjacent nodes. Thus, there exist a path formed by direct links through which the information can be transmitted.

We ran 300 samples of our algorithm implemented in MATLAB on a Pentium IV at $2.8GHz$. Note that each sample represents a different random trajectory and so, a different spanning tree. For the same reason, the number of total mapped features varies although the environment remains unmodified. Fig. 4 shows the mean running time per step. We can see that, for this kind of environment, the algorithm presents a close to linear running time.

We have also tested *CI-Graph* SLAM on Victoria Park data set as it is considered a benchmark for most of the relevant approaches previously mentioned in section I. Additionally, Victoria Park data set suits well due to its complex trajectory topology. As in the Manhattan World we use a grid cell to divide the space in submaps. Fig. 5

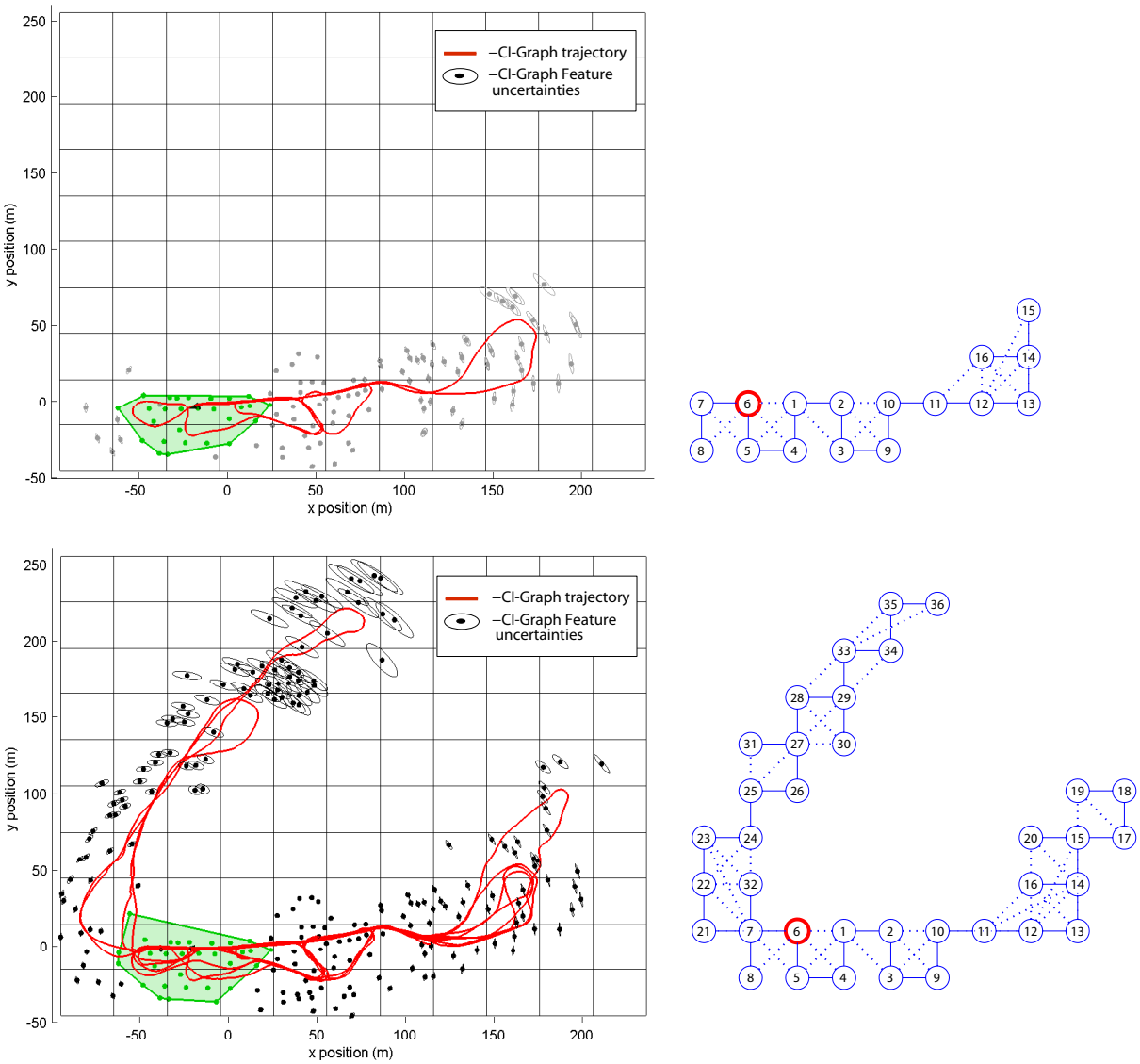

Fig. 5. *CI-Graph* SLAM execution on Victoria Park data set. The robot traverses local regions across the environment (left). During exploration, only the last submap is updated with all information available (top). At the same time, the spanning tree is being created with darker node as the current visited submap (right). The final result is obtained after building 36 nodes (submaps) using $30m$ of resolution in $x-y$ dimensions for our cell-grid (bottom). The accompanying video `video_CIGraph_xvid.mpg` (high quality version available http://webdiis.unizar.es/∼ppinies/video_CIGraph_xvid.avi) shows a slow execution of *CI-Graph* SLAM.

top, shows a partial result when the vehicle explores the environment with known data association. The accompanying video `video_CIGraph_xvid.avi` shows an execution in slow motion to visualize the CI-submaps building process. At the same time, it is posible to see that information is transmitted through direct links of the spanning tree when a node is discovered or when any two nodes share information. In addition, we ran EKF SLAM for comparison purposes. Figure 7 top, shows the running time per step, pointing out a constant behavior for the *CI-Graph* approach. Small peaks in the plot (up to $0.16sec$) represents the steps in which *CI-Graph* performs propagation. At final step, *CI-Graph* computes all optimum submaps propagating the information in $0.12sec$, one third of the time required for EKF SLAM ($0.37sec$). The map obtained is exactly the same as EKF SLAM with an absolute error difference of order 10^{-10} between vector estimates and of order 10^{-11} between estimated diagonal covariances. The differences are mainly due to numerical errors rather than estimation errors. The total cost plot in fig. 7 bottom, shows evidence about the efficient performance of *CI-Graph* SLAM as it is 6.5 times faster than standard EKF SLAM. From fig. 7 top, we could point out a constant behavior of the time per step. However it is difficult to establish this as an insight since the environment presents a disperse feature distribution. The accompanying video `video_CIGraph_EKF_xvid.avi` shows both algorithm executions with same data association both running at their corresponding execution times. A convex hull is

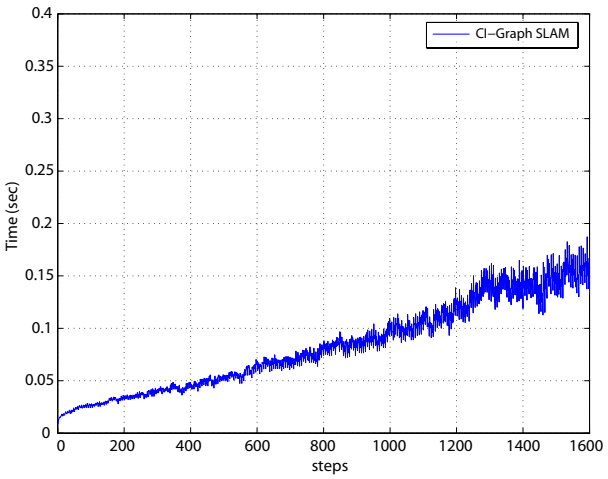

Fig. 4. Average running time per step for 300 random runs in 11×11 Manhattan World experiment. The environment is mapped with total size in a interval of $n = 947$ to $n = 2199$ point features. Also, for each run, the submap size does not exceeds 200 features in average.

drawn around the current submap considering its vehicle and features estimates while submaps already built are drawn in soft color to see the covered total map. The final updated result coincides with the final EKF map. Google Earh tool is used to show map precision on a real image of the environment (see fig. 6).

V. DISCUSSION

The price paid to maintain the conditional independence between submaps is some overhead in the size of the maps. We call overhead to all those elements of a submap that cannot be observed from it, i.e., robot positions corresponding to the transitions between submaps and features included in the current submap because its node is in a path in $\mathcal{T}$ between two nodes that share the features. For example, in figure 2 at $t = k_4$, robot position R_{k_4} is an overhead element for submaps 2 and 3. However features f_3 and f_4 are considered as intrinsic features of submaps 1, 2 and 3 since they can be observed from them. To reduce the overhead due to the robot positions, relocation methods could be applied once the submaps are well estimated. Regarding replicated features, the overhead is clearly dependent on the spanning tree used to transmit information through the graph.

Figure 8 on the left shows a graph with six submaps and its corresponding spanning tree represented with a continuous line linking nodes $5 - 4 - 1 - 2 - 3 - 6$. Suppose now that we are in submap 5 and we observe a feature in submap 6 closing the loop. In order to maintain the CI property between submaps, instead of directly closing the loop introducing a continuous link between nodes 5 and 6 we indirectly close the loop by replicating the observed feature along the submaps in the path between both nodes, as was explained in subsection III-B. This has the drawback of increasing the size of all the intermediate submaps increasing the computational cost. A better alternative would be to change the spanning tree to one with shorter loops as the

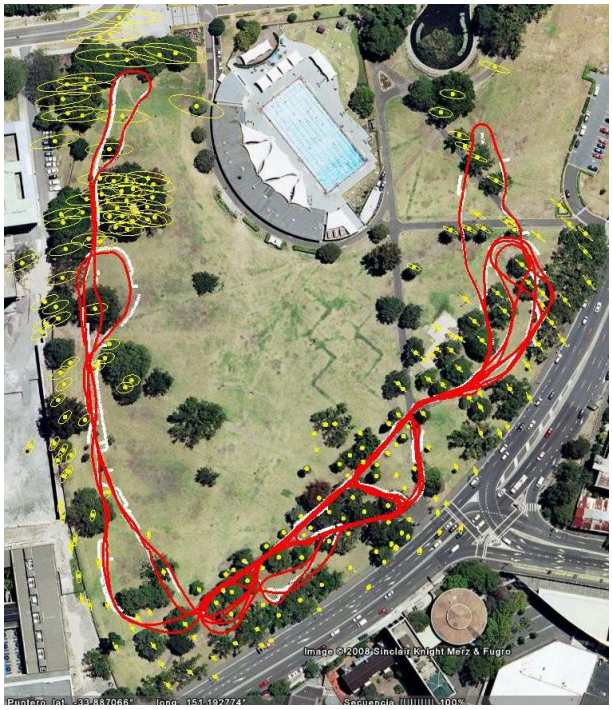

Fig. 6. Final Map obtained after running *CI-Graph* SLAM on Victoria Park data set. The map is projected using google Earth tool together with GPS data (white points).

example shown in Fig. 8 right. The new spanning tree has better properties because common elements between nodes linked with a dashed edge are only locally replicated. For example, common information between nodes 3 and 6 is just replicated in nodes 4 and 5. Therefore, in order to reduce the overhead, it is more convenient to generate a spanning tree with small loops between nodes connected with dashed edges. It is important to point out that the estimated solution obtained with any of the possible spanning trees is exactly the same and, in case of using absolute submaps, identical to the solution obtained with the EKF. The only difference is that the overhead introduced by the replicated features can be reduced and therefore we can improve the computational behavior of the algorithm. A method to find a good spanning tree for a given SLAM graph or how to online change a given spanning tree to a better one is left as future work.

VI. CONCLUSIONS

In this paper we have presented *CI-Graph*, an extension of the CI-submaps that allows us to efficiently solve complex map/trajectory topologies reducing the computational cost without approximations. *CI-Graph* models the SLAM process as a Gaussian Graph that evolves over time. Nodes of the graph correspond to CI-submaps and links between nodes reveal submap relations due to either robot transitions or covisible features. This results in a high level abstraction graph that allows a simple implementation. By building a spanning tree of the graph we have shown that information can be shared and transmitted from map to map without loosing the conditional independence property between submaps.

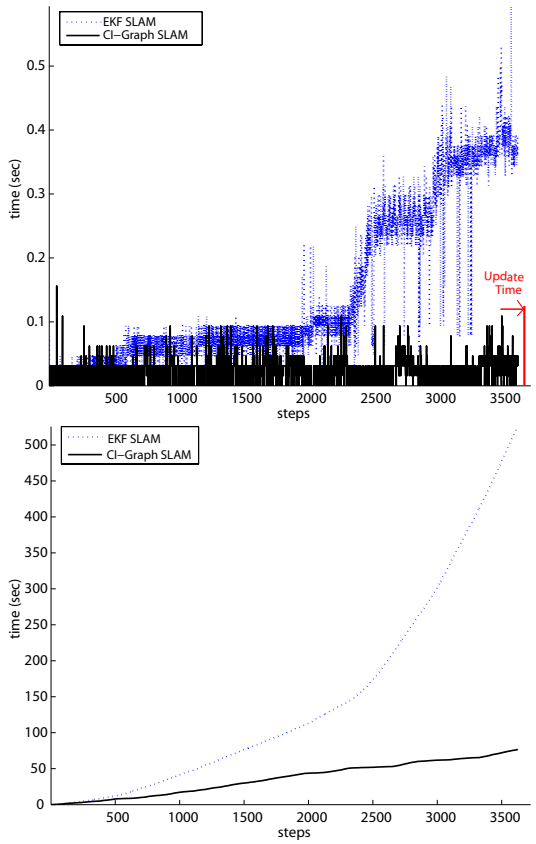

Fig. 7. *CI-Graph* SLAM vs. EKF SLAM running times. Time per step (top). Last time value corresponds to the time due to updating all submaps. Total execution time of EKF SLAM vs. *CI-Graph* SLAM(bottom). The accompanying video `video_CIGraph_EKF_xvid.mpg` (high quality version available in http://webdiis.unizar.es/∼ppinies/video_CIGraph_EKF_xvid.avi) shows that *CI-Graph* SLAM outperforms 6.5 times EKF SLAM. Additionally, *CI-Graph* only requires one third of the time required for EKF SLAM to compute the optimum map.

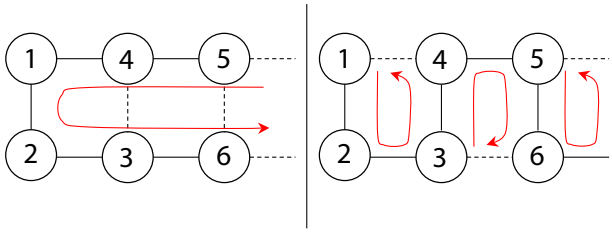

Fig. 8. A bad (left) and a good (right) spanning trees for the same graph.

One of the advantages of using *CI-Graph* with respect to other approaches, is its ability to reduce memory requirements when exploring an environment as it does not need to maintain all covariance matrix entries (correlation terms in EKF SLAM). We have also shown empirically the efficiency of *CI-Graph* SLAM to perform updates. In the presented experiments, we have obtained a cost per step close to linear time in the worst case. However, mathematical proofs about computational cost bounds will be analyzed in future work. A strategy to choose a good spanning tree to efficiently transmit information will also be addressed in future research.

CI-submaps have already shown to be very suitable for applications that involve the use of cameras in large environments. Sharing well localized features and camera states between CI-submaps gives much better results than starting each new submap from scratch. In future work we expect to proof that *CI-Graph* is a very powerful technique to solve Visual SLAM in large and complex environments.

VII. ACKNOWLEDGMENTS

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.

REFERENCES

[1] 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.

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

[3] 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.

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

[5] L. M. Paz, J. D. Tardós, and J. Neira, "Divide and Conquer: EKF SLAM in $O(n)$," *Aceepted in Transactions on Robotics (in Print)*, vol. 24, no. 5, October 2008. [Online]. Available: http://webdiis.unizar.es/ lpaz/publications_archivos/papers/dcslam.pdf

[6] J. J. Leonard and H. J. S. Feder, "A computationally efficient method for large-scale concurrent mapping and localization," in *Robotics Research: The Ninth International Symposium*, D. Koditschek and J. Hollerbach, Eds. Snowbird, Utah: Springer Verlag, 2000, pp. 169–176.

[7] M. Bosse, P. M. Newman, J. J. Leonard, M. Soika, W. Feiten, and S. Teller, "An atlas framework for scalable mapping," in *Proc. IEEE Int. Conf. Robotics and Automation*, Taipei, Taiwan, 2003, pp. 1899–1906.

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

[9] 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.

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

[11] 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.

[12] J. Folkesson and H. Christensen, "Graphical SLAM for outdoor applications," *Journal of Field Robotics*, vol. 23, no. 1, pp. 51–70, 2006.

[13] 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.

[14] P. Piniés and J. D. Tardós, "Large Scale SLAM Building Conditionally Independent Local Maps: Application to Monocular Vision," *Accepted in Transactions on Robotics (in print).*, vol. 24, no. 5, October 2008. [Online]. Available: http://webdiis.unizar.es/ jdtardos/papers/2008_IEEE_TRO_Pinies_Tardos.pdf

[15] S. Huang, Z. Wang, and G. Dissanayake, "Sparse Local Submap Joining Filters for building large-scale maps," *Accepted for publication in IEEE Transactions on Robotics*, 2008.

[16] C. M. Bishop, *Pattern Recognition and Machine Learning*. Springer, 2006.

[17] T. H. Cormen, C. Leiserson, and R. L. Rivest, *Introduction to Algorithms*, 2nd ed., T. M. Press, Ed., Cambridge, Massashusetts, 2001.