Closing the Loop With Graphical SLAM

John Folkesson, *Member, IEEE*, and Henrik I. Christensen, *Member, IEEE*

***Abstract*—The problem of simultaneous localization and mapping (SLAM) is addressed using a graphical method. The main contributions are a computational complexity that scales well with the size of the environment, the elimination of most of the linearization inaccuracies, and a more flexible and robust data association. We also present a detection criteria for closing loops. We show how multiple topological constraints can be imposed on the graphical solution by a process of coarse fitting followed by fine tuning. The coarse fitting is performed using an approximate system. This approximate system can be shown to possess all the local symmetries. Observations made during the SLAM process often contain symmetries, that is to say, directions of change to the state space that do not affect the observed quantities. It is important that these directions do not shift as we approximate the system by, for example, linearization. The approximate system is both linear and block diagonal. This makes it a very simple system to work with especially when imposing global topological constraints on the solution. These global constraints are nonlinear. We show how these constraints can be discovered automatically. We develop a method of testing multiple hypotheses for data matching using the graph. This method is derived from statistical theory and only requires simple counting of observations. The central insight is to examine the probability of not observing the same features on a return to a region. We present results with data from an outdoor scenario using a SICK laser scanner.**

***Index Terms*—Autonomous navigation, data association, localization, mapping, mobile robots, nonlinear estimation, simultaneous localization and mapping (SLAM).**

I. Introduction

THE problem of simultaneous localization and mapping (SLAM) has a robot trying to localize itself to a map while it concurrently is creating the map. SLAM is a widely studied problem [1] that initially was studied using an extended Kalman filter (EKF) [2]–[5]. However, the EKF has some shortcomings. First of all, the algorithm has quadratic complexity with respect to the size of the map. This problem has been solved in a number of ways [6]–[9]. However, the main problem with the EKF is that long before the computational cost becomes problematic, the map is likely to become inconsistent due to nonlinear effects [10], [11]. The linearization also makes it difficult to impose global constraints. One approach is to use a pose graph and impose the constraint on the edges of the graph as done in [12] using scan matching to set the links between poses. For us, a map is a set of features and their locations. These features are observed using some sensor attached to a robot, for instance, a SICK laser scanner or a camera. The features might be walls, lines, points, etc.

The map estimation problem involves topological and geometrical aspects. The topology specifies the global connections in the map, whereas the geometrical part is related to the local poses of the features and the robot. Rather than integrating these two aspects directly in a homogeneous estimation problem, we have decided to adopt a graph-based approach that in some aspects is reminiscent of the work by Lu and Milos [12]. Also relevant here are works by [13]–[16].

In our Graphical SLAM method, the map is represented as a graph of features and robot poses linked by *energy nodes* corresponding to odometry and feature observations. The SLAM problem is posed as an *energy* minimization problem, which is solved iteratively using relaxation. The energy is computed as the negative of the log-likelihood of the measurements.

The extended information filter (EIF) [17], can be thought of as a linearization approximation of our graph. The EIF information matrix can be represented by edges in a graph where the state is represented by the nodes. The robot pose at earlier times is marginalized out, i.e., it is replaced by its expectation value as a function of the remaining state variables and only the current pose is kept in the graph (see Fig. 1). This creates a densely connected graph for which it is computationally expensive to compute the expected state.

In our graph, the edges are replaced by energy nodes connecting the state nodes. These energy nodes are not constant numbers but rather functions. The information matrix corresponds to the Hessian of the energy in our theory. This Hessian is recalculated upon each change of the state. Thus, linearization errors are much smaller than in EIF and EKF. Also, our graph does not become densely connected as we chose to not marginalize out all the pose nodes. By leaving some pose nodes, we achieve sparseness without any of the consistency problems of sparse EIF (SEIF).

In [18], a graphical method is presented for fast computation of the SLAM solution based on organizing the computations into a tree. Each node of the tree represents the contributions from a subset of the measurements. By carefully building up the tree and eliminating state variables as one goes, the dimensionality of the nodes is about constant and thus a log N complexity is achieved. The tree map method and the method which we shall present share the same divide-and-conquer approach to the global optimization problem. The tree map method carries it further to really huge maps, while we concern ourselves somewhat more on the practical aspects of accuracy for moderately sized maps.

Another closely related work is [19]. Their approach is to form a hybrid map with fine detail in local maps and a much coarser graph giving the topology. Each node of their graph is

Manuscript received January 30, 2006; revised July 7, 2006, and November 20, 2006. This paper was recommended for publication by Associate Editor P. Rives and Editor L. Parker upon evaluation of the reviewers' comments. This work was supported by the Swedish Foundation for Strategic Research through the Center for Autonomous Systems, Stockholm, Sweden.

J. Folkesson is with Massachusetts Institute of Technology, Cambridge, MA 02139 USA.

H. I. Christensen is with the College of Computing, Georgia Institute of Technology, Atlanta, GA 30332 USA (e-mail: hic@cc.gatech.edu).

Digital Object Identifier 10.1109/TRO.2007.900632

1552-3098/$25.00 © 2007 IEEE

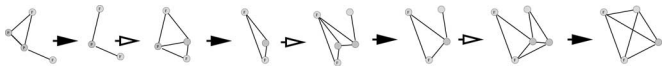

Fig. 1. Progress of the EIF is illustrated graphically. Here, the nodes for the robot poses are marginalized out at each step (filled arrowheads) leading to a densely connected graph of feature nodes and the current pose.

a local map. The local maps are small enough so that problems with linearization do not occur. The larger scale loop constraints are imposed on the graph in a way that is analogous to our coarse adjustment of the graph. For them, features from different local maps remain different and need not be made "consistent." In our approach, we try to bring the individual common features from different star nodes into a best common position. It can be questioned whether this is needed and, perhaps, the local map approach is sufficient for applications.

In [20], the pose chain graph is solved by a method of marginalization of all poses not directly involved in the loop constraints. This greatly simplifies the calculation when there are very many loops in the graph. As in the previous two works, features are not directly part of the graph and no attempt is made to force overlapping regions to be identical. Marginalization is also a part of our method, but we do not look at such extreme numbers of loops and do not need to carry out the marginalization as far.

Another major issue in SLAM is data association. This is the problem in each step to match recently observed features. When closing a loop, there can be a significant difference between the map and the observed features, which is a challenge. One strategy is to address this through the use of expectation maximization (EM), as presented in [21]. The decision to close a loop can be evaluated using the joint compatibility test [22], [23], when the state covariance matrix is available. An alternative solution to these problems was proposed in [24].

The linearization problems of some SLAM methods are avoided by using an explicitly nonlinear energy formulation. This energy is minimized by iteratively linearizing around the current solution. In this way, the system is always linearized around a single and current state, which may be far from some of the intermediate states.

The presented approach has a time complexity that is independent of the size of the map, as long as new areas are being explored. A return to a previously visited area requires a more expensive update to be done to ensure global consistency. The map is held locally consistent by an energy relaxation procedure, but global consistency requires some special treatment. A final advantage is that data associations can be changed at any time during the process.

The basic feature representation is presented in Section II, and the overall graphical model is presented in Section III. The stepwise updating is presented in Section IV. Through simplification of the graph, as presented in Sections V and VI, the complexity can be reduced. This simplification can be utilized after detection of loops (Section VII) for imposing topological constraints as presented in Section VIII. The methodology has been evaluated on a significant number of scenes, both in- and outdoor. Example results for outdoor scenes are presented in Section IX. Finally, the conclusion and discussion are presented in Section X.

II. Feature Representation in the M-Space

Observations made during the SLAM process often contain symmetries, that is to say, directions of change to the state space that do not affect the observed quantities. It is important that these directions do not shift as we approximate the system by, for example, linearization. To this end, we would like to be able to explicitly represent and separate out the symmetries of our measurements. Our feature representation is the measurement subspace or simply the M-space [25], [26]. The features are parameterized by a set of coordinates $\{\mathbf{x}_f\}$, which have well-defined transformation rules under translations and rotations. So, if $\mathbf{x}_o = T(\mathbf{x}_f|\mathbf{x}_r)$ are the feature parameters in a new reference frame,[1] which is parameterized by $\mathbf{x}_r$, then changes to these coordinates satisfy

$$\delta\mathbf{x}_o = J_{of}\delta\mathbf{x}_f + J_{or}\delta\mathbf{x}_r. \tag{1}$$

The J's are the Jacobians of the transformation.

The features are observed by taking measurements. These measurements then give us some information on some directions in the feature parameter space. That is to say, we gain information in a subspace of the full feature space. It is the union of the subspaces from all the measurements that constitute the M-space. Perturbations in these directions (notated as $\delta\mathbf{x}_p$), are projected from changes in the parameter space $\delta\mathbf{x}_f$ by the B matrices.

$$\delta\mathbf{x}_p = B(\mathbf{x}_f)\delta\mathbf{x}_f \tag{2}$$

$$\delta\mathbf{x}_f = \widetilde{B}(\mathbf{x}_f)\delta\mathbf{x}_p \tag{3}$$

$$I_{pp} = B(\mathbf{x}_f)\widetilde{B}(\mathbf{x}_f). \tag{4}$$

The I_{pp} is the identity matrix in the measurement subspace.

The idea is to use the measurements to calculate a $\delta\mathbf{x}_p$ and, then, the previously-mentioned equations to get changes to the $\mathbf{x}_f$. An example of a measurement symmetry is a measurement of a line feature where only the angle and perpendicular distance to the line are measured. These observations then are invariant to changes in the length of the line. The M-space in this case is the movement of the two endpoints perpendicular to the line. The B matrix will have two rows that project out these directions of movement.

III. The SLAM Graph

We have introduced graphical SLAM in [26] and [27]. The graph represents the likelihood of our measurements and data associations. It is built up from state nodes and energy nodes where each edge runs between a state node and an energy node (see Fig. 2). The simplest state nodes are pose nodes that represent

[1]Here the notation $T(\ldots)$ is used to signify a transformation of coordinates to a new rotated and translated basis. The subscript r is used for the transformation's coordinates (often the robot sensor frame), and o is used for the feature coordinates in the new frame (often some observation frame).

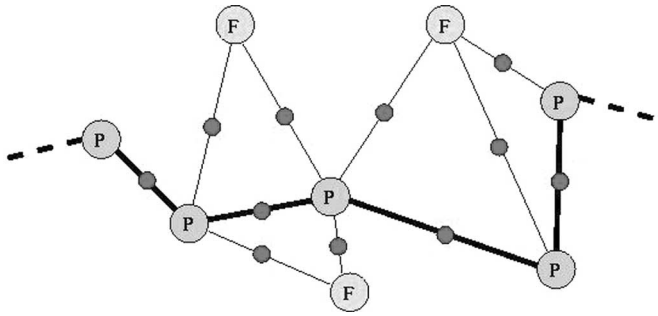

Fig. 2. This shows how a graph is built up. The lighter pose nodes are labeled P, the feature nodes F. The darker smaller circles are the energy nodes representing measurements. We, sometimes, focus attention to the subgraph of just pose nodes and the energy nodes connecting them. This subgraph is shown with heavier edges here.

the state of the robot and sensor poses at some point in time. Other state nodes represent features. An energy node calculates an energy, based on the state of all its attached state nodes. This energy originates from measurements of the relative position of the state nodes. If the measurements[2] η obey some probability distribution, given the states $P(\eta|\mathbf{x}_f)$, then the energy is given by

$$E(\mathbf{x}_f, \eta) = -\log(P(\eta|\mathbf{x}_f)) - \Lambda\kappa \tag{5}$$

Here, κ is 1, if the measurement is matched to an existing feature and 0 otherwise. The Λ then gives some energy gain for using existing features to explain new measurements. Thus, if one finds that the energy decreases after adding an energy node, the match can be considered to be correct. One can test various combinations of data associations by attaching edges between nodes in the graph.

The Λ term arises from an assumption that the priori distribution over the domain of possible maps is $e^{-\Lambda N_f}$, where the number of features in the map is N_f. In other words, we introduce a bias for simple maps. In practice, the Λ enters the data association in the same way as a threshold on the Mahalanobis distance does for the EKF. The difference is that we can evaluate the likelihood more accurately than the EKF. The magnitudes for Λ are typically between 1 and 16 depending on how well separated the features are.

The nodes and edges can be added immediately when observing a new feature. In the M-space model, new features will not have any effect on the map until they collect enough information to initialize some of the M-space dimensions. At that time, all the edges of the feature can be tested again to see if they lower the energy. Thus, the measurements are not added in any final way but can be removed later. We also add energy nodes for odometry linking two pose nodes.

[2]The η here is an innovation vector for the measurements. One chooses a function for the innovation that depends on the observation and the state. The innovation should have zero expectation value averaged over the measurement noise. It will also be necessary to make assumptions about the distribution of noise projected into the innovation. We assume Gaussian, but that is not a fundamental requirement. An example would be the difference in the angle and distance to a wall between the measurement and that predicted by the current state of the robot and wall.

Typically, the measurements η depend on the position of the features relative to some sensor attached to the robot. So, if the robot sensor frame has coordinates $\mathbf{x}_r$ and the relative feature coordinates are $\mathbf{x}_o = T(\mathbf{x}_f|\mathbf{x}_r)$, then we can estimate a small change in the predicted measurement $\eta(\mathbf{x}_o)$

$$\delta\eta = J_{\eta o}\delta\mathbf{x}_o = \left(\underbrace{J_{\eta o}J_{or}}_{robotpose} \quad \underbrace{J_{\eta o}J_{of}\widetilde{B}_f}_{feature} \right) \begin{pmatrix} \delta\mathbf{x}_r \\ \delta\mathbf{x}_p \end{pmatrix}. \tag{6}$$

If the η are Gaussian with covariance $C_{\eta\eta}$ and mean 0, then the energy is given by

$$E = \frac{1}{2}\eta(\mathbf{x}_o)^T C_{\eta\eta}^{-1} \cdot \eta(\mathbf{x}_o) - \Lambda\kappa. \tag{7}$$

Putting this together, we can write the gradient G_m for the energy.[3]

$$G_m = \eta^T C_{\eta\eta}^{-1} \left(J_{\eta o}J_{or} \quad J_{\eta o}J_{of}\widetilde{B}_f \right) \tag{8}$$

We can also calculate the Hessian of the energy in a similar fashion

$$H_{mm} = \begin{pmatrix} J_{or}^T J_{\eta o}^T \\ \widetilde{B}_f^T J_{of}^T J_{\eta o}^T \end{pmatrix} C_{\eta\eta}^{-1} \left(J_{\eta o}J_{or} \quad J_{\eta o}J_{of}\widetilde{B}_f \right). \tag{9}$$

These quantities E, H_{mm}, and G_m are calculated by the energy nodes based on the state nodes. They are the fundamental quantities of the graphical SLAM method.

IV. Graph Relaxation

Each state node has a number of edges connecting it to energy nodes. These energy nodes can calculate the nonlinear energy function and its derivatives at the current state. Then, for each state node, we can sum the contributions to the gradient and Hessian from each attached energy node. Using the gradient and Hessian, we can relax the state node, that is, move it toward a lower energy state

$$H_{mm}\Delta\mathbf{x}_m = -G_m \tag{10}$$

where the Hessian and gradient are taken with respect to just this one state node's coordinates by adding up the contributions from all the attached energy nodes. We use this formula for each node, in turn, for a section of the graph. The result is similar to a Gauss–Seidel iterative solution for the minimum energy state of a linear system. So, after introducing a perturbation to the system in the form of a new energy node, we relax all the state nodes attached to the energy node. We, then, calculate the change in energy of each energy node attached to these state nodes. If significant change to an energy node has occurred, it gets added to a list of energy nodes. We, then, take this list and repeat the process.

How do we define significant change in an energy node. We first require that the energy change be greater than a nominal small amount 0.01. This test eliminates most changes. Then, we test the relative change in energy, that is, the ratio of the change to the energy after relaxation. If this relative change is larger

[3]The subscript $_m$ here is to emphasize that the gradient is with respect to the M-space perturbation as opposed to the full feature space with subscript $_f$. The gradient also includes the robot sensor frame coordinates. So, $_m = (_r, _p)$.

than some small percentage (we used 5%), then we include the energy node in the list.

This way of organizing the updates by state-attached energy nodes that change was inspired by the work in [28]. The result is a more efficient relaxation as no extra calculation needs to be done to decide where to move next. This is because the energy is always calculated after relaxing a state node in order to assure a decrease in energy. If the energy were to go up, the step size is reduced until a decrease is achieved.

We are able to have the relaxation move through the graph to those nodes that have the largest stresses. If the graph has lots of stress, the update will quickly expand to a large portion of the graph. On the other hand, most of the time, the relaxation can stay local and terminates quickly.

We show in the next section how to reduce the number of pose and energy nodes in the graph by forming star nodes. This reduction is not done until the measurements are somewhat mature. Therefore, the most recent path of the robot has a dense representation in the graph. This section of the graph is referred to as the tail as it looks like the robot has a tail of pose nodes that it is dragging after it. In fact, the pose nodes do not follow the robot but are being constantly created at the robot location and reduced by forming stars at the end of the tail. This tail needs special attention during relaxation.

The tail presents problems as stress is normally introduced at one end and is slow to propagate along the tail. We found the solution was to relax the tail as a unit. We solve the linearized system of tail pose states holding the feature states fixed. The technical details follow.

The Hessian H of the subsystem consisting only of the pose nodes in the tail is block tridiagonal with 3×3 blocks.[4] This system can be solved easily with care given to the numerical stability. We do not solve the block tridiagonal system $Hx = -G$ directly but rather $H'x' = -G'$, where $H' = P^T H P$, $x = Px'$, and $G' = P^T G$. Here, P is

$$P = \begin{pmatrix} I & 0 & 0 & \cdots \\ I & I & 0 & \cdots \\ I & I & I & \cdots \\ \cdots & \cdots & \cdots & \cdots \end{pmatrix}. \tag{11}$$

Here, I is the block identity matrix. The reason for doing this is that H is simple but numerically unstable, while H' is strongly block diagonally dominant. There is 100% fill produced by P, but by being careful to exploit the special structure, the complexity remains linear in the number of nodes in the tail. Every 25 steps, we relax the whole tail and, then, the features attached to it.

Using these two relaxation schemes, we found that updates could be done in reasonable time. The calculations needed for an update do not depend on the size of the map but instead on the amount of stress being introduced. When the robot is exploring new areas, very little stress is added and the updates go very fast. When the robot returns to a region previously visited, the graph must loop back on itself creating tension that needs to be resolved. It is then that updates begin to take significant time. This variable update time is awkward for a program that is to run under real-time constraints. It is necessary to have a queue of updates and have the program running out of sync with the sensor. Sometimes, the queue will be growing while, at other times, it will shrink. On the other hand, it is an advantage that the size of the map does not add any calculation so that very large graphs can be built. In practice, we found that the variation in update time was not very significant. The key is not to relax too hard any one iteration but rather rely on the cumulative effects of successive iterations to reduce the tension.

[4]This is because each pose node has three coordinates and is attached to two other pose nodes.

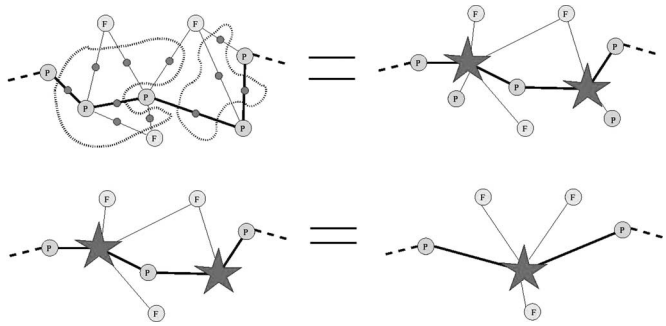

Fig. 3. This shows how a graph is reduced starting from the basic graph. We linearize all the energy nodes (enclosed by the dotted curves) attached to every other pose node into a star. We, then, eliminate the pose nodes that have only one edge. We, then, repeat with the pose node between two stars to merge them.

V. Graph Reduction by Star Nodes

The graph as described so far is a robust and flexible SLAM method. However, we will need to impose global constraints on the graph. The high dimensionality will make this difficult. Therefore, we will reduce the number of state nodes. We also will maintain a local frame of reference and, thus, avoid the problems of linearization in a global frame.

Consider a set of energy nodes. The gradient and Hessian terms from these nodes are collected together in a star node. This star can then replace the set as long as the changes in the state are small. If some state node only has edges to this star, we can eliminate its state from the quadratic energy. This is the reduction we are after, but we will also create fill in the Hessian. In addition, we will exploit the fact that the entire system approximated by the star is invariant with respect to the transformations of all the star's state nodes. This will allow us to transform the linearization coefficients (the Hessian and gradient), to any other local frame. Thus, we can remove the major component of linearization errors. All that will remain is the linearization errors due to relative movements between the star's states. As the star is formed in a mature part of the graph, we have reason to expect such movements to be small. We now will show how to derive a formula for the energy of such a star that can be recentered around any new state. We also show explicitly how the M-space symmetries are preserved.

The energy for the star looks so far like the Taylor expansion around $\bar{\mathbf{x}} = (\bar{\mathbf{x}}_b, \bar{\mathbf{x}}_a, \bar{\mathbf{x}}_f)$:[5]

$$E = E(\bar{\mathbf{x}}) + G_m(\bar{\mathbf{x}}) \cdot \begin{pmatrix} \delta\mathbf{x}_b \\ \delta\mathbf{x}_q \end{pmatrix} + \cdots. \tag{12}$$

Here, we have split the state perturbation vector $\delta\mathbf{x}_m$ into $\delta\mathbf{x}_b$ and $\delta\mathbf{x}_q$. The $\mathbf{x}_b$ is one of the pose node coordinates to become the "base" pose of the star and $\delta\mathbf{x}_q$ are the other pose and feature coordinates. We now transform all the coordinates to the base frame so that they become $\delta\mathbf{x}_o = T(\delta\mathbf{x}_q|\mathbf{x}_b)$. We can write the following identities:

$$I = \begin{pmatrix} I & 0 \\ -BJ_{of}^{-1}J_{ob} & BJ_{of}^{-1} \end{pmatrix} \begin{pmatrix} I & 0 \\ J_{ob} & J_{of}\widetilde{B} \end{pmatrix} \tag{13}$$

$$\begin{pmatrix} \delta\mathbf{x}_b \\ \delta\mathbf{x}_o \end{pmatrix} = \begin{pmatrix} I & 0 \\ J_{ob} & J_{of}\widetilde{B} \end{pmatrix} \cdot \begin{pmatrix} \delta\mathbf{x}_b \\ \delta\mathbf{x}_q \end{pmatrix} \tag{14}$$

$$G_m \begin{pmatrix} \delta\mathbf{x}_b \\ \delta\mathbf{x}_q \end{pmatrix} = G_m \begin{pmatrix} I & 0 \\ -BJ_{of}^{-1}J_{ob} & BJ_{of}^{-1} \end{pmatrix} \begin{pmatrix} \delta\mathbf{x}_b \\ \delta\mathbf{x}_o \end{pmatrix}. \tag{15}$$

The B and J_{of} matrices in (15) are block diagonal matrices with blocks for each state node that is connected to the set of energy nodes. The J_{ob} is a column of blocks. The original energy nodes were invariant under transformations of all the state nodes to a new frame. Thus, the matrix multiplying $\delta\mathbf{x}_b$, as shown in the previous equations, must vanish identically. Also, we can always find the full G_m from the G_q and the symmetry matrices

$$G_m = \begin{pmatrix} G_b & G_q \end{pmatrix} = G_q\begin{pmatrix} BJ_{of}^{-1} \end{pmatrix}\begin{pmatrix} J_{ob} & J_{of}\widetilde{B} \end{pmatrix}. \tag{16}$$

We define $Q = BJ_{of}^{-1}$ and $\widetilde{Q} = (J_{ob}, J_{of}\widetilde{B})$ and write the invariance relation more compactly as

$$G_m(\bar{\mathbf{x}}) = \{G_q(\bar{\mathbf{x}})Q(\bar{\mathbf{x}})\}\widetilde{Q}(\bar{\mathbf{x}}). \tag{17}$$

We can do the same for the Hessian

$$H_{mm}(\bar{\mathbf{x}}) = \widetilde{Q}(\bar{\mathbf{x}})^T\{Q(\bar{\mathbf{x}})^T H_{qq}(\bar{\mathbf{x}})Q(\bar{\mathbf{x}})\}\widetilde{Q}(\bar{\mathbf{x}}). \tag{18}$$

At this point, we examine our choice of linearization point $\bar{\mathbf{x}}$. We chose to relinearize the energy nodes around a point that produces $G_q = 0$. This is the equilibrium point of the star node if it were isolated from the rest of the graph. This choice makes the star node independent of the current state of the graph.

The H_{qq} matrix is the Hessian at the linearization point without the rows and columns of the base node. We want to have an explicitly stable representation; hence, we do a singular value decomposition of H_{qq}.

$$H_{qq} = \sum_{k=1}^{n} \mathbf{U}_k \lambda_k \mathbf{U}_k^T. \tag{19}$$

Here, k runs over the n positive eigenvalues λ and $\mathbf{U}_k$ are the eigenvectors. Define the projection into the eigenspace

$$\bar{u}_k = \mathbf{U}_k^T Q\bar{\mathbf{x}}_o \quad u_k = \mathbf{U}_k^T Q\mathbf{x}_o \quad \Delta u_k = u_k - \bar{u}_k. \tag{20}$$

[5]The star here has two pose nodes, which we we subscript b before/base and a after. It will also have a number of feature nodes that we simply denote by f. From here on, we refer to the entire set of star node coordinates by $\mathbf{x}$.

We can now write the energy for the star node in a compact invariant and stable form

$$E = E(\mathbf{u}) + \sum_{k=1}^{n} \frac{\lambda_k}{2}(\Delta u_k)^2. \tag{21}$$

We can also calculate the derivatives of the energy around any new state. So, for example, the gradient at $\mathbf{x}$ becomes

$$G_m(\mathbf{x}) = \sum_{k=1}^{n} \frac{\Delta u_k(\mathbf{x})\lambda_k}{2} U_k^T Q(\bar{\mathbf{x}})\widetilde{Q}(\mathbf{x}). \tag{22}$$

So, the eigenvalues and eigenvectors are saved along with Q and $\bar{u}_k$. These can, then, be used to find the energy for any new state.

VI. Star Formation

Having seen in the previous section how we can turn a set of energy nodes into a star node, we now discuss the choice of sets to turn into stars. We want to reduce the graph complexity by forming stars that include all edges from some state node. Then, that state node can be eliminated. We also need to limit the size of the stars as the calculation for creating the star eventually becomes a problem. Typically, we stop at a dimensionality of 40 for the star node.

We start building stars out of energy nodes attached to poses older than a certain age. A pose node is selected and all its attached energy nodes are combined into a star. We select every other pose node so as to form a chain of alternating pose and star nodes. We refer to stars made from combining the measurement nodes as level 0 stars. We can, then, again take every other pose and make stars, but now we will be making stars out of two level 0 stars. We call these level 1 stars. By continuing in this way, we combine stars of the same level to form stars of one higher level.

By forming the star nodes, we have committed to a linearization around the relative state in the local frames for the stars. The linearization is taken around the point that minimizes the energy for just the star's measurements, cut out of the rest of the graph. If the data associations are correct and the sensor errors relatively small, then the movement of the state from this linearization point will be small. The point is global movements of the star as a whole cause no error. Only relative movement of states within the star cause linearization error.

We define four sections of the graph. One is the *tail*, which consists of the current robot pose node and all the pose nodes before it back to the first star node. Next is the *formation set* in which stars are still being built up level by level. So, the formation set consists of alternating pose and star nodes. These stars are being combined pairwise to form bigger stars according to the rule of combining stars of the same level. When a star in the formation set gets large enough, all stars before it in the formation set are forced to merge, as much as possible, under a maximum-size constraint. Then, these large stars are removed from the formation set and moved to the *loop set*.

The *loop set* is where we try to build in global constraints to the graph. It will be discussed in the next section. The remainder of the graph is then the fourth set, the *mature nodes*.

The length of the *tail* is a variable that depends on the current features being observed. We do not eliminate energy nodes from features that are still being observed and have not been fully initialized.[6]

When we notice that a star in the *loop set* and one from the *mature nodes* share feature nodes, we try to combine them in order to build in the topological constraint in the graph. This can be done if the features they have in common are sufficient to define the transformation between the base frames.

VII. Finding Loops Automatically

In [27], a simple loop in the graph was found by stopping the SLAM program and entering the match between features from the start of the loop and the end of the loop by hand. Here, we present a way to have the SLAM program itself discover this match. We use an estimate of the energy increase from the match as a criterion.

Our approach is to find sets of features from both ends of the loop that can give us the constraint around the loop. Then, we use the information in the graph to estimate the energy gain from closing the loop. If there is a gain, then we can make the match and close the loop.

As the graph is being built, the new measurements are matched to the features that are close to the current robot pose, as measured along the edges of the graph. Thus, the edges of the graph have a distance associated with them that approximates how strongly the two nodes are correlated. The shortest distance from any feature node to the current pose node can then be found and the node labeled with the distance. The ones nearest the current pose are kept on a list that is matched to at each step.

By only trying to match features close in graph distance, we can prevent false matches from occurring when the robot closes a loop with significant error. Of course, this also prevents the map from ever closing loops; so, we need a second method to find matches over larger graph distances (see Appendixes I and II). That method must look at more features simultaneously. Thus, we wait until a patch has been mapped and star nodes formed before we attempt to do loop closing.

The *loop set* contains the newly formed large stars. The *loop features* are the features with an edge to the *loop set*. This *loop set* has stars added to it at a low frequency as described earlier.

Periodically, the *loop set* will be tested for loop closing. If no loop is found, the oldest stars are removed from the *loop set* and added to the set of *mature stars*. Some small number of stars remain in the *loop set* to be tested again.

Now, we need to narrow the list of potential matches. We use the metric information that we have for the features to match them within a tolerance that increases as the graph distance between the features increases. So, we are not in the so-called kidnapped robot situation where we know nothing. Instead, we are in a gray area between knowing nothing and having a good idea of the relative positions of the map patches we want to match. We use the graph distance to give some measure of this grayness.

We start by forming a list of *match features* that are not attached to the *tail* or *loop set*. Then, for each *loop feature* i we find all features j from the set of *match features* that satisfy this relation:[7]

$$({}_i\mathbf{x}_f - {}_j\mathbf{x}_f)^T B_i^T W_{ij} B_i ({}_i\mathbf{x}_f - {}_j\mathbf{x}_f) < 1. \tag{23}$$

The B_i matrix is evaluated at the current estimate of ${}_i\mathbf{x}_f$ and W_{ij} is a weight matrix that depends on the graph distance between the two features. This graph distance can be roughly approximated by the difference in the distance from the current pose node for the two features. The larger the graph distance between the features, the smaller W becomes. Ideally, one would use an inverse covariance matrix for W as in the joint compatibility test. Unfortunately, the correlations between distant features are not easily found from our graph.[8] One approximation available is the graph distance. Our W was proportional to the reciprocal of the graph distance. Each of the features i now has a list of features j that it could match to.

We now form pairs of features from those *loop features* that were matched. Each pair is tested to see if it defines a transformation, (i.e., for walls: is the angle between them large enough). If so, this pair becomes the base pair of a set of hypotheses. There will be one hypothesis for each distinct match between the pair and the *match features*. Each hypothesis then will be able to define a transformation based on the pair. All other potential matches are transformed with this transformation. Then, each match is tested by applying a threshold to $r_{ij} d_{ij}$

$$d_{ij} = ({}_i\mathbf{x}_f - {}_j\mathbf{x'}_f)^T B_i^T H_i B_i ({}_i\mathbf{x}_f - {}_j\mathbf{x'}_f) \tag{24}$$

$$r_{ij} = \frac{m}{trH_i} + \frac{Ma_{ij}}{M + a_{ij} trH_i} \tag{25}$$

$$a_{ij} = \frac{trH_j}{trH_j + trH_i}. \tag{26}$$

The H_i and H_j are the Hessians of the energy with respect to feature i and j, respectively, and ${}_j\mathbf{x'}_f = T({}_j\mathbf{x}_f)$ is the transformed matched feature. M is a large constant and m a small one. What we are trying to estimate here is the increase in energy caused by matching the two features.

The d_{ij} is an estimate of the energy increase of making the match, assuming we simply transform the end of the graph with no cost from the other features and then do not adjust the feature locations further. Then, a_{ij} estimates the reduction from the relative movement of features i and j after relaxation with no consideration of the other features. The r_{ij} is then a transformation of the estimate to avoid two possible problems. If the information of feature i, trH_i, is well between m and M, then $r_{ij} \approx a_{ij}$. For very high amounts of information, it

[6] In the M-space representation, it is possible to partially initialize the features.

[7] This formula is simplified slightly. Some minor modification needs to be made to B_i to account for different M-spaces of the two features; so that there are projection and scaling matrices to the common M-Space of the i and j features. Before testing with (23), we apply a few simple tests that eliminate the impossible candidates.

[8] These correlations are available in the EKF SLAM, but, typically, are inaccurate due to the accumulated linearization errors.

would be impossible to shift the features even one millimeter to merge them. That is why we chose M to be 10^6 corresponding to millimeter accuracy. This avoids unreasonably stiff criteria. On the other hand, if the information is very low, the features might be moved unreasonably far from the current positions. That is, the function of m, which is set to 100 corresponding to 10 cm.

This estimated increase $r_{ij}d_{ij}$ needs to be balanced against the gain from matching the features as in (5). In addition, we need to consider the likelihood of not having seen the same feature when the robot was in this area previously. Define the probability of observing a feature i in our set of *loop features* as p_i. If we try ni times, we have a probability of $(1-p_i)^{ni}$ of not observing it and $1-(1-p_i)^{ni}$ of observing it at least once. With these, we can now define L_h, a part of the energy of this hypothesis

$$L_h = -\sum_{i\in\text{matched}} \log(1-(1-p_i)^{ni}) - \sum_{i\in\text{unmatched}} \log(1-p_i)^{ni}. \tag{27}$$

Now, how do we estimate ni, the number of times we tried to "see" the looped feature from the *match features* part of the graph and p_i the probability of seeing i. This is done by maintaining two counters for every feature. One counter is incremented each time the feature should be seen from the current robot pose. The other counter is incremented when the feature is seen. The ratio of these is then p_i. The n_i is then the maximum of the first counter over all potential matches j.

By using (27), we make it harder to not match important very visible features and easier to not match hard-to-see features. It also tends to match important features to important features. A nice point is there are no parameters to choose while deciding our thresholds. We need only to count. Combining all these terms, i.e., (27), (24), and (5), we find our cost function that needs to be minimized

$$\text{Cost} = L_h + \sum_{i\in\text{matched}} (r_{ij}d_{ij} - \Lambda(dim)_i). \tag{28}$$

Here, $(dim)_i$ is the dimension of the feature and Λ is the same value used to match individual measurements (5). This gives the thresholds on individual features to produce the smallest cost. We also set some minimum number of matches before accepting the loop closure.

VIII. Solving the Topological Constraints

In the previous section, we showed how loops in the graph can be discovered automatically. Now, we must close the graph explicitly. We start by combining some of the stars in the loop set. We do this so as to form a single star from the loop set that has edges to the pair of features needed to define the transformation between frames. We do the same for the matched stars from the mature nodes. So, we then have a pair of stars, one from the loop set and one from the mature nodes that can be used to fix the transformation between the two parts of the graph. These two stars will eventually be merged, which will build the newly discovered topological constraint into the graph explicitly.

We first constrain the graph to have the indicated transformation when traversing from one star to the other. We solve for the path of pose nodes around the loop with all loop constraints imposed, which is described later. Having a path that satisfies all loop constraints, we move the edges from the loop features to their respective matched features. We can then carefully relax the features by turning on one edge at a time, starting with the strongest edges. The strength of the edge is given by the number of measurements that contributed to it. We move around the graph following the path from star to star, first adding all strong edges and relaxing the features, then relaxing the poses. We then go through the graph a second time adding the weaker edges. The tail section is built up similarly one edge at a time.

After relaxing the graph, the two stars can be merged, which will then build the constraint into the graph explicitly the next time a new loop is discovered. Later, such an explicit constraint can be found by searching for cycles in the graph.

The critical step is how to find the pose states that solve the constraints around all loops in the graph. We start by cutting off the tail from the graph. Thus, the only energy nodes that are still part of the graph are star nodes.

The feature nodes that have edges to more than one star are temporarily assumed to be different features for each star. Thus, these feature states can be eliminated from each star. As we are only interested in the pose nodes, this elimination amounts to taking H^{-1} for the star and ignoring the feature part, as we will explain later in this section. This is simply to write down the matrix in question because we have the eigenvector decomposition of each star node's energy. We give the details now.

We recall that the $\mathbf{x}_o$ variables of star j were the $\mathbf{x}_m$ relative to the base pose node of the star. The innovation in these $\delta\mathbf{x}_o$ was defined relative to the star equilibrium point. Thus, $\delta\mathbf{x}_0 = 0$ implies that $G_j = 0$. The Hessian with respect to the $\mathbf{x}_o$ variables of the jth star and the innovation of the relative coordinates of star j are defined as

$$H_j \equiv Q^T H_{qq} Q, \quad \mathbf{q}_j \equiv \delta\mathbf{x}_o. \tag{29}$$

The imposition of the global constraint on the pose states will require us to solve an equation that looks for each star node j, like

$$H_j\mathbf{q}_j = \begin{pmatrix} \zeta_j \\ \mathbf{0} \end{pmatrix}. \tag{30}$$

Here, ζ_j is obtained from solving the Lagrange multiplier part and has the dimension of the pose nodes for star j less the base pose node. The 0 is showing explicitly that the feature part of the right-hand side is zero. We do not care about the feature part of $\mathbf{q}_j$ either; hence, we do not care about the feature part of H_j^{-1} when solving this. We now show how we get to (30).

We write the constraint for all closed loops in the graph as

$$\sum_{j\in stars} C_j\mathbf{q}_j = \mathbf{c}. \tag{31}$$

Here, C_j are matrices that depend on linearizing the cumulative transformations around the loops. They do not have any parts multiplying feature states. So, the $\mathbf{q}_j$ are the changes to the

relative robot pose coordinates across a star. The sum is taken over all the star nodes. By adding the relative transformations around a loop, one finally ends up at a pose node related to the start pose node by a displacement $\mathbf{c}$. This $\mathbf{c}$ is inferred by the matching of features from the start and end of the loop. We can, then, formulate a cost function with this constraint built in by using a Lagrange multiplier γ

$$\sum_{j\in stars} \mathbf{q}_j^T H_j \mathbf{q}_j + \gamma\left(\sum_{j\in stars} C_j\mathbf{q}_j - \mathbf{c}\right). \tag{32}$$

Setting the variation to 0 leads to the equations

$$\sum_{j\in stars} \begin{pmatrix} H_j & C_j^T \\ C_j & 0 \end{pmatrix}\begin{pmatrix} \mathbf{q}_j \\ \gamma \end{pmatrix} = \begin{pmatrix} \mathbf{0} \\ \mathbf{c} \end{pmatrix} \tag{33}$$

$$\mathbf{q}_j = -\left(H_j^{-1}C_j^T\right)\gamma \tag{34}$$

$$\gamma = -\left(\sum_{j\in stars} C_j H_j^{-1} C_j^T\right)^{-1}\mathbf{c}. \tag{35}$$

So, as promised, we never had to compute the feature parts as we are only interested in correcting the pose nodes. The feature nodes will be relaxed as part of the fine tuning. These coarse calculations are all of low complexity and linear in the number of star nodes. The inverse of a matrix of size the order of the number of loops in the graph is needed. The other inverses, as said, are already available after forming the stars. Calculating C is normally the most difficult part.[9] The computation for each loop is linear in the size of the loop. This coarse calculation takes insignificant time compared to the fine tuning. The fine tuning can take on order of seconds for a large map. The time complexity of the fine tuning is not easy to determine. One needs to relax each node a number of times, which depends on the structure of the graph and the statistics of the measurements.

A variation on this is to use the current relative states of the pose nodes rather than the star equilibrium positions. The motivation is that these are already carefully relaxed locally and will, thus, lead to less local tension at the start of the fine tuning. The star Hessians are still used as described. This tends to give slightly better results when the current state is close to satisfying the constraint. We got the best results from using a weighted average where the weight depends on how large a transformation is needed to close the loop. When the current graph state is close to correct, we give it a higher weight.

IX. Experiments

A. Simulations

In order to demonstrate the basic premise that the graphical method could produce more accurate SLAM estimates more quickly than the EKF in some situations, we ran a simulation. By using simulated data, we can avoid problems of data association and focus on the computational aspects of the two methods. All the measurements are modeled exactly and drawn from Gaussian distributions. The associations are given and outliers are processed along with inliers for two simulations and a third was done with Λ equal to 16.

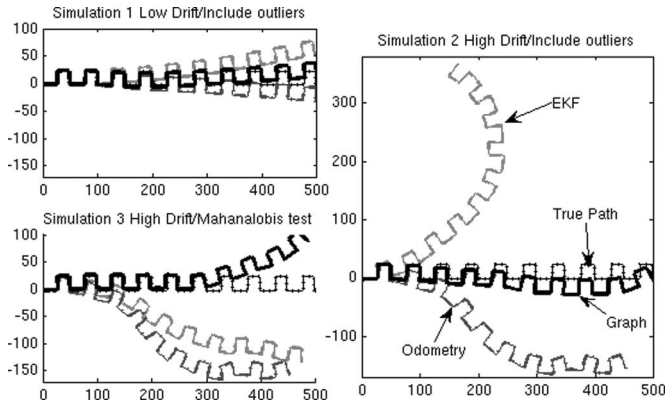

Fig. 4. EKF versus graph SLAM on simulated data. The paths follow the point features (not shown) in a sawtooth pattern. The features are ten units below and two units to the left of the robot along the path. The graph SLAM solution typically is more accurate and much faster in this particular situation.

The simulated environment consists of 1000-point features that could represent underwater sonar reflectors being detected by a surface vessel moving in 2-D. They are arranged in a sawtooth pattern ten units below and two to the left of the robot. The simulated robot receives range and bearing information. The bearing angles are modeled as Gaussian random variables, as is the range.[10] About eight features are within the sensor range along most of the path. The path has 10 809 poses from which 91.423 measurements in total were taken. Two simulations were done with different amounts of odometry error (Fig. 4) and the high-drift data processed with and without outliers included.

The EKF solutions took 56 min, 3 s, 55 min 3 s, and 19 min 49 s, respectively, for the three simulations. The Graph SLAM took 6 min 46 s, 6 min 45 s, and 8 min 14 s. No loop closing or star node reduction was done as this was a test of the basic computational behavior on a simple path with no loops. Loops will significantly increase the computational burden on the graphical method while not affecting the EKF. Adding the star nodes increased computation time by about 15 s while not significantly changing the results.

The statistics from the simulation are shown in Table I. Simulation 1 was with low odometric drift and all outliers, while 2 was with higher drift. The third simulation had the outliers removed. Having the true path allowed us to compare each incremental change in pose tangential and normal to the path as well as the angular change. The mean errors for these three are shown (in that order, in the mean vector). The unbiased estimate of the covariance of the errors is also shown. There were 10 808 increments. Thus, a rough idea of the error in the mean (or bias) is found by dividing the covariance

[9]We should mention as a practical point that if the corrections are significant, the linearization of the constraints may need to be done a few times. This will lead to minor modifications as earlier for linearization away from the $G_q = 0$ point.

[10]Standard deviations were $(0.01, 0.002, \text{and } 0.01)$, the first two being the polar angles in radians and the last is the range.

TABLE I
SUMMARY OF THE STATISTICS FROM THE SIMULATIONS

Sim. 1	mean (10^{-4})	cov (10^{-4})
EKF	$\begin{pmatrix} 0.48 \\ 6.31 \\ 0.18 \end{pmatrix}$	$\begin{pmatrix} 1.8 & 0.094 & 0.004 \\ 0.094 & 0.88 & 0.015 \\ 0.004 & 0.015 & 0.010 \end{pmatrix}$
Graph	$\begin{pmatrix} 0.0038 \\ 0.0656 \\ 0.070 \end{pmatrix}$	$\begin{pmatrix} 2.5 & 0.006 & 0.004 \\ 0.006 & 0.006 & 0.003 \\ 0.004 & 0.003 & 0.030 \end{pmatrix}$
Sim 2	mean (10^{-4})	cov (10^{-4})
EKF	$\begin{pmatrix} 7 \\ 26 \\ 2 \end{pmatrix}$	$\begin{pmatrix} 8 & 2 & 1 \\ 2 & 13 & 1 \\ 1 & 1 & 220 \end{pmatrix}$
Graph	$\begin{pmatrix} 0.35 \\ -0.37 \\ 0.41 \end{pmatrix}$	$\begin{pmatrix} 2.6 & -0.028 & -0.19 \\ -0.028 & 0.030 & 0.12 \\ -0.19 & 0.12 & 1.5 \end{pmatrix}$
Sim. 3	mean (10^{-4})	cov (10^{-4})
EKF	$\begin{pmatrix} 4.6 \\ -2.9 \\ -0.048 \end{pmatrix}$	$\begin{pmatrix} 6.3 & 0.56 & 0.29 \\ 0.56 & 6.32 & 0.52 \\ 0.29 & 0.52 & 0.76 \end{pmatrix}$
Graph	$\begin{pmatrix} 0.77 \\ -0.11 \\ 0.74 \end{pmatrix}$	$\begin{pmatrix} 2.3 & -0.056 & -0.30 \\ -0.056 & 0.037 & 0.096 \\ -0.30 & 0.096 & 1.38 \end{pmatrix}$

matrices by 10 808 and taking the square root. The covariance also gives a direct comparison of the errors inherent in the two methods.

The graphical method had significantly lower errors on the high-odometry drift simulation as compared to the EKF. The first simulation was less conclusive for the tangential and angular parts. The component normal to the path seemed to have significantly lower errors for the graphical case. We believe this is due to the effects of the linearizations. In addition, there is some weak evidence for bias in the EKF solution, which could be the result of the fact that all the features are to the left of the robot path. This could conceivably lead to a bias, although the exact mechanism was not determined. As expected, the inclusion of the outliers was better handled by the graphical method, which can be attributed to the linearizations of the EKF being much worse for the outliers.

B. Outdoor Experiments

We tested our ideas using data sets collected on an outdoor ATRV robot equipped with a SICK laser scanner and a 6-axis inertial sensor.

We show an example data set around our former laboratory building driving the robot by hand. The map is approximately 100 m^2. The path starting inside the laboratory travels down the corridor and out the front door, then around the building in a clockwise sense. After reentering via the front door, the first loop was detected. The robot then proceeded out the back door and detected a second loop. The path then went around to the front door again and closed a third loop. Finally, back in the laboratory, the entire map was relaxed.

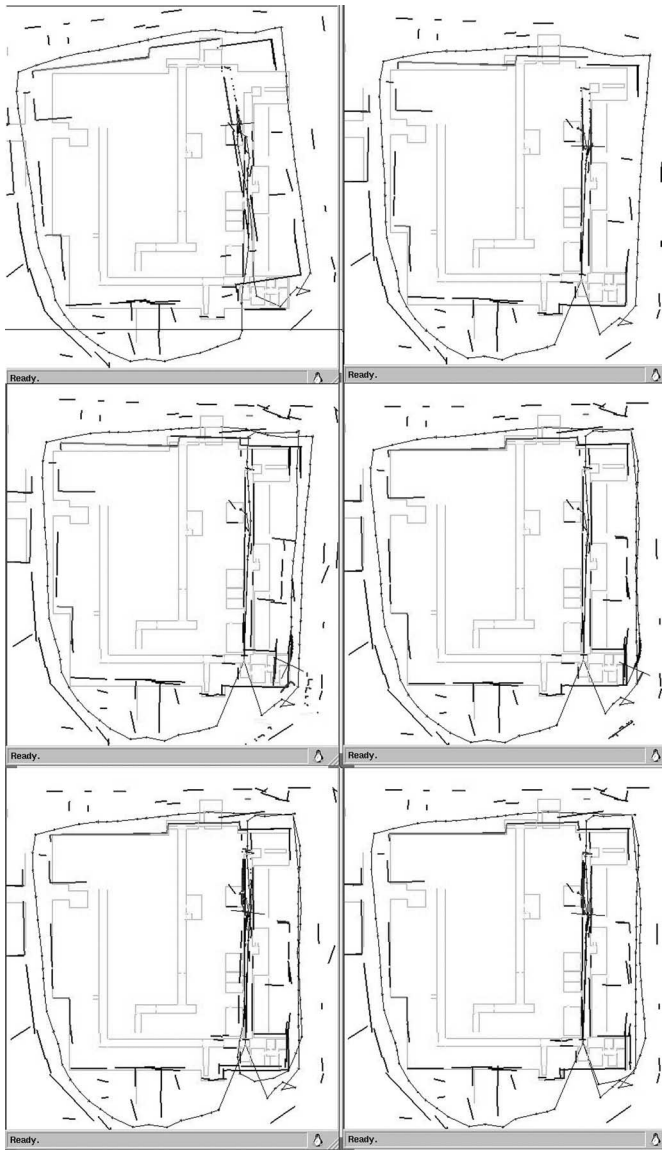

Fig. 5. This shows the map of the example data set just prior (left) to the discovery of each of the three loops and just after (right) imposing the constraint. One can see the improvement after each new constraint is added.

For this data set, the walls are often not found as there was work being done on the building. This forces the robot to cover large distances with no features at all. Such a situation is normally ruining for an SLAM program, but, here, we can recover by imposing topological constraints on the map. The results are shown as a sequence in Fig. 5. One can see that as each new constraint is added to the graph, the map improves. The final graph has 47 star nodes in total. It has two star nodes with more than two pose nodes. There is one star node with four pose nodes attached and one with six. These then allow the constraints around the loops to be reimposed automatically when each global update is done. At the end, the global update is done one last time to remove any extra tension in the map. The result is shown in Fig. 6. One can see that no mistakes were made in loop closing,

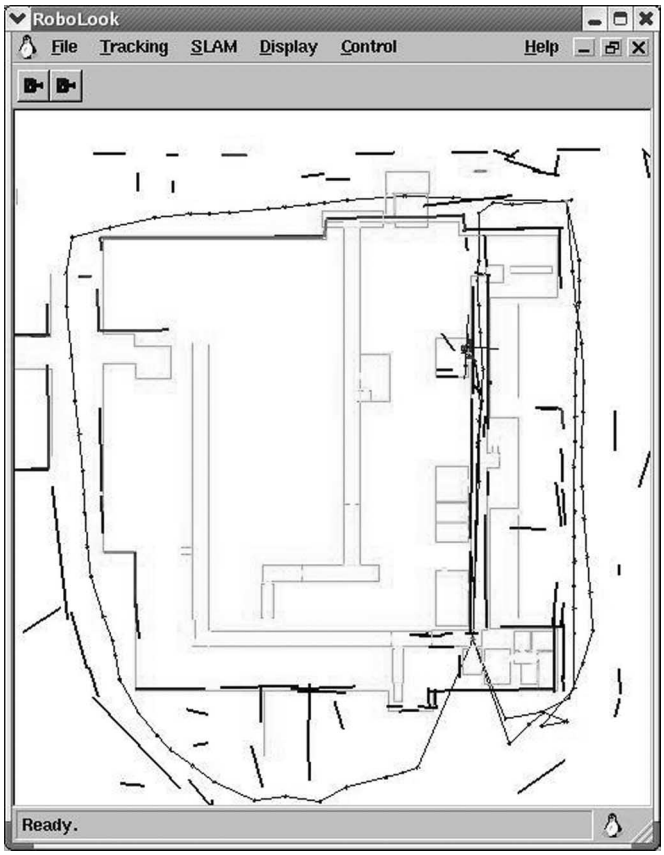

Fig. 6. Final map with the robot parked back in its laboratory. The path of the robot is shown as a solid line connecting the star nodes. The star nodes are spaced about every 2 m along this path. The handmade map of the building is shown for comparison.

but one match was not found in the lower right corner. The "missed" match was due to inconsistency between the indoor and outdoor sections of the map. The indoor sections could be matched or the outdoor but not both. This was probably due to tire slippage on crossing the threshold of the outer doorway to the lab.

The running time for the SLAM program is dependent on the processor. Here, we ran on a 550-MHz Pentium III. Graph updates took about 40 ms per step on average and the search for loops along with the imposition of the constraints added about 15% to the execution time. The scan period is 200 ms. Hence, the updates are able to keep up in this case.

The loop closing has been tried on even larger data sets and found to work similarly, but the time to find loops scaled quadratically with the number of features that needed to be checked. The number of features that need to be checked depends on the path taken by the robot and the density of features in the environment as well as the amount of pose error expected around the loop. So, when the features are dense or there is large pose uncertainty at loop closing, the time for discovering the loops greatly exceeded on-line limits. This is a fundamental limitation of any approach that tries to examine many hypotheses. The loop-closing criteria did correctly close the loops even in the harder situation.

X. Conclusion

We have shown how our graphical SLAM method can be used to both discover and solve multiple topological constraints on the map. We have done extensive tests with real data to confirm the validity of our approach. Our experience with these tests was that the search for and imposition of the topological constraints added about 10–20% overhead to the SLAM program. This depends on how often loops are formed. This overhead comes at the time the robot returns to a previously explored region. At other times, the SLAM is done quite quickly and the calculation time is independent of map size. So, for small maps, the time and quality is comparable to an EKF implementation we have done.

One expects that this method will be substantially faster and produce better maps than an EKF for really large data sets. This is due to the better treatment of nonlinearities and the fact that the updates are all local until global constraints are imposed.

We only linearize measurements in a local frame and explicitly maintain the invariants and symmetries in all measurements. This maintains the consistency in all of our approximations. Methods that linearize in a global frame can suffer from inconsistencies.

The criteria we use for loop detection were reliable and should be generally useful for closing loops even for other SLAM approaches. It seems to be important to consider the chance of not seeing features as well as measuring the similarity of the matched features. The method does scale badly with the number of features to be considered in hypothesis testing. If that number is too high, some other means must be used to lower the number of possibilities before testing with our criteria.

Appendix I

Test of Sufficiency

We need to test if a subset of features ${}_i\mathbf{x}_f$ is sufficient to fully define a transformation based on minimizing (28) to the matched j features. We start with the identity transformation with $\mathbf{x}_r = 0$ as our linearization point. To find the correction, we would need to solve

$$\sum_{i \in matched} r_{ij} J_{or}^T B_i^T H_i B_i ({}_i\mathbf{x}_f - {}_j\mathbf{x'}_f - J_{or}\delta\mathbf{x}_r +) = 0. \quad (36)$$

The sum is over the subset of loop features that are matched. So, if the matrix multiplying $\delta\mathbf{x}_r$ is rank 3, the subset is sufficient.

Notice that here the transformation is applied to ${}_j\mathbf{x}_f$. When we actually close the loop, we apply the inverse transform to ${}_i\mathbf{x}_f$. It is the need for the Hessian H_i that makes it easier to transform ${}_j\mathbf{x}_f$ while estimating the transform. We solve this several times, relinearizing each time until the adjustment to the transformation becomes small.

Appendix II

Graph Distances

We use a distance measure for an edge based on the Hessian of the edge's energy node. Each edge to an energy node has one state node. We consider the submatrix of the energy node's

Hessian that corresponds to the edge's state node. We calculate the eigenvalues and average the reciprocals of the nonzero eigenvalues. This is, then, the distance measure for the edge.

When conducting the graph search, we do not search paths that pass through feature nodes as the features normally are not sufficient to define the transformation. Thus, the paths pass through pose nodes and star nodes along the pose–star edges. The feature distances are then found by following all the feature's edges to the attached energy node and taking the lowest distance.

The search is repeated each time a loop closing is searched for, which happens at a relatively low frequency. During the time between such searches, the distances are approximated locally as we add edges to the graph.

REFERENCES

[1] S. Thrun, W. Burgard, and D. Fox, *Probabilistic Robotics*. Cambridge, MA: MIT Press, 2005.

[2] R. Smith, M. Self, and P. Cheeseman, "A stochastic map for uncertain spatial relationships," presented at the 4th Int. Symp. Robot. Res., Santa Cruz, CA, 1987.

[3] J. A. Castellanos, J. Montiel, J. Neira, and J. D. Tardós, "The SPMap: A probabilistic framework for simultaneous localization and map building," *IEEE Trans. Robot. Autom.*, vol. 15, no. 5, pp. 948–952, Oct. 1999.

[4] G. Dissanayake, H. Durrant-Whyte, and T. Bailey, "A compuationally efficient solution to the simultaneous localization and map building (SLAM) problem," in *Proc. IEEE Int. Conf. Robot. Autom.*, Apr. 2000, pp. 1009–1014.

[5] P. Newman, J. Leonard, J. Tardós, and J. Neira, "Explore and return: Experimental validation of real-time concurrent mapping and localization," in *Proc. IEEE Int. Conf. Robot. Autom.*, Washington, DC, May 2002, pp. 1802–1809.

[6] J. E. Guivant and E. M. Nebot, "Optimization of the simultaneous localization and map-building algorithm for real-time implementation," *IEEE Trans. Robot. Autom.*, vol. 17, no. 3, pp. 242–257, Jun. 2001.

[7] M. Montemerlo, S. Thrun, D. Koller, and B. Wegbreit, "FastSLAM: A factored solution to the simultaneous localization and mapping problem," presented at the Nat. Conf. Artif. Intell., Edmonton, Canada, 2002.

[8] Y. Lui and S. Thrun, "Results for outdoor-SLAM using sparse extended information filters," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2003, vol. 1, pp. 1227–1233.

[9] S. Thrun, Y. Liu, D. Koller, A. Ng, Z. Ghahramani, and H. Durrant-White, "Simultaneous localization and mapping with sparse extended information filters," *Int. J. Robot. Res.*, vol. 23, no. 8, pp. 690–717, 2004.

[10] S. J. Julier and J. K. Uhlmann, "A counter example to the theory of simultaneous localization and map building," in *Proc. IEEE Int. Conf. Robot. Autom.*, May 2001, pp. 4238–4243.

[11] J. A. Castellanos, J. Neira, and J. D. Tardós, "Limits to the consistency of the EKF-based SLAM," presented at the 5th IFAC Symp. Intell. Auton. Veh., M. I. Ribeiro and J. Santos Victor, Eds. Jul. 2004, Lisbon, Portugal.

[12] F. Lu and E. Milios, "Globally consistant range scan alignment for environmental mapping," *Auton. Robots*, vol. 4, pp. 333–349, 1997.

[13] R. Unnikrishnan and A. Kelly, "A constrained optimization approach to globally consistent mapping," in *Proc. IEEE Int. Conf. Intell. Robots Syst.*, 2002, vol. 1, pp. 564–569.

[14] T. Duckett, S. Marsland, and J. Shapiro, "Learning globally consistent maps by relaxation," presented at the IEEE Int. Conf. Robot. Autom., San Francisco, CA, 2000.

[15] N. Tomatis, I. Nourbakhsh, and R. Siegwart, "Simultaneous localization and map building: A global topological model with local metric maps," in *Proc. IEEE Int. Conf. Intell. Robots Syst.*, 2001, vol. 1, pp. 421–426.

[16] A. Victorino and P. Rives, "Global consistency mapping with an hybrid representation," in *Proc. IEEE Int. Conf. Intell. Robots Syst.*, 2005, vol. 1, pp. 2554–2559.

[17] J. E. Guivant, E. M. Nebot, and S. Baker, "Localization and map building using laser range sensors in outdoor applications," *J. Robot. Syst.*, vol. 17, pp. 565–583, 2000.

[18] U. Frese, "Treemap: An $O(\log n)$ algorithm for indoor simultaneous localization and mapping," *Auton. Robots*, vol. 20, pp. 25–42, 2006.

[19] C. Estrada, J. Neira, and J. D. Tardós, "Hierarchical SLAM: Real-time accurate mapping of large environments," *IEEE Trans. Robot. Autom.*, vol. 8, no. 4, pp. 588–597, Aug. 2005.

[20] K. Konolige, "Large-scale map-making," presented at the Nat. Conf. AI (AAAI), San Jose, CA, 2004.

[21] S. Thrun, D. Fox, and W. Burgard, "A probalistic approach to concurrent mapping and localization for mobile robots," *Auton. Robots*, vol. 5, pp. 253–271, 1998.

[22] J. Neira and J. Tardós, "Data association in stocastic mapping using the joint compatibility test," *IEEE Trans. Robot. Autom.*, vol. 17, no. 6, pp. 890–897, Dec. 2001.

[23] J. Tardós, J. Neira, P. Newman, and J. Leonard, "Robust mapping and localization in indoor environments using sonar data," *Int. J. Robot. Res.*, vol. 21, no. 4, pp. 311–330, 2002.

[24] M. Bosse, J. Leonard, and S. Teller, "Simultaneous localization and map building in large-scale cyclic environments using the Atlas framework," *Int. J. Robot. Res.*, vol. 23, pp. 1113–1139, 2004.

[25] J. Folkesson, P. Jensfelt, and H. I. Christensen, "Vision SLAM in the measurement subspace," presented at the IEEE Int. Conf. Robot. Autom., Barcelona, Spain, 2005.

[26] J. Folkesson, P. Jensfelt, and H. I. Christensen, "Graphical SLAM using vision and the measurement subspace," presented at the IEEE/RSJ Int. Conf. Intell. Robots Syst., Edmonton, AB, Canada, 2005.

[27] J. Folkesson and H. I. Christensen, "Graphical SLAM—a self-correcting map," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2004, vol. 1, pp. 383–390.

[28] M. A. Paskin, "Thin junction tree filters for simultaneous localization and mapping," in *Proc. 18th Joint Conf. Artif. Intell.*, G. Gottlob and T. Walsh, Eds. San Francisco, CA: Morgan Kaufmann, 2003, pp. 1157–1164.

John Folkesson (S'02–M'05) received the B.S. degree in physics from Queens College, New York, in 1983. He received the M.S. and Ph.D. degrees in computer science from the Royal Institute of Technology (KTH), Stockholm, Sweden, in 2001 and 2005, respectively. He is currently a Postdoctoral Fellow at the Massachusetts Institute of Technology, Cambridge. His current research interests include autonomous robot systems.

Henrik I. Christensen (M'87) was born in Frederickshavn, Denmark, on July 16, 1962. He received the M.Sc. and Ph.D. degrees in electrical engineering from Aalborg University, Aalborg, Denmark, in 1987 and 1990, respectively.

He is currently the KUKA Chair of Robotics at the College of Computing, Georgia Institute of Technology, Atlanta, where he is also the Director of the Center for Robotics and Intelligent Machines. Dr. Christensen has earlier been associated with the Royal Institute of Technology (KTH), Stockholm, Sweden, Aalborg University, and Oak Ridge National Laboratory, Oak Ridge, TN. He was the Founding Coordinator of European Robotics Network Ž2013 EURON (1999–2006). He is a Consultant to agencies and companies across three continents. He is the author or coauthor of more than 230 papers published in international journals. His current research interests include systems integration, estimation, and human–robot interaction.

Dr. Christensen serves on the editorial boards of several journals including *Service Robotics* and *Autonomous Robots*.