Proceedings of the 2004 IEEE
International Conference on Robotics & Automation
New Orleans, LA • April 2004

Graphical SLAM - A Self-Correcting Map

John Folkesson and Henrik Christensen
Centre for Autonomous Systems, Royal Institute of Technology
Stockholm, Sweden,
johnf@nada.kth.se

Abstract— **In this paper we describe an approach to simultaneous localization and mapping, SLAM. This approach has the highly desirable property of robustness to data association errors. Another important advantage of our algorithm is that non-linearities are computed exactly, so that global constraints can be imposed even if they result in large shifts to the map. We represent the map as a graph and use the graph to find an efficient map update algorithm. We also show how topological consistency can be imposed on the map, such as, closing a loop. The algorithm has been implemented on an outdoor robot and we have experimental validation of our ideas. We also explain how the graph can be simplified leading to linear approximations of sections of the map. This reduction gives us a natural way to connect local map patches into a much larger global map.**

I. Introduction

If robots are to move in the real world they need to be able to keep track of their position. For this they need maps. We want robots to be able to learn these maps as they move about the environment. This problem is known as simultaneous mapping and localization, (SLAM). The SLAM problem is central to autonoumous mobile robotics, [1].

This paper presents a new way to look at the problem focusing on the issues that have caused the most trouble for other methods. We have tried to combine the best features of existing approaches in a way that allows a robust solution to the problem.

Our guiding principle is to retain all the important information and use it in the exact way with no approximations as long as possible. We can then introduce approximations later when we have a better idea of the true state.

We represent this exact information as a graph with the edges representing the measurements from our sensors. The updates and approximations can then be formulated as operations on the graph. Large sections of the graph can be approximated linearly in a local frame leading to a simplification that does not suffer from non-linear effects as the global map is distorted to impose topological constraints.

By using this very general representation we are able to apply any method to find a good solution and just as importantly, we have a measure, the energy as explained later, to test how good that solution is.

II. Background on the Slam Problem

A number of methods been proposed to do SLAM. The methods can be characterized as being batch or recursive, feature based or raw data based and topological or metric. Most methods have some probabilistic interpretation.

In many environments the sensors will give some readings that are not useful for localization. These could be range scans of a bush or hedge, people, cars, sloping surfaces and so on. By trying to extract good features from the raw data much of the noise can be ignored. Also maps with abstract features are sometimes more useful that maps that consist of raw data.

Both feature based and scan based methods suffer from data association errors. Incorrectly matching sensor readings leads to errors in the resulting maps. Once such an error has occurred, many of the existing methods have no way to correct or detect these errors. Others can do limited correction. Matching errors are handled best by expectation maximization, EM, methods, [2]. When applied in the pure form EM will find the best set of matches. In real-time implementations of EM this desirable property must be partially sacrificed as the number of combinations of matches is very large [3].

One reason that data association is such a problem is that the really hard problems are ones that look reasonable locally but lead to global failure. With real data, matching errors often do not lead to inconsistency until a loop is closed which can take thousands of iterations. Up to that point the match seems quite OK. This type of problem is very hard for any method to deal with. All of the methods can produce good locally consistent maps. Some can even correct locally inconsistent coorespondance errors.

Global consistency, the problem of closing large loops, is much harder. Many methods can not use the information that the robot has closed a loop to fix the map. Some methods are designed to use this type of information explicitly [4], [5], [6]. The idea of combining topological and metric information in a unified approach was shown in [7] to be a powerful one. Our method is similar to those methods and can also use the global consistency constraints to improve the map.

Another major difficulty inherent in the SLAM problem is the inconsistent linearizations that arise from observing the same features from different locations. The extended Kalman filter SLAM methods make a linearization of the observation about the current state. As the state evolves, the linearizations of the measurements will be about different points. This leads to inconsistancies in the resulting map after many iterations. Julier and Uhlmann have first explained the problem [8].

The method we propose does not suffer from this problem as we work with the full non-linear problem. The approach does suffer from local minimum. It is possible for our map to become trapped in a local minimum. Any method that doesn't explore the entire space of possible maps will make similar mistakes. This is a fundamental limitation of all methods.

III. The Real Problem of SLAM - Our Goal

What is it that causes SLAM maps to fail? Well a good answer is that there are just too many possible maps. The space of all possible interpretations of the measurements is of a very high dimension.

At the start there is one most likely solution and the less but significantly likely solutions are all 'close' to the best one. Thus, one can safely use the maximum likelihood solution as the solution. This is what the EKF method does.

Now as the robot moves to new areas, some solutions will begin to appear that are significantly likely but not very 'close' to the maximum likelihood solution. This is bound to happen unless the robot explores very tediously with lots of exploration of the areas already mapped and only occasionally expanding the map size.

The number of distinct regions of the state space with significant likelihood will grow as the robot explores the environment. When the robot comes back to areas already mapped the number of regions will decrease as a result of inconsistancies. If the maximum likelihood solution is in one of the regions that become inconsistent, it will no longer be the maximum likelihood solution and won't even be 'close' to a likely solution.

The EKF has no way to find and jump to another region of the state space. Therefore, it can only be part of a larger solution to the SLAM problem and not the ultimate solution. Multiple hypothesis solutions, [9], will suffer from the fact that over large loops the number of hypothesis that will be needed will be way too large.

One needs a single real-time solution for many applications but one would like to be able to impose the loop closing constraint on that solution. The ultimate solution will be continuously providing a map that is locally consistent and a pose estimate on that map. It will also be able to incorporate global consistence constraints as they become available to produce a more correct map and pose estimate. This is our goal.

IV. The Probabilistic Foundation

We have a robot moving through a series of poses $\{\mathbf{x}_i\}$ taking measurements of both the relative movement between poses and of the relative position of features. Here i runs from 1 to N_p. The movement measurements can be denoted by $\{d_i\}$ while the feature measurements will be denoted by $\{f_k\}$, where k runs from 1 to N_m. The set of feature coordinates will be denoted by $\{z_j\}$and the number of features by $N_f \leq N_m$. An association of feature measurements with features will be denoted by Λ_a. This association then indirectly specifies the number of features, N_f.

Now for each Λ_a and values for the poses and features we will have an explanation of the measurements. We will need to express the probability of a particular explanation. The best explanation would then be the one that gives the maximum probability, the maximum likelihood solution. If we denote the probability of an explanation as $P(x, z, d, f, \Lambda)$ we can write:

$$P(x, z, d, f, \Lambda) = P(d, f|x, z, \Lambda)P(x, z, \Lambda) \tag{1}$$

Now the term $P(x, z, \Lambda)$ is the apriori probability of a path, map and an association. We make an assumption here that this does not depend on the map or path and that it only depends on the number of features in the resulting map N_f. The idea is that the smaller the number of features the more apriori likely the association. So our first assumption is:

$$P(x, z, \Lambda) \propto P(\Lambda) = P(N_f) \propto e^{-\lambda N_f} \tag{2}$$

Here we have chosen a rather arbitrary exponential form for the probability which is justified by two facts. It leads to very simple expressions and it works well. We will also assume that the motion and feature measurements are independent.

$$P(x, z, d, f, \Lambda) \propto P(d|x)P(f|x, z, \Lambda)e^{-\lambda N_f} \tag{3}$$

Now we can define the 'energy', or entropy, of our system.

$$E(x, z, d, f, \Lambda) = -\log(P(d|x) - \log(P(f|x, z, \Lambda) + \lambda N_f \tag{4}$$

The last term can be rewritten as:

$$\lambda N_f = N_m - \sum_{j=1}^{N_f} \lambda(n_j - 1). \tag{5}$$

Where n_j is the number of measurements associated with feature j. In this form we have an energy reduction if we associate a new measurement with an existing feature as opposed to creating a new feature for that measurement.

$$E_\Lambda = -\sum_{j=1}^{N_f} \lambda(n_j - 1). \tag{6}$$

The parameter λ specifies how much we can distort the graph edges to make a match and still lower the energy. It will be important when making the data associations. If the energy after matching two measurements is more then before, the match is likely incorrect. This λ is similar to the threshold one normally places on the mahalanobis distance when matching when using a maximum likelihood estimator.

The movement term gives rise to an energy:

$$E_d = -\sum_{i=1}^{N_p} \log P(d_i|x_{i-1}, x_i) = \frac{1}{2}\sum_{i=1}^{N_p} \xi_i k_i \xi_i \tag{7}$$

Here we assume a independent Gaussian model and drop an additive constant. The k_i are the inverse covariance matrices of the dead reckoning estimates and

$$\xi_i = T(\mathbf{x}_i|\mathbf{x}_{i-1}) - \delta_i. \tag{8}$$

Here T denotes the non-linear transformation to the robot frame at $\mathbf{x}_{i-1}$. The term δ_i is the estimated motion in the robot frame.

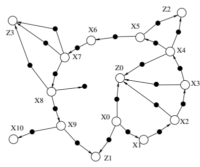

Fig. 1. The energy nodes are filled. The state nodes are not.

Making similar assumptions, the feature measurements give:

$$E_f = -\log(P(f|x,z,\Lambda) = \frac{1}{2}\sum_{k=1}^{N_m} \eta_k k_k \eta_k \tag{9}$$

$$\eta_k = f(T(\mathbf{z}_j|\mathbf{x}_i)) - \zeta_k. \tag{10}$$

Where $\mathbf{x}_i$ is the pose from which measurement k was made and $\mathbf{z}_j$ is coordinates of the feature associated with measurement k. f is a non-linear function describing the measurement. The assumptions about independence are essential but the distributions need not be Gaussian, only C_2 .

V. The Graph

We combine the results from the last section to find:

$$E = E_d(\mathbf{x}) + E_f(\mathbf{x},\mathbf{z}) + E_\Lambda(n_j). \tag{11}$$

It is this energy that we must minimize to find the maximum likelihood solution. The energy is a sum of terms each one relating a few poses and features to one another plus the last term which is the match energy giving us the benefit to matching a measurement to an existing feature. This structure leads us to a represent the map and poses as a graph. This will be easier to work with than matrices and long state vectors.

Our graph has two basic types of nodes, state nodes and energy nodes. The state nodes contain coordinates (the x and z), while the energy nodes contain the functionality and information needed to calculate one of the terms in the energy sum above. Thus, each energy node will have edges connecting it those state nodes that it needs to calculate the energy.

State nodes are composed of pose nodes, x's and feature nodes, z's. The energy nodes are move nodes and feature measurement nodes. So, a move node would have edges to two pose nodes and would store the k_i and δ_i needed to calculate one term of eq. (7).

So that eq. (11) leads to a graph with nodes organized as a chain of pose nodes connected by move nodes. The feature nodes will then be linked to the pose nodes as shown in fig. 1.

VI. Map Update

In general, finding the minimum of eq. (11) is not feasible due to the large dimensionality of the state vector. We can however try to start the state near the measurement values and use the derivatives of the energy drive the system towards lower energy states. If we start at well chosen states we can build up the graph by adding measurements one at a time and then relaxing the graph.

We will now describe how the graph is built up as the robot moves about. At the i^{th} step we will need to add the i^{th} pose node and possibly some new feature nodes to the graph. The nodes can be added at their equilibrium positions assuming the positions of the old nodes are fixed.

Having added the new nodes and edges we must calculate the change induced on the rest of the map. We have to find the minimum of eq. (11) with respect to the pose and feature positions. We expand eq. (11) about the current state out to the quadratic terms. The terms associated with a particular node, call it A, look like:

$$E_A = \sum_{i:edgesofA} [E_i(\bar{\mathbf{x}}_A,\bar{\mathbf{x}}_i) + \bigtriangledown E_i(\bar{\mathbf{x}}_A,\bar{\mathbf{x}}_i)\cdot \begin{pmatrix} \Delta\mathbf{x}_A \\ \Delta\mathbf{x}_i \end{pmatrix} +$$

$$(1/2)\begin{pmatrix} \Delta\mathbf{x}_A, & \Delta\mathbf{x}_i \end{pmatrix}\cdot \bigtriangledown^T \bigtriangledown E_i(\bar{\mathbf{x}}_A,\bar{\mathbf{x}}_i)\cdot \begin{pmatrix} \Delta\mathbf{x}_A \\ \Delta\mathbf{x}_i \end{pmatrix}] \tag{12}$$

Now we condense the notation to:

$$\mathcal{G} = \bigtriangledown E_A = \begin{pmatrix} \mathcal{G}_A \\ \mathcal{G}_i \end{pmatrix} \tag{13}$$

$$\mathcal{H} = \bigtriangledown^T \bigtriangledown E_A = \begin{pmatrix} \mathcal{H}_{AA} & \mathcal{H}_{Ai} \\ \mathcal{H}_{iA} & \mathcal{H}_{ii} \end{pmatrix} \tag{14}$$

Where $\Delta\mathbf{x}_A = \mathbf{x}_A - \bar{\mathbf{x}}_A$, etc. Here we just call everything x and make no distinction between pose and feature nodes. the subscript i indicates all the coordinates that are not node A but share an edge with A. An important number for us is the prediction of the energy gain from moving a node to the minimum position holding all other nodes fixed. This can easily be calculated. First the new node position would be:

$$(\mathbf{x}_A - \bar{\mathbf{x}}_A) = -\mathcal{H}_{AA}^{-1}\cdot\mathcal{G}_A \tag{15}$$

Then the change in energy would be:

$$\triangle E_A = -(1/2)\mathcal{G}_A\cdot\mathcal{H}_{AA}^{-1}\cdot\mathcal{G}_A \tag{16}$$

This $\triangle E_A$ will be used to make decisions as to what optimization method to use and whether an update is needed.

The simplest and slowest method is to use steepest decent. We use this as a last resort when near a saddle point. This is the situation that eq. (16) gives $\triangle E_A > 0$. In that case, we need to move the node away from the positively curved region. We move in the direction of the gradient a small step and then recalculate and repeat until no significant change occurs. The step size is increased if the change in energy seems to indicate a flat region, (ie. the change is given by the gradient times the change in x), and decreased if the curvature is too high.

The next simplest method is to use eq. (15) directly. This will move the node to the bottom of the energy surface but the higher order terms in the energy might require us to iterate this a number of times until there is no significant change in the energy. We will refer to the use of these two methods, (steepest descents and eq. (15)), for per node minimization, as relaxing the node.

Having implemented only this much we found that the maps were already looking better than our previous SLAM maps,[10]. The updates were rather slow however. One must relax the new node then all the adjacent nodes then if any of those changed one must relax any adjacent to them and so on. This causes the update to move back and forth between nodes a lot when what is needed is to move a set of nodes simultaneously. We were able to significantly speed up the procedure by doing what we call chained updates on the pose nodes.

For that we exploit the special nature of our problem. We have a chain of pose nodes connected by the dead-reckoning edges. Let us consider pose node A with the previous pose node labeled B. We will consider the features and the next pose node as fixed while nodes A and B are variable. In this subspace, we can always eliminate a pose node A's coordinates in terms of B's by making the substitution:

$$(\mathbf{x}_A - \bar{\mathbf{x}}_A) = -\mathcal{H}_{AA}^{-1} \cdot [\mathcal{H}_{AB} \cdot (\mathbf{x}_B - \bar{\mathbf{x}}_B) + \mathcal{G}_A^T] \tag{17}$$

Making this substitution we find the edge AB now contributes some extra terms to B's gradient and Hessian at the current state.

$$\Delta\bar{\mathcal{G}}_B = -\mathcal{H}_{BA} \cdot \mathcal{H}_{AA}^{-1} \cdot \mathcal{G}_A^T \tag{18}$$

$$\Delta\bar{\mathcal{H}}_{BB} = -\mathcal{H}_{BA} \cdot \mathcal{H}_{AA}^{-1} \cdot \mathcal{H}_{AB} \tag{19}$$

We now move to B and repeat the procedure using the gradient and Hessian with the extra terms added to them. First test if B needs an update, (ie. does eq. (16) gives $\triangle E_B <$ some threshold). Then, if an update is needed, eliminate B in terms of the previous pose, C. Then move to C.

When we get to a node that doesn't need an update we can then use its coordinate values to update its next node, then the next node's coordinates to update its next node, and so on until we get back to node A.

We have essentially inverted the Hessian matrix for the chain of pose nodes assuming the feature nodes to be fixed. This is easy to do via operations on the pose nodes themselves as each pose node is attached to only two other pose nodes.

After doing this we must move to the feature nodes attached to the updated pose nodes and check if they need updating. This repeats until no updates are needed. In general the number of nodes needing updates only depends on the size of the perturbation caused by the measurement. Most measurements only cause a few nodes to change. This is why in general the method scales well to large maps, (constant time wrt. N). Also we note that we undo any updates that turn out to cause the energy to increase, thus we always move to lower energy.

VII. Feature Matching

We try to build the map up incrementally in a way that the energy in each new feature measurement does not exceed λ. We do this by first adding the feature measurement, then calculating the new equilibrium position. One can then compare the total energy before and after adding the measurement. Thus, the data association problem is reduced to checking the change in energy. We need not be very careful when adding measurements to our graph. We use a relatively loose matching requirement and rely on this check of the energy to uncover any mistakes after recalculating the equilibrium. This formulation is similar in some ways to expectation maximization, EM, methods, [2] and shares those methods robustness with respect to matching errors.

The energy of an energy node is the sum of squares of independent normal variables. It is thus a χ^2 variable. One can then check the energy of individual energy nodes at any time to see if the associations still make sense.

We should mention here one of the major advantages of this representation of the map. We can at any time and with little effort go in and make changes to the graph. We can for example merge features by simply moving the edges from one feature node to the other and then discarding the node. We can check the energy before and after the merge to see if they really should be merged. We can add information to existing edges. We can also remove edges if we find that they no longer make sense. And, of course, the initialization of the features is trivial. All these things are very tricky or impossible with a Kalman filter and thus our approach is both easier to implement and able to use more information.

VIII. Closing the Loop

By closing the loop we are, of course, interested in a loop that did not close by applying the above update rules. This situation presents two separate problems. First, how does the robot realize that there is a problem. Second how can the map be changed to close the loop. We will show a way to solve the second problem. We pause the SLAM program and check the map. Then supply the program with a list of pairs of features that are to be matched, (ie. one wall from the beginning of the loop and one from the end). The map will then be recalculated with the new constraints added.

If one first merges the features then recalculates the poses the result tends to bend to fit the map on the pose update and when the features are updated the gradients are too small to move them much. The initial brute force way we chose to get around this is to ignore most of the map and instead use only the features from the loop closing constraint and the robot path to adjust the existing graph. We then re-attach the feature edges to our graph one at a time. This worked but there is a much better way to do this as we will soon describe.

IX. Reducing the Graph - Star Nodes

An optimization would be to simplify the graph by combining nodes. For a linear system one can reduce the graph with an exact formula. One simply chooses a state node that one wants to remove, call it node A. We let B represent the whole space of neighboring state nodes. One then ends up with a sub-system with an energy that is a quadratic function of $\mathbf{x}_A$ and $\mathbf{x}_B$. Here we have simply called everything $\mathbf{x}$ with

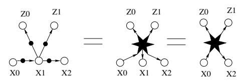

Fig. 2. Here we show how a node can be eliminated by creating edges between all its neighbors. $\mathbf{x}_0$ becomes the relative frame of the star node.

no distinction between features and poses. The energy of this sub-system can be represented as a single energy node, which we call a star node, with edges radiating out to all these nodes. So we combine all the energy nodes into star node and use Gaussian elimination to remove the variables of node A.

In our case we have a nonlinear system. We must therefore expand the equations around the current equilibrium point. We chose to only eliminate pose nodes. Thus, we end up with a solution for the pose node's coordinates in terms of all the attached state coordinates. We make the Taylor expansion of the terms in eq. (11) that come from node A . The A part of the gradient is zero, due to being at equilibrium,

$$E_A = \sum_{j:edgesofA} [E_i(\bar{\mathbf{x}}_A, \bar{\mathbf{x}}_j) + \mathcal{G}_j \cdot (\mathbf{x}_j - \bar{\mathbf{x}}_j) + \frac{1}{2} \cdot$$

$$\cdot \left(\begin{array}{cc} \mathbf{x}_A - \bar{\mathbf{x}}_A, & \mathbf{x}_j - \bar{\mathbf{x}}_j \end{array} \right) \cdot \mathcal{H}_j(\bar{\mathbf{x}}_A, \bar{\mathbf{x}}_j) \cdot \left(\begin{array}{c} \mathbf{x}_A - \bar{\mathbf{x}}_A \\ \mathbf{x}_j - \bar{\mathbf{x}}_j \end{array} \right)] \quad (20)$$

We can now solve for $(\mathbf{x}_A - \bar{\mathbf{x}}_A)$ in terms of the $(\mathbf{x}_B - \bar{\mathbf{x}}_B)$.

$$(\mathbf{x}_A - \bar{\mathbf{x}}_A) = -\mathcal{H}_{AA}^{-1} \cdot \mathcal{H}_{AB} \cdot (\mathbf{x}_B - \bar{\mathbf{x}}_B) \quad (21)$$

Is the optimal solution. We can then plug this into eq. (20) to get the new energy equation with A eliminated,

$$E^* = E_A = \sum_{j:edgesofA} \{E_j(\bar{\mathbf{x}}_A, \bar{\mathbf{x}}_j) + \mathcal{G}_j \cdot (\mathbf{x}_j - \bar{\mathbf{x}}_j) +$$

$$\frac{1}{2} \cdot (\mathbf{x}_j - \bar{\mathbf{x}}_j) \cdot \mathcal{H}_{jj} \cdot (\mathbf{x}_j - \bar{\mathbf{x}}_j) -$$

$$\sum_{i:edgesofA} (\mathbf{x}_i - \bar{\mathbf{x}}_i) \cdot \mathcal{H}_{iA} \cdot \mathcal{H}_{AA}^{-1} \cdot \mathcal{H}_{Aj} \cdot (\mathbf{x}_j - \bar{\mathbf{x}}_j)\} \quad (22)$$

We see that the new Hessian has terms connecting nodes i and j that had no connection before the reduction. This new energy equation is then stored in the star node. We can start by eliminating $\mathbf{x}_1$, fig. 2. We then end up with a star node with edges to $\mathbf{x}_0$, $\mathbf{x}_2$ and $\mathbf{z}_i$ for the features, i, connected to $\mathbf{x}_1$. We can then continue by eliminating $\mathbf{x}_2$ and so on. If we continue all the way to the current pose we end up with a fully connected graph with only one pose variable. This is similar to what happens in the extended Kalman filter.

However, we should not carry the reduction though the whole set of poses. The resulting system would be too hard to update. That is because it is no longer sparsely connected. Therefore, we stop at some point and start a new star node.

If we first chose to eliminate every other pose node, we can then merge two star nodes separated by a pose node into a larger star node by eliminating the pose node between them. One continues in this way forming a tree until there is only one star node representing the interactions of a section of the graph. The advantage of forming a tree rather than just a long chain is numerical stability as we end up with a sum of terms with a small number of multiplications in each term rather than multiplying a single matrix many times. The depth of the tree gives a simple stopping criteria so we can for instance go to a depth of 7 (127 reductions).

We need not do the combining for the most recent poses. One can wait, say, 50 poses before starting to combine. So as each new pose is added to the graph we try to eliminate the pose 50 steps earlier. The advantage is that the linearization will be done about a better point and the match energy will have been observed for some time. This can be compared to the EKF where these steps are done at once.

Making Star Nodes Invariant

The star node calculates the energy using a Hessian matrix as described above. This Hessian should have symmetries due to the nature of the measurements that it represents. These would be expressed by zero eigenvalues of the matrix.

One symmetry is that of translation and rotation of all the state coordinates. Another would be sliding the end-points of walls perpendicular to the wall normal vector. That is, in the case that the end-points were not directly measured, a fairly common situation. By defining natural coordinates for the star node we can eliminate these zero eigenvalues and represent the energy in a lower dimensional space as eigenvectors and eigenvalues > 0. This makes the star node explicitly stable and invariant, two very nice properties.

We define coordinates $\mathbf{q}$ as the state coordinates, $\mathbf{x}$, transformed to the frame of one of the pose nodes, $\mathbf{x}_0$ and then project out the part normal to the walls.

$$\mathbf{q}_i = P \cdot T(\mathbf{x}_i | \mathbf{x}_0). \quad (23)$$

We illustrate the simple case for $P = I$, two pose nodes and the rest point nodes. We can show how to transform the Hessian to the q coordinates.

$$\mathbf{q} = T(\mathbf{x}|\mathbf{x}_0) = W \cdot (\mathbf{x}). \quad (24)$$

$$W = W_0 + \widetilde{W} \quad (25)$$

$$W_0 = \left(\begin{array}{cccc} -\mathcal{R} & 0 & 0 & \cdots \\ 0 & -1 & 0 & \cdots \\ -\mathcal{R} & 0 & 0 & \cdots \\ -\mathcal{R} & 0 & 0 & \cdots \\ \vdots & \vdots & \vdots & \vdots \end{array} \right) \quad (26)$$

$$\widetilde{W} = \left(\begin{array}{cccccccc} 0 & 0 & 0 & \mathcal{R} & 0 & 0 & 0 & \cdots \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & \cdots \\ 0 & 0 & 0 & 0 & 0 & \mathcal{R} & 0 & \cdots \\ 0 & 0 & 0 & 0 & 0 & 0 & \mathcal{R} & \cdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \end{array} \right) \quad (27)$$

$$\mathcal{R} = \begin{pmatrix} cos(\theta_0) & sin(\theta_0) \\ -sin(\theta_0) & cos(\theta_0) \end{pmatrix} \tag{28}$$

Now the Jacobian for this coordinate change looks almost like W.

$$J = J_0 + \widetilde{J} + J_2 \tag{29}$$

$$J_2 = \begin{pmatrix} q_1 \\ -q_0 \\ 0 \\ q_3 \\ -q_4 \\ \vdots \end{pmatrix} \cdot \begin{pmatrix} 0 & 0 & 1 & 0 & 0 & \cdots \end{pmatrix} \tag{30}$$

Where $J_0 = W_0$ and $\widetilde{J} = \widetilde{W}$. Of course q has 3 less dimensions than $\mathbf{x}$. In general:

$$\mathcal{H}_{xx} = J^T \cdot \mathcal{H}_{qq} \cdot J + \frac{\partial^2 \mathbf{q}}{\partial x \partial x} \cdot \mathcal{G}_q \approx J^T \cdot \mathcal{H}_{qq} \cdot J. \tag{31}$$

Where we use the fact that the $\mathcal{G}_q$ term is typically much smaller than the other terms and can be dropped, (ie the state is near the equilibrium point of the star node when we want to use this formula). By working backwards from eq. (31) and noticing that $\widetilde{J} \cdot J^T = I$, we see that,

$$\mathcal{H}_{qq} = \widetilde{J} \cdot \mathcal{H}_{xx} \cdot \widetilde{J}^T. \tag{32}$$

We had more symmetries to deal with, $P \neq I$, so that the dimension of $\mathbf{q}$ was typically nearly half that of $\mathbf{x}$. These projections were done using constant P, so $J = P$ for them. This is an approximation that the wall normals in the star frame don't change significantly.

We now have an explicitly invariant representation that can be reduced using principle component analysis to a set of eigenvalues and eigenvectors. By adjusting the linearization point in the $\mathbf{q}$ space we can eliminate the gradient terms.

$$E^* = E^*(\bar{\mathbf{q}}) + \frac{1}{2}\sum_j [b_j^*(\mathbf{V}_j^* \cdot \Delta\mathbf{q})^2] \tag{33}$$

Here q are the 'natural coordinates', b_j^* are the eigenvalues and V_j^* are the eigenvectors. The q are expanded around the equilibrium point, $\bar{q}$. In this form the star nodes have some very nice properties.

Nice Star Properties

Star nodes can be re-centered. When the the base pose node, $\mathbf{x}_0$, has moved significantly, one can make the q to x transformation by inverting the above formulas to express the energy as a quadratic form in the world coordinates about the current, $\bar{\mathbf{x}} = \mathbf{x}$. One uses this until the base node has again moved significantly. This makes updates very fast. Re-centering removes most of the non-linear effects of large changes.

Star nodes can have information added to them. One starts by re-centering and then adds the Hessian, gradient and constant pieces of new information, possibly adding edges to new state nodes. Then do the x to q transformation to get the invariant form with the new information.

Star nodes can be used to eliminate state nodes. Any state node that is only connected to one star node can be removed by setting its part of $\mathbf{\Delta q} = 0$ in eq. (33).

Star nodes represent a kind of local map. They contain information on the features attached to them that is independent to the information in the other star nodes. The relationships between star nodes are represented by edges to common state nodes. These are 'correct' relationships based on the likelihood function. The $\bar{\mathbf{q}}$ represent the star node equilibrium position for the attached state nodes. The star nodes look much like the local sub-maps of the 'Atlas Framework' [7].

Star nodes can be used to organize the global calculations efficiently. One can ignore the features completely to get an approximate solution by using $\mathbf{\Delta q} = 0$ for them when calculating the energy and its derivatives. This is equivalent to treating the features attached to each star node as being different from the features attached to other nodes. By doing this the pose-pose forces from a star node will reflect the feature measurements, (ie. the links will be stiffer than dead reckoning) but the graph will be able to adjust to stresses more quickly due to the simplified structure. This leads to objects that look like the strong links of [4]. One can then put back in the constraint that the stars have features in common to refine the solution.

The ideas here result in a representation of the map very similar to the sparse extended information filter [11]. The difference here is that we do not linearize once at the time of the observation and lock the approximation in that frame. Instead we linearize a section of the graph in a relative frame and at a time when it is more mature. This avoids most of the non-linear effects that would otherwise result from the local frame being rotated in the world frame. Another difference is that Lui and Thurn 'throw away' some information in order to prevent a fully connected system. We instead leave some pose nodes around which achieve the same end without loss of information.

Closing Simple Loops

We can use the $\mathbf{\Delta}q$'s of eq. (33) and the ideas above to set constraints around a loop. We start by adding a Lagrange multiplier to our cost function,

$$C(\Lambda, \Delta q) = \Lambda \cdot (\sum_i \Delta\mathbf{x}_i - \mathbf{d}_c) + \frac{1}{2}\sum_i \sum_j {}_i\mathbf{b}_j^*({}_i\mathbf{V}_j^* \cdot \Delta\mathbf{q})^2 \tag{34}$$

Where i is the index of the star nodes around the loop, Δx_i is the difference between the two poses attached to star node i and $\mathbf{d}_c$ is the constraint on the change in pose. Now we use,

$$\Delta\mathbf{x}_i = \begin{pmatrix} \mathcal{R}_i & 0 \\ 0 & 1 \end{pmatrix} \cdot (\Delta\mathbf{q}_i + \bar{\mathbf{q}}_i). \tag{35}$$

This is the way we defined the natural coordinates in terms of relative poses. The $\mathbf{q}_i$ above refers to the first 3 components of the q vector for the i^{th} star node. The above equation assumes that the q's have the right sense around the loop, this is not important (possible minus signs). Also all the feature components of $\Delta\mathbf{q}$ are set to zero for now as explained above.

In general the rotation matrices, $\mathcal{R}_i$, make the minimization of the above cost function non-linear. If we assume that the rotation matrices are constant (at the current values), we can solve this easily,

$$\Delta\mathbf{q}_i = -\sum_j \frac{{}_i\widetilde{\mathbf{V}}_j^{*T}\,{}_i\widetilde{\mathbf{V}}_j^{*}}{{}_i\mathrm{b}_j^{*}} \begin{pmatrix} \mathcal{R}_i^T & 0 \\ 0 & 1 \end{pmatrix} \cdot \mathbf{\Lambda}^T. \qquad (36)$$

Where ${}_i\widetilde{\mathbf{V}}_j^{*T}$ are the first 3 components of the the j^{th} eigenvector for the i^{th} star node.

$$\Lambda^T = S^{-1}\{[\sum_i \begin{pmatrix} \mathcal{R}_i & 0 \\ 0 & 1 \end{pmatrix} \cdot \bar{\mathbf{q}}_i] - \mathbf{d}_c\}. \qquad (37)$$

$$S = \sum_i \{ \begin{pmatrix} \mathcal{R}_i & 0 \\ 0 & 1 \end{pmatrix} \cdot [\sum_j \frac{{}_i\widetilde{\mathbf{V}}_j^{*T}\,{}_i\widetilde{\mathbf{V}}_j^{*}}{{}_i\mathrm{b}_j^{*}}] \cdot \begin{pmatrix} \mathcal{R}_i^T & 0 \\ 0 & 1 \end{pmatrix} \}. \quad (38)$$

By using these formula a number of times, (3 in practice), each time with the new rotation terms, one can solve the nonlinear system. Then it is just to turn on the feature matches between star nodes to get the exact solution.

This procedure is quite fast taking a few seconds even for large loops with significant closing errors. The matrix operations are all 3x3. The Lagrange multiplier equation takes order N time,where N is the number of star nodes around the loop. The fine tuning after turning on the features takes the most time and depends on the stopping criteria.

X. Experiments

We have implemented our method on an ATRV2 outdoor robot equipped with a cross-bow D-FOG 6-axis inertial sensor, which we use to adjust the odometry and to obtain the pitch and roll angles of the robot. To measure the walls we use a SICK laser scanner. We measure the angle, perpendicular distance and if possible the angle of the endpoints of the walls. The dead-reckoning data fusion and the feature detection are described in an earlier paper, [10]. The output of the feature detection module is simply fed to the grapher.

The representation for walls is two endpoints each with (x,y) values. The pose is (x,y,z,θ, ϕ, ψ). The measurement nodes do not always constrain all the dimensions. Thus, the z, ϕ and ψ values are, for example, simple fixed at the values provided by the dead-reckoning module.

In figure 3 we show an example map. This map has 7,500 pose nodes and 11,975 pose-wall measurements. The average times per iteration for graph update do not include the time for feature extraction, but does include matching the found features to the map as well as searching for features that the map says should be visible but the feature extractor did not find. This was running on a laptop with a 550 Mhz Pentium III. The pose nodes are so tightly packed as to appear as a continuous line. One can notice that there is a large error at one point of the path to the left side of the building. This was due to the robot sliding down an incline while rotating.

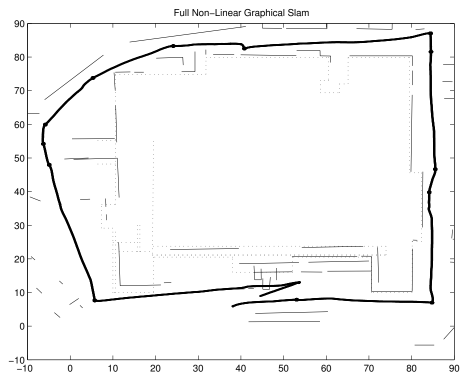

Fig. 3. This is the resulting map and path generated by the graphical SLAM method. The average time per iteration was 30 msec. The doted lines show the actual building outline. The path was traversed counterclockwise. No approximations were made. No attempt is made to match wall that are 'far' from the current node, as measured by the number of graph edges. Thus, the walls from the begining of the loop appear as new walls at the end.

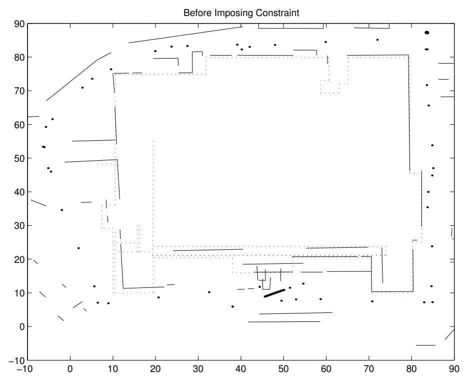

Fig. 4. This map shows the result of reducing the graph by introducing star nodes. The pose nodes that remain are shown as dots about 10 m apart. The 50 pose nodes at the end of the path, where no reduction has been done, remain. Calculation time was less than half that for the full non-linear case.

We then tried out the graph reduction ideas, shown in figure 4. We calculate exactly as previously but then after each update we reduce the pose nodes 50 nodes back from the current pose node. We form a star node tree as described in section IX, stopping at level 7, (127 pose nodes per star).

This method was only a little slower than our compressed Kalman filter but the time for a particular iteration can be longer than the average. It is therefore necessary to collect the data in a queue with separate process and then have the SLAM program read the data from the top of the queue and process it. We use a separate feature extractor/tracker to maintain this queue in our real-time implementation.

The map can be compared with the maps made using our compressed Kalman filter, the errors on the Kalman filter map are typically larger. Also the CEKF could not fix the map as we do next.

The result of our loop closing procedure are shown in

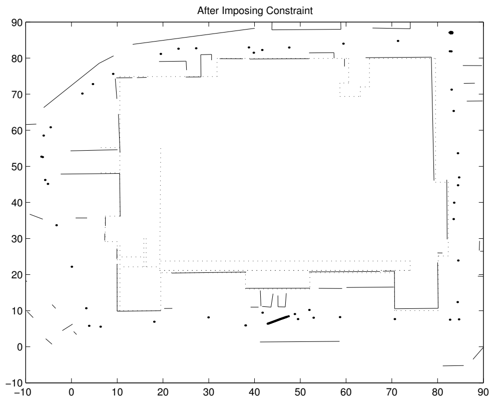

Fig. 5. The map was forced to close using the Lagrange multiplier method.

figure 5. One can see that most of the walls moved closer to their true positions. This map is globally consistent. The loop closing calculation took less than 2 sec.

XI. Discussion

We believe that the results of the experiments confirm the validity of our approach. We can make good quality maps in this straight forward way. The graphs contain information and computational machinery that allows us to do things like imposing global constraints on the map.

Our goal in this paper is to introduce the concept of graphical SLAM and to draw attention to the aspects of the SLAM problem that are the most difficult. Those aspects are characterized by the appearance of inconsistancies in the map. These might be bad data associations or loops that don't close. A solution to the SLAM problem must be able to survive and profit from these inconsistancies when they are discovered.

The representation of the map as a graph allows us to try out different methods to solve inconsistency problems. The graph leads directly to program structures that are easy to understand and work with. It also represents the information compactly. One can say that the edges represent the non-zero components of the information matrix.

We can compare the graphical representation of the map to the Kalman Filter type representation. The Kalman filter represents the probability density of the map as a Gaussian. Although we did assume a Gaussian for the measurements this did not result in a Gaussian for the state space. Also the assumption of Gaussian measurements was only an illustrative example, not essential. So that the graph can represent much more complex probability surfaces.

Conclusions

We have presented an approach to solving the SLAM problem which avoids its three most troublesome stumbling blocks. The non-linear effects are eliminated by postponing the linearization and linearizing in a local frame. The data association errors are detected and corrected automatically. And finally large loops can be closed in a way that is consistent with all the information collected.

We have outlined an efficient implementation that is running the algorithm in on an outdoor robot. The time for a map update is independent of map size and only depends on how well the measurements agree with previous information. Experimental results have also been shown to confirm that the algorithm does produce good maps. A comparison to a Kalman filter SLAM implementation on the same platform shows that the maps are in many ways superior.

Our method, while not being very subtle, does solve the problem in the most straight forward way possible. We have shown that this intuitive approach is robust and easy to implement. The main advantage of our formulation is the fact that all information is stored and accessible in the graph. This allows one to try different methods to improve the map. The energy in the graph gives a useful measure of goodness for the map and allows one to compare two solutions. The graph gives a representation of a general map probability distribution and is not limited to 'Gaussian maps'.

The reduction of the graph using star nodes which contain all the symmetries of the original measurements is a powerful way to correctly impose topological constraints on the map. The beauty being the simplicity and tractability of the resulting equations. Imposing constraints is reduce to solving an easy Lagrange multiplier equation and then fine tuning using all the information.

References

[1] M. G. Dissanayake, P. Newman, S. Clark, H. Durrant-Whyte, and M. Corba, "A solution to the simultaneous localization and map building (slam) problem," *IEEE Transactions on Robotics and Automation*, vol. 17, no. 3, pp. 229–241, June 2001.

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

[3] C. Martin and S. Thurn, "Real-time acquistion of compact volumetric 3d maps with mobile robots," in *Proc. of the IEEE International Conference on Robotics and Automation (ICRA02)*, 2002, pp. 311–316.

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

[5] J. Gutmann and K. Konolige, "Incremental mapping of large cyclic environments," in *Proc. of the 1999 IEEE International Symposium on Computational Intelligence in Robotics and Automation*, vol. 1, 1999, pp. 318–325.

[6] M. Golfarelli, D. Maio, and S. Rizzi, "Elastic correction of dead-reckoning errors in map building," in *Proc. of the IEEE International Conference on Intelligent Robots and Systems*, vol. 1, 1998, pp. 905–911.

[7] M. Bosse, P. Newman, and J. L. et al., "An atlas framework for scalable mapping," in *Proc. of the IEEE International Conference on Robotics and Automation (ICRA03)*, vol. 1, 2003, pp. 1899–1906.

[8] S. J. Julier and J. K. Uhlmann, "A counter examplee to the theory of simultaneous localization and map building," in *Proc. of the IEEE International Conference on Robotics and Automation*, May 2001, pp. 4238–4243.

[9] M. Montemerlo and S. Thurn, "Simultaneous localization and mapping with unknown data association using fastslam," in *Proc. of the IEEE International Conference on Robotics and Automation (ICRA03)*, vol. 1, 2003, pp. 1985–1991.

[10] J. Folkesson and H. I. Christensen, "Outdoor exploration and slam using a compressed filter," in *Proc. of the IEEE International Conference on Robotics and Automation (ICRA03)*, vol. 1, 2003, pp. 419–427.

[11] Y. Lui and S. Thurn, "Results fo outdoor-slam using sparse extended information filters," in *Proc. of the IEEE International Conference on Robotics and Automation (ICRA03)*, vol. 1, 2003, pp. 1227–1233.