2010 IEEE International Conference on Robotics and Automation
Anchorage Convention District
May 3-8, 2010, Anchorage, Alaska, USA

Optimization Techniques for Laser-Based 3D Particle Filter SLAM

Jochen Welle, Dirk Schulz, Thomas Bachran
Fraunhofer FKIE
Neuenahrer Straße 20, D-53343 Wachtberg
{jochen.welle, dirk.schulz, thomas.bachran}@fkie.fraunhofer.de

Armin B. Cremers
Institute of Computer Science III
University of Bonn
Römerstraße 164, D-53117 Bonn
abc@iai.uni-bonn.de

***Abstract*— In recent years multiple simultaneous localization and mapping (SLAM) algorithms have been proposed, which address the challenges of 3D environments in combination with six degress of freedom in the robot position. Commonly, solutions based on scan-matching algorithms are applied. In contrast to these approaches, we propose to use a particle filter transferring the concept of the 2D Rao-Blackwellized particle filter SLAM to 3D. As filter input, 3D laser range data and odometry readings are obtained while the robot is in motion. The ground plane is estimated based on previously built map parts, thereby approaching the problem that not all degrees of freedom are covered by the odometry. To gain control of the high memory requirements for the particles' 3D map representations, we introduce a memory efficient search structure and adapt a technique to efficiently organize and share maps between particles. We evaluate our approach based on experimental results obtained by simulation as well as measurements of a real robot system.**

I. Introduction

The availability of an environmental map is essential for many tasks of a mobile robot system. Thus, building and maintaining a map based on sensor information is a key ability. In order to integrate new sensor information into an existing map the robot position needs to be known. However, the robot position estimate depends on the previously collected map information. Due to this mutual dependency, an error in the robot position estimate has a direct impact on the map consistency, while a mapping error influences the accuracy of the robot location estimate. This problem is known as *simultaneous localization and mapping* (SLAM) and has been addressed by a lot of researchers.

Early SLAM approaches have been applied for 2D map building in flat indoor environments. While 2D maps are often sufficient for this simple type of environment, more recent approaches address the challenge of non-flat, indoor and outdoor environments by using 3D sensor information and building 3D maps. One method is to apply the well-known scan-matching paradigm to 3D [1], [2], [3]. In contrast to these methods, we follow a different approach and apply the particle filter paradigm, which has been explored for 2D SLAM in [4], [5], [6], [7], [8]. We transfer the particle filter concept to the 3D environment and support six degrees of freedom in the robot movement. Our approach uses 3D laser scans and odometry information. A special feature is the support for robot mobility, while a 3D laser scan is performed. Some 3D approaches like [1] suffer from the need to stop during a laser scan. To counter this problem, additional sensor information can be used [3], but this solution is not considered in this paper. Instead, the usage of a particle filter allows to process the data acquired while the robot moves, without the need to first combine it into local maps.

Fig. 1. Our robot platform: Pioneer P3-AT, which carries a SICK LMS laser range scanner mounted vertically on a rotating disk. Image courtesy of Timo Röhling.

When transferring the particle filter approach to 3D, a higher number of particles is required due to the additional three dimensions in the robot state (z location and pitch and roll angles). This increases runtime and memory consumption. To reduce this impact, we include the collected environmental map in the robots motion model. We thereby restrict the number of possible robot positions and compensate the lack of additional sensors to determine all degrees of freedom. In combination with the higher number of required particles, the more complex map built for each particle dramatically increases the memory consumption. In our solution we developed a memory efficient search structure based on an Octree [9] and using delta encoding to reduce this effect. In addition, we adopt a technique to share submaps between different particles if they have a common history. This idea was implemented for 2D maps by Eliazar and Parr in the DP-SLAM algorithm [8].

This paper is organized as follows. First, in section II, we briefly introduce the particle filter concept and some related work. We present our extensions to the basic particle filter concept, i. e. the improved motion model including a ground plane estimation and the modified weighting of

the particles, which is used for the resampling strategy to concentrate on the most probable particles. Afterwards, in section III, we describe the map represenation and the method applied to control its memory consumption. The experimental evaluation based on results on simulation and real robot measurements, obtained with our robot system shown in figure 1, is presented in section IV. Finally we conclude the paper in section V.

II. Particle Filter SLAM

For the SLAM problem, a map m and robot positions (including orientations) $x_{1:t} = x_1, .., x_t$ are sought, which best explain the input information given by odometry measurements $u_{1:t}$ and distance measurements $z_{1:t}$ of the environment. Since the input data does not cover all required information and may be erroneous, the real map and positions are usually described by a probability distribution $\mathrm{P}(x_{1:t}, m|z_{1:t}, u_{1:t})$.

Murphy, Doucet et al. [10], [11] determine this distribution by introducing the Rao-Blackwellized particle filter (RBPF). Particle filters are sequential monte carlo methods. The basic idea is to approximate a probability distribution with a set of samples, called particles. This set is updated if a new filter input alters the probability distribution. Since a direct approximation of $\mathrm{P}(x_{1:t}, m|z_{1:t}, u_{1:t})$ using a particle filter would be inefficient, their key idea is to split up the distribution:

$$\mathrm{P}(x_{1:t}, m|z_{1:t}, u_{1:t}) = \mathrm{P}(m|x_{1:t}, z_{1:t})\,\mathrm{P}(x_{1:t}|z_{1:t}, u_{1:t}) \quad (1)$$

First, the robot positions $\mathrm{P}(x_{1:t}|z_{1:t}, u_{1:t})$ are approximated by a particle filter. Here, each particle represents one possible robot trajectory. Second, the probability distribution for the map $\mathrm{P}(m|x_{1:t}, z_{1:t})$ is calculated conditioned on the robot positions. Thus, conventional solutions to solve the mapping problem can be applied. A characteristic of this approach is that a map has to be maintained for each particle.

The particle filter incrementally updates the robot trajectories of all particles based on new pairs of odometry and distance measurements (u_t, z_t). An update step from time $t-1$ to t is performed in four steps:

(1) For each particle i a new position $x_t^{(i)}$ is sampled from a proposal distribution π. Usually π specifies the assumptions about possible robot movements.

(2) The importance weight $w_t^{(i)}$ is computed for each particle i according to the mismatch between proposal and target distribution:

$$w_t^{(i)} = \frac{\mathrm{P}(x_{1:t}^{(i)}|z_{1:t}, u_{1:t})}{\pi(x_{1:t}^{(i)}|z_{1:t}, u_{1:t})}. \quad (2)$$

(3) In the resampling step, a new set of particles is drawn with repetition from the old set. The sampling is done proportional to the particle weight $w_t^{(i)}$. The weights of the particles in the new set are initialized to one. Due to the resampling, the filter assigns more particles to trajectories, which are underestimated by the proposal distribution in comparison to the target distribution. Particles which are overestimated may be removed.

(4) The map $m_t^{(i)}$ of each particle i is updated based upon the new distance measurement z_t and robot positions $x_t^{(i)}$.

The particle filter concept presented so far has undergone several modifications over time. Montemerlo et al. [4] were first to realize the approach on real robots, based on the ideas of Murphy and Doucet. They developed a landmark-based SLAM algorithm, called FastSLAM. Hähnel et al. [5] modified the FastSLAM algorithm by introducing grid maps and combining it with scan matching to compute improved odometry measurements with the goal to reduce the number of particles. Stachniss et al. [6] continued the work of Hähnel et al. Based on a scan-matching process, which is carried out once per particle, they compute a Gaussian distribution to approximate the optimal proposal distribution. Furthermore, they introduce an intelligent resampling technique to reduce the number of resampling operations. Grisetti et al. [7] concentrate on computational speed and memory efficiency aspects. They reuse an already computed proposal distribution and provide a compact map model. Parallel to the work of Hähnel et al. Eliazar and Parr [8] developed a different grid map based RBPF and described a memory efficient map representation. By maintaining an ancestry tree of particles and storing an observation tree in each grid map cell, they achieve a compact representation of the map hypotheses.

A. Motion Model with Ground Plane Estimation

In the first particle filter step the proposal distribution is used to determine the new robot position. To allow efficient, incremental processing, the distribution is commonly transferred into a recursive form:

$$\begin{aligned}\pi(x_{1:t}|z_{1:t}, u_{1:t}) &= \pi(x_t|x_{1:t-1}, z_{1:t}, u_{1:t}) \\ &\quad \pi(x_{1:t-1}|z_{1:t-1}, u_{1:t-1}). \quad (3)\end{aligned}$$

A frequent choice for $\pi(x_t|x_{1:t-1}, z_{1:t}, u_{1:t})$ is the odometry motion model $\mathrm{P}(x_t|x_{t-1}, u_t)$ [12]. However, in 3D non-planar environments not all relevant degrees of freedom are considered. Especially changes of the ground plane, on which the robot is moving, can lead to large mismatches between target and proposal distributions. These mismatches result in a higher demand of particles to approximate the target distribution with sufficient quality, thereby increasing run-time and memory requirements. To counter this effect, we consider the ground plane, which can be calculated from previously acquired measurements $z_{1:t-1}$ and positions $x_{1:t-1}$. This information is available in the currently known map m_{t-1}, resulting in our motion model:

$$\mathrm{P}(x_t|x_{t-1}, u_t, m_{t-1}). \quad (4)$$

As ground plane, we determine the regression plane of the nearest neighbor points around the current particle position. This plane is calculated using singular value decomposition and then perturbed with a Gaussian error. A new particle position is obtained in that plane by sampling from the odometry motion model. For future work we plan to include the latest measurement z_t to improve the proposal distribution, but omit it for now to simplify calculations.

B. Weighting Particles

The weighting of particles is an important prerequisite to determine the most likely trajectories and for selecting adequate trajectories in the resampling step. Based on equation (2), the weight is given by:

$$
\begin{aligned}
w_t^{(i)} &= \frac{\mathrm{P}(x_{1:t}^{(i)}|z_{1:t},u_{1:t})}{\pi(x_{1:t}^{(i)}|z_{1:t},u_{1:t})} \\
&= \frac{\mathrm{P}(z_t|x_{1:t}^{(i)},z_{1:t-1},u_{1:t})\,\mathrm{P}(x_{1:t}^{(i)}|z_{1:t-1},u_{1:t})}{\mathrm{P}(z_t|z_{1:t-1},u_{1:t})\,\pi(x_{1:t}^{(i)}|z_{1:t},u_{1:t})} \quad . \qquad (5) \\
&\overset{(3)}{=} \eta\frac{\mathrm{P}(z_t|x_{1:t}^{(i)},z_{1:t-1},u_{1:t})\,\mathrm{P}(x_t^{(i)}|x_{1:t-1}^{(i)},z_{1:t-1},u_{1:t})}{\pi(x_t^{(i)}|x_{1:t-1}^{(i)},z_{1:t},u_{1:t})} \\
&\quad \cdot \underbrace{\frac{\mathrm{P}(x_{1:t-1}^{(i)}|z_{1:t-1},u_{1:t-1})}{\pi(x_{1:t-1}^{(i)}|z_{1:t-1},u_{1:t-1})}}_{w_{t-1}^{(i)}} \quad . \qquad (6)
\end{aligned}
$$

The common Markov assumption in 2D SLAM algorithms $\mathrm{P}(x_t|x_{1:t-1},z_{1:t-1},u_{1:t}) = \mathrm{P}(x_t|x_{t-1},u_t)$ is not satisfied here. u_t only measures the movement on the surface and does not give any information about surface changes. Thus, x_t also depends on $x_{1:t-1}, z_{1:t-1}$, for which the map m_{t-1} is assumed as a sufficient statistic:

$$
w_t^{(i)} \propto w_{t-1}^{(i)} \cdot \mathrm{P}(z_t|x_t^{(i)},m_{t-1}^{(i)})\frac{\mathrm{P}(x_t^{(i)}|x_{t-1}^{(i)},m_{t-1},u_t)}{\pi(x_t^{(i)}|x_{1:t-1}^{(i)},z_{1:t},u_{1:t})} \quad . \qquad (7)
$$

Inserting the motion model from section II-A, we obtain:

$$
\begin{aligned}
w_t^{(i)} &\propto \mathrm{P}(z_t|x_t^{(i)},m_{t-1}^{(i)})\,\frac{\mathrm{P}(x_t^{(i)}|x_{t-1}^{(i)},m_{t-1},u_t)}{\mathrm{P}(x_t^{(i)}|x_{t-1}^{(i)},m_{t-1}^{(i)},u_t)} \cdot w_{t-1}^{(i)} \\
&= \mathrm{P}(z_t|x_t^{(i)},m_{t-1}^{(i)}) \cdot w_{t-1}^{(i)} \quad . \qquad (8)
\end{aligned}
$$

Hence, the weight is calculated according to the measurement model. In our work, we use a model similar to the likelihood-field model of [12].

C. Resampling

The resampling step of the particle filter avoids degeneration effects. By avoiding a major number of particles with low importance weights, it renders the approximation of the target distribution with a limited number of particles possible. A drawback of resampling is, that if it is performed too often, e. g. for every particle filter iteration, the filter may concentrate on too few different trajectories.

Due to the counteractive effects of frequent and infrequent resampling Stachniss et al. [6] suggested a criterion whether a resampling step should be carried out or not. For the decision, they estimate the effective number of particles N_{eff} and compare it to the real number of particles N_p. The basic idea behind N_{eff} is that, if the approximation of the target distribution is exact, the importance weights would be equal to each other. N_{eff} is then equal to N_p. The higher the variance in the importance weights, the worse is the approximation of the target distribution and the lower is the effective number of particles. If N_{eff} falls under a certain threshold, in our case $N_p/2$, a resampling step is performed, effectively resetting N_{eff} to N_p. To make N_{eff} comparable to a number of particles, Grisetti et al. [13] use the normalized importance weights $\tilde{w}_t^{(i)}$ instead of the importance weights $w_t^{(i)}$ to determine N_{eff}:

$$
N_{eff} = \frac{1}{\sum_{i=1}^{N_p}\left(\tilde{w}_t^{(i)}\right)^2} \quad . \qquad (9)
$$

III. Map Representation and Organization

In RBPF-SLAM each particle carries its own map, derived from the trajectory of the particle and the measurements of the environment. Fairfield et al. [14] extend the grid map particle filter SLAM for an underwater robot to 3D grid maps. Due to the high memory requirements of 3D grid maps, their approach does not scale very well with respect to particle number and map resolution. In our outdoor scenario with a desired resolution of $10\,\mathrm{cm}^3$ we would require 400 MB per particle, resulting in a total of 400 GB with 1.000 particles. Since 3D environments mainly consist of open space, a lot of grid cells are unused. To avoid this inefficient encoding, we use simple point clouds to represent the maps. The points are organized in search structures to allow efficient neighborhood queries, used in the measurement model and the estimation of the ground planes. Except for simple scenarios which require only a small number of particles, the memory requirements of one complete map for each particle are still too high. We therefore adapt the efficient map organization of Eliazar and Parr [8] to 3D maps and introduce a memory efficient search structure.

Eliazar and Parr organize the maps by introducing an additional data structure, the ancestry tree. During the resampling step, particles may be removed or duplicated. If particles are duplicated, they share a common history and consequently a common submap. To address these circumstances and limit the map information, the evolution of particles is stored in the ancestry tree. The tree is held minimal by pruning tree nodes without any children in the current generation of particles. Additionally, if nodes have only one child, they are merged with their child. Eliazar and Parr maintain a single grid map for all particles instead of assigning a map to each particle. In their grid map, a single grid cell may contain information for multiple particles which have made changes to the cell.

In contrast to this approach, we store a submap, which consists of the changes to the parent map, in each node of the ancestry tree. If the resampling step does not remove any particles, the memory consumption is not improved compared to the naive approach. However, this worst case is highly unlikely and in common scenarios large map parts are shared among different particles. Besides the memory reduction the method greatly improves the resampling performance, since maps need not to be copied when particles are duplicated.

Usually kd-trees are used for fast nearest neighbor searches. Contrary, we developed a search structure based on an octree [9], which we call DeltaOctree. An octree is a tree structure in which each inner node has eight children that equally subdivide the node's volume into octants. The root

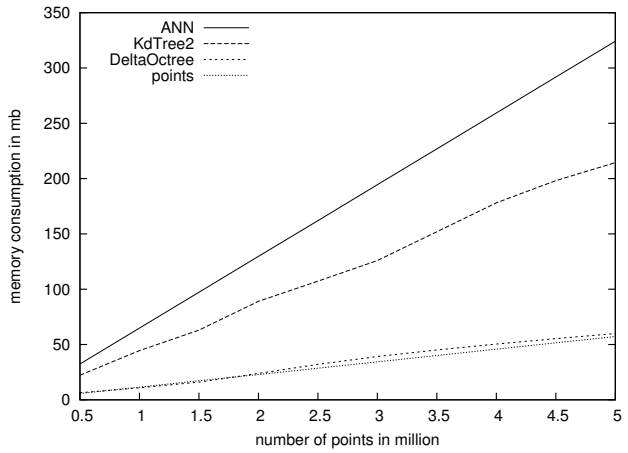


Fig. 2. Comparing the memory consumption of the DeltaOctree with the kd-trees ANN and KdTree2 and to the point list without any search structure.

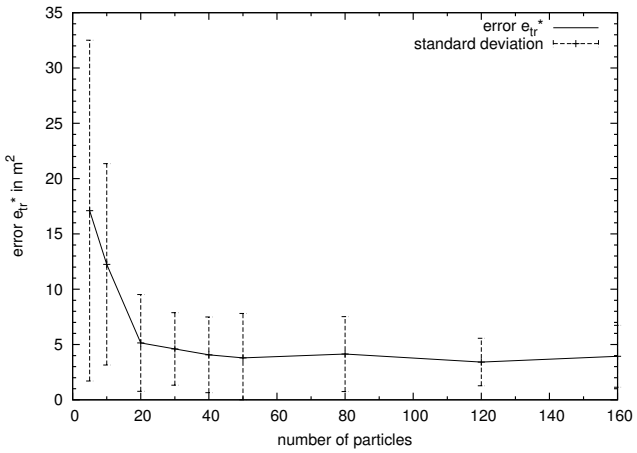


Fig. 3. Mean and standard deviation of e^*_{tr} in scenario 1 with different numbers of particles.

node represents a cube containing the whole environment and the leaf nodes hold the points. To decrease memory consumption, the following three methods are applied:

- *Delta Compression:* Adopting the idea of storing data in form of differences between data records, we store points relative to the center of their octant. Depending on the depth in the tree and the required precision ν, it is possible to store a point with one or two bytes per coordinate. Compared to storing data without delta compression and using four bytes for a 32-bit floating point number per coordinate, the reduction of memory consumption is considerable. This method was the main reason for choosing an octree as base structure, since it is not as easily applied to data structures without fixed space subdivision, e.g. kd-trees. The reduced precision can yield a distance error in nearest neighbor calculations of up to $\sqrt{3}\nu$.
- *Resolution Limit:* The map is limited to a fixed resolution, to avoid storing multiple measurement points in close vicinity to each other, which does not give additional information about the environment. The restriction of the map resolution is realized by introducing a minimal edge length e_{min} and a maximum number of points per octant n_o. Octants with edge length e_{min} are not further subdivided. The resolution limit can introduce a distance error in nearest neighbor calculations of up to $\sqrt{3}e_{min}$.
- *Overhead Reduction:* We reduced the overhead by not storing any redundant information like the center and the edge length of an octant, which can be calculated during tree traversal. Furthermore, we decreased memory management overhead by allocating memory blocks for several nodes at once.

In addition to the savings in memory consumption our DeltaOctree provides an insert operation to add new points. Kd-trees like ANN [15] and KdTree2 [16] lack such an operation and need to be rebuild to include new points.

In our scenarios we used $n_o = 8$, $e_{min} = 5\,\text{cm}$ and $\nu = 1\,\text{mm}$. In comparison to the kd-tree implementations the nearest neighbor query time of our structure is about doubled, but we achieved a significant reduction in memory consumption. The search structure overhead was reduced by a factor of three to six compared to the kd-tree implementations. In figure 2, the overall memory consumption of the search structures is presented. The tested kd-trees dramatically increase the memory consumption in comparison to the point list without any search structure. In contrast to the kd-trees, the overhead of our DeltaOctree is negligible. In some situations it may even consume less memory than the point list. Thus, choosing the appropriate data structure comprises a trade-off between memory consumption and run-time. We focused on lower memory consumption, to be able to apply the system to larger scenarios.

TABLE I

DETAILS OF THE SCENARIOS.

Scenario	area	distance	scans	rec. time
1 simulated	79m x 34m	181m	$\approx$ 20000	5 min
2 real	105m x 100m	385m	$\approx$ 69000	26 min

IV. EXPERIMENTAL RESULTS AND EVALUATION

We carried out experiments on real robots as well as in simulation. The robot platform is a Pioneer P3-AT, which carries a SICK LMS laser range scanner mounted vertically on a rotating disk, see figure 1. The simulator Gazebo was used for simulations. Our SLAM system was successfully applied in multiple different scenarios. For presentation in

TABLE II

RUNTIME AND MEMORY CONSUMPTION ON A 2GHZ DUAL CORE NOTEBOOK WITH 2GB MEMORY. A = ANN, K = KDTREE2, D = DELTAOCTREE, D2 WITH TWO THREADS

Scn.		time (min)				memory (MB)		
	N_p	A	K	D	D2	A	K	D
1	40	4:41	4:06	5:49	2:40	211	161	85
2	100	32	30	31	16	814	626	246
2	200	68	59	69	34	1438	1129	372
2	300	-	99	109	57	-	1715	503
2	1000	-	-	419	234	-	-	1377

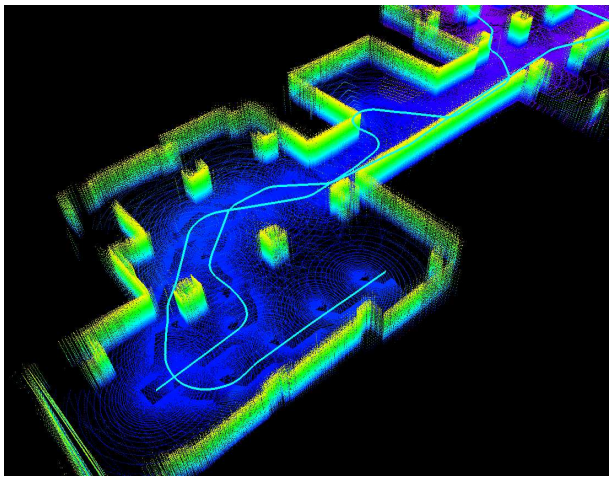


Fig. 4. 3D-View on the corrected map of scenario 1.

this section, we selected two of them. First, the simulation of a simple indoor environment (scenario 1) and second, an experiment on a real robot in an outdoor environment (scenario 2). Table I shows some details of the experiments.

In simulation, the ground truth trajectory of the robot is known and a trajectory error e_{tr} can be calculated for each particle:

$$e_{tr}^{(i)} = \frac{1}{T} \sum_{t=1}^{T} |p_t^{(i)} - \bar{p}_t|^2 \tag{10}$$

In this equation, T specifies the number of robot positions $x_t^{(i)}$, i. e. the number of iterations. $p_t^{(i)} = (x, y, z)^T$ are the ith particle's robot positions without the orientations, and $\bar{p}_t = (\bar{x}, \bar{y}, \bar{z})^T$ are the corresponding ground truth coordinates. Since the orientations influence future positions, the orientation errors are indirectly present in e_{tr}. We refer to the trajectory error of the best rated particle, i. e. the particle with the highest importance weight in the last iteration, as e_{tr}^*. Figure 3 shows the mean and standard deviation of e_{tr}^* in scenario 1, calculated over 70-80 runs per particle number. As the number of particles is increased, the trajectory error of the best rated trajectory decreases. This trend continues up to 50 particles, where it stays on the same level. The remaining error results from small rotational errors in the first part of the trajectory. These errors further propagate until the robot reenters previously measured terrain.

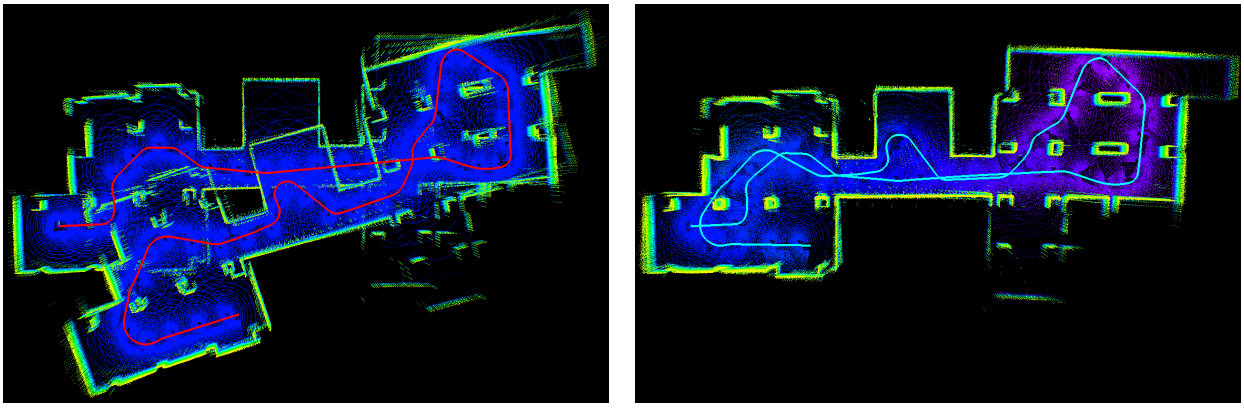
As we observed, the trajectory error is only a rough estimate to determine the quality of the resulting map. A challenging task for the particle filter is to close the loop in our scenario. For low particle numbers the particle filter was able to decrease the trajectory error but the loop was not successfully closed. In order to check whether a built map was consistent or not, we manually assessed all maps. Figure 4 shows a 3D view of a consistent map. Loop closing was usually successful for 40 or more particles. For comparison, figure 5(a) shows the map build based on pure odometry data and figure 5(b) a consistent map result of the particle filter.

In the second experiment, we circled around the cafeteria of the Institute of Computer Science of the University of Bonn using a real robot system. Figure 6(a) shows the map result calculated using the odometry trajectory. Again, the loop around the cafeteria was not successfully closed, resulting in an inconsistent map. For the particle filter loop closing was achieved, but we observed that errors in the absolute elevation and pitch of the calculated trajectory remain in areas traveled only once by the robot. These errors result from propagated local errors in the ground plane estimation. When the robot reenters known terrain and the loop is closed the elevation and pitch are corrected, resulting in a consistent map. Figure 6(b) shows a consistent map using the particle filter with 1.000 particles. In contrast to the simulation environment a higher number of particles is required. This was expected, since our real outdoor environment is more unstructured due to the vegetation and fewer walls and ceilings bounding the area. Furthermore, the larger area and the larger loop complicate the map building. In such a complex environment the particle filter calculations could not be done online anymore on a 2Ghz dual core system with 2GB memory. Table II shows the runtime and memory requirements of our system. For comparison the system was implemented with our proposed DeltaOctree (single and multi threaded) and the kd-trees ANN and KdTree2. The outdoor environment could not be processed with 1.000 particles using the kd-trees, because memory requirements exceeded the actual available resources. Therefore, the experiments were also performed with less particles, even though the resulting maps were inconsistent. As one can see the memory savings are considerable, whereas the runtime increase is not as big as the doubled query time for our DeltaOctree suggests. This results mainly from the insert operation, which the kd-trees lack.

V. Conclusions and Future Work

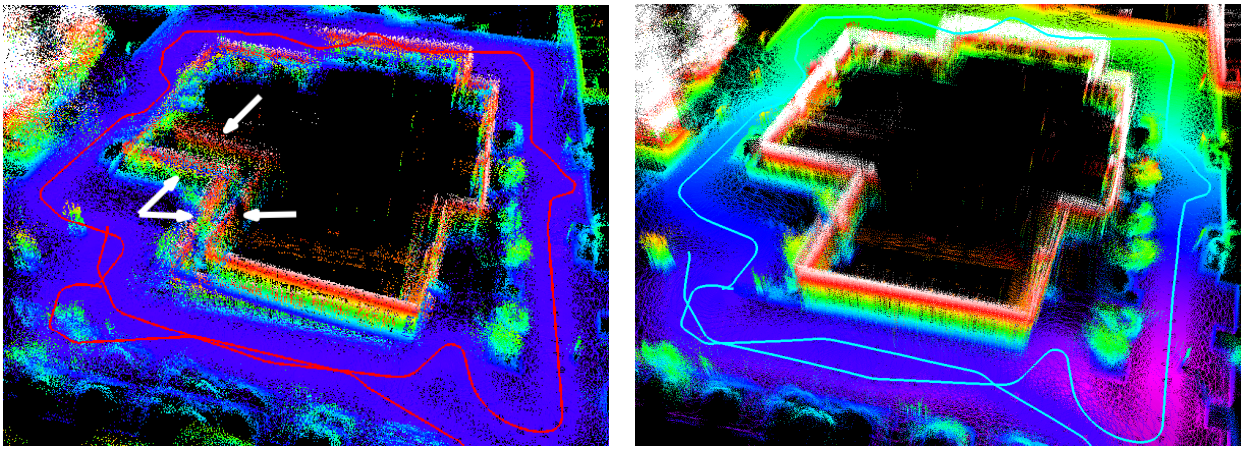
In this paper we presented a novel approach for the SLAM problem in 3D. We applied the particle filter paradigm known from 2D solutions to a mobile robot system having six degrees of freedom in a 3D environment. Our goal was to create a feasible solution concerning the runtime and memory consumption aspects and counter the effects of higher complexity compared to 2D solutions. For this purpose it was crucial to limit the number of particles and the memory consumption of the particles. In our solution the number of required particles was reduced by the robot's ground plane estimation. The memory consumption was controlled by using a special DeltaOctree data structure for map representation and by sharing map parts between particles which have a common history. We carried out multiple experiments in simulation as well as on real robots to test the performance of our approach. It was shown that our particle filter can successfully build 3D maps of non-planar environments, even if the robot platform is in motion while the sensor information is acquired.

Improvements are possible in the ground plane estimation with a calculation which considers more than the local area around the robot. Supplementary the usage of additional sensors to measure the remaining degrees of freedom in



(a) (b)

Fig. 5. Birds eye view of scenario 1. (a) Odometry trajectory and resulting map. (b) Corrected map obtained with 40 particles.



(a) (b)

Fig. 6. Scenario 2: (a) Map resulting from odometry trajectory. The marked walls should overlap. (b) Map of the best rated particle of a run calculated with 1.000 particles.

the robot position would be beneficial and could further reduce the number of required particles. Another approach to reduce the number of particles would be to incorporate laser measurements in the particle generation process as in [5] or [6].

REFERENCES

[1] A. Nüchter, K. Lingemann, J. Hertzberg, and H. Surmann, "6D SLAM - 3D mapping outdoor environments," *Journal of Field Robotics (JFR), Special Issue on Quantitative Performance Evaluation of Robotic and Intelligent Systems*, vol. 24, pp. 699 – 722, Sept 2007.

[2] D. Cole and P. M. Newman, "Using laser range data for 3d SLAM in outdoor environments," in *Proc. of the IEEE Int. Conf. on Robotics and Automation (ICRA)*, Orlando Florida USA, May 2006, pp. 1556–1563.

[3] P. Pfaff, R. Triebel, C. Stachniss, P. Lamon, and W. B. Rol, "Towards mapping of cities," in *Proc. of the IEEE Int. Conf. on Robotics and Automation (ICRA)*, 2007.

[4] M. Montemerlo, S. Thrun, D. Koller, and B. Wegbreit, "FastSLAM: A factored solution to the simultaneous localization and mapping problem," in *Proc. of the AAAI National Conf. on Artificial Intelligence*. Edmonton, Canada: AAAI, 2002.

[5] D. Hähnel, D. Fox, W. Burgard, and S. Thrun, "A highly efficient FastSLAM algorithm for generating cyclic maps of large-scale environments from raw laser range measurements," in *Proc. of the Conf. on Intelligent Robots and Systems (IROS)*, 2003.

[6] C. Stachniss, G. Grisetti, D. Hähnel, and W. Burgard, "Improved rao-blackwellized mapping by adaptive sampling and active loop-closure," Ilmenau, Germany, 2004, pp. 1–15.

[7] G. Grisetti, G. D. Tipaldi, C. Stachniss, W. Burgard, and D. Nardi, "Fast and accurate slam with rao-blackwellized particle filters," *Robotics and Autonomous Systems*, vol. 55, no. 1, pp. 30–38, 2007.

[8] A. Eliazar and R. Parr, "DP-SLAM: Fast, robust simultaneous localization and mapping without predetermined landmarks," in *Proc. 18th Int. Joint Conf. on Artificial Intelligence (IJCAI-03)*. Morgan Kaufmann, 2003, pp. 1135–1142.

[9] H. Samet, "Spatial data structures," in *Modern Database Systems, The Object Model, Interoperability and Beyond*. Addison-Wesley, 1995, pp. 361–385.

[10] A. Doucet, N. de Freitas, K. Murphy, and S. Russell, "Rao-blackwellised particle filtering for dynamic bayesian networks," in *UAI '00 (Uncertainty in AI)*, 2000, pp. 176 – 183.

[11] K. Murphy, "Bayesian map learning in dynamic environments," in *Neural Info. Proc. Systems (NIPS)*. MIT Press, 1999, pp. 1015–1021.

[12] S. Thrun, W. Burgard, and D. Fox, *Probabilistic Robotics*. MIT Press, 2005.

[13] G. Grisetti, C. Stachniss, and W. Burgard, "Improved techniques for grid mapping with rao-blackwellized particle filters," *IEEE Transactions on Robotics*, vol. 23, pp. 34–46, Feb 2007.

[14] N. Fairfield, G. Kantor, and D. Wettergreen, "Towards particle filter slam with three dimensional evidence grids in a flooded subterranean environment," in *Proc. of the IEEE Int. Conf. on Robotics and Automation (ICRA)*, Orlando Florida USA, May 2006.

[15] D. Mount and S. Arya, "ANN: Approximate nearest neighbors," http://www.cs.umd.edu/ mount/ANN/, August 2006.

[16] M. B. Kennel, "Kdtree 2: Fortran 95 and C++ software to efficiently search for near neighbors in a multi-dimensional euclidean space," http://arxiv.org/abs/physics/0408067v2, Institute For Nonlinear Science, 2004.