Intel Serv Robotics (2013) 6:199–209
DOI 10.1007/s11370-013-0137-3

ORIGINAL RESEARCH

# Local map-based exploration for mobile robots

**Hyejeong Ryu · Wan Kyun Chung**

Received: 25 April 2013 / Accepted: 7 September 2013 / Published online: 27 September 2013

**Abstract** For an accurate and efficient exploration, a local map-based exploration strategy is proposed. Segmented frontiers and relative transformations constitute a tree structure; using frontier segmentation and a local map management method, a robot can expand the mapped environment by moving along the tree structure. Although this local map-based exploration method uses only local maps and adjacent node information, mapping completion and efficiency can be greatly improved by merging and updating the frontier nodes. Simulation results demonstrate that the computational time does not increase during the exploration process, or when the resulting map becomes large. Additionally, the resulting path is effective in reducing the uncertainty in simultaneous localization and mapping or localization because of the loop-inducing characteristics from the child node to the parent node.

**Keywords** Mobile robots · Exploration · Graph search · Navigation

## 1 Introduction

Constructing a map of an unknown environment is an important task for mobile robots. Over the past two decades, a number of studies have reported methods of representing the environment and localizing robots using sensor data. Simultaneous localization and mapping (SLAM) techniques can estimate the position of a robot and landmarks simultaneously based on noisy sensor data acquired by the moving robot. Studies of SLAM have mainly focused on the accuracy of the estimated states [1–3]; however, they have not dealt with the mapping strategy, i.e., how to autonomously determine the next position for effective mapping.

H. Ryu · W. K. Chung (✉)
Robotics Laboratory, Department of Mechanical Engineering, Pohang University of Science and Technology (POSTECH), Pohang, Korea
e-mail: wkchung@postech.ac.kr

H. Ryu
e-mail: roohj@postech.ac.kr

### 1.1 Frontier-based exploration

Exploration algorithms determine a path for a robot to achieve autonomous mapping. Most exploration approaches are based on detecting frontiers in the occupancy grid map to reduce the unmapped area and extend information about the environment [4–6]. Frontiers are the edge areas between the explored (i.e., mapped) and unexplored (i.e., unmapped) regions. A method to calculate the information gain in the occupancy grid map was reported in [7]. These previous approaches concentrated on coverage of the entire environment. Recently, integrated exploration techniques have been proposed; this kind of technique combines SLAM and path planning to decide the next position of a robot based on the uncertainty of the SLAM state [8,9]. Integrated exploration methods allow SLAM and path planning to affect each other in the evaluation of utilities of frontier candidates. If the SLAM state is uncertain, the robot prefers a position where it can localize itself more accurately, whereas if the SLAM estimation is reliable, the robot keeps moving to reduce the unmapped area. [9] uses extended Kalman filter (EKF) for SLAM, whereas [10] uses Rao-Blackwellized particle filter (RBPF) and the cost of reaching the possible destination is calculated by the expected information gain. Most of these exploration approaches proceed as follows:

(1) Generate frontier candidates on the grid map.

(2) Evaluate the candidates according to the mapping coverage, the uncertainty of the SLAM state, and the navigation cost.
(3) Determine a destination from a number of candidate destinations.

In the frontier-based method, it is necessary to spread out the map information of an unknown environment and inspect the completion of mapping for the entire environment. For this reason, an occupancy grid map is required to detect frontiers efficiently, even when feature-based SLAM is used for state estimation.

To date, exploration approaches typically construct and manage a single grid map using localization results to extract frontiers on this map. Because the map is updated regardless of the uncertainty in the localization of the robot, the map will be inaccurate if the localization state is inaccurate. An inaccurate map can lead to incorrect frontier information and inefficient exploration. When the position accuracy has been recovered following loop closing, an efficient map updating algorithm is required. However, in general it is difficult to update the grid map using the single global map unless the robot has access to data about all the past trajectories and corresponding sensor measurements. Although the algorithms developed to date [9–11] consider the expected uncertainty of the pose estimation on the frontier candidates during exploration, they do not include a strategy for managing the grid map based on pose uncertainty. Moreover, to cope with the large and complex environment or the long travel distance, a local map approach may be more appropriate for exploration.

## 1.2 Local map-based mapping

Several local map-based mapping approaches have been developed [12–15], in which a hierarchical SLAM creates local maps according to the features or the uncertainty of the vehicle location. Each local map has its own reference frame, and relative transformations between local maps allow construction of a global topological graph of the environment. Loop closing makes these local maps globally consistent. The path of the robot can be reconstructed and used to build hybrid maps, in which abstracted topological nodes have a local metric grid map [16]. Here, RBPF is used for the metric estimation, whereas former approaches use EKF.

In contrast to the above metric feature-based approaches, [17] introduced a probabilistic framework for appearance-based topological mapping. This is the enhanced version of [18] that learns a probabilistic model of scene appearance online using a generative model of visual word and adds the observation of spatial ranges between words. [19] incorporates the odometric information into appearance-based SLAM systems, without performing metric map construction or calculating relative feature geometry. [20] deals with the topological mapping in indistinguishable places of real environments using sonar-based fingerprints of places. These local-map based methods provide accurate and consistent mapping, but do not consider the next target to achieve autonomous mapping.

## 1.3 Local map-based exploration

For accurate and efficient exploration, we propose a local map-based exploration strategy, in which multiple local maps are constructed to map the environment correctly and to determine the next frontier target position systematically. These local maps have a tree structure, using the candidate frontiers (nodes) and their relative transformations (edges). Once a frontier has been visited, a local map, constructed using local sensor data obtained on that frontier position, is assigned to the frontier node. By inspecting the local map assigned for each frontier node, new frontier nodes can be added, and then the next target node will be determined according to the tree structure and the graph search algorithm.

We apply and modify the depth-first search (DFS) to decide the next target in the local map-based exploration. In our exploration method, the robot not only goes to an unvisited node for expanding the map information, but may also return to an already visited node for reducing the pose uncertainty. This behavior is related to the active localization because the robot autonomously selects among frontier nodes for the accurate pose estimation. There have been many researches about the active localization. [21] chooses the action that minimizes the expected entropy. In [22], the orientation of the laser range finder is actively selected to improve the localization results. [23] proposes the method that generates optimal macro actions to localize even in self-similar environment. [24] represents the environment as abstracted semantic landmarks and uses spatial relationship among them to select a robot's action and improve localization results. These researches assume the already known map. However, our method focuses on selecting a frontier node to make a loop constraint between previously visited nodes in an unknown environment. This loop constraint is useful to reduce the pose uncertainties of frontier nodes and edges between them.

Our strategy has a number of significant advantages: (a) it is not restricted by the accumulated position error, because the procedure for detecting frontiers and selecting the next target destination is executed at the local level; (b) it can efficiently update and merge local maps using the tree information and corresponding local maps after loop closing; (c) it can systematically manage the number of nodes that must be explored; (d) computational time does not increase even when the resulting map grows; (e) it can generate a loop-closing path by returning to the parent node after the robot has completed the local exploration at the child node.

The rest of this paper is organized as follows. Section 2 presents an overview of the local map-based exploration process. Section 3 introduces the tree structure of frontier nodes and the local map database (DB). Section 4 describes a method for deciding the next target node using a depth-first search algorithm, and for updating the frontier tree DB after loop closing. Section 5 presents the simulation results for two different kinds of environments. We conclude this paper in Sect. 6.

## 2 Algorithm overview

Figure 1 presents an overview of the algorithm. This flowchart illustrates the procedure and the corresponding data management for registering new frontier nodes to the tree DB and updating the tree and map DBs by merging local grid maps. Because the proposed algorithm uses only local information to detect frontiers and decide the next target, local maps of adjacent neighbor nodes are used to guarantee that mapping of the entire environment is complete. This process differs from conventional frontier-based methods, which use global information.

The algorithm has two parts: registering new frontier nodes and updating the DB upon loop closing. The first is performed at each node when the robot first explores it. Loop closing occurs when the robot revisits that node.

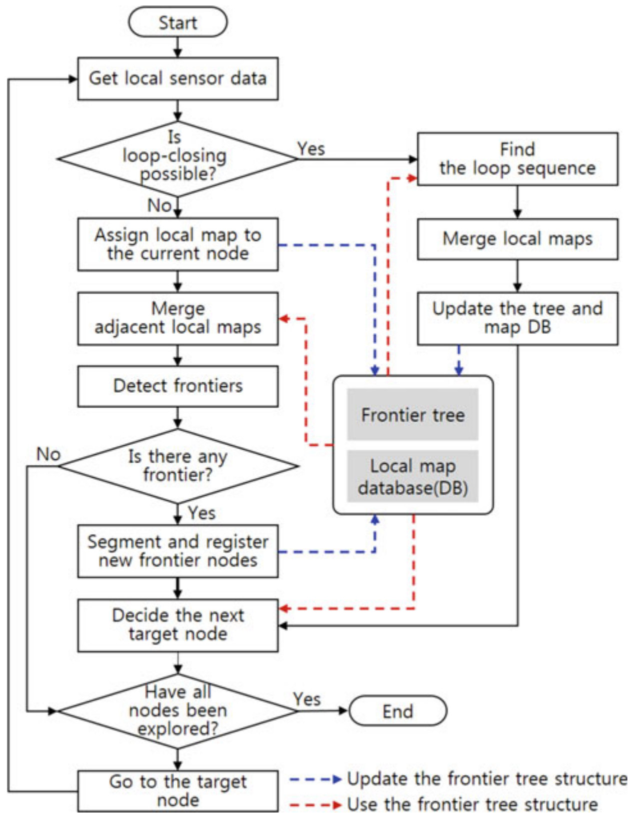
Start
Get local sensor data
Is loop-closing possible?
Yes
No
Find the loop sequence
Assign local map to the current node
Merge local maps
Merge adjacent local maps
Update the tree and map DB
Detect frontiers
Frontier tree
Local map database(DB)
No
Is there any frontier?
Yes
Segment and register new frontier nodes
Decide the next target node
Have all nodes been explored?
Yes
End
Go to the target node
Update the frontier tree structure
Use the frontier tree structure

**Fig. 1** An overview of the algorithm

At the new frontier node, the current local grid map is assigned to the current node of the tree structure and map DB. Adjacent local maps, corresponding to the parent node or previous node, are merged with the current local map to detect frontiers. If frontier cells are detected, they are segmented into representative frontiers to be registered as nodes of the tree. If no new frontier appears, the current node is regarded as completely explored and the algorithm finds the next target node using the tree.

When the robot revisits a node, that is, loop closing becomes possible, it is necessary to find the loop nodes and the corresponding sequence. Because the loop-closing constraint according to the loop sequence leads to accurate relative transformations between loop nodes, local maps assigned to loop nodes can be merged using the corrected relative transformations. The merged local map is assigned to the loop nodes, so that loop nodes have the same grid map. Additionally, the edge information of the tree is updated using the corrected relative transformations. The robot uses the updated frontier tree and local map DB to find the next target. If all nodes have been completely explored, i.e., there is not an unvisited frontier node in the tree, the process of exploration can be considered complete. The following sections present details of the construction of the frontier tree structure and the merging of local maps for detecting frontiers and loop closing are described. The next target is determined using the tree structure and graph search algorithm.

## 3 Tree structure and map database

In the proposed strategy, the robot moves between the nodes of the tree structure during exploration. Because the efficiency of the graph search algorithm depends on the number of nodes, the number of frontier nodes affects the efficiency of exploration. If every frontier cell is registered as a node, the computational load will be large. Segmenting frontiers and registering one representative node for each frontier segment reduce the computational cost of the algorithm.

### 3.1 Frontier node segmentation

Using a grid map and its binary image, we can detect edge cells [6]. Computer vision approaches, such as Canny-edge detection, can be used, and the resulting edge cells include both frontier edge cells and occupied edge cells. It is necessary to identify frontier cells, which are not adjacent to any occupied cells. We can use the entropy of each edge cell, which is a measure of the amount of information of a cell [9], to identify frontier cells. The entropy of an occupancy grid cell $(i, j)$ is

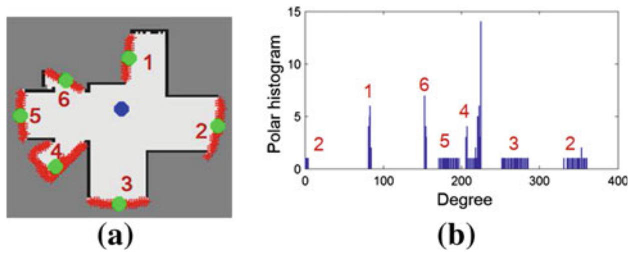

**Fig. 2** Frontier segmentation using a polar histogram. **a** The *blue dot* indicates the current node, the *light green dot* indicates the representative node, and the *red star* indicates a frontier cell. **b** Polar histogram (color figure online)

$$h(i,j) = -P_{i,j}(O)\log P_{i,j}(O) - P_{i,j}(E)\log P_{i,j}(E) \tag{1}$$

where $P_{i,j}(O)$ is the probability of being occupied and $P_{i,j}(E)$ is the probability of being empty. This entropy is highest when there is a uniform probability distribution; in other words, the unknown cell, $P_{i,j}(O) = P_{i,j}(E) = 0.5$, has the highest entropy. The sum of the entropy within a window $W$ around the frontier is calculated from Eq. (2).

$$H = \sum_{i,j\in W} h_{i,j} \tag{2}$$

An edge with a higher sum of entropy corresponds to more unknown regions around that edge, so it is more profitable for exploration. Frontier cells are selected when the sum of the entropy is higher than a constant describing the tolerance for a frontier location.

A polar histogram of selected frontier cells can be calculated based on the current robot position. Frontiers are segmented according to this histogram distribution, as shown in Fig. 2. Before frontier segmentation, frontier cells on the current local grid map are inspected again using the adjacent local grid map. That is, the previous and parent local grid maps are merged with the current local map. Grid cells that are determined to be frontier cells both in the current map and in the merged map are assigned as frontiers, as shown in Fig. 3.

If a robot is in an empty space, with frontiers detected in all directions, only one histogram segment is available. These frontiers must be uniformly divided into at least three segments to define all the current frontiers as “empty” or “occupied” after exploring all child nodes. Figure 4 shows examples in which a robot has two child nodes or three child nodes in the empty space. If frontiers are segmented into just two child nodes, some frontier cells remain unknown even after all child nodes have been explored, which can hinder the completion of mapping. For this reason, we divide the polar histogram segment with an angle difference, $\Delta\theta$, greater than $120°$ into $n = \lceil \frac{\Delta\theta}{120°} \rceil$ smaller segments.

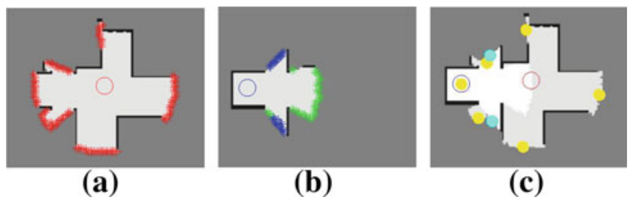

**Fig. 3** Frontier detection on the merged map. **a** The *red stars* indicate frontiers on the parent map, and the *red circle* indicates the position of the parent node. **b** The *blue stars* indicate frontiers on the merged map and the current map, and the *light green stars* indicate frontiers on the current map that are known on the merged map. The *blue circle* indicates the current node position. **c** The *yellow dots* indicate the segmented frontiers (child nodes) on the parent map, and the *cyan dots* indicate child nodes of the current node (color figure online)

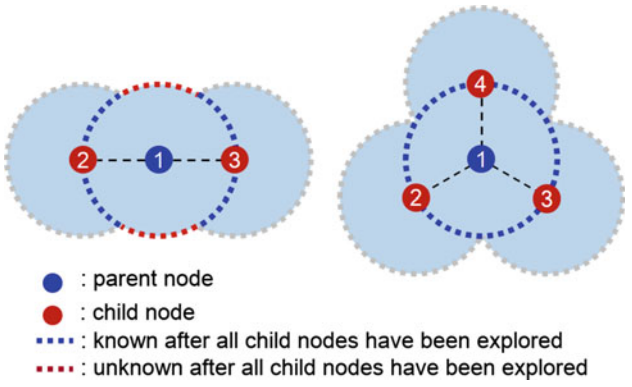
1
2
3
4
● : parent node
● : child node
: known after all child nodes have been explored
: unknown after all child nodes have been explored

**Fig. 4** Frontier segmentation in an empty space

The nearest cell to the average position of each segment, $(i,j) = \left(\sum_{k=1}^{N}\frac{i_k}{N}, \sum_{k=1}^{N}\frac{j_k}{N}\right)$, is selected as the representative node, i.e.,

$$(X,Y) = \text{cell}\,(I,J) = \arg\min_{i_k,j_k}\left\|(i-i_k)^2 + (j-j_k)^2\right\| \tag{3}$$

where $N$ is the number of frontier cells in each segment and $(X,Y)$ is the spatial position in the Cartesian coordinate of the grid cell cell$(I,J)$. This representative cell for each frontier segment is registered as the node of the tree, and the relative transformations between frontier nodes and the current robot position are the edges. Section 3.2 describes the tree structure, which consists of the representative nodes and the map DB.

## 3.2 Tree and local map database

In the local map-based method, exploration is the process of expanding the tree by detecting and adding frontier nodes until no further nodes are discovered. Eventually, the environment can be represented as distributed nodes and edges between these nodes. Figure 5 presents the structure of a frontier tree, and Table 1 lists the details of the components.

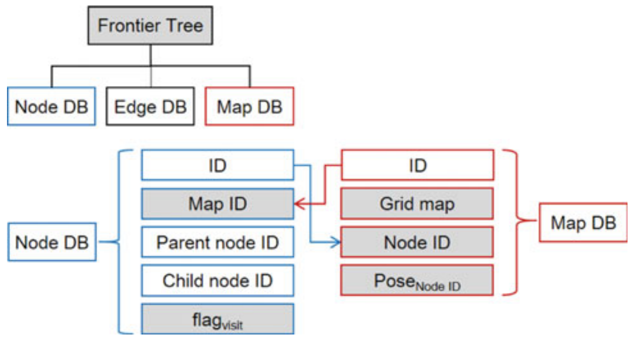

**Fig. 5** Structure of a frontier tree

**Table 1** Frontier tree DB

$Node = \{node_{id} |\ 1 \le id \le N\}$

$node_{id} = \left\{ \begin{matrix} id,\ id_{\text{map}},\ id_{\text{parent}},\ flag_{\text{visit}}, \\ \{id_{\text{child}_1}, \ldots, id_{\text{child}_n}\} \end{matrix} \right\}$

$Map = \{map_{id} |\ 1 \le id \le M\}$

$map_{id} = \left\{ \begin{matrix} id,\ \text{gridmap}_{id},\ \{id_{node_1}, \ldots, id_{node_m}\}, \\ \{pose_{id,node_1}, \ldots,\ pose_{id,node_m}\} \end{matrix} \right\}$

$Edge^{-} = \{edge^{-}_{i,j} |$ predicted relative transformation, $1 \le i \le N,\ 1 \le j \le N\}$

$edge^{-}_{i,j} = \left\{\text{distance}^{-}_{i,j},\ x^{-}_{i,j},\ y^{-}_{i,j},\ \theta^{-}_{i,j}\right\}$

$Edge^{+} = \{edge^{+}_{i,j} |$ actual relative transformation, $1 \le i \le N,\ 1 \le j \le N\}$

$edge^{+}_{i,j} = \left\{\text{distance}^{+}_{i,j},\ x^{+}_{i,j},\ y^{+}_{i,j},\ \theta^{+}_{i,j}\right\}$

$N$ the number of nodes
$n$ the number of child nodes
$M$ the number of maps
$m$ the number of nodes which share the $map_{id}$
$pose_{id,node_m}$ the pose of $node_m$ in $map_{id}$

Information about each frontier node, including the parent node, child nodes, corresponding local map, and the exploration state ($flag_{\text{visit}}$) is stored in the node DB. When new child nodes are detected, each child node is given its own ID and the parent ID is stored at the child node. At the same time, child node IDs are also stored at the parent node. Exploiting the DB where the parent and child node IDs are stored can facilitate more efficient exploration. An exploration state, logical flag can be set to keep a record of whether the node has been explored, and used when a robot subsequently visits the node to improve the efficiency of the search algorithm.

The map DB contains information about the local maps constructed at the frontier nodes, and each local map has its own ID and occupancy data. The information about each local map also includes the corresponding node ID and its position in the local map. At the new frontier node, the local map, constructed with respect to the node, has one node ID and the position of the node is (0, 0, 0). After loop closing, multiple local maps will be merged. Consequently, the map information has several node IDs, corresponding to the loop nodes and their positions on the merged map.

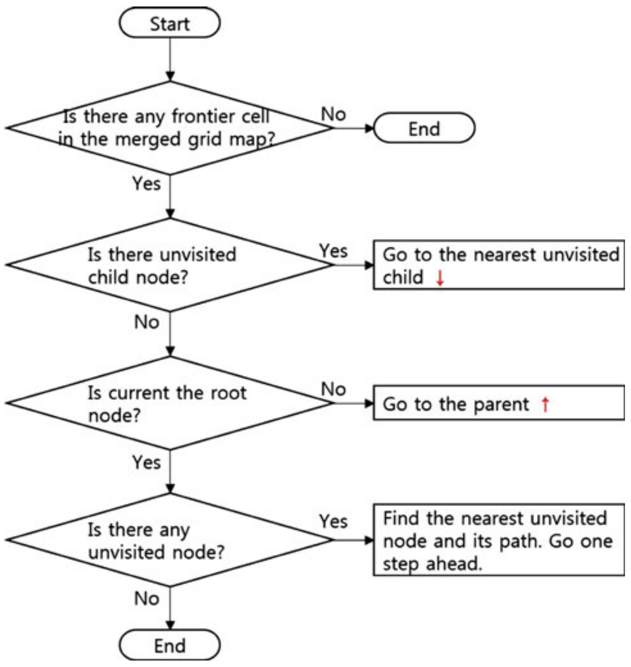
Start
Is there any frontier cell in the merged grid map?
No
End
Yes
Is there unvisited child node?
Yes
Go to the nearest unvisited child ↓
No
Is current the root node?
No
Go to the parent ↑
Yes
Is there any unvisited node?
Yes
Find the nearest unvisited node and its path. Go one step ahead.
No
End

**Fig. 6** The depth-first search algorithm for local map-based exploration

The edge DB includes the relative transformation ($x_{i,j}$, $y_{i,j}$, $\theta_{i,j}$) and the shortest distance ($\text{distance}_{i,j}$) between the nodes. This forms an $N \times N$, where $N$ is the number of nodes. Two kinds of edge DB are used: the predicted edge information ($Edge^{-}$), which is obtained when new nodes are detected at the parent node and added to the node DB; and the relative transformation calculated from the actual travel path after the robot has arrived at the node ($Edge^{+}$). The former can be used to select the next target node, and the latter is useful to decide the loop sequence and the constraint for loop closing. Section 4 describes the procedure for selecting the next target node using this tree structure.

## 4 Determining the next target and loop closing

One key element in an exploration algorithm is efficiently determining the next target. Conventional approaches find the next target by calculating the various costs of the detected frontier cells at each decision stage; in contrast, the local map-based exploration uses the frontier tree structure. The tree includes information about the relationship between the parent and child nodes, as well as the distance between adjacent frontier nodes. This structure can be useful to find the nearest frontier node to the current node, thereby facilitating efficient exploration and search, expediting loop closing, and improving the accuracy of the relative transformations.

### 4.1 Depth-first search for exploration

The next target node can be determined using a depth-first search (DFS) algorithm. DFS is generally applied to search the shortest path in a predetermined environment, i.e., the graph information is perfectly known. However, in the local map-based exploration, the graph structure is initially unknown and is expanded until no further frontier nodes are found. We modify and apply the DFS for exploration. Every frontier node constitutes the tree structure and DFS prefers a forward-searching path, and so DFS is suitable for exploration.

DFS exploration involves two directions. One is downward from the current node to the unexplored child node; this allows exploration of unknown areas and expansion of the map. The other is upward from the current node to the parent node. In the frontier tree, every frontier node has only one parent node (except for the initial starting node). When no new frontiers appear at the current node, this node is regarded as completely explored and the robot returns to the parent node. This direction induces loop closing, and is an important advantage for exploration. It is well known that loop closing reduces the pose uncertainty in SLAM problem [25]. Although the parent has several child nodes and the relative transformations can be obtained from the local map, movement between sibling nodes is not allowed. This restriction makes the movement to the parent occur more frequently.

Figure 6 shows a flowchart describing the DFS local map-based exploration algorithm. First, it determines whether the merged map includes frontiers. The merged map used to detect new frontier cells described in Sect. 3.1 is applied again to decide whether mapping is complete. If the merged map includes no frontiers, this indicates that the robot is in a closed space and all regions of the merged map are known.

The robot continues exploration until no new frontier nodes are detected around the current node. After the current node has been completely explored, the robot returns to the parent and this process is repeated until the robot has visited all nodes in the frontier tree. The predicted edge DB, $edge^{-}_{i,j}$, and in particular distance$^{-}_{i,j}$, is used as the adjacency matrix for DFS exploration.

### 4.2 Loop closing

Assuming that the relative transformations between loop nodes are accurate due to loop closing, corresponding local maps can be merged. The merged map is important for exploration efficiency, because it can overcome the lack of information in the local map, i.e., it can select the actual frontiers. Frontiers in the current local map that are known regions of the merged map can be registered, if we do not compare the local frontier information with the merged map. The map-

Node and map DB before merging

Parent : $node_1 = \{1, m_1, 0, \text{true}, \{2\}\}$

$$map_{m_1} = \left\{ \begin{array}{l} m_1, \text{gridmap}_{m_1}, \{1\}, \\ \{pose_{m_1,node_1}\} \end{array} \right\}$$

Child : $node_2 = \{2, m_2, 1, \text{true}, \emptyset\}$

$$map_{m_2} = \left\{ \begin{array}{l} m_2, \text{gridmap}_{m_2}, \{2\}, \\ \{pose_{m_2,node_2}\} \end{array} \right\}$$

Node and map DB after merging

Parent : $node_1 = \{1, m_1, 0, \text{true}, \{2\}\}$

Child : $node_2 = \{2, m_1, 1, \text{true}, \emptyset\}$

Two nodes shares the same map :

$$map_{m_1} = \left\{ \begin{array}{l} m_1, \text{gridmap}_{m_1} \cup \text{gridmap}_{m_2}, \{1, 2\}, \\ \{pose_{m_1,node_1}, pose_{m_1,node_2}\} \end{array} \right\}$$

**Fig. 7** An example of map DB update: following loop closing at the parent, node$_1$, data about the child, node$_2$, is updated with respect to the parent

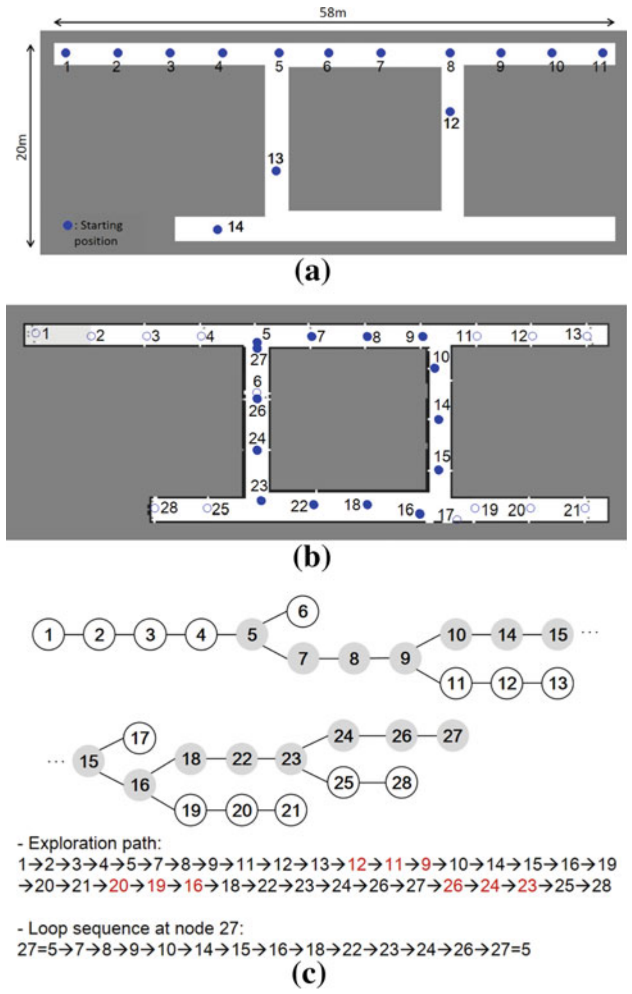

**Fig. 8** Simulation result in environment 1. **a** Environment 1 and the starting positions, indicated by the *blue dots*. **b**, Simulation result using starting position 1; the *blue circles* and *blue dots* indicate frontier nodes, and the number shown is the node ID. **c** The resulting tree structure of **b** (color figure online)

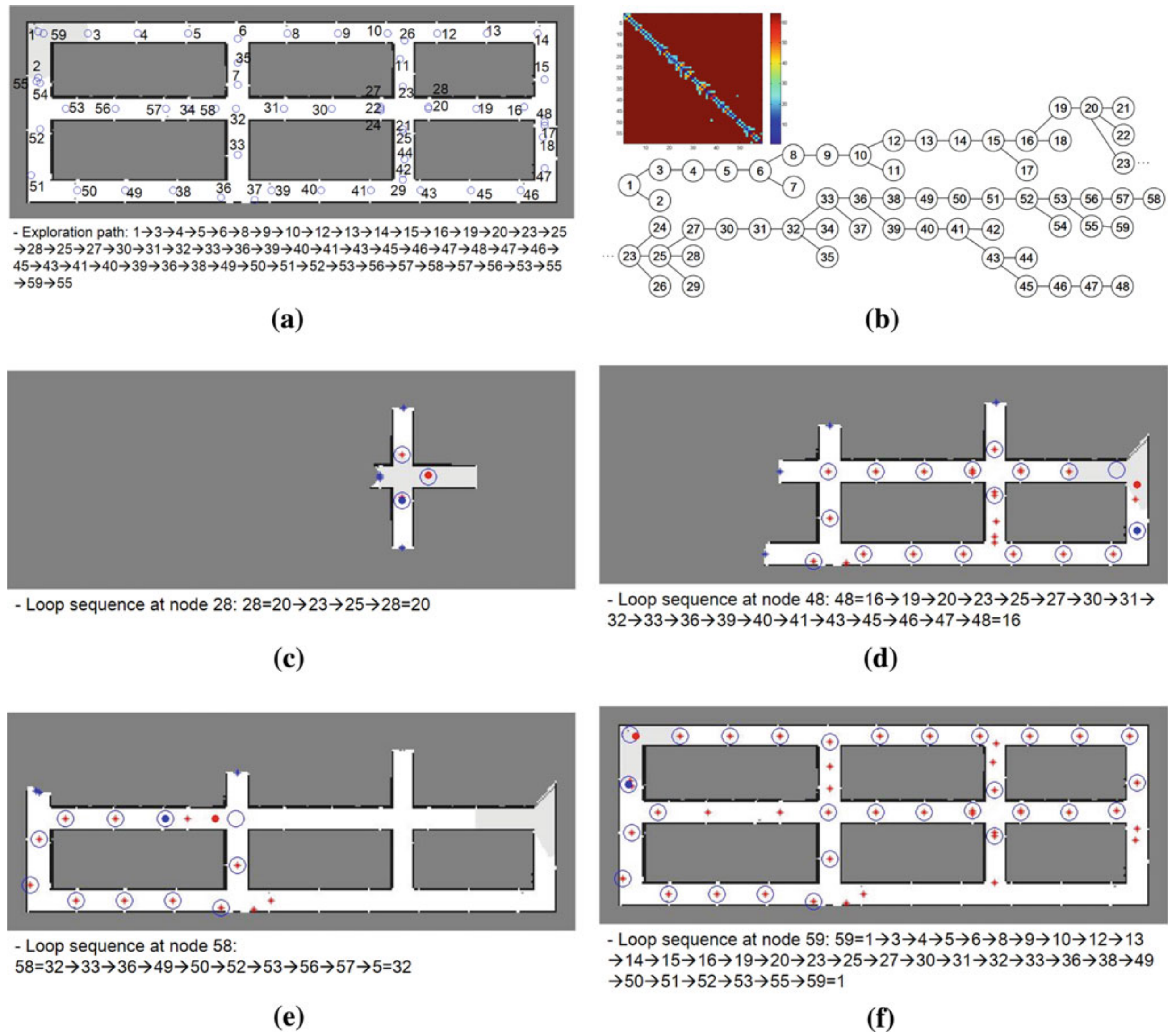
- Exploration path: 1→3→4→5→6→8→9→10→12→13→14→15→16→19→20→23→25→28→25→27→30→31→32→33→36→39→40→41→43→45→46→47→48→47→46→45→43→41→40→39→36→38→49→50→51→52→53→56→57→58→57→56→53→55→59→55
(a)
(b)
- Loop sequence at node 28: 28=20→23→25→28=20
(c)
- Loop sequence at node 48: 48=16→19→20→23→25→27→30→31→32→33→36→39→40→41→43→45→46→47→48=16
(d)
- Loop sequence at node 58: 58=32→33→36→49→50→52→53→56→57→5=32
(e)
- Loop sequence at node 59: 59=1→3→4→5→6→8→9→10→12→13→14→15→16→19→20→23→25→27→30→31→32→33→36→38→49→50→51→52→53→55→59=1
(f)

**Fig. 9** Simulation result in environment 2. **a** Position (*blue circle*) of frontier node and ID (number) on the resulting map **b**. Frontier tree structure and distance matrix of edge DB **c**–**f**. Merged map after accidental loop closing, *blue circle*: positions of loop nodes, *red star*: explored adjacent nodes of loop nodes, *blue star*: unexplored adjacent node of loop nodes (color figure online)

merging process has two parts: (1) finding the loop sequence and (2) updating the map DB.

### 4.2.1 Loop sequence decision

When loop closing is induced from the child to the parent node, the loop sequence is clearly determined as $node_{parent} \rightarrow node_{\text{child}} \rightarrow node_{\text{parent}}$. All local information about the child node is recalculated with respect to the parent node.

Loop closing can also occur when the robot revisits a node by chance during exploration. Detecting this accidental loop closing is beyond the scope of this paper; here, we focus only on how to manage the proposed frontier tree when loop closing is detected.

The loop sequence can be acquired by using the $A^*$ search algorithm [26] together with the edge DB. Unlike the predicted edge DB ($edge^{-}_{i,j}$), the edge DB ($edge^{+}_{i,j}$) contains the actual connections between adjacent nodes, which were obtained according to the movement of the robot. The $A^*$ algorithm determines the shortest loop sequence from the revisited node to the previous node.

We can impose the loop constraint between edges according to this loop sequence as follows:

$$edge^{+}_{1,2} \oplus \cdots \oplus edge^{+}_{n-1,n} \oplus edge^{+}_{n,1} = 0 \tag{4}$$

where $node_1$ is the revisited node and $node_n$ is the previous node. The edge information between loop nodes can be corrected by using the loop constraint and the iterated method,

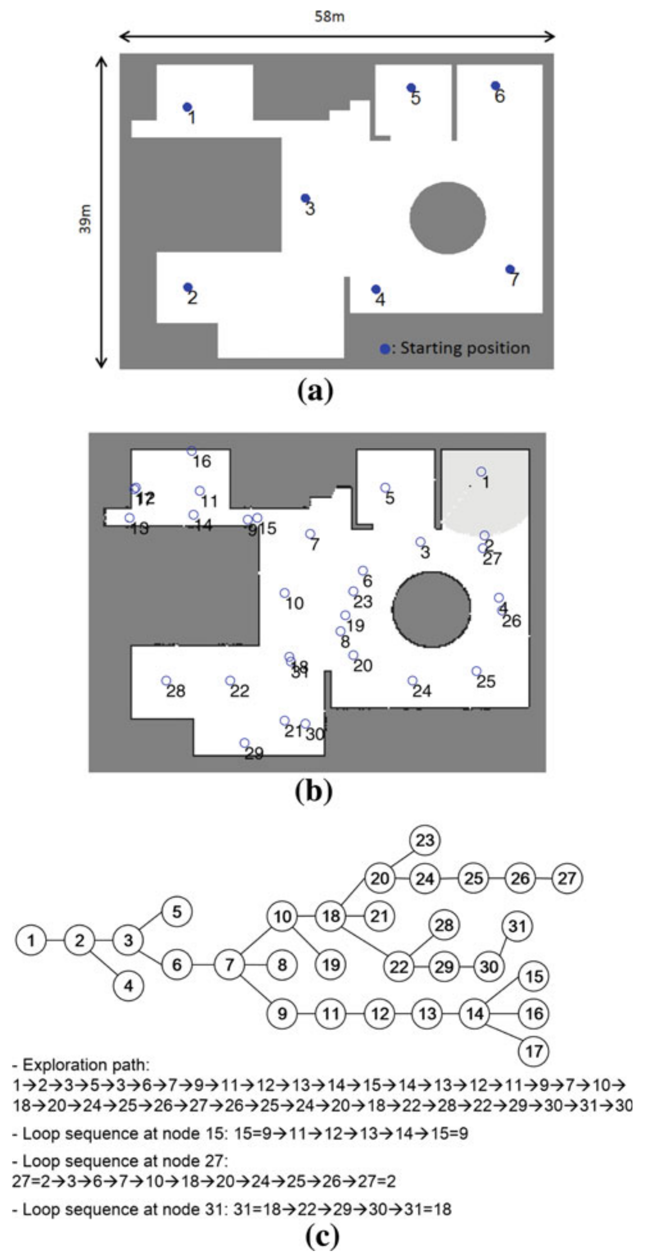

**Fig. 10** Simulation result in environment 3. **a** Environment 3 and the starting positions, indicated by the *blue dots*. **b**, Simulation result using starting position 6; the *blue circles* indicate frontier nodes, and the number shown is the node ID. **c** Frontier tree structure of **b** (color figure online)

such as iterated extended Kalman filter [14]. By using this constrained relative transformation, the corresponding local maps can be consistently merged. If the edge is updated once, it acts on later loop closing as a constraint with a zero uncertainty.

### 4.2.2 Map update after loop closing

The corresponding local grid maps are merged based on the corrected relative transformations through loop closing. Multiple local maps are updated onto the larger map, and each frontier node position on the local map must be recalculated in the new reference frame. For both the induced and the accidental loop closing, the new reference position is the current node position. In other words, the parent node and the revisited node become the new reference node. Algorithm 1 describes the details of the tree DB update. From lines 1 to 3, the position of each loop node is calculated with respect to the reference node, $node_1$. We obtain $pose_{m_i,node_i}$ from the map DB, and the new position of each local map is obtained using the procedure in lines 4–7. The local maps can be merged based on these relative transformations. Eventually, all local information assigned to loop nodes will be unified into the reference node (lines 10–18). Figure 7 shows an example of map DB updating based on child-to-parent induced loop closing.

**Algorithm 1** Tree DB update after loop closing

```
Input: loop sequence $node_1 \rightarrow \cdots \rightarrow node_n \rightarrow node_1$ ($node_1$ is the revisited node), edge DB $\left\{edge^+_{1,2}, edge^+_{2,3}, \ldots, edge^+_{n-1,n}, edge^+_{n,1}\right\}$, map DB $\left\{map_{m_1}, \ldots, map_{m_n}\right\}$
Output: updated node DB and map DB
1: for all loop $node_i$, except $node_1$ do
2:   Calculate the poses of nodes with respect to the $node_1$, $\left\{tr_{1,2}, tr_{1,3}, \ldots, tr_{1,n}\right\}$,
     $tr_{1,i} = edge^+_{1,2} \oplus \cdots \oplus edge^+_{i-1,i}$
3: end for
4: for all loop node $node_i$, except $node_1$ do
5:   calculate the relative transformations from $map_{m_1}$ to $map_{m_i}$.
6:   $tr_{m_1,m_i} = pose_{m_1,node_1} \oplus tr_{1,i} \oplus \left(\ominus pose_{m_i,node_i}\right)$.
7: end for
8: Merge local maps using $tr_{m_1,m_i}$
9: Update $map_{m_1}$, $gridmap_{m_1} \leftarrow gridmap_{merged}$
10: for all loop node $node_i$, except $node_1$ do
11:   get $map_{m_i}$ in map DB
12:   for all $node_k$, which share the $map_{m_i}$ do
13:     $id_{map} \leftarrow m_1$
14:     calculate the pose of $node_k$ in $map_{m_1}$, $pose_{m_1,node_k} = tr_{m_1,m_i} \oplus pose_{m_i,node_k}$
15:     push $k$ and $pose_{m_1,node_k}$ into $map_{m_1}$
16:     take $map_{m_i}$ off map DB.
17:   end for
18: end for
```

## 5 Simulation results

We simulated the local map-based exploration for the corridor environments shown in Figs. 8(a) and 9(a), and the hall environment in Fig. 10(a). The robot had a simulated 360° laser scanner, which was calculated using a ray-tracing algorithm to obtain local occupancy information.

### 5.1 Frontier tree structure

Figure 8(b) and (c) show the position of the frontier nodes and the tree structure in the first corridor environment. The

**Table 2** Simulation results at different starting positions in environment 1

| Starting position | 1 | | 2 | | 3 | | 4 | | 5 | | 6 | | 7 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Method | DFS | SF | DFS | SF | DFS | SF | DFS | SF | DFS | SF | DFS | SF | DFS | SF |
| # of measurements | 25 | 25 | 25 | 25 | 25 | 25 | 25 | 24 | 25 | 25 | 26 | 25 | 26 | 26 |
| Travel distance (cells) | 913 | 801 | 1,134 | 828 | 1,225 | 852 | 1,236 | 854 | 1,226 | 916 | 1,067 | 1,032 | 1,093 | 1,011 |
| # of nodes | 28 | – | 28 | – | 28 | – | 27 | – | 28 | – | 29 | – | 29 | – |
| Min. time (s) | 1.92 | 2.61 | 1.93 | 2.63 | 1.91 | 2.73 | 1.95 | 2.94 | 1.96 | 3.00 | 1.88 | 3.00 | 1.91 | 2.90 |
| Max. time (s) | 3.68 | 11.14 | 4.13 | 11.18 | 4.01 | 10.91 | 4.18 | 10.95 | 3.79 | 21.20 | 3.32 | 14.85 | 3.30 | 13.99 |
| Avg. time (s) | 2.51 | 5.38 | 2.71 | 5.44 | 2.73 | 5.36 | 2.72 | 5.44 | 2.64 | 9.49 | 2.59 | 8.00 | 2.59 | 6.99 |
| Starting position | 8 | | 9 | | 10 | | 11 | | 12 | | 13 | | 14 | |
| Method | DFS | SF | DFS | SF | DFS | SF | DFS | SF | DFS | SF | DFS | SF | DFS | SF |
| # of measurements | 25 | 25 | 25 | 26 | 25 | 24 | 25 | 24 | 25 | 26 | 25 | 25 | 24 | 24 |
| Travel distance (cells) | 1,134 | 833 | 1,067 | 945 | 1,038 | 1,078 | 1,010 | 824 | 1,069 | 999 | 1,165 | 1,027 | 1,258 | 765 |
| # of nodes | 28 | – | 27 | – | 27 | – | 27 | – | 27 | – | 29 | – | 27 | – |
| Min. time (s) | 1.72 | 2.99 | 1.72 | 3.03 | 1.74 | 2.75 | 1.94 | 2.58 | 1.76 | 2.77 | 1.91 | 3.14 | 1.99 | 2.77 |
| Max. time (s) | 4.66 | 20.79 | 4.40 | 16.02 | 4.15 | 24.68 | 3.84 | 13.96 | 3.78 | 12.84 | 4.35 | 15.01 | 4.14 | 9.70 |
| Avg. time (s) | 2.52 | 10.48 | 2.50 | 8.13 | 2.51 | 11.50 | 2.56 | 6.52 | 2.51 | 6.98 | 2.61 | 7.43 | 2.68 | 5.69 |

**Table 3** Simulation results at different starting positions in environment 3

| Starting position | 1 | | 2 | | 3 | | 4 | | 5 | | 6 | | 7 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Method | DFS | SF | DFS | SF | DFS | SF | DFS | SF | DFS | SF | DFS | SF | DFS | SF |
| # of measurements | 29 | 29 | 26 | 25 | 32 | 25 | 25 | 28 | 31 | 26 | 21 | 25 | 28 | 27 |
| Travel distance (cells) | 2,014 | 1,306 | 1,688 | 1,142 | 2,123 | 1,233 | 1,447 | 1,257 | 2,249 | 1,166 | 1,261 | 1,192 | 2,029 | 1,329 |
| # of nodes | 47 | – | 42 | – | 46 | – | 40 | – | 49 | – | 31 | – | 48 | – |
| Min. time (s) | 2.60 | 2.44 | 3.53 | 4.54 | 3.62 | 4.17 | 4.62 | 7.87 | 3.29 | 5.99 | 4.00 | 5.12 | 3.00 | 6.45 |
| Max. time (s) | 22.92 | 78.36 | 15.97 | 158.81 | 11.27 | 96.55 | 19.98 | 77.58 | 14.54 | 109.76 | 16.31 | 92.20 | 16.41 | 113.67 |
| Avg. time (s) | 4.78 | 29.54 | 6.87 | 53.41 | 5.55 | 35.01 | 7.89 | 29.49 | 5.97 | 37.42 | 7.51 | 36.27 | 5.48 | 35.92 |

position of the first node was the starting position. The environment included 28 frontier nodes; these well-distributed nodes represented the entire environment. The robot moved between the nodes according to the DFS algorithm. Figure 8(c) presents the resulting exploration path: child-to-parent loop closing was induced at the red numbered node. Accidental loop closing occurred at node 27, when node 5 was revisited, and this loop sequence was calculated using the $A^*$ algorithm.

Figure 9 presents simulated data for the second corridor environment, which was more complex than the first. In total, 59 nodes were registered to the frontier tree, as shown in Fig. 9(b). Whenever there was no new frontier region at the current node, child-to-parent loop closing was induced. Accidental loop closing occurred at nodes 28, 48, 58, and 59. In Fig. 9(c)–(f), the updated map, constructed according to the loop node, can be seen. In each loop-closing event, the tree DB was successfully updated, so exploration was completed for the entire environment.

We also simulate the proposed algorithm in a hall environment, as shown in Fig. 10. Figure 10(b) and (c) are the distributed frontier nodes and their tree structure using starting position 6 in Fig. 10(a). Accidental loop closing occurred at nodes 15, 27, and 31.

## 5.2 Computational time

To compare the computational expense of the local map-based method with the conventional approach, we also carried out simulations using the shortest frontier-based (SF) algorithm. In the SF algorithm, the robot merges local sensor data with the single global map at every decision step and finds the shortest path to the frontier position. It does not include a loop-inducing strategy or map management methods; near frontiers are segmented into one frontier cell to reduce the computational time.

Table 2 lists simulated data for different starting positions in the environment 1 (corridor). Here, the robot commenced

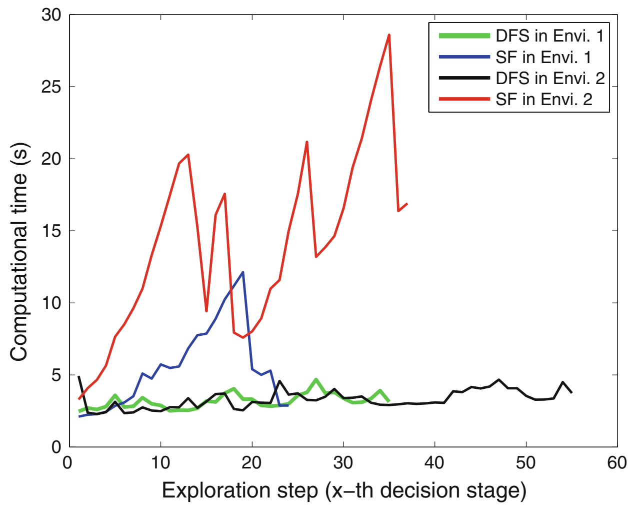

**Fig. 11** Computational time for each decision step in environment 1 and 2

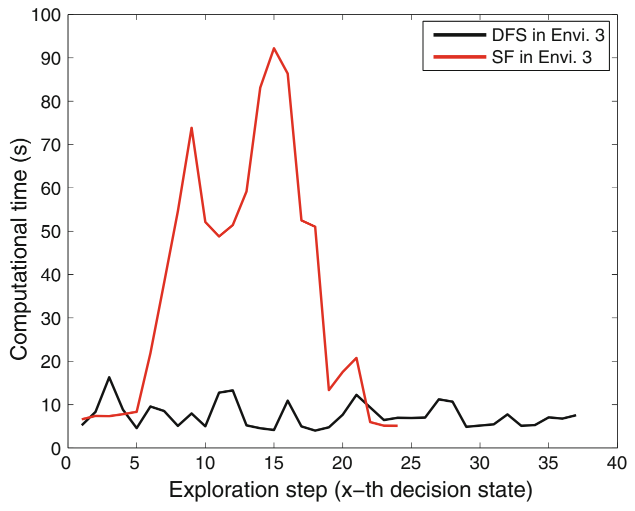

**Fig. 12** Computational time for each decision step in environment 3

exploration at 14 different starting positions. The simulation results in the environment 3 (hall) are in Table 3 and the robot started exploration at seven different positions.

We restricted the movement between siblings, so the robot could move from the current node to the child node or the parent node. This can induce loop closing and lead to map updating, but it also means that the total travel path of DFS exploration can be longer than the results of SF exploration, because the robot always moves to a new place in SF exploration. However, the computational expense of DFS exploration is significantly more stable than that of SF exploration. As shown in Table 2, the longest computation time required for the DFS method was 4.66 s; in contrast, the longest computation time required for the SF method was 24.68 s. Figure 11 presents the computational time for each decision step in environment 1 and 2, and Fig. 12 shows the results in environment 3; it clearly illustrates the computational advantage of the local map-based exploration. The time that elapsed using the DFS method (multiple local map-based) was approximately constant at every decision, while the SF algorithm (single global map-based) took longer and longer as the exploration proceeded. To find the shortest frontier with a larger map requires more computational time. In the local map-based method, the managed tree DB can reduce the computational burden. In a more complex environment, this merit of the local map-based exploration becomes more significant (see the black and red lines in Fig. 11 and 12 ). A similar number of measurements were used for the completion of mapping. This means that although only local information is used in the proposed method, efficiency for completing a map can be guaranteed.

## 6 Conclusions

We have proposed a local map-based exploration algorithm and analyzed its efficiency by comparing it with the single global map-based method. By adding frontier nodes to the tree structure and moving along this tree, a robot can expand the knowledge of the environment. Because segmented frontier nodes are distributed over the entire environment and the tree structure is well defined by the relative transformation, the environment can be efficiently represented by this tree structure and exploration can be completed using only local information. Stable and predictable computational cost can be expected using the proposed method. To determine the next target node, adjacent node information can be used efficiently even in a large environment, as the computational cost at each decision step does not increase during the exploration. Additionally, an exploration path can be planned for inducing loop closing that can be effective in reducing the position uncertainty in SLAM.

## References

1. Choset H, Nagatani K (2001) Topological simultaneous localization and mapping (SLAM): toward exact localization without explicit localization. IEEE Trans Robot Autom 17(2):125–137
2. Dissanayake MG, Newman P, Clark S, Durrant-Whyte HF, Csorba M (2001) A solution to the simultaneous localization and map building (SLAM) problem. IEEE Trans Robot Autom 17(3):229–241
3. Montemerlo M, Thrun S, Koller D, Wegbreit B (2003) Fastslam 2.0: An improved particle filtering algorithm for simultaneous localization and mapping that provably converges. In: International Joint Conference on Artificial Intelligence, vol 18, pp 1151–1156
4. Moorehead SJ, Simmons R, Whittaker WL (2001) Autonomous exploration using multiple sources of information. In: IEEE International Conference on Robotics and Automation, vol 3, pp 3098–3103

5. Schultz AC, Adams W, Yamauchi B (1999) Integrating exploration, localization, navigation and planning with a common representation. Auton Robot 6(3):293–308
6. Yamauchi B (1997) A frontier-based approach for autonomous exploration. In: IEEE International Symposium on Computational Intelligence in Robotics and Automation, pp 146–151
7. Stachniss C, Burgard W (2003) Exploring unknown environments with mobile robots using coverage maps. In: International Joint Conference on Artificial Intelligence, vol 18, pp 1127–1134
8. Bourgault F, Makarenko AA, Williams SB, Grocholsky B, Durrant-Whyte HF (2002) Information based adaptive robotic exploration. In: IEEE/RSJ International Conference on Intelligent Robots and Systems, pp 540–545
9. Makarenko AA, Williams SB, Bourgault F, Durrant-Whyte HF (2002) An experiment in integrated exploration. In: IEEE/RSJ International Conference on Intelligent Robots and Systems, pp 534–539
10. Stachniss C, Grisetti G, Burgard W (2005) Information gain-based exploration using rao-blackwellized particle filters. In: Robotics: science and systems (RSS), pp 65–72
11. Sim R, Roy N (2005) Global a-optimal robot exploration in slam. In: IEEE International Conference on Robotics and Automation, pp 661–666
12. Piniés P, Tardós JD (2008) Large-scale slam building conditionally independent local maps: application to monocular vision. IEEE Trans Robot 24(5):1094–1106
13. Tardós JD, Neira J, Newman PM, Leonard JJ (2002) Robust mapping and localization in indoor environments using sonar data. Int J Robot Res 21(4):311–330
14. Estrada C, Neira J, Tardós JD (2005) Hierarchical slam: real-time accurate mapping of large environments. IEEE Trans Robot 21(4):588–596
15. Ahn S, Choi J, Doh NL, Chung WK (2008) A practical approach for EKF-slam in an indoor environment: fusing ultrasonic sensors and stereo camera. Auton Robot 24(3):315–335
16. Blanco JL, Fernández-Madrigal JA, Gonzalez J (2007) A new approach for large-scale localization and mapping: hybrid metric-topological slam. In: IEEE International Conference on Robotics and Automation, pp 2061–2067
17. Paul R, Newman P (2010) Fab-map 3D: topological mapping with spatial and visual appearance. In: IEEE International Conference on Robotics and Automation, pp 2649–2656
18. Cummins M, Newman P (2009) Highly scalable appearance-only slam-fab-map 2.0. In: Proceedings of Robotics: Science and Systems (RSS), vol 5
19. Maddern W, Milford M, Wyeth G (2011) Continuous appearance-based trajectory SLAM. In: IEEE International Conference on Robotics and Automation, pp 3595–3600
20. Werner F, Sitte J, Maire F (2012) Topological map induction using neighbourhood information of places. Auton Robot 32(4):405–418
21. Fox D, Burgard W, Thrun S (1998) Active Markov localization for mobile robots. Robot Auton Syst 25(3):195–207
22. Kümmerle R, Triebel R, Pfaff P, Burgard W (2008) Monte carlo localization in outdoor terrains using multilevel surface maps. J Field Robot 25(6–7):346–359
23. Khalvati K, Mackworth AK (2012) Active robot localization with macro actions. In: IEEE/RSJ International Conference on Intelligent Robots and Systems, pp 187–193
24. Yi C, Suh IH, Lim GH, Choi BU (2009) Active-semantic localization with a single consumer-grade camera. In: IEEE International Conference on Systems, Man and Cybernetics, pp 2161–2166
25. Bailey T, Nieto J, Guivant J, Stevens M, Nebot E (2006) Consistency of the EKF-slam algorithm. In: IEEE/RSJ International Conference on Intelligent Robots and Systems, pp 3562–3568
26. Dechter R, Pearl J (1985) Generalized best-first search strategies and the optimality of A*. JACM 32(3):505–536