Online Constraint Network Optimization for Efficient Maximum Likelihood Map Learning

Giorgio Grisetti* Dario Lodi Rizzini‡ Cyrill Stachniss* Edwin Olson† Wolfram Burgard*

***Abstract*— In this paper, we address the problem of incrementally optimizing constraint networks for maximum likelihood map learning. Our approach allows a robot to efficiently compute configurations of the network with small errors while the robot moves through the environment. We apply a variant of stochastic gradient descent and use a tree-based parameterization of the nodes in the network. By integrating adaptive learning rates in the parameterization of the network, our algorithm can use previously computed solutions to determine the result of the next optimization run. Additionally, our approach updates only the parts of the network which are affected by the newly incorporated measurements and starts the optimization approach only if the new data reveals inconsistencies with the network constructed so far. These improvements yield an efficient solution for this class of online optimization problems.**

Our approach has been implemented and tested on simulated and on real data. We present comparisons to recently proposed online and offline methods that address the problem of optimizing constraint network. Experiments illustrate that our approach converges faster to a network configuration with small errors than the previous approaches.

I. Introduction

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

Solutions to the SLAM problem can be furthermore divided into online an offline methods. Offline methods are so-called batch algorithms that require all the data to be available right from the beginning [1], [2], [3]. In contrast to that, online methods can re-use an already computed solution and update or refine it. Online methods are needed for situations in which the robot has to make decisions based on the model of the environment during mapping. Exploring an unknown environment, for example, is a task of this category. Popular online SLAM approaches such as [4], [5] are based on the Bayes' filter. Recently, also incremental maximum-likelihood approaches have been presented as an effective alternative [6], [7], [8].

*Department of Computer Science, University of Freiburg, Germany. †MIT, 77 Massachusetts Ave., Cambridge, MA 02139-4307, USA. ‡Department of Information Engineering, University of Parma, Italy.

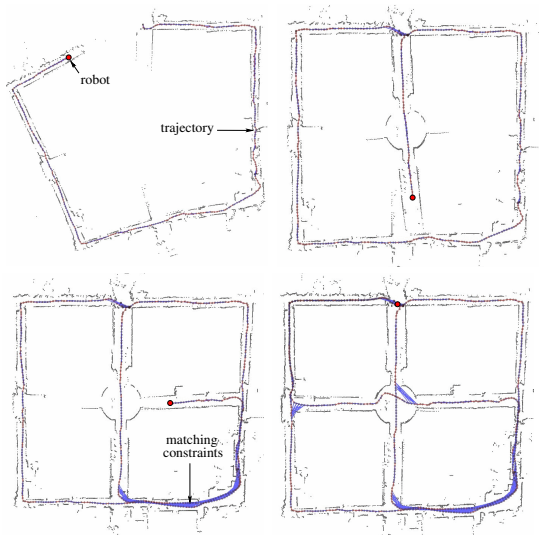

Fig. 1. Four snapshots created while incrementally learning a map.

In this paper, we present an efficient online optimization algorithm which can be used to solve the so-called "graph-based" or "network-based" formulation of the SLAM problem. Here, the poses of the robot are modeled by nodes in a graph and constraints between poses resulting from observations or from odometry are encoded in the edges between the nodes. Our method belongs to the same class of techniques of Olson's algorithm or MLR [8]. It focuses on computing the best map and it assumes that the constraints are given. Techniques like the ATLAS framework [9] or hierarchical SLAM [10], for example, can be used to obtain the necessary data associations (constraints). They also apply a global optimization procedure to compute a consistent map. One can replace these optimization procedures by our algorithm and in this way make them more efficient.

Our approach combines the ideas of adaptive learning rates with a tree-based parameterization of the nodes when applying stochastic gradient descent. This yields an online algorithm that can efficiently compute network configurations with low errors. An application example is shown in Figure 1. It depicts four snapshots of our online approach during a process of building a map from the ACES dataset.

II. Related Work

A large number of mapping approaches has been presented in the past and a variety of different estimation techniques have been used to learn maps. One class of approaches uses constraint networks to represent the relations between poses and observations.

Lu and Milios [1] were the first who used constraint networks to address the SLAM problem. They proposed a brute force method that seeks to optimize the whole network at once. Gutmann and Konolige [11] presented an effective way for constructing such a network and for detecting loop closures while running an incremental estimation algorithm. Frese *et al.* [8] described a variant of Gauss-Seidel relaxation called multi-level relaxation (MLR). It applies relaxation at different resolutions.

Olson *et al.* [2] were the first who applied a variant of stochastic gradient descent to compute solutions to this family of problems. They propose a representation of the nodes which enables the algorithm to perform efficient updates. Our previously presented method [3] introduced the tree parameterization that is also used in this paper. Subsequently, Olson *et al.* [6] presented an online variant of their method using adaptive learning rates. In this paper, we integrate such learning rates into the tree-based parameterization which yields a solution to the online SLAM problem that outperforms the individual methods.

Kaess *el al.* [7] proposed an on-line version of the smoothing and mapping algorithm for maximum likelihood map estimation. This approach relies on a QR factorization of the information matrix and integrates the new measurements as they are available. Using the QR factorization, the poses of the nodes in the network can be efficiently retrieved by back substitution. Additionally they keep the matrices sparse via occasional variable reordering. Frese [12] proposed the Treemap algorithm which is able to perform efficient updates of the estimate by ignoring the weak correlations between distant locations.

The contribution of this paper is an efficient online approach for learning maximum likelihood maps. It integrates adaptive learning rates into a tree-based network optimization technique using a variant of stochastic gradient descent. Our approach presents an efficient way of selecting only the part of the network which is affected by newly incorporated data. Furthermore, it allows to delay the optimization so that the network is only updated if needed.

III. Stochastic Gradient Descent for Maximum Likelihood Mapping

Approaches to graph-based SLAM focus on estimating the most likely configuration of the nodes and are therefore referred to as maximum-likelihood (ML) techniques [8], [1], [2]. The approach presented in this paper also belongs to this class of methods.

The goal of graph-based ML mapping algorithms is to find the configuration of the nodes that maximizes the likelihood of the observations. Let $\mathbf{x} = (x_1 \ \cdots \ x_n)^T$ be a vector of parameters which describes a configuration of the nodes. Let δ_{ji} and Ω_{ji} be respectively the mean and the information matrix of an observation of node j seen from node i. Let $f_{ji}(\mathbf{x})$ be a function that computes a zero noise observation according to the current configuration of the nodes j and i.

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

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

as well as the *residual* $r_{ji} = -e_{ji}(\mathbf{x})$. Let $\mathcal{C} = \{\langle j_1, i_1\rangle, \ldots, \langle j_M, i_M\rangle\}$ be the set of pairs of indices for which a constraint $\delta_{j_m i_m}$ exists. The goal of a ML approach is to find the configuration $\mathbf{x}^*$ of the nodes that minimized the negative log likelihood of the observations. Assuming the constraints to be independent, this can be written as

$$\mathbf{x}^* = \operatorname*{argmin}_{\mathbf{x}} \sum_{\langle j,i\rangle \in \mathcal{C}} r_{ji}(\mathbf{x})^T \Omega_{ji} r_{ji}(\mathbf{x}). \tag{2}$$

In the remainder of this section we describe how the general framework of stochastic gradient descent can be used for minimizing Eq. (2) and how to construct a parameterization of the network which increases the convergence speed.

A. Network Optimization using Stochastic Gradient Descent

Olson *et al.* [2] propose to use a variant of the preconditioned stochastic gradient descent (SGD) to address the compute the most likely configuration of the network's nodes. The approach minimizes Eq. (2) by iteratively selecting a constraint $\langle j, i\rangle$ and by moving the nodes of the network in order to decrease the error introduced by the selected constraint. Compared to the standard formulation of gradient descent, the constraints are not optimized as a whole but individually. The nodes are updated according to the following equation:

$$\mathbf{x}^{t+1} \quad = \quad \mathbf{x}^t + \lambda \cdot \mathbf{H}^{-1} J_{ji}^T \Omega_{ji} r_{ji} \tag{3}$$

Here $\mathbf{x}$ is the set of variables describing the locations of the poses in the network and $\mathbf{H}^{-1}$ is a preconditioning matrix. J_{ji} is the Jacobian of f_{ji}, Ω_{ji} is the information matrix capturing the uncertainty of the observation, r_{ji} is the residual, and λ is the learning rate which decreases with the iteration. For a detailed explanation of Eq. (3), we refer the reader to our previous works [3], [2].

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

B. Tree Parameterization

The poses $\mathbf{p} = \{p_1, \ldots, p_n\}$ of the nodes define the configuration of the network. The poses can be described by a vector of *parameters* $\mathbf{x}$ such that a bidirectional mapping between $\mathbf{p}$ and $\mathbf{x}$ exists. The parameterization defines the subset of variables that are modified when updating a constraint. An efficient way of parameterizing the node is to

use a tree. One can construct a spanning tree (not necessarily a minimum one) from the graph of poses. Given such a tree, we define the parameterization for a node as

$$x_i \quad = \quad p_i - p_{\text{parent}(i)}, \tag{4}$$

where $p_{\text{parent}(i)}$ refers to the parent of node i in the spanning tree. As defined in Eq. (4), the tree stores the differences between poses. This is similar in the spirit to the incremental representation used in the Olson's original formulation, in that the difference in pose positions (in global coordinates) is used rather than pose-relative coordinates or rigid body transformations.

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

$$r'_{ji} \quad = \quad \sum_{k^{[-]} \in \mathcal{P}_{ji}^{[-]}} x_{k^{[-]}} - \sum_{k^{[+]} \in \mathcal{P}_{ji}^{[+]}} x_{k^{[+]}} + R_i \delta_{ji}. \tag{5}$$

Here R_i is the homogeneous rotation matrix of the pose p_i. It can be computed according to the structure of the tree as the product of the individual rotation matrices along the path to the root. Note that this tree does not replace the graph as an internal representation. The tree only defines the parameterization of the nodes.

Let $\Omega'_{ji} = R_i \Omega_{ji} R_i^T$ be the information matrix of a constraint in the global frame. According to [2], we compute an approximation of the Jacobian as

$$J'_{ji} \quad = \quad \sum_{k^{[+]} \in \mathcal{P}_{ji}^{[+]}} \mathcal{I}_{k^{[+]}} - \sum_{k^{[-]} \in \mathcal{P}_{ji}^{[-]}} \mathcal{I}_{k^{[-]}}, \tag{6}$$

with $\mathcal{I}_k = (0 \ \cdots \ 0 \ \underbrace{I}_{k^{\text{th}} \text{ element}} \ 0 \ \cdots \ 0)$. Then, the update of a constraint turns into

$$\mathbf{x}^{t+1} \quad = \quad \mathbf{x}^t + \lambda |\mathcal{P}_{ji}| \mathbf{M}^{-1} \Omega'_{ji} r'_{ji}, \tag{7}$$

where $|\mathcal{P}_{ji}|$ refers to the number of nodes in $\mathcal{P}_{ji}$. In Eq. (7), we replaced the preconditioning matrix $\mathbf{H}^{-1}$ with its scaled approximation $\mathbf{M}^{-1}$ as described in [2]. This prevents from a computationally expensive matrix inversion.

Let the *level* of a node be the distance in the tree between the node itself and the root. We define the *top node* of a constraint as the node on the path with the smallest level. Our parameterization implies that updating a constraint will never change the configuration of a node with a level smaller than the level of the top node of the constraint.

In principle, one could apply the technique described in this section as a batch algorithm to an arbitrarily constructed spanning tree of the graph. However, our proposed method uses a spanning tree which can be constructed incrementally, as described in the next section.

IV. Online Network Optimization

The algorithm presented in the previous section is a batch procedure. At every iteration, the poses of all nodes in the network are optimized. The fraction of the residual used in updating every constraint decreases over time with the learning rate λ, which evolves according to an harmonic progression. During online optimization, the network is dynamically updated to incorporate new movements and observations. In theory, one could also apply the batch version of our optimizer to correct the network. This, however, would require to compute a solution from scratch each time the robot moves or makes an observation which would obviously lead to an inefficient algorithm.

In this section we describe an incremental version of our optimization algorithm, which is suitable for solving online mapping problems. As pointed in [6] an incremental algorithm should have the following properties:

1) Every time a constraint is added to the network, only the part of the network which is affected by that constraint should be optimized. For example, when exploring new terrain, the effects of the optimization should not perturb distant parts of the graph.
2) When revisiting a known region of the environment it is common to re-localize the robot in the previously built map. One should use the information provided by the re-localization to compute a better initial guess for the position of the newly added nodes.
3) To have a consistent network, performing an optimization step after adding each constraint is often not needed. This happens when the newly added constraints are adequately satisfied by the current network configuration. Having a criterion for deciding when to perform unnecessary optimizations can save a substantial amount of computation.

In the remainder of this section, we present four improvements to the algorithm so that it satisfies the discussed properties.

A. Incremental Construction of the Tree

When constructing the parameterization tree online, we can assume that the input is a sequence of poses corresponding to a trajectory of the robot. In this case, subsequent poses are located closely together and there exist constraints between subsequent poses resulting from odometry or scan-matching. Further constraints between arbitrary nodes result from observations when revisiting a place in the environment.

We proceed as follows: the oldest node is the root of the tree. When adding a node i to the network, we choose as its parent the oldest node for which a constraint to the node i exists. Such a tree can be constructed incrementally since adding a new node does not require to change the existing parts of the tree.

The pose p_i and parameter x_i of a newly added node i is initialized according to the position of the parent node and

the connecting constraint as

$$p_i = p_{\text{parent}(i)} \oplus \delta_{i,\text{parent}(i)} \quad (8)$$

$$x_i = p_i - p_{\text{parent}(i)}. \quad (9)$$

The parent node represents an already explored part of the environment and the constraint between the new node and the parent can be regarded as a localization event in an already constructed map, thus satisfying Property 2. As shown in the experiments described below, this initialization appears to be a good heuristic for determining the initial guess of the pose of a newly added node.

B. Constraint Selection

When adding a constraint $\langle j, i \rangle$ to the graph, a subset of nodes needs to be updated. This set depends on the topology of the network and can be determined by a variant of breadth first visit. Let $\mathcal{G}_{j,i}$ be the minimal subgraph that contains the added constraint and has only one constraint to the rest of the graph. Then, the nodes that need to be updated are all nodes of the minimal subtree that contains $\mathcal{G}_{j,i}$. The precise formulation on how to efficiently determine this set is given by Algorithm 1.

Data: $\langle j, i \rangle$: the constraint, $\mathcal{G}$: the graph, $\mathcal{T}$: the tree.
Result: $\mathcal{N}_{ji}$: the set of affected nodes, $\mathcal{E}_{ji}$: the affected constraints.
Queue $f = \text{childrenOf}(\text{topNode}(\langle j, i \rangle))$;
$\mathcal{E}_{ji} := \text{edgesToChildren}(\text{topNode}(\langle j, i \rangle))$;
foreach $\langle a, b \rangle \in \mathcal{E}_{ji}$ **do**
  $\langle a, b \rangle .mark = true$;
end
while $f \neq \{\}$ **do**
  Node $n := \text{first}(f)$;
  $n.mark := true$
  foreach $\langle a, b \rangle \in \text{edgesOf}(n)$ **do**
    if $\langle a, b \rangle .mark = true$ **then**
      continue;
    end
    Node $m := (a = n)?b : a$;
    if $m = \text{parent}(n)$ or $m.mark = true$ **then**
      continue;
    end
    $\langle a, b \rangle .mark = true$;
    $\mathcal{E}_{ji} := \mathcal{E}_{ji} \cup \{\langle a, b \rangle\}$;
    if $\langle a, b \rangle \in \mathcal{T}$ **then**
      $f := f \cup \{m\}$;
    else
      $f := f \cup \text{childrenOf}(\text{topNode}(\langle a, b \rangle))$;
    end
  end
  $f := \text{removeFirst}(f)$;
  $\mathcal{N}_{ji} := \mathcal{N}_{ji} \cup \{n\}$;
end

Algorithm 1: Construction of the set of nodes affected by a constraint. For readability we assume that the frontier f can contain only the nodes which are not already marked.

Note that the number of nodes in $\mathcal{G}_{j,i}$ does depend only on the root of the tree and on the overall graph. It contains all variables which are affected by adding the new costraint $\langle i, j \rangle$.

C. Adaptive Learning Rates

Rather than using one learning rate λ for all nodes, the incremental version of the algorithm uses spatially adaptive learning rates introduced in [6]. The idea is to assign an individual learning rate to each node, allowing different parts of the network to be optimized at different rates. These learning rates are initialized when a new constraint is added to the network and they decrease with each iteration of the algorithm. In the following, we describe how to initialize and update the learning rates and how to adapt the update of the network specified in Eq. (7).

a) Initialization of the learning rates: When a new constraint $\langle j, i \rangle$ is added to the network, we need to update the learning rates for the nodes $\mathcal{N}_{ji}$ determined in the previous section. First, we compute the learning rate λ'_{ji} for the newly introduced information. Then, we propagate this learning rate to the nodes $\mathcal{N}_{ji}$.e

A proper learning rate is determined as follows. Let β_{ji} be the fraction of the residual that would appropriately fuse the previous estimate and the new constraint. Similar to a Kalman filter, β_{ji} is determined as

$$\beta_{ji} = \Omega_{ji}(\Omega_{ji} + \Omega_{ji}^{\text{graph}})^{-1}, \quad (10)$$

where Ω_{ji} is the information matrix of the new constraint, and $\Omega_{ji}^{\text{graph}}$ is an information matrix representing the uncertainty of the constraints in the network. Based on Eq. (10), we can compute the learning rate λ'_{ji} of the new constraint as

$$\lambda'_{ji} = \text{maxrow}\left(\frac{1}{|\mathcal{P}_{ji}|}(\beta_{ji} \oslash \mathbf{M}\Omega'_{ji})\right). \quad (11)$$

Here $\oslash$ represents the row by row division (see [6] for further details). The learning rate of the constraint is then propagated to all nodes $k \in \mathcal{N}_{ji}$ as

$$\lambda_k \leftarrow \max(\lambda_k, \lambda'_{ji}), \quad (12)$$

where λ_k is the learning rate of the node k. According to Eq. (11) constraints with large residuals result in larger learning rate increases than constraints with small residuals.

b) Update of the network: When updating the network, one has to consider the newly introduced learning rates. During an iteration, we decrease the individual learning rates of the nodes according to a generalized harmonic progression [13]:

$$\lambda_k \leftarrow \frac{\lambda_k}{1 + \lambda_k} \quad (13)$$

In this way, one guarantees the strong monotonicity of λ_k and thus the convergence of the algorithm to an equilibrium point.

The learning rates of the nodes cannot be directly used for updating the poses since Eq. (7) requires a learning rate for each constraint and not for each node. When updating the network given the constraint $\langle j, i \rangle$, we obtain an average learning rate $\tilde{\lambda}_{ji}$ from the nodes on $\mathcal{P}_{ji}$ as

$$\tilde{\lambda}_{ji} = \frac{1}{|\mathcal{P}_{ji}|} \sum_{k \in \mathcal{P}_{ji}} \lambda_k. \quad (14)$$

Then, the constraint update turns into

$$\Delta \mathbf{x}_k \quad = \quad \tilde{\lambda}_{ji} |\mathcal{P}_{ji}| \mathbf{M}^{-1} \Omega'_{ji} r'_{ji}. \tag{15}$$

D. Scheduling the Network Optimization

When adding a set of constraints $\langle j, i \rangle \in \mathcal{C}_{\text{new}}$ to a network without performing an optimization, we can incrementally compute the error of the network as

$$e_{\text{new}} = \sum_{\langle j,i \rangle \in \mathcal{C}_{\text{old}}} r_{ji}^T \Omega_{ji} r_{ji} + \sum_{\langle j,i \rangle \in \mathcal{C}_{\text{new}}} r_{ji}^T \Omega_{ji} r_{ji}. \tag{16}$$

Here e_{new} is the new error and $\mathcal{C}_{\text{old}}$ refers to the set of constraints before the modification.

To avoid unnecessary computation, we perform the optimization only if needed. This is the case when the newly incorporated information introduced a significant error compared to the error of the network before. We perform an optimization step if

$$\frac{e_{\text{new}}}{|\mathcal{C}_{\text{new}}| + |\mathcal{C}_{\text{old}}|} > \alpha \max_{\langle j,i \rangle \in \mathcal{C}_{\text{old}}} r_{ji}^T \Omega_{ji} r_{ji}. \tag{17}$$

Here α is a user-defined factor that allows the designer of a mapping system to adapt the quality of the incremental solutions to the needs of the specific application.

If we assume that the network in $\mathcal{C}_{\text{old}}$ has already converged, this heuristic triggers an optimization only if a significant inconsistency is revealed. Furthermore, the optimization only needs to be performed for a subset of the network and not for the whole network. The subset is given by

$$\mathcal{E} \quad = \quad \bigcup_{\langle j,i \rangle \in \mathcal{C}_{\text{new}}} \mathcal{E}_{ji}. \tag{18}$$

Here $\mathcal{E}_{ji}$ is the set of constraints to be updated given a new constraint $\langle j, i \rangle \in \mathcal{C}_{\text{new}}$. The sets $\mathcal{E}_{ji}$ are computed according to Algorithm 1. This criterion satisfies Property 3 and leads to an efficient algorithm for incrementally optimizing the network of constraints.

V. Experiments

This section is designed to evaluate the effectiveness of the proposed methods to incrementally learn maximum likelihood maps. We first show that such a technique is well suited to generate accurate grid maps given laser range data and odometry from a real robot. Second, we provide simulation experiments to evaluate the evolution of the error and provide comparisons to our previously proposed techniques [3], [2], [6]. Finally, we illustrate the computational advantages resulting from our algorithm.

A. Real World Experiments

To illustrate that our technique can be used to learn maps from real robot data, we used the freely available ACES dataset. The motivating example shown in Figure 1 depicts four different maps computed online by our incremental mapping technique. During this experiment, we extracted constraints between consecutive poses by means of pairwise scan matching. Loop closures were determined by localizing the robot in the previously built map by means of a particle filter.

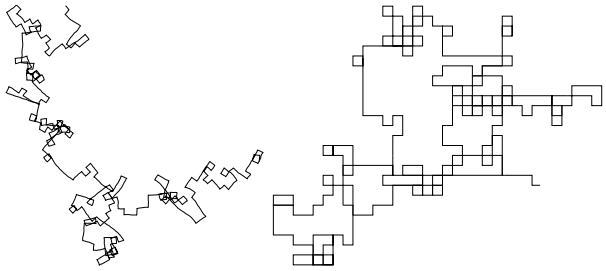

Fig. 2. Network used in the simulated experiments. Left: initial guess. Right: ground truth.

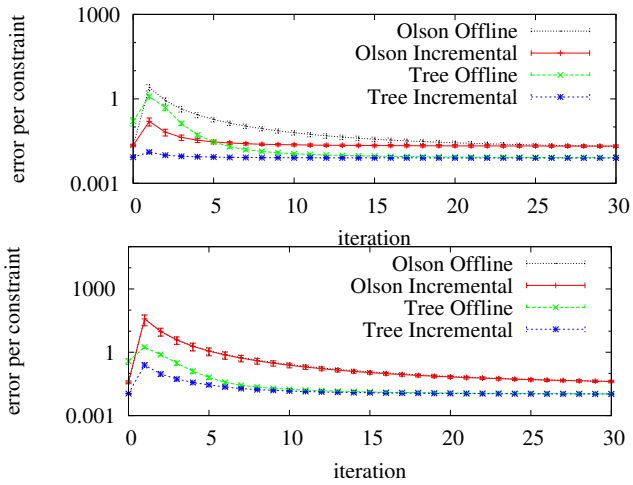

Fig. 3. Statistical experiments showing the evolution of the error per iteration of the algorithm. Top: situation in which the robot is closes a small loop. Bottom: closure of a large loop. The statistics have been generated by considering 10 different realizations of the observation noise along the same path.

As can be seen, our approach leads to accurate maps for real robot data. Similar results were obtained with all datasets we found online or recorded on our own.

B. Statistical Experiments on the Evolution of the Error

In the these experiments, we moved a virtual robot on a grid world. An observation is generated each time the current position of the robot was close to a previously visited location. The observations are corrupted by a given amount of Gaussian noise. The network used in this experiment is depicted in Figure 2.

We compare our approach named *Tree Incremental* with its offline variant [3] called *Tree Offline* which solves the overall problem from scratch. In addition to that, we compare it to the offline version without the tree optimization [2] called *Olson Offline* as well as its incremental variant [6] referred to as *Olson Incremental*. For space reasons, we omit comparisons to LU decomposition, EKF, and Gauss-Seidel. The advantages of our method over these other methods is similar to those previously reported [2].

To allow a fair comparison, we disabled the scheduling of the optimization of Eq. (17) and we performed 30 iterations every time 16 constraints were added to the network. During the very first iterations, the error of all approaches may show

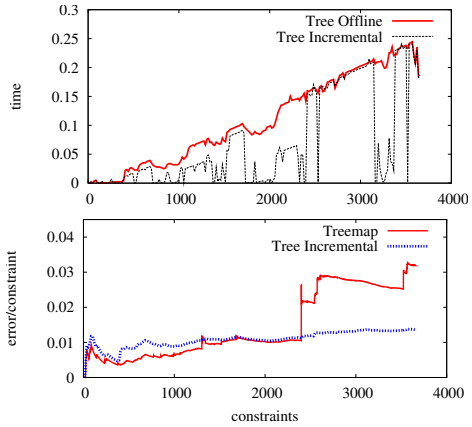

Fig. 4. Top: runtime comparison of the offline and the incremental approaches using a tree parameterization. The optimization is performed only when the error condition specified by Eq. (17) was verified. Bottom: Comparison of the evolution of the global error between Treemap[12] and the online version of our approach.

an increase, due to the bigger correction steps which result from increasing the learning rates.

Figure 3 depicts the evolution of the error for all four techniques during a mapping experiment. We depicted two situations. In the first one, the robot closed a small loop. As can be seen, the introduced error is small and thus our approach corrects the error within 2 iterations. Both incremental techniques perform better than their offline variants. The approach proposed in this paper outperforms the other techniques. The same holds for the second situation in which the robot was closing a large loop. Note that in most cases, one iteration of the incremental approach can be carried out faster, since only a subpart of the network needs to be updated.

C. Runtime Comparison

Finally, we evaluated our incremental version and its offline variant with respect to the execution time. Both methods where executed only when needed according to our criterion specified by Eq. (17). We measured the time needed to run the individual approach until convergence to the same low error configuration, or until a maximum number of iterations (30) was reached. As can be seen in Figure 4(top), the incremental technique requires significantly less operations and thus runtime to provide equivalent results in terms of error. Figure 4(bottom) shows the error plot of a comparison of our approach and Treemap [12] proposed by Frese. As shown in the error-plot, in the beginning Treemap performs slightly better than our algorithm, due to the exact calculation of the Jacobians. However, when closing large loops Treemap is more sensitive to angular wraparounds (see increase of the error at constraint 2400 in Figure 4). This issue is typically better handled by our iterative procedure. Overall, we observed that for datasets having a small noise Treemap provides slightly better estimates, while our approach is generally more robust to extreme conditions.

VI. Conclusion

In this paper, we presented an efficient online solution to the optimization of constraint networks. It can incrementally learn maps while the robot moves through the environment. Our approach optimizes a network of constraints that represents the spatial relations between the poses of the robot. It uses a tree-parameterization of the nodes and applies a variant of gradient descent to compute network configurations with low errors.

A per-node adaptive learning rate allows the robot to reuse already computed solutions from previous steps, to update only the parts of the network, which are affected by the newly incorporated information, and to start the optimization approach only if the new data causes inconsistencies with the already computed solution. We tested our approach on real robot data as well as with simulated datasets. We compared it to recently presented online and offline methods that also address the network-based SLAM problem. As we showed in practical experiments, our approach converges faster to a configuration with small errors.

Acknowledgment

The authors gratefully thank Udo Frese for providing us his Treemap implementation. This work has partly been supported by the DFG under contract number SFB/TR-8 (A3), by the EC under contract number FP6-IST-34120-muFly, and FP6-2005-IST-6-RAWSEEDS.

References

[1] F. Lu and E. Milios, "Globally consistent range scan alignment for environment mapping," *Journal of Autonomous Robots*, vol. 4, 1997.

[2] 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)*, 2006, pp. 2262–2269.

[3] 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. [Online]. Available: http://www.informatik.uni-freiburg.de/ stachnis/pdf/grisetti07rss.pdf

[4] R. Smith, M. Self, and P. Cheeseman, "Estimating uncertain spatial realtionships in robotics," in *Autonomous Robot Vehicles*, I. Cox and G. Wilfong, Eds. Springer Verlag, 1990, pp. 167–193.

[5] M. Montemerlo and S. Thrun, "Simultaneous localization and mapping with unknown data association using FastSLAM," in *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, Taipei, Taiwan, 2003.

[6] E. Olson, J. Leonard, and S. Teller, "Spatially-adaptive learning rates for online incremental slam," in *Robotics: Science and Systems*, Atlanta, GA, USA, 2007.

[7] M. Kaess, A. Ranganathan, and F. Dellaert, "iSAM: Fast incremental smoothing and mapping with efficient data association," in *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 2007.

[8] U. Frese, P. Larsson, and T. Duckett, "A multilevel relaxation algorithm for simultaneous localisation and mapping," *IEEE Transactions on Robotics*, vol. 21, no. 2, pp. 1–12, 2005.

[9] M. Bosse, P. Newman, J. Leonard, and S. Teller, "An ALTAS framework for scalable mapping," in *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, Taipei, Taiwan, 2003.

[10] C. Estrada, J. Neira, and J. Tardós, "Hierachical slam: Real-time accurate mapping of large environments," *IEEE Transactions on Robotics*, vol. 21, no. 4, pp. 588–596, 2005.

[11] 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)*, Monterey, CA, USA, 1999, pp. 318–325.

[12] U. Frese, "Treemap: An $o(logn)$ algorithm for indoor simultaneous localization and mapping," *Journal of Autonomous Robots*, vol. 21, no. 2, pp. 103–122, 2006.

[13] H. Robbins and S. Monro, "A stochastic approximation method," *Annals of Mathematical Statistics*, vol. 22, pp. 400–407, 1951.