Noname manuscript No.
(will be inserted by the editor)

Exploring Unstructured Environment with Frontier Trees

R. Korb · A. Schoettl

the date of receipt and acceptance should be inserted later

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 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. It is a memory of past poses during exploration and is utilized to decide between future navigation goals. This approach is compared to a basic but efficient nearest neighbour exploration. The algorithm is tested in simulation with maps similar to appartement ground maps including furniture as obstacles. The paper shows, that frontier trees minimize the travelled path of the mobile robot by tracking open boundaries.**

Keywords Robotics · Exploration · Path Planning · Frontiers · Domestic Environment

Rudolf F. S. Korb
University of Applied Sciences - Munich
Dept. 04 Electrical Engineering and Information Technology, Laboratory for Autonomous Systems in Munich (LASIM), Lothstr. 64, 80335 Munich, Germany
Tel.: +49-89-1265-3415
E-mail: rudolf.korb@hm.edu

Alfred Schoettl
University of Applied Sciences - Munich
Dept. 04 Electrical Engineering and Information Technology, Laboratory for Autonomous Systems in Munich (LASIM), Lothstr. 64, 80335 Munich, Germany
Tel.: +49-89-1265-3415
E-mail: alfred.schoettl@hm.edu

1 Introduction

Mobile robots must be familiar with 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 planning future exploration steps to uncover the environment efficiently, depending on past steps, 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, dynamic environments and has to deal with interieur and crowded areas. As a consequence, when service robots operate in private appartements or public buildings like hospitals or retirement homes, they need to adapt itselves to their surroudings. Therefore it is crucial for a mobile robot to explore areas autonomously and not rely on an environment designed specifically for a service device.
Basic algorithms follow a greedy approach to select future robot positions during the exploration problem. 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 of known and unknown space. For exploration of simple maps, unfurbished rooms and corridors, basic algorithms provide acceptable results, depending on for example sensor constraints. With high range distance sensors and obstacle free areas, robots cover the entire room before leaving and most likely do 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. Furthermore, the number of possible decisions to choose the next goal from is rising with the environment's complexity. As a result, the greedy al-

gorithm's performance will decrease. The robot has to revisit parts of the map where boundaries have been forgotten, increasing the traveled distance.
Consequently the two main influences for travelling distance during exploration is the complexity of an agent's environment and constraints of its range sensors. Mobile autonomous robots in unstructered environment are dealing with both, high complexity and sensor limitations. Many algorithms use maps with reduced difficulty, expecting an empty environment or assume a sensor with infinite distance measurements. It follows, that algorithms covering only one impact decrease in efficiency, when a complex environment and sensor limitations are present.
Greedy algorithms dealing with both influences have one major flaw, which leads to a rise of traveled distance and exploration time. Approaching a boundary already discovered several steps ahead, is an indication that the robot's current exploration path is a cycle. As a result, open boundaries between the starting and endpoint of the cycle will be reached with high travel costs later on during the exploration task. The frontier tree recognizes cycles and hence is able to avoid high costs and react early on forgotten boundaries.

2 Related Work

Many exploration algorithms use the frontier approach introduced by Yamauchi [4]. A frontier is defined as the boudary between known and unknown space in the currently explored map. The algorithm proposed in this paper also utilizes frontiers as atomic elements in the tree data structure and are discussed in detail in section 4.3. Yamauchi's exploration algorithm is a greedy approch and chooses the shortest obstacle free path to minimize the distance between a robot's position and the next frontier.
Koenig et al. [9] show, that greedy algorithms explore ares with acceptable results. For improvements, map segmentation algorithms were introduced [5, 6] to reduce the travel distance of greedy approaches. Wurm et al. [6] use Voronoi Graphs to divide the explored environment in segments. While large emtpy spaces are segmented well, narrow areas, for example corridors, yield a vast amount of partitions. As a result, the disadvantage of greedy algorithms remains. Holz et al. [5] extend the segmentation by adding a step for merging several partitions.
To keep track of open boundaries, Nasir and Elnagar [7] use a data structure called Gap Navigation Tree, introduced by Tovar et al. [3].
Another approach is using information based heuristics [1,2,8]. Visser and Slamet [2] minimize information entropy by selecting frontiers with the largest area. Additionally, they consider travel distance to each frontier in their cost function, to increase the algoritm's efficiency. In [1], Mobarhani et al. also discuss the tradeoff between an information based heuristic 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 maximising the number of frontier cells at a specific direction and minimizing the travel time to the frontier.
Information based approaches are especially good for rescue scenarios.Focus of these algorithms is uncovering large parts of the map quickly. It is possible that these heuristics result in alternating poses between opposite parts of the map. This behaviour leads to an increased travel distance.

3 Outline

Although the difference to between the worst case of greedy algorithms and the worst case optimal travel distance is reasonable small, the performance is reduced with rising map complexity and sensor limitations. A major weakness of an efficient greedy algorithm is illustrated in figure 1. When entering a room with obstacles, it is possible, that the robot runs in a cycle and returns to a previous area before finishing the room. This behaviour results in higher travel cost, because the robt has to revisit the room. Segmentation approaches [5, 6] to further reduce travel cost, highly rely on the geometrical appearence of the map. With an environment containing obstacles, the probability of revisiting a room is increasing. Segmentation in combination with merging minimizes the number of segments, but does not eliminate the problem, where the robot's path is in a cycle. With Gap Navigation Trees Tovar et al. [3] solve the problem of map complexity with arbitrary obstacles, but assume infinite distance measurement und 360° opening angle and therefore do not have sensor constraints. A reduced map size grants this property for real life experiments.
The exploration algorithm proposed in this paper performs well with complex structure and sensor limitations. It offers the ability to react, when the agent runs in a cycle to prevent further travel cost.

4 The Frontier Tree Algorithm

Foundation of this exploration algorithm is a tree data structure with frontiers as nodes. The definition of a

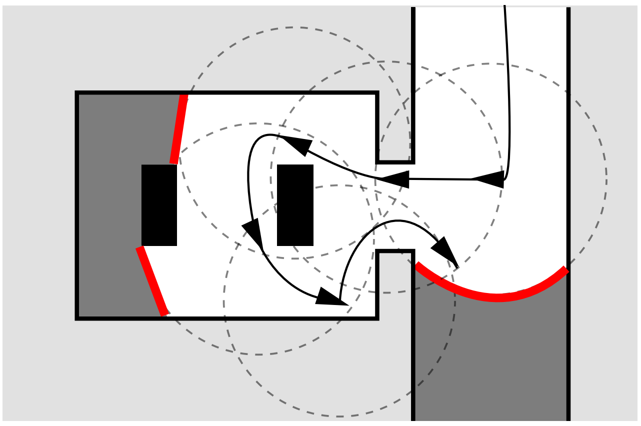


Fig. 1: This illustration shows six robot states during exploration, following a greedy algorithm. The robot enters a room with obstacles and leaves before uncovering the whole area. As a result, the agent has to return at a later exploration step.

frontier is related to Yamauchi's approach [4]. Similar to Gap Navigation Trees [3], the data structure is, inter alia, 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.

4.1 Robot Model

In this paper the robot is implemented 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 x, y as position and θ as orientation. Additionally, the agent provides a distance sensor, represented by $\mathbf{s} = (d, \alpha)^T$. Component d is the sensor's range and α its opening angle. As an assumption, the robot is able to localize itself 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 with values between 0 and 1, where white pixels (1) mark free and black pixels (0) unpassable space. Values in the interval $(0, 1)$ can be interpreted as unknown space. An oocupancy map is the output of a SLAM algorithm. Unpassable 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 (with dependency of $\mathbf{r}$) occupancy map occ_l.

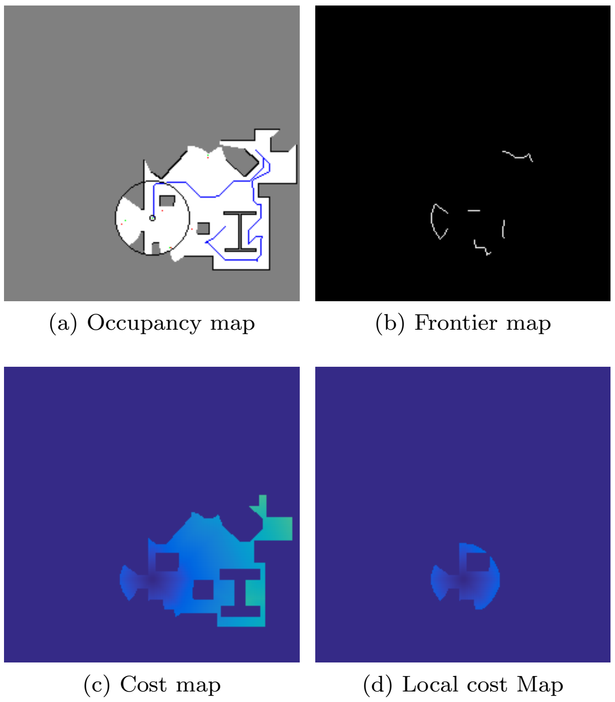


Fig. 2: (a)The Occupancy map displays the knowledge of the robot about its surroundings with white pixels as free, black as occupied and grey as unknown space. (b)Exploration goals for the next steps are displayed in the Frontier Map. (c)The Cost Map combines information about travel cost and reachable points. Pure black is not reachable and travel cost rise with the brighness. (d)The local cost map shows travel cost and reachable points from $\mathbf{r}$ to a distance $d + \sigma$ around the robot.

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

where $\text{dist}(\mathbf{x}, \mathbf{r})$ is the euclidian distance between the considered pixel $\mathbf{x}$ in occ_{local} to the current robot position $\mathbf{r}$.
Both, occ_g and occ_l, as well as the current pose $\mathbf{r}$ are utilized for the calculation of global and local cost maps cost_g and cost_l. These are calculated with Dijstra's Algorithm. Without a return criterion, the result is a map with the $\text{cost}(\mathbf{x})$ to travel from $\mathbf{r}$ to $\mathbf{x}$.
Lastly, occ_g is used to create a frontier map $\mathcal{F}$, containing the possible future goals for exploration. Since occ_g is an inflated representation, every point in $\mathcal{F}$ is a possible navigation goal. Section 4.3 gives a formal description of frontiers.

4.3 Frontiers

Frontiers are extracted from the global occupancy map. A Frontier f^i, with index i as unique id, is the boundary between unknown and free space in occ_g [4]. An elementary part of frontier f^i are frontier cells f^i_j, where superscript j denotes the index of the single cell inside the frontier. A cell is defined by the following two properties:

$$\mathrm{occ}_g(\mathbf{x}) = 1 \tag{2}$$

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

with N_8 as the Moore Neighbourhood of index $\mathbf{x}$ in occupancy map occ_g. Adjacent frontier cells are merged to a single frontier f^i. $\mathbb{F}$ is the set of all frontiers f^i. Every frontier must specify a cell f^i_g that declares a navigation goal to plan a path from $\mathbf{r}$ to f^i. This position is given as

$$f^i_g = \min(\|f^i_j - \bar{f}^i\|_2), \tag{4}$$

The frontier cell with the minimum euclidian distance from f^i_j to the mean value of all frontier cells $\bar{f}^i$ is selected.

4.4 Tree Structure

Root of the tree structure is the initial position $\mathbf{r}_{init}$ of the robot on the map. Each node n has a connection to its parent and contains a list of all children. These links offer the ability to traverse the tree in both directions. The number of a node's descendants is dependent on $\mathbf{r}$ and cost_l. Additionally, every n_i saves f^i_g as data element, representing the navigation goal of the node. Every node can be in one of the three states unmarked, marked or visited. Unmarked nodes are possible goal position for exploration. Marked nodes are a substitution for a delete operation and are not used anymore for the future planning process. A visited node is every node, that has been chosen as exploration goal. In Summary, the frontier tree stores every node thas was a possible goal, all current reachable nodes and contains the complete exploration path. As long as the agent follows frontiers in the same direction, the tree is growing in depth. When the robot has to decide between two or more directions, a subtree is generated and the frontier tree is growing in breadth. For basic operations on the tree, e.g. traverse the data structure, a breadth first search was implemented.

4.5 Tree Operations

The Frontier Tree offers two basic operations to directly manipulate the data structure. An Insert function extends the existing children of a node by attaching an element. Every inserted node is also a leaf and therefor can not be merged between a parent element and its existing descendants.
Instead of a delete function, the Frontier Tree offers an operation for marking nodes. Marked nodes are not considered for further exploration steps and cannot be extended with the insert function, consequently all marked nodes are also leafs.

4.5.1 Frontier Sets

Inserting and marking are straightforward functions. The essential and difficult part is the identification, which frontiers have to be inserted or marked in the tree structure. Frontier trees depend on two sets $\mathbb{F}^t$ and $\mathbb{F}^{t-1}$ to perform the decision procedure. Both sets $\mathbb{F}$ represent the frontiers of two following exploration steps. Initially, $\mathbb{F}^t$ is divided into two disjoint subsets, with

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

$$\mathbb{F}^t_{out} = \overline{\mathbb{F}_{in}} \tag{6}$$

The set of the prior step at time $t-1$ is defined by all the unexplored leaf nodes in the Frontier Tree. Since the tree is in the state before node insertion, the robot's current position f^t_c is also a leaf but already explored. Consequently the f^t_c has to be excluded, resulting in $\mathbb{F}^{t-1}\backslash\{f^t_c\}$.

4.5.2 Binary Relation Matrix and Identification Vector

The frontier set $\mathbb{F}^t_{in}$ can be inserted at the current position. It contains all reachable frontiers from f^t_c. In the following step, the algorithm marks all leafs, which are not considered anymore in the exploration progress. There exist two cases where nodes have to be marked: First, when space is uncovered and as a result, frontiers disappear in the exploration step. Secondly, Frontiers previously discovered are in the set $\mathbb{F}^t_{in}$.
Initially, the distance matrix $\mathbf{D}$ is determinded by a conjunction of $\mathbb{F}^{t-1}\backslash\{f^t_c\}$ and F^t_{out}:

$$\mathbb{F}^{t-1}\backslash\{f^t_c\} \times \mathbb{F}^t_{out} \longrightarrow \mathbb{R} \tag{7}$$

$$\mathbf{D} : (f^{i,t-1}_g, f^{j,t}_g) \longmapsto \|f^{i,t-1}_g, f^{j,t}_g\|_2 \tag{8}$$

The resulting matrix has the dimensions $|\mathbb{F}^{t-1}\backslash\{f^t_c\}| \times |\mathbb{F}^t_{out}|$, containing all distances from frontiers of the set at $t-1$ to the set at time t. $\mathbf{D}$ is the base to construct the binary relation matrix $\mathbf{B}$. A relationship between frontiers is defined as

$$\mathbf{B} = \begin{cases} 1, \text{if } \mathbf{d}_{i,j} = \min(d_{i,*}) \\ 0, \text{else} \end{cases} \tag{9}$$

where $\min(\mathbf{d}_{i,*})$ is the minimum value of the $i-$th row in the distance matrix $\mathbf{D}$. Using $\mathbf{B}$, a $1 \times |F^t_{out}|$ identification vector $\mathbf{s}$ can be calculated by a summation of the rows $\mathbf{b}_{*,j}$.

$$\forall \mathbf{b}_{*,j} : \mathbf{s}_j = \sum_{i=1}^{|\mathbb{F}^{t-1} \setminus \{f^t_c\}|} \mathbf{b}_{i,j} \tag{10}$$

The leafs in the Frontier Tree have to be synchronized with the Frontiers of the current exploration step. Components of $\mathbf{s}$ can take three kind of values to identify, if a node in the Frontier Tree needs to be marked or additionally inserted, to achive a 1-to-1 relationship between the elements of both frontier sets.

$$\nexists \mathbf{d}_{i,j} \in \mathbf{d}_{*,j} : \mathbf{d}_{i,j} = \min(\mathbf{d}_{i,*}) \Rightarrow \mathbf{s}_j = 0 \tag{11}$$

$$\exists! \mathbf{d}_{i,j} \in \mathbf{d}_{*,j} : \mathbf{d}_{i,j} = \min(\mathbf{d}_{i,*}) \Rightarrow \mathbf{s}_j = 1 \tag{12}$$

$$\exists^{>1} \mathbf{d}_{i,j} \in \mathbf{d}_{*,j} : \mathbf{d}_{i,j} = \min(\mathbf{d}_{i,*}) \Rightarrow \mathbf{s}_j > 1 \tag{13}$$

With $\mathbf{s}_j = 0$, $f^{j,t}_g$ does not have a relation to any of the leafs in the Frontier Tree. There exist other frontiers of $\mathbb{F}^t_{\text{out}}$, which have a smaller distence to the elements of $\mathbb{F}^{t-1} \setminus \{f^t_c\}$. It follows, that $f^{j,t}_g$ has to be inserted in the tree to establish a new relation. Since $f^{j,t}_g$ is discovered during the current exploration step, the node is added to the parent element.

When $\mathbf{s}_j$ takes the value 1, $f^{j,t}_g$ is related to exactly one element $f^{i,t-1}_g$. In this best case, nodes have to be neither marked nor inserted and leaf $f^{i,t-1}_g$ has to be considered further in future exploration steps.

If $\mathbf{s}_j > 1$, there exist multiple relations between $f^{j,t}_g$ and the elements of $\mathbb{F}^{t-1} \setminus \{f^t_c\}$. As a consequence, leafs have to be marked because either they 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 $d_{i,j} > min(d_{*,j})$.

After the interpretation of the identification vector, the tree is synchronized with the set of current frontiers, where every element of $\mathbb{F}^{t-1}$ is singularly related to an element of $\mathbb{F}^t$.

4.6 Cycles

Frontier Trees and its operations as defined in section 4.5, can already be used as a greedy exploration algorithm, where the next navigation goal is the frontier with the smallest distance in the reachable area of cost_l. Key feature of the proposed algorithm is the additional use of the tree data structure as topological map, to improve the exploration progress. Initially the Frontier Tree Algorithm follows the greedy strategy until a cycle is found. After detection the robot can adapt the exploration behaviour to the new situation, to eliminate the disadvantage of nearest neighbour approaches. Only one occassion during the marking process has to be considered. The marked nodes, which are now a subset of $\mathbb{F}_{in}$, are important for the detection of cycles. Frontiers, that are uncovered during the exploration step and therefor also marked,. can be neglected.

A cycle is defined by the following difference:

$$\delta = \text{rank}(f^c_g) - \text{rank}(f^i_g) \tag{14}$$

where $\text{rank}(f)$ is the height of the tree structure at the position of frontier f. If $\delta > 1$ between the current position and the marked node, a cycle is detectd.

The higher the value of δ the larger is the detected cycle. Small cycles are usually obstacles like tables or other furniture, while large cycles can be found between different rooms or other global map structures.

After the detection of a loop and before following the next best frontier with minimum distance, the robot has the posibility, to check different goal positions for the next exploration step. Figure 3a shows a tree structure, representing the exploration progress of the robot from figure 1. Black nodes are already marked, white leafs are the possible navigation goals and dotted nodes indicate a loop between two nodes. When the robot chooses the narest frontier for the next step, as greedy algorithms would do, frontier f^2 and f^3 would be missed. Consequently, the agent has to return the whole path after finishing the exploration in the direction of f^4. Since we detected a loop between f^1 and f^4, the Frontier Tree Algorithm can react to this situation. To prevent an increase of the travel distance, the missing frontiers inside the cycle between f^1 and f^4 must be covered. This is done with a bottom up approach. The data structure offers a traversal in both directions. The algorithm walks from the current node's grandparent up to the marked node, that indicated the loop. For every node on the way up, all children are searched for an unmarked node, which is the next exploration goal. When the robot reaches the frontier, it is approaching other nodes in the cycle and covers forgotten frontiers. If there is no such node, because all children are marked or visited, the robot can follow the nearest reachable frontier, because there is no missed frontier inside the loop.

When the robot reaches a first cycle in a map, it is possible, that there exist one or more frontiers, that are already missed before the first loop node as shown in

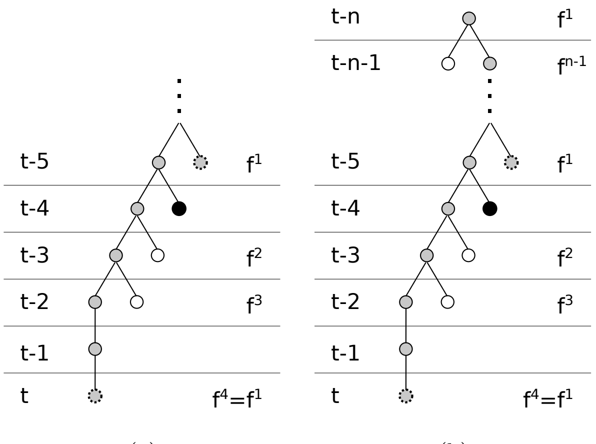


Fig. 3: (a) Frontier Tree with detected cycle including two missed frontiers. (b) Missed frontier in the tree of prior exlporation steps.

Figure 3b with the unmarked node f^{t-n-1}. By following the bottom up approach from the current node to the marked node, only frontiers inside the loop are explored. When there occured open frontiers before the cycle, the robot is going to explore between the loop nodes, but still misses the nodes earlier in the tree. It follows, that traversing only the loop is not sufficient. This issue can be solved by introducing a two step bottom search. The first step is a traversal from the marked node to the frontier tree's root node. When the first step locates an unmarked child, the next exploration step is found. Otherwise, the second step of the bottom up search starts to look for unmarked children. With a node from the first searching step, the Frontier Tree Algorithm will first explore open nodes in the direction of the root node and will afterwards continue with the coverage of frontiers inside the loop.

5 Results

For performance evaluation, the Frontier Tree Algorithm is executed in a simulated environment. The area to be explored has the dimensions $240px \times 240px$. The robot's footprint has a radius of $2px$ and a sensor with $d = 30px$ and $\alpha = 180°$, to add visibility constraints. The maps for testing represent appartements including furniture as arbitrary complex objects. Many algorithms are tested in large areas or empty environment,therfore a suitable dataset for evaluating appartements does not exist. Consequently, the algorithm is tested on self-created maps. where four are shown in Figure 5. The measured parameter is the complete travel distance of the mobile robot from start to the fully explored map. The value is then compared to a basic nearest neighbour algorithm. This greedy approach chooses the next exploration step by minimizing the travel cost on the inflated map cost_g, calculated with Dijkstra's algorithm. Neares Neighbour uses Yamauchi's definition of frontiers [4], too. Additionally, the travel distance of each exploration step, the average travel distance/step and the complete number of steps to fully explore the map, are compared.

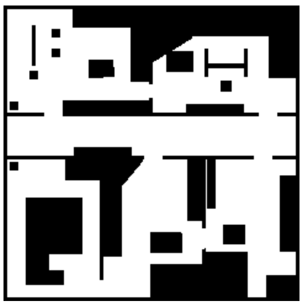
	NN TD	**FTA TD**	**CMP % TD**
A	2136.6	2131.6	**-0.23**
B	2776.2	2648.8	**-4.59**
C	2368.7	2182.4	**-7.86**
D	2146.9	1819.7	**-15.24**

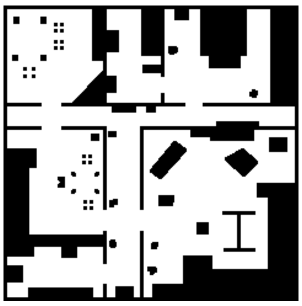
Table 1: Comparison of travel distance between Nearest Neighbour(NN) and the Frontier Tree Algorithm(FTA) on maps A-D

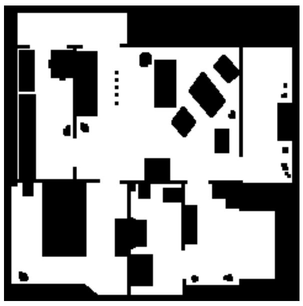
	NN avg	**FTA avg**	**CMP % avg**	**NN steps**	**FTA steps**	**CMP % steps**
A	32.37	32.79	1.3	66	65	**-1.15**
B	35.14	34.85	**-0.82**	79	76	**-3.8**
C	38.20	36.99	**-3.18**	62	59	**-4.84**
D	33.54	31.92	**-4.83**	64	57	**-10.9**

Table 2: Comparison of the average travel distance for each step and exploration steps between Nearest Neighbour(NN) and the Frontier Tree Algorithm(FTA) on maps A-D

The results of the comparison between the nearest neighbour approach (NN) and the Frontier Tree Algorithm (FTA) is presented in table 1. On all maps, the complete travel distance is reduced when using the FTA. While on map A a reduction of 0.23% is nearly the same result of the NN, the efficiency of exploring map D can be increased by 15.24%. Figure 4 shows the comparison lined up for each map in a bar graph.
As shown in table 2,with FTA, the number of steps to explore a map completly is less compared to NN for all maps by up to 10.9%. Except for map A, the average travel distance for each step is reduced by up to 4.83% with an average value of 31.92, approaching the the sensor range d. A visualization of the single steps and the average step distance for NN and FTA is shown in figure 6a-d.


(a) Appartement A


(b) Appartement B


(c) Appartement C

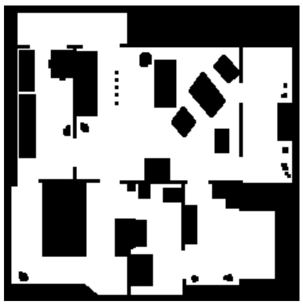
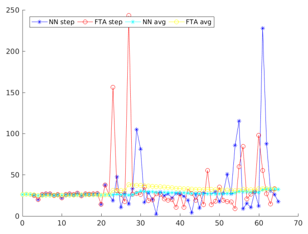
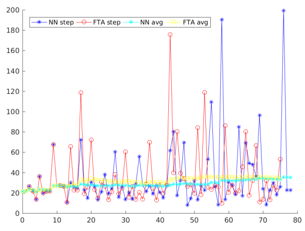

(d) Appartement D

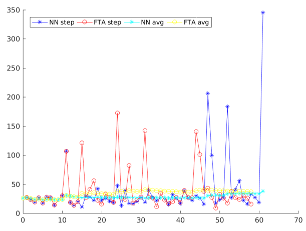
Fig. 5: Maps used for evaluation of the Frontier Tree Algorithm.



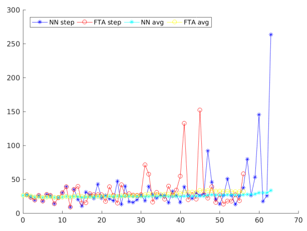
(a) Appartement A



(b) Appartement B



(c) Appartement C



(d) Appartement D

Fig. 6: Comparing single step travel distance and average travel distance for each exploration step between NN and FTA.

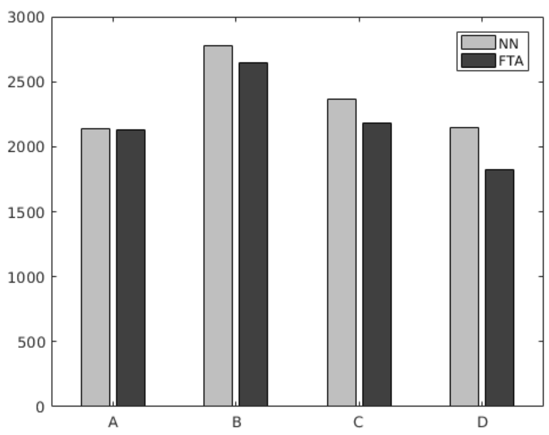


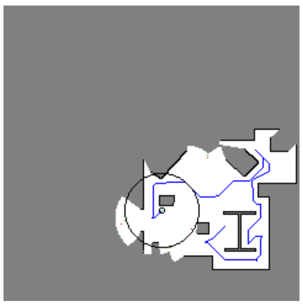
Fig. 4: Complete travel distance on maps A - D of Nearest Neighbour and the Frontier Tree Algorithm.

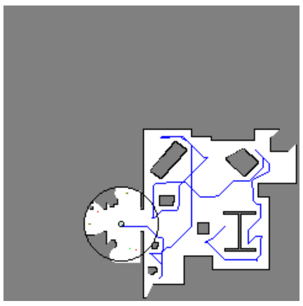
6 Discussion

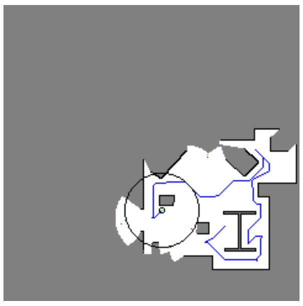
This section interprets the resulting travel distances presented in table 1. Additionally, two cases of exploration decisions by using the Frontier Tree data structure are dicussed in detail, to demonstrate the advantage compared to nearest neighbour algorithms.

6.1 Travel Distance and Single Step Travel Distance

When a map is partly finished (a room is explored completely or the robot is in a dead end), single step distance to the next frontier is rising, leading to values above average during the task. The graphs of Appartement B in figure 6b consist of the most single rooms without direct transitions to other rooms and dead ends inside romms. This leads to an increased average and single step travel distance for both algorithms. All Graphs of figure 6 show large distances for single steps towards the end of exploring by the nearest neighbour approach. This is an indication for missed frontiers during the exploration process. In comparison, the Frontier Tree Algorithm does not have high peaks before finishing the exploration. This behaviour shows, that missed frontiers are found early enough to prevent a collection of forgotten frontiers at the end of exploration. As a result, the average travel distance and the single step distance is rising during the task. That implies that travel distances of NN and FTA align to each other, when a nearest neighbour approach is not missing frontiers. As the result of map A in table 1 shows, travel distance also levels up when the cost of returning to missed frontiers is the same as collecting the missing goals at the end of exploration. Comparing the tree structure of map A (with approximatly the same travel


(a) App. B FTA exp. step 14


(b) App. B FTA exp. step 21


(c) App. B NN exp. step 14

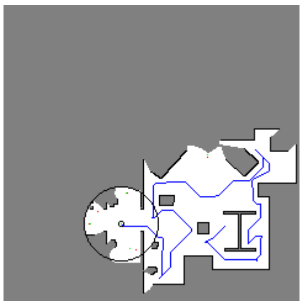
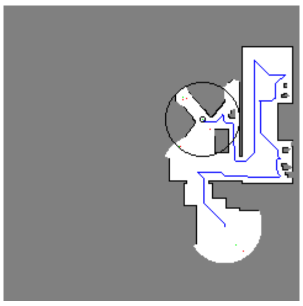
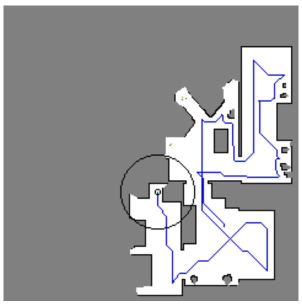
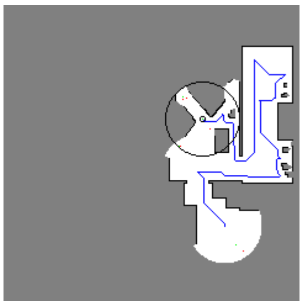

(d) App. B NN exp. step 18

Fig. 7: 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 9a shows related Frontier Tree


(a) App. C FTA exp. step 15


(b) App. C FTA exp. step 24


(c) App. C NN exp. step 15

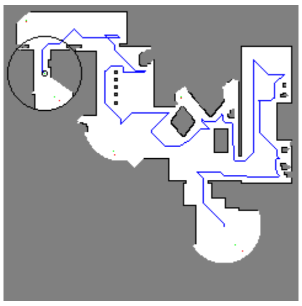

(d) App. C NN exp. step 30

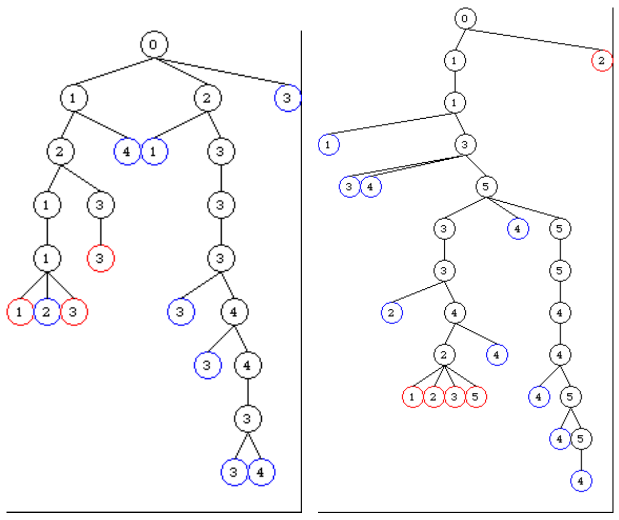
Fig. 8: Behaviour when detecting a global cycle. There exist frontiers between the root node and the marked node. Figure 9b shows related Frontier Tree

distance) and map D (with a reduction by 15.24% using FTA) implies, that trees growing in breadth lead to an increase in efficiency. The tree is divided under the following circumstances: When partially exploring a map, a dead end or finding a cycle. Maps B and C yield stable results. It is hardly possible for the robot, to explore the environment by following a frontier for many steps resulting in a deep tree structure, compared to map A where almost all rooms can be explored in a single run.

6.2 Two Step Bottom Up Search - Case Examples

Figure 7 illustrates a local cycle handling on map B of FTA compared to NN. The related Frontier Trees (FT) are shown in Figure 9. Node 1 with rank 3 was marked, leading to a cycle identification. Since no goal is found when searching from the marked node to the root element, the cycle can be classified as local, resulting in travel steps backwards closing open frontiers. Figure 7c and 7d show, that following the NN approach misses the frontier covered by FTA. Compared to figure 6b FTA explores the respective frontier, approving a step travel distance of 60px and an average value of approximately 35px at exploration step 14. The distance of 60px has to be doubled, taking into account, that the robot has to return to the prior position. This specific frontier is explored at step 77 by the NN algorithm resulting in a step distance > 200. It follows that the decision to explore the missed frontier early, leads to a saving of about 40% for this particular step.

Figure 8a and 8b show the detection of a small loop when the robot moves around an obstacle. The Frontier Tree of map C (fig 9b) shows, that there is no open frontier between the current node 2 rank 9 and marked node 1 rank 4. However, the two step search will find node 2 rank 2 as exploration goal during the first iteration when searching from the marked node towards the tree root, whereby a global cycle is identified.In contrast to local cycles, these loops often start the exploration of global structures like rooms. The NN algorithm (fig. 8c and 8d) 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 right corner which is chosen as last step with a single step distance 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 will leave the top right frontier, but recognizes it as a local cycle to explore it early, saving further travel cost.



(a) App. B FT step 15 (b) App. C FT step 16

Fig. 9: Frontier Trees (FT) constructed during exploration showing a) a local b) a global cycle detection.

7 Conclusion

The nearest neighbour algorithm, used in the experiments, produces acceptable result for exploring an unknown unstructered environment. This paper discoveres cycles of the robot's path during the exploration task as main disadvantage of greedy approaches, resulting in missing frontiers and high travel cost at the end of exploring the area.
This problem is solved 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 for complete travel distance, travel distance for a single exploration step and the average travel distance. Comparing the created frontier trees show, that Frontier Trees work especially well when the data structure grows in breadth. It follows that the performance of the Frontier Tree algorithm can decrease on maps, where frontiers can be followed throug many steps without running in a dead end or cycle, for example maze-like environments. To approach this weakness, the decision making process should be reconsidered, to add a contraint for the maximum loop dept. In contrast, the algorithm yields stable results in unstructered environment, like furbished appartements, offering cycles and partly finished areas to initiate the goal selection on the Frontier Tree. Next step of this research is an implementation and verification of the algorithm in a real world situation.

References

1. Amir Mobarhani, Shaghayegh Nazari, Amir H. Tamjidi, Hamid D. Taghirad: Histogram Based Frontier Exploration (2011)
2. Arnoud Visser, B.A. Slamet: Balancing the information gain against the movement cost for multi-robot frontier exploration. In: European Robotics Symposium 2008 by H. Bruyninckx & L. Peuil & M. Kulich. Prague (2008)
3. Benjamin Tovar, Steven M. LaValle, Rafael Murrieta: Optimal Navigation and Object Finding without Geometric Maps or Localization. In: Robotics and Automation, 2003. Proceedings. ICRA '03 (2003)
4. Brian Yamauchi: A frontier-based approach for autonomous exploration. In: CIRA'97., Proceedings (2007). DOI 10.1109/CIRA.1997.613851
5. Dirk Holz, Nicola Basilico, Francesco Amigoni, Sven Behnke: Evaluating the Efficiency of Frontier-based Exploration Strategies (2010)
6. Kai M. Wurm, Cyrill Stachniss, Wolfram Burgard: Coordinated Multi-Robot Exploration using a Segmentation of the Environment (2008)
7. Reem Nasir, Ashraf Elnagar: Gap Navigation Trees for Discovering Unknown Environments. Intelligent Control and Automation **2015**(6), 229–240 (2015)
8. Robert Grabowski, Pradeep Khosla, Howie Choset: Autonomous Exploration via Regions of Interest. In: Proceedings of the 2003 IEEE Intl. Conference on Intelligent Robots and Systems. Las Vegas (2003)
9. Sven Koenig, Craig Tovey, William Halliburton: Greedy mapping of terrain. In: Robotics and Automation, 2001. Proceedings 2001 ICRA (2001). DOI 10.1109/ROBOT.2001.933175