Research Article

INTERNATIONAL JOURNAL OF ADVANCED ROBOTIC SYSTEMS

International Journal of Advanced Robotic Systems
July-August 2019: 1–14
© The Author(s) 2019
DOI: 10.1177/1729881419865427
journals.sagepub.com/home/arx
SAGE

A sampling-based multi-tree fusion algorithm for frontier detection

Wenchuan Qiao[1,2], Zheng Fang[1] and Bailu Si[2,3]

Abstract

Autonomous exploration is a key step toward real robotic autonomy. Among various approaches for autonomous exploration, frontier-based methods are most commonly used. One efficient method of frontier detection exploits the idea of the rapidly-exploring random tree and uses tree edges to search for frontiers. However, this method usually needs to consume a lot of memory resources and searches for frontiers slowly in the environments where random trees are not easy to grow (unfavorable environments). In this article, a sampling-based multi-tree fusion algorithm for frontier detection is proposed. Firstly, the random tree's growing and storage rules are changed so that the disadvantage of its slow growing under unfavorable environments is overcome. Secondly, a block structure is proposed to judge whether tree nodes in a block play a decisive role in frontier detection, so that a large number of redundant tree nodes can be deleted. Finally, two random trees with different growing rules are fused to speed up frontier detection. Experimental results in both simulated and real environments demonstrate that our algorithm for frontier detection consumes fewer memory resources and shows better performances in unfavorable environments.

Keywords

Exploration, frontier-based, rapidly-exploring random tree

Date received: 27 February 2019; accepted: 30 June 2019

Topic: Mobile Robots and Multi-Robot Systems
Topic Editor: Nak-Young Chong
Associate Editor: Changjoo Nam

Introduction

With the continuous development of robotic technologies, robots, especially autonomous mobile robots, have been integrated into more and more fields of human society, which puts forward higher requirements for robotic autonomy. Autonomous exploration is a key step toward real robotic autonomy. It could map the environment independently and autonomously, which provides the basis for further operations of robots. Autonomous exploration is a key ability for many robots, such as inspection robot,[1] rescue robot,[2] survey robot,[3] and planet exploration robot.[4] Among various approaches for exploration, frontier-based methods[5] are most commonly used. Frontier means the boundary between the unknown area and the free area in the current map. By continuously navigating to frontiers, the robot can obtain more unknown environmental information with its onboard sensors. However, most frontier detection methods process the entire map data.[6] When the environment to be explored is large, the efficiency of those methods will reduce significantly. Those methods would also become inefficient in 3-D environments. To solve this problem, some

[1]Faculty of Robot Science and Engineering, Northeastern University, Shenyang 110819, China
[2]State Key Laboratory of Robotics, Shenyang Institute of Automation, Chinese Academy of Sciences, Shenyang 110016, China
[3]Institutes for Robotics and Intelligent Manufacturing, Chinese Academy of Sciences, Shenyang 110169, China

Corresponding author:
Zheng Fang, Faculty of Robot Science and Engineering, Northeastern University, Shenyang 110819, China.
Email: fangzheng81@gmail.com

researchers proposed exploration methods based on the rapidly-exploring random tree (RRT).[7] The RRT refers to sampling randomly in a certain area and forming tree edges and nodes through sampled points and their nearest valid tree nodes. By doing this repeatedly, it can form an integrated structure which is similar to a tree. At present, there are two ways of exploration based on the RRT.

- Using the modified RRT to generate paths for exploration,[8,9] so that robots can complete the task of mapping an environment through moving along these paths. However, due to the randomness of the tree's growing process, it will cause robots to explore the same area many times and lose the advantage of maximizing the information gain through finding frontiers.
- Using the modified RRT to search for frontiers.[10] The idea is that if a tree edge falls on both the unknown area and the free area at the same time, the unknown point adjacent to the free area on this edge is a frontier point. However, as exploration proceeds, more and more tree nodes and edges need to be stored. A large number of tree nodes and edges and the tree's unrestrained growing will increase the memory burden and reduce the efficiency and practicability of the algorithm. In addition, due to the tree's slow growing in unfavorable environments which often exist in reality (such as floors containing many rooms and long corridors), the robot may not find frontiers for a long time.

In this article, a sampling-based multi-tree fusion algorithm (SMF) for frontier detection is proposed which also uses the idea of the RRT.[7] Firstly, we only maintain tree nodes but abandon tree edges after using them for detecting frontiers. At the same time, we allow the tree to grow across obstacles, which overcomes the disadvantage of its slow growing in unfavorable environments. Secondly, we propose a block structure to judge whether tree nodes in a block play a decisive role in frontier detection, so as to delete a large number of redundant tree nodes generated during exploration. This reduces memory consumption greatly. Finally, we fuse a global exploration tree (global tree) which keeps growing throughout exploration with a local exploration tree (local tree) which regrows every time it finds a frontier point. On one hand, we can make the global tree detect frontiers faster because useful local tree nodes are fused into the global tree. On the other hand, the global tree ensures that frontiers in the map can all be found.[10] The experiments show that our SMF consumes fewer memory resources and shows better performances in unfavorable environments. Figure 1 shows the contrast between the proposed SMF and the algorithm proposed by Umari and Mukhopadhyay[10] (hereafter called rapidly-exploring frontier detector (RFD)).

In summary, the main contributions of this article are:

- A novel frontier detection method based on the RRT is proposed, which can work well in unfavorable environments where classical random trees are not easy to grow.
- A block structure is proposed to identify the area where important tree nodes are located and to delete other redundant tree nodes, which can reduce memory consumption greatly.
- A global exploration tree (global tree) and a local exploration tree (local tree) are fused to speed up frontier detection.
- A lot of comparative experiments are carried out in both simulated and real environments to verify the effectiveness of our algorithm.

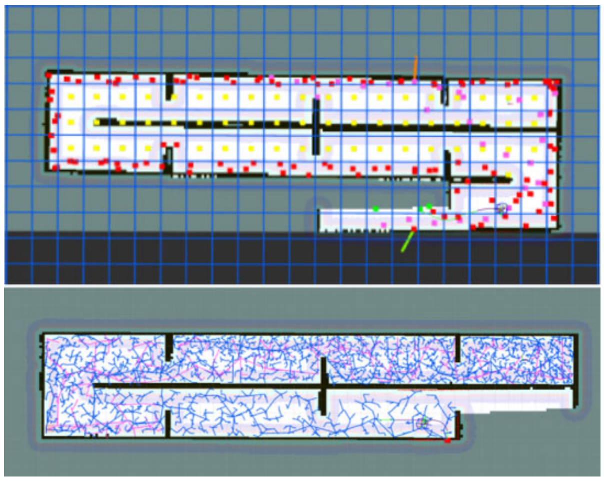
Figure 1. The upper part is a diagram of frontier detection using SMF. The red and purple points respectively represent global tree nodes and local tree nodes. The blue squares represent blocks. If there is a yellow point at the center of a block, it means the redundant global tree nodes in this block have been deleted. The green and red edges respectively represent global tree edges and local tree edges used to detect frontiers. The green points represent currently found frontier points. The lower part is a diagram of frontier detection using RFD. The blue and purple tree structure respectively represent the global tree and the local tree. SMF: sampling-based multi-tree fusion.

The rest of this article is organized as follows. In the second section, we discuss related works. The third section describes the proposed SMF. We validate performances of our SMF in both simulated and real environments in the fourth section and fifth section concludes the article.

Related works

Exploration means finding targets or paths so that robots can perceive unknown environmental information and map the whole environment in the constraint of time, path cost, energy consumption, and so on. Connolly[11] proposed the Next Best View (NBV) problem in 1985. The location where the most unknown scene information can be gained by a sensor was chosen as the NBV. González-Baños and Latombe[12] introduced the safe region concept to select the next robot position in order to maximize the expected information gain under the constraint that the local model at this new position must have a minimal overlap with the current global map. Vasquez-Gomez et al.[13] came up with more complex and sophisticated solutions of the NBV problem in 2014 by considering all the constraints of a reconstruction process.

Yamauchi[5] first proposed the concept of the frontier in 1997, which can be seen as an adaptation of NBV.[14] The frontier means the boundary between the unknown area and the free area in the current map. Edge detection in image processing was used to search for frontiers and the center of a frontier which was nearest to the robot would be regarded as the next navigation target. After that, many frontier-based exploration approaches have been used in single robot exploration[15–17] and multi-robot cooperative exploration.[18–22] However, edge detection used to search for frontiers will become inefficient when the map it processes is large. Keidar and Kaminka[6] proposed two methods to alleviate this problem: wavefront frontier detector (WFD) and fast frontier detector (FFD). WFD only detects frontiers in the known area of the map, thus avoiding processing the whole map. However, as exploration goes on, the known area that needs to be processed will become larger and larger. FFD processes sensor data every time it is obtained to search for frontiers. However, only when the robot obtains unknown environmental information, can new frontiers be extracted effectively from sensor data. Thus, it carries out many unnecessary calculations.

Frontier-based exploration approaches have also been used for 3-D environments.[14,23–25] Adler et al.[23] created a dense watertight 3-D model of the real-world environment and the gaps which had remained throughout the mapping process were chosen as navigation targets. However, this method must establish a dense 3-D environmental model, which would occupy much resources. Heng et al.[24] used a stereo camera to establish a 3-D octomap of the environment and extracted frontiers from a 2-D slice of the 3-D octomap. But this method can only find frontiers of the fixed height in the 3-D octomap. Cieslewski et al.[14] extracted frontiers by finding free voxel grids neighboring unknown voxel grids in the 3-D octomap of the environment and the goal frontier was selected in a way that minimizes the change in velocity necessary to reach it. But this method need to process the whole 3-D octomap, which is computationally expensive especially when the 3-D octomap is large.

In contrast to the above frontier-based approaches, another way of exploration is to generate some candidate viewpoints or movement paths and then use the evaluation function to select the best one for perception. Sampling-based information gathering approaches are proposed based on this idea. Among them, methods based on the RRT[7] are popular. RRT is biased toward unexplored areas and can quickly cover a large subset of the configuration space, which makes it real-time capable even in 3-D environments.[26] Oriolo et al.[8] proposed the sensor-based random tree (SRT) exploration method based on RRT in 2004, which generated motion paths randomly in the safe area around the robot until the robot finally explored all areas. However, this method would generate a lot of unnecessary robot movements, which would cause the revisiting problem. Thus, methods aiming at improving the efficiency of SRT are proposed.[27–29] Then, in 2017, Kim et al.[30] proposed a multi-robot cooperative exploration strategy based on SRT. Bircher et al.[9] used RRT to generate the most informative robot motion paths. They grew a certain number of tree branches and selected the first edge of the tree branch with the most unknown environmental information as the robot motion path. By doing this repeatedly, the robot can finally explore the whole environment. Due to the good performance of RRT in 3-D environments, the authors applied this method to the autonomous exploration in 3-D environments by using a quad-rotor unmanned aerial vehicle. Then, methods which also learn from the growing mode of RRT to generate robot motion paths are proposed.[31–33] However, due to the randomness of RRT's growing process, these methods lose the advantage of maximizing the information gain through finding frontiers.

Some researchers[34–37] try to combine frontier-based methods with sampling-based methods to achieve better performances. Meng et al.[37] sampled viewpoints around frontiers and used a Fixed Start Open Travelling Salesman Problem (FSOTSP) solver to generate the optimal path passing through these viewpoints. Senarathne and Wang[36] extracted surface frontiers from the 3-D map and then sampled viewpoints around these surface frontiers. These methods can reduce the randomness of sampling by firstly extracting frontiers and then sampling around them. However, they all need an efficient algorithm for frontier detection especially in 3-D environments. In 2017, Umari and Mukhopadhyay[10] proposed the idea of detecting frontiers by using RRT. However, this method would occupy a lot of memory resources and the frontier detecting speed would be slow in unfavorable environments.

The algorithm for frontier detection proposed in this article is also based on the idea of RRT, but we fully realize the difference between functions of RRT used in searching for frontiers and in generating feasible robot motion paths. Thus we change the growing and storage rules of RRT. We do not save tree edges after using them for detecting frontiers. At the same time, when newly grown nodes and edges fall in the known area, even if they cross obstacles, we still regard these tree nodes as valid and retain them. Because the growing process of modified RRT does not need to avoid obstacles, frontier detection is sped up in unfavorable environments.

Sampling-based multi-tree fusion algorithm

Frontier detection using the modified RRT

As exploration goes on, more and more environmental information is obtained through on-board sensors and the

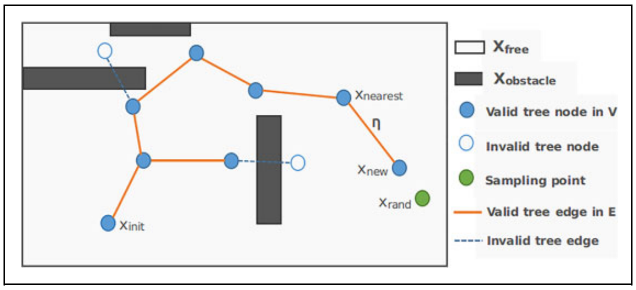

Figure 2. The diagram of the growing process of the classical RRT in path planning. RRT: rapidly-exploring random tree.

environmental map *Map* would be constantly updated. The map model used in this article is the occupancy grid map. *Map* can be divided into the known area X_{known} and the unknown area $X_{unknown}$. Meanwhile, X_{known} can be divided into the obstacle area $X_{obstacle}$ and the free area X_{free}. The frontier refers to the boundary between $X_{unknown}$ and X_{free} in *Map*.

In path planning, the growing process of the classical RRT begins with a starting point (root node) x_{init}. First, x_{init} is added to the valid tree node set V. Then, in each iteration, a point x_{rand} is sampled randomly from a given range and the point $x_{nearest}$ in the valid tree node set V is found which is nearest to x_{rand}. If the distance between $x_{nearest}$ and x_{rand} is longer than the predefined step size η, it will generate a new point x_{new} in the connecting line between $x_{nearest}$ and x_{rand} so that the distance between $x_{nearest}$ and x_{new} is fixed to η. Otherwise, we regard x_{rand} as x_{new}. Then, we grow a tree edge from $x_{nearest}$ to x_{new}. If the tree edge falls on X_{free}, x_{new} is regarded as a valid tree node and is added to V. This tree edge is also added to the valid tree edge set E. Otherwise, x_{new} and the tree edge become invalid and we do not save their information. By repeatedly doing the above operation, it will constantly form an integrated structure similar to a tree, as shown in Figure 2.

In autonomous exploration, the idea of using RRT to search for frontiers is that if the tree edge falls on $X_{unknown}$ and X_{free} at the same time, the unknown point adjacent to X_{free} on this edge is a frontier point, as shown in Figure 3. You can see that there is no frontier point on the edge if $X_{obstacle}$ is adjacent to $X_{unknown}$. Here, the search tree's growing process is the same as the classical RRT's but invalid tree nodes and edges mean that they will be deleted after being used to search for frontiers.

However, we realize that the RRT in autonomous exploration is not used to generate feasible robot motion paths, it is only used to search for frontiers. Therefore, we change the tree's growing and storage rules, as shown in Figure 4. We abandon all tree edges after using them to detect frontiers but only preserve tree nodes. In addition, because the search tree's growing doesn't need to avoid obstacles, x_{new} is still regarded as valid and is added to V if its edge falls in X_{known} but crosses $X_{obstacle}$. By doing this, we speed up frontier detection in unfavorable environments.

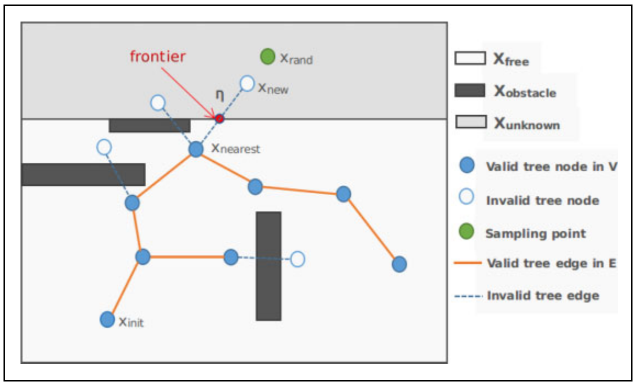

Figure 3. The diagram of the frontier detecting process using the classical RRT. RRT: rapidly-exploring random tree.

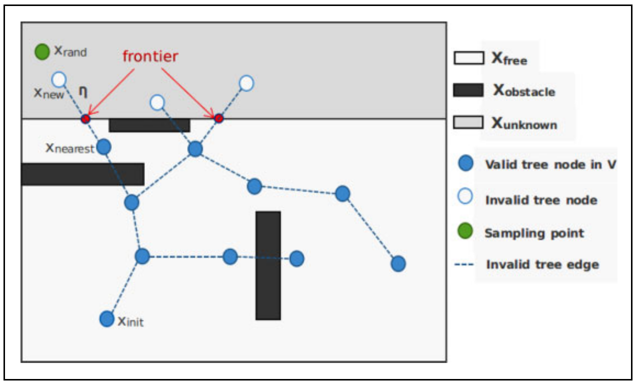

Figure 4. The diagram of the frontier detecting process using our modified RRT. RRT: rapidly-exploring random tree.

Block structure

Although we speed up the growing process of the RRT by changing the tree's growing and storage rules, the number of tree nodes to be stored is still not effectively controlled. As time goes on, it will increase unbounded and the tree's growing will become slower and slower. Thus, exploration cannot be done well especially in large-scale and 3-D environments.

We can find that the frontier detecting process using the RRT is similar to the propagation of water waves. This is because the random tree grows from the initial root node x_{init} and then expands to the surrounding unexplored area to search for frontiers. In order to explore areas where undiscovered frontiers may exist (unexplored areas), the external tree nodes surrounding other tree nodes play an important role. This is because one of them tends to be the $x_{nearest}$ of the x_{rand} which is located in the unexplored areas and can generate x_{new} outward, as shown in Figure 5. On the contrary, internal tree nodes surrounded by external tree nodes do not have much effect on expanding the tree outward and thus become redundant. In Figure 5, the outermost blue nodes with curves of outward arrows represent the external tree nodes which can be chosen as $x_{nearest}$ to expand the tree in the approximate direction of the arrow. The yellow

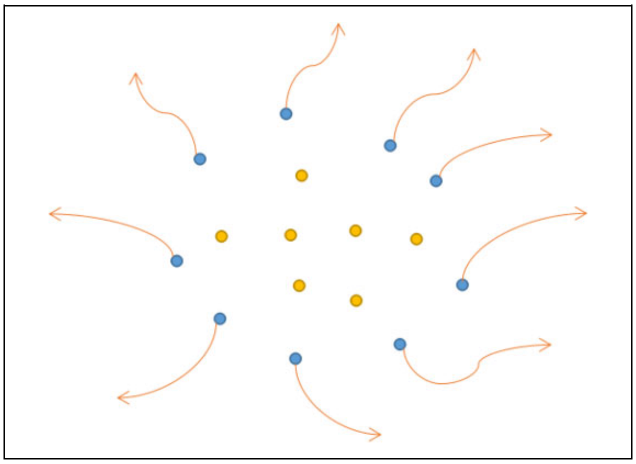
Figure 5. Only blue external tree nodes are needed to search for frontiers and yellow internal tree nodes become redundant.

points represent the redundant internal tree nodes which can be deleted to save memory resources.

In order to identify external and internal tree nodes and delete redundant internal tree nodes, we propose a block structure which divides the map into square blocks of the same size, as shown in Figure 6. Valid tree nodes will fall in these blocks. If a block contains sufficient number ($vertices_{min}$) of tree nodes which can be used to grow new tree nodes efficiently, we call this block $Block_{mature}$. If there is a block which is located in X_{known} and each of the four blocks adjacent to this block is $Block_{mature}$, we categorize it as $Block_{internal}$, as indicated by the A block in Figure 6(a) (here we assume $vertices_{min} = 2$). In addition, if there is a block which is located in X_{known} and each of the four blocks adjacent to it is either $Block_{mature}$ or $Block_{internal}$, this block also is categorized as $Block_{internal}$, as indicated by the B block in Figure 6(b). By contrast, if a block is not $Block_{internal}$, it is categorized as $Block_{external}$. Thus, we can classify all blocks into two categories: $Block_{internal}$ and $Block_{external}$. The tree nodes in $Block_{internal}$ are redundant internal tree nodes which can be deleted and the tree nodes in $Block_{external}$ are important external tree nodes for frontier detection. Thus, we will delete all tree nodes in $Block_{internal}$ to save memory resources. In addition, because $Block_{internal}$ is located in X_{known}, which means there are no frontiers in it, we do not sample x_{rand} and generate x_{new} in it to reduce unnecessary sampling, growing, and detection. This will speed up the tree's growing process and the frontier detecting process.

As mentioned above, we delete the tree nodes in $Block_{internal}$ to reduce memory consumption. But we also need to control the maximum number of tree nodes in $Block_{external}$ so that the number of tree nodes which are needed to be stored in whole exploration can be controlled within a certain range. We can see that if the number of tree nodes in $Block_{external}$ reaches $vertices_{max}$ ($vertices_{max} > vertices_{min}$), only partial tree nodes in it are needed to grow new tree nodes and we will delete $number_{delete}$ tree nodes in it, as shown in Figure 7. In addition, because valid tree

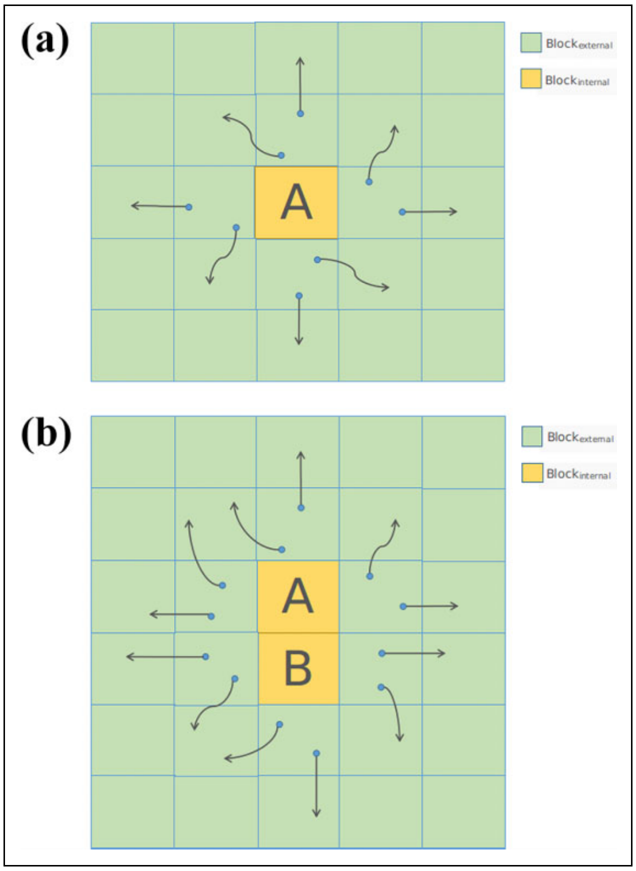

Figure 6. Blocks are categorized into two categories: $Block_{internal}$ and $Block_{external}$. (a) The first case in which $Block_{external}$ is converted to $Block_{internal}$ like the A block. (b) The second case in which $Block_{external}$ is converted to $Block_{internal}$ like the B block.

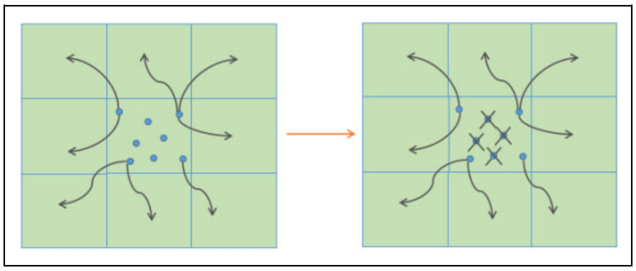
Figure 7. When the number of tree nodes in $Block_{external}$ reaches $vertices_{max}$, partial tree nodes can be deleted.

nodes all exist in X_{known}, when the number of valid tree nodes in $Block_{external}$ reaches $vertices_{max}$, this block is likely to be located in X_{known}, which means that there are no frontiers in it. Therefore, if the number of tree nodes in $Block_{external}$ reaches $vertices_{max}$, we will judge whether $Block_{external}$ is located in X_{known} at the same time. If $Block_{external}$ is located in X_{known}, we will not only delete $number_{delete}$ tree nodes in it but also do not sample x_{rand} and grow x_{new} in it. Otherwise, only $number_{delete}$ tree nodes in it is deleted but we still can sample x_{rand} and grow x_{new} in it. If the number of tree nodes in it reaches $vertices_{max}$ again, we will continue to handle it according to the above rules.

In this article, we divide the map area into $number_{block}$ square blocks whose side lengths are all $resolution_{block}$. If the map to be explored is approximated as a rectangle whose length is x_{map} and width is y_{map}, $number_{block}$ can be calculated by formula (1). The map model used in this article is the occupancy grid map. Each block contains $number_{cell}$ grid cells which can be calculated by formula (2). Thus, whether a block is located in X_{known} or not can be judged by current map information about grid cells' state (free, unknown, or occupied). When exploration is accomplished, $number_{vertices}$ will converge to a fixed proportion of the perimeter L of the known environmental map (containing no unknown area), which can be calculated by formula (3)

$$number_{block} = \lceil x_{map}/resolution_{block} \rceil \times \lceil y_{map}/resolution_{block} \rceil \tag{1}$$

$$number_{cell} = \lceil resolution_{block}/resolution_{map} \rceil^2 \tag{2}$$

$$(vertices_{max} - number_{delete}) \times \lceil L \div resolution_{block} \rceil \leqslant number_{vertices} \leqslant vertices_{max} \times \lceil L \div resolution_{block} \rceil \tag{3}$$

Multi-tree fusion

In RFD algorithm,[10] the global tree and the local tree are growing independently. The global tree keeps growing throughout the whole exploration. The local tree removes all its tree nodes and edges every time it finds a frontier point and then regrows from the current robot position. As we mentioned before, when the tree grows x_{new}, it needs to search for $x_{nearest}$ in the valid tree node set V. When the number of valid tree nodes is large, $x_{nearest}$ would be found slowly and the tree would grow slowly, which could slow down frontier detection. After long-time growing, especially in large-scale and 3-D environments, the global tree grows slower and slower because the number of its tree nodes becomes larger and larger, but it can make sure that all frontiers in the map can be found.[10] By contrast, the number of local tree nodes keeps small because the local tree regrows every time it finds a frontier point, and thus it grows fast. In addition, because the local tree regrows from the current robot position where the robot may obtain new environmental information, it could search for frontiers which are near to the robot or just created by new obtained environmental information faster than the global tree does. However, the authors did not bring the advantage of the local tree into the global tree. In this article, the growing and storage rules of both global and local trees are changed according to the "Frontier detection using the modified RRT" subsection of this section. The redundant global tree nodes are deleted according to "Block structure" subsection of this section. Because the number of local tree nodes is small, we do not delete local tree nodes until it finds a frontier point. We fuse useful local

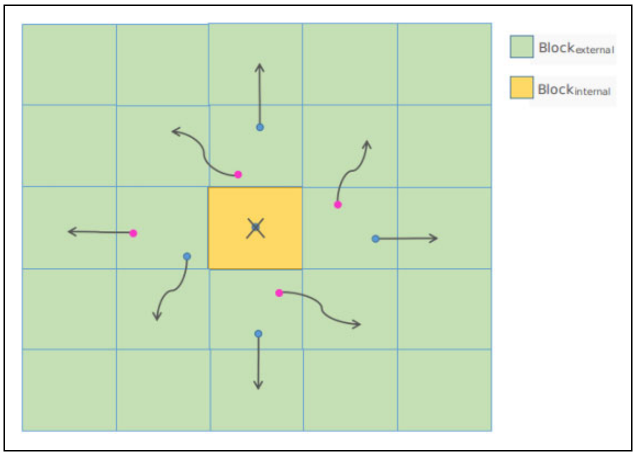

Figure 8. The diagram of fusing purple local tree nodes in *Block*$_{external}$ into the global tree.

tree node x_{local} which is located in $Block_{external}$ into the valid tree node set V of the global tree, as shown in Figure 8. This can speed up the useful outgrowing of the global tree for frontier detection. In addition, because fused local tree nodes have become a part of global tree nodes and the global tree obeys the block rules, some blocks surrounded by them could turn into $Block_{internal}$ and redundant global tree nodes in $Block_{internal}$ can be deleted, as shown in Figure 8.

Implementation details

Global tree algorithm. The global tree algorithm proposed in this article is shown in algorithm 1. The global tree keeps growing throughout the whole exploration and searches for frontiers using the modified RRT. At the same time, the block structure is used to delete redundant global tree nodes and local tree nodes are fused into the global tree to speed up the useful outgrowing of the global tree for frontier detection.

In the global tree algorithm, we first need to initialize blocks' flags and attribute values using the function $BlockInitial(x_{map}, y_{map}, resolution_{block})$. The $DeleteFlag_x$ represents that whether the block where the tree node x is located is $Block_{internal}$. The $Vertices_x$ represents the number of tree nodes in the block where the tree node x is located. The $MatureFlag_x$ represents that whether the block where the tree node x is located is $Block_{mature}$. The attribute value $MatureNeighbor$ of a block represents the number of $Block_{mature}$ that is adjacent to this block. Then, frontier detection and the deletion of redundant tree nodes are carried out. The function $Sample()$ indicates sampling x_{rand} randomly in a certain range. The function $Nearest(V, x_{rand})$ indicates finding the nearest valid tree node $x_{nearest}$ to x_{rand}. The function $Steer(x_{nearest}, x_{rand}, \eta)$ indicates generating x_{new} under the constraints of step size η. The function $FindFrontier(map, x_{nearest}, x_{new}) = 1$ indicates that there is a frontier point on the tree edge between $x_{nearest}$ and x_{new}. The function $FindFrontier(map,$

Algorithm 1. Global exploration tree.

```
$BlockInitial(x_{map}, y_{map}, resolution_{block})$;
$V \leftarrow x_{init}$ ;
do
    do
        $x_{rand} \leftarrow Sample()$;
    while $DeleteFlag_{x_{rand}}$ == true;
    $x_{nearest} \leftarrow Nearest(V, x_{rand})$;
    $x_{new} \leftarrow Steer(x_{nearest}, x_{rand}, \eta)$;
while $DeleteFlag_{x_{new}}$ == true;
if $FindFrontier(map, x_{nearest}, x_{new}) = 1$ then
    Publish the found frontier point;
end
if $FindFrontier(map, x_{nearest}, x_{new}) = 0$ then
    $V \leftarrow V \bigcup x_{new}$;
    $OperationBlock(x_{new})$;
end
if $DeleteFlag_{x_{local}}$ == false then
    $V \leftarrow V \bigcup x_{local}$;
    $OperationBlock(x_{local})$;
end
```

$x_{nearest}, x_{new}) = 0$ indicates that the tree edge is located in X_{known}. The function $OperationBlock(x)$ indicates that the valid tree node x will be used to delete redundant tree nodes according to the block rules, as shown in algorithm 2. Finally, local tree nodes in $Block_{external}$ are fused into the set V of the global tree.

Local tree algorithm. The local tree algorithm proposed in this article is shown in algorithm 3. After finding a frontier or growing long enough, the local tree deletes all tree nodes for reset and starts to grow again from the current robot position. The local tree searches for frontiers using the modified RRT but does not obey the same block rules as the global tree does. The meaning of $Sample()$, $Nearest(V, x_{rand})$, $Steer(x_{nearest}, x_{rand}, \eta)$, $FindFrontier(map, x_{nearest}, x_{new}) = 1$, and $FindFrontier(map, x_{nearest}, x_{new}) = 0$ are the same as that of the global tree algorithm.

Analysis of parameter value. In the global tree, there are five important parameters: η, $resolution_{block}$, $vertices_{min}$, $vertices_{max}$, and $number_{delete}$. These five parameters will have a certain effect on the performance of our algorithm. As mentioned in the classification of blocks, only when each of the four blocks adjacent to a block contains sufficient number of tree nodes, this block has the chance to become $Block_{internal}$ and redundant internal tree nodes in it can be deleted. Meanwhile, when $resolution_{block}$ is equal to η, x_{new} tends to be located in the block which is adjacent to the block where $x_{nearest}$ is located, so that redundant internal tree nodes can be deleted as quickly as possible. Thus, we recommend that the value of $resolution_{block}$ should equal to the value of η. Then, the value of η could be selected according to the complexity of the environment. The more complex the environment is, the smaller value of η should be. The minimum value of $vertices_{min}$ can be calculated by $resolution_{block}/\eta$. But if the value of $vertices_{min}$ is too small, it will cause that too few tree nodes in the block can be used to grow new tree nodes. Though this can speed up the deletion of redundant tree nodes, it slows down frontier detection. In contrast, if the value of $vertices_{min}$ is too large, the tree nodes used to grow new ones are sufficient but the deletion of redundant tree nodes will be slowed down. Therefore, the appropriate value of $vertices_{min}$ should be

Algorithm 2. *OperationBlock*(*x*).

```
$Vertices_x$ ++;
if $Vertices_x \geqslant vertices_{min}$ && $MatureFlag_x$ == false then
    $MatureFlag_x$ = true;
    The $MatureNeighbor$ of each block adjacent to the block where $x$ is located ++;
end
if $Vertices_x \geqslant vertices_{max}$ then
    $DeleteFlag_x$ = true;
    if there is an unknown area in the block where $x$ is located then
        $DeleteFlag_x$ = false;
    end
    Delete $number_{delete}$ tree nodes in the block where $x$ is located;
end
$Checklist \leftarrow$ each block whose $MatureNeighbor$ is 4 and not in $Checklist$;
while Checklist != $\varnothing$ do
    $Block_{current} \leftarrow Checklist.pop()$;
    if there is no unknown area in $Block_{current}$ then
        Delete all tree nodes in $Block_{current}$;
        The $DeleteFlag$ of $Block_{current}$ = true;
    end
end
```

Algorithm 3. Local exploration tree.

```
$V \leftarrow x_{init}$ ;
while $time_{run} \leqslant TIME$ do
    $x_{rand} \leftarrow Sample()$;
    $x_{nearest} \leftarrow Nearest(V, x_{rand})$;
    $x_{new} \leftarrow Steer(x_{nearest}, x_{rand}, \eta)$;
    if $FindFrontier(map, x_{nearest}, x_{new}) = 1$ then
        Publish the found frontier point;
        $V \leftarrow x_{current}$;
    end
    if $FindFrontier(map, x_{nearest}, x_{new}) = 0$ then
        $V \leftarrow V \bigcup x_{new}$;
    end
end
```

selected based on the complexity of the environment and the value of $resolution_{block}$ (the more complex the environment is and the greater value of $resolution_{block}$ is, the larger value of $vertices_{min}$ should be), so as to speed up the deletion of redundant tree nodes without affecting frontier detection. In addition, the greater the value of $vertices_{max}$ is, the higher the tree node density in a block could be. Though this can be beneficial to grow new tree nodes outward and the probability of this block to be located in X_{known} will be bigger, the more memory resources will be consumed. Therefore, the appropriate value of $vertices_{max}$ also should be selected based on the complexity of the environment and the value of $resolution_{block}$ (the more complex the environment is and the greater value of $resolution_{block}$ is, the larger value of $vertices_{max}$ should be), so that the number of tree nodes to be stored is less without affecting frontier detection. In this article, we set $vertices_{max}$ double of $vertices_{min}$. Finally, when the number of tree nodes in a block reaches $vertices_{max}$ and $number_{delete}$ tree nodes in it are deleted, there should still be enough tree nodes in it to grow new ones. And $vertices_{min}$ exactly represents the number of sufficient tree nodes used to grow new tree nodes. Therefore, we get that $number_{delete} \leqslant vertices_{max} - vertices_{min}$. In this article, we set $number_{delete} = vertices_{max} - vertices_{min} = 2\times vertices_{min} - vertices_{min} = vertices_{min}$.

Experiments and results

In order to verify performances of our SMF algorithm for frontier detection, we compare it with the frontier detection algorithm based on the classical RRT (RFD).[10] Our experiment video can be found at https://youtu.be/ck-srMisCis. For fair comparison with RFD, we use the same filter module, task allocation module, path planning module, and simultaneous localization and mapping (SLAM) module as used by RFD.[10] The proposed frontier detection algorithm itself is used as the frontier detection module. The frontier detection module is used to find frontier points in the map, the filter module is used to unify the adjacent frontier points into a central frontier point, the task allocation module is used to determine which frontier point should be assigned to the robot as the navigation target, the path planning module is used to find the feasible motion path to the designated navigation target, and the SLAM module is used to locate the robot and map the environment simultaneously during exploration. The above five modules constitute the whole robot exploration system, as shown in Figure 9, and each module can be modified individually to achieve better exploration performances.[10] For specific details, please refer to their paper[10]

Experiments in simulation environments

In this article, gazebo is used to simulate robot exploration of unknown environments. Two simulation environments

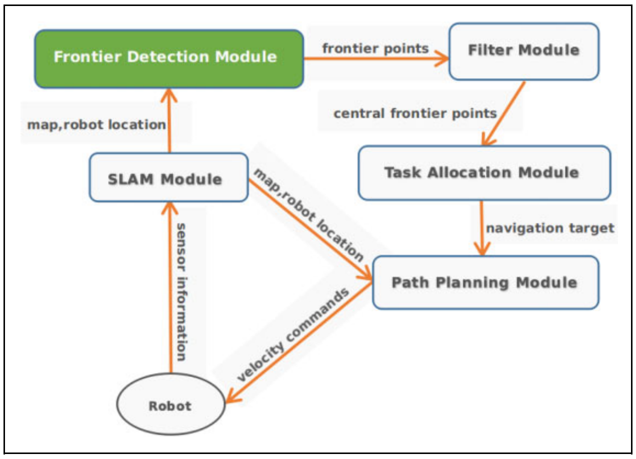

Figure 9. The diagram of the whole robot exploration system.

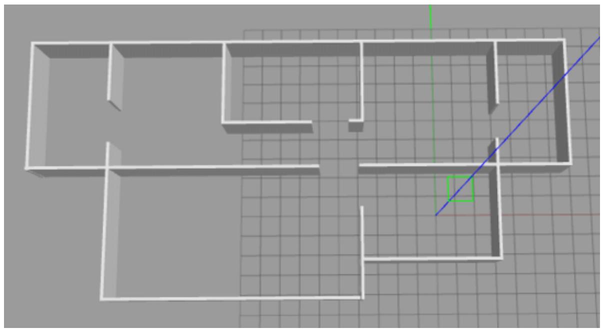

Figure 10. The first simulation environment.

are built. The first simulation environment is about 300 m^2, as shown in Figure 10. Figures 11 and 12 respectively represent the occupancy grid map model of Figure 10 using proposed SMF and RFD. The second simulation environment is about 240 m^2, as shown in Figure 13. Figures 14 and 15 respectively represent the occupancy grid map model of Figure 13 using proposed SMF and RFD. The radius of the robot is 0.175 m. The maximum linear velocity of robot is 0.3 m/s. The maximum linear acceleration of robot is 0.05 m/s^2. The laser's scanning angle is 240° and its maximum detection range is 60 m. In Figures 11 and 14, the red points represent global tree nodes and the purple points represent local tree nodes. If there is a yellow point at the center of a block, it means that this block is $Block_{internal}$ and redundant global tree nodes in it have been deleted. In Figures 12 and 15, the blue tree structure represents the global tree and the purple tree structure represents the local tree.

In simulation experiments, the global tree's step length η in both our SMF and RFD is set to 1, 2, 4, 6, 10 m. Meanwhile, $vertices_{min}$ in our SMF is set to 2, 2, 4, 6, 8 corresponding to each η (1, 2, 4, 6, 10 m). The local tree's step length in both our SMF and RFD is fixed to 1 m. In each simulation environment, we carry out 10 experiments for each global tree's step length using proposed SMF and

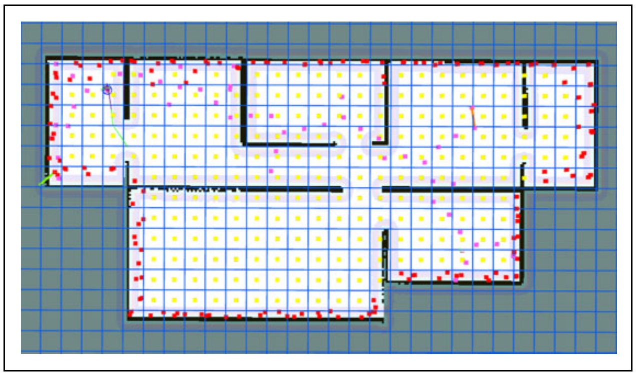

Figure 11. The occupancy grid map model of the first simulation environment after exploration using proposed SMF. SMF: sampling-based multi-tree fusion.

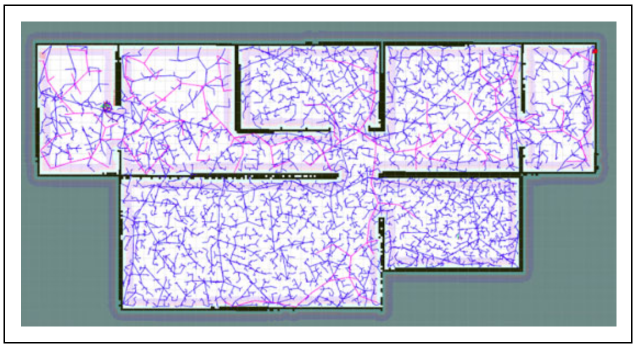

Figure 12. The occupancy grid map model of the first simulation environment after exploration using RFD.

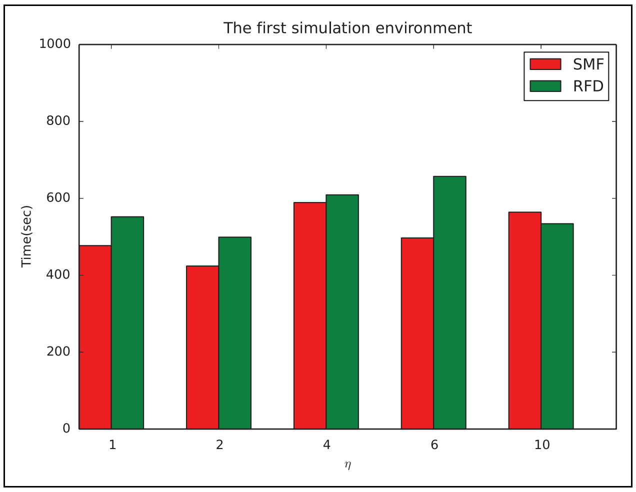

Figure 13. The second simulation environment.

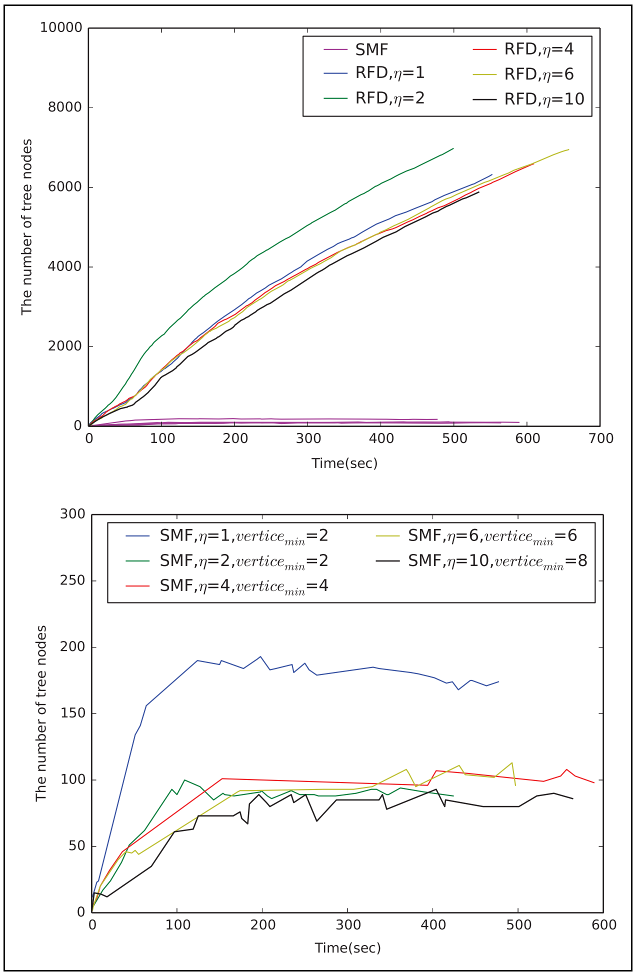

Figure 14. The occupancy grid map model of the second simulation environment after exploration using proposed SMF. SMF: sampling-based multi-tree fusion.

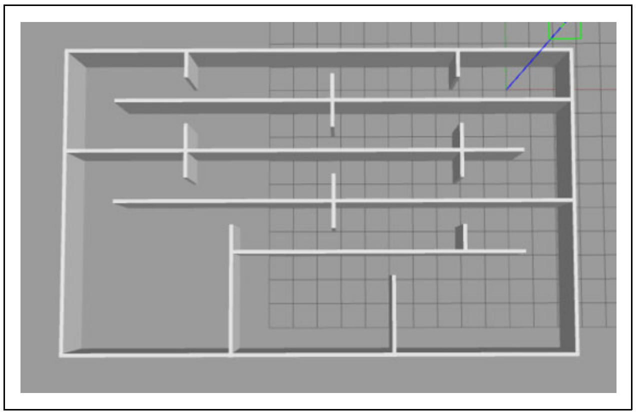

Figure 15. The occupancy grid map model of the second simulation environment after exploration using RFD.

RFD. The total number of experiments in each simulation environment is 100. The mean exploration time is drawn as histograms, as shown in Figures 16 and 17. Because the local tree will regrow after finding a frontier point or a short period of time, it generates few tree nodes and edges and consumes little memory resources which is ignored in this article. We record the number of global tree nodes generated by proposed SMF and RFD at some time points and make them into connection diagrams, as shown in Figures 18 and 19.

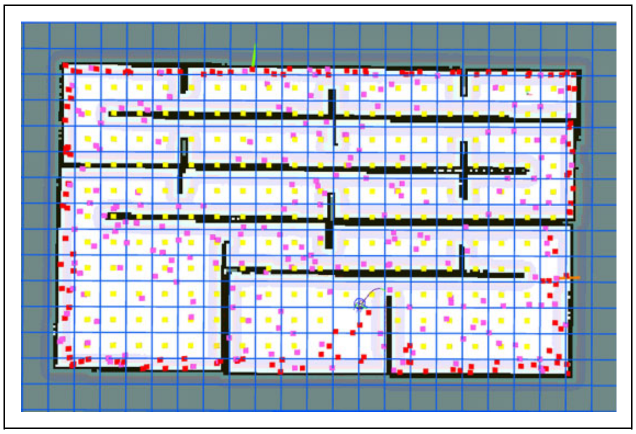
Figure 16. The time contrast histogram of SMF and RFD in the first simulation environment. SMF: sampling-based multi-tree fusion.

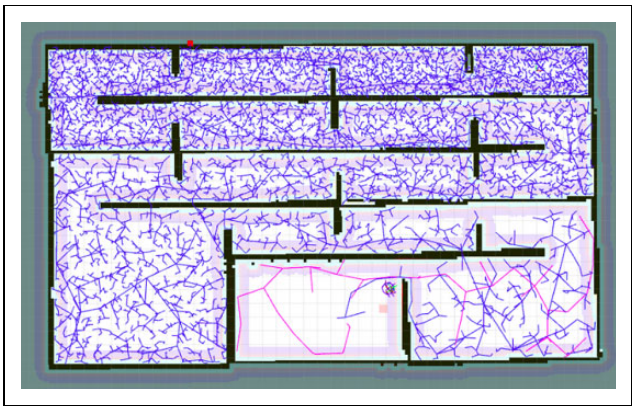
Figure 17. The time contrast histogram of SMF and RFD in the second simulation environment. SMF: sampling-based multi-tree fusion.

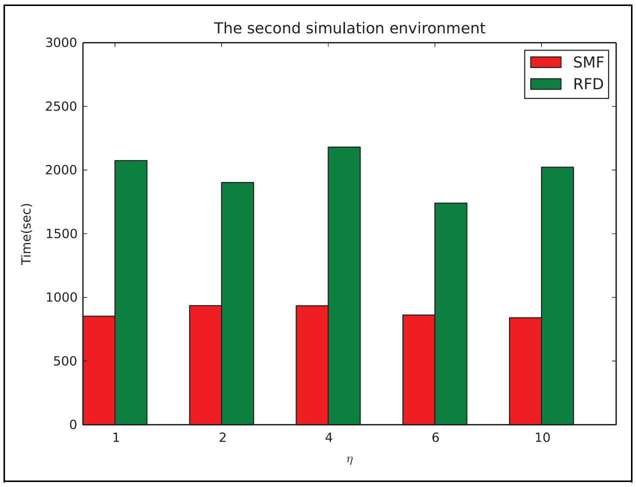

Figure 18. The line chart of the number of tree nodes generated by SMF and RFD in the first simulation environment. SMF: sampling-based multi-tree fusion.

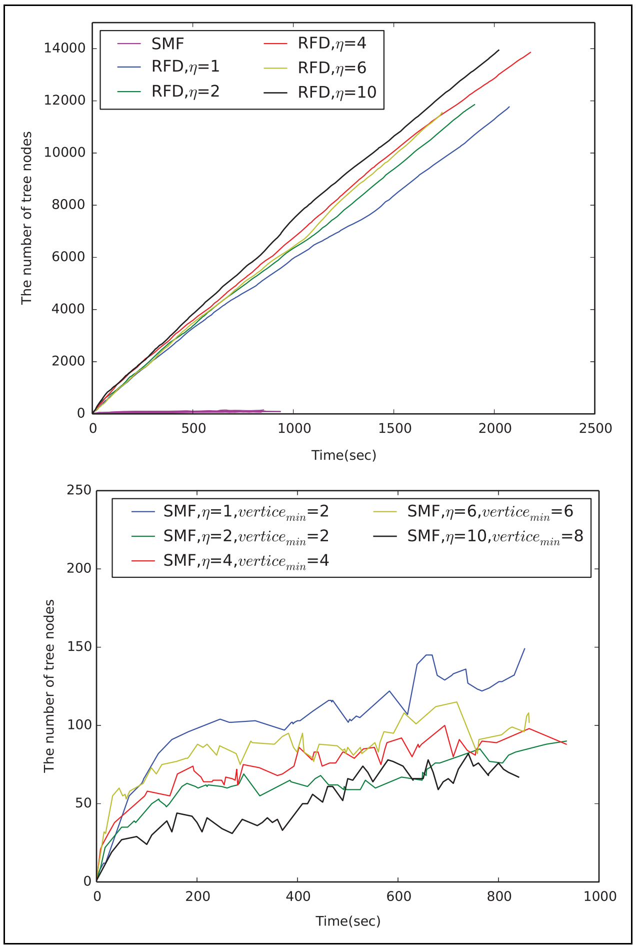

Figure 19. The line chart of the number of tree nodes generated by SMF and RFD in the second simulation environment. SMF: sampling-based multi-tree fusion.

Experiments in the real environment

In this article, a real environment was built with an area of about 170 m^2, as shown in Figure 20. Figures 21 and 22 respectively represent the occupancy grid map model of Figure 20 using proposed SMF and RFD. The Turtlebot 2 is used as the mobile platform. The radius of the robot is about 0.25 m. The maximum linear velocity of robot is set to 0.65 m/s. The maximum linear acceleration of robot is set to 0.06 m/s^2. A Lenovo 80Q4 laptop is used to process data and a Hokuyo UTM-30LX is used to perceive environmental information. The laser's scanning angle is set to 180° and its maximum detection range is 30 m. In Figure 21, the red points represent global tree nodes and the purple points represent local tree nodes. In Figure 22, the blue tree structure represents the global tree and the purple tree structure represents the local tree.

In real experiments, the global tree's step length η in both our SMF and RFD is set to 1, 2, 3, 4 m. Meanwhile, $vertices_{min}$ in our SMF is set to 2, 2, 3, 4 corresponding to each η (1, 2, 3, 4 m). The local tree's step length in both our SMF and RFD is fixed to 1 m. In the real environment, we carry out 10 experiments for each global tree's step length using proposed SMF and RFD. The total number of

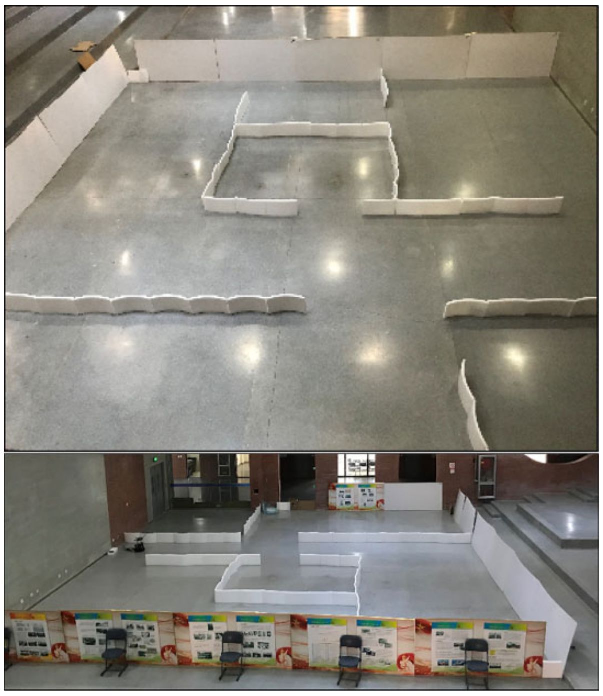

Figure 20. The real environment.

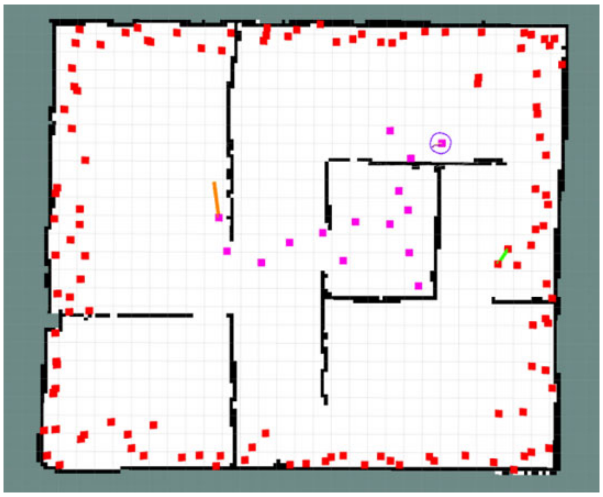

Figure 21. The occupancy grid map model of the real environment after exploration using proposed SMF. SMF: sampling-based multi-tree fusion.

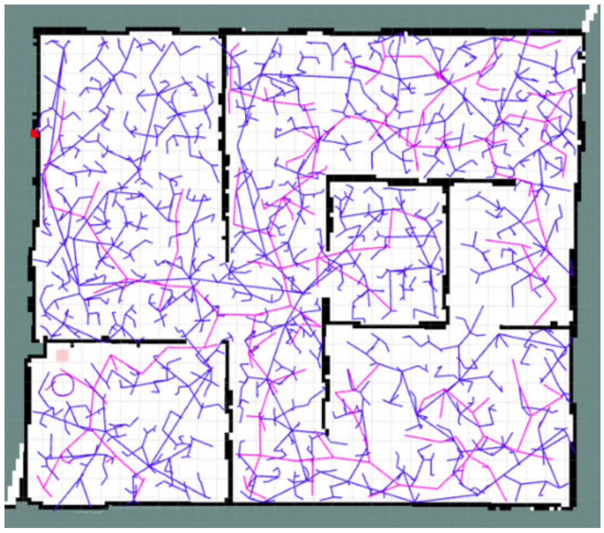

Figure 22. The occupancy grid map model of the real environment after exploration using RFD.

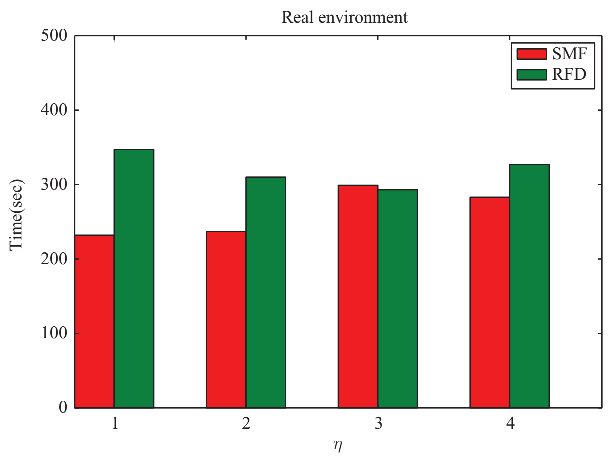

Figure 23. The time contrast histogram of SMF and RFD in the real environment. SMF: sampling-based multi-tree fusion.

experiments in the real environment is 80. The mean exploration time is drawn as histograms, as shown in Figure 23. Because the local tree will regrow after finding a frontier or a short period of time, it generates few tree nodes and edges and consumes little memory resources which is ignored in this article. We record the numbers of global tree nodes generated by proposed SMF and RFD at some time points and make them into connection diagrams, as shown in Figure 24.

Results analysis

As we can see from Figures 14, 19, and 24, the number of tree nodes needed to be stored in SMF converges to the range of formula (3), while the number of tree nodes needed to be stored in RFD increases linearly with time. Under the selected parameters, the number of tree nodes in SMF converges to less than 200 while it can reach more than 10,000 in RFD in the first simulation environment. The number of tree nodes in SMF converges to less than 150 while it can reach more than 12,000 in RFD in the second simulation environment. The number of tree nodes in SMF converges to less than 200 points while it can reach more than 1500 points in RFD in the real environment. Therefore, compared with RFD, our SMF greatly reduces the number of tree nodes needed to be stored during

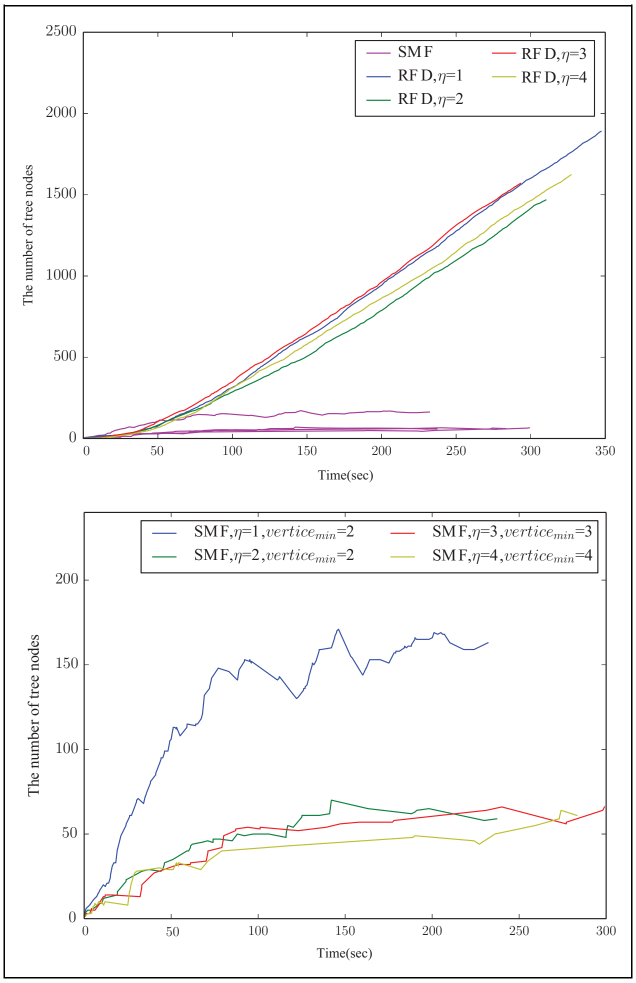

Figure 24. The line chart of the number of tree nodes generated by SMF and RFD in the real environment. SMF: sampling-based multi-tree fusion.

exploration, thus reducing memory consumption. As we can see from Figures 13, 18, and 23, our SMF spent about the same amount of time for exploration as RFD did in the first simulation environment and the real environment. But in the second simulation environment, the time our SMF spent in exploration was reduced to about half the time RFD spent. This is because that the classical RRT grows slowly in the second simulation environment (unfavorable environment), which causes long-time failure to find frontiers in the map. By contrast, we change the random tree's growing and storage rules to overcome the disadvantage of its slow growing under unfavorable environments. In a word, our SMF greatly reduces memory consumption without affecting its performance. In unfavorable environments, our SMF can show a better performance than RFD does and only consumes little memory resources at the same time.

Conclusion

In this article, an SMF algorithm for frontier detection is proposed. Firstly, we allow the RRT to grow across obstacles in known area to overcome its slow growing in unfavorable environments. Secondly, we put forward a block structure and classify blocks into two categorizes ($Block_{internal}$ and $Block_{external}$) to delete redundant tree nodes, which greatly reduces memory consumption. Finally, we fuse the local tree nodes into the global tree to speed up frontier detection. The experimental results show that proposed SMF consumes little memory resources and shows a better performance in unfavorable environments.

In the future, we will focus on expanding our algorithm from 2-D environments to 3-D environments and improving task allocation module and path planning module to obtain better exploration performances.

Declaration of conflicting interests

The author(s) declared no potential conflicts of interest with respect to the research, authorship, and/or publication of this article.

Funding

The author(s) disclosed receipt of the following financial support for the research, authorship, and/or publication of this article: This work was supported by the National Natural Science Foundation of China (grant no. 61573091), the Fundamental Research Funds for the Central Universities (grant no. N182608003, N172608005), the Natural Science Foundation of Liaoning Province (grant no. 20180520006), and the State Key Laboratory of Robotics, China (grant no. 2018-O08).

ORCID iD

Wenchuan Qiao https://orcid.org/0000-0002-5583-514X

References

1. Omari S, Gohl P, Burri M, et al. Visual industrial inspection using aerial robots. In: *Proceedings of the 2014 third international conference on applied robotics for the power industry* (eds S Montambault, Jean-François, N Pouliot and G. Caprari), Foz do Iguassu, Brazil, 14–16 October 2014, pp. 1–5. Piscataway, NJ, USA: IEEE Press.
2. Calisi D, Farinelli A, Iocchi L, et al. Multi-objective exploration and search for autonomous rescue robots. *J Field Robot* 2010; 24(8–9): 763–777.
3. Bennett AA and Leonard JJ. A behavior-based approach to adaptive feature detection and following with autonomous underwater vehicles. *IEEE J Oceanic Eng* 2000; 25(2): 213–226.
4. Weisbin CR and Rodriguez G. NASA robotics research for planetary surface exploration. *IEEE Robot Autom Mag* 2000; 7: 25–34.
5. Yamauchi B. A frontier-based approach for autonomous exploration. In: *Proceedings 1997 IEEE international symposium on computational intelligence in robotics and automation CIRA'97. Towards new computational principles for robotics and automation* (eds A Koivo, T Fukuda, G Lee and C Torras), Monterey, California, USA, 10–11 July 1997, pp. 146–151. Piscataway, NJ, USA: IEEE Press.

6. Keidar M and Kaminka GA. Efficient frontier detection for robot exploration. *Int J Robot Res* 2014; 33(2): 215–236.
7. Lavalle S. *Rapidly-exploring random trees: a new tool for path planning*. Report No. TR 98-11, Computer Science Department, Iowa State University, 1998.
8. Oriolo G, Vendittelli M, Freda LL, et al. The SRT method: randomized strategies for exploration. In: *Proceedings of the IEEE international conference on robotics and automation, ICRA* (eds TJ Tarn and T E;ukuda), Vol. 5, New Orleans, LA, USA, 26 April–1 May 2004, pp. 4688–4694. Piscataway, NJ, USA: IEEE Press.
9. Bircher A, Kamel M, Alexis K, et al. Receding horizon "next-best-view" planner for 3D exploration. In: *2016 IEEE international conference on robotics and automation (ICRA)* (eds D Kragic, T Asfour, M Egerstedt and D Hsu), Stockholm, Sweden, 16–21 May 2016, pp. 1462–1468. Piscataway, NJ, USA: IEEE Press.
10. Umari H, and Mukhopadhyay S. Autonomous robotic exploration based on multiple rapidly-exploring randomized trees. In: *2017 IEEE/RSJ international conference on intelligent robots and systems (IROS)* (ed H Zhang), Vancouver, BC, Canada, 24–28 September 2017, pp. 1396–1402. Piscataway, NJ, USA: IEEE Press.
11. Connolly C. The determination of next best views. In: *Proceedings of the 1985 IEEE international conference on robotics and automation* (ed KS Fu), Vol. 2, St. Louis, Missouri, USA, 25–28 March 1985, pp. 432–435. Piscataway, NJ, USA: IEEE Press.
12. González-Baños HH and Latombe JC. Navigation strategies for exploring indoor environments. *Int J Robot Res* 2002; 21(10–11): 829–848.
13. Vasquez-Gomez JI, Sucar LE, Murrieta-Cid R, et al. Volumetric next-best-view planning for 3D object reconstruction with positioning error. *Int J Adv Robot Syst* 2014; 11(10): 159.
14. Cieslewski T, Kaufmann E, and Scaramuzza D. Rapid exploration with multi-rotors: a frontier selection method for high speed flight. In: *2017 IEEE/RSJ international conference on intelligent robots and systems (IROS)* (ed H Zhang), Vancouver, BC, Canada, 24–28 September 2017, pp. 2135–2142. Piscataway, NJ, USA: IEEE Press.
15. Wettach J, and Berns K. Dynamic frontier based exploration with a mobile indoor robot. In: *ISR 2010 (41st international symposium on robotics) and ROBOTIK 2010 (6th German conference on robotics)* (eds A Verl and K Berns), Munich, Germany, 7–9 June 2010, pp. 1–8. Berlin, Germany: VDE.
16. Mobarhani A, Nazari S, Tamjidi AH, et al. Histogram based frontier exploration. In: *2011 IEEE/RSJ international conference on intelligent robots and Systems* (ed N Amato), San Francisco, CA, USA, 25–30 September 2011, pp. 1128–1133. Piscataway, NJ, USA: IEEE Press.
17. Upadhyay S, Krishna KM, and Kumar S. Fast frontier detection in indoor environment for monocular slam. In: *Proceedings of the tenth Indian conference on computer vision, graphics and image processing, ICVGIP '16* (eds N Ahuja and PK Bora), Guwahati, Assam, India, 18–22 December 2016, pp. 39:1–39:8. New York, NY, USA: ACM.
18. Yamauchi B. Frontier-based exploration using multiple robots. In: *International conference on autonomous agents* (eds KP Sycara and M Wooldridge), St. Paul, Minneapolis, USA, 9–13 May 1998, pp. 47–53. New York, NY, USA: ACM.
19. Kai MW, Stachniss C, and Burgard W. Coordinated multi-robot exploration using a segmentation of the environment. In: *2008 IEEE/RSJ international conference on intelligent robots and systems* (eds M Mataric and P Schenker), Nice, France, 22–26 September 2008, pp. 1160–1165. Piscataway, NJ, USA: IEEE Press.
20. Visser A, and Slamet BA. Including communication success in the estimation of information gain for multi-robot exploration. In: *International symposium on modeling and optimization in mobile, ad hoc, and wireless networks and workshops* (eds H Karl and D Westhoff), Berlin, Germany, 31 March–4 April 2008, pp. 680–687. Piscataway, NJ, USA: IEEE Press.
21. Faigl J, and Kulich M. On determination of goal candidates in frontier-based multi-robot exploration. In: *European conference on mobile robots* (ed J Andrade-Cetto), Barcelona, Catalonia, Spain, 25–27 September 2013, pp. 210–215. Piscataway, NJ, USA: IEEE Press.
22. Reid R, Cann A, Meiklejohn C, et al. Cooperative multi-robot navigation, exploration, mapping and object detection with ROS. In: *2013 IEEE intelligent vehicles symposium (IV)*, Gold Coast City, Australia, 23–26 June 2013, pp. 1083–1088.
23. Adler B, Xiao J, and Zhang J. Autonomous exploration of urban environments using unmanned aerial vehicles. *J Field Robot* 2014; 31(6): 912–939.
24. Heng L, Honegger D, Lee GH, et al. Autonomous visual mapping and exploration with a micro aerial vehicle. *J Field Robot* 2014; 31(4): 654–675.
25. Zhu C, Ding R, Lin M, et al. A 3D frontier-based exploration tool for MAVs. In: *2015 IEEE 27th international conference on tools with artificial intelligence (ICTAI)* (eds L Tsoukalas and G Papadopoulos), Vietri sul Mare, Italy, 9–11 November 2015, pp. 348–352. Piscataway, NJ, USA: IEEE Press.
26. Bähnemann R, Burri M, Galceran E, et al. Sampling-based motion planning for active multirotor system identification. In: *2017 IEEE international conference on robotics and automation, ICRA 2017* (ed I-M Chen), Singapore, Singapore, 29 May–3 June 2017, pp. 3931–3938. Piscataway, NJ, USA: IEEE Press.
27. Freda L, and Oriolo G. Frontier-based probabilistic strategies for sensor-based exploration. In: *Proceedings of the 2005 IEEE international conference on robotics and automation* (ed A Casals), Barcelona, Spain, 18–22 April 2005, pp. 3881–3887. Piscataway, NJ, USA: IEEE Press.
28. El-Hussieny H, Assal SFM, and Abdellatif M. Improved backtracking algorithm for efficient sensor-based random tree exploration. In: *Fifth international conference on computational intelligence, communication systems and networks* (eds D Kotz, S Kalyanaraman and B Raman), Madrid, Spain, 5–7 June 2013, pp. 19–24. Piscataway, NJ, USA: IEEE Press.

29. Palacios AT, Sánchez LA, and Bedolla Cordero JME. The random exploration graph for optimal exploration of unknown environments. *Int J Adv Robot Syst* 2017; 14(1): 1–11.
30. Kim J, Bonadies S, Lee A, et al. A cooperative exploration strategy with efficient backtracking for mobile robots. In: *2017 IEEE international symposium on robotics and intelligent sensors (IRIS)* (ed E Holdrinet), Ottawa, Canada, 5–7 October 2017, pp. 104–110. Piscataway, NJ, USA: IEEE Press.
31. Papachristos C, Khattak S, and Alexis K. Uncertainty-aware receding horizon exploration and mapping using aerial robots. In: *2017 IEEE international conference on robotics and automation (ICRA)* (ed I-M Chen), Singapore, Singapore, 29 May–3 June 2017, pp. 4568–4575. Piscataway, NJ, USA: IEEE Press.
32. Witting C, Fehr M, Bähnemann R, et al. History-aware autonomous exploration in confined environments using mavs. *CoRR* 2018; abs/1803.10558.
33. Dang T, Papachristos C, and Alexis K. Visual saliency-aware receding horizon autonomous exploration with application to aerial robotics. In: *2018 IEEE international conference on robotics and automation (ICRA)* (eds A Zelinsky), Brisbane, Australia, 21–25 May 2018, pp. 2526–2533. Piscataway, NJ, USA: IEEE Press.
34. Visser A, Xingrui-Ji, van Ittersum M, et al. Beyond frontier exploration. In: *RoboCup 2007: Robot Soccer World Cup XI* (eds U Visser, F Ribeiro, T Ohashi and F Dellaert), Atlanta, GA, USA, 9–10 July 2007, pp. 113–123. Berlin, Heidelberg, Germany: Springer.
35. Charrow B, Kahn G, Patil S, et al. Information-theoretic planning with trajectory optimization for dense 3D mapping. In: *Robotics: Science and Systems* (ed LE Kavraki), Sapienza University of Rome, Rome, Italy, 13–17 July 2015, Cambridge, MA, USA: MIT Press.
36. Senarathne PGCN, and Wang D. Towards autonomous 3D exploration using surface frontiers. In: *2016 IEEE international symposium on safety, security, and rescue robotics (SSRR)* (ed A Ijspeert), Lausanne, Switzerland, 23–27 October 2016, pp. 34–41. Piscataway, NJ, USA: IEEE Press.
37. Meng Z, Qin H, Chen Z, et al. A two-stage optimized next-view planning framework for 3-D unknown environment exploration, and structural reconstruction. *IEEE Robot Autom Let* 2017; 2: 1680–1687.