A Tree Parameterization for Efficiently Computing Maximum Likelihood Maps using Gradient Descent

Giorgio Grisetti Cyrill Stachniss Slawomir Grzonka Wolfram Burgard
University of Freiburg, Department of Computer Science, 79110 Freiburg, Germany

***Abstract*— In 2006, Olson *et al.* presented a novel approach to address the graph-based simultaneous localization and mapping problem by applying stochastic gradient descent to minimize the error introduced by constraints. Together with multi-level relaxation, this is one of the most robust and efficient maximum likelihood techniques published so far. In this paper, we present an extension of Olson's algorithm. It applies a novel parameterization of the nodes in the graph that significantly improves the performance and enables us to cope with arbitrary network topologies. The latter allows us to bound the complexity of the algorithm to the size of the mapped area and not to the length of the trajectory as it is the case with both previous approaches. We implemented our technique and compared it to multi-level relaxation and Olson's algorithm. As we demonstrate in simulated and in real world experiments, our approach converges faster than the other approaches and yields accurate maps of the environment.**

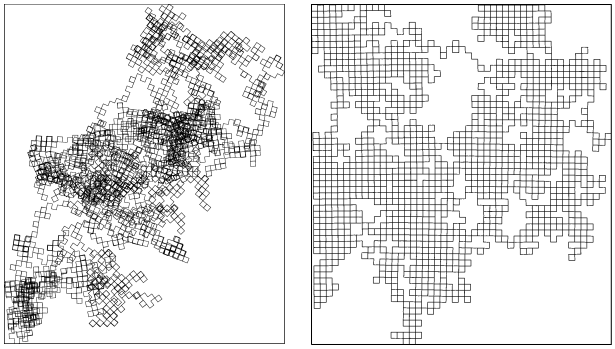

Fig. 1. The left image shows an uncorrected network with around 100k poses and 450k constraints. The right image depicts the network after applying our error minimization approach (100 iterations, 17s on a P4 CPU with 1.8GHz).

I. Introduction

Models of the environment are needed for a wide range of robotic applications, including search and rescue, automated vacuum cleaning, and many others. Learning maps has therefore been a major research focus in the robotics community over the last decades. Learning maps under uncertainty is often referred to as the simultaneous localization and mapping (SLAM) problem. In the literature, a large variety of solutions to this problem can be found. The approaches mainly differ due to the underlying estimation technique such as extended Kalman filters, information filters, particle filters, or least-square error minimization techniques.

In this paper, we consider the so-called "graph-based" or "network-based" formulation of the SLAM problem in which the poses of the robot are modeled by nodes in a graph [2, 5, 6, 7, 11, 13]. Constraints between poses resulting from observations or from odometry are encoded in the edges between the nodes.

The goal of an algorithm designed to solve this problem is to find the configuration of the nodes that maximizes the observation likelihood encoded in the constraints. Often, one refers to the negative observation likelihood as the error or the energy in the network. An alternative view to the problem is given by the spring-mass model in physics. In this view, the nodes are regarded as masses and the constraints as springs connected to the masses. The minimal energy configuration of the springs and masses describes a solution to the mapping problem. Figure 1 depicts such a constraint network as a motivating example.

A popular solution to this class of problems are iterative approaches. They can be used to either correct all poses simultaneously [6, 9, 11] or to locally update parts of the network [2, 5, 7, 13]. Depending on the used technique, different parts of the network are updated in each iteration. The strategy for defining and performing these local updates has a significant impact on the convergence speed.

Our approach uses a tree structure to define and efficiently update local regions in each iteration. The poses of the individual nodes are represented in an incremental fashion which allows the algorithm to automatically update successor nodes. Our approach extends Olson's algorithm [13] and converges significantly faster to a network configuration with a low error. Additionally, we are able to bound the complexity to the size of the environment and not to the length of the trajectory.

The remainder of this paper is organized as follows. After discussing the related work, Section III explains the graph-based formulation of the mapping problem. Subsequently, we explain the usage of stochastic gradient descent to find network configurations with small errors. Section V introduces our tree parameterization and in Section VI we explain how to obtain such a parameterization tree from robot data. We finally present our experimental results in Section VII.

II. Related Work

Mapping techniques for mobile robots can be classified according to the underlying estimation technique. The most popular approaches are extended Kalman filters (EKFs), sparse extended information filters, particle filters, and least square error minimization approaches. The effectiveness of the EKF approaches comes from the fact that they estimate a fully correlated posterior about landmark maps and robot poses [10, 14]. Their weakness lies in the strong assumptions that have to be made on both, the robot motion model and the sensor noise. Moreover, the landmarks are assumed to be uniquely

identifiable. There exist techniques [12] to deal with unknown data association in the SLAM context, however, if certain assumptions are violated the filter is likely to diverge [8].

Frese's TreeMap algorithm [4] can be applied to compute nonlinear map estimates. It relies on a strong topological assumption on the map to perform sparsification of the information matrix. This approximation ignores small entries in the information matrix. In this way, Frese is able to perform an update in $\mathcal{O}(\log n)$ where n is the number of features.

An alternative approach is to find maximum likelihood maps by least square error minimization. The idea is to compute a network of relations given the sequence of sensor readings. These relations represent the spatial constraints between the poses of the robot. In this paper, we also follow this way of formulating the SLAM problem. Lu and Milios [11] first applied this approach in robotics to address the SLAM problem using a kind of brute force method. Their approach seeks to optimize the whole network at once. Gutmann and Konolige [6] proposed an effective way for constructing such a network and for detecting loop closures while running an incremental estimation algorithm. Howard *et al.* [7] apply relaxation to localize the robot and build a map. Duckett *et al.* [2] propose the usage of Gauss-Seidel relaxation to minimize the error in the network of constraints. In order to make the problem linear, they assume knowledge about the orientation of the robot. Frese *et al.* [5] propose a variant of Gauss-Seidel relaxation called multi-level relaxation (MLR). It applies relaxation at different resolutions. MLR is reported to provide very good results and is probably the best relaxation technique in the SLAM context at the moment.

Note that such maximum likelihood techniques as well as our method focus on computing the best map and assume that the data association is given. The ATLAS framework [1] or hierarchical SLAM [3], for example, can be used to obtain such data associations (constraints). They also apply a global optimization procedure to compute a consistent map. One can replace such optimization procedures by our algorithm and in this way make ATLAS or hierarchical SLAM more efficient.

The approach closest to the work presented here is the work of Olson *et al.* [13]. They apply stochastic gradient descent to reduce the error in the network. They also propose a representation of the nodes which enables the algorithm to perform efficient updates. The approach of Olson *et al.* is one of the current state-of-the-art approaches for optimizing networks of constraints. In contrast to their technique, our approach uses a different parameterization of the nodes in the network that better takes into account the topology of the environment. This results in a faster convergence of our algorithm.

Highly sophisticated optimization techniques such as MLR or Olson's algorithm are restricted to networks that are built in an incremental way. They require as input a sequence of robot poses according to the traveled path. First, this makes it difficult to use these techniques in the context of multi-robot SLAM. Second, the complexity of the algorithm depends on the length of the trajectory traveled by the robot and not on the size of the environment. This dependency prevents to use these approaches in the context of lifelong map learning.

One motivation of our approach is to build a system that depends on the size of the environment and not explicitely on the length of the trajectory. We designed our approach in a way that it can be applied to arbitrary networks. As we will show in the remainder of this paper, the ability to use arbitrary networks allows us to prune the trajectory so that the complexity of our approach depends only on the size of the environment. Furthermore, our approach proposes a more efficient parameterization of the network when applying gradient descent.

III. On Graph-based SLAM

Most approaches to graph-based SLAM focus on estimating the most-likely configuration of the nodes and are therefore referred to as maximum-likelihood (ML) techniques [2, 5, 6, 11, 13]. They do not consider to compute the full posterior about the map and the poses of the robot. The approach presented in this paper also belongs to this class of methods.

The goal of graph-based ML mapping algorithms is to find the configuration of the nodes that maximizes the likelihood of the observations. For a more precise formulation consider the following definitions:

- $\mathbf{x} = (x_1 \ \cdots \ x_n)^T$ is a vector of parameters which describes a configuration of the nodes. Note that the parameters x_i do not need to be the absolute poses of the nodes. They are arbitrary variables which can be mapped to the poses of the nodes in real world coordinates.
- δ_{ji} describes a constraint between the nodes j and i. It refers to an observation of node j seen from node i. These constraints are the edges in the graph structure.
- Ω_{ji} is the information matrix modeling the uncertainty of δ_{ji}.
- $f_{ji}(\mathbf{x})$ is a function that computes a zero noise observation according to the current configuration of the nodes j and i. It returns an observation of node j seen from node i.

Given a constraint between node j and node i, we can define the *error* e_{ji} introduced by the constraint as

$$e_{ji}(\mathbf{x}) \quad = \quad f_{ji}(\mathbf{x}) - \delta_{ji} \tag{1}$$

as well as the *residual* r_{ji}

$$r_{ji}(\mathbf{x}) \quad = \quad -e_{ji}(\mathbf{x}). \tag{2}$$

Note that at the equilibrium point, e_{ji} is equal to 0 since $f_{ji}(\mathbf{x}) = \delta_{ji}$. In this case, an observation perfectly matches the current configuration of the nodes. Assuming a Gaussian observation error, the negative log likelihood of an observation f_{ji} is

$$\begin{aligned} F_{ji}(\mathbf{x}) \quad &\propto \quad (f_{ji}(\mathbf{x}) - \delta_{ji})^T \, \Omega_{ji} \, (f_{ji}(\mathbf{x}) - \delta_{ji}) && (3)\\ &= \quad e_{ji}(\mathbf{x})^T \Omega_{ji} e_{ji}(\mathbf{x}) && (4)\\ &= \quad r_{ji}(\mathbf{x})^T \Omega_{ji} r_{ji}(\mathbf{x}). && (5) \end{aligned}$$

Under the assumption that the observations are independent, the overall negative log likelihood of a configuration $\mathbf{x}$ is

$$\begin{aligned} F(\mathbf{x}) \quad &= \quad \sum_{\langle j,i \rangle \in \mathcal{C}} F_{ji}(\mathbf{x}) && (6)\\ &= \quad \sum_{\langle j,i \rangle \in \mathcal{C}} r_{ji}(\mathbf{x})^T \Omega_{ji} r_{ji}(\mathbf{x}). && (7) \end{aligned}$$

Here $\mathcal{C} = \{\langle j_1, i_1\rangle, \ldots, \langle j_M, i_M\rangle\}$ is set of pairs of indices for which a constraint $\delta_{j_m i_m}$ exists.

The goal of a ML approach is to find the configuration $\mathbf{x}^*$ of the nodes that maximizes the likelihood of the observations. This can be written as

$$\mathbf{x}^* = \operatorname*{argmin}_{\mathbf{x}} F(\mathbf{x}). \tag{8}$$

IV. Stochastic Gradient Descent for Maximum Likelihood Mapping

Olson *et al.* [13] propose to use a variant of the preconditioned stochastic gradient descent (SGD) to address the SLAM problem. The approach minimizes Eq. (8) by iteratively selecting a constraint $\langle j, i\rangle$ and by moving the nodes of the network in order to decrease the error introduced by the selected constraint. Compared to the standard formulation of gradient descent, the constraints are not optimized as a whole but individually. The nodes are updated according to the following equation:

$$\mathbf{x}^{t+1} \quad = \quad \mathbf{x}^t + \underbrace{\lambda \cdot \mathbf{H}^{-1} J_{ji}^T \Omega_{ji} r_{ji}}_{\Delta \mathbf{x}_{ji}} \tag{9}$$

Here $\mathbf{x}$ is the set of variables describing the locations of the poses in the network and $\mathbf{H}^{-1}$ is a preconditioning matrix. J_{ji} is the Jacobian of f_{ji}, Ω_{ji} is the information matrix capturing the uncertainty of the observation, and r_{ji} is the residual.

Reading the term $\Delta \mathbf{x}_{ji}$ of Eq. (9) from right to left gives an intuition about the sequential procedure used in SGD:

- r_{ji} is the residual which is the opposite of the error vector. Changing the network configuration in the direction of the residual r_{ji} will decrease the error e_{ji}.
- Ω_{ji} represents the information matrix of a constraint. Multiplying it with r_{ji} scales the residual components according to the information encoded in the constraint.
- J_{ji}^T: The role of the Jacobian is to map the residual term into a set of variations in the parameter space.
- $\mathbf{H}$ is the Hessian of the system and it represents the curvature of the error function. This allows us to scale the variations resulting from the Jacobian depending on the curvature of the error surface. We actually use an approximation of $\mathbf{H}$ which is computed as

$$\mathbf{H} \quad \simeq \quad \sum_{\langle j,i\rangle} J_{ji} \Omega_{ji} J_{ji}^T. \tag{10}$$

  Rather than inverting the full Hessian which is computationally expensive, we approximate it by

$$\mathbf{H}^{-1} \quad \simeq \quad [\operatorname{diag}(\mathbf{H})]^{-1}. \tag{11}$$

- λ is a learning rate which decreases with the iteration of SGD and which makes the system to converge to an equilibrium point.

In practice, the algorithm decomposes the overall problem into many smaller problems by optimizing the constraints individually. Each time a solution for one of these subproblems is found, the network is updated accordingly. Obviously, updating the different constraints one after each other can have opposite effects on a subset of variables. To avoid infinitive oscillations, one uses the learning rate to reduce the fraction of the residual which is used for updating the variables. This makes the solutions of the different sub-problems to asymptotically converge towards an equilibrium point that is the solution reported by the algorithm.

This framework allows us to iteratively reduce the error given the network of constraints. The optimization approach, however, leaves open how the nodes are represented (parameterized). Since the parameterization defines also the structure of the Jacobians, it has a strong influence on the performance of the algorithm.

The next section addresses the problem of how to parameterize a graph in order to efficiently carry out the optimization approach.

V. Network Parameterizations

The poses $\mathbf{p} = \{p_1, \ldots, p_n\}$ of the nodes define the configuration of the network. The poses can be described by a vector of parameters $\mathbf{x}$ such that a bijective mapping g between $\mathbf{p}$ and $\mathbf{x}$ exists

$$\mathbf{x} = g(\mathbf{p}) \qquad\qquad \mathbf{p} = g^{-1}(\mathbf{x}). \tag{12}$$

As previously explained, in each iteration SGD decomposes the problem into a set of subproblems and solves them successively. In this work, a subproblem is defined as the optimization of a single constraint. Different solutions to the individual subproblems can have antagonistic effects when combining them.

The parameterization g defines also the subset of variables that are modified by a single constraint update. A good parameterization defines the subproblems in a way that the combination step leads only to small changes of the individual solutions.

A. Incremental Pose Parameterization

Olson *et al.* propose the so-called incremental pose parameterization. Given a set of node locations p_i and given a fixed order on the nodes, the incremental parameters x_i can be computed as follows

$$x_i \quad = \quad p_i - p_{i-1}. \tag{13}$$

Note that x_i is computed as the difference between two subsequent nodes and not by motion composition. Under this parameterization, the error in the global reference frame (indicated by primed variables) has the following form

$$e'_{ji} \quad = \quad p_j - (p_i \oplus \delta_{ji}) \tag{14}$$

$$= \quad \left(\sum_{k=i+1}^{j} x_k\right) + \underbrace{\left(\prod_{k=1}^{i} \tilde{R}_k\right)}_{R_i} \delta_{ji}, \tag{15}$$

where $\oplus$ is the motion composition operator according to Lu and Milios [11] and $\tilde{R}_k$ the homogenous rotation matrix of the *parameter* x_k. The term R_k is defined as the rotation matrix of the *pose* p_k. The information matrix in the global reference frame can be computed as

$$\Omega'_{ji} \quad = \quad R_i \Omega_{ji} R_i^T. \tag{16}$$

According to Olson *et al.* [13], neglecting the contribution of the angular terms of $x_0, \ldots, x_i$ to the Jacobian results in the following simplified form

$$J'_{ji} = \sum_{k=i+1}^{j} \mathcal{I}_k \quad \text{with} \quad \mathcal{I}_k = (0 \cdots 0 \underbrace{I}_{k} 0 \cdots 0). \quad (17)$$

Here 0 is the 3 by 3 zero matrix and I is the 3 by 3 identity.

Updating the network based on the constraint $\langle j, i \rangle$ with such an Jacobian results in keeping the node i fixed and in distributing the residual along all nodes between j and i.

Olson *et al.* weight the residual proportional to $j-i$ which is the number of nodes involved in the constraint. The parameter x_k of the node k with $k = i+1, \ldots, j$ is updated as follows

$$\Delta x_k \quad = \quad \lambda w_k \Omega'_{ji} r'_{ji}, \quad (18)$$

where the weight w_k is computed as

$$w_k = (j-i) \left[\sum_{m=i+1}^{j} D_m^{-1} \right]^{-1} D_k^{-1}. \quad (19)$$

In Eq. (19), D_k are the matrices containing the diagonal elements of the k^{th} block of the Hessian $\mathbf{H}$. Intuitively, each variable is updated proportional to the uncertainty about that variable. Note that the simple form of the Jacobians allows us to update the parameter vector for each node individually as expressed by Eq. (18).

The approach presented in this section is currently one of the best solutions to ML mapping. However, it has the following drawbacks:

- In practice, the incremental parameterization cannot deal with arbitrarily connected networks. This results from the approximation made in Eq. (17), in which the angular components are ignored when computing the Jacobian. This approximation is only valid if the subsequent nodes in Eq. (13) are spatially close. Furthermore, the way the error is distributed over the network assumes that the nodes are ordered according to poses along the trajectory. This results in adding a large number of nodes to the network whenever the robot travels for a long time in the same region. This requirement prevents an approach from merging multiple nodes into a single one. Merging or pruning nodes, however, is a necessary precondition to allow the robot lifelong map learning.
- When updating a constraint between the nodes j and i, the parameterization requires to change the j-i nodes. As a result, each node is likely to be updated by several constraints. This leads to a high interaction between constraints and will typically reduce the convergence speed of SGD. For example, the node k will be updated by all constraints $\langle j', i' \rangle$ with $i' < k \leq j'$. Note that using an intelligent lookup structure, this operation can be carried out in $\mathcal{O}(\log n)$ time where n is the number of nodes in the network [13]. Therefore, this is a problem of convergence speed of SGD and not a computational problem.

B. Tree Parameterization

Investigating a different parameterization which preserves the advantages of the incremental one but overcomes its drawbacks is the main motivation for our approach. First, our method should be able to deal with arbitrary network topologies. This would enable us to compress the graph whenever robot revisits a place. As a result, the size of the network would be proportional to the visited area and not to the length of the trajectory. Second, the number of nodes in the graph updated by each constraint should mainly depend on the topology of the environment. For example, in case of a loop-closure a large number of nodes need to be updated but in all other situations the update is limited to a small number of nodes in order to keep the interactions between constraints small.

Our idea is to first construct a spanning tree from the (arbitrary) graph. Given such a tree, we define the parameterization for a node as

$$x_i \quad = \quad p_i - p_{\text{parent}(i)}, \quad (20)$$

where $p_{\text{parent}(i)}$ refers to the parent of node i in the spanning tree. As defined in Eq. (20), the tree stores the differences between poses. As a consequence, one needs to process the tree up to the root to compute the actual pose of a node in the global reference frame.

However, to obtain only the difference between two arbitrary nodes, one needs to traverse the tree from the first node upwards to the first common ancestor of both nodes and then downwards to the second node. The same holds for computing the error of a constraint. We refer to the nodes one needs to traverse on the tree as the path of a constraint. For example, $\mathcal{P}_{ji}$ is the path from node i to node j for the constraint $\langle j, i \rangle$. The path can be divided into an ascending part $\mathcal{P}_{ji}^{[-]}$ of the path starting from node i and a descending part $\mathcal{P}_{ji}^{[+]}$ to node j. We can then compute the error in the global frame by

$$\begin{aligned} e'_{ji} &= p_j - (p_i \oplus \delta_{ji}) && (21) \\ &= p_j - (p_i + R_i \delta_{ji}) && (22) \\ &= \sum_{k^{[+]} \in \mathcal{P}_{ji}^{[+]}} x_{k^{[+]}} - \sum_{k^{[-]} \in \mathcal{P}_{ji}^{[-]}} x_{k^{[-]}} - R_i \delta_{ji}. && (23) \end{aligned}$$

Here R_i is the rotation matrix of the pose p_i. It can be computed according to the structure of the tree as the product of the individual rotation matrices along the path to the root.

Note that this tree does not replace the graph as an internal representation. The tree only defines the parameterization of the nodes. It can furthermore be used to define an order in which the optimization algorithm can efficiently process the constraints as we will explain in the remainder of this section. For illustration, Figure 2 (a) and (b) depict two graphs and possible parameterization trees.

Similar to Eq. (16), we can express the information matrix associated to a constraint in the global frame by

$$\Omega'_{ji} \quad = \quad R_i \Omega_{ji} R_i^T. \quad (24)$$

As proposed in [13], we neglect the contribution of the rotation matrix R_i in the computation of the Jacobian. This approximation speeds up the computation significantly. Without

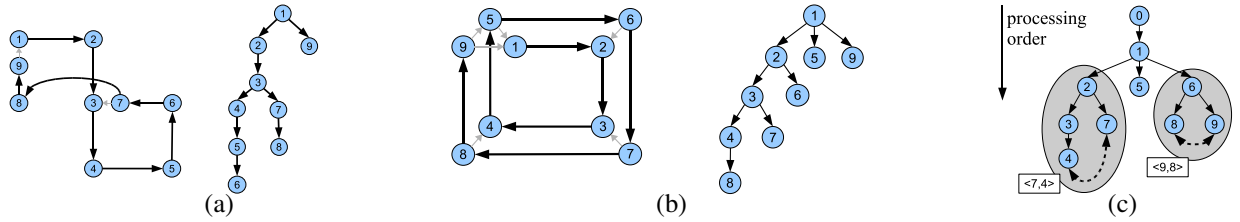

Fig. 2. (a) and (b): Two small example graphs and the trees used to determine the parameterizations. The small grey connections are constraints introduced by observations where black ones result from odometry. (c) Processing the constraints ordered according to the node with the smallest level in the path avoids the recomputation of rotational component of all parents. The same holds for subtrees with different root nodes on the same level.

this approximation the update of a single constraint influences the poses of all nodes up to the root.

The approximation leads to the following Jacobian:

$$J'_{ji} = \sum_{k^{[+]} \in \mathcal{P}_{ji}^{[+]}} \mathcal{I}_{k^{[+]}} - \sum_{k^{[-]} \in \mathcal{P}_{ji}^{[-]}} \mathcal{I}_{k^{[-]}} \tag{25}$$

Compared to the approach described in the previous section, the number of updated variables per constraint is in practice smaller when using the tree. Our approach updates $|\mathcal{P}_{ji}|$ variables rather than $j - i$. The weights w_k are computed as

$$w_k = |\mathcal{P}_{ji}| \left[\sum_{m \in \mathcal{P}_{ji}}^{j} D_m^{-1} \right]^{-1} D_k^{-1}, \tag{26}$$

where D_k is the k-th diagonal block element of $\mathbf{H}$. This results in the following update rule for the variable x_k

$$\Delta x_k = \lambda w_k \cdot \mathrm{s}(x_k, i, j) \cdot \Omega'_{ji} r'_{ji}, \tag{27}$$

where the value of $\mathrm{s}(x_k, j, i)$ is $+1$ or -1 depending on where the parameter x_k is located on the path $\mathcal{P}_{ji}$:

$$\mathrm{s}(x_k, j, i) = \begin{cases} +1 & \text{if } x_k \in \mathcal{P}_{ji}^{[+]} \\ -1 & \text{if } x_k \in \mathcal{P}_{ji}^{[-]} \end{cases} \tag{28}$$

Our parameterization maintains the simple form of the Jacobians which enables us to perform the update of each parameter variable individually (as can be seen in Eq. (27)). Note that in case one uses a tree that is degenerated to a list, this parameterization is equal to the one proposed by Olson *et al.* [13]. In case of a non-degenerated tree, our approach offers several advantages as we will show in the experimental section of this paper.

The optimization algorithm specifies how to update the nodes but does not specify the order in which to process the constraints. We can use our tree parameterization to sort the constraints which allows us to reduce the computational complexity of our approach.

To compute the residual of a constraint $\langle j, i \rangle$, we need to know the rotational component of the node i. This requires to traverse the tree up to the first node for which the rotational component is known. In the worst case, this is the root of the tree.

Let the *level* of a node be the distance in the tree between the node itself and the root. Let z_{ji} be the node in the path of the constraint $\langle j, i \rangle$ with the smallest level. The level of the constraint is then defined as the level of z_{ji}.

Our parameterization implies that updating a constraint will never change the configuration of a node with a level smaller than the level of the constraint. Based on this knowledge, we can sort the constraints according to their level and process them in that order. As a result, it is sufficient to access the parent of z_{ji} to compute the rotational component of the node i since all nodes with a smaller level than z_{ji} have already been corrected.

Figure 2 (c) illustrates such a situation. The constraint $\langle 7, 4 \rangle$ with the path $4, 3, 2, 7$ does not change any node with a smaller level than the one of node 2. It also does not influence other subtrees on the same level such as the nodes involved in the constraint $\langle 9, 8 \rangle$.

In the following section, we describe how we actually build the tree given the trajectory of a robot or an arbitrary network as input.

VI. Construction of the Spanning Tree

When constructing the parameterization tree, we distinguish two different situations. First, we assume that the input is a sequence of positions belonging to a trajectory traveled by the robot. Second, we explain how to build the tree given an arbitrary graph of relations.

In the first case, the subsequent poses are located closely together and there exist constraints between subsequent poses resulting from odometry or scan-matching. Further constraints between arbitrary nodes result from observations when revisiting a place in the environment. In this setting, we build our parameterization tree as follows:

1) We assign a unique id to each node based on the timestamps and process the nodes accordingly.
2) The first node is the root of the tree (and therefore has no parent).
3) As the parent of a node, we choose the node with the smallest id for which a constraint to the current node exists.

This tree can be easily constructed on the fly. The Figures 2 (a) and (b) illustrates graphs and the corresponding trees. This tree has a series of nice properties when applying our optimization algorithm to find a minimal error configuration of the nodes. These properties are:

- The tree can be constructed incrementally: when adding a new node it is not required to change the existing tree.
- In case the robot moves through nested loops, the interaction between the updates of the nodes belonging to the individual loops depends on the number of nodes the loops have in common.

- When retraversing an already mapped area and adding constraints between new and previously added nodes, the length of the path in the tree between these nodes is small. This means that only a small number of nodes need to be updated.

The second property is illustrated in Figure 2 (a). The two loops in that image are only connected via the constraint between the nodes 3 and 7. They are the only nodes that are updated by constraints of both loops.

The third property is illustrated in Figure 2 (b). Here, the robot revisits a loop. The nodes 1 to 4 are chosen as the parents for all further nodes. This results in short paths in the tree when updating the positions of the nodes while retraversing known areas.

The complexity of the approach presented so far depends on the length of the trajectory and not on the size of the environment. These two quantities are different in case the robot revisits already known areas. This becomes important whenever the robot is deployed in a bounded environment for a long time and has to update its map over time. This is also known as lifelong map learning. Since our parameterization is not restricted to a trajectory of sequential poses, we have the possibility of a further optimization. Whenever the robot revisits a known place, we do not need to add new nodes to the graph. We can assign the current pose of the robot to an already existing node in the graph.

Note that this can be seen as an approximation similar to adding a rigid constraint neglecting the uncertainty of the corresponding observation. However, in case local maps (e.g., grid maps) are used as nodes in the network, it makes sense to use such an approximation since one can localize a robot in an existing map quite accurately.

To also avoid adding new constraints to the network, we can refine an existing constraint between two nodes in case of a new observation. Given a constraint $\delta_{ji}^{(1)}$ between the nodes j and i in the graph and a new constraint $\delta_{ji}^{(2)}$ based on the current observation. Both constraints can be combined to a single constraint which has the following information matrix and mean:

$$\Omega_{ji} = \Omega_{ji}^{(1)} + \Omega_{ji}^{(2)} \tag{29}$$

$$\delta_{ji} = \Omega_{ji}^{-1}(\Omega_{ji}^{(1)} \cdot \delta_{ji}^{(1)} + \Omega_{ji}^{(2)} \cdot \delta_{ji}^{(2)}) \tag{30}$$

As a result, the size of the problem does not increase when revisiting known locations. As the experiments illustrate, this node reduction technique leads to an increased convergence speed.

In case the input to our algorithm is an arbitrary graph and no natural order of the nodes is provided, we compute a minimal spanning tree to define the parameterization. Since no additional information (like consecutive poses according to a trajectory) is available, we cannot directly infer which parts of the graph are well suited to form a subtree in the parameterization tree. The minimal spanning tree appears to yield comparable results with respect to the number of iterations needed for convergence in all our experiments.

Fig. 3. The map of the Intel Research Lab before (left) and after (right) execution of our algorithm (1000 nodes, runtime $<$1s).

VII. Experiments

This section is designed to evaluate the properties of our tree parameterization for learning maximum likelihood maps. We first show that such a technique is well suited to generate accurate occupancy grid maps given laser range data and odometry from a real robot. Second, we provide simulation experiments on large-scale datasets. We furthermore provide a comparison between our approach, Olson's algorithm [13], and multi-level relaxation by Frese *et al.* [5]. Finally, we analyze our approach and investigate properties of the tree parameterization in order to explain why we obtain better results then the other methods.

A. Real World Experiments

The first experiment is designed to illustrate that our approach can be used to build maps from real robot data. The goal was to build an accurate occupancy grid map given the laser range data obtained by the robot. The nodes of our graph correspond to the individual poses of the robot during data acquisition. The constraints result from odometry and from the pair-wise matching of laser range scans. Figure 3 depicts two maps of the Intel Research Lab in Seattle. The left one is constructed from raw odometry and the right one is the result obtained by our algorithm. As can be seen, the corrected map shows no inconsistencies such as double corridors. Note that this dataset is freely available on the Internet.

B. Simulated Experiments

The second set of experiments is designed to measure the performance of our approach quantitatively. Furthermore, we compare our technique to two current state-of-the-art SLAM approaches that work on constraint networks, namely multi-level relaxation by Frese *et al.* [5] and Olson's algorithm [13]. In the experiments, we used the two variants of our method: the one that uses the node reduction technique described in Section VI and the one that maintains all the nodes in the graph.

In our simulation experiments, we moved a virtual robot on a grid world. An observation is generated each time the current position of the robot was close to a previously visited location. We corrupted the observations with a variable amount of noise for testing the robustness of the algorithms. We simulated different datasets resulting in graphs with a number of constraints between around 4,000 and 2 million.

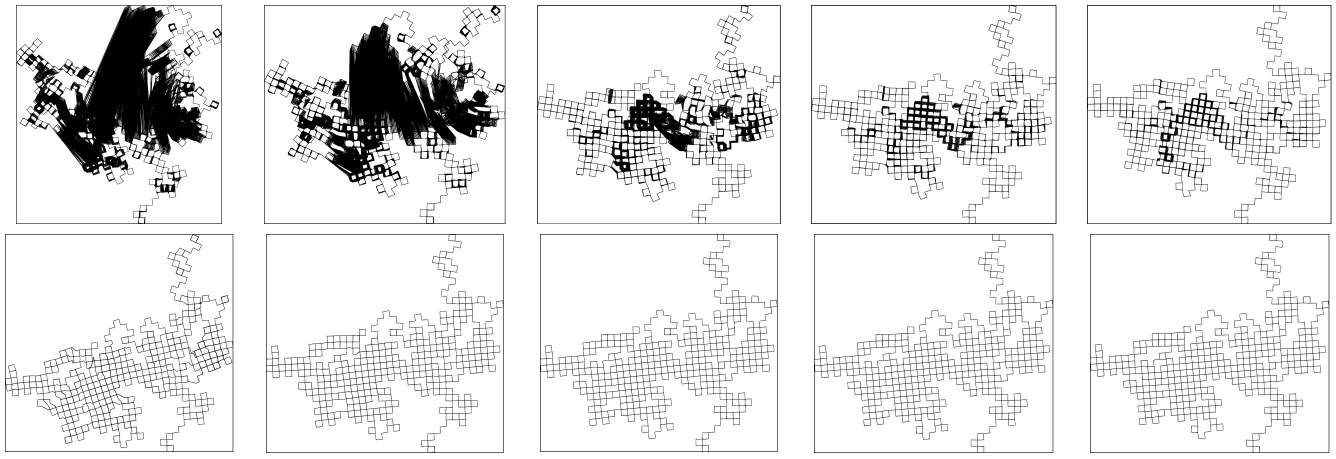

Fig. 4. Results of Olson's algorithm (first row) and our approach (second row) after 1, 10, 50, 100, 300 iterations for a network with 64k constraints. The black areas in the images result from constraints between nodes which are not perfectly corrected after the corresponding iteration (for timings see Figure 6).

Fig. 5. The result of MLR strongly depends on the initial configuration of the network. Left: small initial pose error, right: large initial pose error.

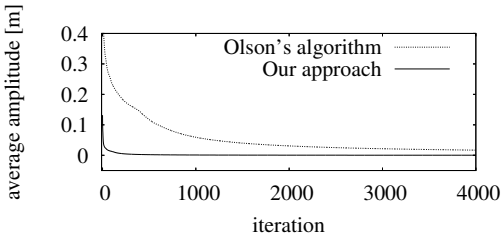

Fig. 7. The average amplitude of the oscillations of the nodes due to the antagonistic effects of different constraints.

Figure 4 depicts a series of graphs obtained by Olson's algorithm and our approach after different iterations. As can be seen, our approach converges faster. Asymptotically, both approaches converge to a similar solution.

In all our experiments, the results of MLR strongly depended on the initial positions of the nodes. In case of a good starting configuration, MLR converges to an accurate solution similar to our approach as shown in Figure 5 (left). Otherwise, it is likely to diverge (right). Olson's approach as well as our technique are more or less independent of the initial poses of the nodes.

To evaluate our technique quantitatively, we first measured the error in the network after each iteration. The left image of Figure 6 depicts a statistical experiments over 10 networks with the same topology but different noise realizations. As can be seen, our approach converges significantly faster than the approach of Olson *et al.* For medium size networks, both approaches converge asymptotically to approximatively the same error value (see middle image). For large networks, the high number of iterations needed for Olson's approach prevented us from showing this convergence experimentally. Due to the sake of brevity, we omitted comparisons to EKF and Gauss Seidel relaxation because Olson *et al.* already showed that their approach outperforms such techniques.

Additionally, we evaluated in Figure 6 (right) the average computation time per iteration of the different approaches. As a result of personal communication with Edwin Olson, we furthermore analyzed a variant of his approach which is restricted to spherical covariances. It yields similar execution times *per iteration* than our approach. However, this restricted variant has still the same converge speed with respect to the number of iterations than Olson's unrestricted technique. As can be seen from that picture, our node reduction technique speeds up the computations up to a factor of 20.

C. Analysis of the Algorithm

The experiments presented above illustrated that our algorithm offers significant improvements compared to both other techniques. The goal of this section is to experimentally point out the reasons for these improvements.

The presented tree parameterization allows us to decompose the optimization of the whole graph into a set of weakly interacting problems. A good measure for evaluating the interaction between the constraints is the average number l of updated nodes per constraint. For example, a network with a large value of l has typically a higher number of interacting constraints compared to networks with low values of l. In all experiments, our approach had a value between 3 and 7. In contrast to that, this values varies between 60 and 17,000 in Olson's approach on the same networks. Note that such a high average path length reduces the convergence speed of Olson's algorithm but does not introduce a higher complexity.

The optimization approach used in this paper as well as in Olson's algorithm updates for each constraint the involved nodes to minimize the error in the network. As a result, different constraints can update poses in an antagonistic way during one iteration. This leads to oscillations in the position

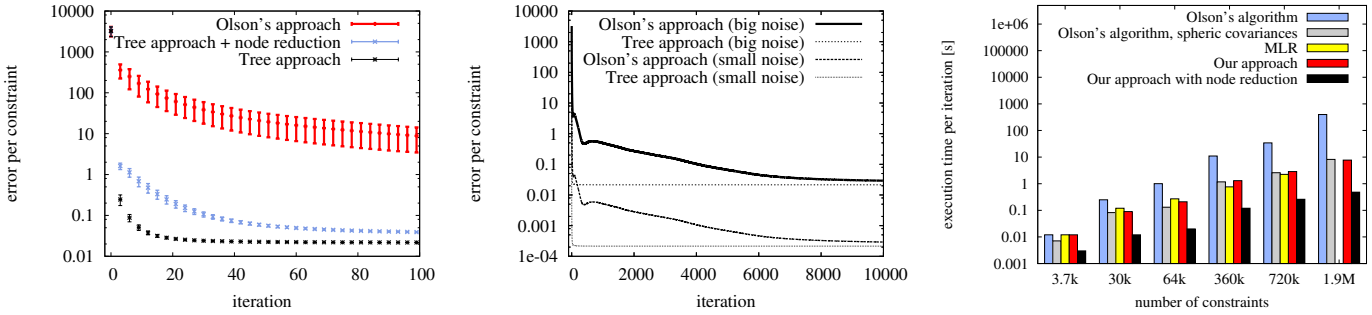

Fig. 6. The left image illustrates shows the error of our and Olson's approach in a statistical experiment ($\sigma = 0.05$ confidence). The image in the middle shows that both techniques converge asymptotically to the same error. The right image shows the average execution time *per iteration* for different networks. For the 1.9M constraints network, the executing of MLR required memory swapping and the result is therefore omitted.

of a node before convergence. Figure 7 illustrates the average amplitude of such an oscillations for Olson's algorithm as well as for our approach. As can be seen, our techniques converges faster to an equilibrium point. This a further reason for the higher convergence speed of our approach.

D. Complexity

Due to the nature of gradient descent, the complexity of our approach per iteration depends linearly on the number of constraints. For each constraint $\langle j, i \rangle$, our approach modifies exactly those nodes which belong to the path $\mathcal{P}_{ji}$ in the tree. Since each constraint has an individual path length, we consider the average path length l. This results in an complexity per iteration of $\mathcal{O}(M \cdot l)$, where M is the number of constraints. In all our experiments, l was approximatively $\log N$, where N is the number of nodes. Note that given our node reduction technique, M as well as N are bounded by the size of the environment and not by the length of the trajectory.

A further advantage of our technique compared to MLR is that it is easy to implement. The function that performs a single iteration requires less than 100 lines of C++ code. An open source implementation, image and video material, and the datasets are available at the authors' web-pages.

VIII. Conclusion

In this paper, we presented a highly efficient solution to the problem of learning maximum likelihood maps for mobile robots. Our technique is based on the graph-formulation of the simultaneous localization and mapping problem and applies a gradient descent based optimization scheme. Our approach extends Olson's algorithm by introducing a tree-based parameterization for the nodes in the graph. This has a significant influence on the convergence speed and execution time of the method. Furthermore, it enables us to correct arbitrary graphs and not only a list of sequential poses. In this way, the complexity of our method depends on the size of the environment and not directly on the length of the input trajectory. This is an important precondition to allow a robot lifelong map learning in its environment.

Our method has been implemented and exhaustively tested on simulation experiments as well as on real robot data. We furthermore compared our method to two existing, state-of-the-art solutions which are multi-level relaxation and Olson's algorithm. Our approach converges significantly faster than both approaches and yields accurate maps with low errors.

Acknowledgment

The authors would like to gratefully thank Udo Frese for his insightful comments and for providing us his MLR implementation for comparisons. Further thanks to Edwin Olson for his helpful comments on this paper. This work has partly been supported by the DFG under contract number SFB/TR-8 (A3) and by the EC under contract number FP6-2005-IST-5-muFly and FP6-2005-IST-6-RAWSEEDS.

References

[1] M. Bosse, P.M. Newman, J.J. Leonard, and S. Teller. An ALTAS framework for scalable mapping. In *Proc. of the IEEE Int. Conf. on Robotics & Automation*, pages 1899–1906, Taipei, Taiwan, 2003.

[2] T. Duckett, S. Marsland, and J. Shapiro. Fast, on-line learning of globally consistent maps. *Autonomous Robots*, 12(3):287 – 300, 2002.

[3] C. Estrada, J. Neira, and J.D. Tardós. Hierachical slam: Real-time accurate mapping of large environments. *IEEE Transactions on Robotics*, 21(4):588–596, 2005.

[4] U. Frese. Treemap: An $o(logn)$ algorithm for indoor simultaneous localization and mapping. *Autonomous Robots*, 21(2):103–122, 2006.

[5] U. Frese, P. Larsson, and T. Duckett. A multilevel relaxation algorithm for simultaneous localisation and mapping. *IEEE Transactions on Robotics*, 21(2):1–12, 2005.

[6] J.-S. Gutmann and K. Konolige. Incremental mapping of large cyclic environments. In *Proc. of the IEEE Int. Symp. on Comp. Intelligence in Robotics and Automation*, pages 318–325, Monterey, CA, USA, 1999.

[7] A. Howard, M.J. Matarić, and G. Sukhatme. Relaxation on a mesh: a formalism for generalized localization. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, pages 1055–1060, 2001.

[8] S. Julier, J. Uhlmann, and H. Durrant-Whyte. A new approach for filtering nonlinear systems. In *Proc. of the American Control Conference*, pages 1628–1632, Seattle, WA, USA, 1995.

[9] J. Ko, B. Stewart, D. Fox, K. Konolige, and B. Limketkai. A practical, decision-theoretic approach to multi-robot mapping and exploration. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, pages 3232–3238, Las Vegas, NV, USA, 2003.

[10] J.J. Leonard and H.F. Durrant-Whyte. Mobile robot localization by tracking geometric beacons. *IEEE Transactions on Robotics and Automation*, 7(4):376–382, 1991.

[11] F. Lu and E. Milios. Globally consistent range scan alignment for environment mapping. *Autonomous Robots*, 4:333–349, 1997.

[12] J. Neira and J.D. Tardós. Data association in stochastic mapping using the joint compatibility test. *IEEE Transactions on Robotics and Automation*, 17(6):890–897, 2001.

[13] E. Olson, J.J. Leonard, and S. Teller. Fast iterative optimization of pose graphs with poor initial estimates. In *Proc. of the IEEE Int. Conf. on Robotics & Automation*, pages 2262–2269, 2006.

[14] R. Smith, M. Self, and P. Cheeseman. Estimating uncertain spatial realtionships in robotics. In I. Cox and G. Wilfong, editors, *Autonomous Robot Vehicles*, pages 167–193. Springer Verlag, 1990.