in: Proc. 6th European Conference on Mobile Robots (ECMR 2013), Barcelona, Spain

Generic 2D/3D SLAM with NDT Maps for Lifelong Application

Erik Einhorn and Horst-Michael Gross
Ilmenau University of Technology, Germany

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 some modifications for the original NDT mapping to handle free-space measurements explicitly and to enable its usage in dynamic environments with moving obstacles and persons. 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.**

I. 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 current robot's 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 [22]. 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 often is used for map acquisition only during an offline map learning phase as part of the initial setup of the robot in its novel environment [7]. 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 [6].

In our previous real-world applications where we implemented tour guide robots and interactive shopping assistants [10], we also followed the philosophy of a map learning phase and a separate operation phase. However, today's complex applications such as robot companions that assist elderly people in their home environments [11] 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 separated map learning phase is no longer acceptable. Instead, the mapping phase must continue during the whole operation time of the robot to adapt the map permanently 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.

Modern assistance robots typically have a variety of sensors that provide different kinds of information about the robots 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 [4]. Therefore, we are interested in applying a generic SLAM approach that is able to process such 2D and 3D information equally well.

Our contribution in this paper is twofold: We first introduce a novel, generic mapping technique that can operate with various range sensors of different dimensionality to generate compact 2D and 3D maps. Based on this mapping technique, we present a lifelong SLAM approach that satisfies all of the aforementioned requirements and hence is suited for real-world applications. In summary, our proposed approach:

1) is implemented in a generic way for 2D and 3D mapping
2) operates in realtime and allows online robot localization
3) allows lifelong mapping with constant complexity
4) operates in semi-static or dynamic environments and adapts the map to the changing environment

This paper is organized as follows. The next section outlines the state of the art in SLAM and robot mapping. In section IV, we describe our approach in detail. In section V, we show several results that we have obtained using the presented approach for different kinds of sensors. Finally, we conclude with an outlook for future work.

II. Related Work

As stated before, a large variety of different SLAM approaches are available. Some techniques interpret the SLAM problem as a filtering problem and apply Extended Kalman filters [2] or Rao-Blackwellized Particle Filters [9], [22] to solve it. Others apply smoothing techniques [13], [12] 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* [16] 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

*This work has received funding from the Federal State of Thuringia and the European Social Fund (OP 2007-2013) under grant agreement N501/2009 to the project SERROGA (project number 2011FGR0107)

loop-closures have a catastrophic impact on the resulting map and the robot's pose estimates. This problem is addressed in [26] 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 [15] 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 is free. These maps, therefore, allow to model free space and unknown areas explicitly using the stored occupancy value. Voxel maps usually are stored using octree representations [19], [5], [28], [3] 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 [1] for efficient laser scan matching and was later extended to three dimensions [17]. 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. As shown in [23], 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 [1], [17], [23] 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 lifelong SLAM 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 [21] an extension is proposed that models an occupancy probability for each cell, and hence allows to represent empty cells. However, in their approach the stored Normal Distributions are not taken into account, when the occupancy values of the cells are updated.

III. Normal Disitribution Transform Mapping

In this section, we are describing our new version of NDT mapping which adds the capability to integrate information about free space in the maps. In contrast to [21] we explicitly consider the stored Normal Distribution in each cell, when updating the occupancy values in a probabilistic sound way. Moreover, we present a fully generic implementation that allows to generate 2D as well as 3D maps without any changes in the algorithms.

Similar to occupancy grid maps and occupancy voxel maps, the mapped volume is partitioned into uniform cells. These cells are managed in a tree structure. 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 [3]. 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 [17], 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.

A. Updating the Map with Range Measurements

A priori, the state of all cells is unknown. Therefore, the covariance in each cell is set to "infinity" which results in the probability mass to take up the whole cell uniformly. Moreover, the occupancy value is set to $o_c = 0.5$ to indicate that the state of the whole cell is unknown.

With each measurement of the used range sensor, the map is updated. Each 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 [8]. Afterwards, the normal distribution in this cell is updated using the following incremental update rule:

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

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 a posteriori 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 changing object position.

Beside updating the normal distribution of the hit cell, the mapping algorithm also updates its occupancy value and the occupancy values of all cells along the sensor beam between the sensor's position and the endpoint of the measurement according to:

$$l_c' = l_c + l_s(r, z) \quad (2)$$

with $l_c = log(o_c) - log(1-o_c)$ being the log-odds of the occupancy value o_c of each such cell. This is the standard update rule of occupancy maps as described in [27]. The function $l_s(r,z)$ is known as inverse sensor model and yields the probability for a cell at distance r being occupied when a range measurement of z was obtained. Typically, such a sensor model gives values > 0 for the cell that is hit by the measurement, since it is most likely to be occupied and values < 0 for cells that are located between the sensor and the endpoint of the measurement, since those cells were traversed by the sensor beam and hence are most likely to be free. Consequently, the occupancy probability of the cell that is hit by the measurement is increased, while it is decreased for the cells along the respective sensor beam.

B. 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 on 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:

$$\begin{aligned} \boldsymbol{\mu}_p &= \textstyle\sum_i w_i\boldsymbol{\mu}_i, \qquad \boldsymbol{\Sigma}_p = \sum_i w_i\boldsymbol{\Sigma}_i + \beta_i(\boldsymbol{\mu}_p - \boldsymbol{\mu}_i)^\top \\ k_p &= \textstyle\sum_i k_i, \qquad\qquad l_p = \sum_i l_i \end{aligned} \quad (3)$$

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.

The described tree structure and the above map update algorithm is independent from the dimension of the map and the used range sensor. Therefore, we encapsulate these operations in a *mapping backend*. Beside this mapping backend, we have different *sensor frontends* for each type of range sensor as shown in the left part of Fig. 1. The advantage of this architecture now is, that the task of each sensor frontend simplifies to processing the range data of the respective sensor and to call the backend to specify the cells that need to be updated as occupied or free depending on the sensor's measurements.

C. Depth-Image Mapping Frontend

In the following, we exemplarily describe our frontend for 3D mapping with depth images that are obtained using a range sensor such as the Microsoft Kinect.

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 occupied as described above and its occupancy value is increased.

Updating cells as free is a little more complex. Other mapping approaches like [28] traverse the cells along the measurement ray explicitly via ray casting. However, this can be very time consuming 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 stored normal distribution and project each sample point to the depth image. If the measured depth z at the projected position is larger than the distance r between the sample point and the sensor, the sample point was traversed by the ray of the measurement. In this case the occupancy value of the cell is decreased according to (2) using the following linear sensor model: $l_{free}(r,z) = c_{free}(z-r)/z$, i.e. the occupancy value of cells near to the sensor are adapted more than cells close to the measured distance z.

IV. Lifelong SLAM

Similar to [15], 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 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 covariance matrix $\boldsymbol{\Omega}_{ji}$.

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

$$\mathbf{X}^* = \operatorname*{argmin}_{\mathbf{X}} \sum_{i,j} \mathbf{e}(\mathbf{x}_i, \mathbf{x}_j, \boldsymbol{\delta}_{ji})^\top \boldsymbol{\Omega}_{ji} \mathbf{e}(\mathbf{x}_i, \mathbf{x}_j, \boldsymbol{\delta}_{ji}) \quad (4)$$

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} [16].

For solving the above optimization problem, we use *g2o* [16], 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.

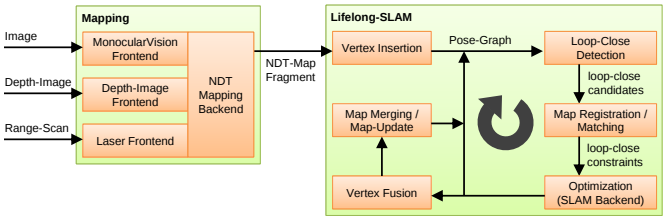

Fig. 1. The flow diagram of our complete mapping pipeline with the mapping component consisting of the mapping backend and different sensor frontends as well as the components of the actual SLAM approach.

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. 1, 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 robots 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.

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. 1, top right). Therefore, it propagates the uncertainties of the pose estimates through the pose graph starting at the newly added vertex. This is done similar 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 [26], we increase the robustness the error function of a loop-closure constraint is weighted 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 totality 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 merged easily. 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 used directly 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.

A. 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 [24] and therefore only briefly described here. For details please refer to that work.

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

$$d(M_n, M_m, \delta_{nm}) = \sum_{i \in M_n} \mathbf{d}_{ij}^\top (\mathbf{R}_{nm}^\top \mathbf{\Sigma}_i \mathbf{R}_{nm}^\top + \mathbf{\Sigma}_j)^{-1} \mathbf{d}_{ij} \tag{5}$$

with $j = \text{argmin}_{j \in M_m} \|\boldsymbol{\mu}_i - \boldsymbol{\mu}_j\|$ 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-Marquard (LM) method for solving the non-linear minimization problem. Since NDT maps are piecewise differentiable, the necessary derivatives can be computed as described in [24]. 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) [14] 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.

B. Normal Space Sampling

As stated before, the map registration and the computation of the distance metric in (5) 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 criteria.

To improve the robustness of our map registration, we adapt a variant of the ICP algorithm that is known as *normal space sampling* [20]. Therefore, we slightly modify Eq.(5). 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. 2) can be vital. 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.(4) are marginal (Fig. 2a). 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 direction information during the iterative registration process that leads to a better convergence if it is exploited.

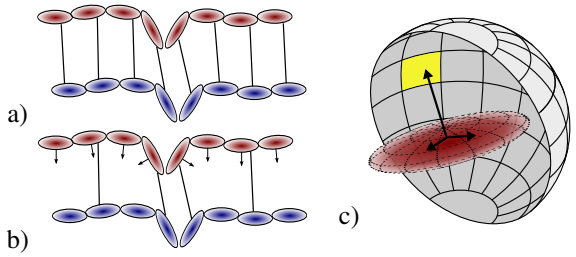

Fig. 2. If the normal distributions for the pairing during the map registration are chosen uniformly, small features are suppressed (a). If *normal space sampling* is used, the normal distributions around the feature have a significant influence (b). The normal distributions are partitioned in a histogram 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 (c).

For this reason, we compute the orientation of each normal distribution $\mathcal{N}_i$ of map M_n in terms of it's "normal vector" $\mathbf{n_i}$. As normal vector we use the eigen vector of the covariance matrix $\mathbf{\Sigma}_i$ that corresponds to the smallest eigen value. 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 2c 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.(5) to ensure, that all orientations are represented equally well (Fig. 2b).

C. 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 prone 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 (4) while removing vertex v_j, we would need to add edges for each pair of neighbors of v_j [15]. This would result in an increasing complexity of the pose graph. Therefore, we use the same approximation that was proposed in [15] and connect all neighbors of v_j to v_i while adapting the information stored in these edges as described in [15]. 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 (3). 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.

V. Results

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

Fig.3 shows a 3D map that was build using 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 $5 \times 8\ m^2$.

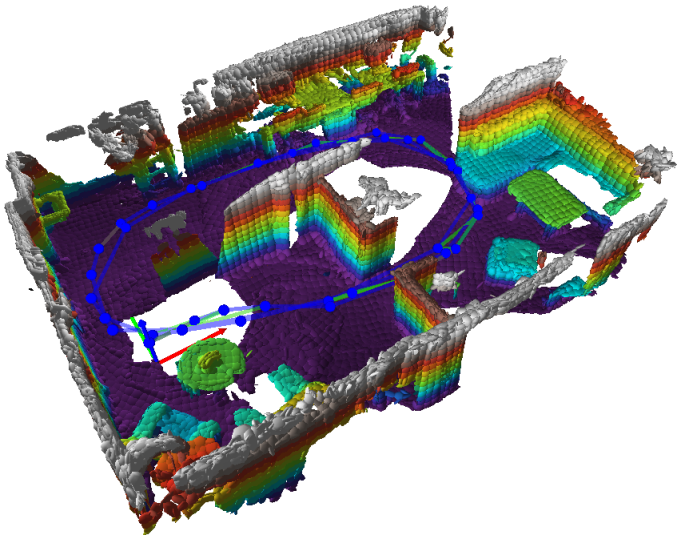

Fig. 3. 3D NDT map created using a Microsoft Kinect while driving three loops through a home environment. Each normal distribution of a NDT cell is shown as ellipsoid and colored according to its height. The pose graph with its vertices and edges is indicated in blue.

In Fig.4 a 2D map is shown that was created using a laser range finder while driving our tour guide robot manually through an office building. The mapped area has a size of $80 \times 35\ m^2$. 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 twig 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.

A. 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 takes 10-20 ms only. 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.

B. Lifelong Operation

To test the ability of our approach for long term operations in a bounded environment, we ran it for two days on one of our guide robots [25] that operate in our office building on a daily basis to autonomously guide and to tour visitors within the building. On a regular office day, when the robots typically operate for about 6 hours, 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 hours of continuous driving. The tests were restricted to a single floor of the building that is shown in Fig.4. 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.5 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.

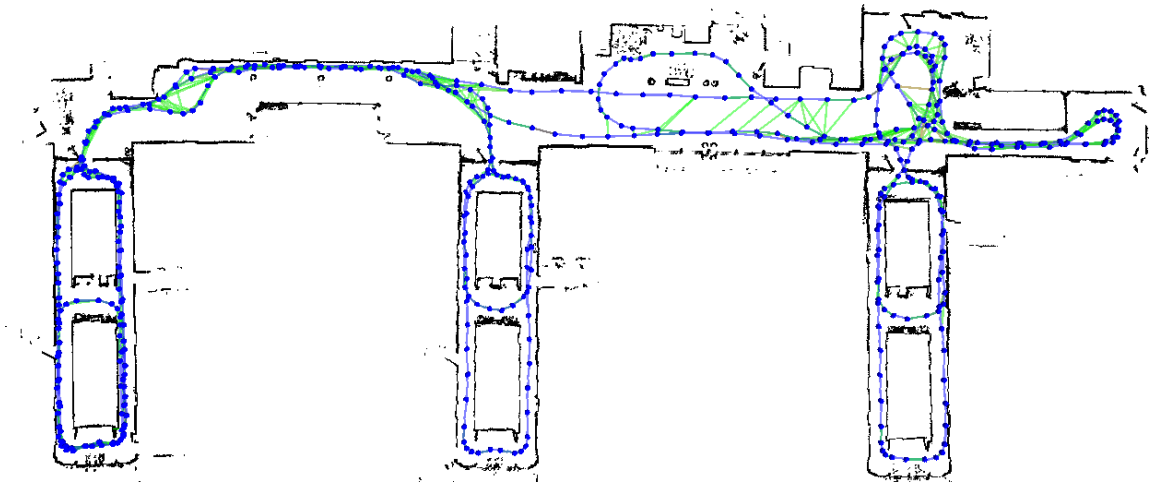

Fig. 4. 2D NDT map created using a laser range finder while driving through an office building. The pose graph with its vertices and edges is indicated in blue.

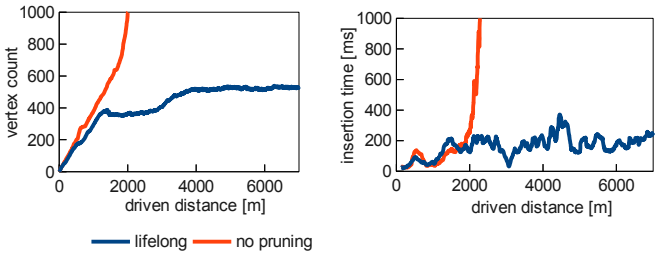

Fig. 5. **left:** Number of vertices in the pose graph during a 2 day long term test for the proposed lifelong SLAM approach (blue) and without fusing vertices (red). **right:** Time for inserting a new map fragment and a new vertex into the pose graph during the test.

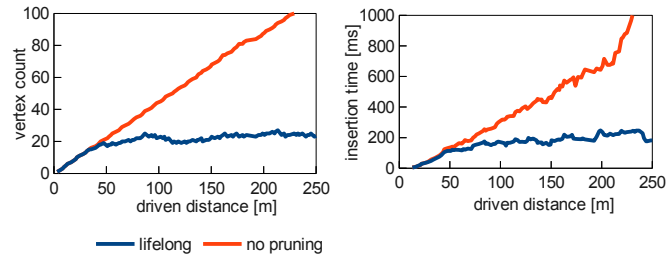

Fig. 6. **left:** Number of vertices in the pose graph while creating a 3D map in a small home environment with pruning (blue) and without fusing vertices (red). **right:** Time for inserting a new map fragment and a new vertex into the pose graph during the test.

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. 5. 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. 3. 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 by 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. 6. 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 depends on the density of the pose graph vertices only. Therefore, it will not increase significantly even if the approach operates in larger environments or for a longer period of time.

C. Dynamic Environments

We also tested our approach in dynamic and semi-static environments. Fig.7 shows, how a moving person is handled by the proposed NDT mapping approach. The explicit free space mapping allows to update cells as free, immediately after the person has left the corresponding volume. This is essential for building consistent maps in dynamic environments.

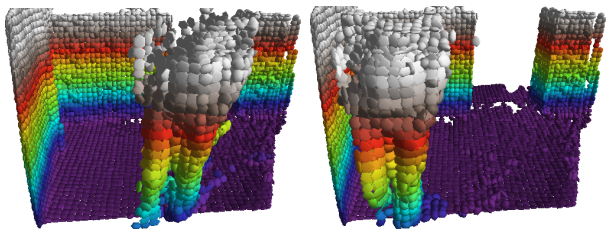

Fig. 7. While a person moves through the scene, previously occupied cells are correctly updated as free.

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.8. During the mapping we moved two armchairs and a table to the opposite corner of the room. Nevertheless, 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.8. Thus, the approach is able to adapt to the changed environment.

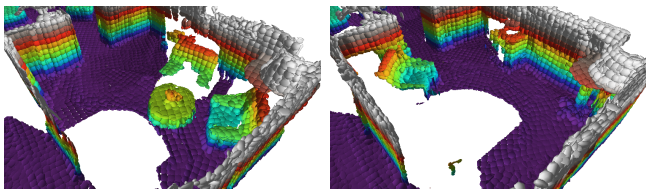

Fig. 8. While the robot was mapping its environment, the table and the armchairs visible in the left image, were moved to the opposite side of the room as shown in the right image.

Several videos of the proposed approach are available one: `http://www.youtube.com/neurobTV`

VI. 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. It is able to 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.

References

[1] P. Biber and W. Straßer. The normal distributions transform: A new approach to laser scan matching. In *In Proceedings of the IEEE International Conference on Intelligent Robots and Systems (IROS)*, pages 2743–2748, 2003.

[2] A. J. Davison, I. D. Reid, N. D. Molton, and O. Stasse. MonoSLAM: Real-Time Single Camera SLAM. *IEEE Trans. Pattern Anal. Mach. Intell.*, 29(6):1052–1067, 2007.

[3] E. Einhorn, C. Schroter, and H.-M. Gross. Finding the adequate resolution for grid mapping - Cell sizes locally adapting on-the-fly. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 1843–1848, 2011.

[4] E. Einhorn, C. Schroter, and H. Gross. Can't take my eye off you: Attention-driven monocular obstacle detection and 3D mapping. In *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*, pages 816–821, 2010.

[5] N. Fairfield, G. Kantor, and D. Wettergreen. Real-time SLAM with octree evidence grids for exploration in underwater tunnels. *Journal of Field Robotics*, 2007.

[6] D. Fox. KLD-Sampling: Adaptive Particle Filters. In *Advances in Neural Information Processing Systems 14*. MIT Press, 2001.

[7] U. Frese, R. Wagner, and T. Röfer. A SLAM Overview from a Users Perspective. *KI - Künstliche Intelligenz*, 24:191–198, 2010.

[8] S. Frisken and R. Perry. Simple and Efficient Traversal Methods for Quadtrees and Octrees. *Journal of Graphics Tools*, 7, 2003.

[9] G. Grisetti, C. Stachniss, and W. Burgard. Improved Techniques for Grid Mapping With Rao-Blackwellized Particle Filters. *Robotics, IEEE Transactions on*, 23(1):34–46, 2007.

[10] H.-M. Gross, H.-J. Böhme, C. Schröter, S. Müller, A. König, E. Einhorn, C. Martin, M. Merten, and A. Bley. Interactive Shopping Guide Robots in Everyday Use - Final Implementation and Experiences from Long-term Field Trials. In *Proc. IEEE/RJS International Conference on Intelligent Robots and Systems (IROS)*, pages 2005–2012, 2009.

[11] H.-M. Gross, C. Schröter, S. Müller, M. Volkhardt, E. Einhorn, A. Bley, C. Martin, T. Langner, and M. Merten. Progress in Developing a Socially Assistive Mobile Home Robot Companion for the Elderly with Mild Cognitive Impairment. In *Proc. IEEE/RJS Int. Conf. on Intelligent Robots and Systems (IROS)*, pages 2430–2437, 2011.

[12] M. Kaess, H. Johannsson, R. Roberts, V. Ila, J. Leonard, and F. Dellaert. iSAM2: Incremental smoothing and mapping with fluid relinearization and incremental variable reordering. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 3281 –3288, 2011.

[13] M. Kaess, A. Ranganathan, and F. Dellaert. iSAM: Incremental Smoothing and Mapping. *Robotics, IEEE Transactions on*, 24(6):1365–1378, 2008.

[14] J. Kennedy and R. Eberhart. Particle swarm optimization. In *Neural Networks, 1995. Proceedings., IEEE International Conference on*, volume 4, pages 1942–1948, 1995.

[15] H. Kretzschmar, G. Grisetti, and C. Stachniss. Lifelong Map Learning for Graph-based SLAM in Static Environments. *KI - Knstliche Intelligenz*, 24:199–206, 2010.

[16] R. Kummerle, G. Grisetti, H. Strasdat, K. Konolige, and W. Burgard. G2o: A general framework for graph optimization. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 3607–3613, 2011.

[17] M. Magnusson. *The Three-Dimensional Normal-Distributions Transform an Efcient Representation for Registration*. PhD thesis, Örebro University, 2009.

[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, and C. Gosselin. Probabilistic octree modeling of a 3-d dynamic environment. In *Proc. of the IEEE Int. Conf. on Robotics and Automation (ICRA)*, 1997.

[20] S. Rusinkiewicz and M. Levoy. Efficient variants of the icp algorithm. In *Int. Conf. on 3-D Digital Imaging and Modeling*, 2001.

[21] J. Saarinen, H. Andreasson, T. Stoyanov, J. , Luhtala, and A. Lilienthal. Normal Distributions Transform Occupancy Maps: Application to Large-Scale Online 3D Mapping. In *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, pages 2225–2230, 2013.

[22] C. Schröter and H.-M. Gross. A sensor-independent approach to RBPF SLAM - Map Match SLAM applied to visual mapping. In *IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS*, pages 2078–2083, 2008.

[23] T. Stoyanov, M. Magnusson, H. Almqvist, and A. Lilienthal. On the accuracy of the 3d normal distributions transform as a tool for spatial representation. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 4080–4085. IEEE, 2011.

[24] T. Stoyanov, M. Magnusson, and A. J. Lilienthal. Point set registration through minimization of the l2 distance between 3d-ndt models. 2012.

[25] R. Stricker, S. Muller, E. Einhorn, C. Schroter, M. Volkhardt, K. Debes, and H. Gross. Interactive mobile robots guiding visitors in a university building. In *RO-MAN, 2012 IEEE*, pages 695–700, 2012.

[26] N. Sünderhauf and P. Protzel. Switchable constraints for robust pose graph SLAM. In *IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS*, pages 1879–1884, 2012.

[27] S. Thrun, W. Burgard, and D. Fox. *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents)*. The MIT Press, 2005.

[28] K. M. Wurm, A. Hornung, M. Bennewitz, C. Stachniss, and W. Burgard. OctoMap: A Probabilistic, Flexible, and Compact 3D Map Representation for Robotic Systems. In *In Proc. of the ICRA 2010*, 2010.