A Partially Fixed Linearization Approach for Submap-Parametrized Smoothing and Mapping

Alexander Kipp, Peter Krauthausen, and Frank Dellaert

College of Computing, Georgia Institute of Technology

Atlanta, Georgia 30332

Email: {mail@alexanderkipp.de, peter@krauthausen.com, dellaert@cc.gatech.edu}

Technical Report number GIT-GVU-05-25

September 2005

Abstract

We present an extension of a smoothing approach to Simultaneous Localization and Mapping (SLAM). We have previously introduced Square-Root SAM, a Smoothing and Mapping approach to SLAM based on Levenberg-Marquardt (LM) optimization. It iteratively finds the optimal nonlinear least squares solution (ML), where one iteration comprises of a linearization step, a matrix factorization, and a back-substitution step. We introduce a submap parametrization which enables a rigid transformation of parts relative to each other during the optimization process. This parameterization is used in a multifrontal QR factorization approach, in which we partially fix the linearization point for a subset of the unknowns corresponding to sub-maps. This greatly accelerates the optimization of an entire SAM graph yet yields an exact solution.

Contents

1 Introduction

In the SLAM problem a robot moves in an unknown environment, while perceiving the environment with its sensors. Both these measurements and odometry are subject to measurement error. Hence, the problem in SLAM is to estimate the most likely solution for the robot's trajectory as well as a map of the environment.

If one makes the assumption that the motion and the sensors are corrupted by a normally distributed noise, the SLAM problem translates to a least-squares optimization problem. We adopt a direct optimization approach in which both the motion and sensor models, which are in general non-linear functions, are linearized to iteratively solve the associated least-squares problem.

We describe a framework which yields an exact (maximum-likelihood) solution for the smoothing version of SLAM and is based on Levenberg-Marquardt (LM). In contrast, many state-of-the-art approaches marginalize the robot poses (filtering) which yields only an approximation, especially in the case where further observations would have strongly changed already marginalized poses. Levenberg-Marquardt is an iterative, non-linear optimization. Non-linear measurement functions are re-linearized in each iteration and the corresponding Jacobian matrix is factorized to iteratively update the solution.

In our approach we use a multifrontal factorization method, which we modify in such a way as to first solve parts of the system. After convergence, a good linearization point for these parts is obtained which will be fixed from that point on. Only a small remaining part of the entire system needs to be re-linearized and factorized after this point. This yields in a substantial acceleration of the global optimization problem, one which has wide applicability in areas such as submaps SLAM, multiple robot SLAM, distributed SLAM and incremental SAM.

2 Related Work

Most state-of-the-art approaches are based on the stochastic map (SC87) and the extended Kalman Filter (EKF), where the update of the covariance matrix takes $O(n^2)$ time for n landmarks. FastSLAM (MT03) estimates the robot's trajectory with a particle filter, and the map using a Kalman filter for each landmark. This approach prevents updating a large covariance matrix. (TL03; FH01) use the information matrix instead of the covariance matrix, which has the advantage that it remains sparse under the assumption that only a bounded number of landmarks can be seen from a robot pose. (BNLT05; LF01) introduces the notion of

submaps with independent local coordinate systems. A graphical model is used by (Pas03), which is based on a Junction tree (clique tree) and it is kept sparse through an approximating thinning operation. A very similar approach is formulated in (Fre05).

(JU01) show that that even for "a simple scenario the full-covariance stochastic mapping approach is guaranteed to produce inconsistent vehicle and beacon estimates,... it is questionable whether the Kalman filter framework developed in (SC87) provides a general, robust and rigorous solution to the stochastic SLAM problem". The question is now how to find an exact and practical solution for SLAM. (ESL05) is an exactly sparse delayed-state filter. This approach needs to recover the state mean vector, which is itself an optimization problem.

(FC04; FPC05) use a graphical model approach to SLAM, in which they relax the graph after adding new nodes by Gauss-Seidel iterations. This paper introduces so called "star nodes", which are a compact representation of the robot's history, and for which the linearization point of parts are fixed. A multilevel relaxation algorithm can be found in (FLD05), which is a standard technique for solving partial differential equations.

(TMHF00) is an excellent and very detailed survey of bundle adjustment in computer vision. The task is to find "the optimal structure and viewing parameter estimates"; expressed in an oversimplified way: where are the pictures taken and where are the features in the environment? This survey summarizes in detail, how to efficiently solve this optimization problem with sparse Newton methods, which can be directly applied to SLAM (LM97).

In (Del05) we introduced "Square-Root Smoothing and Mapping (SAM)", in which the sparse graphical nature of SLAM is fully exploited and particular attention is paid to reordering the variables, which directly affects the efficiency of the associated sparse optimization problem. We extended this approach to distributed SLAM in (DKK05), by using a method called "multifrontal QR-factorization"(Mat94; LB96). In this method, the underlying data structure is a clique tree (also called a junction tree), as described in (PS92; BP93). The problem of finding a good ordering for a sparse QR-factorization is considered in (HM96). (ADD96) describes a good general purpose ordering based on an approximate minimum degree ordering (AMD).

Our approach as explained below rests on a good partitioning of the graph associated with SLAM, for which we use the Metis library(KK98), This library implements several very efficient graph partitioning algorithms and it can be downloaded freely. (LT79a; LT79b) prove that a system of linear equations which

is equivalent to an almost-planar graph can be solved in $O(n^{3/2})$. Under the assumption of a sufficiently bounded sensor range and a scenario without closed loops, the SLAM-graph is almost planar and the upper bound of $O(n^{3/2})$ holds.

3 Solving SLAM with Smoothing

In this section, we describe how to solve SLAM with a smoothing approach. We derive the SLAM inherent objective function, which we optimize to find the optimal solution for the map and the trajectory. Next, we discuss all elements which are necessary for a Levenberg-Marquardt optimization, which are linearization, factorization and Gaussian elimination.

(TMHF00) is an excellent and very detailed survey of bundle adjustment in computer vision. This survey summarizes in detail, how to efficiently solve this optimization problem with sparse Newton methods, which can be directly applied to SLAM. The squared root formulation of SLAM is used in (LM97; Bie78). Another introduction is (Del05).

3.1 The SLAM inherent Objective Function

SLAM can be considered as a relaxation within a graph, which consists of nodes $\theta = (X, L)$ (poses and landmarks) and constraints $\psi = (U, Z)$ between these nodes. A constrain between two poses is an odometry control input u, while a constraint between a pose and a landmark is a measurement z. Contradicting constraints introduce tension into the graph and the optimization task is to find θ, which minimizes this tension.

First, we define a *prediction function* f, which calculates the constraints given the nodes, $\psi = f(\theta)$:

$$f(\theta) = f([X, L]) = [U', Z'] \tag{1}$$

$$u'_i = \hat{u}(x_i, x_{i+1}) + v_i \qquad z'_{i,j} = \hat{z}(x_i, l_j) + w_i \tag{2}$$

v_i and w_i correspond to a Gaussian noise with zero mean, which models the motion and measurement uncertainty. Next, we define an error function, which calculates the difference between the predicted and the "measured" constrains:

$$b(\theta,\Psi) = f(\theta) - \psi \quad \chi^2 = \|b(\theta,\psi)\|^2_\Sigma \tag{3}$$

The optimal solution is defined by $\theta^* = \underset{\theta}{argmin} \quad \chi^2(\theta,\psi)$, while the remaining residual is $e = \left\|\chi^2(\theta^*,\psi)\right\|^2_\Sigma$.

Now, we can find θ^* with Gauss-Newton iterations, given a start point θ^0:

$$\theta^{i+1} = \theta^i - \left(\nabla^2\chi^2(\theta^i)\right)^{-1} \cdot \nabla\chi^2(\theta^i) = \theta^i - H^{-1} \cdot g \tag{4}$$

The constraints ψ are corrupted by noise and we trust different components of the measurements differently. Therefore we need to consider in 3 all measurements within the Mahalanobis space $\|u_i\|^2_{\Sigma_i}$ respectively $\|z_i\|^2_{\Lambda_i}$. Because of $\|x\|^2_\Sigma = x^T\Sigma x = (\Sigma^{-T/2}x)^T(\Sigma^{-T/2}x) = \left\|\Sigma^{-T/2}x\right\|^2_2$ the squared Mahalanobis distance can be always computed by pre-dividing each constraint with the related standard deviation, $u_i = u_i \backslash \sigma_u$ and $z_i = z_i \backslash \sigma_z$.

3.2 Linearized System

The prediction function of the odometry and the landmark measurements are in general non-linear functions, especially in respect to the angular components. Following the standard SLAM literature, we compute the first-order linearized version of the motion model by

$$u^0_i + \delta u_i = \hat{u}_i(x^0_i, x_{i+1}) + F^i\delta x_i + G^{i+1}\delta x_{i+1} + v_i \tag{5}$$

where $F^i$and G^{i+1} are Jacobians of $\hat{u}_i(.)$ at the linearization point $\left(x^0_i, x^0_{i+1}\right)$, defined by

$$F^i = \frac{\partial\hat{u}_i(x_i, x_{i+1})}{\partial x_i} \qquad G^{i+1} = \frac{\partial\hat{u}_i(x_i, x_{i+1})}{\partial x_{i+1}} \tag{6}$$

The linearized landmark measurement equations are obtained similarly,

$$z_k = \hat{z}_k(x^0_{i_k}, l^0_{j_k}) + H^{i_k}_k\delta x_{i_k} + J^{j_k}_k\delta l_{j_k} + w_k \tag{7}$$

where $H_k^{i_k}$ and $J_k^{j_k}$ are the Jacobians of $\hat{z}_k(.)$ with respect to a change in $x_{i_k}$and l_{j_k}, and they are evaluated at the linearization point $(x_{i_k}^0, l_{j_k}^0)$:

$$H_k^{i_k} = \frac{\partial \hat{z}_k(x_{i_k}, l_{j_k})}{\partial x_{i_k}} \qquad J_k^{j_k} = \frac{\partial \hat{z}_k(x_{i_k}, l_{j_k})}{\partial l_{j_k}} \tag{8}$$

3.3 Measurement Matrix A

The gradient of a linearized prediction function is expressed by the measurement matrix *A (*information matrix*)*, which consists of all Jacobians, while the right-hand side (RHS) *b* represents the difference between the predicted and measured constrains:

$$A := \nabla b \tag{9}$$

We obtain the following standard least-squares problem,

$$\theta^* = \underset{\theta}{argmin} \ \left\| A(\theta^i, \psi) \cdot \Delta\theta^i - b(\theta^i, \psi) \right\|_2^2 \tag{10}$$

which is our starting point below. *A* can grow to be very large, but is quite sparse. 11 is an example measurement matrix, with three poses, two landmarks and four landmark measurements. The entire graph can be rotated an translated without introducing tension in it. To fix the graph, we add an prior G_1^1 to the measurement matrix. The upper part represents the odometry chain, while the lower part of *A* corresponds to the landmark measurements. The non-zero pattern unveils the adjacency of the measurement graph. The measurement graph and the measurement matrix are equivalent.

$$A = \begin{bmatrix} G_1^1 & & & & & \\ F_2^1 & G_2^2 & & & & \\ & F_3^2 & G_3^3 & & & \\ H_1^1 & & & J_1^1 & & \\ H_2^1 & & & & J_2^2 & \\ & H_3^2 & & J_3^1 & & \\ & & H_4^3 & & J_4^2 & \end{bmatrix} \tag{11}$$

3.4 Factorization

In a standard Gauss-Newton iteration of 10, we solve for $\Delta\theta^i$ and add it to the current linearization point $\theta^{i+1} = \theta^i + \Delta\theta^i$. Afterward we re-linearize A and b and solve again for $\Delta\theta^{i+1}$and so on. $\Delta\theta^i$ can be obtained by a QR factorization and a following back-substitution.

QR

QR-factorization for A yields R, where Q is orthogonal and represents the product of all rotations which are necessary to transform A to R. For the Gaussian elimination (in following often call back-substitution) we need $Q^T b$, which can be obtained by factorizing A with b as an additional column. Therefore Q^T doesn't need to be computed explicitly:

$$[A|b\,] \overset{qr}{\rightarrow} [R|Q^T b] \tag{12}$$

Because the Q factor is orthogonal, we have:

$$A = QR \;\Rightarrow\; Q^T A = \begin{bmatrix} R \\ 0 \end{bmatrix} \qquad Q^T b = \begin{bmatrix} d \\ e \end{bmatrix} \tag{13}$$

$$\|A \cdot \Delta\theta - b\|_2^2 = \left\|Q^T A \cdot \Delta\theta - Q^T b\right\|_2^2 = \|R \cdot \Delta\theta - d\|_2^2 + \|e\|_2^2 \tag{14}$$

$\|e\|_2^2$ will be the least-squares residual, and $\Delta\theta$ can be obtained by Gaussian elimination of the squared system:

$$R \cdot \Delta\theta = d \tag{15}$$

3.5 Levenberg-Marquardt

The optimum of a function can be found by a nonlinear optimization, which assumes that the function is well approximated by a quadric form close to the optimum. If the initial estimate is good, we know how to jump into the direction of the optimum. On the other side, a poor local approximation might be misleading. In the latter case we can go into the direction of the gradient.

Levenberg-Marquardt belongs to the family of trust region algorithms and can vary smoothly between a Gauss-Newton iterations and a gradient descent by varying λ :

$$(A^T A - \lambda I)\Delta\theta = A^T b \qquad \theta^{i+1} = \theta^i - \Delta\theta^i$$

There are several variations of LM, but all emphases diagonal elements of A. We decided to implement a variation of this LM, where a diagonal matrix is append to the measurement matrix

$$A' = \begin{bmatrix} A \\ \sqrt{\lambda} I \end{bmatrix} \quad A'^T A' = A^T A + \lambda I \tag{16}$$

Factorization of A' leads to a triangular matrix, with strong diagonal elements. λ determines the character of the optimization - the larger λ the more gradient descent rather than Gauss-Newton. We will increase lambda λ by factor 10, if one iteration increases the error, and vice versa. Also the larger elements become in A' the smaller is the update, in other words, if we are far away from the optimum λ is large and the step size is small.

4 Multifrontal QR

We described in 3.4, how to compute $\Delta\theta^i$ with a factorization and a following back-substitution. In this section we describe how to solve both parts with a multifrontal approach, The basic data structure is an assembly tree, which is a variant of a junction tree, which is widely use in standard literature. Information propagation in the multifrontal approach is very similar to the one in belief propagation. A detailed example can be found in the appendix.

4.1 Building an Assembly Tree

In this section we explain how to build an assembly tree, which is the basic data structure for the multi-frontal approach(Mat94; LB96).

Symbolic Marginalization

In the standard SLAM literature Kalman filtering is described as marginalizing variables, which is equivalent to removing these variables from the system and propagating their information to remaining parts, through additional or updated constraints between them (entries in the Hessian or the information matrix).

In graphical SLAM the start point is a factor graph, which is an undirected measurement graph, where nodes correspond to landmarks or robot poses $\theta = [X, L]$ and edges to odometry and landmark measurements $\psi = [U, Z]$. The first step to obtain an assembly tree is the *symbolic marginalization* of all variables; in other words remove a node from the graph, connect all its neighbors (form a clique) and store them. Here, we assume that the elimination ordering is given. The symbolic marginalization - often call *triangulation* - yields an *elimination chain* (*e_chain*), which provides clique of neighbors for each node, at the time when this node is marginalized. These neighbor cliques can be considered as a *Markov Blanket*; given the variables of this clique the node is independent of all other variables in the graph. In following algorithm we summarize how to obtain the e_chain:

Algorithm 1 Symbolic Marginalization

```
given G, Ord
e_chain = {}
for each x in order of Ord do
   connect all neighbors of x in G (form a clique)
   separator = NEIGHBORS(x, G)
   remove x from G
   e_chain ← e_chain ∪ {(x, separator) }
end
return e_chain
```

Junction Tree

In the previous section we described how to obtain an e_chain, which is a list of cliques indexed by the eliminated variable. Now, we build with these cliques a *clique tree* in a way that the *running intersection property* holds, which yields a *junction tree*. The running intersection property requires that all cliques between two cliques, which contains the same variable, have to contain this variable as well. Our junction tree consists of nodes. Each node contains a *variable clique* and is labeled by a clique_id. We link nodes and store on the edge a *node separator clique.*

The junction tree is build by iterating over the e_chain in reverse order. First, we initialize the tree root with the last element X of the ordering. The node separator is empty, the frontal clique and the clique_id is set to X. Then we continue in reverse order and either merge the current variable into its parent node or we add a new node to the tree. The parent variable is defined by the variable in the separator variable, which is the first in respect to the elimination ordering. If the separator clique is equal to the parent's variable clique

then we merge, else we add a new node to the tree. For this algorithm we also need to map a variable to a frontal clique, which is done by the clique_function (cf). In 2 we define the structure of a junction tree node and we summarize how to build a junction tree.

Algorithm 2 building a Junction Tree

```
a node of an JTree is defined by:
  (parent_clique_id, separator clique) - (clique_id, variable clique) - successors
given: e_chain, Ord
root  ← last(Ord)
cf    ← {(root, root)}        //clique_function: maps a variable to a clique
JTree ← {} - (root, {root}) - {}
for each x in reverse order of Ord/{root} do
   separator  ← E_CHAIN (x)
   parent     ← FIRST_IN_ORDER (Ord, separator)
   clique_id  ← CF(parent_id)
   clique     ← JTREE (clique_id)
   if |separator| = |clique| clique then
      // separator(x) = clique(parent(x))
      JTree ← UPDATE (add x to clique(parent(x))
      cf    ← CF {(x, clique_id)}
   else // add a new clique to JTree
      JTree ← EMBED [(parent, separator) - (x, {x} ∪ separator) - {}]
      cf    ← CF ∪ {(x, x)}
   end
end
return JTree, root
```

Assembly Tree

A junction tree node contains a variable clique C and a separator clique S. $F = C \setminus S$ is exactly the set of variables, which can be computed in a multifrontal approach given S; F is called *frontal clique*. To obtain an *assembly tree*, we take a junction tree and replace C by F in each node. The running intersection property doesn't de-facto hold anymore, but the global consistency holds, because this assembly tree is equivalent to a junction tree, where we explicitly store the separator.

The frontal variables of a node N are independent of all frontal variables from the nodes above N given the separator variables. A variable can occur only in one frontal clique, but in several node separator cliques.

4.2 Solving with Multifrontal QR and Assembly Tree

Factorization (bottom-up)

An assembly tree provides all structural information, which is needed for an multifrontal QR-factorization. A full-rank matrix A can be factorized into a matrix R by recursively traversing the tree. Each node forms a *frontal matrix*, which consists of update matrices from its children and a part of A. This frontal matrix will be densely factorized. The result matrix will be split into a piece of the final factorized matrix R and an update matrix, which will be propagated to the parent.

The part of A, which will be gathered for the frontal matrix of a node, are all rows of A, which involves frontal variables of this node. After gathering, these rows will be removed from A. This ensures that they are used only once. The node separator describes, which other variables occur in the gather rows, besides the frontal variables. Children gather rows before their parents, hence they are served first with parts of A, while children of the same parent node will never compete for the same row of A. The factorization of a node is summarized in 3.

Algorithm 3 Factorization of a Node

Given node j, calculate R_j and an update matrix U_j:

1. calculate the update matrices for children $\{U_C\}$
2. gather rows $A(f_j)$ from A, which involves frontal variables $\{f_j\}$, and remove them from A
3. form the frontal matrix F_j:
$$F_j = \begin{bmatrix} A(f_j) \\ U_{c_1} \\ \vdots \\ U_{c_s} \end{bmatrix}$$
4. factorize the frontal matrix F_j using dense QR:
$$Q_j^T F_j = \begin{bmatrix} R_{jj} & R_{j,j+1:n} \\ 0 & U_j \end{bmatrix}$$
5. use $R_j = \begin{bmatrix} R_{jj} & R_{j,j+1:n} \end{bmatrix}$ as the j^{th} slice of R, and return U_j as the update matrix for node j

---

Back-Substitution (top-down)

For back-substitution the assembly tree will be traversed again recursively from the root to the leaves. A node calculates the updates for all of its frontal variables by Gaussian elimination given the R-piece of the node and the updates of the node separator variables, which are computed in the tree above the node. An update of a variable will be propagated to a child node if and only if its separator clique contains this variable. Back-substitution leads to an update vector $\Delta\theta^i$ which will be added to the linearization point θ^i. In 4 is back-substitution summarized.

Algorithm 4 Back-substitution for a Node

---

given a node j with $R_j = \begin{bmatrix} R_{jj} & R_{j,j+1:n} \end{bmatrix}$, the updates for the node separator variables $\{\Delta f_k\}$ and the right-hand-side d, calculate updates for frontal variables $\{\Delta f_j\}$ by:

1. eliminate node separator variables:

$$d' = d - R_{j,j+1:n} \cdot \begin{bmatrix} \Delta f_{j+1} \\ \vdots \\ \Delta f_n \end{bmatrix}$$

2. use Gaussian elimination to calculate updates for the frontal variables:

$$\begin{bmatrix} \Delta f_{j_1} \\ \vdots \\ \Delta f_{j_k} \end{bmatrix} \overset{Gauss}{\longleftarrow} R_{jj} = d'$$

3. propagate an update to a child, iff its node separator contains the corresponding variable

---

Summary

- *Factorization* can be performed from bottom to top of the assembly tree, where each node propagates its update matrix to its parent, while the *R*-piece remains locally.
- *Back-substitution* can be performed from top to bottom of the assembly tree, where the updates of frontal variables are computed by the local *R*-piece and the updates of the node separator variables, which are computed above the node. The updates are propagated to children, if necessary.

5 Multifrontal approach

Where are the benefits of this multifrontal approach? Isn't this just an linear algebra technique to make a black box algorithms fast? The major advantage of this approach are the multiple fronts; a large least-squares-problem is represented in an tree, where a subtree corresponds to a sub-least-squared-problem. Assume the solution for a subtree is already known, then you might not to consider this subtree anymore in the computation. The subtree represents a sub-graph of the original entire measurement graph, which can be interpreted as a submap. Another scenario might be a multi-robot system, where robots see only few structure in common. Each robot represents a subtree of the joint assembly tree, where the top part of the tree corresponds to the shared structure and depending on the elimination ordering also some poses. Computation can be done locally, while communication is minimized.

5.1 Partitioned SLAM

In 3 we considered SLAM as a graph with nodes θ and constraints ψ between these nodes. The graph or map $M = [\theta, \psi]$ can be partitioned into *disjunct submaps* M_i:

$$M \quad \mapsto M_0,\ \{M_1, ..., M_n\}, \qquad [\theta, \psi] \mapsto [\theta_0, \psi_0],\ \{[\theta_1, \psi_1], ..., [\theta_n, \psi_n]\} \tag{17}$$

for $i > 0$, ψ_i is the set of all INTRA-*submap constraints*, which involves only nodes from θ_i, while $\{\theta_i\}$ are pairwise disjunct. The remaining nodes θ_0 and constrains ψ_0 are not assigned to any submap. Assume the optimal solution for the submaps are a priori known, then only the alignment of the submaps relative to each other need to be found, in other words to find a rigid transformation, which puts the pieces right together.

Therefore we assign to each submap a frame, its origin is in following called *basenode*. These basenodes yield a parametrization of the SLAM system, where all submap nodes are relative to their basenode. A transformation of a basenode effects directly the absolute position of its assigned nodes and corresponds to a rigid transformation of the submap. The final partitioning is:

$$M \quad \mapsto [b_0, \theta_0, \psi_0],\ \{[b_1, \theta_1, \psi_1], ..., [b_n, \theta_n, \psi_n]\} \tag{18}$$

ψ_0 are now constraints between nodes from different submaps, which need to be linearized differently, because they involve nodes expressed in a different frame. ψ_0 are INTER-*submap constraints*, which can be

linearized by:

$$u_i = \hat{u}(b_{\pi(i)}, x_i, b_{\pi(i+1)}, x_{i+1}]) + v_i \qquad z_{i,j} = \hat{z}(b_{\pi(i)}, x_i, b_{\pi(j)}, l_j) + w_{i,j} \tag{19}$$

$$\begin{aligned} B^{\pi(i)},\, F^i,\, B^{\pi(i+1)},\, G^{i+1} &= \tfrac{\partial \hat{u}}{\partial b_{\pi(i)}},\, \tfrac{\partial \hat{u}}{\partial x_i},\, \tfrac{\partial \hat{u}}{\partial b_{\pi(i+1)}},\, \tfrac{\partial \hat{u}}{\partial x_{i+1}} \\ B^{\pi_1(i)},\, H^i,\, B^{\pi_2(j)},\, J^j &= \tfrac{\partial \hat{z}}{\partial b_{\pi_1(i)}},\, \tfrac{\partial \hat{z}}{\partial x_i},\, \tfrac{\partial \hat{z}}{\partial b_{\pi_2(j)}},\, \tfrac{\partial \hat{z}}{\partial l_j} \end{aligned} \tag{20}$$

The Jacobian of such an INTER-submap constraint involves four parts.

5.2 Submap Tree with Separator Ordering

Suggest, we cut the system by a line into two parts, which assigns each node to a side and therefore to a submap. The set of all nodes which are connected to a node from another submap defines the *cut separator*. Given this cut separator the submaps are independent of each other. This cut separator defines an elimination ordering, where the cut separator variables are eliminated last and which we named *separator ordering*. The ordering effect enormously the fill of R and therefore the costs of factorization but also the back-substitution. To minimize the cost, we optimize the ordering by AMD. Therefore we consider the sub-graph, which contains only separator variables. Afterward, AMD finds a very efficient ordering of this sub-system. The same is done for the none-separator variables. This partitioned AMD ordering is a good approximation to the global AMD ordering.

With this ordering we can build an assembly tree, where the root area of the tree contains nodes with only cut separator variables and different subtrees include variables from different submaps. Each assembly tree subtree represents a connected submap. Notice, that in case of the line cut the number of subtrees might be larger than two, because the cut might have divided one side into several disconnected parts. An arbitrary partitioning can be defined a by a cut separator, which can be obtained by any graph partitioning algorithm. Afterward, an assembly tree is build with the implied separator ordering. The submaps are obtained by detecting the subtrees of the assembly tree. Submap variables (frontal variables of the subtree) will be assigned to a basenode. The root node of a subtree contains a node separator, which is also the separator of the whole subtree, that means given these variables the subtree is independent of other parts of the tree. We named the node separator of a subtree root *subtree separator*.

For the further process we require that subtree separators are disjunct, that submaps are full-ranked and that they consist of at least k variables. For each subtree root we check all three criteria and if on of

those is violated then the frontal variables of this subtree root will be added to the cut separator and its children will be added to the list of subtree roots, which will be checked. In the worst case, a subtree might be completely absorbed by the cut separator, which is for a regular SLAM-graph and a reasonable graph cut pretty unlikely. To checked whether a subtree is full-ranked either the complete subtree system can be factorized or the sub-graph might be analyzed.

In the case of a 3-dimensional robot poses (x, y, θ) and a 2-dimensional landmark (x, y) a SLAM-graph is full-ranked, if the graph is connected, two separated odometry chains are linked with at least two landmark measurements and one arbitrary pose has a prior. If a subtree root meet all three criteria it will be accepted and its separator variable are assigned to the related submap.

This post processing yields a *submap tree*. Now a SUBMAP can be formally defined by all frontal variables of subtree plus the subtree separator variables. It is assert that these submaps are disjunct, full-ranked and of a minimal size. The algorithm is again described in 5.

Algorithm 5 Building a Submap Tree

---

given a measurement graph G
cut_separator (CS)$\leftarrow$ cut (G)
separator_ordering (SO) $\leftarrow$ AMD $\{x|x \notin CS\}$, AMD $\{x|x \in CS\}$
$atree \leftarrow$ BUILD_ASSEMBLY_TREE (G, SO)
$roots \leftarrow$ DETECT_SUBTREE_ROOTS ($atree$, CS)

for each *root* **in** $roots$ **check:**

1. $|subtree > k|$
2. *subtree* is full-ranked
3. $subtree\_separator$ is disjunct from all subtree_separators of previous accepted $roots$

if all three criteria are satisfied *then* **ACCEPT** *root* ***else*:**

1. add all frontal variables of *root* to the CS
2. add children of *root* to $roots$

---

6 Partially Fixed Linearization Point

6.1 The Idea

In this section we describe how we can use the structure of the multifrontal approach to accelerate factorization, which makes the smoothing approach even in very large system practical. This acceleration preserves optimality, while other approaches need to approximate. The idea is to optimize first small independent subsystems and then fix their linearization point. Hence, these parts needn't to be re-linearized and factorized again. Each sub-system will be represented by a fixed R-piece and a fixed update matrix. Next, we optimize the entire system, by re-linearization and factorizing only the parts, which are related to the cut separator (top of the assembly tree). Factorization involves the re-linearized cut separator and the fixed update matrices, while back-substitution uses the fixed parts of R which belong to the subtrees, but also the recently calculated parts of R, which are related to the cut separator.

First, subtrees are optimized to obtain a good linearization point for them. Then, subtrees won't be re-linearized and factorized anymore; hence the factorization and therefore the optimization of the complete system is very fast. We call this approach: *Partially Fixed Linearization Point* (PFLP).

6.2 Good Linearization Point for Subtrees

The first step is to find a good linearization point for the subtrees. The submap tree asserts that the subtree separators, the node separator of the subtree roots, are pairwise disjunct. Therefore we can assign these subtree separators to the related submap. A submap is defined by all frontal variables of subtree plus the subtree separator. The submap tree also guarantees that the submaps are full-ranked, after we have added a prior to one pose of each subtree. The solution for each subtree can be found iteratively as describe in 4.2. When all subtrees are sufficiently optimized, we remove all subtree priors and factorize the subtrees once to obtain R-pieces and update matrices for each subtree, which we store. Notice, that back-substitution isn't performed. We also store the current values of the subtree variables, which are the linearization point related to the fixed R-pieces and update matrices.

6.3 Fast Globally Smoothing

The next step find the right alignment of the submaps. The linearization point of the submaps is obtained by separating them from other parts of the graph; hence outgoing constraints were not involved. These outgoing

constraints might change the optimal position of the submap nodes in the entire graph in comparison to the separated submap graph. If the linearization point of the subtrees is good enough, a global optimization with fixed R-pieces and update matrices will find the optimal solution of the entire graph.

While traversing the tree for factorization, one might only check, whether a child is a subtree root, then its cached update matrix is gathered. Back-substitution will be performed for the complete system. Variations can even accelerate back-substitution, which needed to be done for the subtrees for each iteration, because in each iteration the updates of the subtree variables will be added always to the same fixed linearization point.

6.4 Experiments

To show the benefits of the partially fixed linearization point for the submap parametrized SLAM, we simulated a robot's walk in a block world. We generated three different walks, which are partitioned by Metis. 2 shows a walk where three different rooms (a) are connected by a hall way. The other two walks are randomly generated and include 280 and 500 poses. Graph partitioning doesn't guarantee to yield connected parts. (a) was partitioned into five parts, which correspond to seven full-ranked, connected subtrees and therefore seven submaps. (b) is partitioned into two parts, which are connected by a small cut separator. (c) is also partitioned into two parts, but the cut separator is not connected and it is not local.

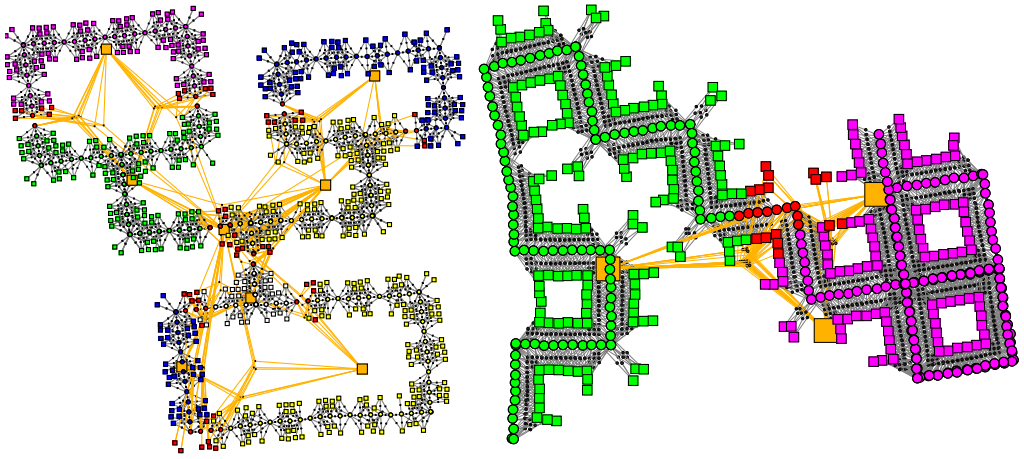
1 shows for all three walks necessary timing for factorization, linearization and back-substitution. Our experiments runs on a 2.4GHz PC with 2GB RAM. Our multifrontal QR-factorization is implemented in OCaml, which is not compatible with a highly optimized C code. Factorization of the entire system of (c) takes 17.4 s, while a LDL-factorization implemented in C needs for the same task only 0.0644s. These experiments are supposed to show the relative speed-up for factorization rather than the absolute performance, while the timing for LDL shows that the smoothing approach is feasible for larger systems.

Timing results are obtained for the *linearizeAll*, which is a standard complete LM and the entire system is always re-linearized, *subtree-phase*, where a good linearization point for the submaps are found and *complete* factorizes only the root area of the assembly tree, where the fixed submap linearization is used. The back-substitution uses the fixed R-pieces and the root updates for the submap variables. The speed-up is 1-2 order of magnitudes. (c) has a large disconnected cut separator, where the basenodes represent a wide spread submap. Hence, the basenodes can't compensate locally corrections in the cut section of a submap, because manipulation of the basenodes will effect the entire submap and therefore also other parts of the

world	phase	factorization	linearization	back-substitution
3R	subtree	1.117	0.107	0.099
	complete	0.204	0.060	0.075
	linearize all	1.263	0.116	0.107
W280	subtree	3.421	0.126	0.095
	complete	0.077	0.051	0.082
	linearize all	3.663	0.131	0.094
W500	subtree	17.40	0.247	0.198
	complete	2.82	0.045	0.181
	linearize all	21.1	0.261	0.149

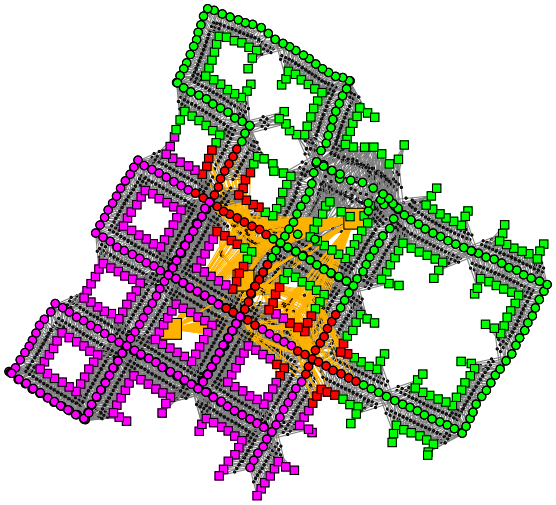
Figure 1: timing results for partially fixed linearization points for one iteration

non-locally cut separator parts. In the case of a non-locally cut separator the system might solved by a complete LM-step, where all parts are always re-linearized to find the optimal solution. On the other side, (a) and (b) involves only small and locally cut separators between two submaps. Submaps are also smaller and therefore a basenode is more *local* and the entire system can be solved by first finding a linearization point for the subtrees, then fix it and factorize only the cut separator.



(a) 3-room walk, 5 parts

(b) W280, 2 parts



(c) W500, 2 parts

Figure 2: simulated walks

6.5 Discussion

The major advantage of the presented partitioning is that the subtrees are fairly independent under the assumption that the fixed linearization of the subtrees are sufficiently close to the globally optimal one. Then one might think about several application.

Distributed Inferences

The multi-frontal approach supports distributed computation by default. Multiple robots might explore an environment, where two robots have only a small observed area in common. A set of subtrees corresponds to robots. Then the robots could distributed find the MAP solution for their observations, as we showed in (DKK05). The linearization point of a robot can be fixed. The other robots might move further and so its map could be extended under the assumption that the cut separator doesn't change.

Submaps

Submaps are a variation of the distributed case. Subtrees of the assembly tree correspond to submaps. Assume that a robot moves only in a particular room then this room could be extended in an incremental fashion again under the assumption that the cut separator doesn't change.

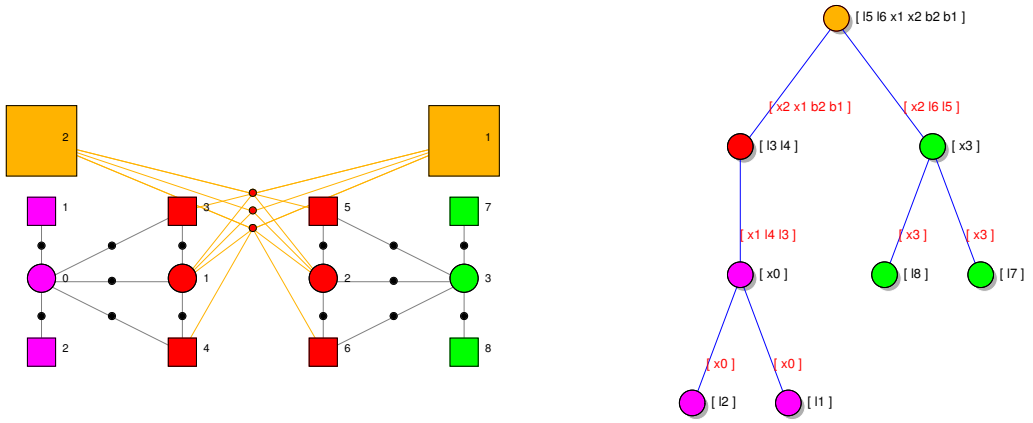
Conclusion

We presented an approach, which partitions a large SLAM graph into smaller sub-graphs, which are independent of each other. A partition is in a multiple robots scenario related to a single robot and in the general case to a submap. We parametrized the SLAM optimization problem in a submap manner, where a set of graph nodes are expressed relative to its basenode. Also a multifrontal QR-factorization is reviewed, which could be replaced by any factorization like Cholesky or LDL-factorization. This multifrontal approach has the advantage that the linearization point of the submaps can be fixed; then a global optimization requires only a factorization of small parts of the entire system. This approach is exact under the assumption that the fixed linearization point is sufficiently close to the optimal one. Assume we fix the linearization of all submaps except of submap A. Then A can be manipulated in a way, that the cut separator doesn't change. A could be extended in an incremental manner or data association might be optimized under the assumption that the other parts are correct. Future work would be to extend the current more batch-like approach and consider questions like, how to detect an appropriate cut on-the-fly or what to do, if the cut separator changes.

7 Appendix

7.1 Detailed Example

Fig. 3(a) shows a small measurement graph. Squares are landmarks, while circles are robot poses. The large orange squares are basenodes, in other words the origin of the submaps' frame. Constraints are drawn by small black circles; involved nodes are connected to these constraints. In our case constraints can be odometry or landmark measurements. The constrain nodes contain the Jacobians of the consecutive nodes.



(a) Measurement graph: circles are poses, squares are landmarks and orange squares are basenodes. Graph is vertically cut in the middle.

(b) Assembly tree: subtrees corresponds to submaps, the cut separator is at the top.

Figure 3: A simple measurement graph and the related assembly tree

The graph is horizontally cut into two pieces. All nodes which are connected to a node from another submap belong to the cut separator (red). The Jacobians of constraints between nodes from different submaps (orange) involve also basenodes. These constraints define the relative alignment of submaps.

After determining the separator ordering, an assembly tree can be built (b), where each node contains a frontal clique and on the edge connected to its parent node sits a node separator. The frontal variables are independent of all variables above in the tree, if the node separator variables are given. The assembly tree shows, that submaps correspond to subtrees of the assembly tree, while the cut separator nodes represent the root area. The node separator of a subtree root is also the subtree separator. All subtree separator variables will be associated with a subtree (different subtree separators are disjunct) and remaining cut separator

variables will be assigned to a 'global frame' b_0, like the basenodes.

SEPARATOR ORDERING

Building an assembly tree requires an ordering, where separator variables are eliminated last. The ordering within the separator and the none-separator variables are optimized with an AMD ordering, which reduces the fill of the factorized matrix R and decreases therefore the computation cost of the factorization. For this example, we choose a separator ordering, which doesn't optimize with AMD, it is just ordered:

{none-separator} - {separator}: {l1, l2, x1, l7, l8, x4, l3, l4} - {x2, l5, l6, x3}

ELIMINATION_CHAIN

Given the elimination ordering we can do the symbolic marginalization or triangulation, where we remove a node and connect all neighbors with each other (form a clique). The elimination_chain stores the current neighbors of a node, when it is marginalized.

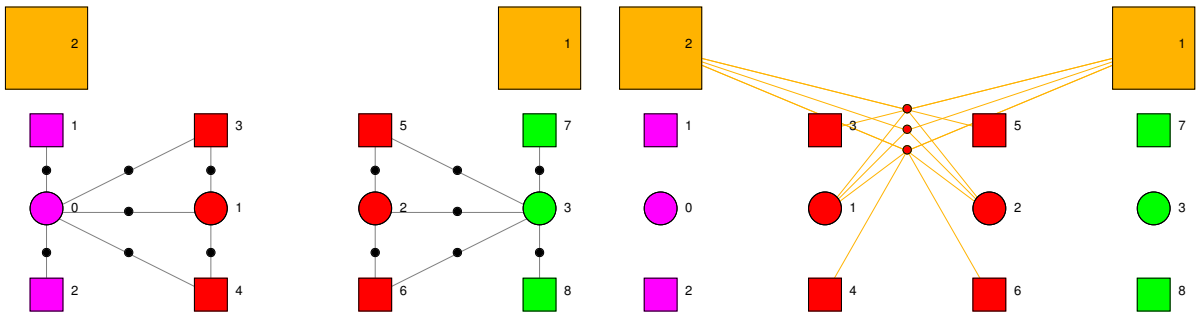
(l1 - x0), (l2 - x0), (x1 - x1, l3, l4), (l7 - x3), (l8 - x3), (x4 - x2, l5, l6), (l3 - l4, x2, x3), (l4 - x2, x3), (x2 - l5, l6), (l5 - x2, l6), (l6 - x2), (x3 - ∅)

ASSEMBLY TREE - SUBMAP TREE

With the elimination_chain an assembly tree [Fig. 3(b)] can be build as described in 4.1. The next step is to detect subtree of the assembly tree and check whether following criteria hold:

1. $|subtree > k|$
2. $subtree$ is full-ranked
3. $subtree\_separator$ is disjunct from all subtree_separators of previous accepted $roots$

If not, the checked subtree root will be added to the cut separator and its children will be checked. This procedure yields a list of subtree roots, which defines the submaps, with the required properties. Afterward submap variables and subtree separator variables will be assigned (if necessary) to a basenode. Finally, all basenodes will be appended at the tail of the separator ordering and the assembly tree is built again, the submap structure is preserved and we obtain a submap tree.

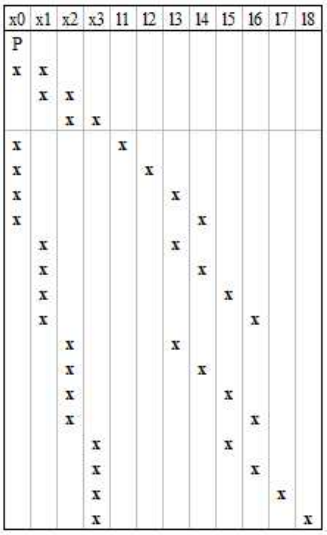


(a) Finding a good linearization point for the subtrees.

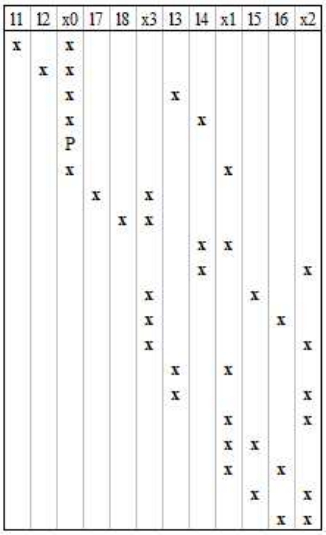
(b) A fast global optimization optimize the alignment of the submaps, while the linearization point of the subtrees is fixed.

Figure 4: Different parts of the measurement used in the subtree and the global optimization

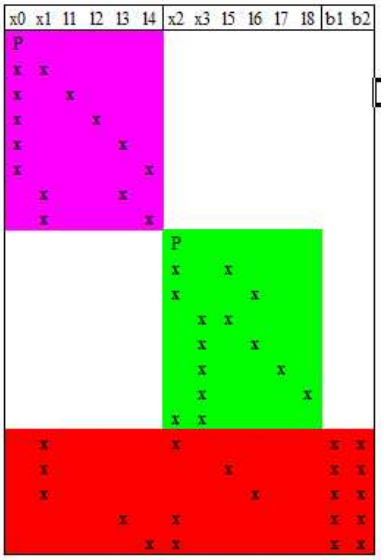
The measurement graph Fig. 5 is equivalent to the measurement matrix A. Columns are related to unknown variables, while rows are odometry or landmark measurements. The canonical ordering (a), shows the typical structure of the measurement matrix or information matrix, where the top part represents the odometry chain. A pose is connected to the successor pose and therefore also to the predecessor pose. Landmark measurements involve a pose and a landmark. All matrix entries are Jacobians of the prediction function, which yields and odometry or landmark measurement, given two poses or a pose and a landmark. (b) shows the matrix in with the separator ordering. During factorization parts (rows) of A are gather from the measurement from top to bottom. Gathered rows will be removed, hence they can't be uses anymore. The matrix is never re-ordered explicitly. (c) illustrates submaps within the measurement matrix. Pink and green are related to the INTRA-submap constraints, which are used to find a good linearization point for the submaps. Each submap has its own prior p. The red part is the cut separator, which contains all INTER-submap constrains. These constraints involve basenodes and define therefore the optimal position of the submaps relative to each other. Given the cut separator variables the variables of a submap are independent of variables from all other submaps.



(a) Canonical ordering of the measurement matrix, with the typical block structure. Top part represents odometry, the other part belongs to landmarks measurements.



(b) Separator ordering of the measurement graph, rows are also re-ordered.



(c) Measurement matrix is reorder in a way that the submap structure is unveiled. The red cut separator part connects the submap parts. Give the separator submaps are independent from each other.

Figure 5: SLAM in a matrix

7.2 Formulas

Standard

robot pose $p = [x, y, \theta]$, odometry $u = [\Delta x, \Delta y, \Delta\theta]$

$$\hat{u}(p_1, p_2) = \begin{bmatrix} \cos(\theta)(x_2 - x_1) & + & \sin(\theta)(y_2 - y_1) \\ -\sin(\theta)(x_2 - x_1) & + & \cos(\theta)(y_2 - y_1) \\ \theta_2 & - & \theta_1 \end{bmatrix} \qquad G = \frac{\partial \hat{u}}{\partial p_2} = \begin{bmatrix} \cos(\theta) & \sin(\theta) & 0 \\ -\sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{21}$$

$$F = \frac{\partial \hat{u}}{\partial p_1} = \begin{bmatrix} -\cos(\theta) & -\sin(\theta) & \sin(\theta)(x_2 - x_1) + \cos(\theta)(y_2 - y_1) \\ \sin(\theta) & -\cos(\theta) & -\cos(\theta)(x_2 - x_1) - \sin(\theta)(y_2 - y_1) \\ 0 & 0 & -1 \end{bmatrix} \tag{22}$$

$$\tag{23}$$

landmark $p = [x, y]$, *measurement* $p = [r, b]$ (range, bearing)

$$\hat{z}(x, l) = \begin{bmatrix} \|p - l\| \\ \arctan \begin{bmatrix} x_p - x_l \\ y_p - y_l \end{bmatrix} - \theta_p \end{bmatrix} \qquad H = \frac{\partial \hat{z}}{\partial x_p}, \qquad J = \frac{\partial \hat{z}}{\partial x_p} \tag{24}$$

Submaps

$$M \quad \mapsto [b_0, \theta_0, \psi_0], \ \{[b_1, \theta_1, \psi_1], ..., [b_n, \theta_n, \psi_n]\} \tag{25}$$

$$u_i = \hat{u}(b_{\pi(i)}, x_i, b_{\pi(i+1)}, x_{i+1}]) + v_i \qquad z_{i,j} = \hat{z}(b_{\pi(i)}, x_i, b_{\pi(j)}, l_j) + w_{i,j} \tag{26}$$

$$\begin{aligned} B^{\pi(i)},\ F^i,\ B^{\pi(i+1)},\ G^{i+1} &= \tfrac{\partial \hat{u}}{\partial b_{\pi(i)}},\ \tfrac{\partial \hat{u}}{\partial x_i},\ \tfrac{\partial \hat{u}}{\partial b_{\pi(i+1)}},\ \tfrac{\partial \hat{u}}{\partial x_{i+1}} \\ B^{\pi_1(i)},\ H^i,\ B^{\pi_2(j)},\ J^j &= \tfrac{\partial \hat{z}}{\partial b_{\pi_1(i)}},\ \tfrac{\partial \hat{z}}{\partial x_i},\ \tfrac{\partial \hat{z}}{\partial b_{\pi_2(j)}},\ \tfrac{\partial \hat{z}}{\partial l_j} \end{aligned} \tag{27}$$

References

[ADD96] P. R. Amestoy, T. Davis, and I. S. Duff, *An approximate minimum degree ordering algorithm*, SIAM Journal on Matrix Analysis and Applications **17** (1996), no. 4, 886–905.

[Bie78] G.J. Bierman, *An application of the square-root information filter to large scale linear interconnected systems*, IEEE Trans. Automat. Contr. **23** (1978), no. 1, 91–93.

[BNLT05] M. C. Bosse, P. M. Newman, J. J. Leonard, and S. Teller, *SLAM in large-scale cyclic environments using the Atlas framework*, Accepted for publication in the International Journal of Robotics Research, 2005.

[BP93] J.R.S. Blair and B.W. Peyton, *An introduction to chordal graphs and clique trees*, Graph Theory and Sparse Matrix Computations (J.A. George, J.R. Gilbert, and J.W-H. Liu, eds.), IMA Volumes in Mathematics and its Applications, vol. 56, Springer-Verlag, 1993, pp. 1–27.

[Del05] F. Dellaert, *Square Root SAM: Simultaneous location and mapping via square root information smoothing*, Robotics: Science and Systems (RSS), 2005.

[DKK05] F. Dellaert, A. Kipp, and P. Krauthausen, *A multifrontal QR factorization approach to distributed inference applied to multi-robot localization and mapping*, AAAI Nat. Conf. on Artificial Intelligence, 2005.

[ESL05] R. Eustice, H. Singh, and J. Leonard, *Exactly sparse delayed-state filters*, IEEE Intl. Conf. on Robotics and Automation (ICRA) (Barcelona, Spain), April 2005, pp. 2428–2435.

[FC04] J. Folkesson and H. I. Christensen, *Graphical SLAM - a self-correcting map*, IEEE Intl. Conf. on Robotics and Automation (ICRA), vol. 1, 2004, pp. 383 – 390.

[FH01] U. Frese and G. Hirzinger, *Simultaneous localization and mapping - a discussion*, Proceedings of the IJCAI Workshop on Reasoning with Uncertainty in Robotics, 2001, pp. 17–26.

[FLD05] U. Frese, P. Larsson, and T. Duckett, *A multilevel relaxation algorithm for simultaneous localisation and mapping*, IEEE Trans. Robototics **21** (2005), no. 2, 196–207.

[FPC05] J. Folkesson, P.Jensfelt, and H. I. Christensen, *Graphical SLAM using vision and the measurement subspace*, IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS), 2005.

[Fre05] U. Frese, *Treemap: An $O(\log n)$ algorithm for simultaneous localization and mapping*, Spatial Cognition IV, Springer Verlag, 2005, pp. 455–476.

[HM96] P. Heggernes and P. Matstoms, *Finding good column orderings for sparse QR factorization*, Second SIAM Conference on Sparse Matrices, 1996.

[JU01] S.J. Julier and J.K. Uhlmann, *A counter example to the theory of simultaneous localization and map building*, IEEE Intl. Conf. on Robotics and Automation (ICRA), vol. 4, 2001, pp. 4238–4243.

[KK98] G. Karypis and V. Kumar, *Multilevel algorithms for multi-constraint graph partitioning*, Supercomputing '98: Proceedings of the 1998 ACM/IEEE conference on Supercomputing (CDROM) (Washington, DC, USA), IEEE Computer Society, 1998, pp. 1–13.

[LB96] Szu-Min Lu and Jesse L. Barlow, *Multifrontal computation with the orthogonal factors of sparse matrices*, SIAM Journal on Matrix Analysis and Applications **17** (1996), no. 3, 658–679.

[LF01] J. J. Leonard and H. J. S. Feder, *Decoupled stochastic mapping*, IEEE Journal of Oceanic Engineering (2001), 561–571.

[LM97] F. Lu and E. Milios, *Globally consistent range scan alignment for environment mapping*, Autonomous Robots (1997), 333–349.

[LT79a] R.J. Lipton and R.E. Tarjan, *Generalized nested dissection*, SIAM Journal on Applied Mathematics **16** (1979), no. 2, 346–358.

[LT79b] ———, *A separator theorem for planar graphs*, SIAM Journal on Applied Mathematics **36** (1979), no. 2, 177–189.

[Mat94] P. Matstoms, *Sparse QR factorization in MATLAB*, ACM Trans. Math. Softw. **20** (1994), no. 1, 136–159.

[MT03] M. Montemerlo and S. Thrun, *Simultaneous localization and mapping with unknown data association using FastSLAM*, IEEE Intl. Conf. on Robotics and Automation (ICRA), 2003.

[Pas03] M.A. Paskin, *Thin junction tree filters for simultaneous localization and mapping*, Intl. Joint Conf. on Artificial Intelligence (IJCAI), 2003.

[PS92] A. Pothen and C. Sun, *Distributed multifrontal factorization using clique trees*, Proceedings of the Fifth SIAM Conference on Parallel Processing for Scientific Computing, Society for Industrial and Applied Mathematics, 1992, pp. 34–40.

[SC87] R. Smith and P. Cheeseman, *On the representation and estimation of spatial uncertainty*, Intl. J. of Robotics Research **5** (1987), no. 4, 56–68.

[TL03] S. Thrun and Y. Liu, *Multi-robot SLAM with sparse extended information filters*, Proceedings of the 11th International Symposium of Robotics Research (ISRR'03) (Sienna, Italy), Springer, 2003.

[TMHF00] B. Triggs, P. McLauchlan, R. Hartley, and A. Fitzgibbon, *Bundle adjustment – a modern synthesis*, Vision Algorithms: Theory and Practice (W. Triggs, A. Zisserman, and R. Szeliski, eds.), LNCS, Springer Verlag, 2000, pp. 298–375.