Density Trees for Efficient Nonlinear State Estimation

Henning Eberhardt, Vesa Klumpp, and **Uwe D. Hanebeck**
Intelligent Sensor-Actuator-Systems Laboratory (ISAS)
Institute for Anthropomatics
Karlsruhe Institute of Technology (KIT), Germany
{henning.eberhardt, vesa.klumpp}@kit.edu, uwe.hanebeck@ieee.org

Abstract — *In this paper, a new class of nonlinear Bayesian estimators based on a special space partitioning structure, generalized Octrees, is presented. This structure minimizes memory and calculation overhead. It is used as a container framework for a set of node functions that approximate a density piecewise. All necessary operations are derived in a very general way in order to allow for a great variety of Bayesian estimators. The presented estimators are especially well suited for multi-modal nonlinear estimation problems. The running time performance of the resulting estimators is first analyzed theoretically and then backed by means of simulations. All operations have a linear running time in the number of tree nodes.*

Keywords: Bayesian estimation, nonlinear estimation, space partitioning, tree structure.

1 Introduction

There exists a variety of different methods for estimating a system's state over time from noisy measurements. The most popular class of estimators are the Gaussian assumed density estimators, and of those the probably best known estimator is the Kalman filter [1] for linear models, and its extensions to slightly nonlinear models, such as the Extended Kalman filter [2] or the Sample-Based Gaussian estimator [3]. These estimators have the virtue of "fixed finite-dimensional sufficient statistics," meaning that their complexity does not increase over time. The development of such estimators, which are limited to exponential densities, is still an active topic of research [4].

A different approach is taken by particle filters [5]. These filters represent the estimated density by means of a set of discrete realizations, i.e., particles, which allows for a representation of arbitrary densities. These filters can cope with highly nonlinear systems and multi-modal densities, but are not deterministic.

This work leads to a class of flexible and adaptive state estimators that allow for the efficient treatment of nonlinear problems. Instead of defining a single Bayesian estimation algorithm that deals with a given density representation, a more general container-based framework is presented in this paper, which is not restricted to a single density representation.

The key idea is to recursively partition the state space into smaller parts, leading to a hierarchical tree structure, where each tree node is a container. The partitioning from one layer to the next is similar to n-dimensional Octrees [6] or B-trees [7].

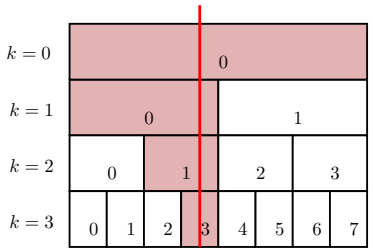


Figure 1: A binary tree: On the left are the layer indices while the node index is written within the nodes. The red line shows a specific point that is covered by the bright red nodes.

The container structure allows for the use of different density representation forms in the tree nodes, from very simple forms, such as Dirac impulses or uniform distributions, to very elaborate and complex ones, such as Gaussian mixture densities [8] or Fourier densities [9], even mixed in the same tree. This offers a compromise between the number of tree nodes and the number of parameters of individual node functions. For simple functions, many nodes are required and for complex functions, i.e., functions that require a large number of parameters, only few nodes are necessary. In general, the tree depth is not limited, which allows arbitrarily precise approximations for any node function.

The tree does not have to be fully populated. An adaptive density approximation algorithm chooses the optimal resolution for different regions in state space, depending on the shape of the true density function and the node functions. This can be beneficial for handling narrow densities, where wide parts of the state space have zero, or almost zero, probability, or regions where the node functions represent the true density very well. This dramatically cuts the number of used nodes in the tree and plays a major role in reducing time and space complexity.

1.1 Related Work

Because many common procedures and approaches are used in this work, a short discussion of related approaches is given in order to state the originality and novelty of this work.

Probability trees [10] are used to represent multi-dimensional probability distributions. For those probability trees, every tree layer represents conditional distributions of a joint distribution. For example, a two-dimensional discrete distribution can be modeled by a hierarchical density tree with two layers. The first layer describes the probability for the first variable, the

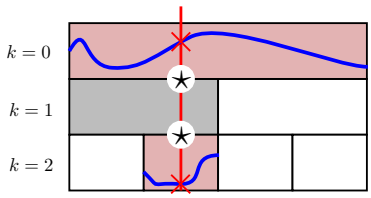


Figure 2: Evaluation of a binary tree: $\boxplus_k^{[0,10)}(3.75)$ have been marked in bright red. The gray node has been selected but contains ϵ data. The evaluation points marked in red on the node functions shown in blue are connected by $\star$.

second layer represents the conditional probabilities of the second variable, depending on the outcome of the first. This representation is clearly different from the proposed approach, where the same domain is described in every tree layer, but with different resolution and approximation quality.

Adaptive grid methods [11] are able to approximate densities by different partitioning approaches. Several approaches exist, especially for kd-trees [12, 13, 14] and bsp-trees [15]. In general, these approximation algorithms generate trees with adaptive, not predefined partitioning, which can be expensive to generate and to handle. Another difference to the proposed method is that the density information is stored in the leaf nodes only, whereas the proposed approach allows inner nodes to contain density information, too. On top of that, the node functions used in adaptive grid methods are usually very simple.

In the context of state estimation, wavelets [16] can be used to approximate density functions. Generally, the shape of wavelet functions is less limited and they have more overlappings. But still they are usually less flexible because the same basis function is used everywhere, i.e., in every node, whereas the density tree approach allows for different functions in the nodes. Thus, the proposed approach is slightly more general than wavelets.

The outline of this paper is as follows. In Section 2, the density tree is formally introduced. The operations required for estimation are described in Section 3. For efficient processing and representation of the densities, an approximation procedure is stated in Section 4. The Bayesian estimator with filter and prediction step based on the given operations is presented in Section 5. A complexity analysis of the proposed estimation framework with execution time evaluations is given in Section 6.

2 Description of Tree Structure

The goal of this chapter is to formalize the tree structure and give a basis for expressing arbitrary probability densities. Any tree shall have a bounded d-dimensional domain of definition $D = [a, b)^d$, which is a hypercube of half-open intervals for practical reasons. Furthermore, a tree consists of two basic elements, nodes and layers, which will be described first: The unique tree layers denoted by T_k are stacked upon one another as depicted in Fig. 1. They are indexed from root to leafs starting with $k = 0$ at the root. Each tree layer has the same domain of definition. Thus, all layers in a tree perfectly overlap. To allow for an evaluation, the layers are connected by a surjective operator $\star$ with

$$x \star y : (\mathbb{R} \cup \{\epsilon\}) \times (\mathbb{R} \cup \{\epsilon\}) \to (\mathbb{R} \cup \{\epsilon\}) \quad ,$$

where ϵ is the neutral element regarding $\star$ as seen in Fig. 2. In order to ease notation, another operator $\bigwedge$ is introduced, which denotes the concatenation of multiple $\star$ just like $+$ and $\sum$. By using the introduced symbols, a tree is described as

$$\underline{\mathsf{T}}^{\star} = \bigwedge_{k=0}^{\infty} \mathsf{T}_k \quad ,$$

where $\underline{\mathsf{T}}^{\star}$ is the symbol of a tree connected by $\star$. Now, the part of the tree that contains the probability density function will be introduced. Each tree layer T_k of a d-dimensional tree consists of a set of $2^{k \cdot d}$ nodes. All nodes on a tree layer are equally sized hypercubes and are indexed per layer as displayed in Fig. 1. So on every layer, there is a node with index zero.

Every node contains a function, which has a normalized domain of definition $[0, 1)^d$. Thus, a node can be described as a function

$$\eta(\underline{x}) : [0, 1)^d \to \mathbb{R} \cup \{\epsilon\} \quad ,$$

where ϵ is the neutral element regarding $\star$.

Now, as all elements the tree consists of are known, the evaluation of the tree, respectively the probability density function contained within, is described. Since all layers (and thus, nodes of the different layers) overlap, multiple nodes have to be combined for evaluation. For the formal description, two helper functions are defined. The first function selects the node index at tree level k

$$\boxplus_k^D(\underline{x}) : \begin{cases} D & \to & \mathbb{N}_0^d \\ x_j & \mapsto & \left\lfloor \frac{x_j - \min D_j}{|D_j|} \cdot 2^k \right\rfloor \end{cases} \quad , \qquad (1)$$

where $\min D$ is the vector of the smallest values in D and $|D|$ is the vector containig the width of D in each dimension. The second function describes the coordinate transformation to a specific node

$$\boxdot_k^D(\underline{x}) := \begin{cases} D & \to & [0, 1)^d \quad , \\ x_j & \mapsto & \frac{2^k (x_j - \min D_j)}{|D_j|} - \boxplus_k^{D_j}(x_j) \end{cases} \quad , \qquad (2)$$

which bears a close similarity to the carry-over of an integer division. The nodes selected by (1) are evaluated at the points given through (2). This is done per tree layer in ascending order and merged by $\star$ as depicted in Fig. 2. In summary, the evaluation can be written as

$$\begin{aligned} \underline{\mathsf{T}}^{\star}(\underline{x}) &= \bigwedge_{j=0}^{\infty} \mathsf{T}_j \left(\boxplus_j^D(\underline{x}), \boxdot_j^D(\underline{x}) \right) \\ &= \mathsf{T}_0 \left(\underline{0}, \boxdot_0^D(\underline{x}) \right) \\ &\star \mathsf{T}_1 \left(\boxplus_1^D(\underline{x}), \boxdot_1^D(\underline{x}) \right) \\ &\star \ldots \quad . \end{aligned}$$

This tree is evaluated at $\underline{x}$, so $\underline{x}$ has to be transformed to the according node index on each layer T_k by using $\boxplus_k^D(\underline{x})$ and to the coordinate within this node by using $\boxdot_k^D(\underline{x})$.

3 Operations on Tree Structure

In order to build a state estimator, several operations on the tree need to be defined. Except for the marginalization, all operations are independent of the layer connection $\star$. For the marginalization, an additive and an annihilating layer connection will be examined in detail. The additive connection just adds up all layers within the tree. The annihilating connection is defined as

$$x \sim y := \begin{cases} x & \text{where } y = \epsilon \\ y & \text{otherwise} \end{cases} \quad . \tag{3}$$

Evaluating a tree connected by this operator means evaluating the deepest node that does not have the value ϵ and overlaps the point of evaluation as seen in Fig. 2.

Finally, through the definition of the operations, several implications for the node functions arise that will be discussed in the end.

3.1 Erection

The simplest operation is the erection. Its goal is to extend a scalar function $f(\underline{x})$ on a d-dimensional domain by n dimensions to a scalar function $h([\underline{x}^T, \underline{y}^T]^T)$ on a $(d+n)$-dimensional domain denoted as

$$h\left(\left[\underline{x}^T, \underline{y}^T\right]^T\right) = f\left(\underline{x}\right) \uparrow_{\underline{y}} \text{ with } f\left(\underline{x}\right) \uparrow_{\underline{y}} := f\left(\underline{x}\right) \quad .$$

For trees, this operation is described as

$$\begin{aligned} \underline{\mathsf{H}}^\star\left(\left[\begin{array}{c} \underline{x} \\ \underline{y} \end{array}\right]\right) &= \bigwedge_{j=0}^{\infty} \mathsf{H}_j\left(\boxplus_j^D\left(\left[\begin{array}{c} \underline{x} \\ \underline{y} \end{array}\right]\right), \boxdot_j^D\left(\left[\begin{array}{c} \underline{x} \\ \underline{y} \end{array}\right]\right)\right) \\ &= \bigwedge_{j=0}^{\infty} \mathsf{F}_j\left(\boxplus_j^D\left(\underline{x}\right), \boxdot_j^D\left(\underline{x}\right)\right) = \underline{\mathsf{F}}^\star\left(\underline{x}\right) \quad . \end{aligned}$$

For density functions, this operation might seem trivial, but in the case of trees it does have the important property that any node contained in a tree at layer k is *duplicated* $2^{n \cdot k}$ times, when extended by n dimensions.

3.2 Product

The second operation required is the product. When calculating the product of two trees, the trees are scaled to have the same domain of definition. The multiplication of the densities f and g with the domains of definition D^f and D^g results in

$$h(x) = f(x \cdot |D^f| + \min D^f) \cdot g(x \cdot |D^g| + \min D^g) \quad ,$$

where h has the domain of definition $D^h = [0,1)^n$. As a consequence, all trees involved should have the same domain of definition expect for some special cases, where the rescaling might be of use. In addition, densities need to have the same dimension so that the erection might have to be applied before multiplication.

Thanks to the tree structure, only overlapping nodes have to be multiplied, which drastically reduces the overall amount of multiplications. In the following, $\underline{h}^k$ is the vector-valued index of a node in the product tree, while $\underline{c}^k$ and $\underline{g}^l$ are the indices of the nodes in the two factor trees at tree layer F_k and G_l. Moreover, $l \leq k$ is assumed without loss of generality. Then, only node combinations defined by

$$\underline{h}^k = \begin{cases} \underline{c}^k & \exists n \in \{0, \ldots, k\} : \underline{c}^{k-n} = \underline{g}^l \\ \text{undefined} & \text{otherwise} \end{cases} \tag{4}$$

are required to calculate the product. Property (4) has consequences for the product of two tree layers F_k and G_l and results in

$$\mathsf{H}_{\max\{l,k\}} = \mathsf{F}_k \cdot \mathsf{G}_l \quad , \tag{5}$$

which means that the product of two tree layers is always described on the layer with the highest index. Unfortunately, overlapping nodes do not necessarily have the same scale, because they might be from different tree layers. Thus, the nodes have to be scaled according to their overlapping areas. Again, assuming $l \leq k$, for the product node function η and the two factor node functions ς, whose node has the index $\underline{c}^k$, and γ, with index $\underline{g}^l$, the product is defined as

$$\eta(\underline{x}) = \varsigma\left(\underline{x}\right) \cdot \gamma\left(\frac{\underline{x} + \underline{c}^k}{2^{k-l}}\right) \quad \text{with } \underline{x} \in [0,1)^n \quad .$$

To account for non-commutative operators, the ordering needs to be correct according to $\star$ and (5), so layer H_k of the product tree is calculated by

$$\mathsf{H}_k = \left(\bigwedge_{j=0}^{k-1} \left(\mathsf{F}_k \cdot \mathsf{G}_j\right) \star \left(\mathsf{F}_j \cdot \mathsf{G}_k\right)\right) \star \left(\mathsf{F}_k \cdot \mathsf{G}_k\right) \quad . \tag{6}$$

In summary, the product operation the product of two trees $\underline{\mathsf{H}}^\star = \underline{\mathsf{F}}^\star \cdot \underline{\mathsf{G}}^\star$ is written as

$$\begin{aligned} \underline{\mathsf{F}}^\star \cdot \underline{\mathsf{G}}^\star &= \bigwedge_{j=0}^{\infty} \underbrace{\left(\bigwedge_{k=0}^{j-1} \left(\mathsf{F}_j \cdot \mathsf{G}_k\right) \star \left(\mathsf{F}_k \cdot \mathsf{G}_j\right)\right) \star \left(\mathsf{F}_j \cdot \mathsf{G}_j\right)}_{\mathsf{H}_j} \\ &= \underline{\mathsf{H}}^\star \quad , \end{aligned}$$

where H_j is the j-th layer of the product tree $\underline{\mathsf{H}}^\star$.

3.3 Joint Tree of Two Trees

In this section, the orthogonal joining of two trees will be described. For densities, the definition of the joint h of two independent densities f and g is defined by

$$h\left(\left[\begin{array}{c} \underline{x} \\ \underline{y} \end{array}\right]\right) = f\left(\underline{x}\right) \cdot g\left(\underline{y}\right) \quad . \tag{7}$$

Since the densities $f(\underline{x})$ and $g(\underline{y})$ are orthogonal, their tree representations $\underline{\mathsf{F}}^\star$ and $\underline{\mathsf{G}}^\star$ are orthogonal as well. The joint of two trees is calculated by use of the erection as defined in (3.1) to generate a fitting domain of definition for both trees and then generate the joint by using the multiplication (3.2) as shown in Fig. 3. This is formally written as

$$\underline{\mathsf{H}}^\star\left(\left[\begin{array}{c} \underline{x} \\ \underline{y} \end{array}\right]\right) = \underline{\mathsf{F}}^\star\left(\underline{x}\right) \uparrow_{\underline{y}} \cdot \underline{\mathsf{G}}^\star\left(\underline{y}\right) \uparrow_{\underline{x}} . \tag{8}$$

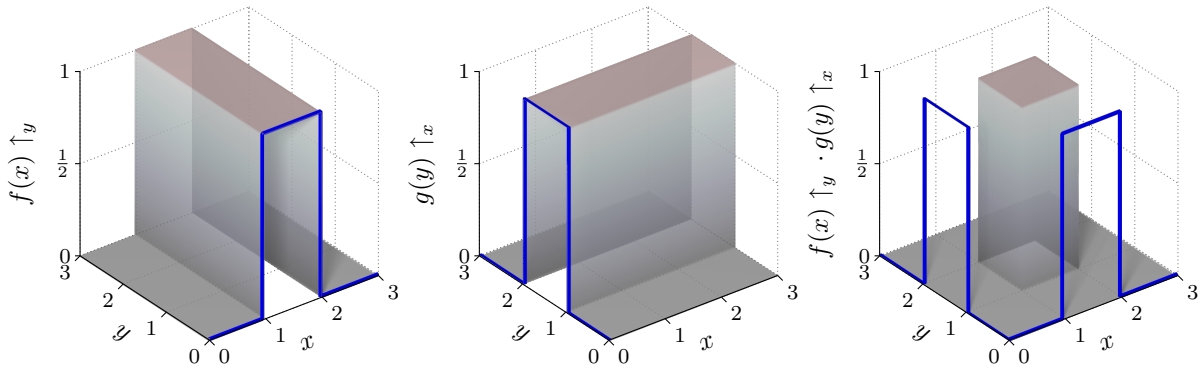


Figure 3: Left and middle: Two densities with extended dimensions. Right: Their joint density.

3.4 Marginalization ...

The marginalization of dimension k of one density f to another density g is defined as

$$g(\underline{x}) = \int_{-\infty}^{\infty} f(\underline{y})\ \mathrm{d}y_k \quad \underline{x} \in \mathbb{R}^{(N-1)}, \underline{y} \in \mathbb{R}^N \ .$$

By Transferring this to their tree representation, one obtains the integral

$$\begin{aligned} \underline{\mathsf{G}}^{\star}(\underline{x}) &= \int_{D_k} \underline{\mathsf{F}}^{\star}(\underline{y})\ \mathrm{d}y_k \\ &= \int_{D_k} \bigwedge_{j=0}^{\infty} \mathsf{F}_j\left(\boxplus_j^D(\underline{y}), \boxdot_j^D(\underline{y})\right) \mathrm{d}y_k \ . \end{aligned} \tag{9}$$

The calculation of this integral is complicated by the fact that its value depends on the layer connection $\star$. Here, the integral will be calculated for the additive connection and the annihilating connection. Therefore, a helper

$$h(\underline{x}, a, b) = [x_0, \ldots, x_{a-1}, b, x_{a+1}, \ldots, x_n]^T$$

is defined. Its main purpose is to ease the notation for the removal of a dimension.

... for the Additive Connection

When the layer connection is additive, the calculation of (9) is quite easy. In this case, it is possible to exchange integral and sum in order to obtain

$$\underline{\mathsf{G}}^{\star}(\underline{x}) = \sum_{j=0}^{\infty} \int_{D_k} \mathsf{F}_j\left(\boxplus_j^D(\underline{y}), \boxdot_j^D(\underline{y})\right) \mathrm{d}y_k \ .$$

As the tree layers are only defined piecewise and the nodes do not overlap within a single layer, the layers may also be written as sums. As a result, the complete marginalization of the k-th dimension is defined by

$$\underline{\mathsf{G}}^{\star}(\underline{x}) = \underbrace{\sum_{j=0}^{\infty} \left(\sum_{i=0}^{2^j-1} \int_{[0..1)} \mathsf{F}_j\left(h\left(\boxplus_j^D(\underline{x}), k, i\right), h\left(\boxdot_j^D(\underline{x}), k, a\right)\right) \mathrm{d}a \right)}_{\mathsf{G}_j} \ .$$

... for the Annihilating Connection

Unfortunately, the exchange of the layer connection and integral is not possible for the annihilating connection. This complicates things quite a bit and requires the definition of another helper

$$q_{lj}(i) = \frac{i}{2^{(l-j)}} - \left\lfloor \frac{i}{2^{(l-j)}} \right\rfloor \ .$$

For the result of the marginalization, only the deepest node that does not contain ϵ is of interest. Thus, the expression

$$\mathsf{G}_l(\underline{x}) = \sum_{i=0}^{2^l-1} \bigwedge_{j=0}^{l} \int_{q_{lj}(i)}^{q_{lj}(i+1)} \mathsf{F}_j\left(\left\lfloor \frac{h\left(\boxplus_j^D(\underline{x}), k, i\right)}{2^{(l-j)}} \right\rfloor, h\left(\boxdot_j^D(\underline{x}), k, a\right) \right) \mathrm{d}a$$

connects the integral of each node with the annihilating connection. While the direct marginalization of a tree connected by the annihilating connection might seem complicated, there is an easy interpretation: A tree connected by the annihilating connection may be transformed into a tree connected by the additive connection by simply moving all data, respectively all node functions that are not equal to ϵ, into its leaves. This means that when evaluating the transformed tree at a specific point, there exists at most one single node overlapping this point that does not evaluate to ϵ. Given this representation, the annihilating connection may be exchanged for the additive connection.

3.5 Conclusions for the Node Function

There are some implications for the node functions that result from the definition of the different operations above. First, the addition of two d-dimensional node functions γ_1 and γ_2 must be defined on axis-aligned areas $A_1, A_2 \in [0,1)^d$. Furthermore, the multiplication of two d-dimensional node functions γ_1 and γ_2 on axis-aligned areas $A_1, A_2 \in [0,1)^d$ must be defined. Additionally, the integration of a d-dimensional node function γ on axis-aligned areas $A \in [0,1)^d$ must be defined for any direction. Finally, for the annihilating connection $\sim$, the image set must be $\mathbb{R}_0^+ \cup \{\epsilon\}$ in order to guarantee admissible densities.

4 Density Approximation

While all mechanisms required for building an estimator have been described, there is still one task remaining. In most applications, the involved densities, e.g., the system noise, will not be available as a tree density. Thus, in order to acquire a density in the tree representation, an approximation of the desired density has to be performed. There are two approximation cases considered here:

- Approximation of arbitrary densities, e.g., Gaussians.
- Reapproximation of densities already in a tree representation, e.g., reduction, smoothing.

The reapproximation case can further be split up into a lossy reduction, which basically reduces the resolution of a tree by setting its nodes to ϵ, and a lossless reduction, also referred to as optimization. For each tree, there usually is an infinite amount of equivalent trees representing the same density. The goal of lossless reduction is to find the tree representation that has the least number of nodes with a value not equal to ϵ, which is usually the tree consuming the least memory when stored.

Getting a globally optimal solution for an approximation or reapproximation of a tree is an (NP)-hard (*without proof*) problem. While an optimal solution would give the lowest possible memory consumption, the following running time analysis in section VI will show that all other operations have linear running time. Thus, the reduction and approximation should have linear running time to prevent slowdowns for the estimation. For this reason, a quick and greedy algorithm is proposed.Furthermore, the same algorithm is used for both cases, approximation and lossy reapproximation. The algorithm consists of two steps:

1. recurse: **if** node approximation sufficient **then** return **else** descent to all children.
2. optimize the tree.

The optimization step is another greedy algorithm with linear running time in the number of nodes.

To demonstrate the performance of the approximation, a Gaussian density is approximated at different node counts in Fig. 5. The distance measure minimized in this case is $\max|\mathcal{N}(\underline{x}) - \underline{\mathsf{N}}^{\star}(\underline{x})|$ for each node, which converges reasonably quick in the KolmogorovâSmirnov distance and is easy to evaluate.

5 Bayesian State Estimator

In this section, the two basic elements of a Bayesian state estimator, the filter step and the prediction step, are introduced.

5.1 Filter Step

The goal of the filter step is the incorporation of the Likelihood f^L, which is the solution to the measurement equation given a certain measurement, into a prior system state f^P resulting in the estimated state f^e. The Bayesian filter step is defined by

$$f^e(\underline{x}) = \frac{f^p(\underline{x}) \cdot f^L(\underline{x})}{\int f^p(\underline{x}) \cdot f^L(\underline{x}) \mathrm{d}\underline{x}} \ .$$

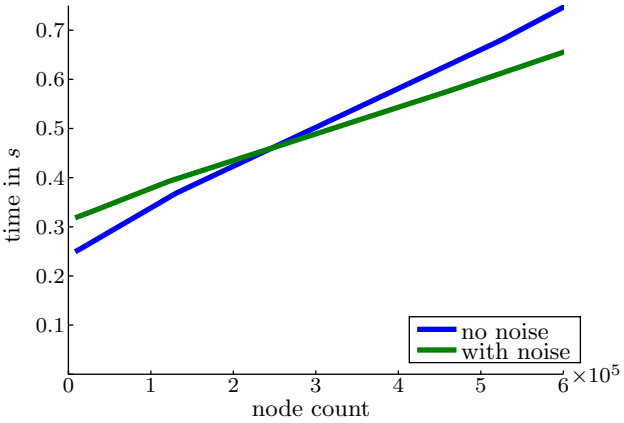


Figure 4: Execution times of the prediction for both, worst- and normal case.

By using the operations *multiplication* and *marginalization* introduced before, the tree representation can be applied to the Bayesian filter step, resulting in

$$\underline{\mathsf{F}}^{e\star}(\underline{x}) = \frac{\underline{\mathsf{F}}^{p\star}(\underline{x}) \cdot \underline{\mathsf{F}}^{L\star}(\underline{x})}{\int \underline{\mathsf{F}}^{p\star}(\underline{x}) \cdot \underline{\mathsf{F}}^{L\star}(\underline{x}) \mathrm{d}\underline{x}} \ ,$$

where the denominator is just for normalization.

5.2 Prediction Step

The prediction step is used to propagate a given system state f^e in time according to the system function. In the Bayesian estimator, the system function and the corresponding system noise are represented by a conditional density, which is called the transition density f^T. The formal description of the prediction step is the famous Chapman-Kolmogorov equation

$$f^p\left(\underline{x}_{k+1}\right) = \int f^T\left(\underline{x}_{k+1} | \underline{x}_k\right) \cdot f^e\left(\underline{x}_k\right) \mathrm{d}\underline{x}_k \ ,$$

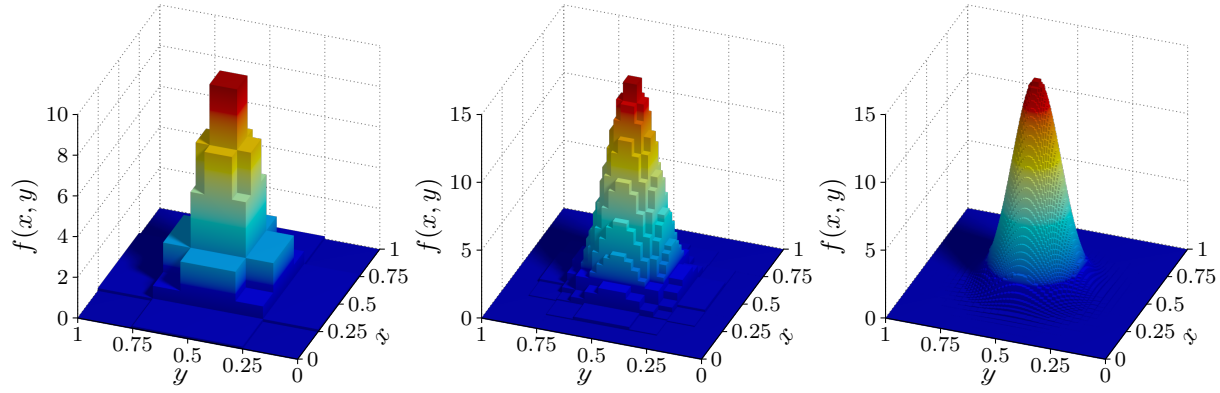
which is a hard to solve parameter integral. To solve this equation using the proposed tree representation, the operations *erection*, *product*, and *marginalization* are required. Given that all required operations are defined for the node functions, the solution of the Chapman-Kolmogorov equation

$$\underline{\mathsf{F}}^{p\star}\left(\underline{x}_{k+1}\right) = \int \underline{\mathsf{F}}^{T\star}\left(\underline{x}_{k+1}, \underline{x}_k\right) \cdot \underline{\mathsf{F}}^{e\star}\left(\underline{x}_k\right) \uparrow_{\underline{x}_{k+1}} \mathrm{d}\underline{x}_k.$$

for the proposed trees is straightforward.

6 Complexity ...

The detailed running time calculation is implementation-dependent. This is because the trees could be implemented as a tree structure, a hash-map, or even an array that directly maps tree nodes to memory addresses. In this section, the worst-case bounds are given in the Landau notation using the number of nodes not equal to epsilon as a basis. In addition, execution time measurements are taken from a slightly optimized C++ implementation. The implementation uses an optimized tree structure. This structure does not store all nodes, but the nodes



(a) 29 nodes (b) 205 nodes (c) 3253 nodes

Figure 5: Approximations of a Gaussian with variance 0.01 and mean 0.5, using uniform distributions as node functions.

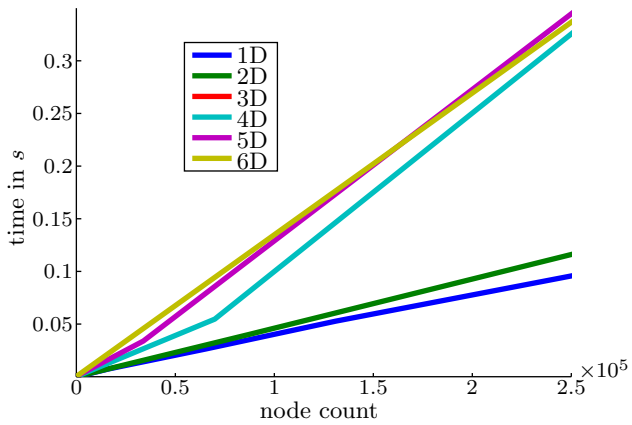


Figure 6: Execution time measurements for the product of two uniform densities.

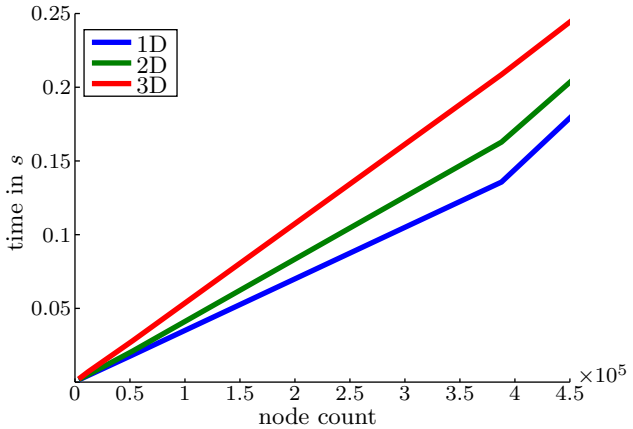


Figure 7: Execution time measurements for marginalizing a four-dimensional density to one, two, and three dimensions.

residing in memory are only those containing data or more than one child.

The node function used in the implementation is the most simple node function possible. In each node, a uniform distribution is stored. Still, all theoretical running time statements made here hold for any type of node function with a limited or fixed amount of parameters. For testing, a 2.4 GHz single core desktop computer with 4 Gigabytes of RAM is used. The tree layers are connected by the annihilating connection.

... of Multiplication and Filtering

The most expensive operation in the context of filtering is the multiplication uld be implemented as a tree structure, a hash-map, or even an array that directly maps tree nodes to memory addresses. In this section, the worst-of two trees. This operation does have a running time of $O(n+m)$ for trees with n and m nodes. The execution time is measured in two experiments. The first, which resembles the worst-case for it has the most nodes overlapping another, multiplies two sub-optimally approximated uniform distributions. Its result is depicted in Fig. 6.

... of Prediction

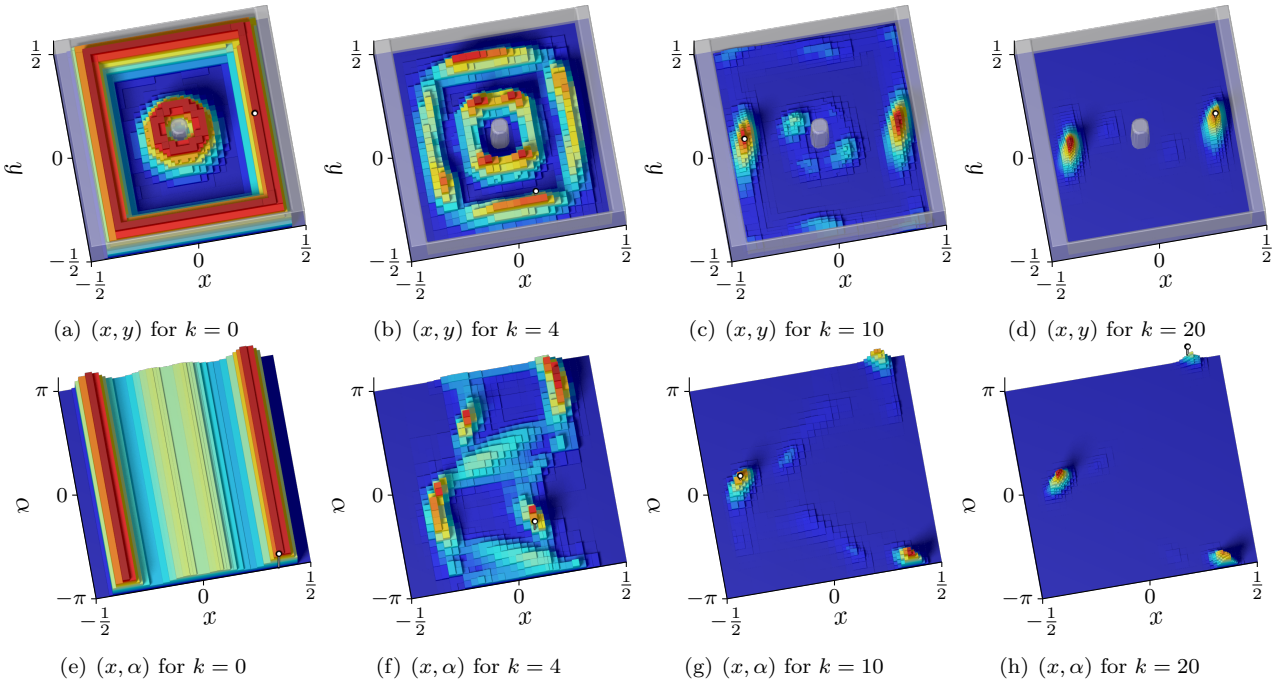
In prediction, the number of nodes contained in the transition density is the crucial factor. Given the fact that the transition density contains n nodes and the prior density m nodes, the worst-case running time is $O(n+m)$. The actual dimension of the densities does not have a direct impact on the running time of the algorithm. But still, given a high-dimensional density, the nodes required for a decent approximation might grow exponentially with the number of dimensions. For the execution time measurements there are two scenarios. First, the worst case is simulated. That is a two-dimensional transition density of a system with no system noise. The second simulation shows the typical case, where the system is disturbed by Gaussian noise. The execution times of both scenarios are shown in Fig. 4. It should be noted that execution times displayed may be further reduced by combining the reduction and prediction step. This technique is applied in the following localization simulations as it does also have the effect of dramatically cutting memory usage during the prediction process.

... of Marginalization and Integration

The marginalization of a density consisting of n nodes to a a lower-dimensional density does have a running time of $O(n)$. The same holds for the complete integration of a density, as required by the filtering step for normalization. The execution time behavior of the marginalization has been simulated by marginalizing a four-dimensional density to one, two, and three dimensions with the results shown in Fig. 7.

... of Other Operations

Finally, an overview of other operations on the tree structure is given, starting with the evaluation of the



(a) (x, y) for $k = 0$ (b) (x, y) for $k = 4$ (c) (x, y) for $k = 10$ (d) (x, y) for $k = 20$

(e) (x, α) for $k = 0$ (f) (x, α) for $k = 4$ (g) (x, α) for $k = 10$ (h) (x, α) for $k = 20$

Figure 8: Four different timesteps of the localization simulation. As the system state is three-dimensional, only its marginals along the (x, y)-axis, that is $\int f^e(x, y, \alpha)\mathrm{d}\alpha$, and the (x, α)-axis, that is $\int f^e(x, y, \alpha)\mathrm{d}y$, are shown. The gray areas in the (x, y)-plots show the map. The small pin is the vehicle's true attitude. The symmetry involved in the simulation results in a bimodal density in the end.

represented density at a specific point. The worst case, which is a special case of a tree consisting of only one branch is $O(n)$. As this case should rarely appear, especially for optimized structures, the usual evaluation lies in $O(log(n))$. The true execution time heavily depended on the depth of the tree and caching effects. On the test system, a seek speed of $5 \cot 10^7$ to $6 \cdot 10^8$ nodes per second is achieved. The proposed greedy approximation works in $O(n)$. The optimization of the tree structure has a running time of $O(k \cdot n)$, where a k of $1 \ldots 10$ is usually sufficient. The reduction or optimization is best applied regularly after filtering or prediction steps in order to control the total node count of the state densities.

7 Simulation

In order to give a proof of concept for the proposed estimators, a simple simulation is performed for a localization problem of an omnidirectional vehicle with very limited sensory input. The vehicle's system equation, which describes its movement, is

$$\begin{bmatrix} \boldsymbol{x}_{k+1} \\ \boldsymbol{y}_{k+1} \\ \boldsymbol{\alpha}_{k+1} \end{bmatrix} = \begin{bmatrix} \boldsymbol{x}_k + \boldsymbol{v}_k^x & + & s\sin(\boldsymbol{\alpha}_k + \boldsymbol{v}_k^\alpha) \\ \boldsymbol{y}_k + \boldsymbol{v}_k^y & + & s\cos(\boldsymbol{\alpha}_k + \boldsymbol{v}_k^\alpha) \\ \boldsymbol{\alpha}_k + \boldsymbol{v}_k^\alpha & + & \alpha_k^\Delta \end{bmatrix}, \tag{10}$$

where k denotes the timestep, $\boldsymbol{x}, \boldsymbol{y}$ the position, $\boldsymbol{\alpha}$ the movement direction and α^Δ its rate of change, s the speed of movement, and finally $\underline{\boldsymbol{v}}$ the system noise, which is distributed according to $\mathcal{N}(0, 0.01)$ for each component $\boldsymbol{v}^x, \boldsymbol{v}^y$, and $\boldsymbol{v}^\alpha$ without any correlation. While the initial starting attitude of the vehicle is $\hat{x}_0 = 0.35, \hat{y}_0 = 0$, and $\hat{\alpha}_0 = -\pi$ the movement speed and the angular change are fixed at $s = 0.094$ and $\alpha_k^\Delta = 0.05 \cdot \pi$ so the vehicle's movement describes a circle within 20 time steps. The goal of the simulation is to obtain an estimation of the attitude of the vehicle that is described by the random vector $[\boldsymbol{x}_k, \boldsymbol{y}_k, \boldsymbol{\alpha}_k]^T$. For this purpose, the vehicle is equipped with an exact map of its surroundings, an almost square room with a pillar in the middle (to make things more interesting), and a primitive sensor. The sensor is capable of measuring the distance to the closest wall to the vehicle without any angular information. A physical equivalent might be an omnidirectional mono-microphone carried by the vehicle measuring time delays of acoustic reflections of the vehicle's sounds from walls. This problem setup is an instance of the common wake-up robot problem.

The distances returned by the sensor are heavily disturbed by additive noise distributed according to $\mathcal{N}(0, 0.05)$. Figure 8 shows the estimated state at different time steps. For initialization of the estimator, the first likelihood is chosen. The estimator automatically keeps the number of nodes at a constant level of about $20,000$ nodes. Hence, the resolution of the density increases when a lot of mass is concentrated in a small area. The final result of the estimation is bimodal as the map and the track of the vehicle share a symmetry.

8 Conclusions

In this paper, a special tree structure, which is a generalization of Octrees or Quadtrees, is used to represent probability densities. The structure of the tree was carefully chosen to a predefined partitioning. This allows for very efficient handling of the tree structure. The data, respectively the densities, are stored by arbitrary node functions. This paper does only make very few restrictions on the node functions and thus, allows for

a great flexibility in their choice. After the tree structure is formally described, several operations required for a Bayesian estimator are introduced. Finally, the prediction step and filter step are defined. To demonstrate the performance of the proposed estimators, the running times are first stated in Landau notation and backed by some execution time measurements on an actual implementation.

As a tree structure is a hierarchical structure and all algorithms presented here have linear running time, they are very easy to parallelize enabling even better performance.

The densities are transformed into the tree representation through an approximation. The approximation error generated here can easily be calculated and updated when the estimator runs. As a result, it is possible to give an upper bound of the maximum error caused by the approximation at any time and introduce corrections if required. In the end, a density tree does not have to be present in memory, since it can easily be calculated online when accessing a certain node.

The proposed technique is especially useful when a density consisting of many modes is updated/processed in very small bits. This means that there are many minor changes to small parts of the density. This does for example include elimination scenarios, where the information gained in a time step is only the exclusion of some configurations in a large parameter space. It does as well include some mapping scenarios, where the sensors can only cover a very small area of a map. In the end, the proposed method gets very efficient when most of the probability mass is concentrated on a small area, enabling efficient tracking as seen in the simulation.

For future work, the use of more complex node functions has to be evaluated. Using polynomials or exponential densities seems promising here. The main gain would lie in a reduced node count. Reducing the node count will still leave the prediction as a major caveat. Due to the fact that the transition density's dimension is twice the dimension of the state to be predicted, it does always contain very many nodes. Here, techniques like the butterfly scheme for calculating fast convolutions, which could give a great speedup, have to be evaluated.

References

[1] R. E. Kalman, "A new Approach to Linear Filtering and Prediction Problems," *Transactions of the ASME, Journal of Basic Engineering*, vol. 82, pp. 35–45, 1960.

[2] A. Papoulis and S. U. Pillai, *Probability, Random Variables and Stochastic Processes*, 4th ed. McGraw-Hill, 2002.

[3] M. F. Huber and U. D. Hanebeck, "Gaussian Filter based on Deterministic Sampling for High Quality Nonlinear Estimation," in *Proceedings of the 17th IFAC World Congress (IFAC)*, Seoul, Korea, Jul. 2008.

[4] F. Daum, "Nonlinear Filters: Beyond the Kalman Filter," *IEEE Aerospace and Electronic Systems Magazine*, vol. 20, no. 8, pp. 57–69, Aug. 2005.

[5] M. S. Arulampalam, S. Maskell, N. Gordon, and T. Clapp, "A Tutorial on Particle Filters for Online Non-linear/Non-Gaussian Bayesian Tracking," *IEEE Transactions on Signal Processing*, vol. 50, no. 2, pp. 174–188, 2002.

[6] H. Samet, "Octree Approximation and Compression Methods," in *Proceedings of the 1st International Symposium on 3D Data Processing Visualization and Transmission*, 2002.

[7] R. Bayer and E. McCreight, "Symmetric Binary B-Trees: Data Structure and Maintenance Algorithms," *Acta Informatica*, vol. 1, pp. 290–306, 1972.

[8] D. L. Alspach and H. W. Sorenson, "Nonlinear Bayesian Estimation using Gaussian Sum Approximations," *IEEE Transactions on Automatic Control*, vol. 17, no. 4, pp. 439–448, Aug. 1972.

[9] D. Brunn, F. Sawo, and U. D. Hanebeck, "Nonlinear Multidimensional Bayesian Estimation with Fourier Densities," in *Proceedings of the 2006 IEEE Conference on Decision and Control (CDC 2006)*, San Diego, California, Dec. 2006, pp. 1303–1308.

[10] B. Stenger, A. Thayananthan, P. H. S. Torr, and R. Cipolla, "Filtering using a tree-based estimator," in *ICCV '03: Proceedings of the Ninth IEEE International Conference on Computer Vision*. Washington, DC, USA: IEEE Computer Society, 2003, p. 1063.

[11] S. J. Owen, "A Survey of Unstructured Mesh Generation Technology," in *Proceedings of the 7th International Meshing Roundtable*, 1998.

[12] V. Klumpp and U. D. Hanebeck, "Dirac Mixture Trees for Fast Suboptimal Multi-Dimensional Density Approximation," in *Proceedings of the 2008 IEEE International Conference on Multisensor Fusion and Integration for Intelligent Systems (MFI 2008)*, Seoul, Republic of Korea, Aug. 2008, pp. 593–600.

[13] A. G. Gray and A. W. Moore, "N-Body Problems in Statistical Learning," *Advances in Neural Information Processing Science*, vol. 13, 2001.

[14] A. G. Gray and A. W. Moore, "Nonparametric Density Estimation: Toward Computational Tractability," in *Proceedings of the third SIAM International Conference on Data Mining*, 2003.

[15] A. V. Kozlov and D. Koller, "Nonuniform Dynamic Discretization in Hybrid Networks," in *Thirteenth Conference on Uncertainty in Artificial Intelligence (UAI-97)*, 1997.

[16] D. R. M. Herrick, G. P. Nason, and B. W. Silverman, "Some New Methods for Wavelet Density Estimation," *Sankhyã: The Indian Journal of Statistics*, vol. 63, no. Series A, Pt. 3, pp. 394–411, 2001.