Robotics and Autonomous Systems 69 (2015) 28–39

Contents lists available at ScienceDirect

Robotics and Autonomous Systems

journal homepage: www.elsevier.com/locate/robot

Generic NDT mapping in dynamic environments and its application for lifelong SLAM[☆]

Erik Einhorn [*], Horst-Michael Gross

Ilmenau University of Technology, Germany

HIGHLIGHTS

- We present a new mapping approach that combines normal distribution transform (NDT) and occupancy mapping.
- The mapping approach is fully generic and suitable for 2D and 3D mapping with different sensors.
- We describe a method for detecting and handling dynamic objects to allow mapping in highly dynamic environments.
- Based on the mapping algorithm a graph based SLAM algorithm is described.
- The presented SLAM approach allows lifelong mapping and localization in real world applications.

ARTICLE INFO

Article history:
Available online 30 August 2014

Keywords:
Lifelong SLAM
Detection and tracking of moving objects
2D and 3D mapping
Normal Distribution Transform
Occupancy mapping
Map registration
Mobile robots

ABSTRACT

In this paper, we present a new, generic approach for Simultaneous Localization and Mapping (SLAM). First of all, we propose an abstraction of the underlying sensor data using Normal Distribution Transform (NDT) maps that are suitable for making our approach independent from the used sensor and the dimension of the generated maps. We present several modifications for the original NDT mapping to handle free-space measurements explicitly. We additionally describe a method to detect and handle dynamic objects such as moving persons. This enables the usage of the proposed approach in highly dynamic environments. In the second part of this paper we describe our graph-based SLAM approach that is designed for lifelong usage. Therefore, the memory and computational complexity is limited by pruning the pose graph in an appropriate way.

1. Introduction

Simultaneous localization and mapping (SLAM) is one of the fundamental challenges in mobile robotics. It constitutes a difficult problem as consistent mapping depends on the knowledge of the robot's current position, while robust self-localization on the other hand requires an accurate map of the environment. Therefore, the localization and the mapping process are inherently coupled [1]. Consequently, the SLAM problem has been thoroughly analyzed for decades and researchers came up with many different solutions. However, when it comes to the practical application of SLAM, it is often used for map acquisition exclusively during an offline map learning phase as part of the initial setup of the robot in its novel environment [2]. During the robot's operation phase, this map then is used for robot localization, i.e. pose tracking, for instance by using particle filter based Monte Carlo localization [3].

In our previous real-world applications where we implemented tour guide robots and interactive shopping assistants [4], we also followed the philosophy of a map learning phase and a separate operation phase. However, todays complex applications such as robot companions that assist elderly people in their home environments [5] require a paradigm shift. Typically, these environments are semi-static or dynamic, i.e. the location of obstacles like chairs or tables change over time. Therefore, a separate map learning phase is no longer acceptable. Instead, the mapping phase must continue during the whole operation time of the robot to permanently adapt the map to the changes in the environment. This results in the so called lifelong SLAM problem.

A lifelong SLAM algorithm that is suitable for such scenarios must be able to constantly update the map of the environment without increasing the complexity for map updates with new measurements. Moreover, it must operate in realtime to continuously provide estimates of the robot's location to other navigation modules, like path planners.

☆ This work has received funding from the German Federal Ministry of Education and Research as part of the ROREAS project under grant agreement no. 16SV6133.

* Corresponding author. Tel.: +49 3677 69 4168.
E-mail address: Erik.Einhorn@tu-ilmenau.de (E. Einhorn).

http://dx.doi.org/10.1016/j.robot.2014.08.008
0921-8890/© 2014 Elsevier B.V. All rights reserved.

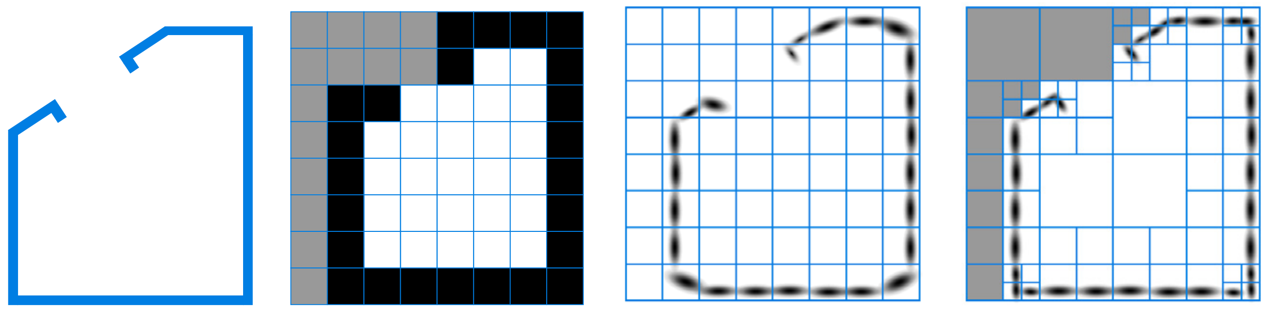
(a) Environment. (b) Occupancy grid-/voxel-map. (c) NDT map. (d) Hybrid NDT occupancy map.

Fig. 1. Different grid-based map representations for the same environment (a room) that is shown on the left. Occupied cells are drawn in black, free cells are colored white, while unknown, unobserved cells are shown gray.

Modern assistance robots typically have a variety of sensors that provide different kinds of information about the robot's environment. Laser Range Finders provide two dimensional range data that can be used to create 2D maps. Depth cameras on the other hand provide depth images that are suitable to create 3D maps. Also a single camera mounted in front of the robot can provide such 3D information when it is used for monocular scene reconstruction [6]. Therefore, we are interested in applying a generic mapping and SLAM approach that is able to process such 2D and 3D information equally well.

In this paper, we describe such a generic and sensor-independent graph-based SLAM system that integrates different recent approaches such as:

- Normal Distribution Transform (NDT) occupancy mapping [7,8]
- NDT map registration [9]
- robust pose-graph optimization [10]
- pruning of the pose-graph for lifelong operation [11].

In addition to describing how these techniques can be combined in a consistent way, another major contribution of this paper is a method for Detecting And Tracking Moving Objects or persons properly, which is also known as DATMO [12]. This method is integrated seamlessly into our mapping approach and allows its application in highly dynamic environments, even if sensors with limited fields of view are used.

Aside from that, there are several minor contributions such as an efficient occupancy update method, the proper handling of range shadows that are caused by structured-light sensors, and improvements in NDT-to-NDT registration.

In summary, the proposed mapping and SLAM approach:

1. is implemented in a generic way for 2D and 3D mapping using different sensors such as laser or depth cameras
2. operates in semi-static or dynamic environments and adapts the map to the changing environment
3. operates in realtime and allows for online robot localization
4. allows for lifelong mapping with constant complexity.

This paper is organized as follows. Section 2 outlines the state of the art in SLAM and robot mapping. In Section 3, we describe the generic mapping approach in detail. Section 4 depicts an extension for the mapping algorithm that allows to detect and handle dynamic objects properly. In Section 5, we finally combine the described mapping algorithms to a SLAM approach. In Section 6, we show several results that we have obtained using the presented approaches for different kinds of sensors. Finally, we conclude with an outlook on future work.

2. Related work

As stated before, a large variety of different SLAM approaches is available. Some techniques interpret the SLAM problem as a filtering problem and apply Extended Kalman filters [13] or Rao–Blackwellized Particle Filters [14,1] to solve it. Others apply smoothing techniques [15,16] to solve the full SLAM problem, i.e. beside the estimation of the most consistent map they keep the complete robot trajectory as part of the estimation problem. While these approaches provide direct solvers for the SLAM problem, others, e.g. *g2o* [17] exploit the sparsity of the SLAM problem by formulating it as a pose graph optimization problem. The problem of such optimizers is that they are not robust against outliers in data association. Hence, wrong loop-closures have a catastrophic impact on the resulting map and the robot's pose estimates. This problem is addressed in [10] by also including the topology of the pose graph into the optimization procedure. It allows the algorithm to switch off erroneous constraints. As a result, the optimization of the pose graph becomes extremely robust against false loop-closure constraints.

Another disadvantage of most graph-based techniques is the increasing complexity that grows with the length of the trajectory since more and more vertices are added to the graph over time. Consequently, this would prohibit the usage of such graph-based techniques for lifelong SLAM. In [11] this problem is tackled by merging vertices of the pose graph so that it only grows when the robot acquires relevant new information about the environment in terms of expected information gain.

Most implementations of the aforementioned SLAM approaches use laser range finders as sensors and occupancy grid maps as representation of the environment. With new devices, like 3D laser range finders, stereo cameras, time-of-flight cameras, or other depth sensing cameras using structured light, that have become available in recent years, 3D information of the local surroundings can be acquired. However, these sensors produce a huge amount of data and an appropriate representation is needed for processing this data efficiently.

A very popular 3D environment representation are voxel maps [18]. Similar to 2D occupancy grid maps, the robot's surroundings are partitioned into regular cubic volumes (*voxels*). Each voxel stores a probability whether the volume is occupied by an obstacle or free (see Fig. 1(b)). These maps, therefore, allow to model free space and unknown areas explicitly using the stored occupancy value. Voxel maps are usually stored using octree representations [19–22] that allow to store large regions of free space more efficiently.

A different map representation is the Normal Distribution Transform (NDT). It was originally proposed in [23] for efficient laser scan matching and was later extended to three dimensions [24]. Similar to octree-based maps, the mapped volume is subdivided into voxels. However, instead of estimating an occupancy probability for the whole voxel, the observed range measurements within each voxel are represented by a normal distribution (see Fig. 1(c)). As shown in [25], such NDT maps achieve a significantly higher accuracy than voxel or octree-based maps when the same cell resolution is used. Moreover, NDT maps are continuously differentiable and hence enable efficient map registration algorithms. Despite of its advantages, the Normal Distribution Transform has not yet been widely accepted by the SLAM community.

The original formulation of NDT maps as they are used in [23–25] has a major drawback: It models the distribution of obstacles only, while free space is not taken into account at all. This disallows their usage for mapping in dynamic environments, since objects and obstacles that were removed in the environment cannot be removed from the map and hence lead to inconsistencies.

In [7,8] an extension is proposed that models an occupancy probability for each cell, in order to overcome this disadvantage. For each measurement, a ray is cast through the cells of the NDT map. If – due to a dynamic obstacle that has moved – the consistency of an observation with respect to an occupied cell containing a normal distribution is low, the occupancy probability of the cell is decreased using a forward sensor model. The occupancy probability for empty cells and cells 'hit' by an observation are adapted by a constant amount without using a sensor model.

3. Normal distribution transform mapping

Independently from the work of Saarinen et al. [7,8], we developed a similar method for NDT mapping, that also combines the ideas of occupancy grid maps and NDT maps, in order to add the capability to integrate information about free space into the maps.

In contrast to [8] where a ray casting approach is used for updating the cells, we use a different method that is better suited for our map representation and achieves a better performance. Moreover, we present a fully generic implementation that allows to generate 2D as well as 3D maps without any changes in the algorithms. Therefore, we also derive a generic beam sensor model, that is used for updating the occupancy probability for all cells.

Similar to occupancy grid maps and occupancy voxel maps, we partition the mapped volume into discrete cells. These cells are managed in a tree structure. This allows to generate maps with an adaptive resolution that is adjusted depending on the level of detail of the mapped surroundings (not covered in this paper). For 2D maps, we use a quadtree, and for 3D maps we use an octree. However, in the following we do no longer distinguish between 2D and 3D maps and quadtrees and octrees. Instead, we use a generalized tree similar to the N^d-tree that was presented in [22]. Depending on the dimensionality d of the map, our data structure splits the cells in each dimension. Consequently, each cell is subdivided into 2^d child cells, which results in a standard quadtree for 2D maps and in an octree for 3D maps.

As in [24], each cell c of such a 2^d tree stores the mean $\boldsymbol{\mu}_c \in \mathbb{R}^d$ and the covariance $\boldsymbol{\Sigma}_c \in \mathbb{R}^d \times \mathbb{R}^d$ of a normal distribution $\mathcal{N}(\boldsymbol{\mu}_c, \boldsymbol{\Sigma}_c)$. It approximates the surface points of the object that is covered by the cell as a probability distribution and therefore achieves a higher precision than a sole voxel map. The totality of all such normal distributions of all cells of the map M can be considered as a Gaussian mixture model that models the probability

$$p(\mathbf{x} \in \mathcal{S}) = \sum_{c \in M} w_c \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_c, \boldsymbol{\Sigma}_c)$$

whether a point $\mathbf{x} \in \mathbb{R}^d$ belongs to the set of surfaces $\mathcal{S}$ of objects and obstacles in the environment.

To be able to represent free space explicitly, we combine the ideas of occupancy maps and NDT maps and additionally store an occupancy value o_c in each cell which acts as a prior for the stored normal distribution in the cell. The final probability distribution of surface points is then expressed as $o_c\mathcal{N}(\boldsymbol{\mu}_c, \boldsymbol{\Sigma}_c)$. In other words, o_c models the probability $o_c = p(c = occ)$ whether the volume that is represented by the stored Gaussian is occupied by an object or free. In empty cells that do not store a Gaussian, we assume a uniform distribution of the occupancy probability. Fig. 1(d) shows such a hybrid NDT occupancy map.

3.1. Mapping-backend and sensor-frontends

As described in the introduction, we want our mapping approach to be generic and suitable for a broad spectrum of distance measuring sensors. As stated above, the used tree-structure for representing the map is already independent from the dimensionality of the used sensor. Moreover, many operations of the mapping algorithm can be implemented independently from the used range sensor. We encapsulate these operations in a *mapping backend*. These operations are described in the next subsections.

Beside the mapping backend, we have different *sensor frontends* for each type of range sensor as shown in the left part of Fig. 7. These frontends act as an abstraction layer between the sensor data and the mapping backend. The advantage of this architecture is, that the task of each sensor frontend is simplified to processing the range data of the respective sensor and calling the backend to perform abstract operations such as updating a cell as occupied or free.

3.2. Updating the map with range measurements

A priori, the state of all cells is unknown. For this reason, the covariance in each cell is set to "infinity". As the occupancy probability is truncated to the volume of each cell, the probability mass will initially take up the whole cell uniformly. This justifies the use of a uniform distribution for unknown or empty cells to represent the occupancy probability, as stated above. Moreover, the occupancy value is initially set to $o_c = 0.5$ to indicate that the state of the whole cell is unknown.

With new measurements of the used range sensor, the map is updated. Basically, the map update consists of two steps. In a first step the "shape" of the map is updated by adapting the Gaussians in the occupied cells according to the sensor measurements or rather according to the surfaces of the objects. In a second update step the stored occupancy probability of each cell is adapted. Here, free cells, that do not contain a Gaussian, are updated as well using the range measurements.

Each single range measurement is defined by the sensor's position $\mathbf{p}$ within the map, the direction $\mathbf{d}$ of the range measurement and the measured range z. Using this information the endpoint $\mathbf{x} \in \mathbb{R}^d$ of the measurement is defined by $\mathbf{x} = \mathbf{p}+z\mathbf{d}$. This endpoint is usually located on the surface of an object in the environment. To integrate the measurement into the map, the mapping algorithm first determines the cell c that is "hit" by the measurement, i.e. the cell where the endpoint of the measurement is located in. This is done using an efficient lookup as described in [26]. Afterwards, the normal distribution in this cell is updated using the following incremental update rule:

$$\begin{aligned}
\boldsymbol{\mu}'_c &= \alpha\boldsymbol{\mu}_c + (1-\alpha)\mathbf{x} \\
\boldsymbol{\Sigma}'_c &= \alpha\boldsymbol{\Sigma}_c + \alpha(1-\alpha)(\boldsymbol{\mu}_c - \mathbf{x})(\boldsymbol{\mu}_c - \mathbf{x})^\top \qquad (1) \\
k'_c &= k_c + 1
\end{aligned}$$

where $\alpha = k_c/(k_c+1)$ and k_c expresses the number of updates of the cell. In a static environment this update rule gives the maximum likelihood estimate of the mean and covariance of all measurements that fall into the same cell. However, since we use our approach in dynamic environments with moving objects and persons, we limit the value of k_c for each cell. As a result, the above update rule then computes an exponentially weighted moving average and covariance where new measurements have a stronger influence than older ones. This allows the normal distributions of each cell to adapt easily to a change in the objects positions.

3.3. Occupancy update using a generic beam-sensor-model

Beside updating the normal distribution of the hit cell, the mapping algorithm also updates its occupancy value and the occupancy

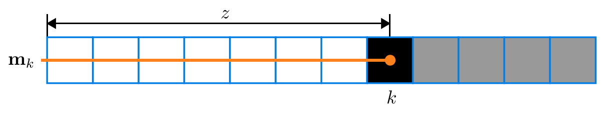

Fig. 2. One dimensional map along the sensor beam where the kth cell is hit by the range measurement z.

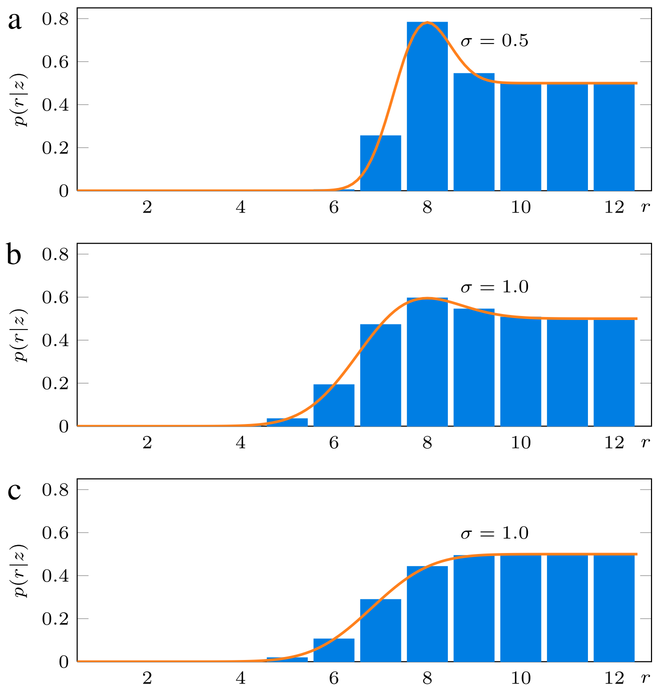

Fig. 3. Plots of the used inverse sensor models. The graph shows the probability $p(r|z)$ plotted against the distance r from the sensor's origin for a sensor measurement $z = 8$. The bars indicate the occupancy probabilities for the cells of the expected map $E[\mathbf{m}|z]$ given the sensor reading z. (a)–(b) The inverse sensor model shown for different variance parameters of the assumed sensor noise. (c) A modified inverse sensor model that provides evidence for free space only.

values of all cells along the sensor beam between the sensor's position and the endpoint of the measurement according to:

$$\operatorname{logit}(o_c') = \operatorname{logit}(o_c) + \operatorname{logit}(p(r|z)) \tag{2}$$

with $\operatorname{logit}(p) = \log(p) - \log(1-p)$ being the log-odds representation of a probability p. This is the standard update rule of occupancy maps as described in [27]. The probability $p(r|z)$ is the inverse sensor model. It yields the probability for a cell at distance r being occupied when a range measurement of z was obtained.

For our approach, we derive a generic one-dimensional beam-sensor model, that can be used for most sensors, whose single distance measurements are obtained along a narrow beam, such as laser range finders, depth- and stereo-cameras, and even approaches for monocular scene reconstruction [6]. Sonar sensors – on the other hand – cannot be represented by such a 1D model, as their sensor beam is wide and covers a large volume.

To derive the inverse sensor model $p(r|z)$, we compute the conditional expectation of a 1D map $\mathbf{m}$ whose cells are located along the sensor beam as shown in Fig. 2:

$$E[\mathbf{m}|z] = \eta \sum_k \mathbf{m}_k p(z|\mathbf{m}_k) p(\mathbf{m}_k). \tag{3}$$

The above equation sums over possible 1D maps $\mathbf{m}_k$, where $p(z|\mathbf{m}_k)$ yields the probability for measuring z given a map $\mathbf{m}_k$, which is known as forward sensor model, and $p(\mathbf{m}_k)$ expresses the probability that the given map occurs in the set of all possible 1D maps. One can verify that it is sufficient to sum over a special type of 1D maps, namely those maps $\mathbf{m}_k$ where the first $k-1$ cells are empty, the kth cell is occupied and the rest of the cells are unknown, expressed by an occupancy of 0.5:

$$\mathbf{m}_k = (\underbrace{0, 0, 0, \ldots, 0}_{k-1 \text{ times}}, \underset{\substack{| \\ k}}{1}, 0.5, 0.5, 0.5, \ldots). \tag{4}$$

One can verify that the probability $p(\mathbf{m}_k)$ for drawing such a map from all possible 1D maps decreases exponentially with an increasing k:

$$p(\mathbf{m}_k) \propto 2^{-k}. \tag{5}$$

Let now $\mathbf{m}_k^*$ be a 1D map, whose kth cell is hit by a range measurement z. With the above considerations, the expectation in Eq. (3) can be computed in closed form by folding the map $\mathbf{m}_k^*$ with the forward sensor model weighted by the exponential probability distribution of Eq. (5). The expected value $E[\mathbf{m}|z]$ can be interpreted as inverse sensor model:

$$p(r|z) = p(z|r)2^{-r} * \underbrace{\begin{cases} 0, & r < z - \frac{w}{2} \\ 1, & z - \frac{w}{2} \le r < z + \frac{w}{2} \\ 0.5, & z + \frac{w}{2} \le r \end{cases}}_{\mathbf{m}_k^*} \tag{6}$$

where the map $\mathbf{m}_k^*$ is expressed as piecewise-defined function and w denotes the width of each cell of the map. For our used sensors, we assume a Gaussian noise with variance σ^2 and therefore use the normal distribution $\mathcal{N}(z|r, \sigma^2)$ as forward sensor model. This allows us to derive the inverse sensor model according to Eq. (6) as:

$$p(r|z) = h\left(\frac{r-z}{w} + \frac{1}{2}, \frac{\sigma}{w}\right) - \frac{1}{2} h\left(\frac{r-z}{w} - \frac{1}{2}, \frac{\sigma}{w}\right) \tag{7}$$

$$\text{with } h(x, s) = \frac{1}{2} + \frac{1}{2} \operatorname{erf}\left(\frac{s^2 \ln 2 + x}{\sqrt{2}s}\right)$$

where *erf* denotes the Gauss error function. In Fig. 3(a)–(b) the derived inverse sensor model is plotted for different variance parameters σ^2. Apparently, there is a sharp peak in the inverse sensor model if the variance is small, while the sensor model becomes diffuse for larger variances. In the latter case, the occupancy of cells near the measured distance is changed only by a small amount, as a large sensor noise does not allow a precise location of the measured obstacle. Depending on the used sensor, we increase the variance with the distance of the measurement, as the accuracy of the sensors decreases with larger distances. We will come back to this in Section 3.5.

3.4. Maintaining a multi-scale representation

With each new range measurement, the update algorithm performs the above update steps for all affected cells that are associated with the leaf nodes at the deepest level of the underlying 2^d tree. After all cells are updated, the changes are recursively propagated to the parent nodes of those cells and thus to higher levels within the tree. To do so, the mean, covariance and occupancy value of a parent node p is computed from its child nodes i as follows:

$$\boldsymbol{\mu}_p = \sum_i w_i \boldsymbol{\mu}_i, \qquad \boldsymbol{\Sigma}_p = \sum_i w_i \boldsymbol{\Sigma}_i + \beta_i (\boldsymbol{\mu}_p - \boldsymbol{\mu}_i)^\top$$

$$k_p = \sum_i k_i, \qquad l_p = \sum_i l_i \tag{8}$$

where $w_i = k_i / k_p$. This allows us to generate a multi-scale map where each level in the tree represents a different level of detail similar to an image pyramid.

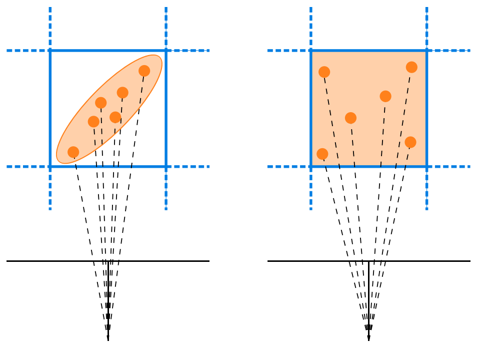

Fig. 4. For updating the occupancy probability of each cell, random points are sampled according to the occupancy distribution and projected onto the image plane of the depth camera. **left:** For occupied cells the points are sampled according to the stored normal distribution. **right:** For empty cells, they are sampled uniformly.

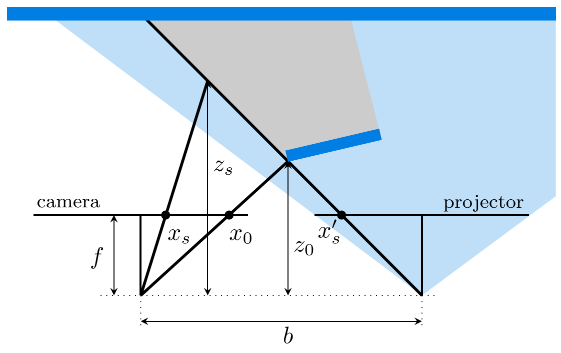

Fig. 5. Top view of the camera and projector geometry of the used Kinect depth camera with base distance b and focal length f. The plane in the foreground casts a shadow on the wall in the background. The depth of the shadow volume z_s at the pixel x_s can be computed using the last valid depth measurement z_0 at pixel x_0.

3.5. Depth-image sensor frontend

After having described the operations and the sensor model that are implemented in a generic mapping backend, we now exemplarily describe a sensor frontend that is specialized for processing depth images obtained using a range sensor such as the Microsoft Kinect. We will describe how the range data is processed in order to invoke the operations for updating the map, that are provided by the backend. Frontends for other sensors such as laser range finders are implemented in a similar way.

Updating occupied cells is trivial. Knowing the intrinsic camera parameters of the depth sensor, for each pixel of the depth image the corresponding 3D position $\mathbf{x}$ in the scene, i.e. the endpoint of the measurement, can be computed. Finally, the cell, where this point is located in, is updated as described in Section 3.2.

Updating the occupancy of cells along the sensor beam as indicated in Section 3.3 is more complex. Other mapping approaches like [8,21] traverse the cells along the measurement ray via ray casting. However, as ray casting is a complex operation in a tree-based map structure, this can be very time-consuming, especially for dense range data, where many rays pass through the same cells. For this reason we use a contrary approach. Instead of spreading out rays from the image plane, we project the cells of the map onto the image plane of the depth camera.

For each cell, we sample random points according to the probability distribution of the occupancy in that cell. For occupied cells this distribution is represented by the stored Gaussian of the Normal Distribution Transform. For empty cells, we assume a uniform distribution as stated above. Each sampled point is projected to the depth image as shown in Fig. 4. At the projected position, we obtain the depth measurement z directly from the depth image. Furthermore, we compute the distance r between the sample point and the sensor. Both values are then used to compute the probability $p(r|z)$ of the cell being occupied given the depth measurement using the inverse sensor model in Eq. (7) and to finally update the overall occupancy probability of the cell according to Eq. (2).

As previously indicated, the variance parameter σ^2 of the inverse sensor model, which models the sensor noise, is increased with the measured distance z, since the accuracy of the sensors decreases with a larger distance.

For laser range finders and time-of-flight cameras the standard deviation of the sensor noise increases linearly with the measured depth. However, for stereo cameras and the Kinect depth camera the standard deviation σ increases with the square of the depth z. We therefore use the following model for the sensor uncertainty:

$$\sigma(z) = \sigma_0 z^2$$

with $\sigma_0 = 0.02$ cm. More sophisticated noise models and calibration methods can be found in [28] or [29].

For some pixels in the depth image, the depth camera cannot provide any depth measurements, e.g. since the measured objects are located too far away. If a sample point is projected onto such an erroneous measurement, we update the corresponding cell using a separate and constant sensor model: $p(r|z) = 0.4$. This model decreases the occupancy probability by a very small amount. The assumption here is, that the affected cell is free, otherwise an object would have been measured by the sensor. This seems to be a strong assumption at first glance but holds well in real applications and is essential to fade out dynamic objects that move in the foreground while the distance of the background cannot be measured. Otherwise, such dynamic objects would reside as artifacts and disallow to use the maps for robot navigation.

Invalid measurements can also have a second cause that is related to the measurement method of the Kinect, where a projector emits a structured light pattern that is captured by a camera (see Fig. 5). If there is a large depth discontinuity in the scene, the foreground object may cast a shadow in the structured light beam. Consequently, for no object within this shadow volume a valid depth estimate can be obtained. This is shown in Fig. 5, where a plane in the foreground casts a shadow on the wall in the background. In the area of the shadow, no depth estimates can be obtained. This needs to be distinguished from the case where no estimate is possible due to a too large distance. Otherwise, we would slowly update the cells on the wall within the shadow volume as free which would result in holes in these areas. For each pixel with no valid measurement, we therefore compute the depth, at which the shadow volume starts, that is cast by a foreground object. This is done in a preprocessing step for each scan line. The processing is performed from right to left, since the shadows are always cast to the left due to the arrangement of the projector and camera. With the help of Fig. 5 and by using the intercept theorem one can verify, that in pixel x_s the casted shadow starts at a depth of:

$$z_s = \frac{z_0 bf}{z_0 x_s - z_0 x_0 + bf} \tag{9}$$

where x_0 and z_0 are the pixel and the corresponding depth of the last valid measurement on that line. b is the base distance between the projector and the camera, and f denotes the focal length.

Knowing the depth of this "shadow border", we can safely update cells in front of it as free. We therefore slightly modify the inverse sensor model in Eq. (7). Instead of folding the 1D map given in Eq. (4) with the forward sensor model and an exponential distribution, we use a map where the first k cells are empty, with k being the cell where the shadow volume starts:

$$\mathbf{m}_k = (\underbrace{0, 0, 0, \ldots, 0}_{k \text{ times}}, 0.5, 0.5, 0.5, \ldots). \tag{10}$$

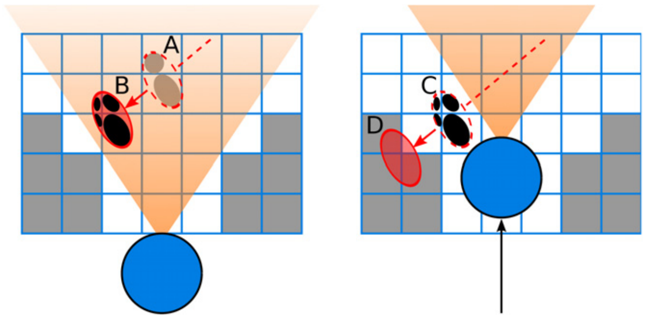

Fig. 6. A person (red ellipse) crosses the field of view (orange) while the robot (blue circle) is moving forward. **left:** The cells near the person's previous position are correctly seen and updated as free (A) while the cells at the person's current position are occupied (B). **right:** Due to the robot's forward movement the person's previous position (C) is no longer within the field of view and the corresponding cells are not updated as free although the person has moved to a different position (D). (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

This leads to the following inverse sensor model:

$$p(r|z_s) = \frac{1}{2} h\left(\frac{r - z_s}{w} + \frac{1}{2}, \frac{\sigma}{w}\right) \tag{11}$$

which only gives evidence for free space as its probabilities stay below or equal to 0.5. Cells within the shadow volume are not affected. This free space sensor model is plotted in Fig. 3(c).

4. Detection and tracking of moving objects

In the previous section, we have described our hybrid NDT occupancy mapping approach that can be used in dynamic environments as it explicitly models free space and therefore can clear moving objects from the map when the corresponding region is observed as free by the sensors. However, in dynamic environments problems can still arise if sensors with a narrow field of view are used, such as stereo or other depth cameras. Fig. 6 shows an example where a person moves through the narrow field of view of the robot while the robot is moving itself. As long as the person stays within the field of view, cells occupied by the person in previous time steps are correctly seen and updated as free (A) when the person moves on. However, when the robot moves forward, the region that was occupied by the person may move out of the sensor's field of view. Hence, the corresponding cells cannot be seen and updated as free (C), although the person has moved to a different location (D). This results in artifacts that remain in the map.

When the mapping is used for creating local maps for obstacle avoidance and local path planning, these artifacts result in unnecessary and spurious avoidance movements of the robot. Furthermore, these artifacts are also unwanted when creating maps of the environment using the SLAM approach, described in the next section of this paper.

We therefore implemented a method to detect and track moving objects in order to propagate their occupancy probability mass even if the objects left the sensor's field of view and to eventually fade them out if necessary.

4.1. Detecting dynamic map content

Beside the occupancy probability $o_c = p(c = occ)$ described in the previous section, we additionally store a second occupancy probability $s_c = p(c = socc)$ for each cell c, that expresses the probability of the cell being statically occupied.

The first occupancy probability o_c can be thought of as a short-term occupancy state that is estimated using the recent sensor readings, and it may quickly change for regions with moving objects or persons. The static occupancy probability s_c is adapted slowly and represents the long-term occupancy state. This probability will be high for occupied cells that represent objects that belong to the static environment, such as walls and furniture.

Just like the occupancy probability o_c we also represent the static occupancy s_c using log-odds and update its value similar to Eq. (2):

$$\operatorname{logit}(s_c') = \operatorname{logit}(s_c) + \operatorname{logit}(p(c = socc|c = occ)) \tag{12}$$

where $p(c = socc|c = occ)$ gives the probability whether the cell is statically occupied when the current short-term occupancy state is given by o_c. Currently, we use the following ad hoc model to compute this probability:

$$p(c = socc|c = occ) = \begin{cases} \frac{1}{2} + \left(o_c - \frac{1}{2}\right) c_1, & o_c \le \frac{1}{2} \\ \frac{1}{2} + \left(o_c - \frac{1}{2}\right) c_2, & o_c > \frac{1}{2}. \end{cases} \tag{13}$$

The constants $c_1 < 1$ and $c_2 < 1$ control how fast the static occupancy probabilities are adapted. The smaller the constants, the slower is the update of the long-term occupancy state. In our experiments, we choose c_1 to be about 5 times larger than c_2 in order to achieve a stronger influence of free space evidence on the long-term occupancy state.

Having estimated the (short-term) occupancy probability o_c and the long-term occupancy probability s_c, we use both values to classify each cell as belonging to the static (S) or dynamic (D) environment as follows:

$$h : c \mapsto \{S, D\}$$

$$\text{with} \quad h(c) = \begin{cases} D, & o_c - s_c > \theta \\ S, & \text{otherwise} \end{cases} \tag{14}$$

with θ being a threshold that we set to 0.6. Hence, a cell is classified as "dynamic cell" if the cell is free according to the long-term occupancy probability but is currently observed as being occupied, e.g. when a person is moving through a region that was previously observed free.

4.2. Tracking moving objects

After the cells have been classified as dynamic or static, neighboring and connected dynamic cells are fused to hypotheses of dynamic objects using a bottom-up clustering algorithm, that starts at the leaf node and recursively fuses neighboring dynamic cells to hypotheses. Each such hypothesis h corresponds to a moving object whose shape is approximated by a normal distribution $\mathcal{N}(\mathbf{p}_h, \mathbf{C}_h)$.

For each hypothesis, we estimate the object's position $\mathbf{p}_h$ and its velocity $\mathbf{v}_h$ using a linear Kalman Filter. Hence, the augmented state to be estimated is given by $\mathbf{x}_h = (\mathbf{p}_h^\top, \mathbf{v}_h^\top)^\top$ and its corresponding covariance matrix $\mathbf{\Sigma}_h$. In the next time step, each hypothesis is predicted using a linear motion model:

$$\hat{\mathbf{x}}_h = \mathbf{F}\mathbf{x}_h + \mathbf{w}_h, \quad \text{where } \mathbf{w}_h \sim \mathcal{N}(0|\mathbf{Q}) \tag{15}$$

with

$$\mathbf{F} = \begin{bmatrix} \mathbf{I} & \Delta t\mathbf{I} \\ \mathbf{0} & \mathbf{I} \end{bmatrix} \quad \text{and} \quad \mathbf{Q} = \begin{bmatrix} \frac{\Delta t^4}{4}\mathbf{I} & \frac{\Delta t^3}{2}\mathbf{I} \\ \frac{\Delta t^3}{2}\mathbf{I} & \Delta t^2\mathbf{I} \end{bmatrix} \tag{16}$$

and $\mathbf{I}$ being the $d \times d$ identity matrix, where d denotes the dimension of the map. Furthermore, Δt is the time interval that has elapsed since the last Kalman Filter update.

After the positions of the dynamic object hypotheses have been predicted, they are tracked by matching each predicted hypothesis from the previous time step with the closest hypothesis that was clustered in the current time step. The position of the new hypothesis is taken as measurement to perform the Kalman Filter update. This allows us to track moving objects and to estimate their position and velocities iteratively.

4.3. Propagating occupancy of moving objects

As each tracked hypothesis corresponds to a cluster of dynamic cells that are occupied by a moving object, we use them for propagating the occupancy probability of these cells according to the object's motion. We do this by modifying the prediction step of the binary Bayes Filter that provides the underlying theoretical framework for the occupancy update given in Eq. (2).

Usually, the occupancy state of each cell is assumed to be constant over time and independent from other cells. However, given the above motion hypotheses and the occupancy probabilities of the neighboring cells, we can compute an a priori occupancy probability that is injected by a moving object into each cell. The general idea is to collect the occupancy probability of the moving cell clusters, represent it by the normal distribution of the corresponding object hypothesis, move it according to the above motion model, and distribute it back into the cells of the map.

For each hypothesis, we therefore sum the occupancy probability of all cells that belong to the cluster represented by the hypothesis. This sum s_h is stored with each hypothesis h. It resembles the total occupancy probability mass that is "carried" by the moving object. In the next time step, the positions of the hypotheses are predicted as specified in Eq. (15) using the estimated velocities of the objects. This moves the occupancy probability of each hypothesis to neighboring cells where it is redistributed to the cells.

This is done after the "shape update" of the NDT map has been computed using the current sensor readings according to Section 3.2 to ensure that the Gaussians in the cells, where the object is located now, are already adapted to the new object location.

For each cell c, that is covered by a dynamic object hypothesis h, we compute the cell's occupancy probability o_c from the summed occupancy mass s_h of the hypothesis as follows:

$$o_c = \eta w_{c,h} s_h \tag{17}$$

where η denotes a normalization factor, and the weight $w_{c,h}$ corresponds to the proportion the Gaussian of the cell c takes up in the object hypothesis h, which itself is represented by a Gaussian as mentioned above. The weight can be computed as the integral of the product of both Gaussians:

$$\begin{aligned} w_{c,h} &= \int \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_c, \boldsymbol{\Sigma}_c)\mathcal{N}(\mathbf{x}|\hat{\mathbf{p}}_h, \mathbf{C}_h + \hat{\boldsymbol{\Sigma}}_h^p)d\mathbf{x} \\ &= \mathcal{N}(\boldsymbol{\mu}_c|\hat{\mathbf{p}}_h, \boldsymbol{\Sigma}_c + \mathbf{C}_h + \hat{\boldsymbol{\Sigma}}_h^p) \end{aligned} \tag{18}$$

where $\boldsymbol{\mu}_c$ and $\boldsymbol{\Sigma}_c$ describe the Gaussian of the NDT cell, $\hat{\mathbf{p}}_h$ is the predicted position of the moving object and $\hat{\boldsymbol{\Sigma}}_h^p$ the corresponding covariance matrix which is also computed in the prediction step of the Kalman Filter and expresses the uncertainty in the position estimate. The covariance matrix $\mathbf{C}_h$ describes the shape of the moving object as mentioned above. The normalization factor is computed as $\eta = {}^1/_{\sum_c w_{c,h}}$ to ensure the normalized weights sum up to 1.

After the a priori occupancy probabilities for the cells, covered by the predicted dynamic object hypotheses, have been computed according to Eq. (17), the occupancy probabilities are updated using the sensor measurements as described in Section 3.3.

Although it is a coarse approximation to represent the overall occupancy probability of a dynamic cell cluster by a single Gaussian, it has proven to work well in our tests. In contrast to other approaches such as [30,31], where the transition of the occupancy probability between neighboring cells is estimated and computed at cell level, our approach can be computed very efficiently and only adds a very small computational overhead.

When using the proposed methods for detecting and handling moving objects, the problems and artifacts that are caused by these objects near the borders of the sensor's field of view as described at the beginning of this section, can be resolved. If a moving object leaves the field of view, its movement will be further propagated according to Eq. (15). As there will be no further observation of the object, the covariance matrix $\hat{\boldsymbol{\Sigma}}_h^p$, i.e. the uncertainty of the position estimate, will increase continuously as more and more uncertainty is added by the matrix $\mathbf{Q}$ in Eq. (16). As a result the object hypothesis will become increasingly diffuse and its occupancy will be distributed across more cells and will eventually fade out.

5. Lifelong SLAM

After having described our mapping approach in detail, we will now show how the above algorithms can be integrated into a SLAM approach. Similar to [11], we use a graph-based formulation of the SLAM problem, which models the poses $\mathbf{x}_{1:n}$ of the robot's trajectory as vertices $v_{1:n}$ of a pose graph. Constraints between two poses of the trajectory that typically arise from odometry, sensor measurements, and loop-closures are stored within edges between the corresponding vertices. Each constraint between two vertices v_i and v_j is represented by a transformation $\boldsymbol{\delta}_{ji}$ that describes the pose $\mathbf{x}_j$ as seen from $\mathbf{x}_i$ and a corresponding information matrix $\boldsymbol{\Omega}_{ji}$.

The SLAM problem can then be described by the following optimization problem:

$$\mathbf{X}^* = \underset{\mathbf{X}}{\operatorname{argmin}} \sum_{i,j} \mathbf{e}(\mathbf{x}_i, \mathbf{x}_j, \boldsymbol{\delta}_{ji})^\top \boldsymbol{\Omega}_{ji} \mathbf{e}(\mathbf{x}_i, \mathbf{x}_j, \boldsymbol{\delta}_{ji}) \tag{19}$$

with $\mathbf{X} = (\mathbf{x}_1^\top, \ldots, \mathbf{x}_n^\top)^\top$ being the vector of the pose estimates of all vertices in the pose graph. The error function $\mathbf{e}(\mathbf{x}_i, \mathbf{x}_j, \delta_{ji})$ measures how well the pose estimates $\mathbf{x}_i$ and $\mathbf{x}_j$ satisfy the constraint δ_{ji} [17].

For solving the above optimization problem, we use *g2o* [17], an open source framework for graph optimization. This optimization step of a graph-based SLAM algorithm is also known as SLAM-backend. The result of the optimization process is the most consistent set of pose estimates $\mathbf{x}^*_{1:n}$ that represent the robot's trajectory.

In this paper, we focus on the SLAM-frontend, which is responsible for generating the pose graph with its vertices and constraints. In the following, we give an overview of our approach shown in Fig. 7, before we discuss its components in more detail.

In our approach, each vertex additionally stores an NDT map fragment which is a small piece of the overall map. During the robot's locomotion the previously described mapping algorithm incrementally integrates the range sensor measurements into the current map fragment, i.e. we combine multiple sensor readings in a single vertex of the pose graph. This is necessary since the single measurements of range sensors with a low measurement range or small field of view, like depth cameras, do not allow to perform loop closures robustly. The position estimates that are necessary for the mapping are obtained from the robot's odometry. Since the odometry is erroneous, this will of course induce a small error in the created map fragment. This error grows continuously with the covered distance. For this reason we cut the map fragment and start a new one whenever the uncertainty in the robot's movement exceeds a certain threshold to limit the effects of the odometry errors in the built map fragment. The uncertainty of the robot's movement is computed using a probabilistic motion model similar to the one described in [27] but approximated using normal distributions.

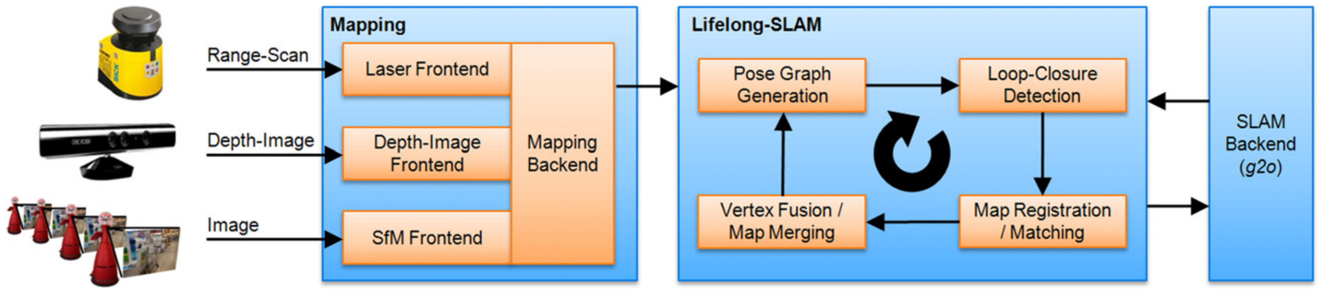

Fig. 7. The flow diagram of our complete mapping pipeline with the mapping component consisting of the mapping backend and different sensor frontends as well as the components of the actual SLAM approach.

With each new map fragment, a new vertex v_n is added to the pose graph. The new vertex is connected with the previously added one v_{n-1} by an edge that stores the robot's relative movement from that vertex. The corresponding covariance is taken from the probabilistic motion model.

Afterwards, our approach checks for potential loop closure candidates (see Fig. 7, top right). Therefore, it propagates the uncertainties of the pose estimates through the pose graph starting at the newly added vertex. This is done similarly to the belief propagation in a Bayesian network along the minimum spanning tree with v_n as root node. As the result of this process, the marginalized covariance of each pose estimate is known relatively to the newly added vertex. This covariance is used in a χ^2 test to determine if the map of another vertex is most likely to overlap with the map of the newly inserted vertex. In this case, a loop-closure edge is created between the new vertex v_n and the loop-closure candidate v_m by aligning the two overlapping maps M_n and M_m using different map registration algorithms to obtain the relative pose of the two vertices.

According to [10], we increase the robustness by weighting the error function of a loop-closure constraint with a switch variable ω_{ji} that controls the influence of that constraint. Beside the pose vertices, the switch variables are also adapted within the graph optimization procedure and hence allow to disable erroneous constraints. Consequently, there is no need to perform any kind of outlier rejection in order to remove wrong map registration results. Instead, the invalid loop-closure constraints will be switched off automatically during the pose graph optimization.

After all loop closure candidates were processed and the pose graph was updated respectively, the SLAM backend is run to optimize the pose graph while taking the newly added poses and constraints into account in order to update the estimated poses of the robot's trajectory. The entirety of all map fragments at the estimated poses constitutes the complete map. Thus, unlike other SLAM algorithms, our approach does not need to perform an additional mapping pass to integrate all sensor readings into a map using the corrected pose estimates. If a single map instead of the map fragments was required, all map fragments could be easily merged. Consequently, no sensor readings need to be stored — the approach is completely online. Furthermore, at this point a fresh estimate of the robot's current location within the environment is generated and can be directly used by navigation algorithms such as path planners. Afterwards, the whole process starts again by adding the next vertex containing the next map fragment that was created in the meantime.

5.1. NDT map registration of loop closure candidates

For the alignment of the two NDT maps, we use two different registration algorithms. The general idea of the map registration is closely related to [9] and therefore only briefly described here.

We try to minimize a distance metric between the two NDT maps M_n and M_m which is defined as:

$$d(M_n, M_m, \delta_{nm}) = \sum_{i \in M_n} \mathbf{d}_{ij}^\top (\mathbf{R}_{nm}^\top \mathbf{\Sigma}_i \mathbf{R}_{nm}^\top + \mathbf{\Sigma}_j)^{-1} \mathbf{d}_{ij} \tag{20}$$

with $j = \operatorname{argmin}_{j \in M_m} \|\boldsymbol{\mu}_i - \boldsymbol{\mu}_j\|$ and $\mathbf{d}_{ij} = (\mathbf{R}_{nm}\boldsymbol{\mu}_i + \mathbf{t}_{nm} - \boldsymbol{\mu}_j)$. This metric sums the pairwise distances between each normal distribution $\mathcal{N}_i$ of map M_n and its closest neighbor distribution $\mathcal{N}_j$ of map M_m. This is closely related to the Iterative Closest Point (ICP) algorithm. We will come back to this point later. In the above equation, δ_{nm} denotes a transformation consisting of a rotation $\mathbf{R}_{nm}$ and a translation $\mathbf{t}_{nm}$ that transforms all normal distributions $(\boldsymbol{\mu}_i, \mathbf{\Sigma}_i)$ of map M_n and therefore the whole map into the reference frame of map M_m. Consequently, δ_{nm} is the desired transformation that relates the two vertices v_n, v_m of a loop closure.

In the registration process the transformation δ_{nm} is varied to minimize the distance metric iteratively. The initial estimate of this transformation is taken from the current pose estimates $\mathbf{x}_n$ and $\mathbf{x}_m$ of the two loop close vertices in the pose graph.

If the initial estimate has a small uncertainty, i.e. if the propagated covariance of the loop-closure candidate is below a threshold, we use the Levenberg–Marquardt (LM) method for solving the non-linear minimization problem. Since NDT maps are piecewise differentiable, the necessary derivatives can be computed as described in [9]. The LM method works fine if the initial estimate of the transformation is near the correct value. Otherwise, it tends to converge to local minima that do not reflect the correct transformation.

In these cases, where the uncertainty in the initial estimate is large, e.g. since the robot has covered a large distance without performing a single loop-closure, we use Particle Swarm Optimization (PSO) [32] to solve the above minimization problem. As score function of the particles, the distance metric is used directly.

To speed up the map registration we take advantage of our multi-scale NDT maps. Instead of using the finest level of detail, the above optimization algorithms operate on a coarser level of the maps with typical cell sizes of 0.4 m. This considerably reduces the overall computational complexity. After the registration on the coarse level has converged, we perform a final second fine-grained registration step using the LM method on the finest map level with cell sizes of 0.05–0.1 m to achieve the highest possible precision.

5.2. Normal space sampling

As stated before, the map registration and the computation of the distance metric in Eq. (20) is closely related to the Iterated Closest Points (ICP) algorithm, that is used for the registration of two point clouds. ICP associates the points of two point clouds also using a nearest neighbor criterion.

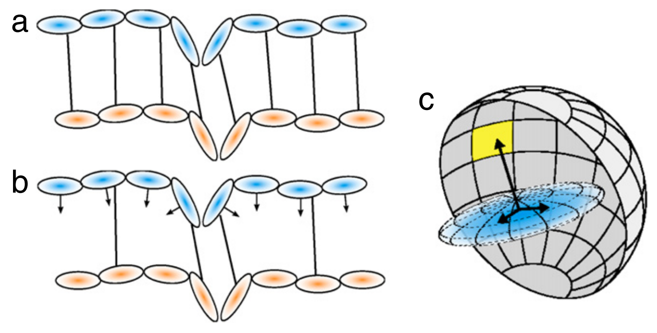

Fig. 8. (a) If the normal distributions for the pairing during the map registration are chosen uniformly, small features are suppressed. (b) If *normal space sampling* is used, the normal distributions around the feature have a significant influence. (c) The normal distributions are partitioned in a histogram according to the direction of their normal vector. Afterwards, they are sampled uniformly from the bins that correspond to the indicated regions on a half-sphere.

To improve the robustness of our map registration, we adapt a variant of the ICP algorithm that is known as *normal space sampling* [33]. Therefore, we slightly modify Eq. (20). Instead of computing the sum of the pairwise distances for all normal distribution $\mathcal{N}_i$ of map M_n, we sample a subset $S_n \subseteq M_n$ of these normal distributions. This further reduces the computational complexity as the number of pairings decreases. More importantly, it allows to choose a subset that leads to a better convergence of the map registration.

To find the correct alignment of the map, small features such as small bulges in a flat hallway (see Fig. 8) can be essential. However, if a uniform sampling scheme is used or all available distributions are taken into account, the influence of these small features in the overall costs of Eq. (19) are marginal (Fig. 8(a)). A stronger influence of these features would be desirable as their NDT representations have a different orientation than the normal distributions in the other cells. This gives an important directional information during the iterative registration process that leads to a better convergence if exploited.

For this reason, we compute the orientation of each normal distribution $\mathcal{N}_i$ of map M_n in terms of its "normal vector" $\mathbf{n}_i$. As normal vector we use the eigenvector of the covariance matrix $\mathbf{\Sigma}_i$ that corresponds to the smallest eigenvalue. Hence, the vector points into the direction of the smallest extend of the normal distribution which corresponds to the normal of the represented surface. We represent each normal vector in the angular space of a spherical coordinate system by using two angles (altitude and azimuth) in the case of 3D maps or a single angle in the case of 2D maps. This allows us to put each normal distribution $\mathcal{N}_i$ of map M_n into a histogram according to the angles of its normal vector. This procedure is shown in Fig. 8(c) where it also becomes apparent that the bins of the histogram correspond to the indicated cells on the surface of a half-sphere. Finally, we form the subset S_n of normal distributions by sampling uniformly across the histogram bins. This set is then used to compute the sum in Eq. (20) to ensure, that all orientations are represented equally well (Fig. 8(b)).

5.3. Fusion of vertices for lifelong operation

By adding more and more map fragments and vertices the size of the pose graph increases over time, and consequently, the memory and computational complexity rises. To be able to use the proposed SLAM approach over a long time period within a robotic application we therefore need to prune the pose graph to reduce the number of vertices.

This is done after each graph optimization step. Vertices whose map fragments cover a similar region of the environment are fused. However, merging vertices and their corresponding map fragments poses the risk of consolidating inconsistencies and inaccuracies. Therefore, only those vertices are merged whose relative position between each other is known with a very high certainty, e.g. if many successful loop closures were performed between the vertices or their neighbors. To measure this uncertainty, we again use the propagated covariance that was already computed to identify possible loop closure candidates. Two vertices v_i and v_j are merged, if the propagated covariance from one vertex to another is below a threshold. Without loss of generality we assume that the vertex v_j was added after v_i. The fusion of the two vertices is done by merging the information of v_j into v_i and removing the vertex v_j from the pose graph. To preserve the global behavior of the SLAM error function (19) while removing vertex v_j, we would need to add edges for each pair of neighbors of v_j [11]. However, this would result in an increasing complexity of the pose graph. To avoid this problem, we use the same approximation that was proposed in [11] and connect all neighbors of v_j to v_i while adapting the information stored in these edges as described in [11]. With this approximation, removing v_j and all of its edges will then reduce the number of vertices and edges at least by 1. During the fusion of both vertices, the map fragment of v_j is also merged into the map of vertex v_i. This is done according to Eq. (8). Since v_j was added to the pose graph after v_i, its map contains the more recent information on the environment. Merging the map fragments in the proposed order therefore ensures, that all map fragments are up-to-date and that our approach is able to adapt to changes in the environment.

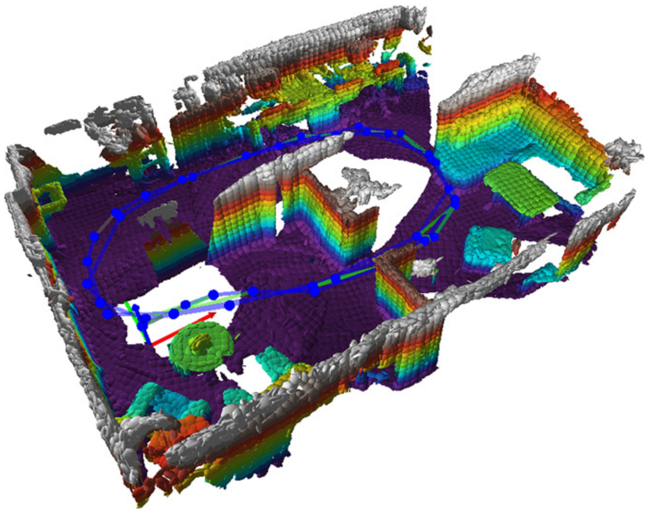

Fig. 9. 3D NDT map created using a Microsoft Kinect while driving three loops through a home environment. Each normal distribution of an NDT cell is shown as ellipsoid and colored according to its height. The pose graph with its vertices and edges is indicated in blue. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

6. Results

We have tested our approach in different environments using different sensors.

Fig. 9 shows a 3D map that was build using the Microsoft Kinect depth camera mounted on our home assistance robot while driving three loops through a narrow home environment with tables, armchairs in the lower left corner and a couch in the right corner. In the NDT map, the normal distributions of cells with an occupancy probability larger than 0.8 are shown as shaded ellipsoids. The colors correspond to the height of the cells. The mapped area has a size of 5×8 m^2.

In Fig. 10(a) 2D map is shown that was created using a laser range finder while driving our tour guide robot manually through

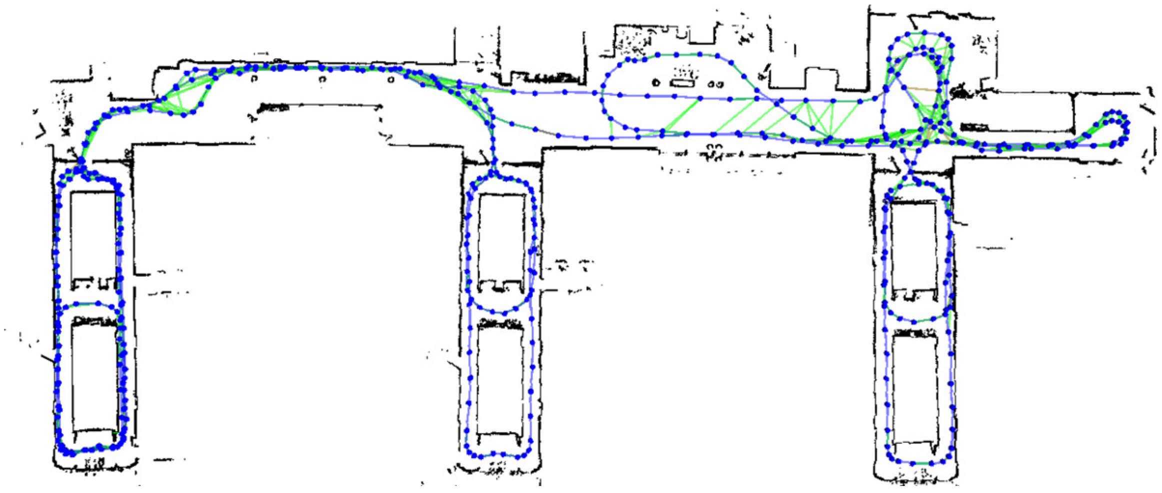

Fig. 10. 2D NDT map created using a laser range finder while driving through an office building. The pose graph with its vertices and edges is indicated in blue. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

an office building. The mapped area has a size of 80 × 35 m^2. The normal distributions of the cells are shown as black small ellipses that are merely larger than dots on this scale. Each cell of the map has a resolution of 0.1 m. However, due to the benefits of the employed NDT maps, the effective resolution is much higher and allows to represent single chair legs and twigs of plants that were standing around in the mid-right area of the map. While creating this map, the robot covered a distance of 700 m. The corresponding trajectory and the generated pose graph is depicted in blue color. Due to the generic implementation no changes in the algorithms were necessary for creating the 2D map and the above 3D map.

6.1. Performance

We tested the presented approach on a machine running on an Intel Core i7, 2.70 GHz CPU which is identical to the hardware we use on our robots.

The insertion of the 2D or 3D range data into the NDT map only takes 10–20 ms. The map registration during a loop-closure is the computationally most expensive part of the presented approach. As described above, this step needs to be performed for each inserted vertex and map fragment. Including all loop-closures, the average computation time when inserting a new map fragment with a cell resolution of 0.1 m is 300 ms for 2D maps and 500 ms for 3D maps when running on a single CPU core. Depending on the driving speed of the robot, a new map fragment is available every 500–1000 ms. Consequently, our approach is able to process the incoming data in real-time for both 2D and 3D maps. It is therefore suitable for online localization and mapping.

6.2. Lifelong operation

To test its ability for long term operations in a bounded environment, we ran our approach for two days on one of our guide robots [34] that operate in our office building on a daily basis to autonomously guide visitors within the building. On a regular office day, when the robots typically operate for about 6 h, each robot travels up to 4000 m.

During the two day test period, the total length of the driven trajectory was 7000 m which corresponds to 3 h of continuous driving. The tests were restricted to a single floor of the building that is shown in Fig. 10. Since the guide robots are equipped with a laser range finder only, this long term test was used to test the 2D variant of our approach.

In the left diagram of Fig. 11 the number vertices of the pose graph are plotted against the driven distance during the overall test run. The solid blue line corresponds to the number of vertices in the pose graph for our proposed lifelong approach. During the first 1000 m the number increases up to 400 vertices while the robot explores most of its environment. Afterwards, it remains constant for the rest of the test. After 3000–4000 m the vertex count slightly increases as the robot visits areas of the building it has not seen before. Consequently, more vertices are necessary to cover the environment which now became larger.

For comparison, we turned off the fusion of vertices in another run. The corresponding graph is colored in red. Here, the number of vertices increases indefinitely with the covered distance.

In the above test, we also measured the computational complexity of the SLAM approach. Since the loop closure computations during the insertion of a new map fragment are the most expensive processes, the insertion time for a new map fragment is shown in the right diagram of Fig. 11. Again, the blue line shows the graph for our proposed approach, where the time for inserting a new map fragment remains constant between 200 and 400 ms. If the fusion of vertices is disabled (the red graph), the insertion time grows significantly with the increasing number of vertices, since much more loop-closure candidates need to be processed if there is a high density of vertices.

A second long run was performed using a smaller robot platform within a typical home environment shown in Fig. 9. In this test, a 3D map was created using a Microsoft Kinect sensor. Although the driven distance of 250 m was much smaller in this test, it is far sufficient to cover the whole home environment exhaustively. The total number of loops was 12. The vertex count and the time for computing the loop-closures are shown in Fig. 12. Again, the vertex count and the performance stays constant over time for our proposed approach.

These tests reveal both the temporal and spacial scalability of the approach. The number of vertices within the pose graph depends on the size of the operational area only and stays constant in bounded environments. Moreover, the complexity of the loop-closure computation is similar in small and large environments, as it only depends on the density of the pose graph vertices. Therefore, it will not increase significantly even if the approach operates in larger environments or for a longer period of time.

6.3. Dynamic environments

We also tested our approach in dynamic and semi-static environments. Fig. 13 shows a sequence of four images where a person moves through the field of view of the used Kinect sensor while the robot is moving forward. The cells that represent the

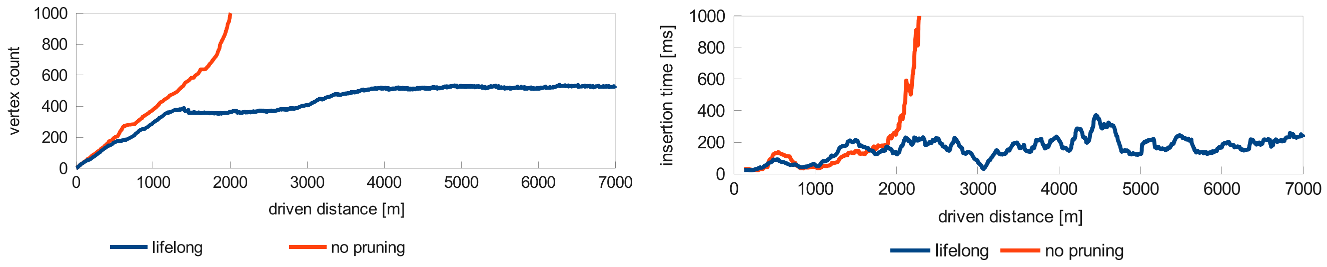

Fig. 11. left: Number of vertices in the pose graph during a 2 day long term test for the proposed lifelong SLAM approach (blue) and without fusing vertices (red). **right:** Time for inserting a new map fragment and a new vertex into the pose graph during the test. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

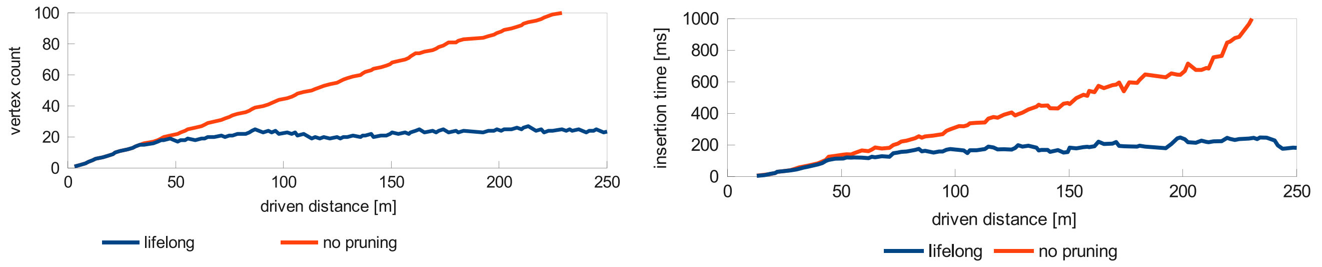

Fig. 12. left: Number of vertices in the pose graph while creating a 3D map in a small home environment with pruning (blue) and without fusing vertices (red). **right:** Time for inserting a new map fragment and a new vertex into the pose graph during the test. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

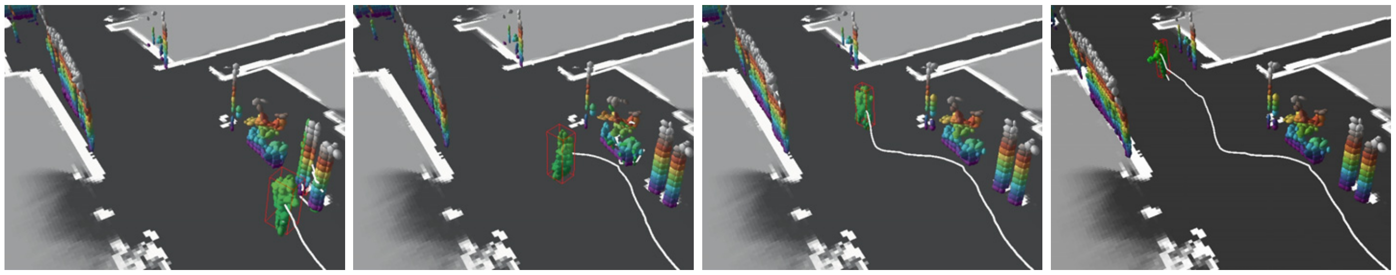

Fig. 13. While the robot was moving through its environment, a person walked through the field of view of the Kinect sensor. The cells that are classified as dynamic are colored in green. The clustered moving person is marked by a red bounding box and the track of the person is shown as white line. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

person are correctly classified as dynamic (shown in green) while the surrounding obstacles are static. The person is clustered as a single moving object, which is indicated by the red bounding box. The person is tracked through the complete sequence. The track is shown as white line in the images. Moreover, the images show that the person does not cause any artifacts as the occupancy is correctly propagated according to the movement of the person.

Beside highly dynamic objects like persons, the SLAM approach is also robust against changes in the environment. This was tested in the home environment as shown in Fig. 14. During the mapping we moved two armchairs and a table to the opposite corner of the room. Despite these changes, the approach is able to keep the correct location within the environment using other parts of the room that remained unchanged. In those parts of the room, enough map fragments can still be matched successfully to establish correct loop closures. After vertices of the pose graph that contain old and new map fragments are merged, the changes in the environment become apparent in the map as shown in the right image of Fig. 14. Thus, the approach is able to adapt to the changed environment.

7. Conclusion

In this paper, we have presented a novel mapping technique for NDT maps that combines NDT mapping and occupancy mapping in a probabilistic sound way. We described an extension that is able to

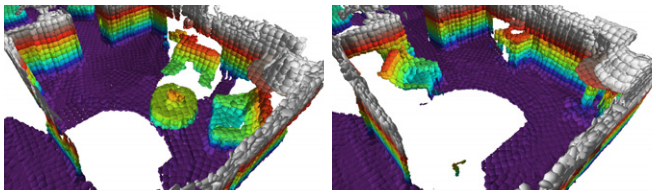

Fig. 14. While the robot was mapping its environment, the table and the armchairs visible in the left image, were moved to the opposite side of the room as shown in the right image.

detect and handle dynamic objects like moving persons. The proposed mapping approach is modular and independent from the dimensionality of the created maps. It allows to implement different mapping frontends for different sensors. Exemplarily, we have presented our mapping frontend for Microsoft Kinect depth images. The generated NDT maps are an abstraction of the underlying sensor data and allow to develop a sensor-independent SLAM approach that was presented as a second contribution of this paper. In the results we have shown that the presented algorithms are able to generate 3D and 2D maps of different, complex environments in real-time. Furthermore, we have shown that this performance remains constant over the whole operation time and therefore allows to apply the approach as lifelong SLAM and localization technique in real-world applications.

Several videos of the proposed approaches are available at: http://www.youtube.com/neurobTV.

References

[1] C. Schröter, H.-M. Gross, A sensor-independent approach to RBPF SLAM—Map Match SLAM applied to visual mapping, in: Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS, 2008, pp. 2078–2083.
[2] U. Frese, R. Wagner, T. Röfer, A SLAM overview from a user's perspective, Künstliche Intel. 24 (2010) 191–198.
[3] D. Fox, KLD-sampling: adaptive particle filters, in: Advances in Neural Information Processing Systems, vol. 14, MIT Press, 2001.
[4] H.-M. Gross, H.-J. Böhme, C. Schröter, S. Müller, A. König, E. Einhorn, C. Martin, M. Merten, A. Bley, Interactive shopping guide robots in everyday use—final implementation and experiences from long-term field trials, in: Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS, 2009, pp. 2005–2012.
[5] H.-M. Gross, C. Schröter, S. Müller, M. Volkhardt, E. Einhorn, A. Bley, C. Martin, T. Langner, M. Merten, Progress in developing a socially assistive mobile home robot companion for the elderly with mild cognitive impairment, in: Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS, 2011, pp. 2430–2437.
[6] E. Einhorn, C. Schröter, H. Gross, Can't take my eye off you: Attention-driven monocular obstacle detection and 3D mapping, in: Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS, 2010, pp. 816–821.
[7] J. Saarinen, H. Andreasson, T. Stoyanov, J. Luhtala, A. Lilienthal, Normal distributions transform occupancy maps: application to large-scale online 3D mapping, in: Proceedings of the IEEE International Conference on Robotics and Automation, ICRA, 2013, pp. 2225–2230.
[8] J.P. Saarinen, H. Andreasson, T. Stoyanov, A.J. Lilienthal, 3D normal distributions transform occupancy maps: an efficient representation for mapping in dynamic environments, Internat. J. Robot. Res. 32 (14) (2013) 1627–1644.
[9] T. Stoyanov, M. Magnusson, A. Lilienthal, Point set registration through minimization of the L2 distance between 3D-NDT models, in: Proceedings of the IEEE International Conference on Robotics and Automation, ICRA, 2012, pp. 5196–5201.
[10] N. Sünderhauf, P. Protzel, Switchable constraints for robust pose graph SLAM, in: Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS, 2012, pp. 1879–1884.
[11] H. Kretzschmar, G. Grisetti, C. Stachniss, Lifelong map learning for graph-based SLAM in static environments, Künstliche Intel. 24 (2010) 199–206.
[12] C.-C. Wang, C. Thorpe, S. Thrun, Online simultaneous localization and mapping with detection and tracking of moving objects: theory and results from a ground vehicle in crowded urban areas, in: Proceedings of the IEEE International Conference on Robotics and Automation, ICRA, 2003.
[13] A.J. Davison, I.D. Reid, N.D. Molton, O. Stasse, MonoSLAM: real-time single camera SLAM, IEEE Trans. Pattern Anal. Mach. Intell. 29 (6) (2007) 1052–1067.
[14] G. Grisetti, C. Stachniss, W. Burgard, Improved techniques for grid mapping with Raoi–Blackwellized particle filters, IEEE Trans. Robotics 23 (1) (2007) 34–46.
[15] M. Kaess, A. Ranganathan, F. Dellaert, iSAM: incremental smoothing and mapping, IEEE Trans. Robotics 24 (6) (2008) 1365–1378.
[16] M. Kaess, H. Johannsson, R. Roberts, V. Ila, J. Leonard, F. Dellaert, iSAM2: Incremental smoothing and mapping with fluid relinearization and incremental variable reordering, in: Proceedings of the IEEE International Conference on Robotics and Automation, ICRA, 2011, pp. 3281–3288.
[17] R. Kummerle, G. Grisetti, H. Strasdat, K. Konolige, W. Burgard, G2o: A general framework for graph optimization, in: Proceedings of the IEEE International Conference on Robotics and Automation, ICRA, 2011, pp. 3607–3613.
[18] H. Moravec, Robot spatial perception by stereoscopic vision and 3d evidence grids. Technical report, Robotics Institute, Pittsburgh, PA, 1996.
[19] P. Payeur, P. Hebert, D. Laurendeau, C. Gosselin, Probabilistic octree modeling of a 3-d dynamic environment, in: Proceedings of the IEEE International Conference on Robotics and Automation, ICRA, 1997.
[20] N. Fairfield, G. Kantor, D. Wettergreen, Real-time SLAM with octree evidence grids for exploration in underwater tunnels, J. Field Robotics (2007).
[21] K.M. Wurm, A. Hornung, M. Bennewitz, C. Stachniss, W. Burgard, OctoMap: a probabilistic, flexible, and compact 3D map representation for robotic systems, in: Proceedings of the IEEE International Conference on Robotics and Automation, ICRA, 2010.
[22] E. Einhorn, C. Schroter, H.-M. Gross, Finding the adequate resolution for grid mapping—cell sizes locally adapting on-the-fly, in: International Conference on Robotics and Automation, ICRA, 2011, pp. 1843–1848.
[23] P. Biber, W. Straßer, The normal distributions transform: A new approach to laser scan matching, in: Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS, 2003, pp. 2743–2748.
[24] M. Magnusson, the three-dimensional normal-distributions transform—an efcient representation for registration (Ph.D. thesis), Örebro University, 2009.
[25] T. Stoyanov, M. Magnusson, H. Almqvist, A. Lilienthal, On the accuracy of the 3d normal distributions transform as a tool for spatial representation, in: Proceedings of the IEEE International Conference on Robotics and Automation, ICRA, IEEE, 2011, pp. 4080–4085.
[26] S. Frisken, R. Perry, Simple and efficient traversal methods for quadtrees and octrees, J. Graphics Tools 7 (2003).
[27] S. Thrun, W. Burgard, D. Fox, Probabilistic Robotics (Intelligent Robotics and Autonomous Agents), The MIT Press, 2005.
[28] S.M. Olesen, S. Lyder, D. Kraft, N. Krüger, J. Jessen, Real-time extraction of surface patches with associated uncertainties by means of kinect cameras, J. Real-Time Image Process. (2012) 1–14.
[29] J.-H. Park, Y.-D. Shin, J.-H. Bae, M.-H. Baeg, Spatial uncertainty model for visual features using a kinect sensor, Sensors 12 (7) (2012) 8640–8662.
[30] T. Gindele, S. Brechtel, J. Schröder, R. Dillmann, Bayesian occupancy grid filter for dynamic environments using prior map knowledge, in: IEEE Intelligent Vehicles Symposium, 2009, pp. 669–676.
[31] S. Brechtel, T. Gindele, R. Dillmann, Recursive importance sampling for efficient grid-based occupancy filtering in dynamic environments, in: Proceedings of the IEEE International Conference on Robotics and Automation, ICRA, 2010, pp. 3932–3938.
[32] J. Kennedy, R. Eberhart, Particle swarm optimization, Proc. IEEE Internat. Conf. Neural Netw. 4 (1995) 1942–1948.
[33] S. Rusinkiewicz, M. Levoy, Efficient variants of the icp algorithm, in: International Conference on 3-D Digital Imaging and Modeling, 2001.
[34] R. Stricker, S. Müller, E. Einhorn, C. Schröter, M. Volkhardt, K. Debes, H. Gross, Interactive mobile robots guiding visitors in a university building, in: RO-MAN, IEEE, 2012, pp. 695–700.

Erik Einhorn is a Ph.D. student at the research lab for Cognitive Robotics at the Ilmenau University of Technology. He received his diploma degree in computer science from the Ilmenau University of Technology in 2007. His research interests are visual 3D perception, environment modeling and mapping for mobile robot navigation.

Horst-Michael Gross is full professor for Computer Science with a focus on Cognitive Robotics at the Ilmenau University of Technology and head of the research lab for Cognitive Robotics. He received his Diploma degree (M.Sc.) in Technical and Biomedical Cybernetics in 1985 and his Doctorate degree (Ph.D.) in Neuroinformatics in 1989 at the Ilmenau University. His research mainly focuses on the development of robust and adaptive techniques for mobile navigation and human–robot interaction that cover aspects such as localization, map-building, SLAM, multimodal user detection and tracking, human-aware navigation, situation recognition, dialog adaptation, and several other aspects. As challenging application field, his research is focused on socially assistive mobile robots for public and home environments. He is a member of IEEE, INNS, ENNS, GI and VDI.