Efficient Estimation of Accurate Maximum Likelihood Maps in 3D

Giorgio Grisetti Slawomir Grzonka Cyrill Stachniss Patrick Pfaff Wolfram Burgard

Abstract— **Learning maps is one of the fundamental tasks of mobile robots. 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 consider the problem of learning maps with mobile robots that operate in non-flat environments and apply maximum likelihood techniques to solve the graph-based SLAM problem. Due to the non-commutativity of the rotational angles in 3D, major problems arise when applying approaches designed for the two-dimensional world. The non-commutativity introduces serious difficulties when distributing a rotational error over a sequence of poses. In this paper, we present an efficient solution to the SLAM problem that is able to distribute a rotational error over a sequence of nodes. Our approach applies a variant of gradient descent to solve the error minimization problem. We implemented our technique and tested it on large simulated and real world datasets. We furthermore compared our approach to solving the problem by LU-decomposition. As the experiments illustrate, our technique converges significantly faster to an accurate map with low error and is able to correct maps with bigger noise than existing methods.**

I. Introduction

Learning maps has been a major research focus in the robotics community over the last decades and 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. 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. Constraints between poses resulting from observations or from odometry are encoded in the edges between the nodes. The goal of algorithms to solve this problem is to find a configuration of the nodes that maximizes the observation likelihood encoded in the constraints.

In the past, this concept has been successfully applied [3], [4], [7], [8], [9], [10], [12], [13], [15]. Such solutions apply an iterative error minimization techniques. They correct either all poses simultaneously [7], [9], [10], [15] or perform local updates [3], [4], [8], [13]. Most approaches have been designed for the two-dimensional space where the robot is assumed to operate on a plane [3], [4], [7], [10], [13]. Among all these approaches, multi-level relaxation [4] or Olson's algorithm [13] belong to the most efficient ones.

In the three-dimensional space, however, distributing an error between different nodes of a network is not straightforward. One reason for that is the non-commutativity of the three rotational angles. As a result, most approaches that provide good results in 2D are not directly applicable in 3D. One way is to ignore the non-commutativity of the rotational angles. In this case, however, the algorithm works only in case of small noise and in small environments. A few maximum likelihood mapping techniques have been proposed for the three-dimensional space [9], [12], [15]. Some approaches ignore the error in pitch and roll [9] whereas others detect loops and divide the error by the number of poses along the loop (weighted with path length, as in [12]). An alternative solution is to apply variants of the approach of Lu and Milios [10] and to correct the whole network at once [15].

All authors are members of the University of Freiburg, Department of Computer Science, 79110 Freiburg, Germany

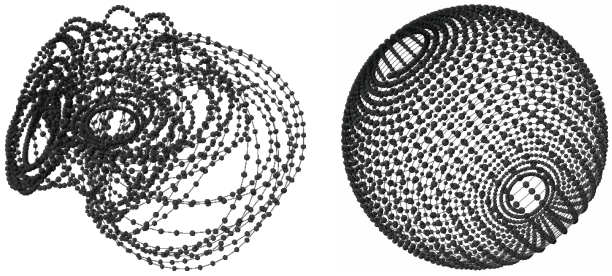

Fig. 1. A simulated trajectory of a robot moving on the surface of a sphere. The left image shows an uncorrected trajectory and the right image depicts the corrected one (approx. 8,600 constraints, 100 iterations, 21s).

The contribution of this paper is a technique to efficiently distribute the error over a sequence of nodes in all six dimensions (x, y, z, and the three rotational angles ϕ, θ, ψ). This enables us to apply a variant of gradient descent in order to reduce the error in the network. As a result, our approach converges by orders of magnitudes faster than the approaches mentioned above to low error configurations. As a motivating example, consider Figure 1. It depicts a trajectory of a simulated robot moving on the surface of a sphere. The left image depicts the input data and the right one the result of the technique presented in this paper.

The remainder of this paper is organized as follows. After discussing related work, we explain in Section III the graph-based formulation of the mapping problem as well as the key ideas of gradient descent in Section IV. Section V explains why the standard 2D approach cannot be used in 3D and introduces our technique to correct the poses given a network of constraints. Section VI analyzes the complexity of our approach. We finally present our experimental results in Section VII.

II. Related Work

A popular approach to find maximum likelihood (ML) maps is to apply least square error minimization techniques based on a network of relations. In this paper, we also

follow this way of describing the SLAM problem. Lu and Milios [10] 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 [7] proposed an effective way for constructing such a network and for detecting loop closures while running an incremental estimation algorithm. Howard *et al.* [8] apply relaxation to localize the robot and to build a map. Duckett *et al.* [3] 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.* [4] propose a variant called multi-level relaxation (MLR). It applies relaxation based on different resolutions. Recently, Olson *et al.* [13] presented a novel method for correction two-dimensional networks using (stochastic) gradient descent. Olson's algorithm and MLR are currently the most efficient techniques available in 2D. All techniques discussed so far have been presented as solutions to the SLAM problem in the two-dimensional space. As we will illustrate in this paper, they typically fail to correct a network in 3D.

Dellaert proposed a smoothing method called square root smoothing and mapping [2]. It applies smoothing to correct the poses of the robot and feature locations. It is one of the few techniques that can be applied in 2D as well as in 3D. A technique that combines 2D pose estimates with 3D data has been proposed by Howard *et al.* [9] 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, y-plane of the vehicle. The roll and pitch is assumed to be measured accurately enough using an IMU.

In the context of three-dimensional maximum likelihood mapping, only a few approaches have been presented so far [11], [12], [15]. The approach of Nüchter *et al.* [12] describes a mobile robot that builds accurate three-dimensional 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 typically requires a small error in the roll and pitch estimate. Newman *et al.* [11] presented a sophisticated approach for detecting loop closures using laser and vision. Such an approach can be used to find the constraints which are the input to our algorithm.

Recently, Triebel *et al.* [15] described an approach that aims to globally correct the poses given the network of constraints in all three dimensions. At each iteration the problem is linearized and solved using LU decomposition. This yields accurate results for small and medium size networks especially when the error in the rotational component is small. We use this approach as a benchmark for our technique presented in this paper.

The contribution of this paper is a highly efficient technique to compute maximum likelihood maps in 3D. We present a way of distributing an error in all three rotational angles that accounts for the non-commutativity of these angles. This technique in combination with a variant of gradient descent allows us to correct larger networks than most state-of-the-art approaches.

III. On Graph-based SLAM

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

- $\mathbf{x}$ is a vector of parameters $(x_1 \ \cdots \ x_n)^T$ which describes a configuration of the nodes.
- δ_{ji} represents a constraint between the nodes i and j based on measurements. These constraints are the edges in the graph structure.
- Ω_{ji} is the information matrix capturing the uncertainty of δ_{ji}.
- $f_{ji}(\mathbf{x})$ is a function that computes a zero noise observation according to the current configuration of nodes. It returns an observation of node j from node i.

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

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

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

$$F_{ji}(\mathbf{x}) = \frac{1}{2} \left(f_{ji}(\mathbf{x}) - \delta_{ji}\right)^T \Omega_{ji} \left(f_{ji}(\mathbf{x}) - \delta_{ji}\right) \tag{2}$$

$$\propto r_{ji}(\mathbf{x})^T \Omega_{ji} r_{ji}(\mathbf{x}). \tag{3}$$

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

$$F(\mathbf{x}) = \frac{1}{2} \sum_{<j,i> \in \mathcal{C}} r_{ji}(\mathbf{x})^T \Omega_{ji} r_{ji}(\mathbf{x}) \tag{4}$$

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

A maximum likelihood map learning approach seeks to find the configuration $\mathbf{x}^*$ of the nodes that maximizes the likelihood of the observations which is equivalent to minimizing the negative log likelihood written as

$$\mathbf{x}^* = \operatorname*{argmin}_{\mathbf{x}} F(\mathbf{x}). \tag{5}$$

IV. Gradient Descent for Maximum Likelihood Mapping

Gradient descent (GD) is an iterative technique to find the minimum of a function. Olson *et al.* [13] were the first who applied it in the context of the SLAM problem in the two-dimensional space. GD seeks for a solution of Eq. (5) by iteratively selecting a constraint $< j, i >$ and by moving a set of nodes of the network in order to decrease the error introduced by the selected constraint. The nodes are updated according to the following equation:

$$\mathbf{x}^{t+1} = \mathbf{x}^t + \underbrace{\lambda \cdot J_{ji}^T \Omega_{ji} r_{ji}}_{\Delta \mathbf{x}} \tag{6}$$

Here $\mathbf{x}$ is the set of variables describing the locations of the poses in the network. 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 Δx of Eq. (6) from right to left gives an intuition about the iterative procedure used in GD:

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

In practice, GD decomposes the overall problem into many smaller problems by optimizing the constraints individually. The difference between GD and stochastic GD is that the stochastic variant selects the constraints in a random order. 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 subproblems to asymptotically converge towards an equilibrium point that is the solution found by the algorithm. This equilibrium point is then reported as the maximum liklihood solution to the mapping problem.

V. 3D Graph Optimization

The graph-based formulation of the SLAM problem does not specify how the poses are presented in the nodes of the graph. In theory, one can choose an arbitrary parameterization. Our algorithm uses a tree based parameterization for describing the configuration of the nodes in the graph. To obtain such a tree from an arbitrary graph, one can compute a spanning tree. The root of the spanning tree is the node at the origin p_0. Another possibility is to construct a graph based on the trajectory of the robot in case this is available. 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.
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.

In the following, we describe how to use this tree to define the parameterization of the nodes in the network. Each node i in the tree is related to a pose p_i in the network and maintains a parameter x_i which is a 6D vector that describes its configuration. Note that the parameter x_i can be different from the pose p_i. In our approach, the parameter x_i is chosen as the relative movement from the parent of the node i in the tree to the node i itself

$$x_i = p_i \ominus p_{\text{parent}(i)}, \tag{7}$$

with $x_0 = p_0$. The operator $\ominus$ is the motion decomposition operator in 3D which is analogous to the one defined in 2D (see Lu and Milios [10]). A detailed discussion on tree parameterizations in combination with GD is out of the scope of this document and we refer the reader to [6].

Fig. 2. A simple example that illustrates the problem of distributing the error in 3D. The left image shows the input data which was obtained by moving a simulated robot over a hexagon twice with small Gaussian noise. The middle image show the result obtained if the non-commutativity of the rotation angles is ignored. The right images shows the result of our approach which is very close to the ground truth.

Before presenting our approach for correcting the poses in a network, we want to illustrate the problem of distributing an error over a sequence of nodes. Consider that we need to distribute an error e over a sequence of n nodes. In the two-dimensional space, this can be done in a straightforward manner as follows. Given the residual $r^{2D} = (r_x, r_y, r_\theta)$, we can simply change the pose of the i-th node in the chain by i/n times r^{2D}. This error propagation works well in 2D and is performed in most maximum-likelihood methods in the 2D space. In the three-dimensional space, however, such a technique is not applicable (with exception of very small errors). The reason for that is the non-commutativity of the three rotations

$$R(\phi, \theta, \psi) \quad \neq \quad \prod_1^n R(\frac{\phi}{n}, \frac{\theta}{n}, \frac{\psi}{n}), \tag{8}$$

where $R(\phi, \theta, \psi)$ is the three-dimensional rotation matrix. As illustrated in Figure 2, applying such an error propagation leads to divergence even for small and simple problems. Therefore, one has to find a different way of distributing the error over a chain of poses which is described in the following.

A. The Error Introduced by a Constraint

Let P_i be the homogenous transformation matrix corresponding to the pose p_i of the node i and X_i the transformation matrix corresponding to the parameter x_i. Let $\mathcal{P}_{i,0}$ be the ordered list of nodes describing a path in the tree from the root (here referred to as node 0) to the node i. We can express the pose of a node as

$$P_i = \prod_{k \in \mathcal{P}_{i,0}} X_k. \tag{9}$$

The homogenous transformation matrix X_i consists of a rotational matrix R and a translational component t. It has

the following form

$$X_i = \begin{pmatrix} R_k & t_k \\ 0 & 1 \end{pmatrix} \text{ with } X_i^{-1} = \begin{pmatrix} R_k^T & -R_k^T t_k \\ 0 & 1 \end{pmatrix} \tag{10}$$

In order to compute the transformation between two nodes i and j, one needs to consider the path $\mathcal{P}_{ji}$ from node i to node j. Since the nodes are arranged in a tree, this path consists of an ascending part and a descending part. Let $\mathcal{P}_{ji}^a$ be the ascending part of the path starting from node i and $\mathcal{P}_{ji}^d$ the descending part to node j. We can then compute the error e_{ji} in the reference frame of p_i as

$$e_{ji} = (p_j \ominus p_i) \ominus \delta_{ji}. \tag{11}$$

Using the matrix notation, the error is

$$E_{ji} = \Delta_{ji}^{-1} P_i^{-1} P_j \tag{12}$$

$$= \Delta_{ji}^{-1} \prod_{k^d \in \mathcal{P}_{ji}^d} X_{k^d}^{-1} \cdot \prod_{k^a \in \mathcal{P}_{ji}^a} X_{k^a}, \tag{13}$$

where Δ_{ji} is the matrix corresponding to δ_{ji}.

So far, we described the prerequisites for applying GD to correct the poses of a network. The goal of the update rule in GD is to iteratively update the configuration of a set of nodes in order to reduce the error introduced by a constraint. In Eq. (6), the term $J_{ji}^T \Omega_{ji}$ maps the variation of the error to a variation in the parameter space. This mapping is a linear function. As a result, the error might increase when applying GD in case of non-linear error surfaces. In the three-dimensional space, the rotational components often lead to highly non-linear error surfaces. Therefore, GD as well as similar minimization techniques cannot be applied directly to *large* mapping problems.

In our approach, we therefore chose a slightly different update rule. To overcome the problem explained above, we allow the usage of non-linear functions to describe the variation. The goal of this function is to compute a transformation of the nodes along the path in the tree so that the error introduced by the corresponding constraint is reduced. In detail, we design this function in a way so that it computes a new configuration of the variables $x_k \in \mathcal{P}_{ji}$ so that it corrects only a fraction λ of the error, where λ is the learning rate. 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. In our approach, this deformation is done in two steps. First, we update the rotational components R_k of the variables x_k and second, we update the translational components t_k.

B. Update of the Rotational Component

This section explains how to deform a path in order to reduce the error introduced by a constraint. Without loss of generality, we consider the origin of the path p_i 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 rotational matrices along the path as $\mathcal{R}_k$ neglecting the indices (compare Eq. (13)). The orientation of p_j is described by

$$\mathcal{R}_1 \mathcal{R}_2 \dots \mathcal{R}_n = \mathcal{R}_{1:n}, \tag{14}$$

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

Distributing a given error over a sequence of 3D 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 $\mathcal{R}_{1:n}B$ where B the matrix that rotates x_j to the desired orientation based on the error. Formulated in a mathematical way, we need to compute a set of rotations A_k so that

$$\mathcal{R}_{1:n} B = \prod_{k=1}^{n} \mathcal{R}_k A_k. \tag{15}$$

Once the matrices A_k are known, the new rotational matrices of the parameters x_k are updated by

$$\mathcal{R}_k \leftarrow \mathcal{R}_k A_k. \tag{16}$$

We can decompose the matrix B into a set of incremental rotations $B = B_{1:n}$. In our current implementation, we compute the individual matrices B_k by using the spherical linear interpolation (slerp) [1]. We can decompose B using the slerp function with a parameter $u \in [0, 1]$ with $slerp(B, 0) = I$ and $slerp(B, 1) = B$. According to this framework, we can compute the rotation B_k as

$$B_k = [slerp(B, u_{k-1})]^T \, slerp(B, u_k). \tag{17}$$

To determine the values u_{k-1} and u_k, we consider the eigenvalues of the covariances of the constraints connecting the nodes $k-1$ and k. This is an approximation which works well in 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.

Using this decomposition of B leads to Eq. (15) in which B is replaced by $B_{1:n}$. This equation admits an infinitive number of solutions. However, we are only interested in solutions which can be combined incrementally. Informally speaking, this means when truncating the path from n to $n-1$ nodes, the solution of the truncated path should be part of the solution of the full path. Formally, we can express this property by the following system of equations:

$$\forall_{k=1}^{n} : \quad R_1 A_1 \dots R_k A_k = R_{1:k} B_{1:k} \tag{18}$$

Given this set of equations, the solution for the matrices A_k can be computed as

$$A_k = R_k^T (B_{1:k-1})^T R_k B_{1:k}. \tag{19}$$

This is an exact solution that is always defined since A_k, R_k, and B_k are rotation matrices. The proof of Eq. (19) is given in the Section IX at the end of this document. Based on Eq. (16) and Eq. (19), we have a closed form solution for updating the rotational matrices of the parameters x_k along the path $\mathcal{P}_{ji}$ from the node i to the node j.

Note that we also use the slerp function to compute the fraction of the rotational component of the residual that is introduced by λ (see Section V-A).

For simplicity of presentation, we showed how to distribute the rotational error while keeping the 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 path in 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 a clearer formulation of the problem. In our implementation, however, we use quaternions for representing rotations because they are numerically more stable. In theory, however, both formulations are equivalent. An open source implementation is available [5].

C. Update of the Translational Component

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

We distribute the translational error by linearly moving the individual nodes along the path by a fraction of the error. This fraction depends in the uncertainty of the individual constraints encoded in the corresponding covariance matrices. Equivalent to the case when updating the rotational component, these fractions is also scaled with the learning rate.

VI. Computational Complexity

A single iteration of our algorithm requires to distribute the error introduced by the individual constraints over a set of nodes. Therefore, the complexity is proportional to the number of constraints times the number of operations needed to distribute the error of a single constraint.

In the remainder of this section, we analyze the number of operations needed to distribute the error of a single constraint. Once the poses of the nodes involved in an update step are known, the operations described in Section V-B and V-C can be carried out in a time proportional to the number of nodes $|\mathcal{P}|$ along the path $\mathcal{P}$. Computing the poses of the nodes along a path requires to traverse the tree up to the root according to Eq. (9). A naive implementation requires repeated traversals of the tree up to the root. This, however, can be avoided by choosing an intelligent order in which to process the constraints.

Let the "top node" of a path be the node with the smallest level in the tree. In our current implementation, we sort the constraints according to level of the corresponding top node. This can be done as a preprocessing step. We can process the constraints according to this order. The advantage of this order is that a constraint never modifies a node that has a smaller level in the tree. By storing the pose for each node in the tree, we therefore do not have to traverse the tree up to the root anymore. It is sufficient to access the parent of the top node in order to compute the poses for all nodes along a path $\mathcal{P}$. As a result, updating a constraint requires a time proportional to $|\mathcal{P}|$ and the overall complexity per iteration turns into $\mathcal{O}(M \cdot \mathbb{E}(|\mathcal{P}|))$. Here M is the number of constraints, and $\mathbb{E}(|\mathcal{P}|)$ is the average path length. In all our experiments, we experienced that the average path length grows more or less logarithmically with the number of nodes in the graph. This explains the fast pose updates of our approach shown in the experimental section.

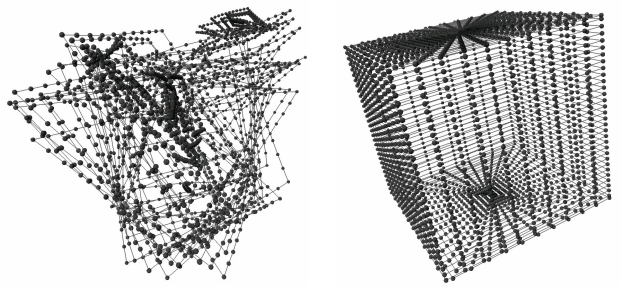

Fig. 3. A simulated trajectory of a robot moving on the surface of a cube. The left image shows an uncorrected trajectory and the right image depicts the corrected one (approx. 4,700 constraints, 100 iterations, 11s).

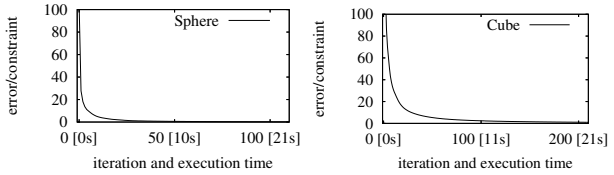

Fig. 4. The evolution of the error for the sphere and cube experiment.

VII. Experiments

The experiments are designed to show the properties of our technique. We first present results obtained in simulated experiments and then show results using real robot data.

A. Experiments with Simulated Data

In order to give the reader an intuition about the accuracy of our approach, we generated two datasets in which the virtual robot moved on the surfaces of easy to visualize geometric objects. In particular, we used a sphere and a cube. The nodes of the network as well as the constraints between the nodes were distorted with Gaussian noise. The left images of Figure 1 and Figure 3 depict the distorted input data whereas the images on the right illustrate the results obtained by our approach. As the figures indicate, the pose correction nicely recovers the original geometric structure.

To provide more quantitative results, Figure 4 depicts the evolution of the average error per link versus the iteration number as well as execution time for the sphere and the cube experiment. As can be seen, our approach converges to a configuration with small errors in less than 100 iterations.

We also applied the approach of Triebel *et al.* to both datasets. As mentioned above, this approach linearizes the

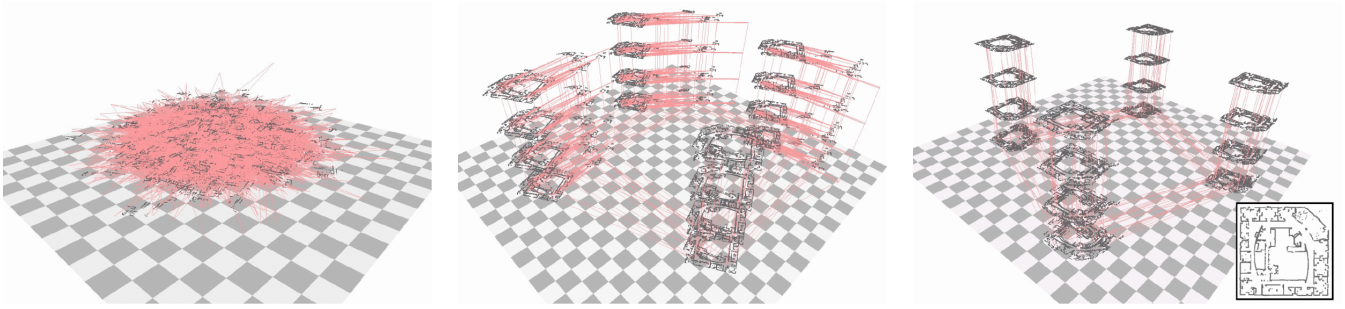

Fig. 5. The real world dataset of the Intel Research Lab recorded in 2D is used to generate a large 3D dataset. Each of the four virtual buildings consist of four identical floors. The left image depicts the starting configuration. The image in the middle depicts an intermediate result and the right one the corrected map after 50 iterations of our approach. We plotted in the images constraints between buildings and floors. For a better visibility, we furthermore plotted the constraints between individual nodes which introduce a high error and not all constraints. Constraints are plotted in light gray (red) and the laser data in black. The small image on the right shows a (corrected) map of the two-dimensional laser range data.

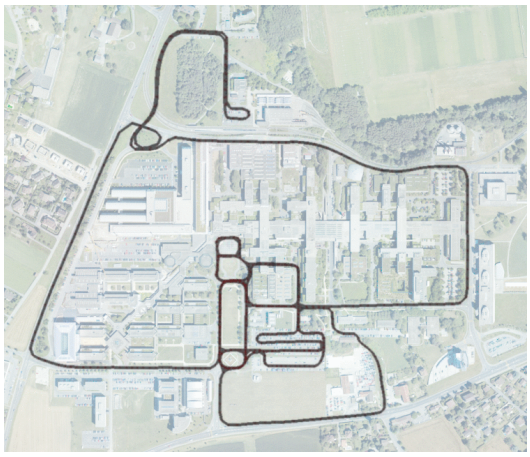

Fig. 6. The corrected trajectory plotted on top of an aerial image of the EPFL campus.

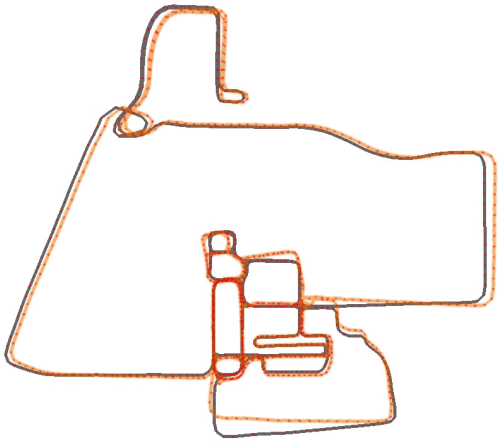

Fig. 7. The trajectory corrected by our approach is shown in black and the trajectory of the (D)GPS and IMU-based localization system is shown in orange/gray. By considering Figure 6 one can see that the black one covers the streets accurately.

problem and solves the resulting equation system using LU decomposition. Due to the comparably high noise in the simulated experiments, the linearization errors prevented this approach to find an appropriate configuration of the nodes.

B. Experiments with Partially Real Robot Data

The next experiment is obtained by extending data obtained from a 2D laser range finder into three dimensions. We used the 2D real world dataset of the Intel Research Lab in Seattle and constructed virtual buildings with multiple floors. The constraints between buildings and floors are manually added but all other data is real robot data. The dataset consists of 15.000 nodes and 72.000 constraints. We introduced a high error in the initial configuration of the poses in all dimensions. This initial configuration is shown in the left image of Figure 5. As can be seen, no structure is recognizable. When we apply our mapping approach, we get an accurate map of the environment. The image in the middle depicts an intermediate result and the right image show the resulting map after 50 iterations. To compute this result, it took around 3 minutes on a dual core Pentium 4 processor with 2.4 GHz.

C. Mapping with a Car-like Robot

Finally, we applied our method to a real world three-dimensional dataset. We used a Smart car equipped with 5 SICK laser range finders and various pose estimation sensors to record the data. The robot constructs local three-dimensional maps, so-called multi-level surface maps [15], and builds a network of constrains where each node represents such a local map. Constraints between the maps are obtained by matching the individual local maps.

We recorded a large-scale dataset at the EPFL campus in which the robot moved on a 10 km long trajectory. Figure 6 depicts an overlay of the corrected trajectory on an aerial image. As can be seen from the trajectory, several loops have been closed. Furthermore, it includes multiple levels such as an underground parking garage and a bridge with an underpass. The localization system of the car which is based on (D)GPS and IMU data is used to compute the incremental constraints. Additional constraints are obtained by matching local maps. This is achieved by first classifying cells of the local maps into different classes and then applying a variant of the ICP algorithm that considers these classes. More details on this matching can be found in our previous

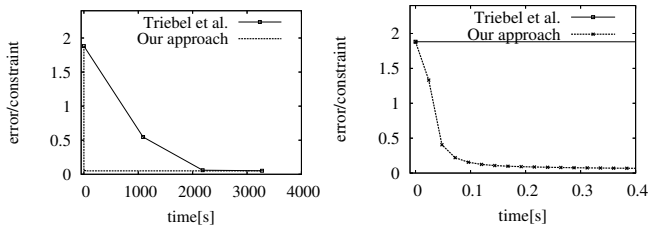

Fig. 8. The evolution of the average error per constraint of the approach of Triebel *et al.* [15] and our approach for the dataset recorded with the autonomous car. The right image shows a magnified view to the first 400 ms.

work [14]. Figure 7 plots the trajectory corrected by our approach and the one of the (D)GPS/IMU-based localization system.

We used the dataset from the EPFL campus to compare our new algorithm to the approach of Triebel *et al*. 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. This fact is illustrated in Figure 8 which plots the average error per constraints versus the execution time required by both techniques.

We also applied our 3D optimizer to pure 2D problems and compared its performance to our 2D method [6]. Both techniques lead to similar results, the 2D version, however, is around 3 times faster that the 3D version. This results from the additional DOF in the state space.

VIII. Conclusion

In this paper, we presented a highly efficient solution to the problem of learning three-dimensional maximum likelihood maps for mobile robots. Our technique is based on the graph-formulation of the simultaneous localization and mapping problem and applies a variant of gradient descent to minimize the error in the network of relations.

Our method has been implemented and exhaustively tested in simulation experiments as well as with real robot data. We furthermore compared our method to a common existing approach to learn such models in the three-dimensional space. As shown in the experiments, our approach converges significantly faster and yields accurate maps with low errors.

IX. Appendix: Proof of Eq. (19)

We can proof by induction that the equation system in Eq. (18) always has a solution which is given by

$$A_k \quad = \quad R_k^T (B_{1:k-1})^T R_k B_{1:k} \qquad k = 1, \ldots, n. \tag{20}$$

- **Basis** $(n = 1)$:
  Based on Eq. (18) with $n = 1$ and by knowing that R_1 is a rotation matrix, a solution always exists and is given by
  $$A_1 = R_1^{-1} R_1 B_1 = B_1. \tag{21}$$
- **Inductive Step**:
  Assuming that Eq. (19) holds for $k = 1, \ldots, n-1$, we show that it holds also for $k = n$. We use Eq. (18) with $k = n - 1$ to substitute the term $R_1 A_1 \ldots R_{n-1} A_{n-1}$ in the equation for $k = n$. This leads to
  $$(R_{1:n-1} B_{1:n-1}) R_n A_n \quad = \quad R_{1:n-1} R_n B_{1:n}. \tag{22}$$
  By multiplying $(R_{1:n-1} B_{1:n-1} R_n)^{-1}$ from the left hand side, this turns into
  $$A_n \quad = \quad R_n^{-1} (B_{1:n-1})^{-1} (R_{1:n-1})^{-1} R_{1:n-1} R_n B_{1:n}$$
  Since R_k and B_k are rotation matrices, the inverse is always defined and given by the transposed matrix:
  $$A_n \quad = \quad R_n^T (B_{1:n-1})^T R_n B_{1:n} \qquad \text{q.e.d.} \tag{23}$$

Acknowledgment

This work has been supported by the DFG within the Research Training Group 1103 and under contract number SFB/TR-8 and by the EC under contract number FP6-2005-IST-5-muFly, FP6-2005-IST-6-RAWSEEDS, and FP6-004250-CoSy. Thanks to Udo Frese for his insightful comments and to Rudolph Triebel for providing his mapping system. Further thanks to Pierre Lamon for the joint effort in recording the EPFL dataset.

References

[1] T. Barrera, A. Hast, and E. Bengtsson. Incremental spherical linear interpolation. In *SIGRAD*, volume 13, pages 7–13, 2004.
[2] F. Dellaert. Square Root SAM. In *Proc. of Robotics: Science and Systems (RSS)*, pages 177–184, Cambridge, MA, USA, 2005.
[3] T. Duckett, S. Marsland, and J. Shapiro. Fast, on-line learning of globally consistent maps. *Autonomous Robots*, 12(3):287 – 300, 2002.
[4] 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.
[5] G. Grisetti, C. Stachniss, S. Grzonka, and W. Burgard. TORO project at OpenSLAM.org. http://www.openslam.org/toro.html, 2007.
[6] G. Grisetti, C. Stachniss, S. Grzonka, and W. Burgard. A tree parameterization for efficiently computing maximum likelihood maps using gradient descent. In *Proc. of Robotics: Science and Systems (RSS)*, Atlanta, GA, USA, 2007.
[7] J.-S. Gutmann and K. Konolige. Incremental mapping of large cyclic environments. In *Proc. of the IEEE Int. Symposium on Computational Intelligence in Robotics and Automation (CIRA)*, pages 318–325, Monterey, CA, USA, 1999.
[8] 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 (IROS)*, 2001.
[9] A. Howard, D.F. Wolf, and G.S. Sukhatme. Towards 3d mapping in large urban environments. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, pages 419–424, 2004.
[10] F. Lu and E. Milios. Globally consistent range scan alignment for environment mapping. *Autonomous Robots*, 4:333–349, 1997.
[11] P. Newman, D. Cole, and K. Ho. Outdoor slam using visual appearance and laser ranging. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, Orlando, FL, USA, 2006.
[12] A. Nüchter, K. Lingemann, J. Hertzberg, and H. Surmann. 6d SLAM with approximate data association. In *Proc. of the 12th Int. Conference on Advanced Robotics (ICAR)*, pages 242–249, 2005.
[13] E. Olson, 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 (ICRA)*, pages 2262–2269, 2006.
[14] P. Pfaff, R. Triebel, C. Stachniss, P. Lamon, W. Burgard, and R. Siegwart. Towards mapping of cities. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, Rome, Italy, 2007.
[15] R. Triebel, P. Pfaff, and W. Burgard. Multi-level surface maps for outdoor terrain mapping and loop closing. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2006.