Journal of Intelligent & Robotic Systems (2018) 91:617–628
https://doi.org/10.1007/s10846-017-0754-8

Exploring Unstructured Environment with Frontier Trees

R. Korb[1] · **A. Schöttl**[1]

Received: 27 January 2017 / Accepted: 1 December 2017 / Published online: 19 December 2017

Abstract
This paper presents the frontier tree exploration algorithm, a novel approach to autonomously explore unknown and unstructured areas. Focus of this work is the exploration of domestic environments with lots of arbitrary obstacles, for example furbished appartements. Existing and well-studied approaches like greedy algorithms perform worse when obstacles are included and the range of distance sensors is limited. Base of this research is the frontier tree. This data structure offers two main features. Firstly it serves as a memory of past poses during exploration, where boundaries between known and unknown space are inserted into the tree. Secondly, the data structure is then utilized to decide between future navigation goals. This approach is compared to the class of nearest neighbor exploration algorithms and a decision theoretic approach. The algorithm is tested in a simulation study with furbished ground maps and in a real life domestic environment. The paper shows, that frontier trees exhibit a superior performance of distance traveled and the number of exploration steps compared to a nearest neighbor algorithm.

Keywords Robotics · Exploration · Path planning · Frontiers · Domestic environment

1 Introduction

Mobile robots must be aware of their surroundings to execute complex tasks, for example pick-up and delivery services. As an initial step, the workspace has to be explored autonomously. This problem can be divided into three parts: mapping, localization and exploration. Focus of this research is the efficient exploration of an unstructured area with dependencies on past robot states, a map of the area already explored and the agent's current position.

When robots participate in everyday life, their operation space consists of unstructured environment. For example, they have to deal with furniture in all shapes turning geometrically simple rooms into complex areas. As a consequence, when service robots operate in private apartments or public buildings like hospitals or retirement homes, they need to adapt themselves to their surroudings. Therefore it is crucial for a mobile robot to explore areas autonomously and not to rely on a prepared map designed specifically for a service device.

Basic algorithms, for example frontier exploration by Yamauchi [5], follow a greedy approach to select robot positions during the exploration process. These approaches use the robot's current state and plan one step ahead by minimizing a cost function, for example the agent's Euclidian distance to a boundary between known and unknown space.

These algorithms provide reasonable results for the exploration of simple maps, which are typical for unfurbished rooms and corridors. Robots endowed with high range distance sensors in obstacle free areas, may cover entire rooms before leaving and do most likely not create open boundaries. Therefore, rooms do not have to be visited again and the traveled distance is less compared to low range sensing robots. The exploration performance of greedy algorithms decreases significantly if the sensor range is limited or the environment contains a lot of obstacles because the number of possible candidates to choose the next goal from rises with the environment's complexity. Greedy algorithms suffer from the fact that the robot has

✉ R. Korb
rudolf.korb@hm.edu

A. Schöttl
alfred.schoettl@hm.edu

[1] Department 04 Electrical Engineering and Information Technology, University of Applied Sciences, Lothstr. 64, 80335 Munich, Germany

to revisit parts of the map several times at locations where boundaries have been left open. This behavior increases the travel distance especially at the end of exploration, when all open positions in different regions of the map have to be covered.

Consequently, the distance traveled during exploration depends on the complexity of the environment and on constraints of the range sensors. Mobile autonomous robots operating in unstructered environment have to deal with both, high complexity and sensor limitations. An indication of an improvable travel distance is the presence of cycles in the exploration path, which appear when the robot is approaching a frontier already discovered. Figure 1 shows a small part of a map with obstacles and range constraints where the exploration path includes a cycle. A greedy algorithm results in missed frontiers inside the room. In comparison, the frontier tree algorithm uses a tree data structure as a memory of past exploration states. This is used to recognize cycles and hence is able to avoid high costs by reacting early on forgotten boundaries.

2 Related Work

Yamauchi [5] introduced the frontier approach as exploration algorithm. A frontier is defined as the boundary between known and unknown space in the currently explored map. The frontier tree algorithm also utilizes frontiers as atomic elements in the tree data structure, discussed in detail in Section 4.3. Yamauchi's exploration algorithm is a greedy approach and chooses the shortest obstacle free path to minimize the distance between the current position and the next frontier.

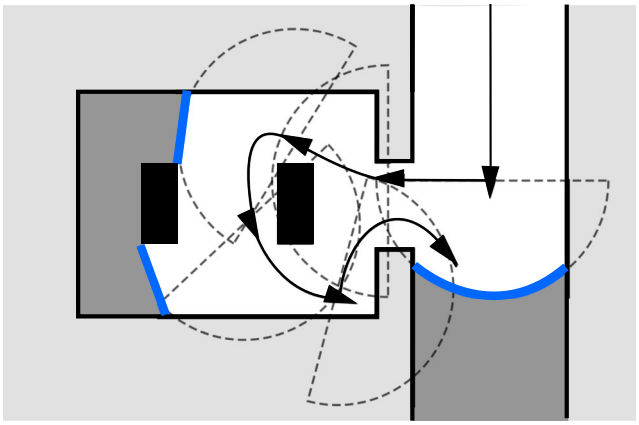


Fig. 1 The figure shows six robot states during exploration with a cycle in the exploration path

Koenig et al. [19] show that greedy algorithms explore areas with reasonable performance. Map segmentation algorithms [6, 11] were introduced to reduce the travel distance of greedy approaches. Wurm et al. [11] use Voronoi graphs to segment the explored environment. The segmentation prevents the agent from visiting areas multiple times by setting a priority to finish exploration of each segment. While large empty spaces are segmented well with Voronoi graphs, narrow areas such as corridors yield multiple partitions. Holz et al. [6] extend the segmentation algorithm by adding an additional step to merge several partitions to further reduce the complete travel distance.

Wattanavekin et al. [20] also minimize the exploration path. Main goal in their approach is to optimize the trajectory by replanning during the exploration process, but multiple visits are not taken into account.

Nasir and Elnagar [17] use a data structure called Gap Navigation Tree, introduced by Tovar et al. [4] to handle map complexity with arbitrary obstacles and to track open boundaries, but assume infinite range measurements at 360° opening angle. The process of appending reachable exploration goals to the data structure is similar between the frontier tree and Gap Navigation Trees. The main differences are the definition of frontiers, the data synchronisation process and the heuristic for goal decision. Freda and Oriolo [13] present a probabilistic algorithm using Sensor-based Random Trees. Franchi et al. [2] introduce the Sensor-based Random Graphs. Compared to [4, 17] and the proposed algorithm both data structures do not only rely on the current robot configuration. They sample different agent configuration to get the most promising exploration goal. Since this approach does not exclude multiple visits, El-Hussieny et al. [9] extend the algorithm by introducing a backtracking heuristic.

Another approach is to use information-based heuristics [1–3, 18]. Visser and Slamet [3] minimize information entropy by selecting frontiers with the largest area. They achieved an improved performance by utilizing an extended cost function, which also takes into account the travel distance. In [1], Mobarhani et al. discuss the tradeoff between frontier size and travel distance. They count the number of points of all frontiers at an angle interval from the robot's current position and fill an angle histogram. The next best frontier is selected with a weighted cost funtion by maximizing the number of frontier cells at a specific direction and minimizing the travel time to the frontier.

Information based approaches are especially useful for rescue scenarios. Focus of these algorithms is to uncover large parts of the map quickly. It is possible that these heuristics result in alternating poses between opposite parts of the map.

Landa et al. [21] create a visibility map to explore an unknown environment. They take into account limited range sensor data.

The frontier tree algorithm uses loop closures and cycle detection as opportunity to detect missed frontiers during the exploration process. Modern SLAM systems like ORB Slam [16], RTAB-Map [14, 15] or RGBD-Slam[7] with an underlying graph structure heavily rely on loop closure detection utilizing visual features of RGB or depth sensors. Lenac et al. [12] combine exploration and SLAM to an active system. Loop closures with SLAM and the frontier tree algorithm is discussed in Section 6.

3 Outline

In this paper, we introduce the frontier tree algorithm. Section 4 presents our approach in detail. All parts of the algorithm use different map types which are discussed in Section 4.2. For initialization, frontiers associated with the current position are extracted. Frontiers are described in 4.3. The exploration process is partitioned into several steps and continued until no more frontiers are extracted from the map. During an exploration step, maps based on a global map are generated (Section 4.2). The next part is the extraction of three frontier sets (Section 4.4.3), which are the foundation of the tree synchronization process shown in 4.5. Section 4.6 shows the determination of the next exploration goal based on the detection of cycles in the exploration path. After executing the planned path, the next exploration step is started by the extraction of the new frontiers.

The results of the frontier tree algorithm are presented in Section 5, where the travel distance, the average travel distance per exploration step, the number of single exploration steps and the relation between travel distance and map coverage of our approach is compared to a nearest neighbor exploration and a decision theoretic approach. Lastly, the results are discussed in Section 6.

4 The frontier Tree Algorithm

Foundation of the exploration algorithm is a tree data structure with frontiers as nodes. Similar to Gap Navigation Trees [4], the data structure is used for tracking open boundaries during exploration. Additionally, the resulting topological tree map is used to select the next exploration goal. The final path planning to the new pose is executed on a 2D grid map. Listing 1 shows a pseudo-code implementation of our algorithm.

Algorithm 1 The frontier tree algorithm

```
1  Function explore is
       Input: r, occ_g
2      ftree ← initializeFrontierTree(r);
3      F^t ← extractFrontiers(occ_g);
4      while F^t ≠ ∅ do
5          goal ← explorationStep(r, ftree, occ_g, F^t);
6          [r, occ_g] ←executeAction(goal);
7          F^t ← extractFrontiers(occ_g);
8      end
9  end
10 Function explorationStep is
       Input: r, ftree, occ_g, F^t
       Output: navigation goal
       /* Section 4.2: Generate maps */
11     occ_l ← generateLocalOccupancyMap(occ_g);
12     cost_{g,l} ← generateCostMaps(occ_g, occ_l);
       /* Section 4.4.3: Determine
          frontier sets */
13     [F^t_in, F^t_out] ← extractFrontierSets(F^t, cost_l);
14     F^{t-1} ← ftree.getUnmarkedNodes();
       /* Section 4.5: Tree
          synchronization */
15     D ← DistanceMatrix(F^{t-1}, F^t_out, cost_g);
16     B ← BinaryRelationMatrix(D);
17     s ← SumVector(B);
18     marked nodes ← ftree.synchronize(D, s);
19     ftree.insert(F^t_in)
       /* Section 4.6, 4.7: Cycle
          handling */
20     if isCycle(r, marked nodes) then
21         goal ← ftree.twoStepSearch();
22     else
23         goal ← extractNearestNeighbor(r, ftree);
24     end
25     return goal
26 end
```

4.1 Robot Model

In this paper, the robot is modeled as mobile base with a circular footprint and three degrees of freedom. The position and orientation is defined by $\mathbf{r} = (x, y, \theta)^T$ with position (x, y) and orientation θ. The agent is equipped with a proximity sensor, represented by its configuration $\mathbf{s} = (d, \alpha)^T$. Component d is the sensor's range and α its opening angle. Localization is performed on a 2D grid map,

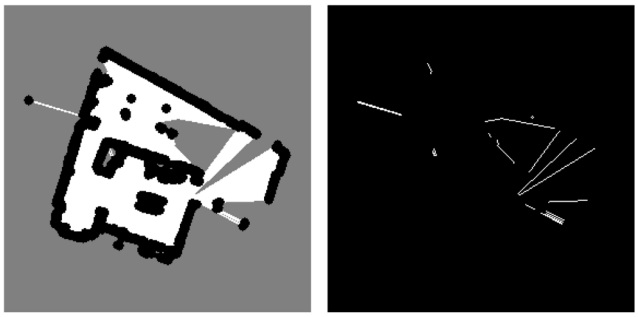
using a simultaneous localization and mapping (SLAM) approach.

4.2 Map Representations

To fulfill the exploration task, the algorithm uses five maps of three different types. A basic map type is the *occupancy grid map* (see Fig. 2a) with values in [0, 1], where white pixels (1) mark free and black pixels (0) occupied space. Values in the interval (0, 1) can be interpreted as occupancy probability. An occupancy map is the output of a SLAM algorithm. Occupied pixels are inflated by the radius of the robot's footprint, resulting in a *global occupancy map* occ_g where the agent can be treated as a point. occ_g is used to create a *local occupancy map* occ_l.

$$\text{occ}_l(\mathbf{x}) = \begin{cases} 0.5, & \text{if } \|\mathbf{x} - \mathbf{r}\|_2 \geq l \\ \text{occ}_g(\mathbf{x}), & \text{otherwise} \end{cases} \quad (1)$$

where $\mathbf{x}$ is two dimensional vector representing a cell in the map, $\mathbf{r}$ is the current position of the robot and l defines the map radius. l is set to the sum of the sensing distance d and the footprint radius. With addition of the footprint cells in the local maps with distance d are approached safely.



(a) Occupancy map (b) Frontier map

(c) Cost map (d) Local cost Map

Fig. 2 **a** The *global occupancy map* stores the knowledge of the robot about its surroundings with white pixels as free, black as occupied and grey as unknown space. **b** Exploration goals for the next steps are displayed in the *frontier map*. **c** The *global cost map* combines information about travel costs and reachable points. The darkest pixels are not reachable and travel costs rise with pixel brightness. **d** The *local cost map* shows travel costs and reachable points from $\mathbf{r}$ to a distance limit l around the robot

Both, occ_g and occ_l, as well as the current pose $\mathbf{r}$ are utilized for the calculation of a *global cost map* cost_g (Fig. 2c) and a *local cost map* cost_l (Fig. 2d). Cell values of cost_g and cost_l are the costs for travelling from $\mathbf{r}$ to $\mathbf{x}$, computed with Dijkstra's algorithm taking occupancy grid maps as input.

Lastly, occ_g is used to create a *frontier map* F, shown in Fig. 2b, containing the possible future goals for exploration. Since occ_g is an inflated representation, every point in F is a possible navigation goal. Section 4.3 gives a formal description of frontiers.

4.3 Frontiers

Frontiers are extracted from the inflated *global occupancy map*. A frontier f^i, is the boundary between unknown and free space in occ_g, as defined by Yamauchi [5]. An elementary part of frontier f^i are *frontier cells* f^i_j, where subscript j denotes the index of the single cell inside the frontier. A cell is defined by the following two properties:

$$\text{occ}_g(\mathbf{x}) = 1 \quad (2)$$

$$\exists\, \text{occ}_g(\mathbf{y}) \in N_8(\text{occ}_g(\mathbf{x})) : \text{occ}_g(\mathbf{y}) = (0, 1) \quad (3)$$

with N_8 as the Moore neighborhood of a free cell $\mathbf{x}$ in the global occupancy map occ_g and cells $\mathbf{y}$ as part of N_8 of $\mathbf{x}$. Adjacent frontier cells are merged to a single frontier f^i. $\mathbb{F}$ is the set of all frontiers f^i. A *center cell* f^i_g is assigned to every frontier. It specifies a navigation goal point of a path from $\mathbf{r}$ to f^i. This position is defined as

$$f^i_g = \underset{f^i_j}{\text{argmin}}(\|f^i_j - \bar{f}^i\|_2), \quad (4)$$

minimizing the Euclidian distance between each f^i and the mean $\bar{f}^i$ of all frontier cells.

4.4 Frontier Tree

4.4.1 Tree Structure

The root of the frontier tree structure is the initial position $\mathbf{r}_{\text{init}}$ of the robot on the map. The number of children of a node depend on $\mathbf{r}$ and cost_l. Every node n in the frontier tree is associated to a frontier f^i with f^i_g as the navigation goal of node n. Nodes can be in one of the states *unmarked*, *marked* or *visited*. All *unmarked* nodes are potential exploration goals, which is the default state after appending a new node to the frontier tree. *Marked* nodes are not used anymore for future planning which is equivalent to a deletion operation. A node is *visited*, when it has been chosen as exploration goal. As long as children of the current node are selected as exploration goals, the tree grows in depth. When a branch is fully explored, and

all direct children of a node are *marked*, and there exist *unmarked* nodes, a subtree is generated and the frontier tree grows in breadth. For traversing the data structure, a breadth first search was implemented.

4.4.2 Tree Operations

The frontier tree offers two basic operations to directly manipulate the data structure. An append function extends an existing leaf node by attaching an element.

The second operation is setting nodes into the marked state. These nodes are not considered any further. All nodes in the tree structure not being leaves, are implicitly set into the visited state.

4.4.3 Frontier Sets

Let $\mathbb{F}^t$ be the set of all frontiers in frontier map F (Fig. 2b) at exploration step t. Initially, $\mathbb{F}^t$ is partitioned in two disjoint subsets

$$\mathbb{F}^t_{\text{in}} = \{f^i \in \mathbb{F}^t \mid \text{cost}_l(f^i_g) > 0\} \tag{5}$$

containing all frontiers in the sensing range and

$$\mathbb{F}^t_{\text{out}} = \mathbb{F}^t \backslash \mathbb{F}^t_{\text{in}}, \tag{6}$$

with $\mathbb{F}^t_{\text{out}}$ as the set of all frontiers $\mathbb{F}^t$ excluding $\mathbb{F}^t_{\text{in}}$. Let $\mathbb{F}^{t-1}$ be the frontier set of the previous exploration step. Every *unmarked* leaf in the frontier tree represents one element in $\mathbb{F}^{t-1}$. Since the current frontier $f^{i,t}_g$ is visited now, we exclude it by

$$\mathbb{F}^{t-1} := \mathbb{F}^{t-1} \backslash \{f^{i,t}_g\} \tag{7}$$

4.5 Tree Synchronisation

Before the frontier tree can be used for planning next exploration goals, the nodes representing $\mathbb{F}^{t-1}$ are synchronized with the frontiers of the current exploration step $\mathbb{F}^t$. There exist two cases where nodes are marked: First, when unknown space is uncovered and, as a result, frontiers disappear during the execution of the planned path to the navigation goal. Secondly, frontiers previously discovered are now in the set $\mathbb{F}^t_{\text{in}}$.

Let $\mathbf{D} = (\mathbf{d}_{i,j})$ be the *distance matrix* between the frontier sets $\mathbb{F}^{t-1}$ and $\mathbb{F}^t_{\text{out}}$:

$$\mathbb{F}^{t-1} \times \mathbb{F}^t_{\text{out}} \longrightarrow \mathbb{R} \tag{8}$$

$$\mathbf{D} : (f^{i,t-1}_g, f^{j,t}_g) \longmapsto \| f^{i,t-1}_g - f^{j,t}_g \|_2 \tag{9}$$

The cartesian product $\mathbb{F}^{t-1} \times \mathbb{F}^t_{\text{out}}$ is mapped to $\mathbb{R}$ by calculating the Euclidian distance between each element.

The closest frontiers from elements of $\mathbb{F}^{t-1}$ to $\mathbb{F}^t_{\text{out}}$ are noted in a *binary relationship matrix* $\mathbf{B} = (\mathbf{b}_{i,j})$. A relationship between frontiers is defined as

$$\mathbf{b}_{i,j} = \begin{cases} 1: & d_{i,j} = \underset{k}{\text{argmin}}(d_{i,k}) \\ 0: & \text{otherwise} \end{cases} \tag{10}$$

where k is the column index to extract the minimum of the i-th row in distance matrix $\mathbf{D}$.

Let $\mathbf{s}$ be the sum vector of $\mathbf{B}$, to identify the number of related frontiers of each element in $\mathbb{F}^{t-1}$, which is calculated by a sum of the row elements $\mathbf{b}_{*,j}$, with $|\mathbb{F}^{t-1}|$ as the cardinality of the frontier set of the previous exploration step.

$$\forall \mathbf{b}_{*,j} : \mathbf{s}_j = \sum_{i=1}^{|\mathbb{F}^{t-1}|} \mathbf{b}_{i,j} \tag{11}$$

We distinguish between three cases:

(a) $\mathbf{s}_j = 0$:
With $\mathbf{s}_j = 0$, $f^{j,t}_g$ does not have a relation to any of the leafs in the frontier tree. Since there exist other frontiers of $\mathbb{F}^t_{\text{out}}$ which have a smaller distance to the elements of $\mathbb{F}^{t-1}$, we insert $f^{j,t}_g$ in the tree to establish a new relation. Since $f^{j,t}_g$ is discovered during path execution, the node is added to the parent element of the current exploration goal.

(b) $\mathbf{s}_j = 1$:
When $\mathbf{s}_j$ takes the value 1, $f^{j,t}_g$ is related to exactly one element $f^{i,t-1}_g$ and is considered as potential goal in the further exploration process.

(c) $\mathbf{s}_j > 1$:
If $\mathbf{s}_j > 1$, there exist multiple relations between $f^{j,t}_g$ and the elements of $\mathbb{F}^{t-1}$. In this case, leaves are put into the marked state because they either do not exist anymore or they are in the currently reachable area as an element of $\mathbb{F}_{\text{in}}$. Every node $f^{i,t-1}_g$ is marked, if $d_{i,j} > \underset{k}{\text{argmin}}(d_{i,k})$.

4.6 Cycles

Frontier trees can already be used as a greedy exploration algorithm, with the next navigation goal being a node with the smallest distance.

Key feature of the proposed algorithm is the additional use of the tree data structure as topological map to improve the exploration process. Initially, the frontier tree algorithm follows the greedy idea. The next exploration goal is selected among the children of the current node in the tree until a cycle in the exploration path is detected. The presence of a cycle is detected, when

$$\text{rank}(f^c_g) - \text{rank}(f^i_g) > 1 \tag{12}$$

where $\text{rank}(f)$ is the height of the tree structure at the position of frontier f with a precondition that f^i_g

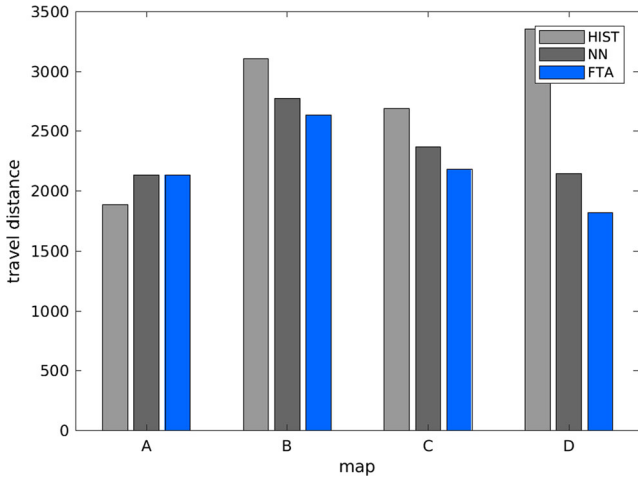


Fig. 3 Complete travel distance on maps A - D of the histogram based exploration, nearest neighbor and the frontier tree algorithm

was marked during the tree synchronisation. With height differences larger than one, the area currently approached was visited several exploration steps before. With detection of the cycle, the exploration behavior is adapted to the new situation to eliminate shortcomings of nearest neighbor approaches.

The frontier tree as memory of the exploration process recognizes previously visited areas. The algorithm has the ability to examine the traveled path to find and react properly to open frontiers.

4.7 Exploration Goal Selection

The algorithm uses a two step bottom-up approach to distinguish the next exploration goal as soon as a cycle is identified while iterating over the marked nodes. At first, the search algorithm traverses the tree starting at the marked node up to the frontier tree's root node. If the first search step yields an unmarked node, the goal selection is finished. The unmarked node is returned as next navigation goal. An unsuccessful first step leads to a second search step, where tree nodes from the current position up to the marked node are considered. If there exists an unmarked node in the second step, the node is returned as exploration goal. When both iterations are not successful, the node with the shortest path from the current position is selected.

When several children of a node are unmarked, the first occurence of an unmarked node is selected as navigation goal. There is no differentiation in the picking process since siblings of an unmarked node are spatially related.

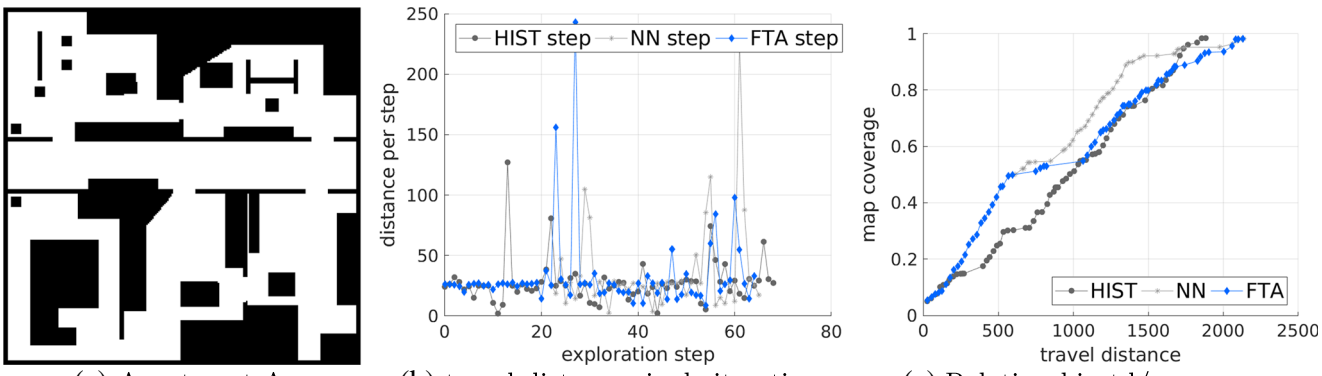
Unmarked nodes of the first search step represent global landmarks and map structures of the exploration are e.g. transitions between two rooms. These *global cycles* also have a high difference between the nodes ranks. When nodes are returned in the second search step, the exploration path is a *local cycle*, representing local landmarks and structures, for example chairs, tables or other furniture.

When retrieving the next exploration goal, the path from the current position to the navigation goal is executed. After arrival, the frontiers in the global occupancy map are extracted and the next exploration step is planned by the frontier tree algorithm until no frontiers are left in occ_g and the whole map is explored.

5 Results

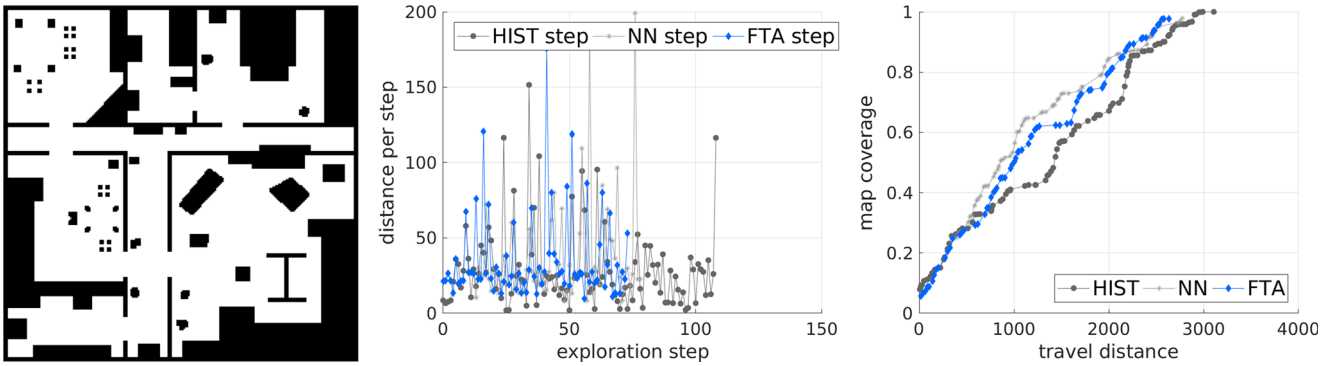
5.1 Simulation

For performance evaluation, the frontier tree algorithm is executed in a simulated environment. To add visibility constraints, the exploration area has the dimensions $240px \times 240px$. The robot's footprint has a radius of $2px$ and a sensor with $d = 30px$ and $\alpha = 180°$. The test maps represent apartments including furniture as complex objects



(a) Apartment A (b) travel distance single iterations (c) Relationship td/coverage

Fig. 4 Result dataset map A

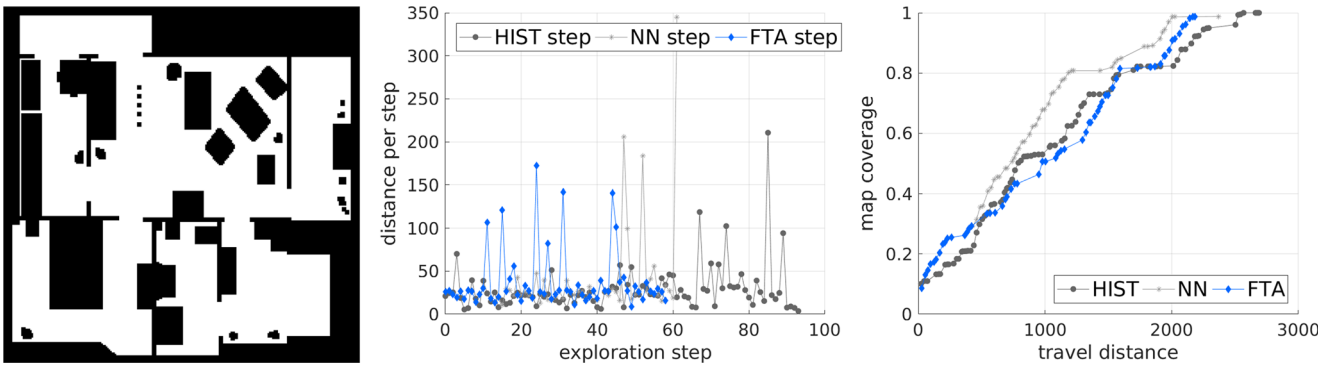


(a) Apartment B (b) travel distance single iterations (c) Relationship td/coverage

Fig. 5 Result dataset map B

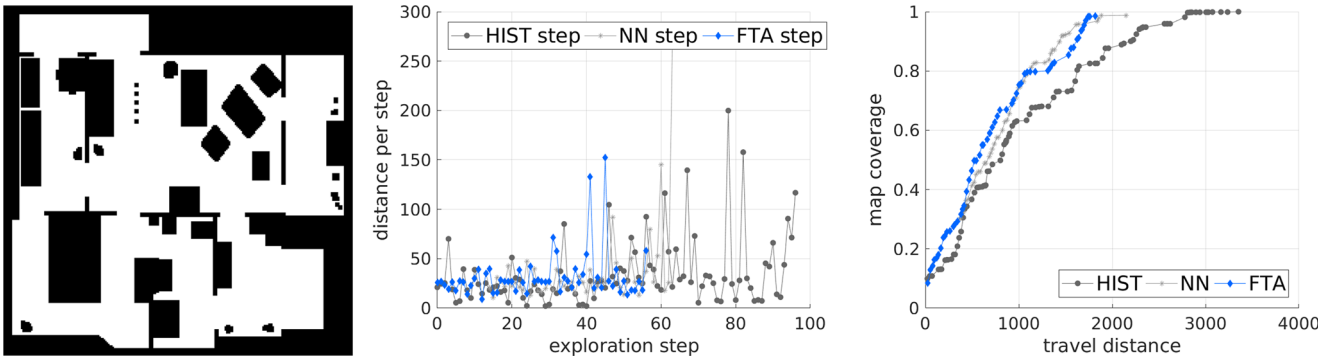
since a suitable dataset for evaluating domestic environment does currently not exist. The algorithm is tested on self-created maps. The performance criterions are the complete travel distance by the mobile robot from start to the fully explored map, the average distance of a single exploration step, and the number of exploration steps of the exploration process. These values are then compared to a histogram based exploration approach (HIST) [1] and the nearest neighbor algorithm (NN). The HIST algorithm takes into account the proximity and the number of frontier points at a certain angle of the robot position. Frontiers are organized in an angle histogram. The trade off between frontier distance and size is handled with weighting parameters. During tests the distance weights are priorized. The greedy approach chooses the next exploration step by minimizing the travel cost on the inflated map cost_g, calculated with Dijkstra's algorithm. Nearest neighbor uses Yamauchi's definition of frontiers [5], active loop closing is implemented in the sense of Lenac et al. [12]. Due to the rich structure of the environment, which is typical for household applications, the estimation variances of the localization filter remained low during the whole campaign and active loop closing was not activated.

The results of the comparison between HIST, NN and the frontier tree algorithm (FTA) are presented in Table 1. The complete travel distance is reduced on maps B, C and D when using the FTA. Compared to HIST map A has a decreased efficiency by 13.17% but increases up to 45%. While on map A and the NN approach a reduction of 0.23% is neglectable, efficiency of exploring map D can be increased by up to 15.24%. Figure 3 shows the comparison lined up for each map in a bar graph. Table 2 shows the results for the average distance traveled for each exploration step. The average distance for each step compared to NN is reduced by up to 4.83% with an average value of 31.92, approaching the sensor range d. HIST shows a significantly lower value for average distances but on the opposite significant higher values for exploration steps. A comparison of the number of steps by HIST, NN and FTA is shown in Table 3, where the number of steps to explore a map completely is less compared to HIST and NN for all maps by up to 41.24%. Figures 4a, 5, 6 and 7a present the simulated apartment maps A-D. Figures 4b–7b show the results including the distance traveled for each exploration step in environments A-D, where ∘-graph is the histogram approach, ∗-graph is the nearest neighbor approach and the



(a) Apartment C (b) travel distance single iterations (c) Relationship td/coverage

Fig. 6 Result dataset map C



(a) Apartment D (b) travel distance single iterations (c) Relationship td/coverage

Fig. 7 Result dataset map D

◇-graph the frontier tree algorithm. The graphs of Figs. 4c–7c show the relationship between travel distance and the percentage of map coverage. The slope shows the rate of uncovering the map while linearity indicates a steady gain of the map coverage.

5.2 Real World Implementation

The frontier tree algorithm is implemented on real hardware to validate the results gathered during simulation. FTA runs in the combination of ROS and the turtlebot platform, equipped with a Hokuyo URG-04LX-UG01 laser scanner with a distance of 5.6m and an opening angle of 180°. An implementation of a particle filter is used as SLAM method [8]. The resulting occupancy grid map is the input to the frontier tree algorithm. The robot explores a domestic environment with an area of approximately 42.5m^2. The results between three test runs of NN and FTA are shown in Table 4. The real world experiment confirms the significant increase in efficiency of the proposed method by up to 20.7%.

Figure 8a is a photo of the explored environment. The related map including the trajectories of FTA (solid) and NN (dashed) are shown in Fig. 8b. Figure 8c contains the relationship of the normalized travel distance to map coverage in %.

NN builds up the map quick but has to spend half the time of exploration for uncovering the last 10-40% (dataset 1, dataset 3). In comparison, FTA has a more linear approach for the exploration task.

5.3 Execution Time

Downside of the current implementation of the frontier tree algorithm is an increased computation effort for maintaining multiple maps and the tree structure which leads to an increase in execution time in simulation and the real world experiment. The NN algorithm needs an average of 5 ms in the range of 3 ms to 8 ms for each exploration step. Computation time for an iteration with frontier trees is at an average of 60 ms ranging from 30 ms to 80 ms. The calculations where performed on an Intel i5-4200U CPU/1.60GHz and non-optimized code for both algorithms.

6 Discussion

This section interprets the results presented in Tables 1, 2 and 3. Two exploration decisions of the frontier tree algorithm are dicussed in detail to demonstrate the benefits compared to nearest neighbor algorithms. Following the

Table 1 Comparison of travel distance between a histogram based exploration approach (HIST), nearest neighbor (NN) and the frontier tree algorithm (FTA) on maps A-D

	HIST TD	NN TD	FTA TD	CMP HIST %	CMP NN %
A	1883.5	2136.6	2131.6	13.17	−0.23
B	3107.6	2776.2	2632.9	−15.28	−5.16
C	2689	2368.7	2182.4	−18.84	−7.86
D	3352.4	2146.9	1819.	−45.72	−15.24

Table 2 Comparison of the average travel distance for each step between a histogram based exploration approach (HIST), nearest neighbor (NN) and the frontier tree algorithm (FTA) on maps A-D

	HIST avg	NN avg	FTA avg	CMP HIST %	CMP NN %
A	27.3	32.37	32.79	20.14	1.3
B	28.51	35.14	35.58	24.79	1.2
C	28.6	38.20	36.99	29.31	−3.18
D	34.56	33.54	31.92	−7.63	−4.83

Table 3 Comparison of the exploration steps between a histogram based exploration approach (HIST), nearest neighbor (NN) and the frontier tree algorithm (FTA) on maps A-D

	HIST steps	NN steps	FTA steps	CMP HIST %	CMP NN %
A	69	66	65	−5.8	−1.15
B	109	79	74	−32.11	−6.33
C	94	62	59	−37.23	−4.84
D	97	64	57	−41.24	−10.9

properties of the frontier tree and modern SLAM systems are mentioned.

6.1 Complete Travel Distance and Distance for Single Exploration Steps

All graphs in Figs. 4b–7b show large distances for exploration steps towards the end of exploring with the nearest neighbor approach. This is an indication for missed frontiers during the exploration process. In comparison, the frontier tree algorithm does not have high peaks at the end of the exploration process. This behavior which shows that missed frontiers are found early, prevents an accumulation of forgotten frontiers towards the end of exploration.

On map A and B, FTA's average travel distance is higher compared to NN since FTA does not always choose the nearest frontier and accepts higher distances to avoid a returning with higher travel costs.

The distances for each exploration iteration in apartment A (Fig. 4) show lower variance compared to the other maps in Figs. 5–7, which leads to frontier trees growing in depth and leveling distance traveled between nearest neighbor and the frontier tree algorithm. The structure of maps B, C and D offers a combination of local cycles, global cycles and rooms with one entrance so the frontier tree grows in breadth.

The graphs in Fig. 5b of the exploration of apartment B, which contains many single rooms without direct transitions to other rooms, show high variance throughout the exploration process. The exploration of the map is partly finished when two consecutive exploration steps yield a reasonable increase of average travel distance. This situation is typical for a fully explored room.

Table 4 Comparison of travel distance between nearest neighbor (NN) and the frontier tree algorithm (FTA) implemented on the turtlebot platform

	NN TD in m	FTA TD in m	CMP NN %
Run 1	43.79	41.52	−5.1
Run 2	47.83	37.93	−20.7
Run 3	34.88	28.01	−19.6

6.2 Frontier Tree Properties

Figure 9a and b illustrate the handling of a local cycle on map B of FTA compared to NN. The related frontier tree (FT) is shown in Fig. 11a. When reaching exploration step 13 at node 19 (Fig. 12), node 4 is in the sensing radius again and is therefore marked, leading to a cycle identification. Since no goal is found when searching from the marked node 4 to the root element, the cycle can be classified as local. Local cycles initialize the final exploration of missed frontiers in the current global room structure where the robot is located. Figure 9c and d show that the NN approach misses the frontier in the north covered by FTA after detecting this cycle. As shown in the graph in Fig. 5b, FTA explores the respective frontier at node 18 with a travel distance of approximately 75px at exploration step 14. This specific frontier is explored at step 77 in the graph of NN in Fig. 5b, resulting in a travel distance for this step greater than 200. The decision to explore the missed frontier early leads to a saving of about 62.5% for this particular step.

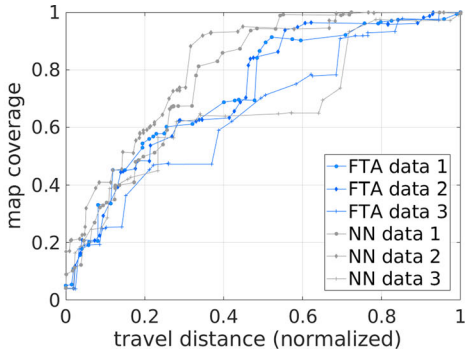
Figure 10a and b show the detection of a small loop when the robot moves around an obstacle. The frontier tree of map C (Fig. 11b) shows that there is no open frontier between the current node 18 and marked node 5. However, the two step search will find node 3 as exploration goal during the first iteration when searching from the marked node towards the tree root and identify a global cycle. In contrast to local cycles, these loops often start the exploration of new global structures like rooms. The NN algorithm (Fig. 10c and d) follows the frontier in north west direction and will return south east at the end of exploration. This path leads to a missed frontier in the top left corner, which is selected as last step with a distance traveled for this step of $350px$. With the frontier tree algorithm, the bottom room is already explored due to the detection of the global cycle. Similar to NN, FTA leaves the top left frontier but recognizes it as a local cycle so that it is going to be explored early, saving further travel cost.

6.3 SLAM and loop Closing

The frontier tree algorithm uses spatial cycles to update the current exploration strategy which are identified by a simple comparison of the nodes' ranks. For the implementation, only the occupancy grid map is used as continuous input generated by the SLAM system. Modern localization and mapping methods [7, 10, 14–16] rely on loop closure detection by matching for example visual features or

(a) Photo of the environment

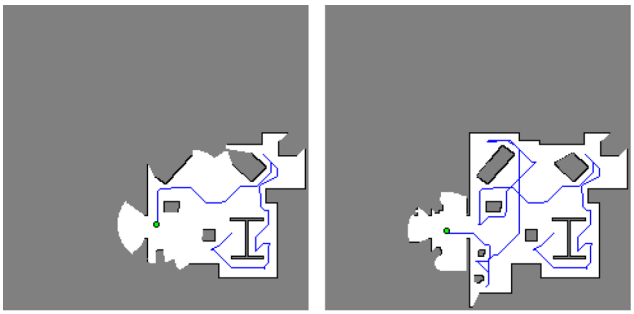
(b) Acquired map and trajectories for FTA (solid) and NN (dashed)



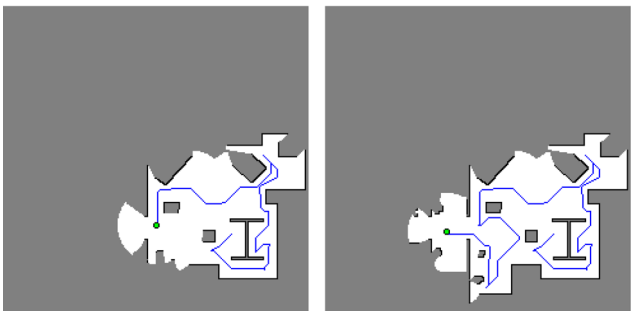
(c) Relationship between the normalized travel distance and map coverage

Fig. 8 Results of the real world experiments

proximity sensor scans. Loops are used to initiate a trajectory optimization process and are an essential part of SLAM algorithms to decrease the uncertainty of the robot's position and increase the robustness significantly. These loop closure detections can be used as a second input to the frontier tree algorithm for a replacement of the current cycle detection. We implemented the method proposed by Lenac et al. [12]. As already stated, the application provides a feature rich environment to maintain a good observability state. Another possibility is to include the frontier tree approach as an exploration algorithm to the active SLAM method.

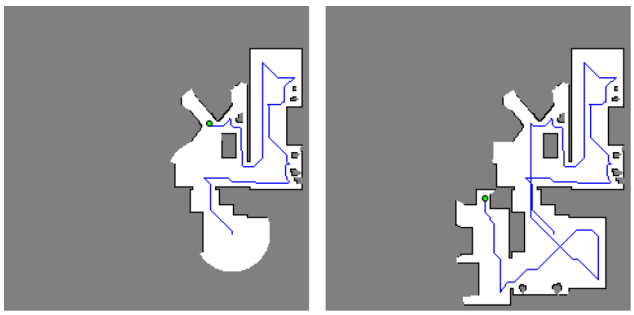


(a) Apt. B FTA exp. step 13 (b) Apt. B FTA exp. step 19

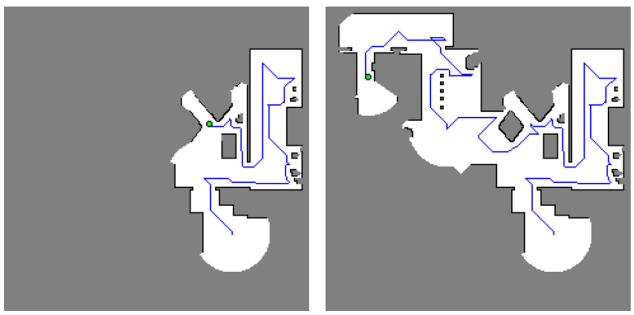


(c) Apt. B NN exp. step 14 (d) Apt. B NN exp. step 18

Fig. 9 Behaviour of FTA compared to NN, when a local cycle is detected. There are no frontiers to explore between the root node and the marked node. Figure 11a shows the related frontier tree

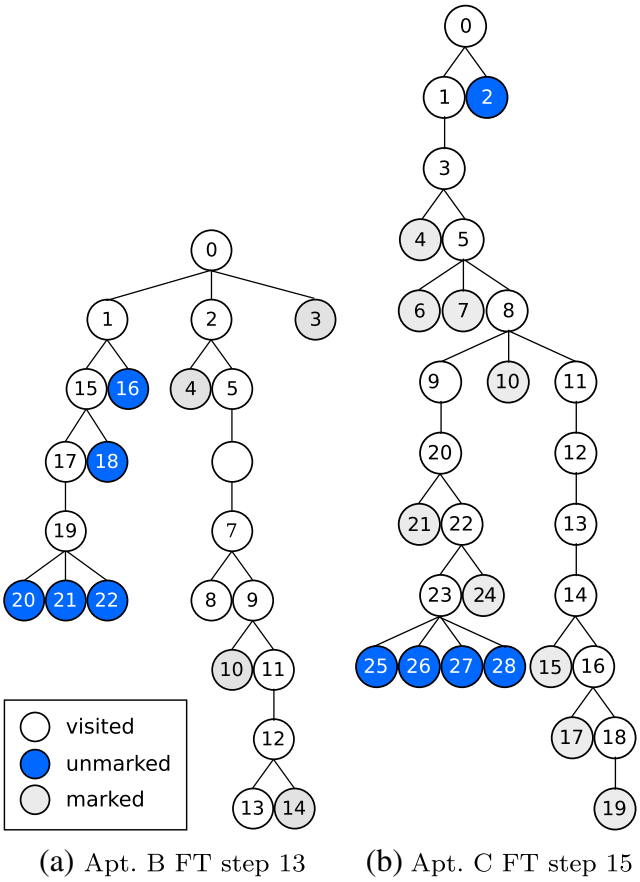


(a) Apt. C FTA exp. step 15 (b) Apt. C FTA exp. step 24



(c) Apt. C NN exp. step 15 (d) Apt. C NN exp. step 30

Fig. 10 Behaviour when detecting a global cycle. There exist frontiers between the root node and the marked node. Figure 11b shows the related frontier tree



(a) Apt. B FT step 13 (b) Apt. C FT step 15

Fig. 11 Frontier trees (FT) constructed during exploration showing **a**) a local **b**) a global cycle detection

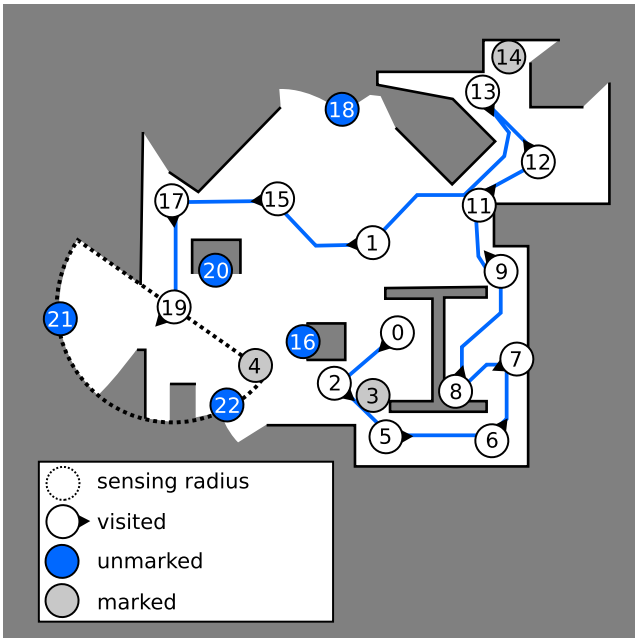


Fig. 12 Exploration path from start to step 13

7 Conclusion

This paper discusses cycles of the robot's path during the exploration task as main disadvantage of greedy approaches, resulting in missing frontiers and high travel cost towards the end of exploring the area.

The problem is tackled by the introduction of the frontier tree algorithm. It offers a tree data structure to detect cycles in the robot's path and a goal selection based on the created topological tree structure. The algorithm increases the efficiency of the complete travel distance, the travel distance for each exploration step and the average travel distance. The comparison of frontier trees show that they work especially well when the data structure grows in breadth, a phenomen typical for complex environments. The gain of the frontier tree algorithm can decrease on maps where frontiers can be followed many steps without the detection of a cycle or fully explored rooms, for example in maze-like environments. The algorithm yields stable results in unstructured environments, like furbished apartments which offer cycles and partly finished areas to initiate the goal selection on the frontier tree and open new exploration branches in the data structure. Single rooms inside the explored area are often found with their own branch in the frontier tree representing a spatial relationship. The proposed algorithm has shown a significant performance gain compared to the nearest neighbor approach in complex environments. With an added tree memory structure home environments can be explored with less exploration iterations, minimized travel distance and a steady uncoverage rate of the map. A drawback of this algorithm is the increased computational effort. This can be tolerated compared to the increase in travel distance towards the end of exploration when greedy algorithms update only few percent of the map.

References

1. Mobarhani, A., Nazari, S., Tamjidi, A.H., Taghirad, H.D.: Histogram based frontier exploration. In: Proceedings of the Conference on Intelligent Robots and Systems (IROS). 2011 IEEE/RSJ International, San Francisco (2011). https://doi.org/10.1109/IROS.2011.6095018
2. Franchi, A., Freda, L., Oriolo, G., Vendittelli, M.: The sensor-based random graph method for cooperative robot exploration. IEEE/ASME Trans. Mechatron. **14**(2), 163–175 (2009)
3. Visser, A., Slamet, B.A.: Balancing the information gain against the movement cost for multi-robot frontier exploration. In: Bruyninckx, H., Peuil, L., Kulich, M. (eds.) European Robotics Symposium 2008, Prague (2008)
4. Tovar, B., LaValle, S.M., Murrieta, R.: Optimal navigation and object finding without geometric maps Or localization. In: Robotics and Automation, 2003. Proceedings. ICRA '03 (2003). https://doi.org/10.1109/ROBOT.2003.1241638

5. Yamauchi, B.: A frontier-based approach for autonomous exploration. In: CIRA'97., Proceedings (2007). https://doi.org/10.1109/CIRA.1997.613851
6. Holz, D., Basilico, N., Amigoni, F., Behnke, S.: Evaluating the efficiency of frontier-based exploration strategies. In: Proceedings for the Joint Conference of ISR 2010 (41St International Symposium on Robotics) Und ROBOTIK 2010 (6Th German Conference on Robotics), Munich (2010)
7. Endres, F., Hess, J., Sturm, J., Cremers, D., Burgard, W.: 3-D mapping with an RGB-D camera. IEEE Trans. Robot. **30**(1), 177–187 (2014)
8. Grisetti, G., Stachniss, C., Burgard, W.: Improved techniques for grid mapping with Rao- Blackwellized particle filters. IEEE Trans. Robot. **23**(1), 34–46 (2007). https://doi.org/10.1109/TRO.2006.889486
9. El-Hussieny, H., Assal, S.F.M., Abdellatif, M.: Robotic exploration: new heuristic backtracking algorithm, performance evaluation and complexity metric. Int. J. Adv. Robot. Syst. 12(4), 33 (2015)
10. Hess, W., Kohler, D., Rapp, H., Andor, D.: Real-time loop closure in 2D LIDAR SLAM. In: 2016 IEEE International Conference on Robotics and Automation (ICRA), pp. 1271–1278. IEEE, Stockholm (2016)
11. Wurm, K.M., Stachniss, C., Burgard, W.: Coordinated multi-robot exploration using a segmentation of the environment. Nice, France (2008). https://doi.org/10.1109/IROS.2008.4650734
12. Lenac, K., Kitanov, A., Maurović, I., Dakulović, M., Petrović, I.: Fast active SLAM for accurate and complete coverage mapping of unknown environments. Intelligent Autonomous Systems **13**(302), 415–428 (2016)
13. Freda, L., Oriolo, G.: Frontier-based probabilistic strategies for sensor-based exploration. In: Proceedings of the 2005 IEEE International Conference on Robotics and Automation, Barcelona (2005)
14. Labbe, M., Michaud, F.: Appearance-based loop closure detection for online large-scale and long-term operation. IEEE Trans. Robot. **29**(3), 734–745 (2013)
15. Labbe, M., Michaud, F.: Online global loop closure detection for large-scale multi-session graph-based SLAM. In: Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems, pp. 2661–2666 (2014). https://doi.org/10.1109/IROS.2014.6942926
16. Mur-Artal, R., Montiel, J.M.M., Tardos, J.D.: ORB-SLAM: a versatile and accurate monocular SLAM system. IEEE Trans. Robot. **31**(5), 1147–1163 (2015). https://doi.org/10.1109/TRO.2015.2463671
17. Nasir, R., Elnagar, A.: Gap navigation trees for discovering unknown environments. Intell. Control. Autom. **2015**(6), 229–240 (2015)
18. Grabowski, R., Khosla, P., Choset, H.: Autonomous exploration via regions of interest. In: Proceedings of the 2003 IEEE International Conference on Intelligent Robots and Systems, Las Vegas (2003)
19. Koenig, S., Tovey, C., Halliburton, W.: Greedy mapping of terrain. In: Robotics and Automation, 2001. Proceedings 2001 ICRA, vol. 4 (2001). https://doi.org/10.1109/ROBOT.2001.933175
20. Wattanavekin, T., Ogata, T., Hara, T., Ota, J.: Mobile robot exploration by using environmental boundary information. ISRN Robotics 2013 (2013)
21. Landa, Y., Galkowski, D., Huang, Y.R., Joshi, A., Lee, C., Leung, K.K., Malla, G., Treanor, J., Voroninski, V., Bertozzi, A.L., Tsai, Y.-H.R.: Robotic path planning and visibility with limited sensor data. New York City (2007). https://doi.org/10.1109/ACC.2007.4282381

R. Korb is a PhD candidate at the University of Applied Sciences, Munich. He has a B.Sc. and M.Sc both in computer science from the University of Applied Sciences, Landshut. After graduation Rudolf worked in the field of medical robotics and image processing. His research interests are autonomous systems, localization, navigation, machine learning and perception.

A. Schöttl is professor for computer engineering and autonomous systems at the University of Applied Sciences, Munich. He graduated in mathematics and computer science at the TU Munich. Alfred received his doctorate in applied stochastics. After his habilitation, he has been a lecturer at the TU Munich. Afterwards, he worked in the field of aerospace enginieering in responsible positions. Focus of his research are autonomous robotic systems, including their perception, path planning, control and machine learning.