Nonlinear Constraint Network Optimization for Efficient Map Learning

Giorgio Grisetti, Cyrill Stachniss, and Wolfram Burgard

***Abstract*—Learning models of the environment is one of the fundamental tasks of mobile robots since maps are needed for a wide range of robotic applications, such as navigation and transportation tasks, service robotic applications, and several others. In the past, numerous efficient approaches to map learning have been proposed. Most of them, however, assume that the robot lives on a plane. In this paper, we present a highly efficient maximum-likelihood approach that is able to solve 3-D and 2-D problems. Our approach addresses the so-called graph-based formulation of simultaneous localization and mapping (SLAM) and can be seen as an extension of Olson's algorithm toward non-flat environments. It applies a novel parameterization of the nodes of the graph that significantly improves the performance of the algorithm and can 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. Furthermore, our approach is able to appropriately distribute the roll, pitch, and yaw error over a sequence of poses in 3-D mapping problems. We implemented our technique and compared it with multiple other graph-based SLAM solutions. As we demonstrate in simulated and real-world experiments, our method converges faster than the other approaches and yields accurate maps of the environment.**

***Index Terms*—Graph-optimization, mapping, simultaneous localization and mapping (SLAM), (stochastic) gradient descent.**

I. Introduction

To efficiently solve the majority of robotic applications such as transportation tasks, search and rescue, or automated vacuum cleaning, a map of the environment is required. Acquiring such models has therefore been a major research focus in the robotics community over the past few decades. Learning maps under pose 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 in the underlying estimation technique such as extended Kalman filters (EKFs), information filters, particle filters, smoothing, or least-squares error-minimization techniques.

Manuscript received February 1, 2008; revised October 27, 2008. First published July 14, 2009; current version published September 1, 2009. This work was supported in part by the German Research Foundation under Contract SFB/TR-8 (A3) and in part by the European Commission under Contract FP6-2005-IST-5-muFly, Contract FP6-2005-IST-6-RAWSEEDS, and Contract FP7-ICT-231888-EUROPA. The Associate Editor for this paper was C. Laugier.

The authors are with the Laboratory for Autonomous Intelligent Systems, Department of Computer Science, University of Freiburg, 79110 Freiburg, Germany (e-mail: grisetti@informatik.uni-freiburg.de; stachnis@informatik.uni-freiburg.de; burgard@informatik.uni-freiburg.de).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TITS.2009.2026444

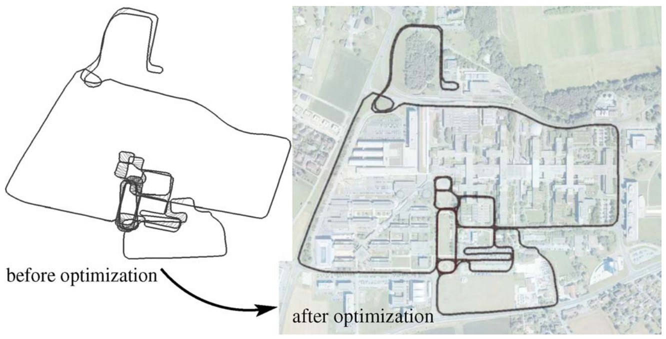

Fig. 1. Constraint network corresponding to a data set recorded with an instrumented car at the École Polytechnique Fédérale de Lausanne (EPFL) campus in Switzerland (left) before and (right) after optimization. The corrected network is overlaid with an aerial image.

In this paper, we consider the popular and 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 [5], [8], [11], [13], [14], [16], [22], [26], [27], [36]. Spatial constraints between poses that result from observations and from odometry are encoded in the edges between the nodes.

In the context of graph-based SLAM, one typically considers two different problems. The first problem is to identify the constraints based on sensor data. This so-called data association problem is typically hard due to potential ambiguities or symmetries in the environment. A solution to this problem is often referred to as the SLAM front end, and it directly deals with the sensor data. The second problem is to correct the poses of the robot to obtain a consistent map of the environment *given* the constraints. This part of the approach is often referred to as the optimizer or the SLAM back end. To solve this problem, one seeks for a configuration of the nodes that maximizes the likelihood of the observations 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. As a motivating example, Fig. 1 depicts an uncorrected constraint network and the corresponding corrected one.

Popular solutions to compute a network configuration that minimizes the error introduced by the constraints are iterative approaches. They can be used either to simultaneously correct all poses [14], [20], [22], [36] or to locally update parts of the network [5], [11], [13], [16], [26], [27]. Depending on the technique used, different parts of the network are updated at each iteration. The strategy for defining and performing these local updates has a significant impact on the convergence speed.

In this paper, we restrict ourselves to the problem of finding the most likely configuration of the nodes *given* the constraints. To find the constraints from laser range data, one can, for example, apply the front end of the ATLAS framework introduced in Bosse *et al.* [2], hierarchical SLAM [6], or the work of Nüchter *et al.* [26]. In the context of visual SLAM, a potential approach to obtain such constraints has recently been proposed in Steder *et al.* [33].

Our approach uses a tree structure to define and efficiently update local regions in each iteration by applying a variant of stochastic gradient descent (SGD). It extends Olson's algorithm [27] and converges significantly faster to highly accurate network configurations. Compared with other approaches to 3-D mapping, our technique utilizes a more accurate way to distribute the rotational error over a sequence of poses. Furthermore, the complexity of our approach scales with the size of the environment and not with the length of the trajectory as with the case for most alternative methods.

The remainder of this paper is organized as follows: In Section II, we formally introduce the graph-based formulation of the mapping problem and explain the usage of SGD to reduce the error of the network configuration. Whereas Section III introduces our tree parameterization, Section IV describes our approach to distribute the rotational errors over a sequence of nodes. In Section V, we then provide an upper bound for this error distribution. Section VI explains how to obtain a reduced graph representation to limit the complexity. After describing the experimental results with our approach in Section VII, we provide a detailed discussion of related work in Section VIII.

II. Maximum-Likelihood Mapping Using a Constraint Network

Most approaches to network- or graph-based SLAM focus on estimating the most likely configuration of the nodes and are therefore referred to as maximum-likelihood (ML) techniques [5], [11], [13], [14], [22], [27], [36]. Such techniques do not 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.

A. Problem Formulation

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.

1) Let $\mathbf{x} = (x_1 \quad \cdots \quad x_n)^T$ be a vector of parameters that 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 that can be mapped to the poses of the nodes in real-world coordinates.
2) Let us furthermore assume that δ_{ji} describes a constraint between 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.

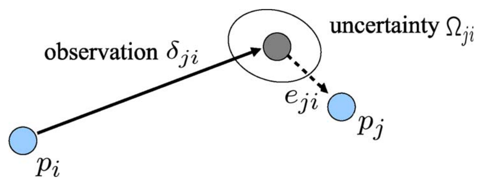

Fig. 2. Example of an observation of node j seen from node i.

3) The uncertainty in δ_{ji} is represented by the information matrix Ω_{ji}.
4) Finally, $f_{ji}(\mathbf{x})$ is a function that computes a zero-noise observation according to the current configuration of nodes j and i. It returns an observation of node j seen from node i.

Fig. 2 illustrates an observation between two nodes.

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

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

as well as the *residual* r_{ji}, i.e.,

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

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 corresponding negative log likelihood results in

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

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

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

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

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

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

There are multiple ways of solving (8). They range from approaches applying gradient descent, conjugate gradients, Gauss–Seidel relaxation, multilevel relaxation (MLR), or *LU* decomposition. In the following section, we briefly introduce SGD, which is the technique on which our approach is based.

B. SGD for ML Mapping

Olson *et al.* [27] propose to use a variant of the preconditioned SGD to address the SLAM problem. The approach minimizes (8) by sequentially selecting a constraint $\langle j, i\rangle$ (without replacement) and by moving the nodes of the network to decrease the error introduced by the selected constraint. Compared with 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:

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

Here, $\mathbf{x}$ is the set of variables describing the locations of the poses in the network, and K 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 (9) from right to left gives an intuition about the iterative procedure.

1) The term r_{ji} is the residual, which corresponds to the negative error vector. Changing the network configuration in the direction of the residual r_{ji} will decrease the error e_{ji}.
2) The term 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.
3) The Jacobian J_{ji}^T maps the residual term into a set of variations in the parameter space.
4) The term K is a preconditioning matrix. It is used to scale the variations resulting from the Jacobian depending on the curvature of the error surface. Approaches such as Olson's algorithm [27] or our previous work [13] apply a diagonal preconditioning matrix computed from the Hessian H as

$$K = [\mathrm{diag}(H)]^{-1}. \tag{10}$$

5) Finally, the quantity λ is a learning rate that decreases with each iteration of SGD and that ensures the convergence of the system.

In practice, the algorithm decomposes the overall problem into many smaller problems by individually optimizing each constraint. Thus, a portion of the network, namely, the nodes involved in a constraint, is updated in each step. Obviously, updating the different constraints one after another can have antagonistic effects on a subset of variables. To merge the contribution of the individual constraints, one uses the learning rate to reduce the fraction of the residual that is used for updating the variables. This makes the solutions of the different subproblems asymptotically converge toward an equilibrium point that is the solution reported by the algorithm.

Whereas this framework allows us to iteratively reduce the error given the network of constraints, it leaves open how the nodes are represented or parameterized. However, the choice of the parameterization has a strong influence on the performance of the algorithm. The next section addresses the problem of how to parameterize a graph so that the optimization can efficiently be carried out.

III. Tree Parameterization for SGD

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

$$\mathbf{x} = g(\mathbf{p}), \qquad \mathbf{p} = g^{-1}(\mathbf{x}). \tag{11}$$

As explained earlier in this paper, in each iteration, SGD decomposes the problem into a set of subproblems and sequentially solves them, where a subproblem is the optimization of a single constraint.

The parameterization g defines not only how the variables of the nodes are described but the subset of variables that are modified by a single constraint update as well. A good parameterization defines the subproblems in a way that the combination step only leads to small changes of the individual solutions.

Olson *et al.* [27] proposed to use the so-called incremental pose parameterization for 2-D problems. For each node i in the graph, they store the parameter x_i, which is the vector difference between the poses of node i and node $i-1$, i.e.,

$$x_i = p_i - p_{i-1}. \tag{12}$$

This parameterization has the advantage of allowing fast constraint updates. As discussed in [13], updating a constraint between two nodes i and j requires one to update all nodes $k = i+1, \ldots, j$. This leads to a low convergence speed if $i \ll j$. Furthermore, this parameterization requires that the nodes are arranged in a sequence given by the trajectory.

As aforementioned, a major contribution of this paper is an algorithm that preserves the advantages of the incremental approach but overcomes its drawbacks. The first goal is to be able to deal with arbitrary network topologies since this enables us to compress the graph whenever the robot revisits a place. As a result, the size of the network is proportional to the visited area and not to the length of the trajectory. The second goal is to make the number of nodes in the graph, which are updated by each constraint, mainly dependent on the topology of the environment and not on the trajectory taken by the vehicle. For example, in the 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 to keep the interactions between constraints small.

Our idea is to define a parameterization based on a tree structure. To obtain a tree from a given graph, we compute a spanning tree. Given such a tree, we define the parameterization for a node as

$$x_i = p_i \ominus p_{\mathrm{parent}(i)} \tag{13}$$

where $p_{\mathrm{parent}(i)}$ refers to the parent of node i in the spanning tree. The operators $\oplus$ and $\ominus$ are the standard pose compounding operators [22]. As defined in (13), the tree stores the relative transformations between poses.

Given a root node that represents the origin, such a spanning tree can be obtained by using Dijkstra's algorithm. In this paper, we use the uncertainty encoded in the information matrices

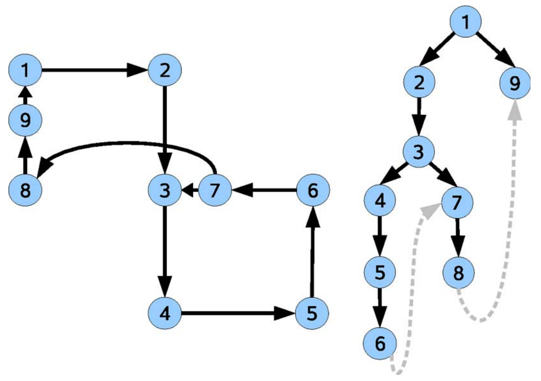

Fig. 3. (Left) Example for a constraint network. (Right) Possible tree parameterization for this graph. For illustration reasons, the off-tree constraints are also plotted (as a gray dashed line).

of the constraints as costs. This way, Dijkstra's algorithm provides the "lowest uncertainty tree" (shortest path tree) of the graph.

Note that this tree does not replace the graph as an internal representation. The tree only defines the parameterization of the nodes. For illustration, Fig. 3 depicts a graph, together with one potential parameterization tree.

According to (13), 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 relative transformation between two arbitrary nodes, one needs to traverse the tree from the first node upward to the first common ancestor of both nodes and then downward to the second node. The same holds for computing the error of a constraint. Let the *path* $\mathcal{P}_{ji}$ of a constraint between nodes i and j be the sequence of nodes in the tree that need to be traversed to reach node j, staring from node i. Such a 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 refer to the length of path of a constraint on the tree as $|\mathcal{P}_{ji}|$. We can then compute the residual of the constraint by

$$r_{ji} = (p_i \oplus \delta_{ji}) \ominus p_j. \tag{14}$$

For simplicity of notation, we will refer to the pose vector of a node as the 6-D vector $p_i = (x \quad y \quad z \quad \phi \quad \theta \quad \psi)^T$ and to its associated homogeneous transformation matrix as P_i. The same holds for the parameters used for describing the graph. We denote the parameter vector of the pose i by x_i and its transformation matrix by X_i. The transformation matrix corresponding to a constraint δ_{ji} is referred to as Δ_{ji}.

A transformation matrix X_k consists of a rotational matrix R_k and a translational component t and has the following form:

$$X_i = \begin{pmatrix} R_k & t_k \\ 0 & 1 \end{pmatrix} \tag{15}$$

with

$$X_i^{-1} = \begin{pmatrix} R_k^T & -R_k^T t_k \\ 0 & 1 \end{pmatrix}. \tag{16}$$

Accordingly, we can compute the residual in the reference frame of node j as

$$r_{ji} = P_j^{-1}(P_i \Delta_{ji}) \tag{17}$$

$$= \left(\prod_{k^{[+]} \in \mathcal{P}_{ji}^{[+]}} X_{k^{[+]}} \right)^{-1} \prod_{k^{[-]} \in \mathcal{P}_{ji}^{[-]}} X_{k^{[-]}} \Delta_{ji}. \tag{18}$$

At this point, one can directly compute the Jacobian from the residual and apply (9) to update the constraint. Note that, with this parameterization, the Jacobian has exactly $|\mathcal{P}_{ji}|$ nonzero blocks, since only the parameters in the path of the constraint appear in the residual.

IV. Updating the Tree Parameterization

So far, we have described the prerequisites for applying the preconditioned SGD to correct the poses of a network. The goal of the update rule in SGD is to iteratively update the configuration of a set of nodes to reduce the error introduced by a constraint. In (9), the term $J_{ji}^T \Omega_{ji}$ maps the variation of the error to a variation in the parameter space. This mapping, however, is a linear function. As illustrated in Frese and Hirzinger [10], the error might increase when applying such a linear function in the case of nonlinear error surfaces. In the 3-D space, the three rotational components often lead to highly nonlinear error surfaces. Therefore, it is problematic to directly apply SGD and similar minimization techniques to *large* mapping problems in combination, particularly when there is *high noise* in the observations.

In our approach, we therefore choose a modified update rule. To overcome the aforementioned problem, we apply a *nonlinear function* to describe the variation. As in the linear case, the goal of this function is to compute a transformation of the nodes along the path $\mathcal{P}_{ji}$ of the tree so that the error introduced by the corresponding constraint is reduced. The design of this function is presented in the remainder of this section. In our experiments, we observed that such an update typically leads to a smooth deformation of the nodes along the path when reducing the error. This deformation is done in two steps. We first update the rotational components R_k of the variables x_k before we update the translational components t_k.

A. Update of the Rotational Component

Without loss of generality, we consider the origin p_i of the path $\mathcal{P}_{ji}$ to be in the origin of our reference system. The orientation of p_j (in the reference frame of p_i) can be computed by multiplying the rotational matrices along the path $\mathcal{P}_{ji}$. To increase the readability of the document, we refer to the individual rotational matrices along this path as R_k, neglecting the indices [compare with (18)]. The orientation of p_j is described by

$$R_{1:n} := R_1 R_2 \cdots R_n \tag{19}$$

where n is the length of the path $\mathcal{P}_{ji}$.

Distributing a given error over a sequence of 3-D rotations can be described in the following way: We need to determine a set of increments in the intermediate rotations of the chain so that the orientation of the last node (here, node j) is $R_{1:n}B$, where B the matrix that rotates x_j to the desired orientation based on the error/residual. Formulated in a mathematical way, we need to compute a set of new rotational matrices R'_k to update the nodes so that

$$R_{1:n}B = \prod_{k=1}^{n} R'_k. \tag{20}$$

To obtain these matrices R'_k, we compute a rotation Q in the global reference frame such that

$$A_n B = Q A_n \tag{21}$$

where A_n denotes the rotation of the nth node *in the global reference frame*. By multiplying both sides of (21) with A_n^T, from the right-hand side, we obtain

$$Q = A_n B A_n^T. \tag{22}$$

We now decompose the rotation Q into a set of incremental rotations, i.e.,

$$Q_{1:n} := Q = Q_1 Q_2 \cdots Q_n \tag{23}$$

and compute the individual matrices Q_k by using the spherical linear interpolation (slerp) [1].

For this decomposition of Q, we use the parameter $u \in [0, 1]$ with $\mathbf{slerp}(Q, 0) = I$ and $\mathbf{slerp}(Q, 1) = Q$. Accordingly, the rotation Q_k is

$$Q_k = [\mathbf{slerp}(Q, u_{k-1})]^T \, \mathbf{slerp}(Q, u_k). \tag{24}$$

Furthermore, the rotation matrix A'_k of the poses P'_k along the path is

$$A'_k = Q_{1:k} A_k. \tag{25}$$

We furthermore compute the new rotational components R'_k of each node k as

$$R'_k = [A'_{\text{parent}(k)}]^T A'_k. \tag{26}$$

In (27), the learning rate λ is directly incorporated in the computation of the values of u_k. This way, the slerp function takes care of the appropriate scaling of the rotations.

In addition to that, we consider the preconditioning matrix and the length of the path when computing u_k. Similar to Olson *et al.* [27], we clamp the product $\lambda|\mathcal{P}_{ji}|$ to lie between [0, 1] so that it does not overshoot. In our implementation, we compute these values as

$$u_k = \min\left(1, \lambda|\mathcal{P}_{ji}|\right) \left[\sum_{m \in \mathcal{P}_{ji} \wedge m \leq k} d_m^{-1}\right] \left[\sum_{m \in \mathcal{P}_{ji}} d_m^{-1}\right]^{-1}. \tag{27}$$

Here, d_m is defined as the sum of the smallest eigenvalues of the information matrices of all constraints connecting node m, i.e.,

$$d_m = \sum_{\langle i,m \rangle} \min\left[\text{eigen}(\Omega_{im})\right]. \tag{28}$$

This is an approximation that works well in the case of roughly spherical covariances. Note that the eigenvalues need to be computed only once in the beginning and are then stored in the tree.

For simplicity of presentation, we demonstrated how to distribute the rotational error while keeping node i fixed. In our implementation, however, we fix the position of the so-called "top node" in the path, which is the node that is closest to the root of the tree (smallest level in the tree). As a result, the update of a constraint has less side effects on other constraints in the network. Fixing the top node instead of node i can be obtained by simply saving the pose of the top node before updating the path. After the update, one transforms all nodes along the path in a way that the top node maintains its previous pose. Furthermore, we used the matrix notation in this paper to formulate the error distribution since it provides an easier notation. However, in our implementation, we use quaternions to represent the rotations because they are numerically more stable. Both formulations are theoretically equivalent. Note that an open-source implementation of our optimizer is available online [32].

B. Update of the Translational Component

Compared with the previously described update of the rotational component, the update of the translational component can be done in a straightforward manner. In our current system, we distribute the translational error over the nodes along the path without changing the previously computed rotational component.

All nodes along the path are translated by a fraction of the residuals in the x, y, and z components. This fraction depends on the uncertainty of the individual constraints encoded in the corresponding covariance matrices and is scaled with the learning rate, similarly to the case of updating the rotational component.

V. Analysis of the Rotational Residual

When distributing a rotational error over a sequence of nodes $i, \ldots, j$, one may increase the absolute value of the residual $r_{k,k-1}$ between consecutive constraints along the path (and, thus, the error $e_{k,k-1}$). For the convergence of SGD, however, it is important that this error is bounded. Therefore, in this section, we analyze the evolution of the rotational residual after distributing an error according to Section IV-A.

A generic 3-D rotation can be described in terms of an axis and an angle. Given a rotational matrix $\mathcal{R}$, we will refer to its axis of rotation as $\mathbf{axisOf}(\mathcal{R})$ and $\mathbf{angleOf}(\mathcal{R})$, respectively. According to [1], the slerp interpolation returns a set of rotation along the same axis as follows:

$$\mathcal{R}' = \mathbf{slerp}(\mathcal{R}, u) \tag{29}$$

$$\mathbf{axisOf}(\mathcal{R}') = \mathbf{axisOf}(\mathcal{R}) \tag{30}$$

$$\mathbf{angleOf}(\mathcal{R}') = u \cdot \mathbf{angleOf}(\mathcal{R}). \tag{31}$$

When distributing the rotation Q over a sequence of poses according to (23), we decompose it into a sequence of incremental rotations $Q = Q_1 Q_2 \cdots Q_n$. From (24), we know that

$$\alpha_k = \mathbf{angleOf}(Q_k) = (u_k - u_{k-1}) \cdot \mathbf{angleOf}(Q). \quad (32)$$

In the following, we will show that when distributing the rotational error along a loop, the *angle* of the residual $\mathbf{angleOf}(r_{k,k-1})$ between the consecutive poses $k-1$ and k along the path does not increase by more than α_k.

According to (18), the residual of a constraint between nodes $k-1$ and k is

$$r_{k,k-1} = X_k^{-1} \Delta_{k,k-1}. \quad (33)$$

Since we are only focusing on the rotational component of the residual, we ignore the translational part, i.e.,

$$r_{k,k-1} = R_k^T \Delta_{k,k-1} \quad (34)$$

$$R_k^T = r_{k,k-1} \Delta_{k,k-1}^T. \quad (35)$$

After updating the rotations $A_1, \ldots, A_n$ along the chain using (25), we obtain a new set of rotations $A'_1, \ldots, A'_n$ in the global reference frame. From these rotations, we recover the updated rotational parameters R'_k by using (26), i.e.,

$$R'_k \overset{(26)}{=} A'^T_{k-1} A'_k \quad (36)$$

$$\overset{(25)}{=} [Q_{1:k-1} R_{1:k-1}]^T \cdot [Q_{1:k} R_{1:k}] \quad (37)$$

$$= [R_{1:k-1}]^T Q_k R_{1:k} \quad (38)$$

$$= [R_{1:k-1}]^T Q_k R_{1:k-1} R_k. \quad (39)$$

We then compute the residual $r'_{k,k-1}$ after the update as

$$r'_{k,k-1} \overset{(34)}{=} R'^T_k \Delta_{k,k-1} \quad (40)$$

$$\overset{(39)}{=} R_k^T [R_{1:k-1}]^T Q_k^T R_{1:k-1} \Delta_{k,k-1} \quad (41)$$

$$\overset{(35)}{=} r_{k,k-1} \underbrace{\Delta_{k,k-1}^T [R_{1:k-1}]^T}_{=:Y^T} Q_k^T \underbrace{R_{1:k-1} \Delta_{k,k-1}}_{=:Y}$$

$$= r_{k,k-1} Y^T Q_k^T Y. \quad (42)$$

In (42), the term $Y^T Q_k^T Y$ quantifies the increase in the residual of a constraint between two consecutive nodes after the update. Since Y and Q are rotation matrices, we obtain

$$\left|\mathbf{angleOf}\left(Y^T Q_k^T Y\right)\right| = |\mathbf{angleOf}(Q_k)| = |\alpha_k|. \quad (43)$$

Thus, the change of the new residual is at most α_k and is therefore bounded. This is a relevant advantage compared with the error distribution presented in Grisetti *et al.* [12], which was not bounded in such a way.

VI. Complexity and Graph Reduction

Due to the nature of SGD, the complexity of our approach per iteration linearly depends on the number of constraints since each constraint is selected once per iteration (in random order). For each constraint $\langle j, i \rangle$, our approach exactly modifies those nodes that belong to the path $\mathcal{P}_{ji}$ in the tree.

The path of the constraint is defined by the tree parameterization. As a result, different paths have different lengths. Thus, we consider the average path length l to specify the complexity. It corresponds to the average number of operations needed to update a single constraint during one iteration. This results in the complexity of $\mathcal{O}(M \cdot l)$, where M is the number of constraints. In our experiments, we found that l is typically on the order of $\log N$, where N is the number of nodes.

Note that there is further space for optimizations. 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 if the robot revisits areas that are already known. 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 and update the constraints with respect to that node.

To avoid adding new constraints to the network, we can refine an existing constraint between two nodes in the case of a new observation. Let $\delta_{ji}^{(1)}$ be a constraint that has already been stored in the graph, and let $\delta_{ji}^{(2)}$ be the new constraint that would result from the current observation. Both constraints can be combined to a single constraint, which has the following information matrix and mean:

$$\Omega_{ji} = \Omega_{ji}^{(1)} + \Omega_{ji}^{(2)} \quad (44)$$

$$\delta_{ji} = \Omega_{ji}^{-1} \left(\Omega_{ji}^{(1)} \cdot \delta_{ji}^{(1)} + \Omega_{ji}^{(2)} \cdot \delta_{ji}^{(2)} \right). \quad (45)$$

This can be seen as an approximation similar to adding a rigid constraint between nodes. However, if local maps (e.g., grid maps) are used as nodes in the network, it makes sense to use such an approximation since one can quite accurately localize a robot in an existing map.

As a result, the size of the problem does not increase when revisiting known locations. The previously specified complexity stays the same, but M and N are referred to as the reduced quantities. As our experiments illustrate, this node-reduction technique leads to an increased convergence speed since fewer nodes and constraints need to be considered.

VII. Experiments

This section is designed to evaluate the properties of our previously described approach. We first demonstrate that our method is well suited to cope with the motion and sensor noise from an instrumented car equipped with laser range scanners. Second, we present the results of simulated experiments based on large 2-D and 3-D data sets. Finally, we compare our approach with different other methods, including Olson's algorithm [27], MLR [11], and SAM [4], [19].

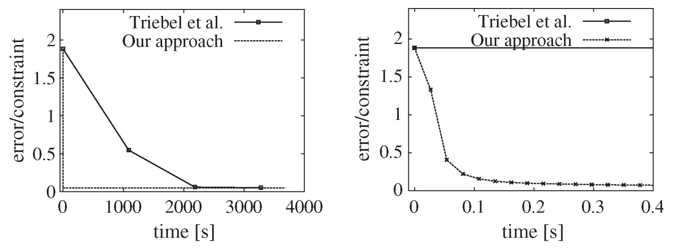

Fig. 4. Evolution of the average error per constraint (computed according to (7) divided by the number of constraints) of the approach of Triebel *et al.* [36] and our approach for the data set recorded with the autonomous car. The right image shows a magnified view to the first 400 ms.

A. Mapping With a Carlike Robot

In the first experiment, we applied our method to a real-world 3-D data set recorded with an instrumented car. Using such cars as robots has become popular in the robotics community [3], [29], [34], [38]. We used a Smart car equipped with five SICK laser range finders and various pose estimation sensors for data acquisition. Our robot constructs local 3-D maps, i.e., so-called multilevel surface maps [36], and builds a network of constrains, where each node represents such a local map. The localization system of the car is based on the Differential Global Positioning System (here, using only the standard Global Positioning System) and Inertial Measurement Unit (IMU) data. This information is used to compute the incremental constraints between subsequent poses. Constraints resulting from revisiting an already known area are obtained by matching the individual local maps using the Iterative Closest Point. More details on this matching can be found in our previous work [29].

We recorded a large-scale data set at the EPFL campus, where the robot moved on a 10-km-long trajectory. The data set includes multiple levels such as an underground parking garage and a bridge with an underpass. The motivating example of this paper (see Fig. 1) depicts the input trajectory and an overlay of the corrected trajectory on an aerial image. As can be seen, the trajectory actually matches the streets in the aerial image (image resolution: 0.5 m/pixel).

We used this data set to compare our new algorithm with the approach of Triebel *et al.* [36], which iteratively applies *LU* decomposition. In this experiment, both approaches converge to more or less the same solution. The time needed to achieve this correction, however, is by orders of magnitudes smaller when applying our new technique. Fig. 4 plots the average error per constraint versus the execution time.

B. Quantitative Results and Comparison With SAM in 3-D

The second set of experiments is designed to measure the performance of our approach for correcting 3-D constraint networks and in comparison with the smoothing and mapping (SAM) approach of Dellaert [4]. In these simulation experiments, we moved a virtual robot on the surface of a sphere. An observation was 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 Gaussian noise to investigate the robustness of the algorithms.

Fig. 5 depicts a series of graphs obtained by our algorithm using three data sets generated with different noise levels. The observation and motion noise was set to $\sigma = 0.05/0.1/0.2$ in each translational component (in meters) and rotational component (in radians).

As can be seen, our approach converges to a configuration with a low error. In particular, for the last data set, the rotational noise with a standard deviation of 0.2 (in radians) for each movement and observation is high. After around 250 iterations, the system converged. Each iteration took 200 ms for this data set with around 85 000 constraints.

We furthermore compared our approach with the SAM approach of Dellaert [4]. The SAM algorithm can operate in two modes: as a batch process, which optimizes the entire network at once, or in an incremental mode. The latter only performs an optimization after a fixed number of nodes have been added. This way of incrementally optimizing the network is more robust since the initial guess for the network configurations is computed based on the result of the previous optimization. As a result, the risk of getting stuck in local minima is typically reduced. However, this procedure leads to a significant computational overhead. Table I summarizes the results obtained with the SAM algorithm. As can be seen, the batch variant of the SAM algorithm got stuck in local minima for the sphere data sets with medium and large noise. The incremental version, in contrast, always converged but still required substantially more computation time than our current implementation of our approach.

C. Comparison to MLR and Olson's Algorithm in 2-D

In this third experiment, we compare our technique with two current state-of-the-art SLAM approaches that aim to correct constraint networks, namely, MLR proposed in the Frese *et al.* [11] and Olson algorithms [27]. Since both techniques are designed for 2-D scenarios, we also used the 2-D version of our system, which is identical to the 3-D version, except that the three additional dimensions (z, roll, pitch) are not considered.

We furthermore tested two variants of our method: one that uses the node reduction technique described in Section VI and one that maintains all the nodes in the graph.

In these simulation experiments, we moved a virtual robot on a grid world. Again, we corrupted the observations with a variable amount of noise for testing the robustness of the algorithms. We simulated different data sets, resulting in graphs that consisted of 4000 and 2 000 000 constraints.

Fig. 6 depicts the actual graphs obtained by Olson's algorithm and our approach for different time steps. 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 depend on the initial positions of the nodes. We found that, in the case of a good starting configuration of the network, MLR converges to a highly accurate solution similar to our approach (see Fig. 7, left). Otherwise, it is likely to diverge (Fig. 7, right). Olson's approach and our technique are more or less independent of the initial poses of the nodes.

To quantitatively evaluate our technique, we measured the error in the network after each iteration. Fig. 8 (left) depicts a

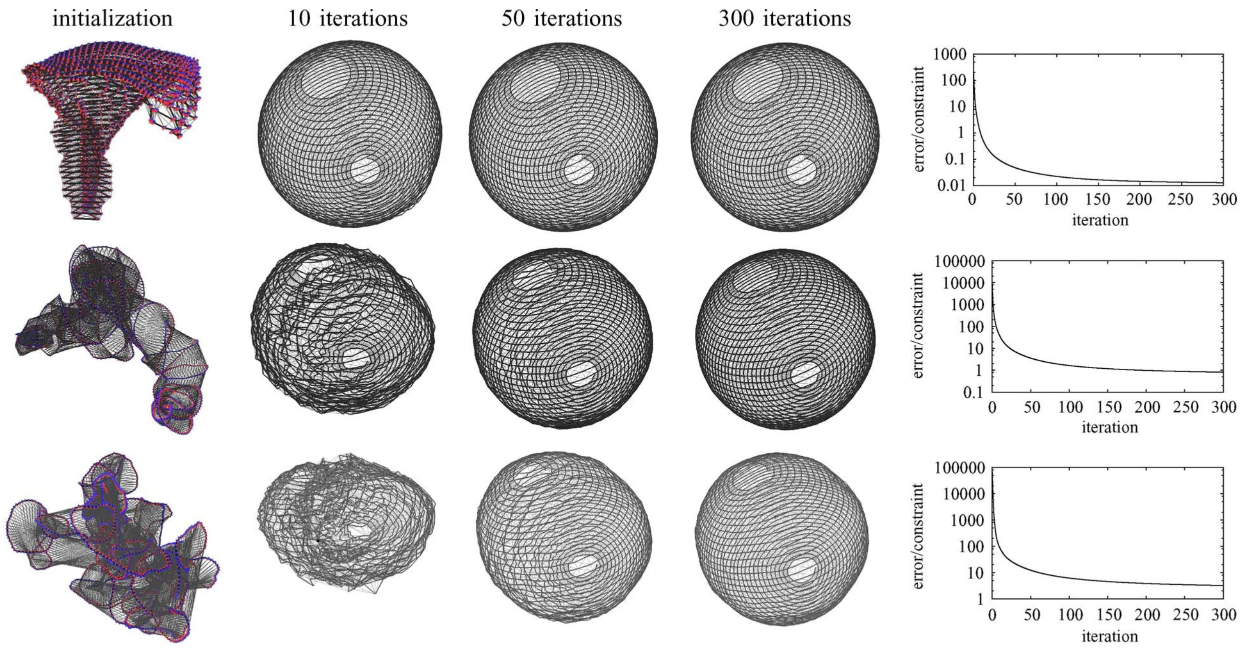

Fig. 5. Results obtained by our approach using a virtual robot moving on a sphere with three different noise realizations in motion and observations (row 1: $\sigma = 0.05$; row 2: $\sigma = 0.1$; row 3: $\sigma = 0.2$). Each network consists of around 85 000 constraints. The error is computed according to (7) divided by the number of constraints.

TABLE I
COMPARISON WITH SAM

noise level	SAM (batch)	SAM (incremental)	Our method (batch)
$\sigma = 0.05$	119 s	not tested (see batch)	20 s (100 iterations)
$\sigma = 0.1$	diverged	270 s (optimized each 100 nodes)	40 s (200 iterations)
$\sigma = 0.2$	diverged	510 s (optimized each 50 nodes)	50 s (250 iterations)

statistical experiment over ten 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-sized networks, both approaches asymptotically converge to approximatively the same error value (see Fig. 8, middle). For large networks, the high number of iterations needed for Olson's approach prevented us from experimentally analyzing the convergence. For the sake of brevity, we omitted comparisons with the EKF and Gauss–Seidel relaxation because Olson *et al.* already showed that their approach outperforms those techniques.

Additionally, we evaluated the average computation time per iteration of the different approaches (see Fig. 8, right) and analyzed a variant of Olson's approach, which is restricted to spherical covariances. The latter approach yields execution times *per iteration* similar to our algorithm. However, this variant has still the same convergence speed with respect to the number of iterations as Olson's original technique. As can be seen in Fig. 8 (right), our node-reduction technique speeds up the computations up to a factor of 20.

We also applied our 3-D optimizer to such 2-D problems and compared its performance with our 2-D version. Both techniques lead to more or less the same results. The 2-D version, however, is around three times faster than the 3-D version. This results from removing the irrelevant components from the state space and thus avoids the corresponding trigonometric operations.

D. Error Distribution in 3-D

We furthermore compared our technique to distribute a rotational error in 3-D with our previously proposed method [12]. Compared with this method, our new distribution limits the fraction of the error that is added to the intermediate nodes—a bound that is not available in [12]. Without this bound, it can happen that the error of the overall network drastically increases because a high error is introduced in the intermediate nodes. Note that, even if this effect rarely occurs in real data sets, it can lead to divergence. Fig. 9 illustrates such an example recorded with a car in a parking lot with three floors.

While the previous method diverges after a few iterations, our new algorithm leads to a limited and balanced distribution of the error. This results in a more stable algorithm, which successfully solved all tested data sets.

E. Constraint Sampling

SGD selects in each iteration a random order in which the constraints are updated. In our previous work [13], we neglected this randomization and selected a fixed order based on the level of a constraint in the tree. This was needed to perform efficient updates, given our previously presented parameterization of the nodes.

With the parameterization presented in this paper, we are free to choose an arbitrary order. We therefore compared two different sampling techniques: 1) random sampling and 2) a variant in which we sample a constraint without replacement with a probability inversely proportional to the path length. We figured

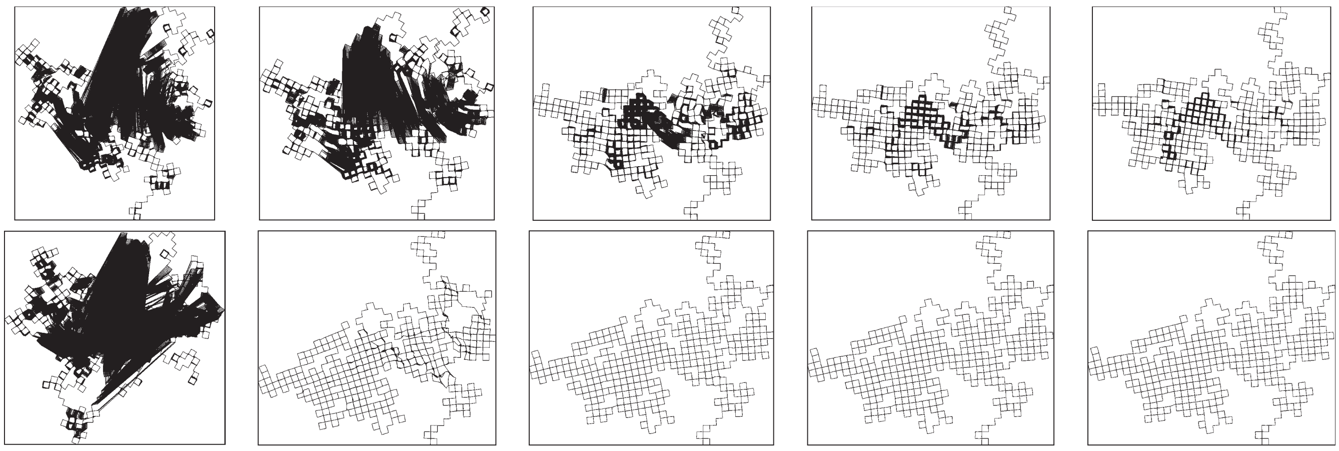
Fig. 6. Results obtained with (first row) Olson's algorithm and (second row) our approach after one, 10, 50, 100, and 300 iterations for a network with 64 000 constraints. The black areas in the images result from constraints between nodes that are not perfectly corrected after the corresponding iteration (for timings, see Fig. 8).

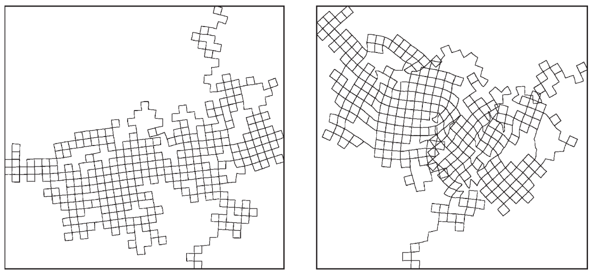
Fig. 7. The result of MLR strongly depends on the initial configuration of the network. (Left) Small initial pose error. (Right) Large initial pose error.

out that, in situations with nested loops, it is advantageous to first process the constraints that have a shorter path length (and thus correspond to the smaller loops). This is due to angular "wraparounds" that are more likely to occur when first correcting a large loop starting with a poor initial guess. A wraparound is an error in the initial guess of a relative configuration between two nodes that is bigger than 180°. Such wraparounds cause the algorithm to converge to a local minimum.

This effect can be observed in Fig. 10. It illustrates a statistical experiment carried out using the sphere data set (ten runs per strategy). As can be seen, sampling the constraints in each iteration inversely proportional to the length of their path in the tree gives the best results. In contrast to this, getting stuck in local minima is more likely when performing random sampling. Note that this effect only occurs for large networks or high noise in the rotational components. Otherwise, both strategies provide comparable results. As a result, we sample without replacement the constraints in each iteration inversely proportional to the length of the their path in the parameterization tree.

VIII. Related Work

Mapping techniques for mobile robots can be classified according to the underlying estimation technique. The most popular approaches are EKFs [21], [31], sparse extended information filters (SEIFs) [7], [35], particle filters [23], and least-squares error minimization approaches [11], [14], [22]. For some applications, it might be even be sufficient to learn local maps only [15], [34], [39].

The effectiveness of the EKF approaches comes from the fact that they estimate a fully correlated posterior about landmark maps and robot poses [21], [31]. Their weakness lies in the strong assumptions that have to be made on both the robot motion model and the sensor noise. If these assumptions are violated, the filter is likely to diverge [18], [37].

Thrun *et al.* [35] proposed a method to correct the poses of a robot based on the inverse of the covariance matrix. The advantage of SEIFs is that they make use of the approximative sparsity of the information matrix. Eustice *et al.* [7] presented a technique that more accurately computes the error bounds within the SEIF framework and therefore reduces the risk of becoming overly confident.

Recently, Dellaert *et al.* have proposed a smoothing method called square root SAM [4], [19], [30]. It has several advantages compared with EKF-based solutions since it better covers the nonlinearities and is faster to compute. In contrast to SEIFs, it furthermore provides an exactly sparse factorization of the information matrix. In addition to that, SAM can be applied in an incremental way [19] and is able to learn maps in 2-D and 3-D. Paskin [28] presented a solution to the SLAM problem using thin junction trees. This way, Paskin is able to reduce the complexity compared with the EKF approaches since thin junction trees provide a linear-time filtering operation.

Frese's TreeMap algorithm [9] 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. This way, Frese is able to perform an update in $\mathcal{O}(\log n)$, where n is the number of features.

An alternative approach to find ML maps is the application of least-squares 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 [22] first applied this approach in robotics to address the SLAM problem using a kind of brute-force method. Their approach seeks to optimize

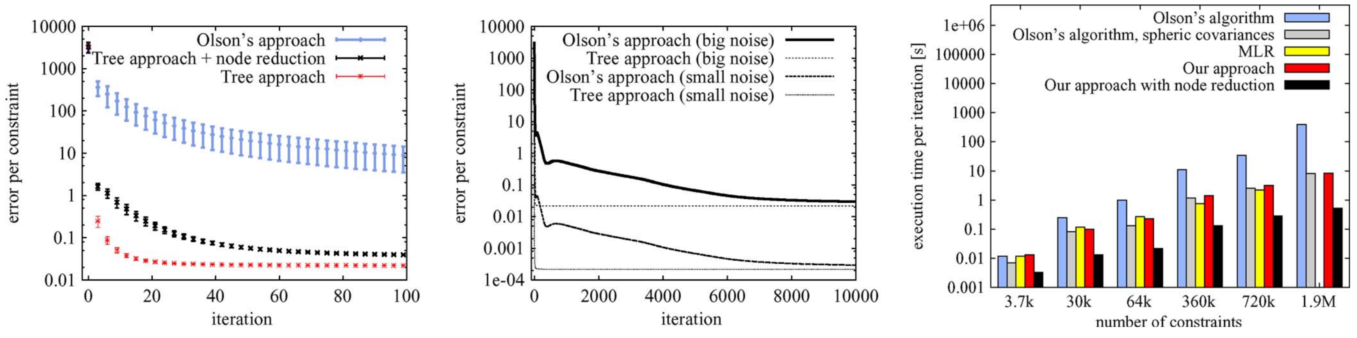

Fig. 8. (Left) Error of our approach and Olson's approach in a statistical experiment ($\sigma = 0.05$ confidence). (Middle) Both techniques asymptotically converge to the same error. (Right) Average execution time *per iteration* for different networks. For the network consisting of 1 900 000 constraints, the execution of MLR required too many resources. The result is therefore omitted. The error is computed according to (7) divided by the number of constraints.

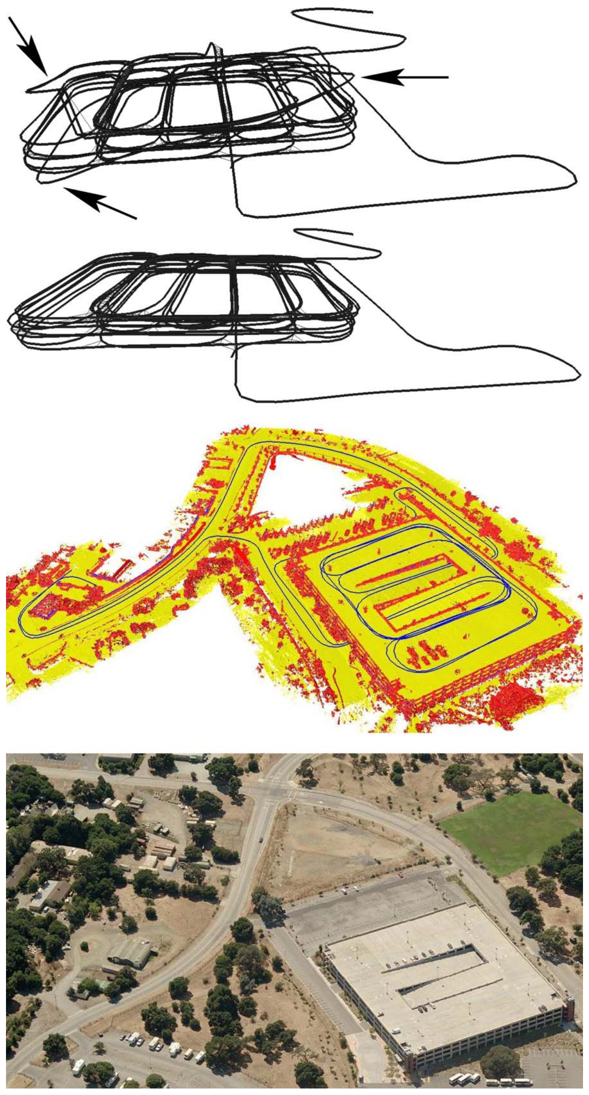

Fig. 9. Network obtained from a car driving multiple times through a parking lot with three floors. Different error-distribution techniques result in different networks. The inconsistencies are marked by the arrows. (First row) Previous method [12]. (Second row) Our approach, both after three iterations of the optimizer. (Third row) Multilevel surface map created from the corrected constraint network. (Fourth row) Aerial image of the parking lot.

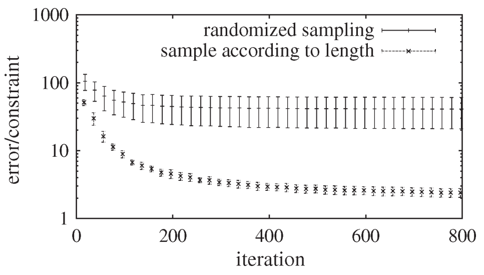

Fig. 10. Evolution of the error per constraint in a statistical experiment using different strategies to sample the constraint that is to be updated next. The error is computed according to (7) divided by the number of constraints.

the whole network at once. Gutmann and Konolige [14] proposed an effective way to construct such a network and to detect loop closures while running an incremental estimation algorithm. Howard *et al.* [16] apply relaxation to localize the robot and build a map. Duckett *et al.* [5] propose the usage of Gauss–Seidel relaxation to minimize the error in the network of constraints. To make the problem linear, they assume knowledge about the orientation of the robot. Frese *et al.* [11] propose a variant of Gauss–Seidel relaxation called MLR. It applies relaxation at different resolutions. MLR is reported to provide very good results in flat environments, particularly if the error in the initial guess is limited.

Note that techniques such as Olson's algorithm, MLR, or our method focus on computing the best map and assume that the constraints are given. The ATLAS framework [2], hierarchical SLAM [6], or the work of Nüchter *et al.* [26], for example, can be used to obtain the data associations (constraints). They also apply a global optimization procedure to compute a consistent map. One can replace their optimization procedures by our algorithm and, in this way, make them more efficient.

A technique that combines 2-D pose estimates with 3-D data has been proposed by Howard *et al.* [17] to build maps of urban environments. They avoid the problem of distributing the error in all three dimensions by correcting only the orientation in the x and y planes of the vehicle. The roll and pitch are assumed to be measured accurately enough by an IMU.

In the context of 3-D ML mapping, only a few approaches have been presented so far [24]–[26], [36]. The approach of Nüchter *et al.* [26] describes a mobile robot that builds accurate 3-D models. In their approach, loop closing is achieved by uniformly distributing the error resulting from odometry over the poses in a loop. This technique provides good estimates but cannot deal with multiple/nested loops.

Montemerlo and Thrun [24] proposed to utilize the conjugate gradients to efficiently invert the sparse information matrix of the system. Their approach was used to learn large campus maps using a Segway robot. Recently, Triebel *et al.* [36] described an approach that aims to globally correct the poses given the network of constraints in all six dimensions. At each iteration, the problem is linearized and solved using LU decomposition. This yields accurate results for small- and medium-sized networks, particularly when the error in the rotational component is small. As illustrated in our experimental section, this approach is orders of magnitudes slower than our method and is thus not suited to learn maps of large scenes.

The approach closest to the work presented here is the work of Olson *et al.* [27]. They apply SGD to reduce the error in the network. 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. Furthermore, our approach allows us to avoid adding new nodes and constraints to the graph when revisiting areas that have already been mapped. As a result, the complexity of our algorithm only depends on the size of the environment and not on the length of the trajectory traveled by the robot. This is an advantage compared with approaches such as MLR or Olson's algorithm since it allows for lifelong map learning.

The work presented in this paper furthermore extends two previous conference publications [12], [13]. The first one [13] is only applicable to 2-D scenarios and uses a different parameterization of the nodes. The second one is an extension to 3-D [12]. It allows a robot to distribute a rotational error over a sequence of poses. This distribution, however, was not bounded, as was the one presented in this paper. As demonstrated in the experimental section, the previous error-distribution approach more often leads to divergence.

IX. Conclusion

In this paper, we have presented a highly efficient solution to the problem of learning 2-D and 3-D ML maps for mobile robots. Our technique is based on the graph formulation of the SLAM problem and applies a variant of SGD. Our approach extends an existing 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. 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 for lifelong map learning. Additionally, we have presented a way to accurately distribute a 3-D rotational error over a sequence of poses, which increases the robustness over previous approaches.

Our method has been implemented and exhaustively tested in simulation experiments and on real robot data. We furthermore compared our method to three existing state-of-the-art algorithms. The experiments demonstrate that our method converges faster and yields more accurate maps than the other approaches.

Acknowledgment

The authors would like to thank the following: U. Frese for his insightful comments and for providing them with his MLR implementation for comparisons; E. Olson for fruitful discussions; M. Kaess and F. Dellaert for carrying out the experiments with their SAM/iSAM implementation; S. Grzonka for his valuable input on the slerp interpolation used in this paper and for his support while carrying out experiments; D. Hähnel and R. Kümmerle for providing them with the parking lot data set recorded with Stanford's autonomous car Junior; and R. Siegwart and his laboratory at EPFL and the Swiss Federal Institute of Technology (ETH), Zürich, for financial and technical support while working with the Smart car.

References

[1] T. Barrera, A. Hast, and E. Bengtsson, "Incremental spherical linear interpolation," in *Proc. SIGRAD*, 2004, vol. 13, pp. 7–13.

[2] M. Bosse, P. M. Newman, J. J. Leonard, and S. Teller, "An Atlas framework for scalable mapping," in *Proc. IEEE ICRA*, Taipei, Taiwan, 2003, pp. 1899–1906.

[3] D. Braid, A. Broggi, and G. Schmiedel, "The TerraMax autonomous vehicle," *J. Robot. Syst.*, vol. 23, no. 9, pp. 693–708, Sep. 2006.

[4] F. Dellaert, "Square root SAM," in *Proc. RSS*, Cambridge, MA, 2005, pp. 177–184.

[5] T. Duckett, S. Marsland, and J. Shapiro, "Fast, on-line learning of globally consistent maps," *J. Auton. Robots*, vol. 12, no. 3, pp. 287–300, May 2002.

[6] C. Estrada, J. Neira, and J. D. Tardós, "Hierarchical slam: Real-time accurate mapping of large environments," *IEEE Trans. Robot.*, vol. 21, no. 4, pp. 588–596, Aug. 2005.

[7] R. Eustice, H. Singh, and J. J. Leonard, "Exactly sparse delayed-state filters," in *Proc. IEEE ICRA*, Barcelona, Spain, 2005, pp. 2428–2435.

[8] J. Folkesson and H. Christensen, "Graphical slam—A self-correcting map," in *Proc. IEEE ICRA*, Orlando, FL, 2004, pp. 383–390.

[9] U. Frese, "Treemap: An $o(\log n)$ algorithm for indoor simultaneous localization and mapping," *J. Auton. Robots*, vol. 21, no. 2, pp. 103–122, Sep. 2006.

[10] U. Frese and G. Hirzinger, "Simultaneous localization and mapping—A discussion," in *Proc. IJCAI Workshop Reasoning Uncertainty Robot.*, Seattle, WA, 2001, pp. 17–26.

[11] U. Frese, P. Larsson, and T. Duckett, "A multilevel relaxation algorithm for simultaneous localization and mapping," *IEEE Trans. Robot.*, vol. 21, no. 2, pp. 1–12, Apr. 2005.

[12] G. Grisetti, S. Grzonka, C. Stachniss, P. Pfaff, and W. Burgard, "Efficient estimation of accurate maximum likelihood maps in 3D," in *Proc. IEEE/RSJ IROS*, San Diego, CA, 2007, pp. 3472–3478.

[13] G. Grisetti, C. Stachniss, S. Grzonka, and W. Burgard, "A tree parameterization for efficiently computing maximum likelihood maps using gradient descent," in *Proc. RSS*, Atlanta, GA, 2007.

[14] J.-S. Gutmann and K. Konolige, "Incremental mapping of large cyclic environments," in *Proc. IEEE Int. Symp. CIRA*, Monterey, CA, 1999, pp. 318–325.

[15] J. Hermosillo, C. Pradalier, S. Sekhavat, C. Laugier, and G. Baille, "Towards motion autonomy of a bi-steerable car: Experimental issues from map-building to trajectory execution," in *Proc. IEEE ICRA*, 2003, pp. 2430–2435.

[16] A. Howard, M. J. Matarić, and G. Sukhatme, "Relaxation on a mesh: A formalism for generalized localization," in *Proc. IEEE/RSJ IROS*, 2001, pp. 1055–1060.

[17] A. Howard, D. F. Wolf, and G. S. Sukhatme, "Towards 3D mapping in large urban environments," in *Proc. IEEE/RSJ IROS*, 2004, pp. 419–424.

[18] S. Julier, J. Uhlmann, and H. Durrant-Whyte, "A new approach for filtering nonlinear systems," in *Proc. Amer. Control Conf.*, Seattle, WA, 1995, pp. 1628–1632.

[19] M. Kaess, A. Ranganathan, and F. Dellaert, "iSAM: Fast incremental smoothing and mapping with efficient data association," in *Proc. IEEE ICRA*, Rome, Italy, 2007, pp. 1670–1677.

[20] J. Ko, B. Stewart, D. Fox, K. Konolige, and B. Limketkai, "A practical, decision-theoretic approach to multi-robot mapping and exploration," in *Proc. IEEE/RSJ IROS*, Las Vegas, NV, 2003, pp. 3232–3238.

[21] J. J. Leonard and H. F. Durrant-Whyte, “Mobile robot localization by tracking geometric beacons,” *IEEE Trans. Robot. Autom.*, vol. 7, no. 3, pp. 376–382, Jun. 1991.

[22] F. Lu and E. Milios, “Globally consistent range scan alignment for environment mapping,” *J. Auton. Robots*, vol. 4, no. 4, pp. 333–349, Oct. 1997.

[23] M. Montemerlo, S. Thrun, D. Koller, and B. Wegbreit, “FastSLAM 2.0: An improved particle filtering algorithm for simultaneous localization and mapping that provably converges,” in *Proc. IJCAI*, Acapulco, Mexico, 2003, pp. 1151–1156.

[24] M. Montemerlo and S. Thrun, “Large-scale robotic 3-D mapping of urban structures,” in *Proc. ISER*, 2004, pp. 141–150.

[25] P. Newman, D. Cole, and K. Ho, “Outdoor slam using visual appearance and laser ranging,” in *Proc. IEEE ICRA*, Orlando, FL, 2006, pp. 1180–1187.

[26] A. Nüchter, K. Lingemann, J. Hertzberg, and H. Surmann, “6d SLAM with approximate data association,” in *Proc. 12th ICAR*, 2005, pp. 242–249.

[27] E. Olson, J. Leonard, and S. Teller, “Fast iterative optimization of pose graphs with poor initial estimates,” in *Proc. IEEE ICRA*, 2006, pp. 2262–2269.

[28] M. A. Paskin, “Thin junction tree filters for simultaneous localization and mapping,” in *Proc. IJCAI*, Acapulco, Mexico, 2003, pp. 1157–1164.

[29] P. Pfaff, R. Triebel, C. Stachniss, P. Lamon, W. Burgard, and R. Siegwart, “Towards mapping of cities,” in *Proc. IEEE ICRA*, Rome, Italy, 2007, pp. 4807–4813. under review.

[30] A. Ranganathan, M. Kaess, and F. Dellaert, “Loopy SAM,” in *Proc. IJCAI*, 2007, pp. 2191–2196.

[31] R. Smith, M. Self, and P. Cheeseman, “Estimating uncertain spatial relationships in robotics,” in *Autonomous Robot Vehicles*, I. Cox and G. Wilfong, Eds. New York: Springer-Verlag, 1990, pp. 167–193.

[32] C. Stachniss and G. Grisetti, *TORO project at OpenSLAM.org*, 2007. [Online]. Available: http://openslam.org/toro.html

[33] B. Steder, G. Grisetti, S. Grzonka, C. Stachniss, A. Rottmann, and W. Burgard, “Learning maps in 3D using attitude and noisy vision sensors,” in *Proc. IEEE/RSJ IROS*, San Diego, CA, 2007, pp. 644–649.

[34] S. Thrun, M. Montemerlo, H. Dahlkamp, D. Stavens, A. Aron, J. Diebel, P. Fong, J. Gale, M. Halpenny, G. Hoffmann, K. Lau, C. M. Oakley, M. Palatucci, V. Pratt, P. Stang, S. Strohband, C. Dupont, L.-E. Jendrossek, C. Koelen, C. Markey, C. Rummel, J. van Niekerk, E. Jensen, P. Alessandrini, G. R. Bradski, R. Davies, S. Ettinger, A. Kaehler, A. V. Nefian, and P. Mahoney, “Stanley: The robot that won the DARPA Grand Challenge,” *J. Field Robot.*, vol. 23, no. 9, pp. 661–692, Sep. 2009.

[35] S. Thrun, Y. Liu, D. Koller, A. Y. Ng, Z. Ghahramani, and H. Durrant-Whyte, “Simultaneous localization and mapping with sparse extended information filters,” *Int. J. Robot. Res.*, vol. 23, no. 7/8, pp. 693–716, 2004.

[36] R. Triebel, P. Pfaff, and W. Burgard, “Multi-level surface maps for outdoor terrain mapping and loop closing,” in *Proc. IEEE/RSJ IROS*, 2006, pp. 2276–2282.

[37] J. Uhlmann, “Dynamic map building and localization: New theoretical foundations,” Ph.D. dissertation, Univ. Oxford, Oxford, U.K., 1995.

[38] C. Urmson, “Navigation regimes for off-road autonomy,” Ph.D. dissertation, Robot. Inst., Carnegie Mellon Univ., Pittsburgh, PA, 2005.

[39] M. Yguel, C. T. M. Keat, C. Braillon, C. Laugier, and O. Aycard, “Dense mapping for range sensors: Efficient algorithms and sparse representations,” in *Proc. RSS*, Atlanta, GA, 2007.

Giorgio Grisetti received the Ph.D. degree from the University of Rome “La Sapienza,” Rome, Italy, in 2006.

He is currently a Postdoctoral Researcher with the Laboratory for Autonomous Intelligent Systems, Department of Computer Science, University of Freiburg, Freiburg, Germany. His previous and current contributions in robotics aim to provide effective solutions to various mobile robot navigation problems, including simultaneous localization and mapping, localization, and path planning. His research interests are in the areas of mobile robotics.

Cyrill Stachniss received the Ph.D. degree in computer science from the University of Freiburg, Freiburg, Germany, in 2006.

After receiving the Ph.D. degree, he joined the Swiss Federal Institute of Technology (ETH) Zurich, Zurich, Switzerland, as a Senior Researcher. Since 2007, he has been an Academic Advisor with the Laboratory for Autonomous Intelligent Systems, Department of Computer Science, University of Freiburg. His research interests are in the areas of robot navigation, exploration, simultaneous localization and mapping, and learning approaches.

Wolfram Burgard received the Ph.D. degree in computer science from the University of Bonn, Bonn, Germany, in 1991.

He is currently a Professor of computer science with the University of Freiburg, Freiburg, Germany, where he heads the Laboratory for Autonomous Intelligent Systems, Department of Computer Science. His areas of interest are in artificial intelligence and mobile robots. In the past, he and his group developed several innovative probabilistic techniques for robot navigation and control. They cover different aspects such as localization, map building, path planning, and exploration.

Prof. Burgard is a Fellow of the European Coordinating Committee for Artificial Intelligence. He has been the recipient of several best paper awards from outstanding national and international conferences.