IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received May 10, 2018, accepted June 21, 2018, date of publication July 5, 2018, date of current version July 30, 2018.

Digital Object Identifier 10.1109/ACCESS.2018.2853146

Scalable Coverage Path Planning for Cleaning Robots Using Rectangular Map Decomposition on Large Environments

XU MIAO[1], JAESUNG LEE[2], AND BO-YEONG KANG[1], (Member, IEEE)
[1]School of Mechanical Engineering, Kyungpook National University, Daegu, South Korea
[2]School of Computer Science and Engineering, Chung-Ang University, Seoul, South Korea

Corresponding author: Bo-Yeong Kang (kby09@knu.ac.kr)

This work was supported by the National Research Foundation of Korea through the Korean Government under Grant NRF-2016R1A2B4006873.

ABSTRACT The goal of coverage path planning is to create a path that covers the entire free space in a given environment. Coverage path planning is the most important component of cleaning robot technology, because it determines the cleaning robot's movement. When the environment covered by a cleaning robot is extremely large and contains many obstacles, the computation for coverage path planning can be complicated. This can result in significant degradation of the execution time for coverage path planning. Not many studies have focused on the scalability of coverage path planning methods. In this paper, we propose a scalable coverage path planning method based on rectangular map decomposition. The experimental results demonstrate that the proposed method reduces the execution time for coverage path planning up to 14 times when compared with conventional methods.

INDEX TERMS Cleaning robot, robot path planning, coverage path planning, scalability, computational efficiency.

I. INTRODUCTION

For the past two decades, coverage path planning (CPP) has been regarded as an important research issue in the intelligent robot community for popular applications that include demining robots [1], lawn mowers [2], and harvesting robots [3]. The goal of CPP is to create a path that covers the entire free space in a given known or unknown environment [4], [5]. Although the goal of covering the entire free space can be achieved by employing the well-known random walk process [6] because the random walk-based direction control suffers from path repetition [4], [5], many researchers have dedicated efforts to developing more efficient and effective CPP methods. For example, a cleaning robot, which is a popular robotic application, can extend cleaning time unnecessarily and waste its battery by cleaning the same locations multiple times if unplanned direction control is involved. To overcome this drawback, many strategic CPP methods have been proposed for indoor/outdoor cleaning scenarios [4], [7], [8].

A basic CPP procedure can be described as follows. First, the cleaning robot attempts to identify the boundary of the given environment by following the outermost wall (or obstacle). After returning to its initial position, the robot creates a map representing the environment. Next, the robot begins cleaning map using a coverage path such as spiral [9] or zigzag [10]. At the cleaning process, the robot can be surrounded by previously cleaned space; the current position of the robot at this moment is known as a blind alley. To escape the blind alley situation, a path planning method such as the Dijkstra algorithm [11] or A* [12], [13] is invoked to identify the robot's next cleaning position. Finally, these tasks are repeated until there is no uncleaned space. The majority of CPP methods use strategies based on grid information gathered in the process of cleaning [9], [14]–[20]. However, grid-based strategies require significant computational cost, which increases the total execution time [21], [22].

Recently, demands have emerged from industrial communities for efficient and effective cleaning robots that can cover significantly larger environments such as libraries, warehouses, stadiums, and airports [23]. When conventional CPP methods are used in these large-spaced environments, the results are unsatisfactory because of the large number of grids used in the process. To efficiently address these, the CPP method to control the cleaning robots must be scalable to the size of any given environment. However, to the best of our knowledge, there remains a lack of studies focused on the scalability of CPP methods. In this paper, we propose a novel CPP method scalable to large environments. The experimental results on large environments confirm that the proposed method completes the CPP task considerably more quickly than conventional methods.

II. RELATED WORK

CPP is one of the most important control components influencing the efficiency and effectiveness of a cleaning robot because it schedules the entire cleaning path. In the majority of studies, the initial position of a cleaning robot is assumed to be near the boundary [24] because of practical constraints such as power supply. In general, spiral shape-based CPP is implemented on a grid-based map that is obtained by gridizing the sensed environment [9], [15], [25]. This is one of the most widely considered map-encoding schemes in cleaning robot studies owing to its easy implementation and rapid situation awareness of a given map [5]. After the robot identifies the boundary of the given environment, the robot divides the environment into specific grid shapes such as disks [26], rectangles [27], triangles [28], or hexagons [29]. Typically, the size of each grid is set to the same size as the cleaning robot to avoid unintended uncleaned space. Based on the size and shape of the grid, the map is created and the CPP process is initiated using the obtained map.

In the earliest studies of CPP, the map was obtained by gridizing a known environment. For example, Zelinsky *et al.* [30] illustrated a simple CPP method based on transformed distance values that are primarily assigned to all the grids before the actual CPP is executed. However, this also decreases the availability of the cleaning robot because of its impractical assumption that the environment is known. In this study, we assume that the environment is unknown, and therefore the robot must explore and clean the environment simultaneously. To address an unknown environment for which the cleaning robot cannot initially obtain a map, Gabriely and Rimon [15], [31], [32] proposed spanning tree-based coverage (STC) path planning. This method gridizes the environment with grids twice the size of the cleaning robot to obtain the map and then sets two virtual walls, left/right and top/bottom, from the center of each grid. Based on these virtual walls, the robot creates a spanning tree that connects all the uncleaned grids. In the case where the robot encounters a blind alley situation, the method executes CPP recursively. Thus, when the given map is extremely large or there are many obstacles on the environment, CPP can suffer from an impractically high computational cost and memory consumption due to an excessive number of recursive calls.

Another well-known CPP method is the backtracking spiral algorithm (BSA) based on the STC method [9], [25], where the final coverage path is created by following virtual walls. This method performs CPP using backtracking grids that are composed of potential alternative paths and invoking a simple breadth-first search (BFS) algorithm rather than using recursive CPP. Choi *et al.* [14] proposed linked spiral path (LSP), which employs the concept of backtracking grids and the constrained inverse distance transform (CIDT) method with distance wave. This strategy was also used in the works of Baek *et al.* [33] and Lee *et al.* [8], [27]. However, if there are excessive backtracking grids due to a large environment with obstacles, these algorithms eventually consume unacceptable computational resources when using the BFS and distance transform algorithm. Similarly, Viet *et al.* [16] proposed the BA* method using the well-known A* search algorithm. In this case, if a large number of grids are employed to represent a sizable environment, the computational efficiency of the BA* method is significantly degraded because the A* search algorithm must consider a large number of grids to determine a safe path to the next uncleaned grid [22].

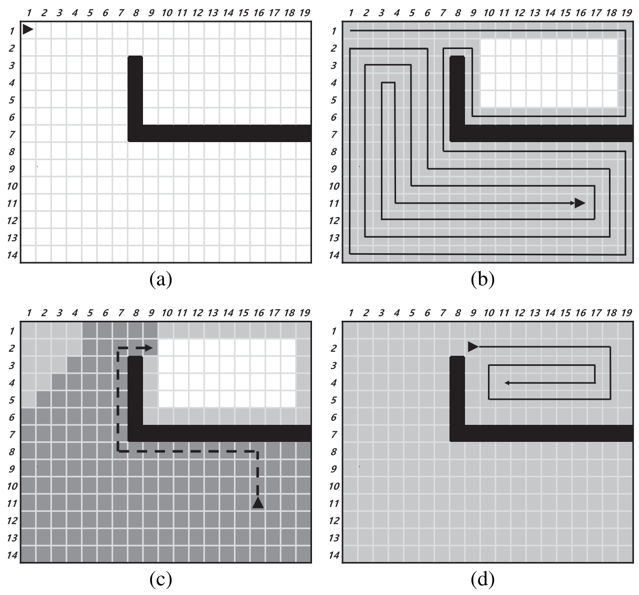


FIGURE 1. Example of CPP using BSA method. (a) Map with L-type obstacles. (b) Encounter with blind alley situation. (c) Creation of path through BFS. (d) Cleaning uncleaned space.

The process of CPP used by the BSA method, which is among the conventional CPP methods, is illustrated in Figure 1. In Figure 1(a), a map with L-type obstacles is displayed. The cleaning robot is marked by a triangle; the black grid represents the obstacles. Figure 1(b) displays the cleaning process of the given map by a cleaning robot using the BSA method. The BSA method uses the spiral coverage path indicated by the solid lines and the cleaning robot follows the spiral coverage path and cleans. It is confirmed that if the cleaning robot uses the spiral coverage path, it cannot clean the entire space in one process owing to the obstacles. The gray grid represents the cleaned spaces and the white grid indicates the spaces that have not been cleaned. If the cleaning robot moves to the position on the grid marked by coordinates (11, 16), it is surrounded by previously cleaned space, hence, a blind alley occurs. In this case, the cleaning robot moves to the next grid that has not been cleaned, and continues cleaning.

Figure 1(c) indicates the process of moving to an uncleaned grid from a blind alley using BFS path planning from the BSA method. Using the BFS algorithm, the cleaning robot at the grid coordinates (11, 16) explores whether there is a grid that has not been cleaned among the four surrounding grids in the horizontal and vertical directions (10, 16), (11, 15), (12, 16), and (11, 17). If among the four surrounding grids there is not a grid that has not been cleaned, it continually explores grids in the horizontal and vertical directions. For example, for the grids, (9, 16), (10, 15), and (10, 17) that surround grid (10, 16), the grids in the horizontal and vertical directions that have not been explored, are addressed. This process is repeated on the remainder of the surrounding grids until a grid that has not been cleaned is identified. The explored range is marked by a dark gray color. When the cleaning robot goes to (2, 9) and determines that it has not yet been cleaned, it marks the grid (2, 9) as a point of entry to an uncleaned region. Using the explored grids, the created path's route is marked by the dotted lines.

Figure 1(d) indicates the process of cleaning an uncleaned space as the cleaning robot moves to grid (2, 9) and again follows a spiral coverage path. As indicated in Figure 1, if there are obstacles on the map, a blind alley can be created. Each time a blind alley is encountered, the next step of the robot is to determine a path to a location that requires cleaning. When the BFS algorithm is used in a map with L-type obstacles, more than half of the grids can require exploration. In summary, conventional CPP methods are difficult to apply on large maps because they use ineffective path planning strategies. To solve this problem, this study analyzes the pseudocode of conventional CPP methods including BSA, LSP, and BA*.

Algorithm 1 represents the pseudocode for the BSA. In the BSA, the cleaning robot first cleans the map using a spiral path (line 3). The robot must check for the occurrence of blind alleys at every move because it is cleaning an unknown map (line 4). If a blind alley is detected, the robot uses the BFS method to plan a path that leads to the next uncleaned space (line 6). If no blind alleys are detected, the robot continues to clean along the spiral path (line 3). Figure 2(a) displays a map with a blind alley that can be divided into cleaned and uncleaned areas with the BSA pseudocode. The white and gray areas in Figure 2(a) represent the uncleaned and cleaned areas, respectively. Figure 2(b) indicates that when the robot detects a blind alley, it uses BFS to travel to an uncleaned area. The red spaces represent the areas explored when using BFS.

Algorithm 2 represents the pseudocode for the LSP method. First, the cleaning robot follows the walls and uses information on the outline of the map to match the coordinates of the robot and the map (line 2). The robot checks for the presence of blind alleys at every move, as they can occur on an unknown map (line 4). If the robot detects a blind alley, it uses the CIDT method to plan a path that leads to an uncleaned space (line 6). If no blind alleys are detected, the robot continues to clean the map by following a spiral path (line 12). Figure 2(c) displays a map with a blind alley that can be divided into cleaned and uncleaned areas with the LSP pseudocode. The white and gray areas in Figure 2(c) represent the uncleaned and cleaned areas, respectively. The red areas in Figure 2(d) represent the area explored by the robot using CIDT to travel to an uncleaned area.

Algorithm 3 represents the pseudocode of the BA* method. In BA*, the cleaning robot first cleans the map following zigzag paths (line 3). The robot must check for the possible occurrence of blind alleys at every move (line 4). If a blind alley is detected, the robot uses the A* algorithm to plan

Algorithm 1 BSA Method
```
1:  Input: current          {cureent: cleaning robot's current position}
2:  while true do
3:    $grid_{candidate}$ ← Execute map cleaning use spiral paths;
      {$grid_{candidate}$: back-tracking grids}
4:    if blind alley occurs in current then
5:      if $grid_{candidate} \neq \phi$ then
6:        path ← Execute BFS;   {grid map-based search path}
7:        current ← Follows the path go to uncleaned space;
8:      else
9:        Break;
10:     end if
11:   end if
12: end while
```

Algorithm 2 LSP Method
```
1:  Input: current          {cureent: cleaning robot's current position}
2:  $grid_{candidate}$ ← Execute wall following and map coordinate assignment;   {$grid_{candidate}$: back-tracking grids}
3:  while true do
4:    if blind alley occurs in current then
5:      if $grid_{candidate} \neq \phi$ then
6:        path ← Execute CIDT; {grid map-based search path}
7:        current ← Follows the path go to uncleaned space;
8:      else
9:        Break;
10:     end if
11:   else
12:     $grid_{candidate}$ ← Execute map cleaning use spiral paths;
13:   end if
14: end while
```

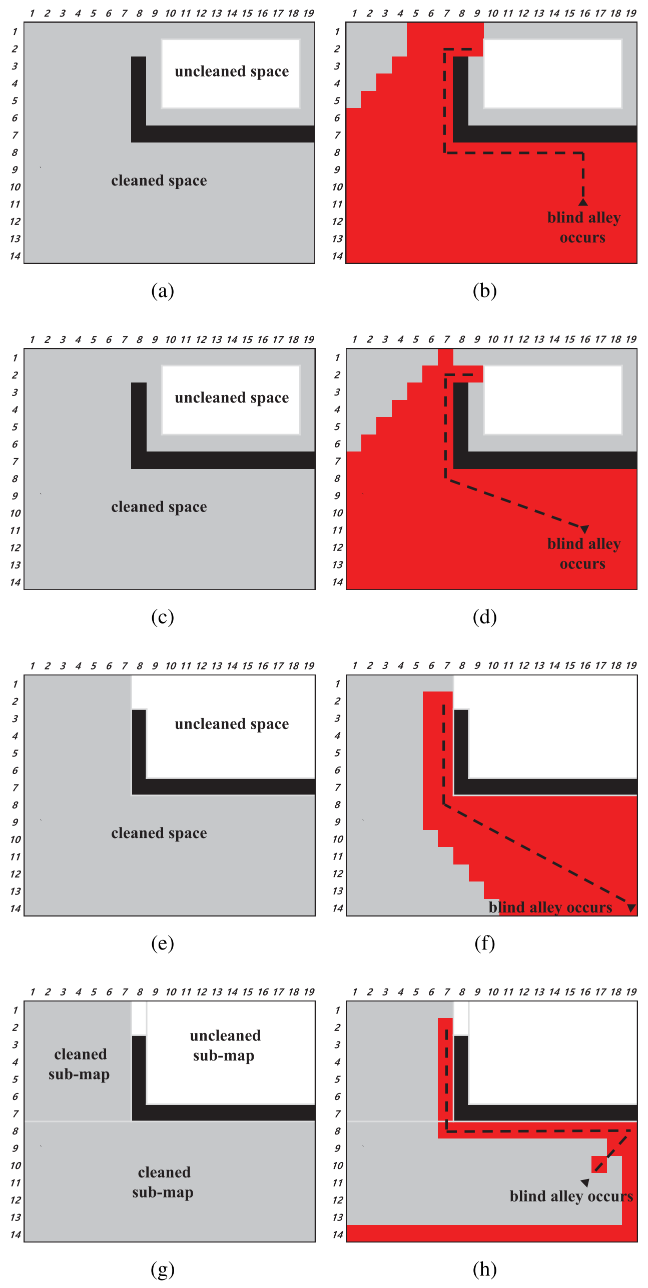


FIGURE 2. **Example of explored areas for path planning with the BSA, LSP, BA* and proposed method. (a) Cleaned and un-cleaned areas in BSA. (b) Path planning using BFS in BSA. (c) Cleaned and un-cleaned areas in LSP. (d) Path planning using CIDT in LSP. (e) Cleaned and un-cleaned areas in BA*. (f) Path planning using A* in BA*. (g) Cleaned and un-cleaned sub-maps in the proposed method. (h) Path planning using BFS in the proposed method.**

Algorithm 3 BA* Method

```
1:  Input: current        {cureent: cleaning robot's current position}
2:  while true do
3:      $grid_{candidate}$ ← Execute map cleanning use zigzag paths;    {$grid_{candidate}$: back-tracking grids }
4:      if blind alley occurs in current then
5:          if $grid_{candidate} \neq \phi$ then
6:              path ←Execute A*;    {grid map-based search path}
7:              path ← Smoothed path;
8:              current ← Follows the path go to uncleaned space;
9:          else
10:             Break;
11:         end if
12:     end if
13: end while
```

a path that leads to an uncleaned space (line 6). With the BA* method, the path planned using the A* algorithm is revised to obtain a shorter path (line 7). If no blind alleys are detected, the robot continues to clean the map following the zigzag paths (line 3). Figure 2(e) displays a map with a blind alley that can be divided into cleaned and uncleaned areas. Figure 2(f) indicates that when the robot detects a blind alley, it uses the A* algorithm to travel to an uncleaned area. The red areas represent the areas explored using the A* algorithm. With conventional CPP algorithms, a large amount of space must be explored on a grid map when a blind alley occurs while planning coverage paths on an unknown map. However, with the proposed method, only a relatively small amount of space is required for exploration, as indicated in Figure 2(h), by utilizing the edges that connect the sub-maps when a blind alley occurs. This is because rectangular sub-maps can be generated with the proposed method, as indicated in Figure 2(g). Thus, this study confirms that the proposed method demonstrates high efficiency because it utilizes the connective edges between sub-maps instead of all the grids on a map.

In summary, blind alleys can be detected while a cleaning robot is cleaning a map using CPP; in this case, the robot must travel to uncleaned areas from cleaned areas. Conventional CPP methods require high computational cost in this process because they use a large number of grids on the map. However, the method proposed in this study utilizes the connective edges between rectangular sub-maps rather than all the grids on a map to determine paths from a blind alley. This enhances the performance of CPP by effectively managing problems related to path finding. This paper proposes a new CPP method that provides an effective application of the cleaning robot to a large map.

The proposed method divides the map into rectangles and path planning is based on the edges created in the map decomposition. That is, a planning method based on a map decomposition divided into rectangles and edges created in a map decomposition is proposed. A large map can be divided into rectangular sub-maps and edges that connect the divided sub-maps created. In the decomposed sub-map, when blind alleys are encountered, the cleaning robot calculates a path to the next sub-map a distance away that must be cleaned. In this case, instead of high-calculation path planning that considers all of the grids and possible edges, path planning based on the created edges is used in the map decomposition process and the path's exploration range is dramatically reduced.

III. PROPOSED METHOD

Figure 3 displays the proposed CPP method flowchart based on rectangular map decomposition that can be applied to a large map. The stages are as follows: map exploration, map decomposition, sub-map cleaning, and sub-map selection. During the map exploration process, where the cleaning robot explores unknown maps, the robot searches for walls and obstacles; as indicated in Figure 3(a), it collects information on conventional corners and edges. Next, map decomposition uses the collected corners, and divides the map into sub-maps as indicated in Figure 3(b). For map cleaning, as displayed in Figure 3(c), using the map divided into sub-maps, the cleaning robot follows and cleans the proposed spiral coverage path based on the edges. Then, as illustrated in Figure 3(d), the edge information is used. Another sub-map is selected and a path to move to the selected sub-map is created.

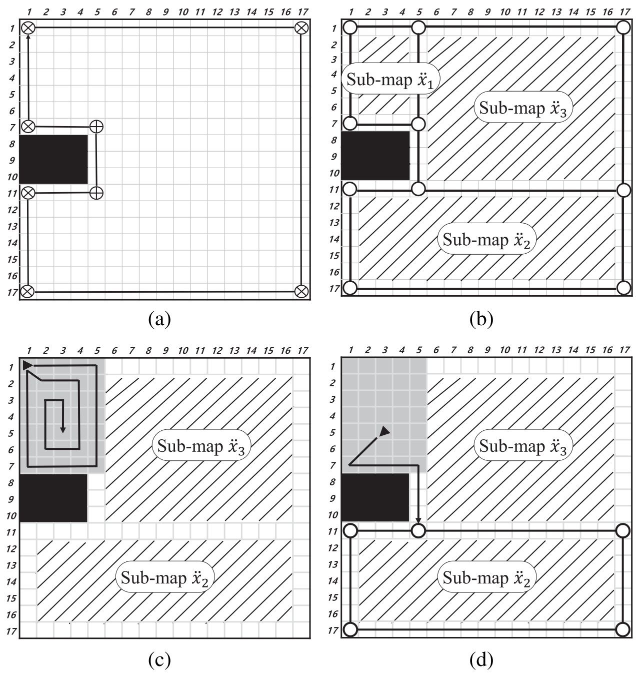


FIGURE 3. **Proposed CPP flowchart based on rectangular map decomposition. (a) Map exploration. (b) Map decomposition. (c) Sub-map cleaning. (d) Selection of next sub-map.**

Figure 4 displays the detailed algorithm for map exploration, map decomposition, sub-map cleaning, and sub-map selection. The cleaning robot follows the walls and collects information on the corners and edges of the unknown map. The map corners can be classified as either concave or convex. Map decomposition or sub-map cleaning is selected based on the classification of the corner. If a corner is not convex, then perform sub-map cleaning, else perform map decomposition. After performing either map decomposition or map cleaning, a sub-map is added or deleted to/from the remainder of the sub-maps. If a sub-map already exists in the set of sub-maps, the next sub-map to be cleaned is selected, and the map is explored. This process is repeated until there are no remaining sub-maps among the set of sub-maps and cleaning of the entire map is complete. Then, the proposed CPP method terminates. The procedural steps of the proposed method with additional detail are described in Algorithm 4.

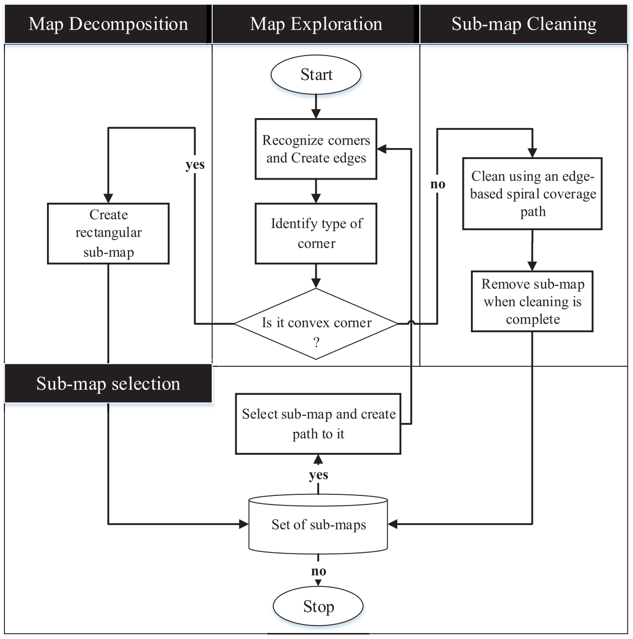

Map Decomposition
Map Exploration
Sub-map Cleaning
Start
Recognize corners and Create edges
Identify type of corner
Is it convex corner ?
yes
no
Create rectangular sub-map
Clean using an edge-based spiral coverage path
Remove sub-map when cleaning is complete
Sub-map selection
Select sub-map and create path to it
yes
Set of sub-maps
no
Stop

FIGURE 4. **Summary of proposed CPP method.**

Algorithm 4 Proposed Method

```
1: Initialize:
     1) The size of the grid is set according to the size of the
        robot which is 351.5 mm in diameter and the robot
        cleans one grid at a time as it moves.
     2) follows the walls and get $current$, $concave$, $convex$,
        $boundary$ of the unknown map {$cureent$: cleaning
        robot's current position}
2: if $convex = \{\phi\}$ then
3:    $convex \leftarrow$ Execute Submap cleanning;
4: end if
5: if $convex \neq \{\phi\}$ then
6:    Execute Map decomposition;
7:    while set of sub-map $\neq \{\phi\}$ do
8:       $convex \leftarrow$ Execute Submap cleanning;
9:       if $convex \neq \{\phi\}$ then
10:         Execute Map decomposition;
11:      else
12:         current sub-map is deleted from set of sub-map;
13:      end if
14:      Execute Submap selection;
15:   end while
16: end if
```

A. MAP EXPLORATION AND RECTANGULAR SUB-MAP DECOMPOSITION

In this paper, we assume that infrared (IR) sensors are used in the cleaning robot to identify its surroundings in unknown maps. This indicates that the robot can detect the presence of obstacles in its surroundings (on the front, right, and left sides) while traveling. Through the IR sensors, obstacles are detected and information is collected regarding the corners in the map [33], [34]. The cleaning robot explores the map by moving along the walls. This study includes two cases of map exploration. (A) Exploration of the outline of a map or an obstacle. In essence, the robot navigates around the boundaries of the map or obstacle and explores their outline by moving along the walls. (B) Exploration of sub-maps. The robot can encounter obstacles on a sub-map generated through map decomposition. To explore the presence of such obstacles, the robot searches for obstacles while it moves along the boundaries of the sub-map. If the robot detects an obstacle, it begins exploring the outline of the obstacle. The robot can explore unknown maps and gather information on existing corners with the proposed map exploration method. Among the collected corners, convex corners are used, and the process of map decomposition initiates.

Figure 5 illustrates the process of exploring walls and obstacles and identifying convex and concave corners on an unknown map. Figure 5(a) depicts a 17 × 17-sized unknown map. The gray area represents an unexplored obstacle and the triangle represents the cleaning robot. As illustrated in Figure 5(b), the robot explores the map by following the walls. The arrows indicate the direction of the robot's path. The gray area is indicated in black after the robot identifies an obstacle through map exploration. Figure 5(c) indicates that as the cleaning robot follows the walls, it collects the corners and boundary edges marked as map boundaries. A set of eight corners are collected. The i^{th} corner is defined as $\{l_i | i = 1, 2, ..., 8\}$. The boundary edges consist of two corners and are defined as $\langle l_i, l_{i+1}\rangle$. This indicates that the edges are non-directional edges, which are represented with solid lines. Next, using the grids on the boundary edge and the grids surrounding the corners, the type of corner is determined.

Figure 5(d)–(e) uses two boundary edges $\langle l_4, l_5\rangle$ and $\langle l_5, l_6\rangle$ and three grids l_4, l_5, and l_6. It displays the process of determining the type of grid l_5. Figure 5(d) indicates the grids including the boundary edges $\langle l_4, l_5\rangle$ and $\langle l_5, l_6\rangle$ connected to grid l_5 as ⊠. The grids that comprise the dark gray colored region mark the grids surrounding l_5 in the horizontal and vertical direction. First, among the grids surrounding l_5, grids included along the boundary edges $\langle l_4, l_5\rangle$ and $\langle l_5, l_6\rangle$ are selected using (1).

$$g_1 = N(l_i) \cap \langle l_{i-1}, l_i\rangle$$
$$g_2 = N(l_i) \cap \langle l_i, l_{i+1}\rangle \qquad (1)$$

$N(l_i)$ defines the corner l_i's surrounding grids and g_1 and g_2 define the grids included among $N(l_i)$, $\langle l_{i-1}, l_i\rangle$, and $\langle l_i, l_{i+1}\rangle$. If at l_i, $i = 1$, then l_{i-1} is selected as l_8; if at l_i, $i = 8$, then l_{i+1} is selected as l_1. If the grids g_1 and g_2 selected using (1) are marked as coordinates, they are (12, 1) and (11, 2), respectively. Figure 5(e) uses (2); it then indicates that with the exception of grid l_i, which must be classified as either a concave or convex corner, grid g_3 is selected from the vertical and horizontal grids neighboring g_1 and g_2.

$$g_3 = N(g_1) \cap N(g_2) - l_i \qquad (2)$$

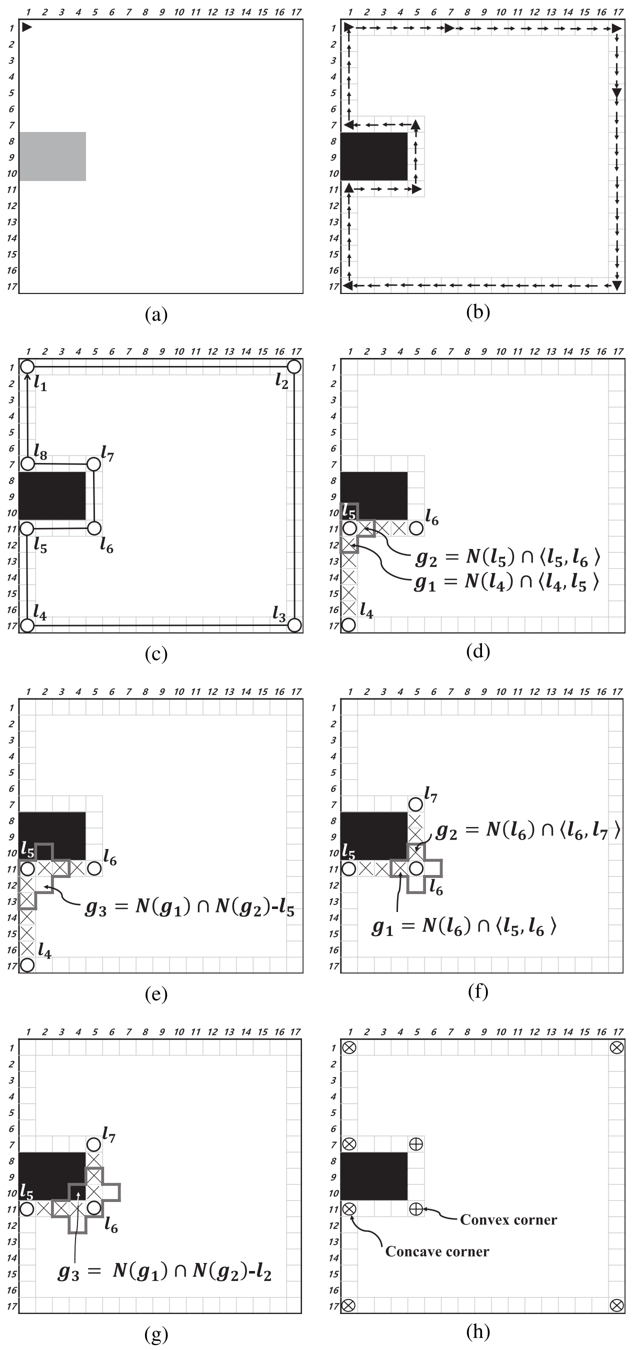

(a)
(b)
l_1
l_2
l_3
l_4
l_5
l_6
l_7
l_8
(c)
$g_2 = N(l_5) \cap \langle l_5, l_6\rangle$
$g_1 = N(l_4) \cap \langle l_4, l_5\rangle$
(d)
$g_3 = N(g_1) \cap N(g_2)$-l_5
(e)
$g_2 = N(l_6) \cap \langle l_6, l_7\rangle$
$g_1 = N(l_6) \cap \langle l_5, l_6\rangle$
(f)
$g_3 = N(g_1) \cap N(g_2)$-l_2
(g)
Convex corner
Concave corner
(h)

FIGURE 5. Process of identifying concave corners and convex corners. (a) Unknown map. (b) Map exploration(follows the wall). (c) Search for corner while following wall. (d) Selection of corner l_5's surrounding grids g_1; g_2. (e) Selection of grid g_3, which is surrounded by g_1 and g_2. (f) Selection of corner l_6's surrounding grids g_1; g_2. (g) Selection of grid g_3, which is surrounded by g_1 and g_2. (h) Final results identifying all corners.

For grid g_3, if there are obstacles on the map, l_i is defined as a convex corner; if g_3 is an empty grid, l_i is defined as a concave corner. Thus, the dark gray colored regions represent the grids surrounding g_1 and g_2. Further, using (2), the coordinates of selected grid g_3 are $(12, 2)$. Because grid g_3 is an empty grid, grid l_5 is defined as a concave corner. Figure 5(f)–(g) displays the process of categorizing the type of corner of grid l_6. Equations (1) and (2) are used for grid l_6, and because there are obstacles in the finally selected g_3, grid l_6 is defined as a convex corner. Lastly, Figure 5(h) displays the results of identifying the type of all the corners; the symbols $\otimes$ and $\oplus$ represent concave and convex corners, respectively. After the classification of corners into concave and convex, map decomposition using the convex corners selected in the proposed method is initiated. As in Figure 5(h), the information on six concave corners and two convex corners is collected. Next, the collected convex corners are used to divide the map into rectangles and edges are created for map decomposition.

Figure 6 displays the decomposition process of the map using the decomposition edges created by creating decomposition edges using the two convex corners $(7, 5)$ and $(11, 5)$. The vertical and horizontal edges (solid lines) that are the boundary lines of the convex corner $(7, 5)$ are displayed in Figure 6(a) at $\langle(1, 1), (7, 1)\rangle$, $\langle(1, 1), (1, 17)\rangle$, $\langle(1, 17), (17, 17)\rangle$, and $\langle(17, 1), (17, 17)\rangle$. These display the four decomposition edges; displayed as coordinates, they are represented by $\langle(7, 5), (7, 1)\rangle$, $\langle(7, 5), (1, 5)\rangle$, $\langle(7, 5), (7, 17)\rangle$, and $\langle(7, 5), (17, 5)\rangle$. Because decomposition into rectangular sub-maps is possible if one edge among the four edges is used, one decomposition edge that meets the following conditions is selected.

1) A decomposition edge should not overlap with map boundary edges and the boundary edges of the explored obstacles.
2) A decomposition edge should not pass through explored obstacles.
3) A decomposition edge should not overlap with the decomposition edge formed first by another convex corner.
4) If there are decomposition edges that meet conditions 1), 2), and 3), then one will be selected by a random method.

Using the convex corner $(7, 5)$, among the created four edges, the decomposition edge $\langle(7, 5), (1, 5)\rangle$ that satisfied all four conditions is selected. Next, using the selected decomposition edge, the boundary edges that intersect are divided. In Figure 6(b), intersection point $(1, 5)$ where the decomposition edge $\langle(7, 5), (1, 5)\rangle$ and boundary edge $\langle(1, 1), (1, 17)\rangle$ intersect is marked with a circle. Based on the intersection point $(1, 5)$ where boundary edges $\langle(1, 1), (1, 17)\rangle$ intersect, the boundary edges are divided into $\langle(1, 1), (1, 5)\rangle$ and $\langle(1, 5), (1, 17)\rangle$. Figure 6(c) indicates how the four decomposition edges $\langle(11, 5), (11, 1)\rangle$, $\langle 11, 5), (1, 5)\rangle$, $\langle(11, 5), (11, 17)\rangle$, and $\langle(11, 5), (17, 5)\rangle$ with respect to the convex corner (11,5) are created. Of the four decomposition edges, because $\langle(11, 5), (11, 1)\rangle$ and $\langle(11, 5), (1, 5)\rangle$ satisfy Conditions 1) and 3), they are removed. Of the two edges $\langle(11, 5), (11, 17)\rangle$ and $\langle(11, 5), (17, 5)\rangle$, $\langle(11, 5), (11, 17)\rangle$ is randomly selected and lastly, the convex corner $(11, 5)$ is selected as a decomposition edge. In Figure 6(d), the convex corners $(11, 5)$ and $(7, 5)$ are used and the selected two decomposition edges $\langle(11, 5), (11, 17)\rangle$ and $\langle(7, 5), (1, 5)\rangle$ are indicated.

Next, based on the selected two decomposition edges and ten boundary edges, a sub-map is created. In the process of creating a rectangular sub-map, the A* algorithm, which uses the heuristic estimated value for the problem of calculating the shortest path, is used as it is frequently employed to determine the shortest path from a starting point to an ending point in a map [12], [13]. Following this, a sub-map is created. Figure 6(e) displays the process of creating a rectangular sub-map using the A* algorithm. From the current position of the cleaning robot (triangle), edge $\langle(1, 1), (1, 5)\rangle$ is selected and grids $(1, 1)$ and $(1, 5)$ are defined as the starting point and target point, respectively. The A* algorithm is used and exploration begins from the starting point; the arrow indicates the range of exploration. The result is that edges $\langle(1, 1), (1, 5)\rangle$, $\langle(1, 5), (7, 5)\rangle$, $\langle(7, 5), (7, 1)\rangle$, and $\langle(7, 1), (1, 1)\rangle$ are constructed as one sub-map defined as $\ddot{\chi}_1$. In Figure 6(f), through the same process, the second sub-map is created. Because the edges $\langle(1, 1), (1, 5)\rangle$, $\langle(1, 5), (7, 5)\rangle$, $\langle(7, 5), (7, 1)\rangle$, and $\langle(7, 1), (1, 1)\rangle$ are already used, among the remaining edges, a random edge is selected, Further, the starting point and target point are selected.

Edge $\langle(17, 1), (11, 1)\rangle$ is selected and grids $(17, 1)$ and $(11, 1)$ are selected as the starting point and target point, respectively. Using the A* algorithm, the second sub-map is created and defined as $\ddot{\chi}_2$. Figure 6(g) indicates that the third created sub-map is defined as $\ddot{\chi}_3$. After creating sub-map $\ddot{\chi}_3$, because all edges are used, the map decomposition process terminates. The final decomposed three sub-maps $\ddot{\chi}_1$, $\ddot{\chi}_2$, and $\ddot{\chi}_3$ are displayed in Figure 6(h). Each sub-map is displayed by a rectangle with a pattern. The decomposition edge and boundary edges that appeared first become the boundary edges displayed in the sub-maps with solid lines. Further, owing to the fact that the collected concave corner, convex corner, and intersection points could become corners in the divided sub-maps, and the fact that the type of corner of a decomposed sub-map can change, all corners are returned to a nonclassified situation and marked with the condition ($\bigcirc$). The Map decomposition function is described in Algorithm 5.

B. RECTANGULAR SUB-MAP CLEANING AND NEXT SUB-MAP SELECTION IN BLIND ALLEY SITUATION

Following the completion of the map decomposition, the cleaning robot can use the corner of each sub-map to create the proposed spiral path based on the edges. In this manner, the robot cleans using the spiral coverage path based on the created edges. Figure 7 displays the cleaning process whereby the cleaning robot uses the spiral path based on its edges. The cleaned grids are represented in gray.

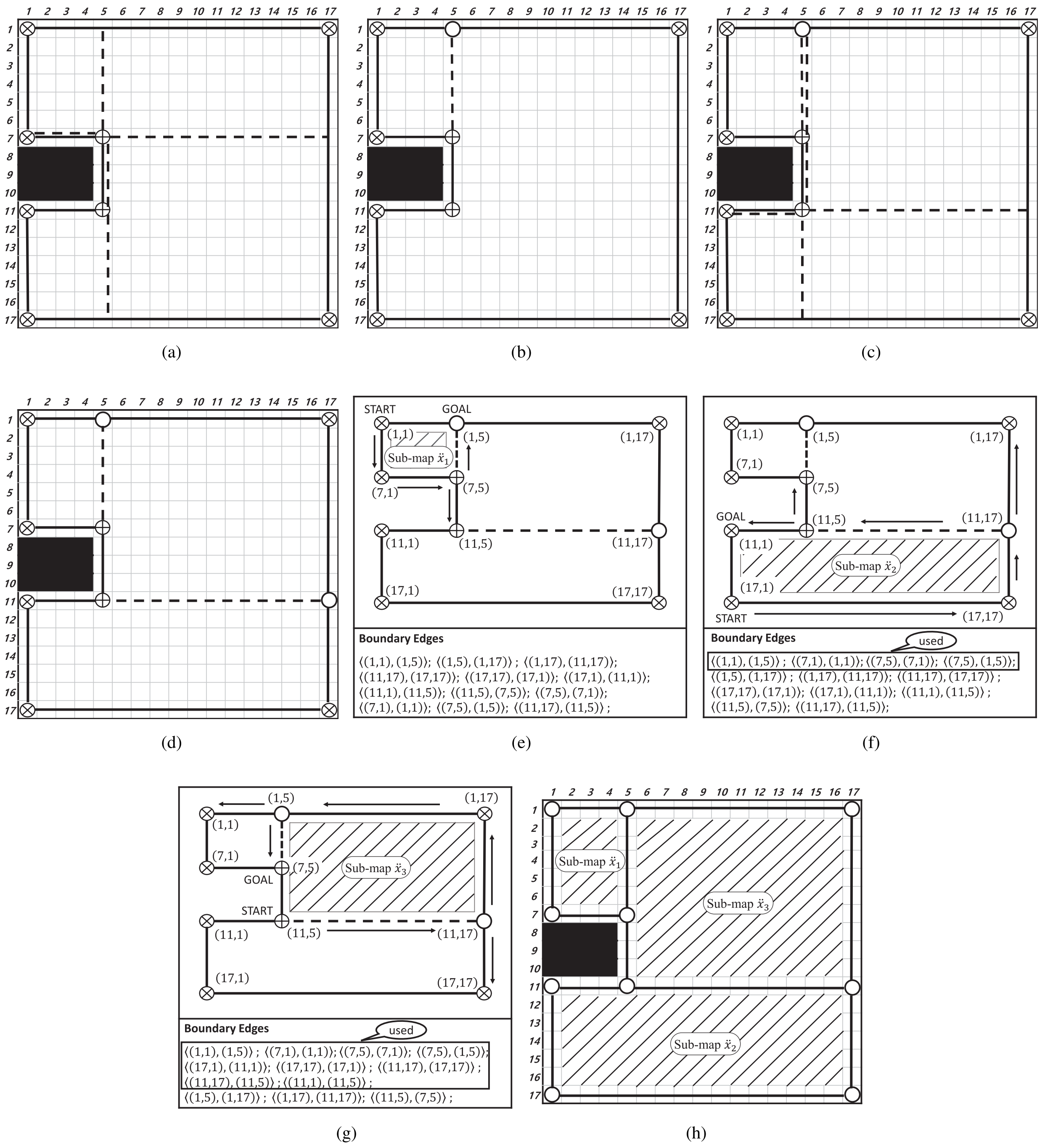


FIGURE 6. Process of decomposition into rectangular sub-maps (Solid line: boundary edge, Dotted line: decomposition edge, Arrow: Exploration range). (a) Start of creating edge using convex corner (7, 5). (b) Selecting edge that connects (1, 5) and (7, 5). (c) Start of creating edge using convex corner (11, 5). (d) Selecting edge that connects (11, 5) and (11, 17). (e) Creating rectangular sub-map $\ddot{x}_1$ that used A*. (f) Creating the rectangular sub-map $\ddot{x}_2$ that used A*. (g) Creating rectangular sub-map $\ddot{x}_3$ that used A*. (h) Division into three rectangular sub-maps.

Figure 7(a) represents the edges in the map decomposition process of $\ddot{x}_1$ as $\langle(1, 1), (1, 5)\rangle$, $\langle(1, 5), (7, 5)\rangle$, $\langle(7, 5), (7, 1)\rangle$, and $\langle(7, 1)(1, 1)\rangle$. As the cleaning robot (triangle) follows the wall based on the edges from sub-map $\ddot{x}_1$, it collects information on the corners.

Figure 7(a) displays the path following the edges with a solid line. Figure 7(b) indicates the collected four concave corners with respect to sub-map $\ddot{x}_1$. The number next to the circle displays the order in which information regarding the concave corners is received. Because a convex corner does not exist in sub-map $\ddot{x}_1$, map cleaning begins. When cleaning the map, the process of creating the spiral coverage path based on using the edges is as follows. First, for the collected concave corners, (3) is used and neighboring grids

Algorithm 5 Map Decomposition

```
1: Input: $current, convex, \ddot{x}_{boundary}, o_{boundary}$   {$\ddot{x}_{boundary}$: boundary of sub-map, $o_{boundary}$: boundary of obstacle}
2: Output: $submap$   {$submap$: set of sub-map}
3: $D \leftarrow o_{boundary}$;   {$D$: set of edges}
4: $submap \leftarrow \{\phi\}$;
5: for all $convex_i \in convex$ do
6:     $cedge \leftarrow$ Use $convex_i$ to create edges;
7:     $sedge \leftarrow$ Select decomposition edges from $cedge$;
8:     $D \leftarrow D \cup \{sedge\}$;
9: end for
10: $\langle start, target \rangle \leftarrow$ Find an edge from $\ddot{x}_{boundary}$ that include $current$;
11: while $\ddot{x}_{boundary} \neq \{\phi\}$ do
12:     $\ddot{x}_{boundary} \leftarrow$ Edge $\langle start, target \rangle$ is deleted from the $\ddot{x}_{boundary}$;
13:     $s \leftarrow D \cup \ddot{x}_{boundary}$;
14:     $createmap \leftarrow$ A* $(start, target, s)$;
15:     $submap \leftarrow submap \cup \{createmap\}$;
16:     $D \leftarrow D \cup \{createmap\}$;
17:     $\ddot{x}_{boundary} \leftarrow createmap$ is deleted from the $\ddot{x}_{boundary}$;
18:     $\langle start, target \rangle \leftarrow$ Select an edge from the $\ddot{x}_{boundary}$ randomly;
19: end while
```

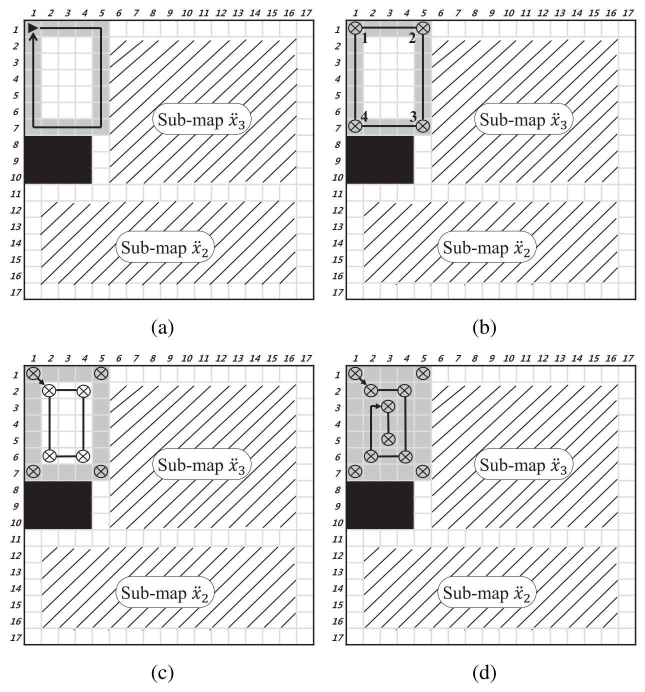


FIGURE 7. Cleaning process of the sub-map $\ddot{x}_1$ using edge-based spiral coverage path. (a) Following wall. (b) Sub-map $\ddot{x}_1$'s edge. (c) Creating spiral coverage path based on edges. (d) Cleaning sub-map $\ddot{x}_1$ through an edge-based spiral coverage path.

in a diagonal direction are calculated. For example, for the first concave corner, the four neighboring diagonal grids are marked using (3).

$$\{(x_j, y_j) | j = 1, 2, 3, 4\} \tag{3}$$

(x_j, y_j) is the coordinate of the j^{th} neighboring grid on the first concave corner. Equation (4) is used to mark the highest and lowest coordinates on the X and Y axes of the sub-map $\ddot{x}_1$ where edges are represented.

$$\begin{aligned}(max_x, max_y) &= max(\ddot{x}_1)\\ (min_x, min_y) &= min(\ddot{x}_1)\end{aligned} \tag{4}$$

Next, using (5), the location relationship between sub-map $\ddot{x}_1$ and the surrounding grids can be calculated.

$$\begin{aligned}T &= sign(x_j - min_x) + sign(x_j - max_x)\\ &\quad + sign(y_j - min_y) + sign(y_j - max_y)\\ sign(n) &= \begin{cases}1 & n \geq 0\\ -1 & n < 0\end{cases}\end{aligned} \tag{5}$$

When $T = 0$, the (x_j, y_j) grid implies inside sub-map $\ddot{x}_1$; when $T \neq 0$, the (x_j, y_j) grid implies outside sub-map $\ddot{x}_1$ or on the edges. Through the value of T, among the neighboring grids, a grid inside sub-map $\ddot{x}_1$ can be chosen. In Figure 7(c), (3), (4), and (5) are used on the four concave corners and the final grids (2, 2), (2, 4), (6, 4), and (6, 2) are selected and marked with the symbol $\otimes$. Next, the concave corners are connected to the grids in their respective order and become new edges $\langle(2, 2), (2, 4)\rangle$, $\langle(2, 4), (6, 4)\rangle$, $\langle(6, 4), (6, 2)\rangle$, and $\langle(6, 2), (2, 2)\rangle$. The new edges calculated above are selected as the edges of the current sub-map $\ddot{x}_1$.

The cleaning robot follows the new edges and continuously cleans. When it reaches (3, 2), it uses the same process to calculate another grid inside sub-map $\ddot{x}_1$. Using (3), (4), and (5) on grids (2, 2), (2, 4), (6, 4), and (6, 2), the grids (3, 3) and (5, 3) are obtained. In the same manner, sub-map $\ddot{x}_1$'s edge changes to edge $\langle(3, 3), (5, 3)\rangle$. Figure 7(d) indicates that the cleaning robot can follow the edge $\langle(3, 3), (5, 3)\rangle$ and clean; when it arrives at grid (5, 3), it again uses (3), (4), and (5) on sub-map $\ddot{x}_1$'s current edge $\langle(3, 3), (5, 3)\rangle$ and grids (3, 3), (5, 3). However, it cannot obtain a sufficient grid inside sub-map $\ddot{x}_1$. Hence, map cleaning using the edge-based spiral coverage path terminates. The Submap cleaning function is described in Algorithm 6.

When the cleaning of the cleaning robot inside sub-map $\ddot{x}_1$ completes, because the neighboring grids of the cleaning robot are all cleaned, a blind alley is produced. The robot then quickly moves to the next sub-map. To create a path to the next sub-map to be cleaned, it uses edges to explore the path. Figure 8 displays the process by which the robot moves to sub-map $\ddot{x}_2$ after completing the cleaning of sub-map $\ddot{x}_1$. Figure 8(a) indicates the cleaning robot's (triangle) current position, sub-map $\ddot{x}_2$, and the edges of sub-map $\ddot{x}_2$, which are marked by solid lines after the robot has cleaned sub-map $\ddot{x}_1$. Because maps $\ddot{x}_1$ and $\ddot{x}_2$ can be adjoined, the cleaning robot moves first to sub-map $\ddot{x}_1$'s corner. As indicated in Figure 8(b), four corners are included on the sub-map. Using the following process, a corner is selected and the cleaning robot moves.

1) A corner in common with sub-map $\ddot{x}_2$ is selected among the sub-map $\ddot{x}_1$ corners.

Algorithm 6 Submap Cleaning

```
1: Input: $current$, $\ddot{x}_{boundary}$
2: Output: $current$, $o_{boundary}$
3: while true do
4:     $(current, o_{boundary}) \leftarrow$ Follows the $\ddot{x}_{boundary}$;
5:     if $o_{boundary} \neq \{\phi\}$ then
6:         Break;
7:     end if
8:     $temporary_{boundary} \leftarrow \{\phi\}$;
9:     $(max_x, max_y) \leftarrow max(\ddot{x}_{boundary})$;
10:    $(min_x, min_y) \leftarrow min(\ddot{x}_{boundary})$;
11:    for all $corner \in \ddot{x}_{boundary}$ do
12:        $(x, y) \leftarrow$ Find four neighbors of $corner$ in a diagonal direction;
13:        for $j \leftarrow 1$ to 4 do
14:            $T \leftarrow sign(x_j, y_j, max_x, max_y, min_x, min_y)$;
15:            if $T \neq 0$ then
16:                $temporary_{boundary} \leftarrow temporary_{boundary} \cup \{(x_j, y_j)\}$;
17:            end if
18:        end for
19:        if $temporary_{boundary} = \{\phi\}$ then
20:            Break;
21:        end if
22:        $\ddot{x}_{boundary} \leftarrow temporary_{boundary}$;
23:    end for
24: end while
```

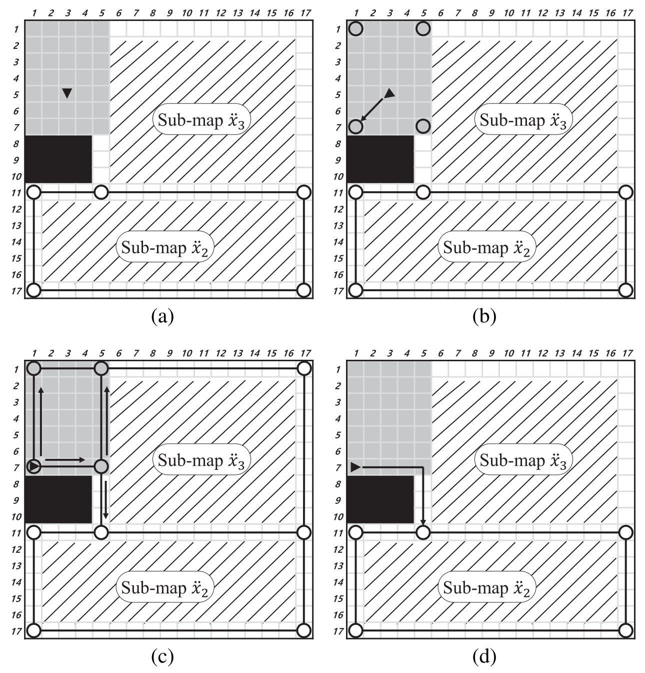


FIGURE 8. Using edge-based BFS algorithm and creating path to move to sub-map $\ddot{x}_2$. (a) Current location of robot and sub-map $\ddot{x}_2$. (b) Movement within sub-map $\ddot{x}_1$. (c) Creating path using BFS. (d) Movement to sub-map $\ddot{x}_2$.

2) If there are multiple corners in common with map $\ddot{x}_2$, the Euclidean distances between the cleaning robot and the corners are calculated. The shortest path is selected; if the distances are the same, one is chosen randomly.

3) If there is no corner in common between sub-map $\ddot{x}_1$ and sub-map $\ddot{x}_2$, on sub-map $\ddot{x}_1$, the Euclidean distances between all the corners in sub-map $\ddot{x}_1$ and the cleaning robot are calculated and the shortest path is selected; if the distances are the same, one is chosen at random.

Algorithm 7 Submap Selection

```
1: Input: $current$, $submap$, $o_{boundary}$
2: Output: $current$, $index$
3: $index \leftarrow$ Check $current$ in which corner of $submap$;
4: if $index \neq \phi$ then
5:     Return;
6: else
7:     $dist \leftarrow$ Calculate the distance from $current$ to $submap$;

8:     $index \leftarrow$ Use $dist$ to find the nearest sub-map;
9:     $start \leftarrow current$;
10:    $target \leftarrow submap\{index\}$;
11:    $s \leftarrow submap \cup o_{boundary}$;
12:    $path \leftarrow$ BFS $(start, target, s)$;
13:    $current \leftarrow$ Follows the $path$
14: end if
```

Using process 3) indicated in Figure 8(b), corner (7, 1) is selected and the cleaning robot moves to the selected corner (7, 1). Because corner (7, 1) is not a corner of sub-map $\ddot{x}_2$, the cleaning robot uses the created edges from the map decomposition process and determines a path to move to sub-map $\ddot{x}_2$. Figure 1(c) indicates that when a conventional CPP method calculates path planning to the next area to be cleaned, it considers all the grids on the map and creates a path. To decrease the explored space for creating a path, Figure 8(c) indicates that based on the edges produced in the map decomposition process, the conventional BFS algorithm can be used to create a path to the sub-map $\ddot{x}_2$. First, boundary edge information is used. The cleaning robot first explores corners connected to corner (7, 1). Then, it explores corners (7, 5) and (1, 1). Because these two corners are not corners on sub-map $\ddot{x}_2$, on the basis of corner (7, 5) and corner (1, 1), a new connected corner is explored. From corner (7, 5), corner (11, 5) and corner (1, 5) are explored. Because corner (11, 5) is a corner on sub-map $\ddot{x}_2$, the BFS algorithm is terminated; the arrow indicates the entire explored region. In Figure 8(d), the cleaning robot follows the path determined by the BFS algorithm. It then moves to sub-map $\ddot{x}_2$. Next, it begins cleaning from corner (11, 5). The Select submap function is described in Algorithm 7.

C. ADDRESSING THE OCCURRENCE OF UNKNOWN OBSTACLES

The rectangular sub-maps can be divided using the proposed map decomposition method. However, owing to the lack of information on an unknown map, there can be unexplored

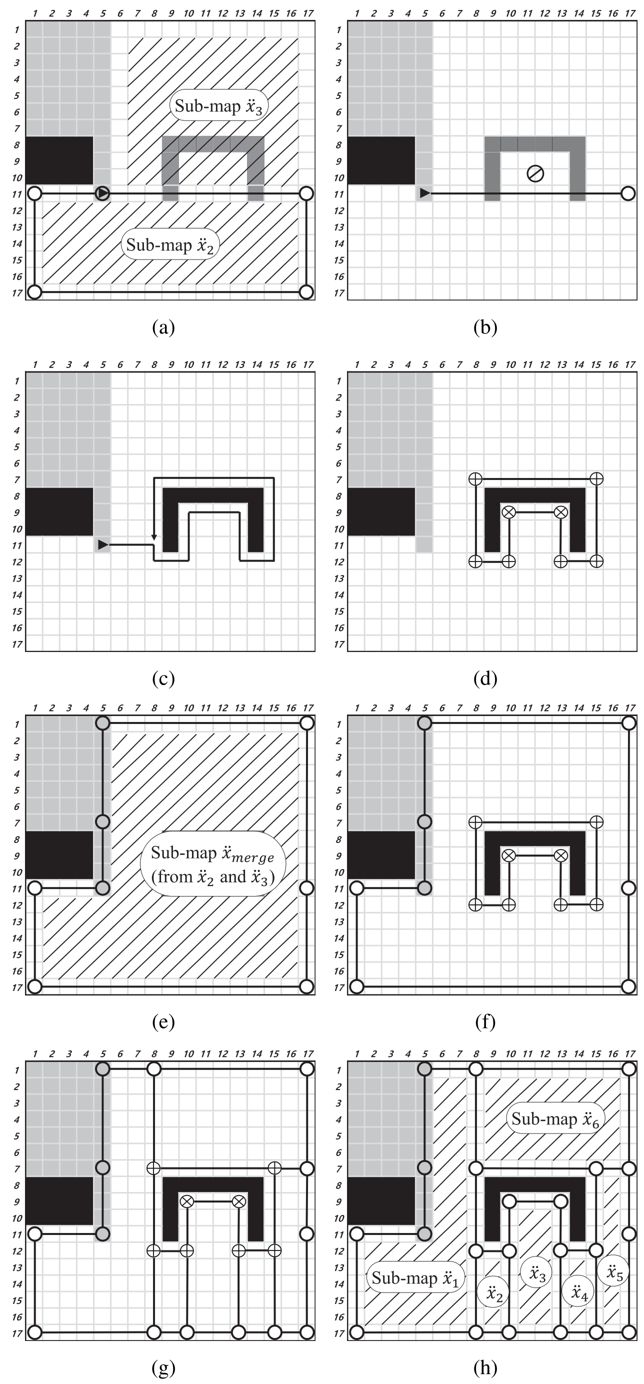


FIGURE 9. **Prevention of closed area regarding obstacles not explored in map. (a) Case when unexplored obstacles exist sub-map $\ddot{x}_2$ and sub-map $\ddot{x}_3$. (b) Closed area formed by sub-map $\ddot{x}_2$'s edges and obstacles. (c) Following obstacle's walls. (d) Recognizing obstacle's corner. (e) Constrution of a $\ddot{x}_{merge}$ map that combines sub-map $\ddot{x}_2$ and $\ddot{x}_3$. (f) Obstacles corner and $\ddot{x}_{merge}$ map. (g) Creating edge. (h) Map decomposition using A*.**

obstacles on the divided sub-maps. Thus, a closed space is produced between a divided sub-map and the obstacles. If a closed space is produced, the cleaning robot recognizes this as an obstacle and does not perform cleaning of the closed space. Therefore, for a divided sub-map similar to that in Figure 9, map decomposition is again performed. In Figure 9(a), $\prod$-type unexplored obstacles exist in sub-map $\ddot{x}_2$ and sub-map $\ddot{x}_3$. Among the edges of sub-map $\ddot{x}_2$, edge $\langle(11, 5), (11, 17)\rangle$ and unexplored obstacles intersect. In Figure 9(b), because unexplored obstacles and edge $\langle(11, 5), (11, 17)\rangle$ intersect, a closed space with symbol $\oslash$ is produced. Because a closed space exists outside of sub-map $\ddot{x}_2$, the cleaning robot cannot clean the closed space. Further, in sub-map $\ddot{x}_3$, a closed space ($\oslash$) is encircled by neighboring obstacles; hence, it is impossible for the cleaning robot to enter and clean the closed space. Thus, to prevent a closed space, edge $\langle(11, 5), (11, 17)\rangle$ is deleted. Sub-maps relating to edge $\langle(11, 5), (11, 17)\rangle$ are combined and map decomposition is performed again.

Figure 9(c)–(h) displays the entire process of preventing closed spaces. First, for the cleaning robot to clean sub-map $\ddot{x}_2$, in the process of moving, it follows the edges of sub-map $\ddot{x}_2$ and discovers new obstacles. Next, as in Figure 9(c), the cleaning robot follows the walls of obstacles and collects information on the corners and edges of the obstacles. Figure 9(d) indicates that in the process of following the obstacles, eight edges are determined and indicated by solid lines, with six convex corners by ($\oplus$), and two concave corners by ($\otimes$). The cleaning robot identifies obstacles at edge $\langle(11, 5), (11, 17)\rangle$, hence, edge $\langle(11, 5), (11, 17)\rangle$ is deleted and sub-maps that include edge $\langle(11, 5), (11, 17)\rangle$ are selected. Sub-map $\ddot{x}_2$ and sub-map $\ddot{x}_3$ are connected as one sub-map and defined as $\ddot{x}_{merge}$.

In Figure 9(e), sub-map $\ddot{x}_2$ and sub-map $\ddot{x}_3$ form sub-map $\ddot{x}_{merge}$, the edges of sub-map $\ddot{x}_{merge}$ are marked as solid lines. In Figure 9(e), solid lines mark the edges of sub-map $\ddot{x}_{merge}$, which is formed by sub-map $\ddot{x}_2$ and sub-map $\ddot{x}_3$. Circles represent the corners of the $\ddot{x}_{merge}$ map. Figure 9(f) displays the $\prod$-type obstacles and sub-map $\ddot{x}_{merge}$'s corners and edges. In Figure 9(g), the convex corners of the obstacles are used. Edges are created; solid lines indicate the final created edges. The map decomposition results using the A* algorithm on the created edges are indicated in Figure 9(h). Sub-map $\ddot{x}_{merge}$, which was formed from sub-map $\ddot{x}_2$ and sub-map $\ddot{x}_3$, is again divided into six new sub-maps and the closed space generated by the $\prod$-type obstacles is prevented.

IV. EXPERIMENTAL RESULTS

To test the performance of the proposed method from the viewpoint of computational efficiency and effectiveness, we chose five conventional CPP methods: LSP [14], STC [15], BSA [9], BA* [16] and TWPS [18]. These methods employ different path planning techniques for determining the next uncleaned grid: STC recursively calls the spanning tree-based CPP, BSA and LSP use the BFS and CIDT methods with backtracking grids, BA* uses the A* algorithm, and TWPS use a two-way proximity search. We conducted experiments related to the performance of the proposed method on eight maps composed of diverse shapes and different numbers of obstacles. Figure 10 displays an illustration of each map and its corresponding name.

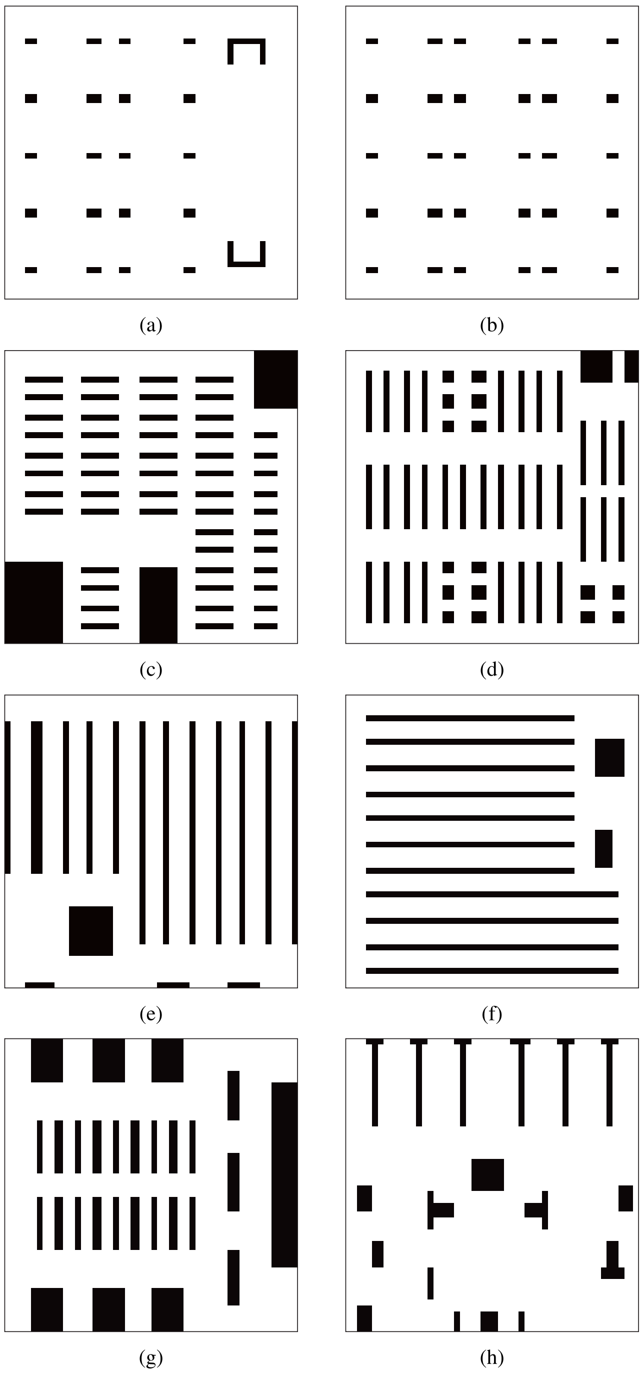


FIGURE 10. **Illustrations of experimental maps. (a) Gym 01. (b) Gym 02. (c) Library 01. (d) Library 02. (e) Warehouse 01. (f) Warehouse 02. (g) Airport 01. (h) Airport 02.**

To verify the scalability of the proposed strategy and compare it against conventional CPP methods, the eight given maps were increased 1 $\sim$ 10 times the original size while maintaining their shapes. Maps sized 100×100 to $1,000 \times 1,000$ were used to compare the efficiencies of each method. The CPP quality was measured in both terms of coverage ratio and execution time based on the path created by each method and the running time for CPP. The coverage ratio measures the portion of cleaned grids among uncleaned grids, where a value of 1.0 represents perfect coverage. The execution time measures the running time required to obtain the final coverage path. The simulation was performed on a 3.40 GHz Intel Core i7 with 16 GB of memory, running a MATLAB 8.0 environment.

TABLE 1. **Coverage ratio of compared methods on four experimental maps with different sizes**

Map	Size	Proposed	BSA	LSP	BA*	TWPS	STC
Gym 01	100×100	1.0	1.0	1.0	1.0	1.0	1.0
	200×200	1.0	1.0	1.0	1.0	1.0	N/A
	500×500	1.0	1.0	1.0	1.0	1.0	N/A
	$1,000 \times 1,000$	1.0	1.0	1.0	1.0	1.0	N/A
Library 01	100×100	1.0	1.0	1.0	1.0	1.0	1.0
	200×200	1.0	1.0	1.0	1.0	1.0	N/A
	500×500	1.0	1.0	1.0	1.0	1.0	N/A
	$1,000 \times 1,000$	1.0	1.0	1.0	1.0	1.0	N/A
Warehouse 01	100×100	1.0	1.0	1.0	1.0	1.0	1.0
	200×200	1.0	1.0	1.0	1.0	1.0	N/A
	500×500	1.0	1.0	1.0	1.0	1.0	N/A
	$1,000 \times 1,000$	1.0	1.0	1.0	1.0	1.0	N/A
Airport 01	100×100	1.0	1.0	1.0	1.0	1.0	1.0
	200×200	1.0	1.0	1.0	1.0	1.0	N/A
	500×500	1.0	1.0	1.0	1.0	1.0	N/A
	$1,000 \times 1,000$	1.0	1.0	1.0	1.0	1.0	N/A

Table 1 presents the coverage ratio of the six compared methods on four experimental maps of different sizes: Gym 01, Library 01, Warehouse 01, and Airport 01. In each row of Table 1, the coverage ratio achieved by each CPP method on a given map ranging from 100×100 to $1,000 \times 1,000$ size is represented. The experimental results confirm that the proposed method achieved a perfect coverage ratio regardless of the size of the given map. As mentioned previously, the coverage ratio is the ratio between the total free spaces on the map and the cleaned spaces. In this study, 1.0 indicates a 100% coverage ratio. However, we failed to record the coverage ratio in the 200×200-sized map because recording the STC of conventional CPP methods requires a large amount of memory resource. Thus, the relevant information is recorded as N/A. Furthermore, STC could not produce the final coverage path if the size of the map was larger than 200×200. We observed the same tendency in the remaining four maps of different sizes; all CPP methods achieved a perfect coverage ratio except STC.

Figure 11 displays the execution time required for each compared method to plan the final coverage path; the execution time for each method on the 500×500, 700×700, and $1,000 \times 1,000$ map sizes are represented by overlaid blue, green and red bars, respectively (this can be viewed in the color-printed or PDF version of this paper). Because STC consistently failed to produce the coverage path for a given map if the size of the map was larger than 200×200, we excluded STC from the experiments. The experimental results reveal that the execution times of each method differ according to the map, and that overall, the execution times of the proposed method is less than conventional CPP methods. Further, the difference between the execution times increases

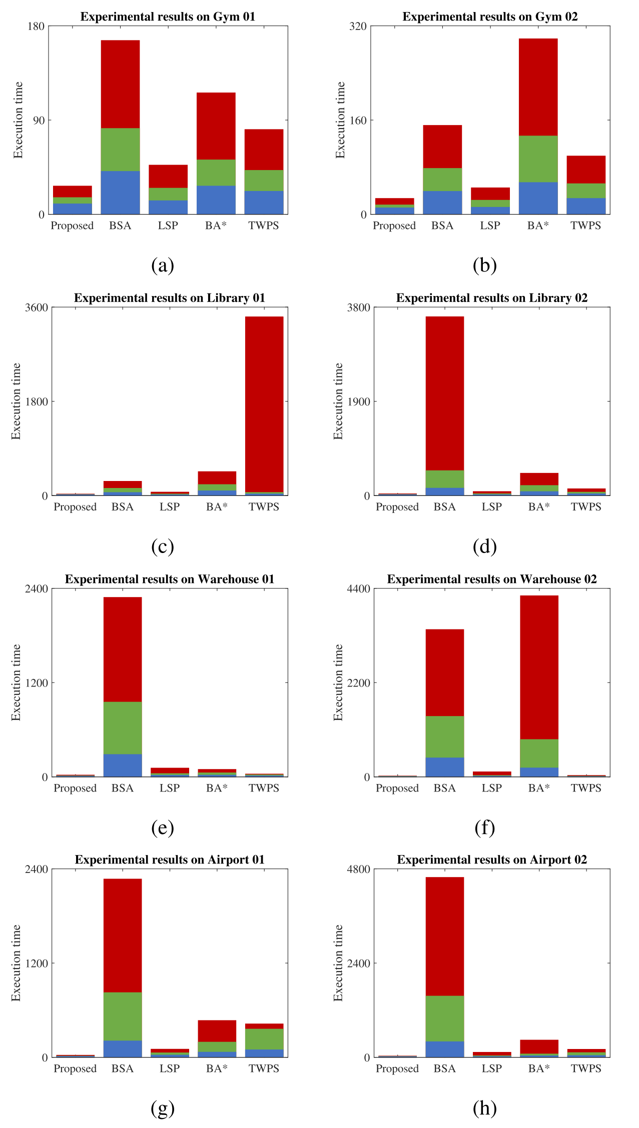


FIGURE 11. Comparison results of CPP methods in terms of execution time in seconds(s) (the execution time of each method on maps with sizes of 500 × 500, 700 × 700, and 1, 000 × 1, 000 are represented by blue, green and red overlaid bars, respectively). (a) Gym 01. (b) Gym 02. (c) Library 01. (d) Library 02. (e) Warehouse 01. (f) Warehouse 02. (g) Airport 01. (h) Airport 02.

as the maps become larger. For example, the execution times of the proposed method, BSA, LSP, BA*, and TWPS on Gym 01 when the map size was 500×500 were 10 s, 41 s, 13 s, 27 s, and 22 s, respectively. The largest and smallest differences between the execution times were 31 s and 3 s. The execution times of each method when the map size was 700×700 were 16 s, 82 s, 25 s, 52 s, and 42 s. The greatest and least differences between the execution times were 66 s and 9 s. The execution times of each method when the map size was $1,000 \times 1,000$ were 27 s, 166 s, 47 s, 116 s, and 81 s. The greatest and least differences between the execution times were 139 s and 20 s. Moreover, with the more complicated Airport 02 map, the execution times when the map size was $1,000 \times 1,000$ were 34 s, 4,582 s, 129 s, 440 s, and 83 s. The greatest and least differences between the execution times were 4,548 s and 49 s. The results demonstrate that with the proposed method, the cleaning robot could perform path planning in a reduced amount of time on large-sized maps compared to conventional CPP methods, particularly compared to the BSA and BA* methods. Further, as the sizes of the maps increased, the execution time of the proposed method increased more slowly, whereas that of the compared methods increased rapidly.

TABLE 2. Comparison results of execution time required for clean map (SM) and path to next uncleaned grid or sub-map (Move). Least execution times among five CPP methods are marked with † symbol.

	Proposed	BSA	LSP	BA*	TWPS
Map	Gym 01				
SM	21	102	35	**8**†	9
Move	**6**†	64	12	108	72
Total	**27**†	166	47	116	81
Map	Gym 02				
SM	21	102	34	**8**†	10
Move	**6**†	49	11	290	89
Total	**27**†	151	45	298	99
Map	Library 01				
SM	18	78	26	**7**†	8
Move	**12**†	195	41	451	3404
Total	**30**†	273	67	458	3412
Map	Library 02				
SM	19	85	28	**8**†	9
Move	**15**†	3521	54	443	129
Total	**34**†	3606	82	451	138
Map	Warehouse 01				
SM	17	84	28	**6**†	8
Move	**6**†	2201	84	89	28
Total	**23**†	2285	112	95	36
Map	Warehouse 02				
SM	17	85	28	**7**†	8
Move	**4**†	3354	90	4221	26
Total	**21**†	3439	118	4228	34
Map	Airport 01				
SM	16	78	26	**7**†	8
Move	**11**†	2192	78	462	58
Total	**27**†	2270	104	469	66
Map	Airport 02				
SM	20	95	32	**8**†	9
Move	**14**†	4487	97	432	74
Total	**34**†	4582	129	440	83
Average standard deviation for all maps					
Move	**4**†	1629	32	1296	1104

To validate the superiority of the proposed method in more detail, we analyzed the experimental results of $1,000 \times 1,000$ maps from the viewpoint of execution time required for clean sub-maps and the path to the next grid. Table 2 represents the execution time that each method required for a clean map (SM) and the path from blind alley to the next uncleaned grid or uncleaned sub-map (Move), where the best performance among the compared CPP methods is identified by a † symbol. Experimental results confirm that the least total execution time for all experimental maps was achieved by the proposed method, indicating that the proposed method produced the final coverage path faster than all other methods. It is interesting to note that the proposed method required a significantly smaller execution time for planning the path to the next grid than the compared methods. For example, the proposed method required 21 s to clean map and 6 s to plan the path to the next grid in an uncleaned sub-map on the $1,000 \times 1,000$-sized Gym 01 map. Conversely, BA* required 8 s to clean map and 108 s to plan the path to the next uncleaned grid. Moreover, the average standard deviation of the time (Move) for all maps used in the proposed method to travel to an uncleaned grid from a blind alley was 4 s. With conventional CPP methods, the average standard deviation of the time (Move) for all maps were 1629 s, 32 s, 1296 s, and 1104 s each. Therefore, we can conclude that the proposed method requires a significantly reduced time (Move), rendering the method robustly applicable to a variety of different maps. Consequently, the proposed method produced the final coverage path considerably more quickly than BA*, supporting the assertion that the proposed strategy to invoke path planning based on edges is more efficient.

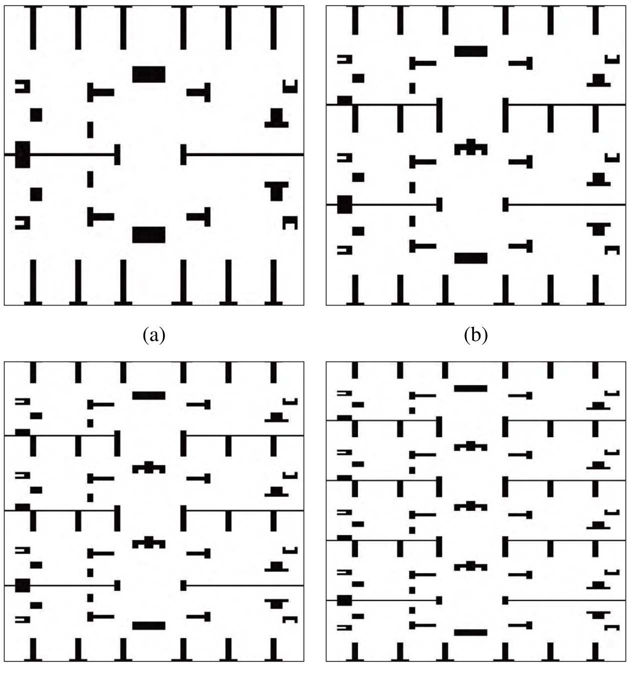


FIGURE 12. Variations of Airport 02. (a) Map size $2,000 \times 2,000$. (b) Map size $3,000 \times 3,000$. (c) Map size $4,000 \times 4,000$. (d) Map size $5,000 \times 5,000$.

As indicated in Figure 12, experiments were conducted with larger and more complicated maps to test the scalability of the proposed method. In Figure 12, each map was rendered in sizes $2,000 \times 2,000 \sim 5,000 \times 5,000$ and the number of obstacles in each map was increased accordingly. The execution times and number of sub-maps were compared between the proposed method and conventional CPP methods. As LSP and TWPS demonstrated the best performance among the conventional methods, the performance of the proposed method was compared to these two methods in Table 3. Table 3 indicates that the proposed method required a shorter

TABLE 3. Comparison results of LSP, TWPS, and proposed method on variations of Airport 02 with execution time(s) (NSM: number of sub-maps).

Map Size	LSP Time	TWPS Time	Proposed Time (NSM)
$2,000 \times 2,000$	949	77346	236 (97)
$3,000 \times 3,000$	2175	N/A	480 (143)
$4,000 \times 4,000$	8721	N/A	795 (183)
$5,000 \times 5,000$	17590	N/A	1263 (237)

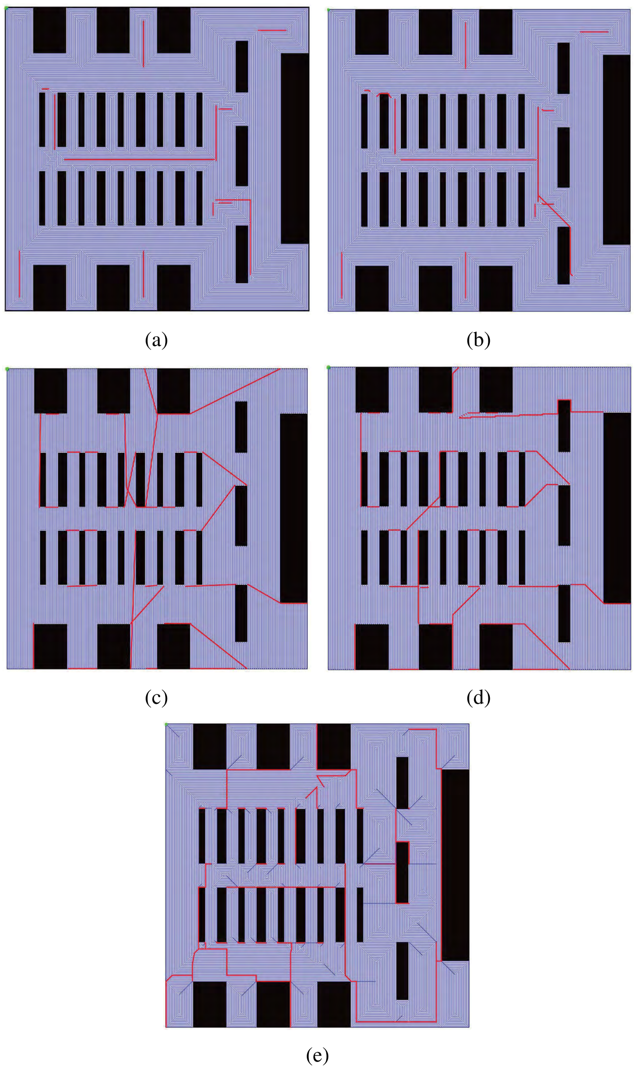


FIGURE 13. Simulation experiment of CPP methods. (a) BSA. (b) LSP. (c) BA*. (d) TWPS. (e) Proposed method.

execution time compared to LSP and TWPS. For example, the execution times of the proposed method, LSP, and TWPS when the map size was $2,000 \times 2,000$ were 236 s, 949 s, and 77,346 s, respectively. Because TWPS required excessive time, the remainder of the relevant information is recorded as N/A. In particular, when the map size was $5,000 \times 5,000$, the proposed method required 14 times less than the execution times of LSP. With the proposed method, the number of sub-maps generated increases with the number of obstacles; nonetheless, a reduced execution time is guaranteed.

In summary, the experimental results strongly validate that the proposed method is significantly more scalable to the size of any given map than conventional methods because the proposed method can plan the coverage path with a reduced execution time, even when a large map is involved. Lastly, we illustrate the final coverage paths of each CPP method on a 300×300-sized Airport 01 map to demonstrate the different characteristics of the planned paths; we intentionally chose 300×300-sized maps for clarity. Figure 13 displays the final coverage paths planned by the five CPP methods; a blue line displays the trajectory of a sub-map and a red line represents the trajectory of the robot when it moves to the next uncleaned grid. In Figure 13, we can observe that rectangular sub-maps were planned by the proposed method. Consequently, the robot could quickly plan the path to the next grid using sub-map boundaries without backtracking or path planning based on previously explored grids, which significantly degrades the efficiency of a cleaning robot. Readers can view the simulation video at http://ibot.knu.ac.kr/vediocleaningrobot.html.

V. CONCLUSION

In this paper, we proposed a scalable CPP method for extremely large environments based on rectangular sub-map and boundary edges to accelerate the CPP process. To achieve this, we developed a new map decomposition method that uses a spiral path for large, unknown environments. The experimental results on large maps demonstrated that the proposed method created the final coverage path without expending excessive computational resources for path planning, resulting in a significant acceleration of the cleaning process. Conversely, the conventional methods required considerably greater execution times on large maps than the proposed method, indicating that the proposed method is considerably more scalable than conventional CPP methods for sizable environments. Specifically, CPP using the proposed method was up to 14 times faster than CPP using conventional methods. A possible extension of this study could focus on the map selection process that selects the next cleaning sub-map by Euclidean distance from a set of created sub-map. Alternatively, the proposed approach could lengthen the final coverage path compared to conventional methods because the proposed method uses boundary and decomposition edges as opposed to diagonal edges, which can't shorten the final coverage path. The proposed method could solve some of the problems of conventional studies. However, its limitations include battery shortage when applying the method on a larger map with a single cleaning robot and sudden malfunctions. To solve these problems, we plan to expand the method into a multi cleaning robot system and test the method in real environments.

REFERENCES

[1] E. U. Acar, H. Choset, Y. Zhang, and M. Schervish, "Path planning for robotic demining: Robust sensor-based coverage of unstructured environments and probabilistic methods," *Int. J. Robot. Res.*, vol. 22, nos. 7–8, pp. 441–466, Jul. 2003.
[2] Z. L. Cao, Y. Huang, and E. L. Hall, "Region filling operations with random obstacle avoidance for mobile robots," *J. Robot. Syst.*, vol. 5, no. 2, pp. 87–102, Apr. 1988.
[3] T. Pilarski, M. Happold, H. Pangels, M. Ollis, K. Fitzpatrick, and A. Stentz, "The demeter system for automated harvesting," *Auton. Robots*, vol. 13, no. 1, pp. 9–20, Jul. 2002.
[4] H. Choset, "Coverage for robotics—A survey of recent results," *Annal. Mathemat. Intell.*, vol. 31, nos. 1–4, pp. 113–126, Oct. 2001.
[5] E. Galceran and M. Carreras, "A survey on coverage path planning for robotics," *Robot. Auton. Syst.*, vol. 61, no. 12, pp. 1258–1276, 2013.
[6] T. Palleja, M. Tresanchez, M. Teixidó, and J. Palacin, "Modeling floor-cleaning coverage performances of some domestic mobile robots in a reduced scenario," *Robot. Auton. Syst.*, vol. 58, no. 1, pp. 37–45, Jan. 2010.
[7] C. Hofner and G. Schmidt, "Path planning and guidance techniques for an autonomous mobile cleaning robot," *Robot. Autono. Syst.*, vol. 14, nos. 2–3, pp. 199–212, May 1995.
[8] T.-K. Lee, S.-H. Baek, Y.-H. Choi, and S.-Y. Oh, "Smooth coverage path planning and control of mobile robots based on high-resolution grid map representation," *Robot. Auton. Syst.*, vol. 59, no. 10, pp. 801–812, Oct. 2011.
[9] E. González, O. Álvarez, Y. Díaz, C. Parra, and C. Bustacara, "BSA: A complete coverage algorithm," in *Proc. IEEE Int. Conf. Robot. Autom.*, Barcelona, Spain, Apr. 2005, pp. 2040–2044.
[10] T.-K. Lee, S. Baek, and S.-Y. Oh, "Sector-based maximal online coverage of unknown environments for cleaning robots with limited sensing," *Robot. Auton. Syst.*, vol. 59, no. 10, pp. 698–710, Oct. 2011.
[11] M. Barbehenn, "A note on the complexity of Dijkstra's algorithm for graphs with weighted vertices," *IEEE Trans. Comput.*, vol. 47, no. 2, p. 263, Feb. 1998.
[12] S.-G. Cui, H. Wang, and L. Yang, "A simulation study of a-star algorithm for robot path planning," in *Proc. 16th Int. Conf. Mechatronics Technol.*, 2012, pp. 506–510.
[13] W. Zeng and R. L. Church, "Finding shortest paths on real road networks: The case for A*," *Int. J. Geograph. Inf. Sci.*, vol. 23, no. 4, pp. 531–543, Apr. 2009.
[14] Y.-H. Choi, T.-K. Lee, S.-H. Baek, and S.-Y. Oh, "Online complete coverage path planning for mobile robots based on linked spiral paths using constrained inverse distance transform," in *Proc. IEEE/RSJ Int. Conf. Intell. Robot. Syst.*, St. Louis, MI, USA, Oct. 2009, pp. 5788–5793.
[15] Y. Gabriely and E. Rimon, "Spanning-tree based coverage of continuous areas by a mobile robot," *Ann. Math. Artif. Intell.*, vol. 31, nos. 1–4, pp. 77–98, Oct. 2001.
[16] H. H. Viet, V.-H. Dang, M. N. U. Laskar, and T. Chung, "BA*: An online complete coverage algorithm for cleaning robots," *Appl. Intell.*, vol. 39, no. 2, pp. 217–235, Sep. 2013.
[17] M. A. Yakoubi and M. T. Laskri, "The path planning of cleaner robot for coverage region using genetic algorithms," *J. Innov. Digit. Ecosyst.*, vol. 3, no. 1, pp. 37–43, Jun. 2016.
[18] A. Khan, I. Noreen, H. Ryu, N. L. Doh, and Z. Habib, "Online complete coverage path planning using two-way proximity search," *Intell. Service Robot.*, vol. 10, no. 3, pp. 229–240, Jul. 2017.
[19] Y. Zhang and S. Li, "Distributed biased min-consensus with applications to shortest path planning," *IEEE Trans. Autom. Control*, vol. 62, no. 10, pp. 5429–5436, Oct. 2017.
[20] J. Song and S. Gupta, "$\varepsilon^\star$: An online coverage path planning algorithm," *IEEE Trans. Robot.*, vol. 34, no. 2, pp. 526–533, Apr. 2018.
[21] J. Lee, B.-Y. Kang, and D.-W. Kim, "Fast genetic algorithm for robot path planning," *Electron. Lett.*, vol. 49, no. 23, pp. 1449–1451, Nov. 2013.
[22] B.-Y. Kang, M. Xu, J. Lee, and D.-W. Kim, "ROBIL: Robot path planning based on PBIL algorithm," *Int. J. Adv. Robot. Syst.*, vol. 11, no. 147, pp. 1–14, Sep. 2014.

[23] A. Nikitenko, J. Grundspenkis, A. Liekna, M. Ekmanis, G. Kulikovskis, and I. Andersone, "Multi-robot system for vacuum cleaning domain," in *Proc. 12th Int. Conf. Pract. Appl. Agents Multi-Agent Syst.*, Salamanca, Spain, Jun. 2014, pp. 363–366.

[24] I. Shnaps and E. Rimon, "Online coverage of planar environments by a battery powered autonomous mobile robot," *IEEE Trans. Automat. Sci. Eng.*, vol. 13, no. 2, pp. 425–436, Apr. 2016.

[25] E. González, M. Alarcón, P. Aristizábal, and C. Parra, "BSA: A coverage algorithm," in *Proc. IEEE/RSJ Int. Conf. Intell. Robot. Syst.*, vol. 2. Las Vegas, NV, USA, Oct. 2003, pp. 1679–1684.

[26] Y. Guo and M. Balakrishnan, "Complete coverage control for nonholonomic mobile robots in dynamic environments," in *Proc. IEEE Int. Conf. Robot. Automat. (ICRA)*, May 2006, pp. 1704–1709.

[27] T.-K. Lee, S.-H. Baek, S.-Y. Oh, and Y.-H. Choi, "Complete coverage algorithm based on linked smooth spiral paths for mobile robots," in *Proc. 11th Int. Conf. Control Autom. Robot. Vis.*, Singapore, Dec. 2010, pp. 609–614.

[28] J. S. Oh, Y. H. Choi, J. B. Park, and Y. F. Zheng, "Complete coverage navigation of cleaning robots using triangular-cell-based map," *IEEE Trans. Ind. Electron.*, vol. 51, no. 3, pp. 718–726, Jun. 2004.

[29] L. Paull, S. Saeedi, H. Li, and V. Myers, "An information gain based adaptive path planning method for an autonomous underwater vehicle using sidescan sonar," in *Proc. IEEE Conf. Autom. Sci. Eng.*, Toronto, ON, Canada, Aug. 2010, pp. 835–840.

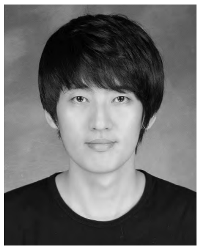
[30] A. Zelinsky, R. A. Jarvis, J. C. Byrne, and S. Yuta, "Planning paths of complete coverage of an unstructured environment by a mobile robot," in *Proc. 6th Int. Conf. Adv. Robot.*, vol. 13. Tokyo, Japan, Nov. 1993, pp. 533–538.

[31] Y. Gabriely and E. Rimon, "Spiral-STC: An on-line coverage algorithm of grid environments by a mobile robot," in *IEEE Conf. Robot. Autom.*, vol. 1. Washington, DC, USA, May 2002, pp. 954–960.

[32] Y. Gabriely and E. Rimon, "Competitive on-line coverage of grid environments by a mobile robot," *Comput. Geom.*, vol. 24, no. 3, pp. 197–224, Apr. 2003.

[33] S. Baek, T.-K. Lee, O. Se-Young, and K. Ju, "Integrated on-line localization, mapping and coverage algorithm of unknown environments for robotic vacuum cleaners based on minimal sensing," *Adv. Robot.*, vol. 25, nos. 13–14, pp. 1651–1673, Jan. 2011.

[34] S. C. Wong and B. MacDonald, "A topological coverage algorithm for mobile robots," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, vol. 2, Oct. 2003, pp. 1685–1690.



XU MIAO received the M.S. degree in mechanical engineering from Kyungpook National University, Seoul, South Korea, in 2013. He is currently pursuing the Ph.D. degree with the School of Robot Engineering, Kyungpook National University, Daegu, South Korea. His research interests include path planning of autonomous robots and artificial intelligence.

JAESUNG LEE received the B.S., M.S., and Ph.D. degrees in computer science from Chung-Ang University, Seoul, South Korea, in 2007, 2009, and 2013, respectively. He is currently a Research Professor with Chung-Ang University, Seoul. His research interests include data mining with applications to affective computing and ambient intelligence. In theoretical domain, he also studies classification, feature selection, and especially multi-label learning with information theory.

BO-YEONG KANG received the B.S., M.A., M.S., and Ph.D. degrees from Kyungpook National University (KNU), Daegu, South Korea. She has been an Associate Professor with the School of Mechanical Engineering, KNU, since 2009. Prior to coming to KNU, she was with Seoul National University as a Research Professor and held a post-doctoral position with KAIST. She is currently interested in artificial intelligence implementation for social and intelligent robots.