Treemap: An *O*(log *n*) Algorithm for Indoor Simultaneous Localization and Mapping

Udo Frese

SFB/TR 8 Report No. 006-03/2006
Report Series of the Transregional Collaborative Research Center SFB/TR 8 Spatial Cognition
Universität Bremen / Universität Freiburg

Contact Address:

Dr. Thomas Barkowsky
SFB/TR 8
Universität Bremen
P.O.Box 330 440
28334 Bremen, Germany

Tel +49-421-218-8625
Fax +49-421-218-8620
barkowsky@sfbtr8.uni-bremen.de
www.sfbtr8.uni-bremen.de

Treemap: An $O(\log n)$ Algorithm for Indoor Simultaneous Localization and Mapping

Udo Frese ufrese@informatik.uni-bremen.de[*†]
FB 3, Mathematik und Informatik,
SFB/TR 8 Spatial Cognition,
Universität Bremen
http://www.informatik.uni-bremen.de/~ufrese/

March 28, 2006

Abstract

This article presents a very efficient SLAM algorithm that works by hierarchically dividing a map into local regions and subregions. At each level of the hierarchy each region stores a matrix representing some of the landmarks contained in this region. To keep those matrices small, only those landmarks are represented that are observable from outside the region.

A measurement is integrated into a local subregion using $O(k^2)$ computation time for k landmarks in a subregion. When the robot moves to a different subregion a full least-square estimate for that region is computed in only $O(k^3 \log n)$ computation time for n landmarks. A global least square estimate needs $O(kn)$ computation time with a very small constant (12.37ms for $n = 11300$). Furthermore, the proposed hierarchy allows the rotation of individual regions to reduce linearization errors.

The algorithm is evaluated for map quality, storage space and computation time using simulated and real experiments in an office environment.

1 Introduction

The problem of building a map from local observations of the environment is a very old one, as old as maps themselves. While geodesy, the science of surveying in general, dates back to 8000 B.C., it was C.F.

[*]Sections 1–5, 7–10 have appeared in Autonomous Robots (Frese, 2006b).
[†]This article is based on the authors studies at the German Aerospace Center.

Gauss who first formalized the problem from the perspective of statistical estimation in his article *"Theoria combinationis observationum erroribus minimis obnoxiae"*[1] (1821).

In the much younger realm of robotics, the corresponding problem is that of simultaneous localization and mapping (SLAM). It requires the robot to continuously build a map from sensor data while exploring the environment. It has been a subject of research since the mid 1980s gaining enormous popularity in recent years. Most approaches adhere to the Gaussian formalization. They estimate a vector of n features, e.g. landmarks or laser scan reference frames, by minimizing a quadratic error function, i.e. by implicitly solving a linear equation system. With this well established methodology the main question is how to compute or approximate the estimate efficiently. To make this more explicit, we have proposed three important requirements which an ideal SLAM algorithm should fulfill (Frese and Hirzinger, 2001; Frese, 2006a).

> **(R1) Bounded Uncertainty** *The uncertainty of any aspect of the map should not be much larger than the minimal uncertainty that could theoretically be derived from the measurements.*
>
> **(R2) Linear Storage Space** *The storage space of a map covering a large area should be linear in the number of landmarks.*
>
> **(R3) Linear Update Cost** *Incorporating a measurement into a map covering a large area should have a computational cost at most linear in the number of landmarks.*

(R1) binds the map to reality and limits approximations. (R2) and (R3) regard efficiency, requiring linear space and time consumption. We feel that one should aim at (R1) rather than at the much weaker criterion of asymptotic convergence. Even after going through the environment only a single time, a useful map is desirable. Most sensor data allows this and, according to (R1) so should the SLAM algorithm.

The contribution of this article is *treemap*, a hierarchical SLAM algorithm that meets the requirements (R1)-(R3). It works by dividing the map into regions and subregions. When integrating a measurement it needs $O(k^2)$ computation time for updating the estimate for a region with k landmarks, $O(k^3 \log n)$ when the robot moves to a different region, and $O(kn)$ to compute an incremental estimate for the whole map. The algorithm is landmark based and requires known data association. It has two drawbacks. First, it requires a "topologically suitable building" (Sec. 4); and second, its implementation is relatively complex.

[1] "Theory of the combination of observations least subject to error."

		(R1)		(R2)		(R3)	
	UDA	non-linear	map quality	memory	update	global update	loop
ML	C	$\surd$	$\surd$	m		……$(n+p)^3$……	
EKF	C		$\surd$	n^2		………n^2………	
CEKF	C		$\surd$	$n^{\frac{3}{2}}$	k^2	….$kn^{\frac{3}{2}}$….	
Relaxation		$\surd$	$\surd$	kn	……kn……		kn^2
MLR		$\surd$	$\surd$	kn		……… kn ………	
FastSLAM	$\surd$	$\surd$	see §2	Mn		……$M \log n$……	
SEIF				kn		………k^2………	
w. full update			$\surd$	kn	……kn……		kn^2
TJTF	C	$\surd$	$\surd$	k^2n	k^3	….k^3n….	
Treemap	C	$\surd$	$\surd$	kn	k^2	..$k^3 \log n$..	
w. global map	C	$\surd$	$\surd$	kn		……… kn ………	

Table 1: Performance of different SLAM algorithms with n landmarks, m measurements, p robot poses and k landmarks local to the robot (cf. §2). UDA stands for 'Uncertain Data Association'. A $\surd$ means the algorithm can handle landmarks with uncertain identity. A C means covariance is available for performing χ^2 tests.

The article is organized as follows. After a brief review of related work (Sec. 2), we derive the algorithm (Sec. 3–7). It follows with a comparison to the closely related Thin Junction Tree Filter (Sec. 8) by Paskin (2003) and an investigation of map quality and computation time based on simulations (Sec. 9) and experiments in a 60m $\times$ 45m office building (Sec. 10). Finally, we discuss a nonlinear extension that reduces the linearization error caused by error in the robot orientation (Sec. 11). The appendix provides the algorithm's pseudocode (App. A) and a worked out example (App. B). This report is an extended version of an article with the same title appearing in Autonomous Robots (Frese, 2006b).

2 State of the Art

After the fundamental article by Smith et al. in 1988 most work on SLAM was based on the Extended Kalman Filter (EKF) that allows SLAM to be treated as an estimation problem in a theoretical framework. However, the problem of large computation time remained. The EKF maintains the posterior distribution of the robot pose and n landmarks as a $3 + 2n$ dimensional Gaussian with correlations between all landmarks. This is essentially for SLAM but requires to update the EKF's covariance matrix after each measurement, taking $O(n^2)$ time.

This limited the use to the order of hundreds of landmarks.

Recently, interest in SLAM has increased dramatically and several more efficient algorithms have been developed. Many approaches exploit the fact that observations are *local* in the sense that from a single robot pose only a few (k) landmarks are visible. In the following the more recent contributions will be briefly reviewed (Tab. 1). An overview is given by Thrun et al. (2005) and a discussion by Frese (2006a).

To meet requirement (R1) an algorithm must maintain some form of correlations in the whole map. To my knowledge the first SLAM algorithm achieving this with a computation time below $O(n^2)$ per measurement was the relaxation algorithm (Duckett et al., 2000, 2002). The algorithm employs an iterative equation solver called *relaxation* to the linear equation system appearing in maximum likelihood estimation. One iteration is applied after each measurement with $O(kn)$ computation time and $O(kn)$ storage space. After closing a loop, more iterations are necessary leading to $O(kn^2)$ computation time in the worst case. This was later improved by the Multilevel Relaxation (MLR) algorithm (Frese et al., 2004). This algorithm optimizes the map at different levels of resolution, leading to $O(kn)$ computation time.

Montemerlo et al. (2002) derived an algorithm called *FastSLAM* from the observation that the landmark estimates are conditionally independent, given the robot pose. Basically, the algorithm is a particle filter (M particles) in which every particle represents a sampled robot trajectory plus a set of n Kalman filters estimating the landmarks conditioned on the trajectory. The number of particles M is a difficult tradeoff between computation time and map quality. However, the algorithm can handle uncertain landmark identification (Nieto et al., 2003), which is a unique advantage over the other algorithms discussed. Later Eliazar and Parr (2003) as well as Stachniss and Burgard (2004) extended the framework to using plain evidence grids as particles (in a compressed representation). Their approach constructs maps in difficult situations without landmark extraction or scan matching.

Guivant and Nebot (2001, 2003) developed the *Compressed EKF* (*CEKF*) that allows the accumulation of measurements in a local region with k landmarks at cost $O(k^2)$ independent from n. When the robot leaves this region, the accumulated result is propagated to the full EKF (*global update*) at cost $O(kn^2)$. The global update can be approximated more efficiently in $O(kn^{3/2})$ with $O(n^{3/2})$ storage space needed.

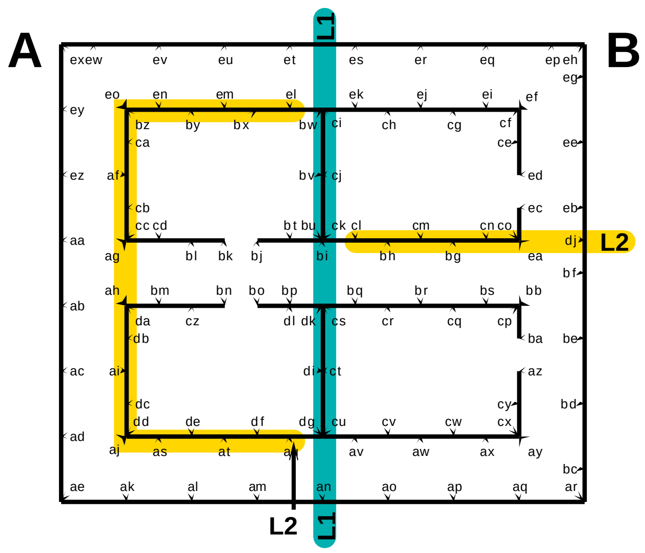
Thrun et al. (2004) presented a "constant time" algorithm called the *Sparse Extended Information Filter (SEIF)*, which represents uncertainty with an information matrix instead of a covariance matrix. The algorithm exploits the observation that the information matrix is approximately sparse requiring $O(kn)$ storage space. This property has later been proven by Frese (2005). To compute a map estimate, a system of n linear equations has to be solved. Thrun et al. use relaxation but update only $O(k)$ landmarks after each measurement (using so-called amortization). This can derogate map quality, since in the numerical literature, relaxation is reputed to need $O(kn^2)$ time for reducing the equation error by a constant factor (Press et al., 1992). Imagine closing a loop of length n and going around that loop a second time. Still the estimate will not have reasonably converged because only $O(k^2n)$ time has been spent. If all landmarks are updated each step *(SEIF w. full update)* performance is asymptotically the same as with relaxation.

Leonard and Feder (2001) avoid the problem of updating an estimate for n landmarks by dividing the map into submaps. Their *Decoupled Stochastic Mapping (DSM)* approach represents each submap in global coordinates by an EKF and updates only the current local submap. The approach is very fast ($O(k^2)$) but, as they note, can introduce overconfidence when passing the robot pose between submaps.

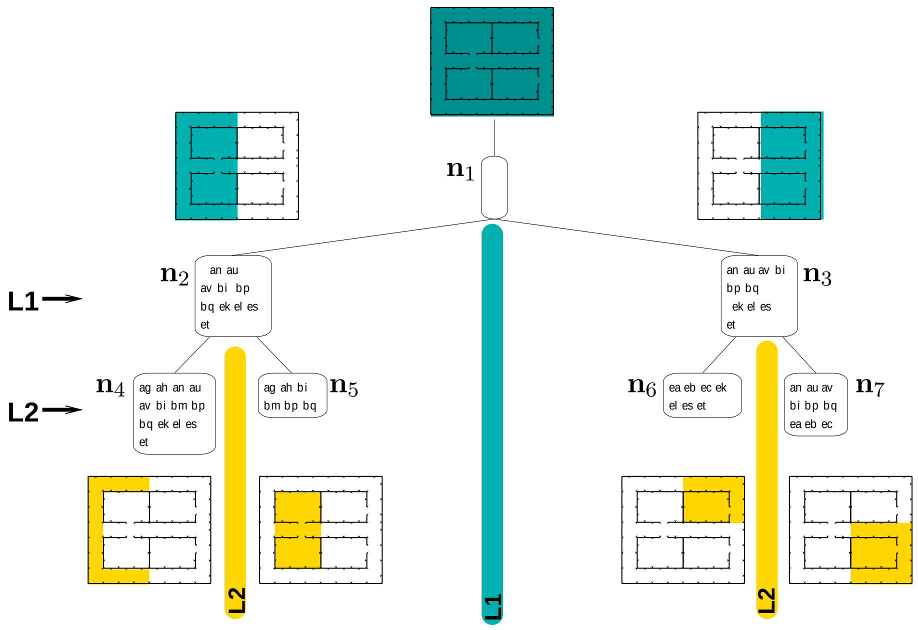
Bosse et al. (2004) in contrast make submaps probabilistically independent in their *Atlas* framework by using a local reference frame. Then, links between adjacent frames are derived by matching local maps. From the resulting graph an estimate is computed. A similar approach is taken by Estrada et al. (2005). Both systems are heterogenous. On the local level a full least square solution is obtained. But on the global level the complex probabilistic relation between submaps is aggregated into a single 3-DOF link between their reference frames.

Paskin (2003) derived the *Thin Junction Tree Filter* (TJTF) from viewing the problem as a Gaussian graphical model. This approach is closely related to treemap although both have been independently derived from different perspectives. They are compared in Section 8.

In the following the treemap algorithm will be introduced. It can be used in the same way as CEKF providing an estimate for k landmarks of a local region but with only $O(k^3 \log n)$ computation time when changing the region instead of $O(kn^{3/2})$ for CEKF. Alternatively the algorithm can also compute a global estimate for all n landmarks. Computation time is then $O(kn)$, but with a constant so small that this can be done for almost "arbitrarily" large maps (12.37ms for n=11300, Intel Xeon 2.7GHz). This is the main contribution from a practical perspective. Note, that while treemap can also provide covariance information, this requires additional computation in contrast to CEKF.



(a)



(b)

Figure 1: **Geometric View.** (a) A building hierarchically decomposed in two levels (L1, L2) and (b) respective tree representation with 7 nodes ($\mathbf{n}_{1\ldots7}$). The region corresponding to a node is shown next to the node. Landmarks inside a region that are visible from outside are listed in the node. Only the marginal distribution of these landmarks is needed if the robot is outside the respective region.

3 Treemap Data Structure

3.1 Motivating Idea

We will first discuss the general idea that motivates the treemap approach. The description differs slightly from the actual implementation but provides a large-picture understanding of the algorithm.

Imagine the robot is in a building (Fig. 1a) that is virtually divided into two parts A and B. Now consider the following question here:

If the robot is in part A, what is the information needed about B?

Some of B's landmarks are observable from A and involved in measurements while the robot is in A. The algorithm must have all previously gathered information about these landmarks explicitly available. This information is more than just the measurements that directly involve those landmarks. Rather all measurements in B can indirectly contribute to the information. So probabilistically speaking, the information needed about B is the marginal distribution of landmarks observable from A conditioned on measurements taken in B[2].

The idea can be applied recursively by dividing the building into a binary[3] tree of regions (Fig. 1). The recursion stops when the size of a region is comparable to the robot's field of view.

The marginal distribution for a region can be computed recursively. The marginals for the two subregions are multiplied and landmarks are marginalized out that are not visible from the outside of that larger region anymore. This core computation is the same as employed by TJTF. The key benefit of this approach is that for integrating a measurement only the region containing the robot and its super-regions need to be updated. All other regions remain unaffected.

Treemap consists of three parts: Core propagation of information in the tree (Sec. 3–4); preprocessing to get information into the tree (Sec. 5–6); and hierarchical tree partitioning to find a good tree (Sec. 7).

3.2 Formal Bayesian View

In a preprocessing step the original measurements are converted into probabilistic constraints $p(X|z_i)$ on the state vector of landmark positions X. At the moment, let us take an abstract probabilistic perspective as to how treemap computes an estimate $\hat{x}$ from these constraints. We will subsequently describe the Gaussian implementation as well as how to get the constraints z_i from the original measurements.

[2]This only holds strictly when there is no odometry (cf. Sec. 5).

[3]Using a binary hierarchy simplifies bookkeeping.

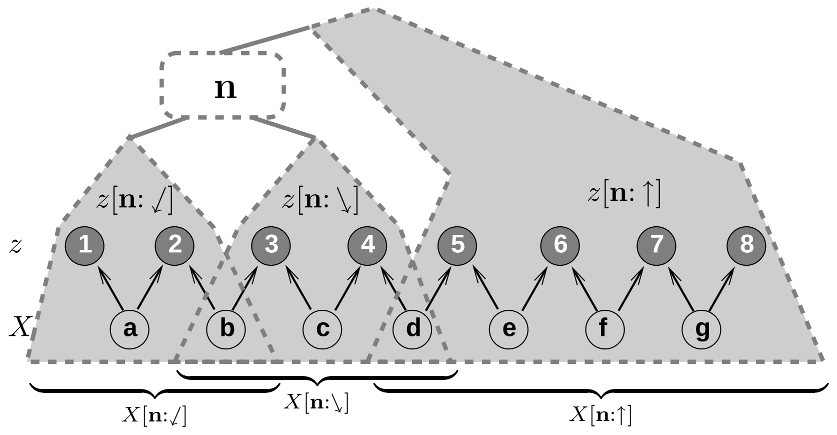

n
$z[\mathbf{n}\colon \swarrow]$
$z[\mathbf{n}\colon \searrow]$
$z[\mathbf{n}\colon \uparrow]$
z
X
1
2
3
4
5
6
7
8
a
b
c
d
e
f
g
$X[\mathbf{n}\colon \swarrow]$
$X[\mathbf{n}\colon \searrow]$
$X[\mathbf{n}\colon \uparrow]$

Figure 2: **Bayesian View.** In this example landmarks $x_{\mathsf{a}\dots\mathsf{g}}$ are observed. They are connected by constraints $z_{2\dots7}$ between consecutive landmarks and two absolute constraints z_1, z_8. The arrows and circles show this probabilistic input as a Bayes net with observed nodes in gray. The dashed outlines illustrate the information from the view of a single node $\mathbf{n}$. It divides the tree into three parts, *left-below* $\swarrow$, *right-below* $\searrow$ and *above* $\uparrow$ (more precisely not below). Hence the constraints z are disjointly divided into $z[\mathbf{n}\colon \swarrow] = z_{1\dots2}$, $z[\mathbf{n}\colon \searrow] = z_{3\dots4}$ and $z[\mathbf{n}\colon \uparrow] = z_{5\dots8}$. The corresponding landmarks $X[\mathbf{n}\colon \swarrow] = X_{\mathsf{a}\dots\mathsf{b}}$, $X[\mathbf{n}\colon \searrow] = X_{\mathsf{b}\dots\mathsf{d}}$ and $X[\mathbf{n}\colon \uparrow] = X_{\mathsf{d}\dots\mathsf{g}}$ however overlap ($X[\mathbf{n}\colon \swarrow\searrow] = X_{\mathsf{b}}$, $X[\mathbf{n}\colon \uparrow\searrow] = x_{\mathsf{d}}$). The key insight is, that $X[\mathbf{n}\colon \downarrow\uparrow] = X[\mathbf{n}\colon \swarrow\uparrow \vee \searrow\uparrow] = X_{\mathsf{d}}$ separates the constraints $z[\mathbf{n}\colon \downarrow]$ and landmarks $X[\mathbf{n}\colon \downarrow\not\uparrow]$ below $\mathbf{n}$ from the constraints $z[\mathbf{n}\colon \uparrow]$ and landmarks $X[\mathbf{n}\colon \not\downarrow\uparrow]$ above $\mathbf{n}$, so both are conditionally independent given $X[\mathbf{n}\colon \downarrow\uparrow]$.

The constraints are assigned to leaves of the tree with the intention to group constraints that share landmarks. With respect to the motivating idea each leaf defines a local region and correspondingly each inner node a super-region. However formally a node $\mathbf{n}$ just represents the set of constraints assigned to leaves below $\mathbf{n}$ without any explicit geometric definition. For a node $\mathbf{n}$, the left and right child and the parent are denoted by $\mathbf{n}_\swarrow$, $\mathbf{n}_\searrow$ and $\mathbf{n}_\uparrow$, respectively. We often have to deal with subsets of observations or landmarks according to where they are represented within the tree relative to the node $\mathbf{n}$ (Fig. 2). Thus, let $z[\mathbf{n}\colon\downarrow]$, $z[\mathbf{n}\colon\swarrow]$, $z[\mathbf{n}\colon\searrow]$, and $z[\mathbf{n}\colon\uparrow]$ denote the constraints assigned to leaves *below* ($\downarrow$), *left-below* ($\swarrow$), *right-below* ($\searrow$), and *above* ($\uparrow$) node $\mathbf{n}$, respectively. The term *above* $\mathbf{n}$ refers to *all regions outside the subtree below* $\boldsymbol{n}$ (!). As a special case, for a leaf $\mathbf{n}$, let $z[\mathbf{n}\colon\swarrow\searrow\uparrow]$ denote all constraints at $\mathbf{n}$. Analogous expressions $X[\mathbf{n}\colon\ldots]$ denote the landmarks involved in the corresponding constraints $z[\mathbf{n}\colon\ldots]$. While constraint sets for different directions $\{\swarrow,\searrow,\uparrow\}$ are disjoint, the corresponding landmark sets may overlap because different constraints may involve the same landmark. These shared features dictate the computations at node $\mathbf{n}$ as the tree is updated. With the assumptions presented in Section 4, their number is small ($O(k)$) and, thus, the overall computation involves many low-dimensional distributions instead of one high-dimensional.

3.3 Update (upwards)

Figure 3 depicts the data flow in treemap that consists of integration ($\odot$) and marginalization (Ⓜ), i.e. multiplying and factorizing probability distributions. We will now derive and prove the computation.

As input, treemap receives a distribution $p^I_{\mathbf{n}}$ defined as $p\big(X[\mathbf{n}\colon\downarrow]\big|z[\mathbf{n}\colon\downarrow]\big)$ at each leaf. It is computed from the probabilistic model for the constraints assigned to $\mathbf{n}$. The output is the integrated information $p_{\mathbf{n}} = p\big(X[\mathbf{n}\colon\downarrow]\big|z\big)$ at each leaf. During the computation, intermediate distributions $p^M_{\mathbf{n}}$ and $p^C_{\mathbf{n}}$ are passed through the tree and stored at the nodes, respectively. In general, $p^I_{\mathbf{n}}$, $p^M_{\mathbf{n}}$, $p^C_{\mathbf{n}}$, and $p_{\mathbf{n}}$ refer to distributions actually computed by the algorithm, whereas all distributions $p\big(X[\ldots]\big|z[\ldots]\big)$ refer to *the* distribution of the landmarks $X[\ldots]$ given the constraints $z[\ldots]$ according to the abstract probabilistic input model shown in Fig. 2. With this notion proving the algorithm means to derive equations $p^{\cdots}_{\mathbf{n}} = p\big(X[\ldots]\big|z[\ldots]\big)$ expressing that the computed result equals the desired distribution from the input model.

Let us first consider the update operation that begins at the leaves and is recursively applied upwards. The update computes the marginal $p^M_{\mathbf{n}}$ and conditional $p^C_{\mathbf{n}}$ either from the input distribution $p^I_{\mathbf{n}}$ or from the

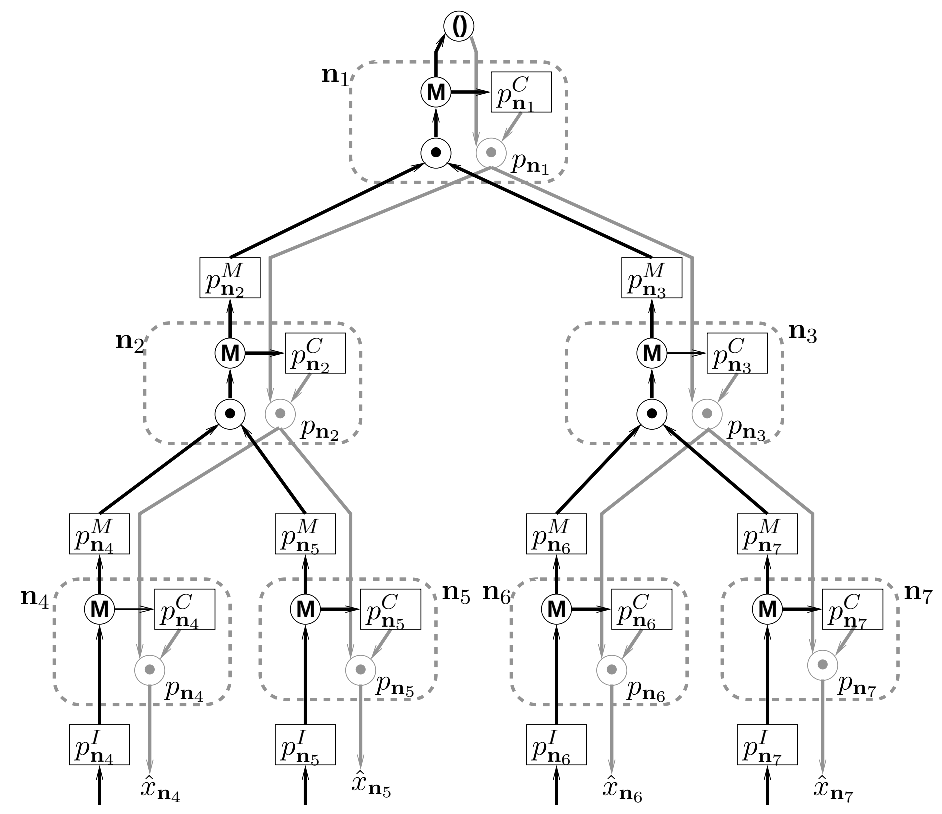


Figure 3: **Data flow view.** The probabilistic computations performed in the tree shown in figure 1b. The leaves store the input constraints $p^I_{\mathbf{n}}$. During updates (black arrows) a node $\mathbf{n}$ integrates ($\odot$) the distributions $p^M_{\mathbf{n}_\swarrow}$ and $p^M_{\mathbf{n}_\searrow}$ passed by its children. Then the result is factorized (Ⓜ) as the product of a marginal p^M_n passed to the parent and a conditional p^C_n stored at the node. To compute an estimate (gray arrows) each node $\mathbf{n}$ receives a distribution $p_{\mathbf{n}_\uparrow}$ from its parent, integrates ($\odot$) it with the conditional $p^C_{\mathbf{n}}$, and passes the result $p_{\mathbf{n}}$ down to its children. In the end estimates $\hat{x}_{\mathbf{n}}$ for all landmarks are available at the leaves.

children's marginals $p^M_{\mathbf{n}_\swarrow}$ and $p^M_{\mathbf{n}_\searrow}$.

$$p^M_{\mathbf{n}} = p\big(X[\mathbf{n}\colon\downarrow\uparrow]\big|z[\mathbf{n}\colon\downarrow]\big) \tag{1}$$
$$p^C_{\mathbf{n}} = p\big(X[\mathbf{n}\colon\swarrow\searrow\not\uparrow]\big|X[\mathbf{n}\colon\downarrow\uparrow], z\big). \tag{2}$$

The marginal distribution (1) describes the posterior for the landmarks both above and below $X[\mathbf{n}\colon\downarrow\uparrow]$ conditioned upon the constraints $z[\mathbf{n}\colon\downarrow]$ below $\mathbf{n}$. These landmarks are by definition also involved in constraints which are not yet integrated into $p^M_{\mathbf{n}}$. So $p^M_{\mathbf{n}}$ is passed to the parent for further processing. In contrast, $p^C_{\mathbf{n}}$ contains those landmarks $X[\mathbf{n}\colon\swarrow\searrow\not\uparrow]$ for which $\mathbf{n}$ is the least common ancestor of all constraints involving them. These constraints have already been integrated, so $p^C_{\mathbf{n}}$ needs no more processing and can be finally stored at $\mathbf{n}$. Overall, a landmark is passed upwards in $p^M_{\mathbf{n}}$ up to the node where all constraints involving that landmark have been integrated and then it is stored in $p^C_{\mathbf{n}}$.

We now derive the recursive computation of $p^M_{\mathbf{n}}$ and $p^C_{\mathbf{n}}$ proving (1) and (2) by induction. An inner node $\mathbf{n}$ multiplies ($\odot$) the marginals $p^M_{\mathbf{n}_\swarrow}$ and $p^M_{\mathbf{n}_\searrow}$ passed by its children. Assuming (1) for $\mathbf{n}_\searrow$ we get

$$p^M_{\mathbf{n}_\searrow} = p\big(X[\mathbf{n}_\searrow\colon\downarrow\uparrow]\big|z[\mathbf{n}_\searrow\colon\downarrow]\big). \tag{3}$$

Being above $\mathbf{n}_\searrow$ means either to be above $\mathbf{n}$ or to be left-below $\mathbf{n}$.

$$= p\big(X[\mathbf{n}\colon\searrow\uparrow \vee \swarrow\searrow\not\uparrow]\big|z[\mathbf{n}\colon\searrow]\big) \tag{4}$$

To multiply $p^M_{\mathbf{n}_\swarrow}$ and $p^M_{\mathbf{n}_\searrow}$ we must formally interpret both as a distribution for the union of landmarks. This is possible, since $X[\mathbf{n}\colon\swarrow\not\searrow\uparrow]$ are by definition not involved in $z[\mathbf{n}\colon\searrow]$ at all.

$$= p\big(X[\mathbf{n}\colon\searrow\uparrow \vee \swarrow\searrow\not\uparrow \vee \swarrow\not\searrow\uparrow]\big|z[\mathbf{n}\colon\searrow]\big) \tag{5}$$
$$= p\big(X[\mathbf{n}\colon\downarrow\uparrow \vee \swarrow\searrow\not\uparrow]\big|z[\mathbf{n}\colon\searrow]\big) \tag{6}$$

This argument may appear rather technical at first sight but it ensures that in defining $p^M_{\mathbf{n}}$ by (1) we actually found that part of $p\big(X\big|z[\mathbf{n}\colon\downarrow]\big)$ that cannot be fully processed below $\mathbf{n}$ and has to be passed to the parent. Certainly a symmetric result holds for $p^M_{\mathbf{n}_\swarrow}$, so both can be multiplied ($\odot$) gathering all information below $\mathbf{n}$.

$$p^M_{\mathbf{n}_\swarrow} \cdot p^M_{\mathbf{n}_\searrow} = p\big(X[\mathbf{n}\colon\downarrow\uparrow \vee \swarrow\searrow\not\uparrow]\big|z[\mathbf{n}\swarrow]\big) \cdot p\big(X[\mathbf{n}\colon\downarrow\uparrow \vee \swarrow\searrow\not\uparrow]\big|z[\mathbf{n}\colon\searrow]\big) \tag{7}$$
$$= p\big(X[\mathbf{n}\colon\downarrow\uparrow \vee \swarrow\searrow\not\uparrow]\big|z[\mathbf{n}\downarrow]\big) \tag{8}$$

Let $Y = X[\mathbf{n}\colon\downarrow\uparrow \vee \swarrow\searrow\not\uparrow]$ be the vector of landmarks involved in $p^M_{\mathbf{n}_\swarrow}$ or $p^M_{\mathbf{n}_\searrow}$. Treemap divides $Y = \binom{U}{V}$ into those landmarks $V = X[\mathbf{n}\colon\downarrow\uparrow]$ involved in constraints above $\mathbf{n}$ and those $U = X[\mathbf{n}\colon\swarrow\searrow\not\uparrow]$ for which $\mathbf{n}$ is the least common ancestor of all constraints involving them. Landmarks U are marginalized out (Ⓜ) factorizing the distribution as the product

$$p^M_{\mathbf{n}\swarrow}\left(\begin{smallmatrix}u\\v\end{smallmatrix}\right)\;\cdot\;p^M_{\mathbf{n}\searrow}\left(\begin{smallmatrix}u\\v\end{smallmatrix}\right)=p^M_{\mathbf{n}}(v)\;\cdot\;p^C_{\mathbf{n}}(u|v) \tag{9}$$

of the marginal $p^M_{\mathbf{n}}(V)$ passed to the parent and the conditional $p^C_{\mathbf{n}}(U|V)$ stored at $\mathbf{n}$. We can verify, that $p^M_{\mathbf{n}}$ satisfies (1) and $p^C_{\mathbf{n}}$ satisfies (2) by marginalizing respectively conditioning both sides of (8).

$$\begin{aligned} p^M_{\mathbf{n}} &= p\big(X[\mathbf{n}\colon\downarrow\uparrow]\big|z[\mathbf{n}\downarrow]\big) && (10)\\ p^C_{\mathbf{n}} &= p\big(X[\mathbf{n}\colon\swarrow\searrow\not\uparrow]\big|X[\mathbf{n}\colon\downarrow\uparrow],z[\mathbf{n}\downarrow]\big) && (11)\\ &= p\big(X[\mathbf{n}\colon\swarrow\searrow\not\uparrow]\big|X[\mathbf{n}\colon\downarrow\uparrow],z\big) && (12)\end{aligned}$$

The second step (12) is the formal key point of the overall approach. In Bayes net terminology (Fig. 2) $X[\mathbf{n}\colon\downarrow\uparrow]$ separates the constraints $z[\mathbf{n}\colon\downarrow]$ and landmarks $X[\mathbf{n}\colon\downarrow\not\uparrow]$ below $\mathbf{n}$ from the constraints $z[\mathbf{n}\colon\uparrow]$ and landmarks $X[\mathbf{n}\colon\not\downarrow\uparrow]$ above $\mathbf{n}$. So $X[\mathbf{n}\colon\swarrow\searrow\not\uparrow]$, which is part of $X[\mathbf{n}\colon\downarrow\not\uparrow]$, is conditionally independent from the remaining constraints $z[\mathbf{n}\colon\uparrow]$.

Now we have established that if (1) holds for a given nodes children, then (1) and (2) hold for $p^M_{\mathbf{n}}$ and $p^C_{\mathbf{n}}$ computed by the node. We still have to verify these equations for leaves. Then by induction they hold for all nodes. At a leaf $\mathbf{n}$ all original constraints that were assigned to that leaf are multiplied and stored as input distribution

$$\begin{aligned} p^I_{\mathbf{n}} &= \prod_{\text{i assigned to }\mathbf{n}} p(X|z_i)=p\big(X[\mathbf{n}\colon\downarrow]\big|z[\mathbf{n}\colon\downarrow]\big) && (13)\\ &= p\big(X[\mathbf{n}\colon\downarrow\uparrow\vee\swarrow\searrow\not\uparrow]\big|z[\mathbf{n}\colon\downarrow]\big). && (14)\end{aligned}$$

For a leaf $\mathbf{n}$ we defined $X[\mathbf{n}\colon\swarrow\searrow]$ as those landmarks involved in constraints assigned to that leaf. So for a leaf, $p^I_{\mathbf{n}}$ satisfies the same condition (8) as $p^M_{\mathbf{n}\swarrow}\cdot p^M_{\mathbf{n}\searrow}$ for an inner node. Hence, after marginalization (1) and (2) hold for leaves with the same arguments as for inner nodes.

As a final remark, $p^M_{\mathbf{root}}=()$ is empty by (1), because there is nothing above **root**. So it is the end of the upward update-arrows and the start of the downward state-recovery arrows (Fig. 3).

3.4 State Recovery (Downwards)

Now let us consider how to compute a state estimate from the $p^C_{\mathbf{n}}$ (gray arrows pointing downwards). Here the goal is that every node $\mathbf{n}$ passes

$$p_{\mathbf{n}}=p\big(X[\mathbf{n}\colon\downarrow\uparrow\vee\swarrow\searrow\not\uparrow]\big|z\big) \tag{15}$$

down. Hence a leaf computes the marginal of landmarks involved since $X[\mathbf{n}\colon\downarrow\uparrow\vee\swarrow\searrow\not\uparrow]$ equals $X[\mathbf{n}\colon\downarrow]$. The final estimate $\hat{x}_{\mathbf{n}}$ is computed as

$$\hat{x}_{\mathbf{n}}=E(p_{\mathbf{n}})=E\big(X[\mathbf{n}\colon\downarrow]\big|z\big). \tag{16}$$

Since every update changes $p_{\mathbf{n}}$, it is computed on the fly and not stored.

Now we derive (15) by induction. Let us assume a node $\mathbf{n}$ receives

$$p_{\mathbf{n}_\uparrow} = p\big(X[\mathbf{n}_\uparrow\colon {\downarrow}{\uparrow} \vee {\swarrow}{\searrow}{\dashuparrow}]\big|z\big) \tag{17}$$

from its parent. A landmark below $\mathbf{n}_\uparrow$ is either below $\mathbf{n}$ or below the sibling of $\mathbf{n}$. The latter ones are marginalized out resulting in

$$p\big(X[\mathbf{n}\colon {\downarrow}{\uparrow}]\big|z\big). \tag{18}$$

This step is not shown in figure 3 because it is implicitly done in the actual Gaussian implementation (cf. Sec. 3.5). The result is multiplied ($\odot$) with the conditional $p^C_{\mathbf{n}}$ stored at $\mathbf{n}$ and passed downwards as $p_{\mathbf{n}}$.

$$p_{\mathbf{n}} = p\big(X[\mathbf{n}\colon {\downarrow}{\uparrow}]\big|z\big) \;\cdot\; p^C_{\mathbf{n}} \tag{19}$$
$$= p\big(X[\mathbf{n}\colon {\downarrow}{\uparrow}]\big|z\big) \;\cdot\; p\big(X[\mathbf{n}\colon {\swarrow}{\searrow}{\dashuparrow}]\big|X[\mathbf{n}\colon {\downarrow}{\uparrow}], z\big) \tag{20}$$
$$= p\big(X[\mathbf{n}\colon {\downarrow}{\uparrow} \vee {\swarrow}{\searrow}{\dashuparrow}]\big|z\big) \tag{21}$$

We have shown, that if node $\mathbf{n}$ receives $p_{\mathbf{n}\uparrow}$ with (15) it passes a distribution $p_{\mathbf{n}}$ to its children holding (15) too. As the induction start $X[\mathbf{root}\colon {\downarrow}{\uparrow}]$ in (18) is empty, so by induction (15) holds for all $p_{\mathbf{n}}$.

3.5 Gaussian Implementation

Treemap uses Gaussians for all probability distributions. Thereby the probabilistic computations reduce to matrix operations and the algorithm becomes an efficient linear equation solver for a specific class of equations. The performance is much improved by using different representations for updates ($p^I_{\mathbf{n}}$, $p^M_{\mathbf{n}}$), for state recovery ($p_{\mathbf{n}}$), and for the conditional $p^C_{\mathbf{n}}$ linking both. We will now derive formulas for the three operations $\odot$ (update), $\textcircled{M}$, and $\odot$ (state recovery) involved.

Distributions $p^I_{\mathbf{n}}$ and $p^M_{\mathbf{n}}$ are stored in information form as

$$-\log\, p^M_{\mathbf{n}}(y) = y^T A_{\mathbf{n}} y + y^T b_{\mathbf{n}} + \text{const}\,. \tag{22}$$

Update (Upwards)

If treemap is used directly with landmark–landmark constraints, $p^I_{\mathbf{n}}$ is computed as usual by linearizing the constraints, expressing the approximated χ^2 error by an information matrix and vector and adding these for all constraints assigned to the leaf $\mathbf{n}$ (Thrun et al., 2005, §11.4.3). In Section 5 we will discuss how to derive $p^I_{\mathbf{n}}$ in a preprocessing step from robot–landmark and robot–robot constraints.

To perform the multiplication $\odot$ at node $\mathbf{n}$, first $(A_{\mathbf{n}_\swarrow}, b_{\mathbf{n}_\swarrow})$ as well as $(A_{\mathbf{n}_\searrow}, b_{\mathbf{n}_\searrow})$ are permuted and extended with 0-rows/columns such that the same row/column corresponds to the same landmark in both. Additionally landmarks of $X[\mathbf{n}\colon \swarrow\searrow\not\uparrow]$ are permuted to the upper rows / left columns and landmarks of $X[\mathbf{n}\colon \downarrow\uparrow]$ to the lower rows / right columns. This will help later for marginalization. Then they are added.

$$-\log\Big(p^M_{\mathbf{n}_\swarrow}(y)\,p^M_{\mathbf{n}_\searrow}(y)\Big) = -\log p^M_{\mathbf{n}_\swarrow}(y) - \log p^M_{\mathbf{n}_\searrow}(y) \tag{23}$$

$$= y^T(A_{\mathbf{n}_\swarrow} + A_{\mathbf{n}_\searrow})y + y^T(b_{\mathbf{n}_\swarrow} + b_{\mathbf{n}_\searrow}) + \text{const} \tag{24}$$

To perform the marginalization Ⓜ, $A_{\mathbf{n}_\swarrow} + A_{\mathbf{n}_\searrow}$ is viewed as a 2×2 block matrix and $b_{\mathbf{n}_\swarrow} + b_{\mathbf{n}_\searrow}$ as a 2 block vector

$$= \left(\begin{smallmatrix} u \\ v \end{smallmatrix}\right)^T \left(\begin{smallmatrix} P & R^T \\ R & S \end{smallmatrix}\right)\left(\begin{smallmatrix} u \\ v \end{smallmatrix}\right) + \left(\begin{smallmatrix} u \\ v \end{smallmatrix}\right)^T\left(\begin{smallmatrix} c \\ d \end{smallmatrix}\right) + \text{const}\,. \tag{25}$$

The first block row / column corresponds to landmarks $U = X[\mathbf{n}\colon \swarrow\searrow\not\uparrow]$ to be marginalized out and stored in $p^C_{\mathbf{n}}$. The second block row / column corresponds to landmarks $V = X[\mathbf{n}\colon \downarrow\uparrow]$ to be passed in $p^M_{\mathbf{n}}$. By a straight-forward but rather lengthy calculation it follows that

$$\begin{aligned} = \quad & v^T\big(S - RP^{-1}R^T\big)v + v^T\big(-RP^{-1}c + d\big) + \text{const} \\ + & \big(Hv + h - u\big)^T P\big(Hv + h - u\big), \end{aligned} \tag{26}$$

$$\text{with } H = -P^{-1}R^T \text{ and } h = -P^{-1}c/2. \tag{27}$$

The first line of (26) defines a Gaussian for v in information form not involving u at all. The second line defines a Gaussian for u with covariance P^{-1} and mean $Hv + h$. The first does not contribute to the conditional $p(U|V)$ and the second not to the marginal $p(v)$. Thus

$$-\log p^M_{\mathbf{n}}(v) = v^T\big(S - RP^{-1}R^T\big)v + v^T\big(-RP^{-1}c + d\big) + \text{const} \tag{28}$$

$$-\log p^C_{\mathbf{n}}(u|v) = \big(Hv + h - u\big)^T P\big(Hv + h - u\big) \tag{29}$$

holds. Algorithmically treemap computes the information matrix $A^M_{\mathbf{n}}$ and vector $b^M_{\mathbf{n}}$ of $p^M_{\mathbf{n}}$ by

$$A^M_{\mathbf{n}} = S - RP^{-1}R^T, \quad b^M_{\mathbf{n}} = -RP^{-1}c + d \tag{30}$$

and passes it to the parent node. This is the well known marginalization formula for Gaussians (Thrun et al., 2005, Tab. 11.6) which is also known as Schur-complement (Horn and Johnson, 1990). Equation (29) is remarkable. It represents $p(u|v)$ *in terms of* v as a single Gaussian in u with mean $Hv + h$. For general distributions no such simple relation will hold. Treemap stores $p^C_{\mathbf{n}}$ as (P^{-1}, H, h).

State Recovery (Downwards)

With this representation for $p^C_{\mathbf{n}}$ state recovery $\odot$ can be implemented very efficiently in covariance form. The mean $\hat{v} = E(v|z)$ is passed by the parent node and the mean of u is correspondingly

$$\hat{y} = \left(\begin{smallmatrix}\hat{u}\\ \hat{v}\end{smallmatrix}\right), \quad \hat{u} = E(u|z) \overset{Gaussian}{=} E(u|v = E(v|z), z) = H\hat{v} + h. \tag{31}$$

Note, that $E(u|z) = E(u|v = E(v|z))$ only holds for Gaussians. In general the full distribution $p(v|z)$ is necessary to compute $E(u|z)$ from $E(u|v, z)$. So for recovering the global state estimate $\hat{x} = E(X|z)$, it suffices to propagate the mean downwards – it is not necessary to propagate covariances at all. In this case, state recovery requires only a single matrix-vector product in each node and is extremely efficient.

If the covariance is desired, it can be propagated the same way. If $\operatorname{cov}(v) = C_v$ is passed by the parent node, $\operatorname{cov}\left(\begin{smallmatrix}u\\ v\end{smallmatrix}\right)$ can be computed as

$$C = \operatorname{cov}\hat{y} = \operatorname{cov}\left(\begin{smallmatrix}u\\ v\end{smallmatrix}\right) = \operatorname{cov}\left(\left(\begin{smallmatrix}Hv+h\\ v\end{smallmatrix}\right) - \left(\begin{smallmatrix}Hv+h-u\\ 0\end{smallmatrix}\right)\right) \tag{32}$$

$$= \begin{pmatrix} HC_vH^T & HC_v \\ C_vH^T & C_v \end{pmatrix} + \left(\begin{smallmatrix}P^{-1} & 0\\ 0 & 0\end{smallmatrix}\right) = \begin{pmatrix} HC_vH^T+P^{-1} & HC \\ C_vH^T & C_v \end{pmatrix}. \tag{33}$$

The last equation follows from $p^C_{\mathbf{n}}$ defining a P^{-1} covariance Gaussian on $Hv + h - u$ by (29). This Gaussian is independent from the one passed by the parent node, since the marginalization (Ⓜ) factorizes into two independent distributions $p^M_{\mathbf{n}}$ and $p^C_{\mathbf{n}}$. The result of recursive propagation is a covariance matrix for each leaf yielding correlations between all landmarks involved in measurements at the same leaf.

3.6 Performance and Discussion

As evident from the description in this section, there is *no approximation* involved in the update and state-recovery operations computing $\hat{x}$ from the different $p^I_{\mathbf{n}}$. The estimate $\hat{x}$ computed by this core part of treemap is the *same* as the one provided by linearized least square or EKF when using the same linearization point. Approximation errors are introduced by linearization in computing $p^I_{\mathbf{n}}$ from the original non-linear landmark–landmark constraints. When, as usual, the input are robot–landmark and robot–robot constraints, a further preprocessing step is necessary (cf. Sec. 5). This step marginalizes out old robot poses and in doing so creates further landmark constraints. To avoid getting too many constraints, a so called sparsification is necessary, which is a second source of error. No further approximations are involved.

Local and global levels are treated conceptually the same way by least square estimation on landmarks. This is different from Atlas and the algorithm by Estrada et al. (2005), that use a graph over relations of reference frames on the global level. So as with CEKF and TJTF the division into submaps is mostly transparent for the user.

There are three key ideas that make this computation fast.

- **Many small matrices instead of one large matrix.** This is the motivation. For the matrices actually to be small the building must have a hierarchical partitioning with limited overlap (cf. Sec. 4) and the partitioning subalgorithm must actually find one (cf. Sec. 7.3).
- **Only a single path from leaf to root needs to be updated** after a new constraint is added to that leaf. Since $p_{\mathbf{n}}^{M}$ and $p_{\mathbf{n}}^{C}$ depend only on $z[\mathbf{n}\colon\downharpoonleft]$ all other nodes still remain valid[4]. So if the tree is balanced, only $O(\log n)$ nodes are updated.
- **State-recovery is fast,** because it needs only a single matrix-vector product per node (31) to propagate the mean. Alternatively two matrix products are needed to propagate the covariance (33). This makes computing a global estimate extremely fast, because then recursive propagation is the dominant operation $O(n)$.

Appendix B shows a worked out example for propagation of distributions in the tree corresponding to the example in figure 2.

4 Assumptions on Topologically Suitable Buildings

The time needed for computation at a node $\mathbf{n}$ depends on the size of the matrices involved, which is determined by the number of landmarks in $X[\mathbf{n}\colon\downharpoonleft\upharpoonright \vee \swarrow\searrow\twoheaduparrow] = X[\mathbf{n}_{\swarrow}\colon\downharpoonleft\upharpoonright \vee \mathbf{n}_{\searrow}\colon\downharpoonleft\upharpoonright]$. So for each node only few landmarks should at the same time be involved in constraints below and in constraints above $\mathbf{n}$. Or intuitively speaking, the region represented by a node should only have a small border with the rest of the building.

As the experiments in Section 10 and the following considerations confirm, typical buildings allow such a hierarchical partitioning as a tree because they are hierarchical themselves, consisting of floors, corridors and rooms. Different floors are only connected through a few staircases, different corridors through a few crossings and different rooms most often only through a single door and the adjacent parts of the corridor. Thus, on the different levels of the hierarchy natural regions are: rooms, part of a corridor including adjacent rooms, one or several adjacent corridors and one or several consecutive floors (Fig. 4).

Let us formally define a *"suitable hierarchical partitioning"* and thus a *"topologically suitable building"* having such a partitioning.

[4]An exception is discussed in Section 7 but does not affect the $O(\log n)$ claim.

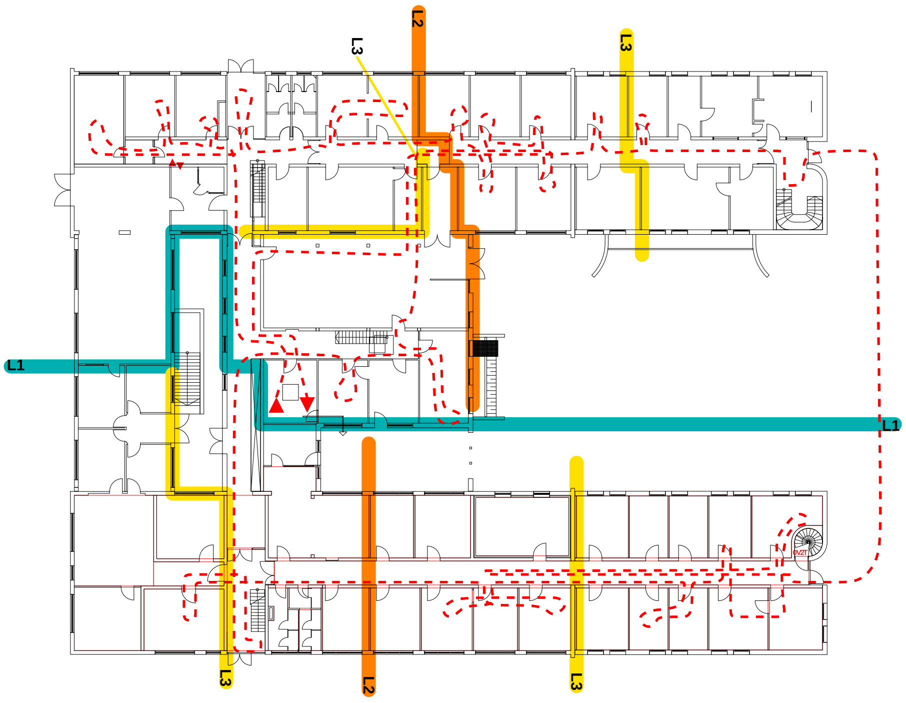


Figure 4: DLR Institute of Robotics and Mechatronics – A typical topologically suitable building with the first three levels (L1, L2, L3) of a suitable hierarchical partitioning. It has been mapped in the experiments (Sec. 10), with the dashed line sketching the robots trajectory. The start and finish are indicated by small triangles.

Definition 1 (Suitable Hierarchical Partitioning) *Let the measurements z be assigned to leaves of a tree. Let k be the maximum number of landmarks involved in measurements from a single robot pose. Then the tree is a suitable hierarchical partitioning, if*

1. *For each node $\boldsymbol{n}$ the number of landmarks in $X[\boldsymbol{n}\colon\downarrow\uparrow]$ is $O(k)$.*
2. *For each leaf $\boldsymbol{n}$ the number of leaves $\boldsymbol{n}'$ for which $X[\boldsymbol{n}\colon\downarrow]$ and $X[\boldsymbol{n}'\colon\downarrow]$ share a landmark is $O(1)$.*

Definition 2 (Topologically suitable building) *A topologically suitable building is a building where a suitable hierarchical partitioning exists regardless how the robot moves.*

The parameter k is small, since the robot can only observe a few landmarks simultaneously because its field of view is limited both by walls and sensor range. In particular, k does not increase when the map gets larger ($n \to \infty$). Although by this argument $k = O(1)$, the asymptotical expressions in this article explicitly show the influence of k. All expressions hold strictly if two heuristic assumptions are valid.

- The encountered building is topologically suitable, i.e. a suitable partitioning exists.
- The hierarchical tree partitioning (HTP) subalgorithm (Sec. 7.3) succeeds in finding such a suitable partitioning.

If definition 1 is restricted only to leaves, it is mostly equivalent to general sparsity in the information form as exploited by SEIF and other algorithms. In general it is stronger since it demands $O(k)$ connections even between large regions. Still it is compatible with loops and nested loops as evident from the experiments (Fig. 10 contains 200 medium loops nested in 10 large loops) but it does preclude grid-like structures. So large open halls as well as most outdoor environments are not topologically suitable. If for instance an $l \times l$ square with n landmarks is divided into 2 halves, the border involves $O(l) = O(\sqrt{n})$ landmarks. So there will be an $\sqrt{n}\times\sqrt{n}$ matrix at the root node increasing update time to $O(n^{3/2})$. Estimation quality will not be affected. However, if the goal is to explore rather than to "mow the lawn" the robot will operate on a network of paths. Treemap can still be reasonable efficient then.

4.1 Computational Efficiency

By definition 1 there are $O(\frac{n}{k})$ nodes in the tree (part 2) and each stores matrices of dimension $O(k\times k)$ (part 1). Thus, the storage requirement of the treemap is $O(k^2 \cdot \frac{n}{k}) = O(nk)$ meeting requirement (R2). Updating one node takes $O(k^3)$ time for (30) and (27). State-recovery by (31) needs $O(k^2)$ time (mean only) and by (33) $O(k^3)$ time (covariance).

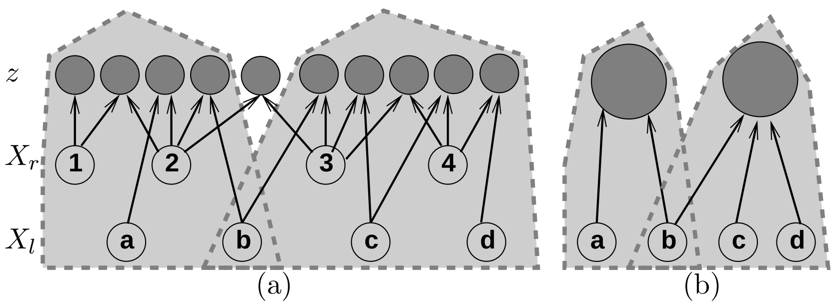


Figure 5: **Bayesian View.** (a) A part of the example shown in (Fig. 2) with landmarks X_l ($\mathsf{a} \dots \mathsf{d}$) and robot poses X_r ($1 \dots 4$). The odometry constraint between pose 2 and 3 (shown outside both regions) is ignored. In this manner the robot poses are involved only in their respective region and are marginalized out. (b) The result for each region is a single Gaussian (big circle) on all landmarks in that region.

So after integrating new constraints into $p^I_{\mathbf{n}}$ at some leaf $\mathbf{n}$ $O(k^3 \log n)$ time is needed for updating. An estimate for the landmarks involved at some leaf $\mathbf{n}'$ can be provided in the same computation time. This way treemap can be used the same way as CEKF maintaining only local estimates but replacing CEKF's $O(kn^{3/2})$ global update with treemap's $O(k^3 \log n)$ update. As long as $\mathbf{n} = \mathbf{n}'$ we skip the treemap update and proceed as CEKF using the EKF equations in $O(k^2)$ time.

In order to compute an estimate for all landmarks, (31) must be applied recursively taking $O(k^2 \frac{n}{k}) = O(kn)$ (mean only). It will turn out in the experiments in Section 9 that the constant factor involved is extremely small. So while the possibility to perform updates in sublinear time is most appealing from a theoretical perspective, in practice treemap can compute a global estimate even for extremely large maps.

Overall, definition 1 is both the strength and weakness of treemap. The insight that buildings have such a loosely connected topology distinguishes indoor SLAM from many other estimation problems and enables treemap's impressive efficiency. On the other hand it precludes dense planar mapping mainly ruling out outdoor environments.

5 EKF based Preprocessing Stage

The part of treemap discussed so far is very general. It can estimate random variables with any meaning given some Gaussian constraints with suitable topology. However it cannot marginalize out random variables that are not needed any more, i.e. old robot poses.

In this section we will derive an EKF based preprocessing stage. It receives landmark observations and odometry measurements and converts these into information on the current robot pose and information on local landmarks marginalizing out old poses. The information on landmarks is passed into the treemap as a Gaussian constraint.

In each moment there is one local region, i.e. one leaf **c** that is active corresponding to where the robot currently is. New landmark information is multiplied into $p^I_{\mathbf{c}}$ and the EKF maintains an estimate for all landmarks involved there. In this sense, the framework is similar to that of the CEKF, Atlas, and Feder's submap algorithm. Unlike Atlas and Feder's algorithm treemap employs a full least square estimator on top of this local estimate, namely the tree discussed so far. So as with CEKF, the EKF's local estimate includes information from all measurements not just from measurements in the current region.

We will first derive a simple solution where no robot pose information is passed across regions. It follows the relocation idea by Walter et al. (2005) as well as Frese and Hirzinger (2001) and sacrifices odometry information to preserve sparsity when marginalizing out old robot poses. The next section discusses a more sophisticated sparsification scheme that passes the robot pose between regions at the expense of sacrificing some landmark information. While the experiments used that scheme, relocation is much easier and works very convincingly as we recently observed (Frese and Schröder, 2006).

5.1 Bayesian View

Figure 5 shows an example as a Bayes net with landmarks and robot poses for two regions. The odometry measurement that connects poses in both regions is ignored. Then all poses are only involved inside one region and can be marginalized out. This means, that whenever the robot enters a new region, its position is only defined by the measurements made there. The regions are however connected by overlapping landmarks. Note, though, that odometry can still be used for data association, so this does not mean the robot is actually "kidnapped". Odometry is only ignored in the sense that no constraint is integrated.

5.2 Data Flow View

This process can be conveniently implemented as a preprocessing EKF (Fig. 6). When entering a region **c**, treemap computes the marginal $p_{\mathbf{c}}$ (mean and covariance) of landmarks $X[\mathbf{c}\colon\downarrow]$ involved there.

$$p_{\text{EKF}} = p_{\mathbf{c}} = p\big(X[\mathbf{c}\colon\downarrow]\big|z_-\big) \qquad (34)$$

We write z_- to indicate that the distribution is conditioned on the measurements made before entering **c**. The EKF is initialized with this

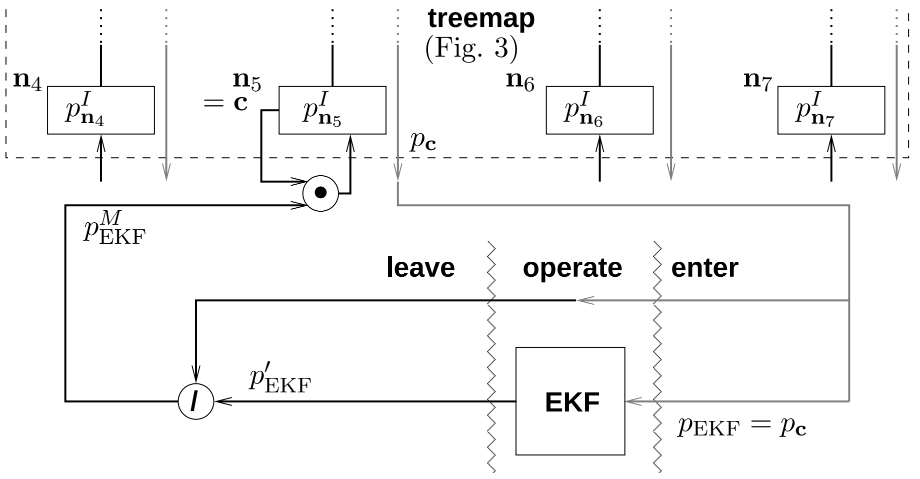


Figure 6: **Data flow view.** The figure's top shows the lower part of the treemap (i.e. leaves) as depicted in figure 3. It illustrates how information is passed from the treemap into the preprocessing EKF and vice versa. When entering region $\mathbf{c}$ the EKF is initialized with the marginal $p_\mathbf{c}$ from the treemap. When leaving $\mathbf{c}$ again, the new information p^M_{EKF} on landmarks is multiplied into $p^I_\mathbf{c}$ integrating it into the treemap. Each time the robot pose is discarded and redefined by the next measurement.

distribution and an ∞-covariance prior for the robot pose. While the robot stays in the region, the EKF maintains

$$p'_{\text{EKF}} = p\big(X_r, X[\mathbf{c}\colon\downarrow]\big|z\big). \tag{35}$$

After leaving $\mathbf{c}$, information must be passed from the EKF to the treemap. For that purpose we take the EKF's marginal on landmarks

$$\begin{aligned} p\big(X[\mathbf{c}\colon\downarrow]\big|z\big) &= \int_{X_r} p\big(X_r, X[\mathbf{c}\colon\downarrow]\big|z\big) && (36)\\ &= \int_{X_r} p\big(X[\mathbf{c}\colon\downarrow]\big|z_-\big)\cdot p\big(X_r, X[\mathbf{c}\colon\downarrow]\big|z_+\big) && (37)\\ &= p\big(X[\mathbf{c}\colon\downarrow]\big|z_-\big)\int_{X_r} p\big(X_r, X[\mathbf{c}\colon\downarrow]\big|z_+\big) && (38)\\ &= p\big(X[\mathbf{c}\colon\downarrow]\big|z_-\big)\cdot p\big(X[\mathbf{c}\colon\downarrow]\big|z_+\big). && (39)\end{aligned}$$

This equation relies on the odometry constraint being removed, because otherwise X_r would be involved in both factors and neither one could be moved out of the integral. The marginal is then divided ($\oslash$) by $p_\mathbf{c}$ the information already stored in the treemap. The result is

$$p^M_{\text{EKF}} = \frac{\int_{x_r} p'_{\text{EKF}}}{p_\mathbf{c}} = \frac{p\big(X[\mathbf{c}\colon\downarrow]\big|z_+\big)\cdot p\big(X[\mathbf{c}\colon\downarrow]\big|z_-\big)}{p\big(X[\mathbf{c}\colon\downarrow]\big|z_-\big)} = p\big(X[\mathbf{c}\colon\downarrow]\big|z_+\big) \tag{40}$$

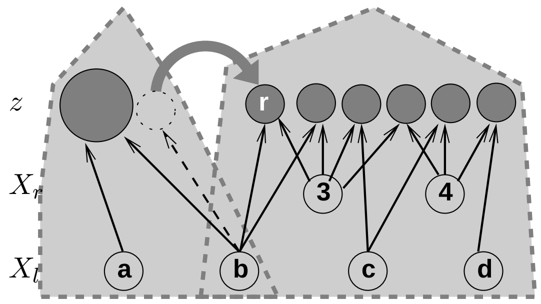


Figure 7: **Bayesian View.** The same example as in figure 5 but with passing robot pose information. In the old region (left) robot poses have already been marginalized out. The resulting conditional (r) is passed into the new region (big gray arrow) and used there to define the robot pose X_3.

the information obtained by new measurements. It is independent from z_- and can be multiplied into $p^I_{\mathbf{c}}$ passing that information to the treemap.

6 Passing the Robot Pose

Discarding all information on the robot pose when entering a new region is a useful but not completely satisfying approach. So in the following we will discuss a method for passing the robot pose information on to the new region. The method is statistically consistent but sacrifices some information on landmarks. This is not surprising because marginalizing out old robot poses is well known to lead to a dense information matrix (Thrun et al., 2004). Since the tree defines a sparse matrix some approximation is needed. Thrun et al. observed and Frese (2005) later proved that most entries of the information matrix are small. This justifies the approximation of it by a sparse matrix.

6.1 Bayesian View

Figure 7 provides a general depiction of the distributions involved in passing the robot pose. In contrast to the simpler approach described in the previous section, we want to have some prior (shown as ⓡ) on the robot pose when entering the new region. Since we marginalize out the robot pose in the old region $\mathbf{c}_{\text{prev}}$ by

$$p\big(X_r, X[\mathbf{c}_{\text{prev}}\colon\downarrow]\big|z\big) = p\big(X[\mathbf{c}_{\text{prev}}\colon\downarrow]\big|z\big) \cdot p\big(X_r\big|X[\mathbf{c}_{\text{prev}}\colon\downarrow], z\big), \qquad (41)$$

an immediate idea would be to pass the corresponding conditional (dotted circle). This distribution is independent from the marginal and thus

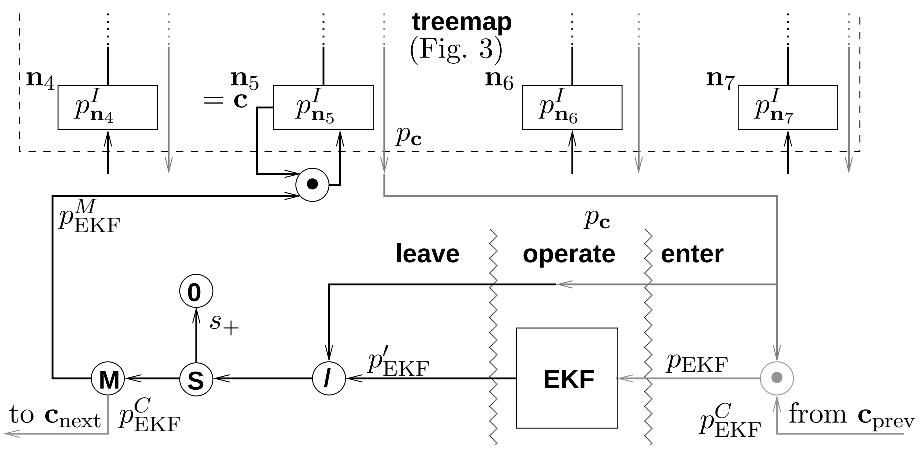


Figure 8: **Data flow view.** Before multiplying the obtained information into the treemap as in figure 6 the robot pose is marginalized out (Ⓜ). The resulting conditional $p\big(X_r \big| X[\mathbf{c}\colon\downarrow], z, s^{-1}\big)$ is passed to the next region $\mathbf{c}_{\text{next}}$. There it is multiplied into the new EKF state ($\odot$). Before that sparsification (ⓢ) removes couplings between X_r and $X[\mathbf{c}\colon\downarrow \wedge \mathbf{c}_{\text{next}}\colon \not\downarrow]$ discarding information (⓪). So the conditional is $p\big(X_r \big| X[\mathbf{c}\colon\downarrow \wedge \mathbf{c}_{\text{next}}\colon\downarrow], z, s^{-1}\big)$ and can indeed be multiplied ($\odot$) into the new EKF.

from the treemap. Hence it can be used as if it was new information even though it is derived from measurements taken in the old region.

If we did so directly, ⓡ was conditioned on all landmarks in the old region. So multiplying it into the new region $\mathbf{c}$, we would end up with a region involving all landmarks seen so far. For example in figure 7 ⓡ would involve $X_{\mathsf{a}}, X_{\mathsf{b}}$ and so would the new region.

To avoid this, before marginalization all couplings of the robot pose are removed by sparsification except those with landmarks in the new region. This sacrifices some landmark information. Then the robot pose is marginalized out and the resulting conditional is only conditioned on landmarks also involved in the new region $\mathbf{c}$. Thus it can be directly integrated there. In figure 7 ⓡ is conditioned only on X_{b} not on X_{a}.

6.2 Data Flow View

Figure 8 shows the data flow between EKF and treemap when passing robot pose information to the next region. It is the same as in figure 6 except for the sparsification (ⓢ) and marginalization step (Ⓜ).

The approach unfortunately is difficult to write down in $p(\ldots|\ldots)$ notation. The are two notational complications: First, since we sparsify, distributions are not conditioned on $\ldots|z)$ but rather on "$\ldots|z$ *except discarded information*)" instead. We will denote this by $\ldots|z, s^{-1})$ with

s referring to the distributions discarded. This notation suggests

$$p(X|z, s^{-1}) \cdot p(X|s) = p(X|z, s^{-1}, s) = p(X|z) \tag{42}$$

and actually the original distribution is the product of the approximated distribution and the distribution(-s) we discarded by sparsification. As the notation implies, $p(X|s)$ is a proper distribution, in our case a Gaussian with positive semidefinite information matrix. So the approximation is probabilistically consistent and the covariance reported by treemap is at least as large as the exact one without sparsification.

Second p^M_{EKF} is formally different than in (40) because it is also conditioned on the robot pose information passed from the previous region, not only on z_+. We symbolically denote this as $p\big(X[\mathbf{c}\colon\downarrow]\big|z_+, r\big)$ since r can indeed be treated as a probabilistic constraint (Fig. 7). The constraint r is derived from previous measurements z_- but is still independent from the landmark-marginal in the treemap. Thus it can be formally included in the constraints $z[\mathbf{c}\colon\downarrow]$ assigned to leaf $\mathbf{c}$ in the treemap. Computationally this is no difference anyway, because treemap just stores $p^I_{\mathbf{n}} = p\big(X[\mathbf{n}\colon\downarrow]\big|z[\mathbf{n}\colon\downarrow]\big)$ as a single Gaussian at each leaf. It does not keep track of which measurement is integrated there.

After these preliminaries we will derive the probabilistic computations. When entering region $\mathbf{c}$, $\mathbf{c}_{\text{prev}}$ passes the robot pose constraint

$$p^C_{\text{EKF}} = p\big(X_r\big|X[\mathbf{c}_{\text{prev}}\colon\downarrow \wedge \mathbf{c}\colon\downarrow], z_-, s_-^{-1}\big) \tag{43}$$

which is shown as $\textcircled{r}$ in figure 7. It is multiplied ($\odot$) with $p_{\mathbf{c}}$ resulting in the initial EKF state

$$\begin{aligned} p_{\text{EKF}} = p_{\mathbf{c}} \cdot p^C_{\text{EKF}} &= p\big(X[\mathbf{c}\colon\downarrow]\big|z_-, s_-^{-1}\big) \cdot p\big(X_r\big|X[\mathbf{c}\colon\downarrow], z_-, s_-^{-1}\big) && (44)\\ &= p\big(X_r, X[\mathbf{c}\colon\downarrow]\big|z_-, s_-^{-1}\big). && (45)\end{aligned}$$

Pose X_r depends only on $X[\mathbf{c}_{\text{prev}}\colon\downarrow \wedge \mathbf{c}\colon\downarrow]$ so p^C_{EKF} defines a distribution on $X_r, X[\mathbf{c}\colon\downarrow]$. While operating in $\mathbf{c}$ the EKF maintains the posterior

$$p'_{\text{EKF}} = p\big(X'_r, X[\mathbf{c}\colon\downarrow]\big|z, s_-^{-1}\big) \tag{46}$$

on landmarks and the new robot pose X'_r.

Now let us assume we leave $\mathbf{c}$ again (Fig. 8). The EKF's distribution p'_{EKF} is divided by $p_{\mathbf{c}}$. We cannot factor into $p(\dots|z_+) \cdot p(\dots|z_-, s_-^{-1})$ as in (40) because both are connected by pose X_r. Factorization holds only as long as X_r is not marginalized out.

$$= \int_{X_r} p\big(X'_r, X_r, X[\mathbf{c}\colon\downarrow]\big|z, s_-^{-1}\big) \tag{47}$$

$$= \int_{X_r} p\big(X'_r, X_r, X[\mathbf{c}\colon\downarrow]\big|z_+\big) \cdot p\big(X_r, X[\mathbf{c}\colon\downarrow]\big|z_-, s_-^{-1}\big) \tag{48}$$

$$= \int_{X_r} p\big(X_r', X_r, X[\mathbf{c}\colon\downarrow]\big|z_+\big)\; p\big(X[\mathbf{c}\colon\downarrow]\big|z_-, s_-^{-1}\big)\; p\big(X_r\big|X[\mathbf{c}\colon\downarrow], z_-, s_-^{-1}\big) \tag{49}$$

$$\frac{p'_{\text{EKF}}}{p_{\mathbf{c}}} = \int_{x_r} p\big(X_r', X_r, X[\mathbf{c}\colon\downarrow]\big|z_+\big) \cdot p\big(X_r\big|X[\mathbf{c}\colon\downarrow], z_-, s_-^{-1}\big) \tag{50}$$

The $p(X_r|\dots)$ term is the robot pose passed. We write $p(\dots|r)$ getting[5]

$$= \int_{X_r} p\big(X_r', X_r, X[\mathbf{c}\colon\downarrow]\big|z_+\big) \cdot p\big(X_r, X[\mathbf{c}\colon\downarrow]|r\big) \tag{51}$$

$$= \int_{X_r} p\big(X_r', X_r, X[\mathbf{c}\colon\downarrow]\big|z_+, r\big) \tag{52}$$

$$= p\big(X_r', X[\mathbf{c}\colon\downarrow]\big|z_+, r\big). \tag{53}$$

Overall, after dividing by $p_{\mathbf{c}}$ we have the constraints obtained by new measurements z_+ plus the constraint r passed from the previous region. The sparsification operator $\textcircled{s}$ factorizes the result as a product

$$= p\big(X_r', X[\mathbf{c}\colon\downarrow]\big|z_+, r, s_+^{-1}\big) \cdot p\big(X_r', X[\mathbf{c}\colon\downarrow]\big|s_+\big) \tag{54}$$

of a sparser distribution and a distribution s_+ discarded ($\textcircled{0}$). It makes X_r' depend only on landmarks both in $\mathbf{c}$ and in the next region $\mathbf{c}_{\text{next}}$.

$$p\big(X_r'\big|X[\mathbf{c}\colon\downarrow], z_+, r, s_+^{-1}\big) = p\big(X_r'\big|X[\mathbf{c}\colon\downarrow \wedge \mathbf{c}_{\text{next}}\colon\downarrow], z_+, s_+^{-1}, r\big) \tag{55}$$

So when X_r' is marginalized out in the next step ($\textcircled{M}$) the result is

$$p^M_{\text{EKF}} \cdot p^C_{\text{EKF}} \text{ with } \quad p^M_{\text{EKF}} = p\big(X[\mathbf{c}\colon\downarrow]\big|z_+, s_+^{-1}, r\big) \text{ and} \tag{56}$$

$$p^C_{\text{EKF}} = p\big(X_r'\big|X[\mathbf{c}\colon\downarrow \wedge \mathbf{c}_{\text{next}}\colon\downarrow], z_+, s_+^{-1}, r\big) \tag{57}$$

$$= p\big(X_r'\big|X[\mathbf{c}\colon\downarrow \wedge \mathbf{c}_{\text{next}}\colon\downarrow], z, s^{-1}\big). \tag{58}$$

The marginal p^M_{EKF} is independent from the landmarks given z_-, s_-^{-1}. It is multiplied into $p^I_{\mathbf{c}}$ thereby passing the landmark information to the treemap. The conditional p^C_{EKF} is independent of both and passed to the next region $\mathbf{c}_{\text{next}}$, where it is used to initialize the EKF again.

6.3 Gaussian Implementation

In this section we will derive the sparsification operation $\textcircled{s}$. The remaining operations are the same as in (Sec. 3.5) being implemented by (24), $\odot$ (information form); (27) and (30), $\textcircled{M}$; (31) and (33), $\odot$ (covariance form). Operation $\textcircled{0}$ is just a symbol for discarding information.

[5]Note, that X_r is the robot pose when entering $\mathbf{c}$, i.e. a random variable, whereas r is the distribution integrated as a prior in the new region, i.e. a Gaussian constraint.

Table 2: Random variables in the distribution $\frac{p'_{\mathrm{EKF}}}{p_{\mathbf{c}}}$ to be sparsified.

Block in A	Notation	Random variables
1	X'_r	Current robot pose.
2	$X[\mathbf{c}\colon\downarrow \wedge \mathbf{c}_{\mathrm{next}}\colon \not\downarrow]$	Landmarks in old but not in new region.
3	$X[\mathbf{c}\colon\downarrow \wedge \mathbf{c}_{\mathrm{next}}\colon \downarrow]$	Landmarks in both old and new region.

The sparsification procedure has the same job as the SEIF sparsification procedure (Thrun et al., 2004). In contrast to the approach taken here, SEIF introduces overconfidence (Eustice et al., 2005), possibly reporting smaller covariances than actually true. On the other hand the sparsification procedure derived here can not be used for SEIF since it introduces further links that spoil overall sparsity. Treemap is concerned only with removing the links between X_r and $X[\mathbf{c}\colon\downarrow \wedge \mathbf{c}_{\mathrm{next}}\colon \not\downarrow]$ and sparsifies a local Gaussian (54) that is already dense. The difference is subtle. SEIF first integrates and then sparsifies the global information matrix. Treemap first sparsifies a local matrix and then integrates the result into the overall sparse matrix represented by the tree.

We do not change the Gaussian's mean so sparsification only concerns the information matrix A. Let therefore A be permuted as a 3×3 block matrix according to table 2. Sparsification replaces A by A' with $A'_{21} = A'_{12} = 0$, $0 \le A' \le A$, and $A - A'$ being as small as possible. Thereby links between X_r (block row / column 1) and landmarks not in the new region (block row / column 2) are removed. We want to loose as little information as possible. So following requirement (R1), the goal is to minimize the largest factor

$$\lambda_{\max}(A'^{-1}, A^{-1}) = \max_v \frac{v^T A'^{-1} v}{v^T A^{-1} v} = \min_{\alpha A' \ge A} \alpha \qquad (59)$$

by which the covariance increases considering any linear combination of random variables. This is the largest generalized eigenvalue of A'^{-1} relative to A^{-1} (Frese, 2006a). We cannot provide an optimal solution in general, but the following theorem gives an optimal solution if the first block row / column is 1-D. It removes the links between $X[\mathbf{c}\colon\downarrow \wedge \mathbf{c}_{\mathrm{next}}\colon \not\downarrow]$ and one of the three DOF of the robot pose. The proof can be found in (Frese, 2004, §3.7). Here the theorem is used as a black box formula, that for a matrix A (60) makes the desired part $A'_{21} = 0$ zero (61) without being overconfident ($0 \le A' \le A$).

Theorem 1 *Let* $0 \le A$ *be a* 3×3 *block matrix being decomposed as*

$$A = \begin{pmatrix} \psi & r^T & w^T \\ r & S & W^T \\ w & W & X \end{pmatrix} \qquad (60)$$

with 1-dimensional first block row / column. Then among all

$$A' = \left(\begin{smallmatrix} * & 0 & * \\ 0 & * & * \\ * & * & * \end{smallmatrix}\right), \ \textit{with } 0 \le A' \le A \textit{ and } A'_{21} = A'_{12} = 0 \tag{61}$$

the following matrix A' minimizes $\lambda_{\max}(A'^{-1}, A^{-1})$.

$$A' = A - uu^T, \ \textit{with } u = A\begin{pmatrix} \gamma \\ \delta S^{-1} r \\ 0 \end{pmatrix}, \ \textit{and} \tag{62}$$

$$\begin{gathered} \gamma = \beta(\lambda - \lambda^{-1}\alpha), \ \delta = \beta(-\lambda + \psi\lambda^{-1}), \\ \alpha = r^T S^{-1} r, \quad \beta = (\psi - \alpha)^{-1}, \lambda = \sqrt[4]{\psi(r^T S^{-1} r)}. \end{gathered} \tag{63}$$

The matrix S can sometimes be singular when some landmark has not been observed while operating in a region. Still since A is positive semidefinite $S^{-1}r$ exists[6], i.e. there is a solution to $Sv = r$ and all solutions lead to the same $r^T v$ and Wv. This solution can be rigorously obtained by Singular Value Decomposition (SVD) (Press et al., 1992, §2.6) or less rigorously by adding a small ϵ to the diagonal of S.

In order to remove the links between the robot pose X_r and $X[\mathbf{c}\colon\downarrow \wedge \mathbf{c}_{\text{next}}\colon \not\downarrow]$ treemap applies theorem 1 three times. In the first step row / column 1 of A is the robot's x position, in the next step y, then θ. After each step the first column is marginalized out to avoid that following steps are introducing links again. The final sparsified matrix A' is

$$A' = A - u_1 u_1^T - u_2 u_2^T - u_3 u_3^T, \tag{64}$$

where $u_{1,2,3}$ are the u vectors (62) in the three applications of theorem 1.

Each step is optimal with respect to (R1) among all matrices with the sparsity pattern (61). However the overall sparsification after repeatedly changing regions will probably be suboptimal. Still each measurement is affected by sparsification only once, namely when it is passed from the EKF to the treemap. At the very least, this serves to prevent the errors introduced by sparsification accumulating over time. We will experimentally investigate treemap's performance with respect to (R1) in (Sec. 9) reporting the actual increase of error encountered.

7 Maintenance of the Tree

In this section we will discuss the bookkeeping part of the algorithm. It maintains the tree that is not defined a-priori but built while the map grows. There are three subtasks.

[6]For a positive semidefinite matrix the image space of an off-diagonal block is contained in the image space of the corresponding diagonal block. Let therefore $0 \le \left(\begin{smallmatrix} \psi & r^T \\ r & S \end{smallmatrix}\right)$ and $v \in \text{kernel}(S)$. Then $0 \le \left(\begin{smallmatrix} 1 & 0 \\ 0 & v \end{smallmatrix}\right)^T \left(\begin{smallmatrix} \psi & r^T \\ r & S \end{smallmatrix}\right) \left(\begin{smallmatrix} 1 & 0 \\ 0 & v \end{smallmatrix}\right) = \left(\begin{smallmatrix} \psi & r^T v \\ v^T r & 0 \end{smallmatrix}\right)$. It follows that $r^T v$ must be 0, so r is orthogonal to kernel(S) and hence $r \in \text{image}(S)$.

1. Determine for which nodes to update $p^M_{\mathbf{n}}$ and $p^C_{\mathbf{n}}$ by (24) ($\odot$), as well as (27) and (30) ($\circledM$). This task is pure bookkeeping.
2. Control the transition between the current region, $\mathbf{c}$, and the next region, $\mathbf{c}_{\text{next}}$. This defines which constraints are assigned to which leaf although the assignment is not explicitly stored. We rely upon a heuristic that limits a region's geometric extension by *maxD*.
3. Rearrange the tree so it is balanced and well partitioned, i.e. $x[\mathbf{n}\colon\downarrow\uparrow]$ contains few landmarks in all nodes $\mathbf{n}$. Balancing is not difficult but hierarchical tree partitioning (HTP) is NP-complete. So we follow the tradition in graph partitioning (Fiduccia and Mattheyses, 1982) and optimize in greedy steps with each step being optimal.

The goal was to make treemap $O(k^3 \log n)$ in a strict asymptotical sense given that the HTP subalgorithm succeeds in finding a suitable tree (Def. 1). Unfortunately this results in a relatively involved implementation. We therefore discuss the general approach and leave the details to the pseudocode in appendix A. We currently investigate (Frese and Schröder, 2006) how treemap can be simplified when sacrificing the $O(k^3 \log n)$ bound.

7.1 Update

Treemap has to keep track of which landmark is involved where and when to marginalize out a landmark. So distributions $p^M_{\mathbf{n}}$, $p^C_{\mathbf{n}}$ contain a sorted list $\mathcal{L}^M_{\mathbf{n}}$, $\mathcal{L}^C_{\mathbf{n}}$ denoting the landmarks represented by the different rows / columns of the corresponding matrices and vectors. For each landmark it also contains a counter that is 1 in the leaves and added when multiplying distributions ($\odot$). There is also a global landmark array $\mathcal{L}$ with corresponding counters. We treat both as multisets writing $\uplus$ for union with adding counters and $\mathsf{l}\#\mathcal{L}$ for the counter of l in $\mathcal{L}$. Treemap detects when to marginalize out a landmark by comparing the counters passed to the node with the global counter.

$$\mathcal{L}'_{\mathbf{n}} = \mathcal{L}^M_{\mathbf{n}_\swarrow} \uplus \mathcal{L}^M_{\mathbf{n}_\searrow} \tag{65}$$

$$\mathcal{L}^M_{\mathbf{n}} = \left\{\mathsf{l} \in \mathcal{L}'_{\mathbf{n}} | 0 < \mathsf{l}\#\mathcal{L}'_{\mathbf{n}} < \mathsf{l}\#\mathcal{L}\right\} \tag{66}$$

$$\mathcal{L}^C_{\mathbf{n}} = \left\{\mathsf{l} \in \mathcal{L}'_{\mathbf{n}} | 0 < \mathsf{l}\#\mathcal{L}'_{\mathbf{n}} = \mathsf{l}\#\mathcal{L}\right\} \tag{67}$$

It further maintains an array $\mathbf{lca}[\mathsf{l}]$ storing for each landmark l the least common ancestor of all leaves involving l. It is that node, that satisfies $\mathsf{l}\#\mathcal{L}'_{\mathbf{lca}[\mathsf{l}]} = \mathsf{l}\#\mathcal{L}$ and where l is marginalized out.

If $p^I_{\mathbf{n}}$ changes – for instance by multiplying p^M_{EKF} into $p^I_{\mathbf{c}}$ – all $p^M_{\mathbf{m}}$ and $p^C_{\mathbf{m}}$ are updated from $\mathbf{m} = \mathbf{n}$ up to the root. The same applies if a new leaf has been inserted. Additionally $\mathbf{lca}[\mathsf{l}]$ can change for a landmark involved in $p^I_{\mathbf{n}}$ and all nodes from the old $\mathbf{lca}[\mathsf{l}]$ to the root are updated too. By definition 1.2, only $O(1)$ leaves share landmarks with a given leaf so $O(\log n)$ nodes are updated in $O(k^3 \log n)$ computation time.

In the following we need to find all leaves involving a given landmark l. We recursively go down from $\mathbf{m} = \mathbf{lca}[\mathsf{l}]$ as far as $\mathsf{l} \in \mathcal{L}^M_{\mathbf{m}}$. By definition 1.2 this holds for only $O(1)$ leaves taking $O(k \log n)$ time.

7.2 Region Changing Control Heuristic

Let treemap currently operate in a region, i.e. a leaf $\mathbf{c}$. The EKF directly handles odometry, observation of new landmarks, and of landmarks in $X[\mathbf{c}\colon\downarrow]$. There are two reasons to leave $\mathbf{c}$ and enter another region $\mathbf{c}_{\text{next}}$. First a landmark may be observed that is not within $\mathbf{c}$, i.e. $X[\mathbf{c}\colon\not\downarrow]$. In this case we transition to a region containing this landmark so as to pass information on that landmark from the treemap to the EKF. A second reason is the need to limit the number of landmarks in a region for efficiency. We actually limit the distance *maxD* between landmarks in the same region instead of directly limiting the number of landmarks. This allows us to later add landmarks that have been overlooked.

As we transition from $\mathbf{c}$ to $\mathbf{c}_{\text{next}}$, the two regions must share at least two landmarks, to avoid disintegration of the map due to the omitted odometry link. Thus, treemap checks, whether $\mathbf{c}$ must be left for one of the two reasons above and determines $\mathbf{c}_{\text{next}}$ with these steps:

1. Find all leaves sharing at least two landmarks with $\mathbf{c}$.
2. For each of these leaves verify whether *maxD* would be exceeded when adding the landmarks currently in the robot's field of view.
3. Among those where it is not exceeded, choose $\mathbf{c}_{\text{next}}$ as the one that already involves most of the landmarks in the robot's field of view.
4. If all leaves would exceed *maxD* then add a new leaf as $\mathbf{c}_{\text{next}}$.
5. Leave $\mathbf{c}$. Add landmarks observed to $\mathbf{c}_{\text{next}}$ and enter $\mathbf{c}_{\text{next}}$.

When a new leaf is added, it is inserted directly above the root node. It will then be moved to a better location by the HTP subalgorithm.

7.3 Hierarchical Tree Partitioning (HTP)

The HTP subalgorithm optimizes the tree while the robot moves. The goal is to meet definition 1, which is the prerequisite for our $O(\dots)$ analysis. The problem is equivalent to the *Hierarchical Tree Partitioning Problem* known from graph theory and parallel computing and being NP-complete. However, successful heuristic algorithms have been developed (Vijayan, 1991) – the most popular of which is the Kernighan and Lin heuristic (Fiduccia and Mattheyses, 1982). It employs a greedy strategy in each step moving that node which minimizes the cost function. Hendrickson and Leland (1995) report that it works especially well when applied hierarchically. We can do this easily since we optimize an existing tree. Overall the HTP subalgorithm makes $O(1)$ optimization steps (5 in our experiments) whenever changing regions, so the time spent in partitioning is limited. It is heuristic experience and formally part of definition 1 that this suffices to maintain a well partitioned tree.

In each optimization step we choose one node $\mathbf{r}$ to optimize. We move a subtree somewhere left-below that node to the right side or vice versa. This affects $\mathbf{r}$ and its descendents but we only consider $\mathbf{r}$ itself, priorizing parents over children. The cost function that is optimized is

$$\operatorname{par}(\mathbf{r}) = |\mathcal{L}^M_{\mathbf{r}_\swarrow}| + |\mathcal{L}^M_{\mathbf{r}_\searrow}| = \left|X[\mathbf{r}_\swarrow\colon\downarrow\uparrow]\right| + \left|X[\mathbf{r}_\searrow\colon\downarrow\uparrow]\right| \tag{68}$$

the number of landmarks involved in $p^M_{\mathbf{r}_\swarrow}$ and $p^M_{\mathbf{r}_\searrow}$. This number determines the size of the matrices involved in computation at $\mathbf{r}$. The subtree that we choose to move from one side of $\mathbf{r}$ to the other is that which minimizes $\operatorname{par}(\mathbf{r})$. The cost function depends only on which subtree to move, not on where to move it. Therefore the optimal subtree is found by recursively going through all descendants $\mathbf{s}$ of $\mathbf{r}$ that share a landmark with $\mathbf{r}$. At each node, $\operatorname{par}(\mathbf{r})$ is evaluated for the situation that $\mathbf{s}$ was moved to the other side of $\mathbf{r}$.

$$\begin{aligned} \mathsf{l} \in \mathcal{L}'^M_{\mathbf{r}_\swarrow} \quad &\Leftrightarrow \quad 0 < \mathsf{l}\#\mathcal{L}^M_{\mathbf{r}_\swarrow} \mp \mathsf{l}\#\mathcal{L}^M_{\mathbf{s}} < \mathsf{l}\#\mathcal{L} \\ \mathsf{l} \in \mathcal{L}'^M_{\mathbf{r}_\searrow} \quad &\Leftrightarrow \quad 0 < \mathsf{l}\#\mathcal{L}^M_{\mathbf{r}_\searrow} \pm \mathsf{l}\#\mathcal{L}^M_{\mathbf{s}} < \mathsf{l}\#\mathcal{L} \end{aligned} \tag{69}$$

$$\begin{aligned} \operatorname{par}(\mathbf{r})_{\mathbf{s}} = &\left|\left\{\mathsf{l}\,\middle|\,0 < \mathsf{l}\#\mathcal{L}^M_{\mathbf{r}_\swarrow} \mp \mathsf{l}\#\mathcal{L}^M_{\mathbf{s}} < \mathsf{l}\#\mathcal{L}\right\}\right| \\ &+ \left|\left\{\mathsf{l}\,\middle|\,0 < \mathsf{l}\#\mathcal{L}^M_{\mathbf{r}_\searrow} \pm \mathsf{l}\#\mathcal{L}^M_{\mathbf{s}} < \mathsf{l}\#\mathcal{L}\right\}\right| \end{aligned} \tag{70}$$

The case with $-$ and $+$ corresponds to moving from $\swarrow$ to $\searrow$, $+$ and $-$ corresponds to the other way. Each evaluation is performed by (70) in $O(k)$ time using the counters in $\mathcal{L}^M_{\dots}$. Node $\mathbf{r}$ involves $O(k)$ landmarks each in turn involved at $O(1)$ leaves so overall $O(k \log n)$ nodes are checked in $O(k^2 \log n)$ computation time. The tree should be kept balanced. Thus $\mathbf{s}$ is only considered, if after moving

$$\tfrac{1}{2}\mathbf{r}_{\swarrow\,size} \le \mathbf{r}_{\searrow\,size} \le 2\mathbf{r}_{\swarrow\,size} \tag{71}$$

where $\mathbf{n}_{size}$ is the number of leaves below $\mathbf{n}$.

We still have to determine exactly where to insert $\mathbf{s}$. For $\operatorname{par}(\mathbf{n})$ it only matters, whether $\mathbf{s}$ is inserted *somewhere left-below* ($\operatorname{par}(\mathbf{n})_\swarrow$), *somewhere right-below* $\mathbf{n}$ ($\operatorname{par}(\mathbf{n})_\searrow$), or *directly above* ($\operatorname{par}(\mathbf{n}_\uparrow)_\uparrow$).

$$\operatorname{par}(\mathbf{n}_\uparrow)_\uparrow = \left|\mathcal{L}^M_{\mathbf{s}}\right| + \left|\mathcal{L}^M_{\mathbf{n}}\right| \tag{72}$$

$$\mathrm{par}(\mathbf{n})_{\swarrow} = \left|\left\{\mathsf{l}\middle|0 < \mathsf{l}\#\mathcal{L}^M_{\mathbf{n}_\swarrow} + \mathsf{l}\#\mathcal{L}^M_{\mathbf{s}} < \mathsf{l}\#\mathcal{L}\right\}\right| + \left|\mathcal{L}_{\mathbf{n}_\searrow}\right| \tag{73}$$

$$\mathrm{par}(\mathbf{n})_{\searrow} = \left|\mathcal{L}_{\mathbf{n}_\swarrow}\right| + \left|\left\{\mathsf{l}\middle|0 < \mathsf{l}\#\mathcal{L}^M_{\mathbf{n}_\searrow} + \mathsf{l}\#\mathcal{L}^M_{\mathbf{s}} < \mathsf{l}\#\mathcal{L}\right\}\right| \tag{74}$$

So the insertion point that minimizes par(...) priorizing parents over children can be found by descending through the tree as follows:

1. Start with $\mathbf{n} = \mathbf{r}_\searrow$ (or $\mathbf{r}_\swarrow$ resp.)
2. Evaluate par($\mathbf{n}$) for each of the three choices *directly above* (72), *somewhere left-below* (73), or *somewhere right-below* (74).
3. If directly above is best and the new parent of $\mathbf{n}$ and $\mathbf{s}$ would be balanced (71) then insert $\mathbf{s}$. Update from old and new $\mathbf{s}$ to the root.
4. Else set $\mathbf{n}$ to $\mathbf{n}_\swarrow$ or $\mathbf{n}_\searrow$ whichever is better and go to step 2.

8 Comparison with the Thin Junction Tree Filter

Paskin (2003) has proposed an algorithm, the *Thin Junction Tree Filter* (TJTF), which is closely related to the treemap algorithm, although both have been independently developed from completely different perspectives[7]. Paskin views the problem as a Gaussian graphical model. He utilizes the fact that if a set of nodes (i.e. a set of landmarks) separates the graphical model into two parts, then these parts are conditionally independent given estimates for the separating nodes. The algorithm maintains a junction tree, where each edge corresponds to such a separation, passing marginalized distributions along the edges.

Treemap's tree is very similar to TJTF's junction tree. The most important difference is how treemap and TJTF ensure that no node involves too many landmarks. TJTF further sparsifies thereby sacrificing information for computation time. Treemap on the other hand tries to rearrange the tree with its HTP subalgorithm to reduce the number of landmarks involved. It never sacrifices information except when integrating an observation into the tree the first time. There are arguments in favor of both approaches. If treemap succeeds in finding a good tree, that is certainly better than sacrificing information. However, if no such suitable tree exists, this question is debatable.

Consider the example in Section 4 of densely mapping an open plane. This is not topologically suitable and treemap's computation time will increase to $O(n^{3/2})$. TJTF in contrast will force each node to involve

[7]Originally I developed treemap from a hierarchy-of-regions and linear-equation-solving perspective. I later added the Bayesian view provided in this article.

only $O(k)$ landmarks by sparsification, saving it's $O(k^3 n)$ computation time. But is the posterior represented still a good approximation?

Let us consider one node of TJTF's tree that roughly divides the map into equal halves. Originally these halves have an $O(\sqrt{n})$ border where landmarks are tightly linked to both halves of the map. The considered node represents only $O(k)$ landmarks, so most of these landmarks loose their probabilistic link to one half of the map during sparsification. This is not just slightly increasing the estimation error but actually introduces breaks in the map, violating for instance (R1).

A further difference between treemap and TJTF is that treemap views its tree as a part-whole hierarchy with a designated root corresponding to the whole building. For TJTF on the other hand the tree is just an acyclic graph without designated root. This difference leads to the data flow structure in treemap whereby the posterior is updated as information matrices are passed upwards after which inference occurs with the mean and, optionally covariance calculations downwards. Together with the representation of $p^C_{\mathbf{n}}$ by H, h (27) this reduces the computation time for the mean from $O(k^3)$ per node to $O(k^2)$ per node compared to passing information matrices downwards.

Treemap saves a further factor of $O(k)$ by taking care that each landmark is only involved in $O(1)$ leaves, so there are $O(\frac{n}{k})$ nodes. This is at least typically enforced by the region changing control heuristic, and for the analysis it is formally assumed by definition 1.2. Thereby, treemap groups measurements as geometrically contiguous regions, whereas TJTF chooses the node that minimizes the KL divergence during sparsification. Overall this leads to a computation time for mean recovery of $O(kn)$ for treemap vs. $O(k^3 n)$ for TJTF.

Treemap maintains a balanced tree thereby limiting update and computation of a local estimate to $O(k^3 \log n)$. Paths in TJTF's tree however may have a length of $O(n)$ so it cannot update that fast exactly.

Summarizing the discussion, treemap applies a more elaborate bookkeeping to reduce computation time. This bookkeeping on the other hand makes it considerable more difficult to implement than TJTF.

9 Simulation Experiments

This section presents the simulation experiments conducted to verify the algorithm with respect to the requirements (R1)-(R3). Clearly space (R2) and time (R3) consumption are straightforward to measure but how should one assess map quality with respect to requirement (R1)? It should be kept in mind, that our focus is on the core estimation algorithm, not on the overall system. So relative, not absolute, error is the quantity to be considered. This is achieved by generalized eigenvalues.

Table 3: Artificial noise proportional to: [a]distance, [b]observation angle, [c]square root of distance traveled (effectively), [d]distance traveled. The distance bias causes a huge orientation error (140°) in the large noise experiment reported in Section 11.

Scenario	Landmark sensor			Odometry		
	distance noise[a]	distance bias[b]	angular noise	velocity noise[c]	orient. bias[d]	robot radius
Small noise / Large scale	2.5%		2°	$0.01\sqrt{\text{m}}$		0.3m
Large noise	10%	$4.5\,\frac{\text{mm}}{\circ}$	5°	$0.07\sqrt{\text{m}}$	$5\,\frac{\circ}{m}$	0.3m

We therefore repeat the same experiment with independent measurement noise 1000 times passing the same measurements to treemap, EKF and the optimal ML estimator. We derive an error covariance matrix $C_{\text{treemap}}, C_{\text{ML}}, C_{\text{EKF}}$ for all three[8] and compare the square root of the generalized eigenvalue spectrum (Frese, 2006a). This spectrum illustrates the relative error in different aspects of the map, i.e. different linear combinations of landmark coordinates. In particular the smallest and largest eigenvalue bound the relative error of any aspect.

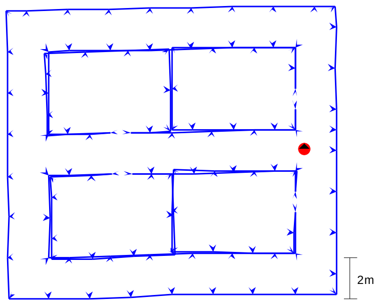
The experiments use the more sophicasted scheme for passing the robot pose described in the companion report. They have been conducted on an Intel Xeon, 2.67 GHz processor with the simulation parameters shown in table 3. The algorithm's parameters are $\mathit{optHTPSteps} = 5$ optimization steps and $\mathit{maxD} = 5$m region size.

9.1 Small Noise Experiment

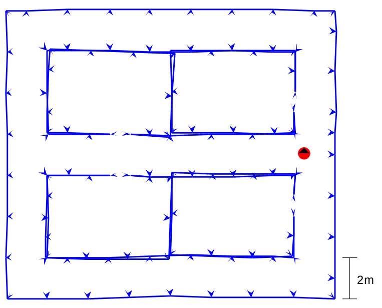
The small noise simulation experiment allows statistical evaluation of the estimation error and comparison with EKF and ML (Fig. 9). At first sight all three appear to be of the same quality (except for the left upper room in the treemap estimate) and perfectly usable for navigation. The orientation of the rooms appears to be an issue. There are no overlapping landmarks between room and corridor. Thus the larger error in treemap is likely caused when changing regions.

Figure 9d reports the relative error as a generalized eigenvalue spectrum. When comparing treemap vs. ML, the smallest relative error is 110% (87% vs. EKF) and the largest, 395% (181% vs. EKF). The median relative error is 137% compared to ML with two outliers of 395% and 293% and median 125% compared to EKF. The outliers are also apparent in the plot comparing EKF to ML, so they are probably caused by linearization errors occurring in EKF and treemap.

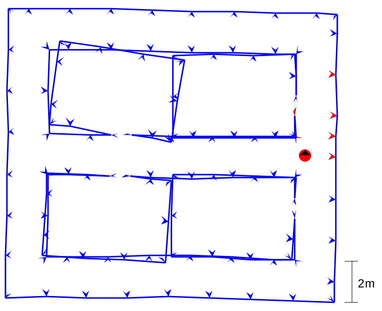
[8]To limit the number of necessary runs, only eight selected landmarks are used.



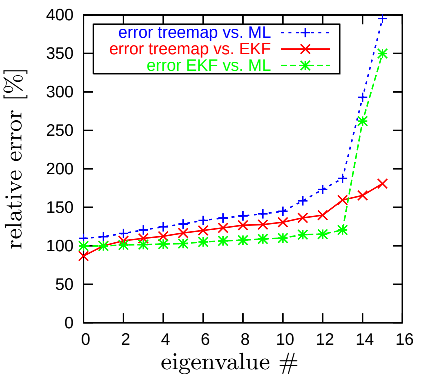
(a) Optimal ML estimate.



(b) EKF estimate.



(c) Treemap estimate.



(d) Relative error spectrum.

Figure 9: Small noise simulation results. (a)-(c) shows the estimate of ML, EKF, and treemap. (d) compares the relative error as a generalized eigenvalue spectrum. Treemap performs well relative to EKF but both suffer from linearization error.

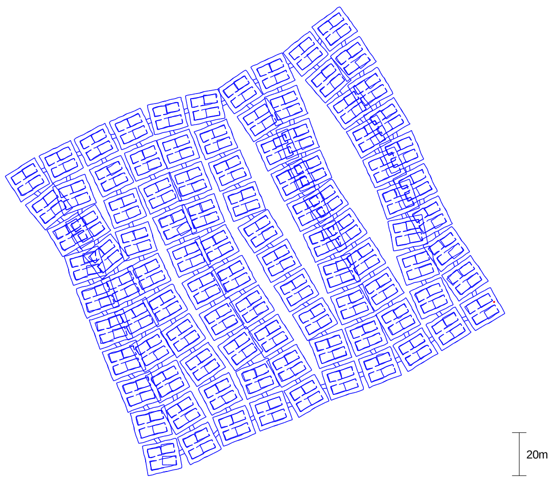


Figure 10: Large scale simulation experiment: Treemap estimate ($n = 11300$).

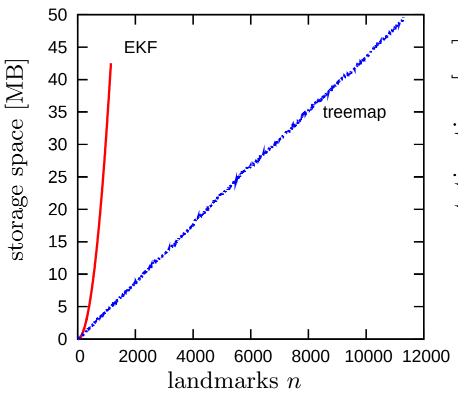
9.2 Large Scale Map Experiment

The second experiment is an extremely large map consisting of 10×10 copies of the building used before (Fig. 10). There are $n = 11300$ landmarks, $m = 312020$ measurements and $p = 63974$ robot poses. The EKF experiment was aborted due to large computation time.

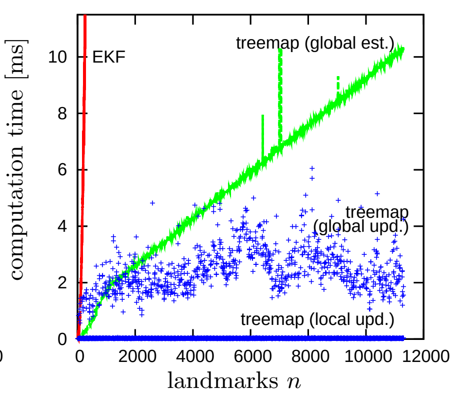
In figure 11a storage space consumption is clearly shown to be linear for treemap ($O(kn)$) and quadratic ($O(n^2)$) for EKF. Overall computation time was 31.34s for treemap and 18.89 days (extrapolated $\sim mn^2$) for EKF. Computation time per measurement is shown in figure 11b. Time for three different computations is given: Local updates (dots below < 0.5ms), global updates computing a local map (scattered dots above 0.5ms) and the additional cost for computing a global map are plotted w.r.t. n. Note that the global updates have a very fluctuating computation time because the number of nodes updated depends on the subtrees moved by the HTP subalgorithm. The spikes in the global estimation plot are caused by lost timeslices[9].

Overall the algorithm is extremely efficient updating an $n = 11300$ landmark map in 12.37ms. Average computation time is $1.21\mu\text{s} \cdot k^2$ for a local update, $0.38\mu\text{s} \cdot k^3 \log n$ for a global update, and $0.15\mu\text{s} \cdot kn$ for a global map (mean only), respectively ($k \approx 5.81$). The latter is surely the most impressive practical result. It allows the computation of a global map even for extremely large n, avoiding the complications of local map handling. Recently, we could even improve this result updating an $n = 1033009$ landmarks map in 443ms or in 23ms for a

[9]Processing time had 5ms resolution, so clock time has been used.



(a) Storage space.



(b) Computation time per measurement. Observe local updates ≈ 0.02ms.

Figure 11: Large scale simulation experiment: Storage space and computation time over number of landmarks n.

local update of ≈ 10000 landmarks (Frese and Schröder, 2006).

10 Real World Experiments

The real world experiment reported in this section shows how treemap works in practice by mapping the DLR Institute of Robotics and Mechatronics' building (Fig. 4) that serves as an example of a typical office building. The robot is equipped with a camera system (field of view: ±45°) at a height of 1.55m and controlled manually. We set circular fiducials throughout the floor (Fig. 12) that were visually detected by Hough-transform and a gray-level variance criterion (Otsu, 1979).

Since the landmarks are identical, identification is based on their relative position employing two different strategies in parallel. Local identification is performed by simultaneously matching all observations from a single robot pose to the map, taking into account both error in each landmark observation and error in the robot pose. For global identification we encountered considerable difficulties in detecting closure of a loop. Before closing the largest loop, the accumulated robot pose error was 16.18m (Fig. 13) and the average distance between adjacent landmarks was ≈ 1m. With indistinguishable landmarks, matching observations from a single image was not reliable enough.

Instead, the algorithm matches a map patch of radius 5m around the robot. When the map patch is recognized somewhere else in the map, the identity of all landmarks in the patch is changed accordingly and the loop is closed (Frese, 2004). It is a particular advantage of the treemap algorithm to be able to change the identity of landmarks already

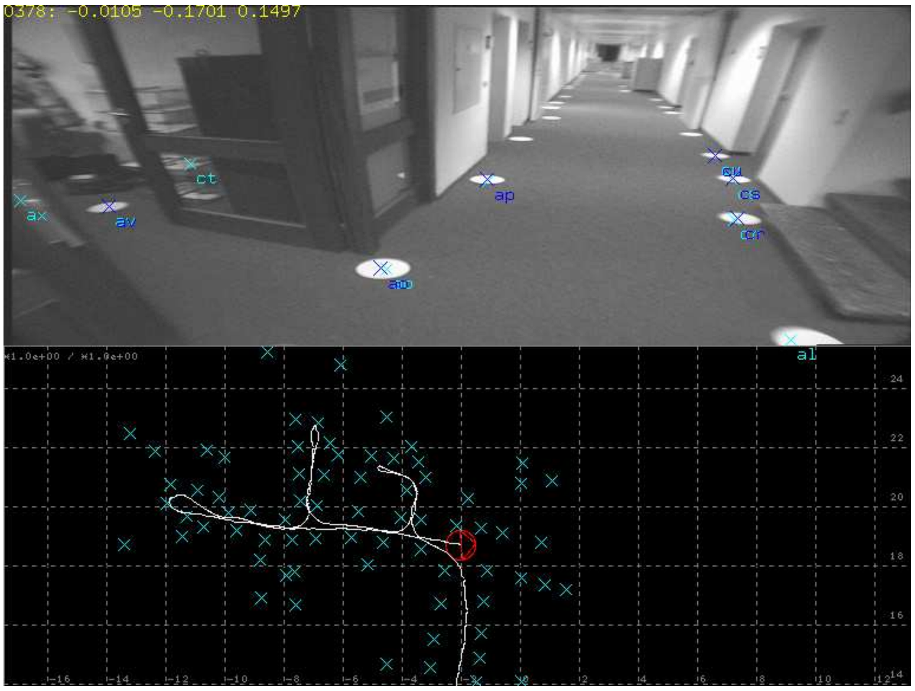


Figure 12: Screen shot of the SLAM implementation mapping the DLR building. The corresponding video can be downloaded from the author's website: `http://www.informatik.uni-bremen.de/~ufrese/slamvideos2_e.html`.

integrated into the map. This allows the use of the *lazy data association* framework by Hähnel et al. (2003). The regions were $maxD = 7$m large.

The final map contains $n = 725$ landmarks, $m = 29142$ measurements and $p = 3297$ robot poses (Fig. 13). The results highlight the advantage of using SLAM, because after closing the loop the map is much better. Figure 14 shows the internal tree representation ($k \approx 16.39$). The tree is balanced and well partitioned, i.e. no node represents too many landmarks. It can be concluded that the building is indeed topologically suitable in the sense discussed in Section 4.

If only a local update is performed, as is often the case (Fig. 15), then computation time is extremely low. The average time is $0.77\mu\text{s} \cdot k^2$ for a local update, $0.02\mu\text{s} \cdot k^3 \log n$ for a global update and $0.04\mu\text{s} \cdot kn$ for a global map (mean) respectively. Accumulated computation time is 2.95s for treemap and 660s (extrapolated $\sim mn^2$) for EKF.

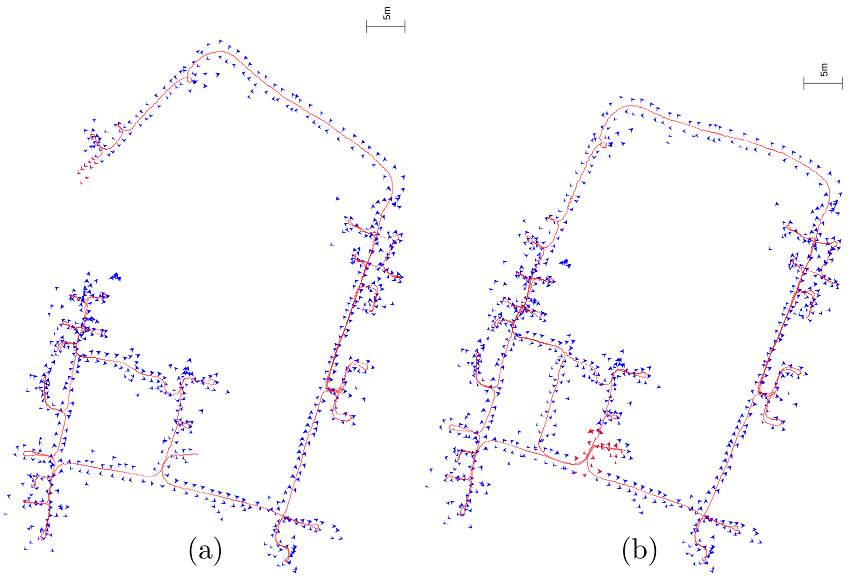


Figure 13: (a) Treemap estimate before closing the large loop having an accumulated error of 16.18m mainly caused by the robot leaving the building in the right upper corner. (b) Final treemap estimate after closing the large loop and returning to the starting position closing another loop.

11 A Nonlinear Extension

SLAM is essentially nonlinear. Nearly all approaches linearize the measurement functions and are thus subject to linearization error. As discussed by Frese (2006a), the most difficult source of nonlinearity occurs in the robot's orientation ϕ as a $\left(\begin{smallmatrix}\cos\phi & -\sin\phi \\ \sin\phi & \cos\phi\end{smallmatrix}\right)$ rotation matrix. It severely distorts the map, when the error in ϕ exceeds $\approx 15^\circ$ (Fig. 17c).

A thorough way to handle nonlinearities is to use an iterative nonlinear least square algorithm like Levenberg-Marquardt (Press et al., 1992, §12). This algorithm reevaluates all measurement Jacobians iteratively (*relinearization*) at the current estimate. It converges to the optimal ML solution but violates both (R2) and (R3). Treemap has a nonlinear extension that offers an intermediate solution between nonlinear ML and linearized least squares. It rotates the local distributions $p^I_{\mathbf{n}}$ and $p^M_{\mathbf{n}}$ according to the current estimate whenever updating a node (Fig. 16). As we will derive later, this is equivalent to reevaluating all Jacobians of measurements integrated below at a rotated linearization point.

Few approaches address nonlinearity without storing all measurements. FastSLAM (Montemerlo et al., 2002) can do so and Thrun and Liu (2003) have applied rotations to local Gaussians for joining maps made by different robots. The mathematics of rotation is the same as

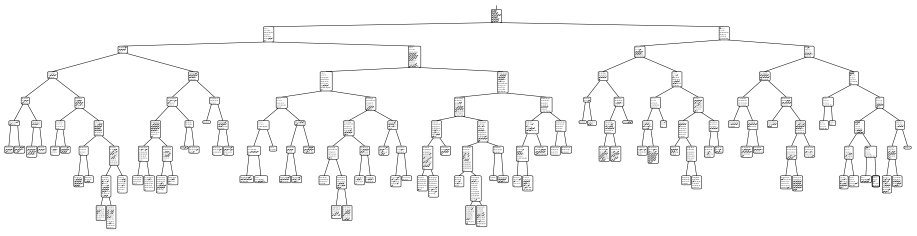
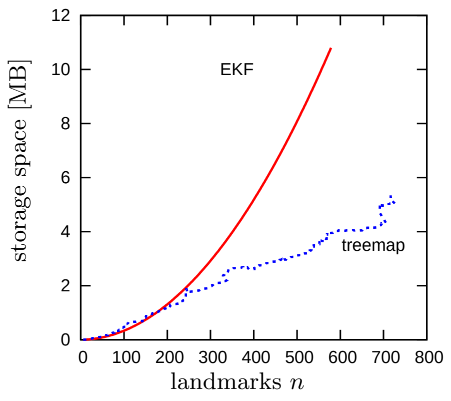
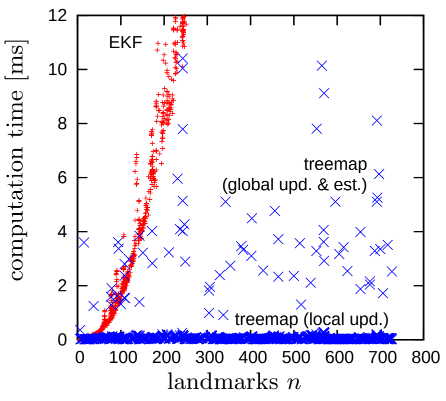


Figure 14: Tree representation of the map. The size of the node ovals is proportional to number of landmarks represented.



(a) Storage space.



(b) Computation time per measurement. Observe local updates ≈ 0.07ms.

Figure 15: Real experiment performance.

for treemap but the overall framework differs. Both approaches highlight an important point: to rotate parts of the map later on, these parts must be kept separate and not integrated into a large sparse matrix.

11.1 Data Flow View

Treemap's nonlinear extension does not aim at a rigorous treatment of all nonlinearities. The goal is simply to do better than linearized least square without storing all measurements and without much increase in computation time. To this end, treemap rotates the distributions passed by a node's children whenever this node is updated anyway as well as when closing a large loop. Primarily, submaps are still aligned by the integration of Gaussians during regular updates. The additional rotations just serve to reduce the error in the linearization point. Since linearization error grows quadratically with the error in the linearization point, the rotations do not have to fit perfectly anyway.

Figure 16 shows the nonlinear update in a node $\mathbf{n}$. Before multiplying $p^M_{\mathbf{n}_\swarrow}$ and $p^M_{\mathbf{n}_\searrow}$ both are rotated. Treemap basically keeps track of

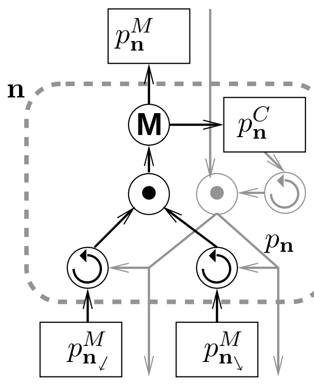


Figure 16: **Data flow view.** Data flow in a single node $\mathbf{n}$ with nonlinear extension. Before multiplying $p^M_{\mathbf{n}_\swarrow}$ and $p^M_{\mathbf{n}_\searrow}$ they are rotated according to the estimate $E(p_{\mathbf{n}})$.

the estimate that was used when the measurements integrated into a distribution have been linearized. This serves as a linearization point in a rough sense and is propagated together with the distribution.

- **Creation** When p'_{EKF} is constructed from the EKF state, store the current estimate for the landmarks as linearization point.

⊙ Combine both linearization points by taking a weighted average. As weight use the diagonal entries in the information matrix.

Ⓜ Drop landmarks marginalized out from the linearization point.

↻ Rotate the linearization point by least square matching to the current estimate. Rotate the distribution by the same angle.

Rotating $p^M_{\mathbf{n}}$ is equivalent to rotating the linearization point of all measurements assigned to leaves below $\mathbf{n}$. Thus, even when only updating a single path up the root, all linearization points are improved. Certainly large subtrees are rotated as a whole so this is not the same as if rotating all $p^I_{\mathbf{n}}$ independently. Still for a node $\mathbf{n}$ the relative orientation error of $\mathbf{n}_\swarrow$ and $\mathbf{n}_\searrow$ is usually much smaller than the absolute orientation error of $\mathbf{n}$. So the linearization error is improved without additional updates.

When $p^M_{\mathbf{n}}$ is rotated all $p^C_{\mathbf{n}'}$ below must be rotated too to be consistent. To save computation time we defer this step to when $p^C_{\mathbf{n}'}$ is actually used for recovering the mean. It is then rotated (↻) by the sum of all angles applied to marginal distributions above.

If $p^I_{\mathbf{n}}$ involves an absolute pose constraint it is not rotation invariant and cannot by relinearized by rotation. Thus the initial leaf and all ancestors are never rotated. In theory absolute information is passed along with the robot pose making all $p^I_{\mathbf{n}}$ non rotation invariant. However it decays exponentially, so we rotate all leaves except the initial one.

When a loop with very large orientation error is closed (Fig. 17) a useful estimate requires several iterations. Treemap iterates whenever a landmark is observed that is not involved at any leaf that shares landmarks with the current leaf. Then old and new current leaf and all that share landmarks with them are updated. This is repeated as long as any rotation above 2° is applied with $O(k^3 \log n)$ time per iteration.

11.2 Gaussian Implementation

Now we derive the concrete formulas for rotating Gaussians. First assume that the input to treemap's tree is directly defined by Gaussian landmark constraints without the preprocessing EKF stage.

$$-2 \log p(y|z) = \big(f(y) - z\big)^T C^{-1} \big(f(y) - z\big) \tag{75}$$

This expression defines the constraint, that $f(y) \approx z$ with an error covariance of C. The Gaussian $p^I_{\mathbf{n}}$ is derived by linearizing (75) at some linearization point $\breve{y}$ usually the current estimate.

$$\begin{aligned} -2 \log p^I_{\mathbf{n}}(y) &= \big(f(\breve{y}_i) + f'(\breve{y})(y - \breve{y}) - z\big)^T C^{-1} \big(f(\breve{y}) + f'(\breve{y})(y - \breve{y}) - z\big) \\ &= y^T A y + y^T b + \text{const, with} \\ A = f'(\breve{y})^T C^{-1} f'(\breve{y}), &\quad b = 2 f'(\breve{y})^T C^{-1} \big(f(\breve{y}_i) - f'(\breve{y})\breve{y} - z\big) \end{aligned} \tag{76}$$

SLAM constraints are rotation invariant (except initial robot pose)

$$f(\mathrm{R}_\alpha \breve{y}) = f(\breve{y}) \quad \forall \breve{y}, \alpha \tag{77}$$

where $\mathrm{R}_\alpha(y)$ is a rotation matrix with $\left(\begin{smallmatrix} \cos\phi & -\sin\phi \\ \sin\phi & \cos\phi \end{smallmatrix}\right)$ blocks for each landmark. We take the derivative on both sides

$$\big(f(\mathrm{R}_\alpha \breve{y})\big)' = f'(\mathrm{R}_\alpha \breve{y}) \cdot \mathrm{R}_\alpha = f'(\breve{y}) \tag{78}$$

$$f'(\mathrm{R}_\alpha \breve{y}) = f'(\breve{y}) \cdot \mathrm{R}_{-\alpha} \,. \tag{79}$$

So by post multiplying the Jacobian f' with a rotation matrix $\mathrm{R}_{-\alpha}$ we can get the Jacobian at a rotated linearization point $\breve{y}' = Rot_\alpha \breve{y}$. If we derive the Gaussian $p^I_{\mathbf{n}}$ by (76) using $\breve{y}'$, we get

$$A' = f'(\breve{y}')^T C^{-1} f'(\breve{y}') = f'(\mathrm{R}_\alpha \breve{y})^T C^{-1} f'(\mathrm{R}_\alpha \breve{y}) \tag{80}$$

$$\overset{(79)}{=} \mathrm{R}^T_{-\alpha} f'(\breve{y})^T C^{-1} f'(\breve{y}) \,\mathrm{R}_{-\alpha} = \mathrm{R}^T_{-\alpha} A \,\mathrm{R}_{-\alpha} \tag{81}$$

$$b' = 2 f'(\breve{y}')^T C^{-1} \big(f(\breve{y}') - f'(\breve{y}')\breve{y}' - z\big) \tag{82}$$

$$= 2 f'(\mathrm{R}_\alpha \breve{y})^T C^{-1} \big(f(\mathrm{R}_\alpha \breve{y}_i) - f'(\mathrm{R}_\alpha \breve{y}) \,\mathrm{R}_\alpha \breve{y} - z\big) \tag{83}$$

$$\overset{(79)}{=} 2 \,\mathrm{R}^T_{-\alpha} f'(\breve{y})^T C^{-1} \big(f(\breve{y}_i) - f'(\breve{y}) \,\mathrm{R}_{-\alpha} \mathrm{R}_\alpha \breve{y} - z\big) = \mathrm{R}^T_{-\alpha} b. \tag{84}$$

So by multiplying A and b with $\mathrm{R}_{-\alpha}$ we get the Gaussian corresponding to $\breve{y}' = \mathrm{R}_\alpha \, \breve{y}$ without recomputing Jacobians. By (24) and (30) the factor propagates through multiplication ($\odot$) and marginalization (Ⓜ). So by rotating $p^M_{\mathbf{n}}$ of an inner node $\mathbf{n}$ we can rotate the linearization point of *all* constraints below $\mathbf{n}$ without recomputing anything.

When the tree's input distributions are derived from the preprocessing EKF, linearization is implicitly done by the EKF. So we still can apply the rotation in the tree the same way.

The conditionals $p^C_{\mathbf{n}}$ are represented by P^{-1}, H, h. We substitute (81) and (84) into (27) and get the rotation formula

$$P'^{-1} = \mathrm{R}^T_{-\alpha} \, P^{-1} \, \mathrm{R}_{-\alpha}, \tag{85}$$

$$H' = -\,\mathrm{R}^T_{-\alpha} \, P^{-1} \, \mathrm{R}_{-\alpha} \, \mathrm{R}^T_{-\alpha} \, R \, \mathrm{R}_{-\alpha} = \mathrm{R}^T_{-\alpha} \, H \, \mathrm{R}_{-\alpha}, \tag{86}$$

$$h' = -\,\mathrm{R}^T_{-\alpha} \, P^{-1} \, \mathrm{R}_{-\alpha} \, \mathrm{R}^T_{-\alpha} \, c = \mathrm{R}^T_{-\alpha} \, h. \tag{87}$$

Finally α (as well as a translation d) is determined by minimizing $\sum_{\mathsf{l}} w_{\mathsf{l}} \big((\mathrm{R}_\alpha \, \breve{y} + d) - \hat{y} \big)^2$. Lu and Milios (1997) give an analytical solution

$$\alpha = \operatorname{atan2}\left(S_{12} - S_{21}, S_{11} - S_{22} \right), \tag{88}$$

$$S = \Big(\sum_{\mathsf{l}} w_{\mathsf{l}} \Big) \Big(\sum_{\mathsf{l}} w_{\mathsf{l}} \hat{y}_{\mathsf{l}} \breve{y}_{\mathsf{l}}^T \Big) - \Big(\sum_{\mathsf{l}} w_{\mathsf{l}} \hat{y}_{\mathsf{l}} \Big) \Big(\sum_{\mathsf{l}} w_{\mathsf{l}} \breve{y}_{\mathsf{l}} \Big)^T, \tag{89}$$

where $\hat{y}_{\mathsf{l}}$ and $\breve{y}_{\mathsf{l}}$ denote the coordinates for landmark l in $\hat{y}$ and $\breve{y}$ respectively. The weight w_{l} is defined as $w_{\mathsf{l}} = \operatorname{tr}(A_{\mathsf{ll}})$ trace in the corresponding block in the information matrix A.

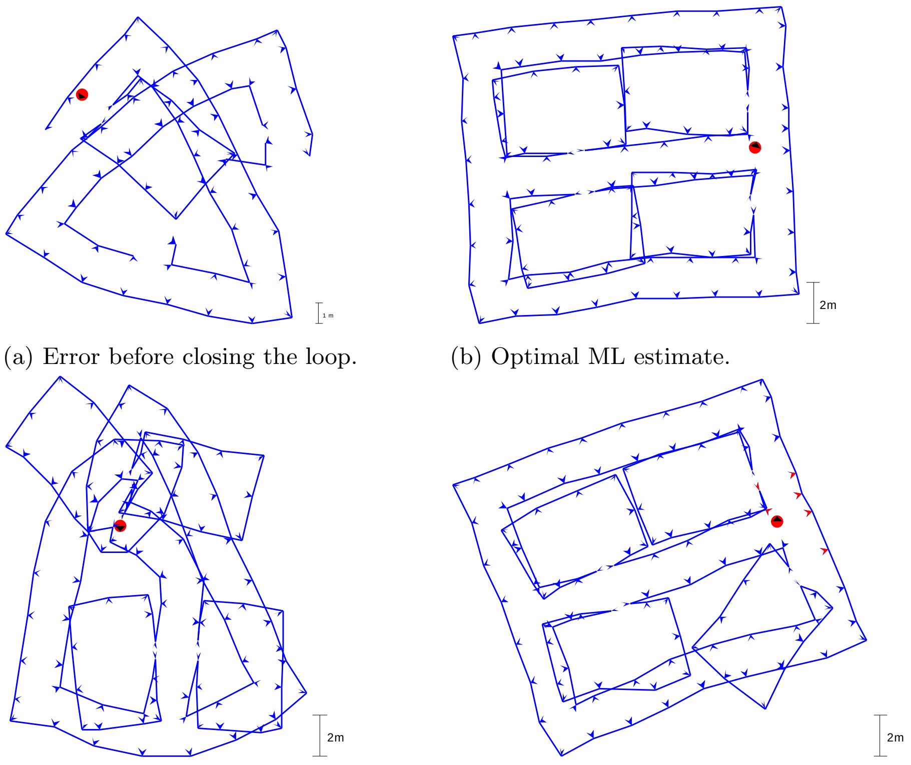
11.3 Large Noise Experiment

This experiment evaluates treemap's nonlinear extension with a large accumulated orientation error (140°, Fig. 17a). Figures 17b-d show the ML, EKF and treemap estimate. The EKF estimate is clearly unusable. It appears to be even topologically inconsistent despite perfect data association. To see what happens, let us try to minimize $\left| \begin{pmatrix} -1-\cos\alpha \\ 0-\sin\alpha \end{pmatrix} \right|$. The nonlinear minimum is certainly 180°, but when we linearize at $\breve{\alpha} = 0$, we get $\left| \begin{pmatrix} -1 \\ -\alpha \end{pmatrix} \right|$ and a minimum at $\alpha = 0$. This roughly happens in figure 17c. The area around the robot is barely rotated although it should be but extremely distorted instead.

In contrast to that, the treemap estimate appears to be worse than the ML estimate but still reasonable, showing that, with the nonlinear extension, treemap works well, especially compared to the EKF.

12 Conclusion

The treemap SLAM algorithm proposed in this article works by dividing the map into a hierarchy of regions represented as a binary tree.



(a) Error before closing the loop.

(b) Optimal ML estimate.

(c) EKF estimate.

(d) Treemap estimate.

Figure 17: Large noise simulation experiment results. The estimate of treemap with nonlinear extension is still okay but the EKF estimate is clearly unusable.

With this data structure, the computations necessary for integrating a measurement are limited essentially to updating a leaf of the tree and all its ancestors up to the root. From a theoretical perspective the main advantage is that a local map can be computed in $O(k^3 \log n)$ time. In practice, it is equally important that a global map can be computed in $O(kn)$ time allowing the update of a map with $n = 11300$ landmarks in 12.37ms on an Intel Xeon, 2.67 GHz. Treemap is exact up to linearization and sparsification where some information on the landmarks is sacrificed to pass information on the robot pose between regions. Still despite the sparsification, if two landmarks are observed together, the fact that we know their precise relative location will be reflected by the estimate after the next update.

With respect to the three criteria (R1)-(R3) proposed in Section 1, the algorithm was verified theoretically, by simulation experiments, and by experiments with a real robot. There are two preconditions for achieving these results. a) The environment must be topologically suitable, i.e. have a hierarchical partitioning and b) the HTP subalgorithm must find one as explained in Section 4. This is indeed a major drawback since it precludes mapping dense outdoor environments. The second drawback is that performing the bookkeeping in $O(k^3 \log n)$ significantly complicates the algorithm. Consequently we are currently working on a simplified algorithm (Frese and Schröder, 2006). We plan to generalize it so it can handle different SLAM variants such as 2D-, 3D-, landmark-, pose-, and bearing-only-SLAM and publish the code as an open source implementation.

A major future challenge will be uncertain data-association. Some authors address this issue with a framework that evaluates the likelihood of several data association hypotheses (Hähnel et al., 2003). Regardless of how the overall framework is conceived, it needs a classical SLAM algorithm as the core engine to evaluate a single hypothesis. Efficiency is then even more crucial because updates must be performed for each of the hypothesis considered.

Acknowledgments

I would like to thank Bernd Krieg-Brückner, Robert Ross, Matthew Walter, and the anonymous reviewers for valuable comments.

References

Bosse, M., P. Newman, J. Leonard, and S. Teller: 2004, 'SLAM in Large-scale Cyclic Environments using the Atlas Framework'. *International Journal on Robotics Research* **23**(12), 1113–1140.

Duckett, T., S. Marsland, and J. Shapiro: 2000, 'Learning Globally Consistent Maps by Relaxation'. In: *Proceedings of the IEEE International Conference on Robotics and Automation, San Francisco.* pp. 3841–3846.

Duckett, T., S. Marsland, and J. Shapiro: 2002, 'Fast, On-line Learning of Globally Consistent Maps'. *Autonomous Robots* **12**(3), 287 – 300.

Eliazar, A. and R. Parr: 2003, 'DP-SLAM: Fast, Robust Simulataneous Localization and Mapping without Predetermined Landmarks'. In: *Proceedings of the International Joint Conference on Artificial Intelligence, Acapulco.* pp. 1135–1142.

Estrada, C., J. Neira, and J. Tardós: 2005, 'Hierarchical SLAM: Real-Time Accurate Mapping of Large Environments'. *IEEE Transactions on Robotics* **21**(4), 588–596.

Eustice, R., M. Walter, and J. Leonard: 2005, 'Sparse Extended Information Filters: Insights into Sparsification'. In: *Proceedings of the International Conference on Intelligent Robots and Systems, Edmonton.*

Fiduccia, C. and R. Mattheyses: 1982, 'A linear-time heuristic for improving network partitions'. In: *Proceedings of the 19th ACM/IEEE Design Automation Conference, Las Vegas.* pp. 175–181.

Frese, U.: 2004, 'An $O(\log n)$ Algorithm for Simulateneous Localization and Mapping of Mobile Robots in Indoor Environments'. Ph.D. thesis, University of Erlangen-Nürnberg. `http://www.opus.ub.uni-erlangen.de/opus/volltexte/2004/70/`.

Frese, U.: 2005, 'A Proof for the Approximate Sparsity of SLAM Information Matrices'. In: *Proceedings of the IEEE International Conference on Robotics and Automation, Barcelona.* pp. 331–337.

Frese, U.: 2006a, 'A Discussion of Simultaneous Localization and Mapping'. *Autonomous Robots* **20**(1), 25–42.

Frese, U.: 2006b, 'Treemap: An $O(\log n)$ Algorithm for Indoor Simultaneous Localization and Mapping'. *Autonomus Robots.* to appear.

Frese, U. and G. Hirzinger: 2001, 'Simultaneous Localization and Mapping - A Discussion'. In: *Proceedings of the IJCAI Workshop on Reasoning with Uncertainty in Robotics, Seattle.* pp. 17 – 26.

Frese, U., P. Larsson, and T. Duckett: 2004, 'A Multigrid Algorithm for Simultaneous Localization and Mapping'. *IEEE Transactions on Robotics* **21**(2), 1–12.

Frese, U. and L. Schröder: 2006, 'Closing a Million-Landmarks Loop'. In: *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems, Beijing.* submitted.

Gauss, C.: 1821, 'Theoria combinationis observationum erroribus minimis obnoxiae'. *Commentationes societatis regiae scientiarum Gottingensis recentiores* **5**, 6–93.

Guivant, J. and E. Nebot: 2001, 'Optimization of the Simultaneous Localization and Map-Building Algorithm for Real-Time Implementation'. *IEEE Transactions on Robotics and Automation* **17**(3), 242 – 257.

Guivant, J. and E. Nebot: 2003, 'Solving computational and memory requirements of feature-based simultaneous localization and mapping algorithms'. *IEEE Transactions on Robotics and Automation* **19**(4), 749–755.

Hähnel, D., W. Burgard, B. Wegbreit, and S. Thrun: 2003, 'Towards lazy data association in SLAM'. In: *In Proceedings of the 10th International Symposium of Robotics Research*.

Hendrickson, B. and R. Leland: 1995, 'A Multilevel Algorithm for Partitioning Graphs'. In: *Proceedings of the ACM International Conference on Supercomputing, Sorrento*. pp. 626–657.

Horn, R. and C. Johnson: 1990, *Matrix Analysis*. Cambridge University Press.

Leonard, J. and H. Feder: 2001, 'Decoupled Stochastic Mapping'. *IEEE Journal of Ocean Engineering* **26**(4), 561 – 571.

Lu, F. and E. Milios: 1997, 'Globally Consistent Range Scan Alignment for Environment Mapping'. *Autonomous Robots* **4**(4), 333 – 349.

Montemerlo, M., S. Thrun, D. Koller, and B. Wegbreit: 2002, 'FastSLAM: A Factored Solution to the Simultaneous Localization and Mapping Problem'. In: *Proceedings of the AAAI National Conference on Artificial Intelligence, Edmonton*. pp. 593–598.

Nieto, J., J. Guivant, E. Nebot, and S. Thrun: 2003, 'Real Time Data Association for fastSLAM'. In: *Proceedings of the IEEE Conference on Robotics and Autonomation, Taipeh*. pp. 412–418.

Otsu, N.: 1979, 'A Threshold Selection Method from Gray-Level Histograms'. *IEEE Transactions on Systems, Man and Cybernetics* **9**(1), 62 – 66.

Paskin, M.: 2003, 'Thin Junction Tree Filters for Simultaneous Localization and Mapping'. In: *Proceedings of the 18th International Joint Conference on Artificial Intelligence*. San Francisco, pp. 1157–1164.

Press, W., S. Teukolsky, W. Vetterling, and B. Flannery: 1992, *Numerical Recipes, Second Edition*. Cambridge University Press, Cambridge.

Smith, R., M. Self, and P. Cheeseman: 1988, 'Estimating Uncertain Spatial Relationships in Robotics'. In: I. Cox and G. Wilfong (eds.): *Autonomous Robot Vehicles*. Springer Verlag, New York, pp. 167 – 193.

Stachniss, C. and W. Burgard: 2004, 'Exploration with Active Loop-Closing for FastSLAM'. In: *In Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*. pp. 1505–1510.

Thrun, S., W. Burgard, and D. Fox: 2005, *Probabilistic Robotics*. MIT Press.

Thrun, S. and Y. Liu: 2003, 'Multi-Robot SLAM With Sparse Extended Information Filers'. In: *Proceedings of the 11th International Symposium of Robotics Research, Sienna.*

Thrun, S., Y. Liu, D. Koller, A. Ng, Z. Ghahramani, and H. Durrant-Whyte: 2004, 'Simultaneous Localization and Mapping With Sparse Extended Information Filters'. *International Journal of Robotics Research* **23**(7–8), 613–716.

Vijayan, G.: 1991, 'Generalization of Min-Cut Partitioning to Tree Structures and Its Applications'. *IEEE Transactions on Computers* **40**(3), 307 – 314.

Walter, M., R. Eustice, and J. Leonard: 2005, 'A Provably Consistent Method for Imposing Exact Sparsity in Feature-based SLAM Information Filters'. In: *Proceedings of the 12th International Symposium of Robotics Research.*

A Pseudocode

This sections lists the overall treemap algorithm in detail as pseudocode. The main routine is `observation`$\big(z, C_z, \mathcal{L}_z\big)$ integrating observations z of landmarks $\mathcal{L}_z$ with covariance C_z. Odometry is directly handled by the preprocessing EKF (Smith et al., 1988, Eqn. (12)). Subroutines are printed in order of appearance. Each node $\mathbf{n}$ stores

- Input distribution $p^I_{\mathbf{n}}$ in information form as $A^I_{\mathbf{n}}, b^I_{\mathbf{n}}, \mathcal{L}^I_{\mathbf{n}}$ (leaves).
- Marginal distribution $p^M_{\mathbf{n}}$ in information form as $A^M_{\mathbf{n}}, b^M_{\mathbf{n}}, \mathcal{L}^M_{\mathbf{n}}$.
- Conditional distribution $p^C_{\mathbf{n}}$ as $P^{C-1}_{\mathbf{n}}, H^C_{\mathbf{n}}, h^C_{\mathbf{n}}, \mathcal{L}^C_{\mathbf{n}}$.
- Number of leaves $\mathbf{n}_{size}$ below that node.
- Last estimate computed $\hat{x}_{\mathbf{n}}, C_{\mathbf{n}}$.

Additionally, treemap globally stores

- EKF state in covariance form $\hat{x}_{\mathrm{EKF}}, C_{\mathrm{EKF}}, \mathcal{L}_{\mathrm{EKF}}$.
- Global set of landmarks $\mathcal{L}$ with counters for the number of leaves involving a landmark (as an array indexed by landmark).
- Global array **lca** maintaining the least common ancestor of all nodes involving a landmark.
- Pointer **root** to the root node and **c** to the current leaf.

A.1 `observation` $\left(z, C_z, \mathcal{L}_z\right)$ $O(k^2|\mathcal{L}_z|), O(k^3 \log n), O(kn)$ ***resp.***

Integrates a vector z of measurements with covariance C_z into the treemap. Maybe changes the current leaf (Fig. 8). Then integrates z into the preprocessing EKF.

```
IF   isTooLarge ($\hat{x}_{\mathrm{EKF}}, \mathcal{L}_{\mathrm{EKF}}, z, \mathcal{L}_z$)  $\vee$  $\mathcal{L}_z \cap (\mathcal{L} \setminus \mathcal{L}_{\mathrm{EKF}}) \neq \emptyset$
THEN newLeaf := findOrCreateLeave ($z$)
     $((A, b), (P^I, H, h))$ := extractFromEKF ($\hat{x}_{\mathrm{EKF}}, C_{\mathrm{EKF}}, \mathcal{L}_z$)
     multiplyIntoLeaf ($\mathbf{c}, (A, b, \mathcal{L}_{\mathrm{EKF}})$)
     FOR optHTPSteps times DO optimizeHTP ()
     c := newLeaf
     update (c)
     IF desired THEN computeGlobalEstimate (root)
     $(\hat{x}, C)$ := compileEKF $\left(\mathbf{c}, (P^I, H, h)\right)$
Integrate $z, C_z$ into the EKF state $\hat{x}_{\mathrm{EKF}}, C_{\mathrm{EKF}}$ (Smith et al., 1988, Eqn. (16))
```

A.2 `isTooLarge` $\left(\hat{x}, \mathcal{L}_x, z, \mathcal{L}_z\right)$ $O(k^2)$

Checks, whether adding all landmarks in z to a region would violate the region's maximum diameter *maxD*. $\hat{x}$ is an estimate for the considered region.

```
Compute transform $T$ from $\hat{x}_{\mathrm{EKF}}$ to $\hat{x}$ based on the landmarks in $\mathcal{L}_x \cap \mathcal{L}_{\mathrm{EKF}}$
$z'$ := Transform $z$ to global coordinates using robot pose from $\hat{x}_{\mathrm{EKF}}$ and $T$
FOR All landmarks $l_1 \in \mathcal{L}_z$
    FOR All landmarks $l_2 \in \mathcal{L}_x$
        IF distance $d$ from $l_1$ in $z'$ to $l_2$ in $\hat{x} > maxD$ THEN return true
return false
```

A.3 `findOrCreateLeave` $\left(z, \mathcal{L}_z\right)$ $O(k^3 + k^2 \log n)$

Finds a leave in the tree that is appropriate for integrating measurements z into. If none can be found creates a new leaf.

```
Compute set of landmarks $\mathcal{O}$ in the robot's field of view from $\hat{x}_{\mathrm{EKF}}$
leaves := $\bigcup_{\mathsf{l} \in \mathcal{L}_{\mathrm{EKF}}}$ allLeavesInvolving ($\mathsf{l}$, root)
$\mathcal{M} := \mathcal{L}_z \cup \mathcal{O}$;  best := $|\mathcal{M}|$;  bestN := nil
FOR All nodes $\mathbf{n} \in$ leaves
    IF   $|\mathcal{L}_{\mathrm{EKF}} \cap \mathcal{L}^I_{\mathbf{n}}| \geq 2$ $\wedge$ $\neg$isTooLarge ($\hat{x}_{\mathbf{n}}, z$)
    THEN IF $|\mathcal{M} \setminus \mathcal{L}^I_{\mathbf{n}}| < best$ THEN best := $|\mathcal{M} \setminus \mathcal{L}^I_{\mathbf{n}}|$;  bestN := n
IF   bestN $\neq$ nil
THEN Extend $(A^I_{\mathbf{bestN}}, b^I_{\mathbf{bestN}}, \mathcal{L}_{\mathbf{bestN}})$ with all landmarks from $\mathcal{M} \cap \mathcal{L}$.
ELSE IF $|\mathcal{M} \cap \mathcal{L}_{\mathrm{EKF}}| < 2$ THEN $\mathcal{M} := \mathcal{M} \cup$ {last observed landmark}
     IF $|\mathcal{M} \cap \mathcal{L}_{\mathrm{EKF}}| < 2$ THEN $\mathcal{M} := \mathcal{M} \cup$ {second last observed landmark}
     bestN := createEmptyLeave ($\mathcal{M} \cap \mathcal{L}$)
return bestN
```

A.4 `allLeavesInvolving` $(\mathsf{l}, \mathbf{n})$ $O(k \log n)$

Returns the set of all leaves below $\mathbf{n}$ that involve landmark l. $\mathbf{n}$ is below $\mathbf{lca}[\mathsf{l}]$.

```
IF $\mathbf{n}$ is leaf $\wedge$ $\mathsf{l} \in \mathcal{L}^I_{\mathbf{n}}$ THEN return $\{\mathbf{n}\}$
IF $\mathbf{n} \neq \mathbf{lca}[\mathsf{l}]$ $\wedge$ $\mathsf{l} \notin \mathcal{L}^M_{\mathbf{n}}$ THEN return $\emptyset$
return allLeavesInvolving $(\mathsf{l}, \mathbf{n}_\swarrow)$ $\cup$ allLeavesInvolving $(\mathsf{l}, \mathbf{n}_\searrow)$
```

A.5 `createEmptyLeaf` $(\mathcal{M})$ $O(k \log n)$

Creates a new leaf that involves landmarks $\mathcal{M}$ but still has 0 information and put it at a good position in the tree.

```
$\mathbf{b}$ := new empty leaf; $\mathcal{L}^I_{\mathbf{b}} := \mathcal{M}$; $A^I_{\mathbf{b}} = 0$; $b^I_{\mathbf{b}} = 0$
$\mathbf{root}$ := new node with $\mathbf{root}$ as left child and $\mathbf{b}$ as right child
$\mathcal{L} := \mathcal{L} \uplus \mathcal{M}$
FOR $\mathsf{l} \in \mathcal{M}$ DO
    invalidate $(\mathbf{lca}[\mathsf{l}])$
updateLandmarkLists $(\mathbf{root})$
$\mathbf{a}$:= findBestTransferTo $(\mathbf{b}, \mathbf{root}_\swarrow)$
transferSubtree $(\mathbf{b}, \mathbf{a})$
```

A.6 `invalidate` $(\mathbf{n})$ $O(1)$ ***per node invalidated***

Marks all nodes from $\mathbf{n}$ to the root invalid.

```
WHILE $\mathbf{n} \neq \mathbf{nil} \wedge \mathbf{n}$ marked valid DO
    mark $\mathbf{n}$ invalid; $\mathbf{n} := \mathbf{n}_\uparrow$
```

A.7 `updateLandmarkLists` $(\mathbf{n})$ $O(k)$ ***per node updated***

Updates $\mathcal{L}^M_{\mathbf{n}}$, $\mathcal{L}^C_{\mathbf{n}}$ and $\mathbf{n}_{size}$ of node $\mathbf{n}$ and all descendents recursively. Sets $\mathbf{lca}[\mathsf{l}]$ for all landmarks l marginalized out in updated nodes.

```
IF $\mathbf{n}$ is not marked *invalid* THEN return
IF $\mathbf{n}$ is no leaf
THEN
    updateLandmarkLists $(\mathbf{n}_\swarrow)$;   updateLandmarkLists $(\mathbf{n}_\searrow)$
    $\mathcal{L}^M_{\mathbf{n}} := \mathcal{L}^M_{\mathbf{n}_\swarrow} \uplus \mathcal{L}^M_{\mathbf{n}_\searrow}$;   $\mathbf{n}_{size} := \mathbf{n}_{\swarrow size} + \mathbf{n}_{\searrow size}$
ELSE
    $\mathcal{L}^M_{\mathbf{n}} := \mathcal{L}^I_{\mathbf{n}}$;   $\mathbf{n}_{size} := 1$
$\mathcal{L}^C_{\mathbf{n}} := \{\mathsf{l} \in \mathcal{L}^M_{\mathbf{n}} \mid \mathsf{l}\#\mathcal{L}^M_{\mathbf{n}} = \mathsf{l}\#\mathcal{L}\}$;   $\mathcal{L}^M_{\mathbf{n}} := \mathcal{L}^M_{\mathbf{n}} \setminus \mathcal{L}^C_{\mathbf{n}}$
FOR $\mathsf{l} \in \mathcal{L}^C_{\mathbf{n}}$ DO
    $\mathbf{lca}[\mathsf{l}] := \mathbf{n}$
Mark $\mathbf{n}$ as *landmark list valid.*
```

A.8 `findBestTransferTo` $(\mathbf{s}, \mathbf{n})$ $O(k \log n)$

Finds the best node below **n** above which to move **s**.

```
WHILE $\mathbf{n}$ is not leaf
    Compute $\text{par}_{\uparrow,\swarrow,\searrow}$ by (72), (73), (74).
    IF    $\mathbf{s}_{size} \geq \mathbf{n}_{size}/2 \;\wedge\; \text{par}_{\uparrow} < \text{par}_{\swarrow} \;\wedge\; \text{par}_{\uparrow} < \text{par}_{\searrow}$
    THEN  return n
    ELSE  IF $\text{par}_{\swarrow} < \text{par}_{\searrow}$ THEN $\mathbf{n} := \mathbf{n}_{\swarrow}$
return $\mathbf{n}$
```

A.9 `transferSubtree` $(\mathbf{s},\ \mathbf{a})$ $O(\log n)$

Moves **s** and the subtree below to above **a**.

```
invalidate $(\mathbf{s}_{\uparrow})$
invalidate $(\mathbf{a}_{\uparrow})$
Change pointers, so $\mathbf{s}_{\uparrow}$ becomes parent of $\mathbf{a}$.
```

A.10 `extractFromEKF` $(\hat{x}_{\mathbf{EKF}}, C_{\mathbf{EKF}}, \mathcal{N})$ $O(k^3)$

Converts the EKF state $\hat{x}_{\text{EKF}}, C_{\text{EKF}}$ into information form and separates the landmark information from the robot pose information by sparsification (Ⓢ) and marginalization (Ⓜ) (Fig. 8). $\mathcal{N}$ is the set of landmarks involved in the new leaf.

```
$A := C_{\text{EKF}}^{-1} - C_{\mathbf{c}}^{-1}; \quad b := -2C_{\text{EKF}}^{-1}\hat{x}_{\text{EKF}} - 2C_{\mathbf{c}}^{-1}\hat{x}_{\mathbf{c}}$
Permute $A$ as $\begin{pmatrix} * & * & * \\ * & * & * \\ * & * & * \end{pmatrix}$, $b$ as $\begin{pmatrix} * \\ * \\ * \end{pmatrix}$, with block rows / columns:
robot pose, landmarks $\notin \mathcal{N}$, landmarks $\in \mathcal{N}$
$(A, b) :=$ sparsify $(A, b, \hat{x}, \mathcal{N})$. Now $A = \begin{pmatrix} * & 0 & * \\ 0 & * & * \\ * & * & * \end{pmatrix}$, $b = \begin{pmatrix} * \\ * \\ * \end{pmatrix}$.
Group block row / column 2 and 3 as one block. $A = \begin{pmatrix} * & \begin{pmatrix} 0 & * \end{pmatrix} \\ \begin{pmatrix} 0 \\ * \end{pmatrix} & \begin{pmatrix} * & * \\ * & * \end{pmatrix} \end{pmatrix}$, $b = \begin{pmatrix} * \\ \begin{pmatrix} * \\ * \end{pmatrix} \end{pmatrix}$.
Compute $A_{\text{EKF}}^{M}, b_{\text{EKF}}^{M}$ by (30) and $P_{\text{EKF}}^{C-1}, H_{\text{EKF}}^{C}, h_{\text{EKF}}^{C}$ by (27).
Now $A_{\text{EKF}}^{M} = \begin{pmatrix} * & * \\ * & * \end{pmatrix}, b_{\text{EKF}}^{M} = \begin{pmatrix} * \\ * \end{pmatrix}$, $P_{\text{EKF}}^{C-1} = \begin{pmatrix} * \end{pmatrix}, H_{\text{EKF}}^{C} = \begin{pmatrix} 0 & * \end{pmatrix}, h_{\text{EKF}}^{C} = \begin{pmatrix} * \end{pmatrix}$
Remove first block columns from $H_{\text{EKF}}^{C}$ since it is 0.
return $\left((A_{\text{EKF}}^{M}, b_{\text{EKF}}^{M}), (P_{\text{EKF}}^{C-1}, H_{\text{EKF}}^{C}, h_{\text{EKF}}^{C})\right)$
```

A.11 `sparsify` $(A, b, \hat{x}, \mathcal{N})$ $O(k^3)$

Removes the off-diagonal block A_{21} between robot pose (block row / column 1, dimension 3) and landmarks $\notin \mathcal{N}$ (block row / column 2)

$$\text{of } A = \begin{pmatrix} * & * & * \\ * & * & * \\ * & * & * \end{pmatrix} \text{ by returning a matrix } A' = \begin{pmatrix} * & 0 & * \\ 0 & * & * \\ * & * & * \end{pmatrix} \text{ with } 0 \leq A' \leq A.$$

Information vector b is updated so the mean $\hat{x}$ is preserved.

```
$A^{\text{orig}} := A$
FOR i=1...3
    Permute $A$: block row / col. 1: r. pose component $i$, 2: landmarks $\notin \mathcal{N}$, 3: others
    Compute $u_i$ by theorem 1 (62).
    $A := A - u_i u_i^T$;    $A := A - A_{\bullet 1} A_{11}^{-1} A_{\bullet 1}^T$
Permute $u_1, u_2, u_3$ back so rows correspond to $A$
return $\left(A^{\text{orig}} - u_1 u_1^T - u_2 u_2^T - u_3 u_3^T, b - 2(u_1 u_1^T + u_2 u_2^T + u_3 u_3^T)\hat{x}\right)$
```

A.12 `multiplyIntoLeaf` $\left(\mathbf{n}, A', b', \mathcal{L}'\right)$ $O(k^3 \log n)$

Multiplies the Gaussian defined by A', b' into the input distribution at leaf $\mathbf{n}$.

```
Permute and extend $A', b', \mathcal{L}'$ and $A^I_{\mathbf{n}}, b^I_{\mathbf{n}}, \mathcal{L}^I_{\mathbf{n}}$ so corresponding rows and columns represent the same landmark.
$A^I_{\mathbf{n}} := A^I_{\mathbf{n}} + A'$;    $b^I_{\mathbf{n}} := b^I_{\mathbf{n}} + b'$
invalidate ($\mathbf{n}$)
FOR All landmarks $\mathsf{l} \in \mathcal{L}'$ DO invalidate ($\mathbf{lca}[\mathsf{l}]$)
```

A.13 `optimizeHTP` $()$ $O(k^3 \log n)$

Picks a node and tries to reduce the number of landmark involved there by optimally moving a subtree from left below to right below or vice versa.

```
updateLandmarkLists ($\mathbf{root}$)
$\mathbf{r}$ := findNodeToBeOptimized ()
IF $\mathbf{r} = \mathbf{nil}$ THEN return
$\mathbf{s}$ := findBestTransferFrom ($\mathbf{r}$)
IF $\mathbf{s} = \mathbf{nil}$ THEN Mark $\mathbf{r}$ as not *to be optimized*;    return
$\mathbf{a}$ := findBestTransferTo ($\mathbf{s}, \mathbf{r}_?$), with $? \in \{\swarrow, \searrow\}$ =child that is not ancestor of $\mathbf{s}$.
transferSubtree ($\mathbf{s}, \mathbf{a}$)
```

A.14 `findNodeToBeOptimized` $()$ $O(\log n)$

Finds the next node to be optimized. It descends along *to be optimized* marks down the tree. If there is a choice a node not being ancestor of the current leaf is preferred.

```
$\mathbf{r} := \mathbf{root}$;    $\mathbf{p}$ := path from $\mathbf{root}$ to $\mathbf{c}$
IF $\mathbf{r}$ is marked *to be optimized*
THEN
    WHILE $\mathbf{r}_\swarrow$ or $\mathbf{r}_\searrow$ are marked *to be optimized*
        Remove first element from $\mathbf{p}$
        IF $\mathbf{r}_\swarrow$ and $\mathbf{r}_\searrow$ are marked *to be optimized*
        THEN IF $\mathbf{r}_\swarrow = \mathbf{p}[0]$ THEN $\mathbf{r} := \mathbf{r}_\searrow$
        ELSE IF $\mathbf{r}_\swarrow$ is marked *to be optimized* THEN $\mathbf{r} := \mathbf{r}_\swarrow$
    return $\mathbf{r}$
ELSE
    return $\mathbf{nil}$
```

A.15 `findBestTransferFrom` $(\mathbf{r})$ $O(k^2 \log n)$

Finds the optimal node below $\mathbf{r}$ to move from the left to the right side or vice versa.

```
$s_{low} := \lceil \mathbf{r}_{\swarrow size} - \frac{2}{3}\mathbf{r}_{size} \rceil$;   $s_{high} := \lfloor \mathbf{r}_{\swarrow size} - \frac{1}{3}\mathbf{r}_{size} \rfloor$
IF $\frac{1}{3}\mathbf{r}_{size} \le \mathbf{r}_{\swarrow size} \le \frac{2}{3}\mathbf{r}_{size}$
THEN $\mathbf{best} := \mathbf{nil}$;   $bestValue := \left|\mathcal{L}^M_{\mathbf{r}_\swarrow}\right| + \left|\mathcal{L}^M_{\mathbf{r}_\searrow}\right|$
ELSE $bestValue := \infty$
IF $s_{high} > 0$ THEN recursiveBest $(\mathbf{r}_\swarrow, \mathbf{r}, s_{low}, s_{high}, \mathbf{best}, bestValue)$
IF $s_{low} < 0$ THEN recursiveBest $(\mathbf{r}_\searrow, \mathbf{r}, -s_{high}, -s_{low}, \mathbf{best}, bestValue)$
return $\mathbf{best}$
```

A.16 `recursiveBest` $(\mathbf{s}, \mathbf{r}, s_{low}, s_{high}, \mathbf{best}, bVal)$ $O(k^2)$ *per node*

Recursively searches through all descendants of $\mathbf{s}$ that share a landmark with $\mathbf{r}$. Looks, whether there is a node $\mathbf{s}$ that, when moving it to the other side of $\mathbf{r}$ reduces par($\mathbf{r}$) below $bVal$. If there is, it replaces **best** and the corresponding par($\mathbf{r}$) value replaces $bVal$. It considers only nodes with size in $s_{low} \ldots s_{high}$.

```
IF $\mathcal{L}^M_{\mathbf{s}} \cap (\mathcal{L}^M_{\mathbf{r}_\swarrow} \cup \mathcal{L}^M_{\mathbf{r}_\searrow}) = \emptyset$ THEN return
Compute par$'$ by (70)
IF par$' < bVal \ \wedge\ \mathbf{s}_{size} \in [s_{low} \ldots s_{high}]$
THEN $\mathbf{best} := \mathbf{s}$;   $bVal :=$ par
IF $\mathbf{s}$ is no leaf
THEN recursiveBest $(\mathbf{s}_\swarrow, \mathbf{r}, [s_{low} \ldots s_{high}], \mathbf{best}, bVal)$
     recursiveBest $(\mathbf{s}_\searrow, \mathbf{r}, [s_{low} \ldots s_{high}], \mathbf{best}, bVal)$
```

A.17 `update` $(\mathbf{n})$ $O(k^3)$ *per node updated*

Updates $A^M_{\mathbf{n}}$, $b^M_{\mathbf{n}}$, $P^{C-1}_{\mathbf{n}}$, $H^C_{\mathbf{n}}$, and $h^C_{\mathbf{n}}$ of node $\mathbf{n}$ and all descendents recursively.

```
IF $\mathbf{n}$ is not marked *invalid* or *landmark lists valid* THEN return
updateLandmarkLists $(\mathbf{n})$
IF $\mathbf{n}$ is no leaf
THEN update $(\mathbf{n}_\swarrow)$;   update $(\mathbf{n}_\searrow)$
     Permute $A^M_{\mathbf{n}_\swarrow}, b^M_{\mathbf{n}_\swarrow}$ and $A^M_{\mathbf{n}_\searrow}, b^M_{\mathbf{n}_\searrow}$ so rows / cols. corresp. to same landmarks.
     $A := A^M_{\mathbf{n}_\swarrow} + A^M_{\mathbf{n}_\searrow}$;   $b := b^M_{\mathbf{n}_\swarrow} + b^M_{\mathbf{n}_\searrow}$
ELSE $A := A^I_{\mathbf{n}}$;   $b := b^I_{\mathbf{n}}$
Permute $A, b$ so block row / col. 1 corresp. to $\mathcal{L}^M_{\mathbf{n}}$ and block row / col. 2 to $\mathcal{L}^C_{\mathbf{n}}$.
Compute $A^M_{\mathbf{n}}, b^M_{\mathbf{n}}$ by (30) and $P^{C-1}_{\mathbf{n}}, H^C_{\mathbf{n}}, h^C_{\mathbf{n}}$ by (27).
Mark node $\mathbf{n}$ as *valid*.
```

A.18 `compileEKF` $(\mathbf{c}, (P^{C-1}_{\mathbf{EKF}}, H^C_{\mathbf{EKF}}, h^C_{\mathbf{EKF}}))$ $O(k^3 \log n)$

Computes a new EKF state $\hat{x}, C$ representing the landmarks involved in region $\mathbf{c}$ and the robot pose information as passed by $(P^{C-1}_{\mathrm{EKF}}, H^C_{\mathrm{EKF}}, h^C_{\mathrm{EKF}})$.

`computeEstimateWithCovariance` $(\mathbf{c})$
Compute $\hat{x}$ by (31) and C by (33) with $\hat{v} = \hat{x}_\mathbf{c}$ and $cov(v) = C_\mathbf{c}$.
return $(\hat{x}, C)$

A.19 `computeEstimateWithCovariance` $\big(\mathbf{n}\big)$

Computes the estimate $\hat{x}_\mathbf{n}$ with covariance $C_\mathbf{n}$ for node $\mathbf{n}$.

IF	$\mathbf{n} = \mathbf{nil}$	THEN	return $\big((), ()\big)$
$(\hat{v}, C_v) :=$ `computeEstimateWithCovariance` $(\mathbf{n}_\uparrow)$			
Remove all rows / cols. with landmarks $\notin \mathcal{L}^M_\mathbf{n}$ from v, C_v.			
Compute $\hat{x}_\mathbf{c}$ by (31) and $C_\mathbf{c}$ by (33) from $\hat{v}, C_v$ and $P^{C-1}_\mathbf{n}, H^C_\mathbf{n}, h^C_\mathbf{n}$.			

A.20 `computeGlobalEstimate` $\big(\mathbf{n}\big)$ $O(kn)$

Computes an estimate for all landmarks involved at $\mathbf{n}$ and below.

Get estimates $\hat{v}$ for landmarks $\in \mathcal{L}^M_\mathbf{n}$ from the global estimate $\hat{x}$ (Note: $\mathcal{L}^M_\mathbf{root} = \emptyset$).	
Compute $\hat{u}$ by (31) from $\hat{v}$ and $H^C_\mathbf{n}, h^C_\mathbf{n}$ and store in the global estimate $\hat{x}$.	
IF	$\mathbf{n}$ is not leaf
THEN	`computeGlobalEstimate` $(\mathbf{n}_\swarrow)$; `computeGlobalEstimate` $(\mathbf{n}_\searrow)$

B Example for Propagation of Distributions in the Tree

Figure 18 shows the computation at node $\mathbf{n}$ in the example from figure 2. Gaussians are denoted by $\mathcal{N}(\mu, C)$ in covariance form and by $\mathcal{N}^{-1}(b, A)$ in information form. On the left $z[\mathbf{n}_\swarrow\colon\downarrow] = z_{1,2}$, $X[\mathbf{n}_\swarrow\colon\downarrow] = X_{\mathsf{a},\mathsf{b}}$ but $X[\mathbf{n}_\swarrow\colon\downarrow\uparrow] = X_\mathsf{b}$. So $\mathbf{n}_\swarrow$ passes

$$p^M_{\mathbf{n}_\swarrow} = p(X_\mathsf{b}|z_{1,2}) = \mathcal{N}^{-1}(-\tfrac{2}{3}, \tfrac{1}{3}) = \mathcal{N}(1, 3) \tag{90}$$

to $\mathbf{n}$. On the right $z[\mathbf{n}_\searrow\colon\downarrow] = z_{3,4}$, $z[\mathbf{n}_\searrow\colon\downarrow] = X_{\mathsf{b},\mathsf{c},\mathsf{d}}$, and $X[\mathbf{n}_\searrow\colon\downarrow\uparrow]$ consists of X_b (shared with $\mathbf{n}_\swarrow$) and X_d (shared with $\mathbf{n}_\uparrow$). So $\mathbf{n}_\searrow$ passes

$$p^M_{\mathbf{n}_\searrow} = p(X_{\mathsf{b},\mathsf{d}}|z_{3,4}) = \mathcal{N}^{-1}\Big(\big(\begin{smallmatrix} 2 \\ -2 \end{smallmatrix}\big), \Big(\begin{smallmatrix} 1/2 & -1/2 \\ -1/2 & 1/2 \end{smallmatrix}\Big)\Big). \tag{91}$$

to $\mathbf{n}$. This Gaussian is degenerate, because $z_{3,4}$ contain no absolute position information. It declares the difference $X_\mathsf{d} - X_\mathsf{b}$ as $\mathcal{N}(2, 2)$ plus an infinite uncertainty for $X_\mathsf{d} + X_\mathsf{b}$. This is typical for SLAM and no problem in the information form. Node $\mathbf{n}$ multiplies both ($\odot$) by (24).

$$p^M_{\mathbf{n}_\swarrow} \cdot p^M_{\mathbf{n}_\searrow} = p(X_{\mathsf{b},\mathsf{d}}|z_{1\ldots 4}) = \mathcal{N}^{-1}\Big(\Big(\begin{smallmatrix} 4/3 \\ -2 \end{smallmatrix}\Big), \Big(\begin{smallmatrix} 5/6 & -1/2 \\ -1/2 & 1/2 \end{smallmatrix}\Big)\Big) = \mathcal{N}\Big(\big(\begin{smallmatrix} 1 \\ 3 \end{smallmatrix}\big), \big(\begin{smallmatrix} 3 & 3 \\ 3 & 5 \end{smallmatrix}\big)\Big)$$

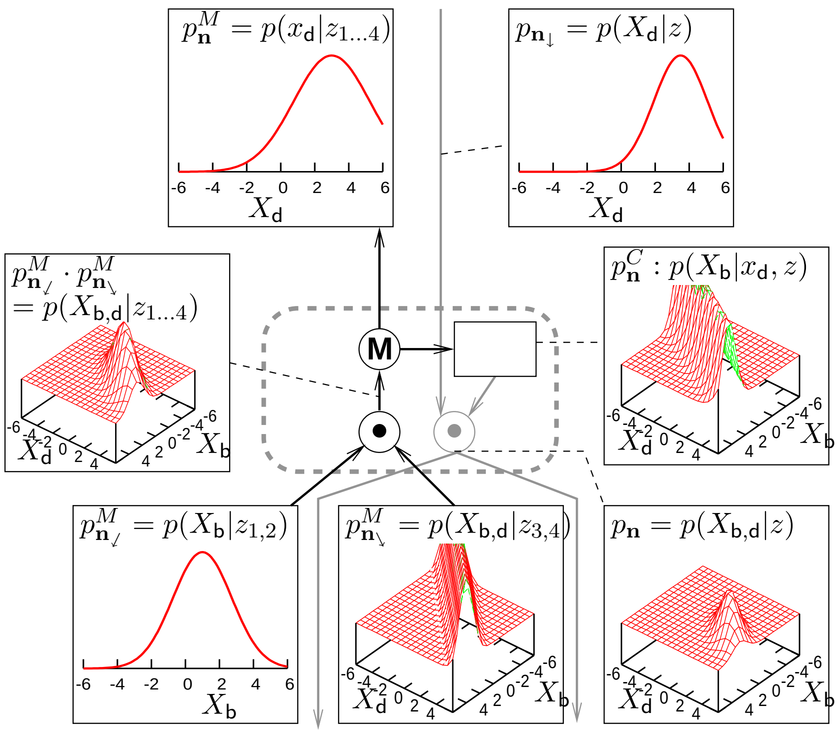

$p^M_{\mathbf{n}} = p(x_{\mathsf{d}}|z_{1...4})$
$p_{\mathbf{n}_\downarrow} = p(X_{\mathsf{d}}|z)$
$p^M_{\mathbf{n}_\swarrow} \cdot p^M_{\mathbf{n}_\searrow} = p(X_{\mathsf{b,d}}|z_{1...4})$
$p^C_{\mathbf{n}} : p(X_{\mathsf{b}}|x_{\mathsf{d}}, z)$
M
$p^M_{\mathbf{n}_\swarrow} = p(X_{\mathsf{b}}|z_{1,2})$
$p^M_{\mathbf{n}_\searrow} = p(X_{\mathsf{b,d}}|z_{3,4})$
$p_{\mathbf{n}} = p(X_{\mathsf{b,d}}|z)$
X_{d}
X_{b}

Figure 18: The probability distributions involved in computation at node **n** in figure 2. The example assumes 1D landmarks. Constraint z_1 declares X_{a} to be 0 with variance 2. Constraints $z_{2...7}$ declare the difference between successive landmarks to be 1 with variance 1. With only these constraints the result would be $E(X|z_{1...7}) = (0\ 1\ 2\ 3\ 4\ 5\ 6)^T$. The last constraint z_8 contradicts by declaring X_{g} as 7 with variance 2. Thus the estimate stretches to $E(X|z_{1...8}) = (0.2\ 1.3\ 2.4\ 3.5\ 4.6\ 5.7\ 6.8)^T$. The text explains how treemap computes this result.

The next step is to marginalize (Ⓜ) out $X[\mathbf{n}\colon \swarrow\!\!\!/ \searrow \uparrow\!\!\!/] = X_{\mathsf{b}}$ by (28).

$$p^M_{\mathbf{n}} = p(X_{\mathsf{d}}|z_{1\ldots4}) = \mathcal{N}^{-1}(-6/5, 1/5) = \mathcal{N}(3, 5) \tag{92}$$

$$p^C_{\mathbf{n}} = p(X_{\mathsf{b}}|x_{\mathsf{d}}, z) = \mathcal{N}(3/5\, x_{\mathsf{d}} - 4/5, 1) \tag{93}$$

Note, that the mean of $p^C_{\mathbf{n}}$ is defined as a linear function $Hx_{\mathsf{d}} + h$ of x_{d}. In this example $H = \left(\begin{smallmatrix} 3/5 \end{smallmatrix} \right)$ and $h = \left(\begin{smallmatrix} -4/5 \end{smallmatrix} \right)$ by (27). H, h and the covariance $P^{-1} = \left(\begin{smallmatrix} 6/5 \end{smallmatrix} \right)$ are stored at $\mathbf{n}$. $p^M_{\mathbf{n}}$ is passed to $\mathbf{n}_{\uparrow}$. Up to now only $z_{1\ldots4}$ have been integrated. Imagine $p^M_{\mathbf{n}}$ was directly fed back into $\mathbf{n}$ using it as $p_{\mathbf{n}\uparrow}$, which should actually be computed by the parent. Then $z_{5\ldots8}$ were bypassed and $\mathbf{n}$ would provide $\hat{x}_{\mathsf{a}\ldots\mathsf{d}} = \left(\begin{smallmatrix} 0 & 1 & 2 & 3 \end{smallmatrix} \right)$. Instead at some point above $\mathbf{n}$ the contradicting distributions $p(X_{\mathsf{d}}|z_{1\ldots4}) = \mathcal{N}(3, 5)$ and $p(X_{\mathsf{d}}|z_{5\ldots8}) = \mathcal{N}(4, 5)$ are integrated and

$$p_{\mathbf{n}_{\uparrow}} = p(X_{\mathsf{d}}|z) = \mathcal{N}(3.5, 5/2) \tag{94}$$

is passed from $\mathbf{n}_{\uparrow}$ to $\mathbf{n}$. It is multiplied (⊙) with $p^C_{\mathbf{n}}$ by (33) yielding

$$p_{\mathbf{n}} = p(X_{\mathsf{b},\mathsf{d}}|z) = \mathcal{N}\left(\left(\begin{smallmatrix} 1.3 \\ 3.5 \end{smallmatrix} \right), \begin{pmatrix} 21/10 & 3/2 \\ 3/2 & 5/2 \end{pmatrix} \right) \tag{95}$$

which is passed down to $\mathbf{n}_{\swarrow}$ and $\mathbf{n}_{\searrow}$.