Received May 30, 2019, accepted July 5, 2019, date of publication July 15, 2019, date of current version July 31, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2928467

Graph Theory-Based Approach to Accomplish Complete Coverage Path Planning Tasks for Reconfigurable Robots

KU PING CHENG[1], RAJESH ELARA MOHAN[1], NGUYEN HUU KHANH NHAN[2], AND ANH VU LE[1,2]

[1]ROAR Laboratory, Engineering Product Development Pillar, Singapore University of Technology and Design, Singapore 487372
[2]Optoelectronics Research Group, Faculty of Electrical and Electronics Engineering, Ton Duc Thang University, Ho Chi Minh City 700000, Vietnam

Corresponding author: Nguyen Huu Khanh Nhan (nguyenhuukhanhnhan@tdtu.edu.vn)

This work was supported by the National Robotics Program Office (NRPO), Singapore, to the Engineering Product Development at the Singapore University of Technology and Design under Grant RGAST1702.

ABSTRACT Extensive studies regarding complete coverage problems have been conducted, but a few tackle scenarios where the mobile robot is equipped with reconfigurable modules. The reconfigurability of these robots creates opportunities to develop new navigation strategies with higher dexterity; however, it also simultaneously adds in constraints to the direction of movements. This paper aims to develop a valid navigation strategy that allows tetromino-based self-reconfigurable robots to perform complete coverage tasks. To this end, a novel graph theory-based model to simulate the workspace coverage and make use of dynamic programming technique for optimal path searching and adaptive robot morphology shifting algorithms is proposed. Moreover, the influence of algorithms starting variables on workspace coverage outcome is analyzed thoughtfully in this paper. The simulation results showed that the proposed method is capable of generating navigation paths throughout the workspace, which ensures complete workspace coverage while minimizing the total number of actions performed by the robot.

INDEX TERMS Complete coverage path planning, self-reconfigurable robots, graph theory, dynamic programming, Dijkstra algorithm.

I. INTRODUCTION

Complete Coverage Path Planning (CCPP) algorithms focus on the task of determining a path that passes through every region in the workspace while avoiding obstacles. These algorithms have been extensively studied and many have been integrated to a wide range of real-world robotic platforms, such as cleaning robots [1], painter robots [2], demining robots [3], [4], lawnmowers [5], [6], and so forth.

Based on whether any prior knowledge regarding the workspace is being stored in the system, coverage path planning algorithms can be categorized into online (or called sensor-based) approaches and off-line approaches. Online approaches rely on data feed from onboard sensors on the robot to construct environment maps and to direct the coverage operation [7]. These approaches focus on coverage navigation tasks of unknown spaces and decisions of robot actions are being made at each time instant based on robot surroundings, making these algorithms powerful dealing with workspaces with the presence of dynamic obstacles. Nevertheless, due to the limitations of sensor readings in terms of sensor reach and accuracy, an optimal solution with complete coverage of the workspace is not always guaranteed [7]. Off-line approaches, on the other hand, assume that workspaces are static and fully observable, implying that the path planner can be executed and the navigation path can be generated in advance. Generally, off-line approaches yield solutions with better workspace coverage and more optimized paths, but the approaches may be unrealistic to be put into practice if space is unknown. Popular CCPP algorithms that have been developed previously include spanning-trees [8], [9], spiral filling paths [10], [11], and neural networks [12]. The genetic algorithms [13], [14] are powerful meta-heuristic approaches that excel at searching for the shortest path that

The associate editor coordinating the review of this manuscript and approving it for publication was Vivek Kumar Sehgal.

2169-3536 © 2019 IEEE. Translations and content mining are permitted for academic research only.
Personal use is also permitted, but republication/redistribution requires IEEE permission.
See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

fulfills certain conditions. However, the nature of complete coverage optimization problems is quite different from the shortest path optimization. The objective functions in shortest path optimization problems are usually continuous functions and will eventually converge to particular optima (whether it is local or global). However, in complete coverage path planning problems as presented in this paper, the maximum coverage is a harsh constraint to the objective functions and heuristic approaches does not work well in this situation as the result of the objective function is easily influenced by single changes in robot agent action sequence. This is the reason we have provided an alternative approach to deal with this particular optimization problem. Under the assumption that all algorithms can achieve maximum workspace coverage, the efficiency of a CCPP algorithm can be evaluated by the time elapsed [15], or by the total power consumption throughout the navigation process [16].

Space decomposition technique is a crucial element of many of the CCPP algorithms. Selecting an adequate space decomposition technique simplifies the construction of the system model and may significantly reduce the computational complexity of the algorithms to be implemented. Among all decomposition techniques, grid-based decomposition is a popular candidate that has been adopted in the spiral filling path, genetic algorithms, and some heuristic-based coverage algorithms [17]. Grid-based decomposition methods represent the free space as a union of smaller regions called cells, where all cells are identical in size and shape without any overlapping area between cells. Rectangular cells are commonly used for most navigation applications, while triangular cells [18] are sometimes adopted for flexible robot platforms to operate at higher efficiency. The cell sizes chosen for grid-based methods determine the resolution of the map. A high-resolution grid map provides a better estimation of workspace and obstacle boarders and yields a higher workspace coverage as it allows the robot to navigate to free spaces that could potentially be recognized as obstacle cells in a low-resolution grid map. In most CCPP task scenarios, the grid size of a cell is approximately equal to the sweeping width of the robot for better overall workspace coverage.

Since the early 1980s, reconfigurable robots have received increasing attention and platforms with a wide variety of reconfigurable mechanics have been deployed. Reconfigurable robot platforms can be categorized into three major types [19]: intra-reconfigurable, inter-reconfigurable, and nested reconfigurable. An intra-reconfigurable robot has the ability to change its internal morphology without the requirement of external assembly or disassembly. An inter-reconfigurable robot consists of a congregation of homogeneous or heterogeneous robots and is capable of forming a variety of morphologies through assembly and disassembly process. A nested reconfigurable robot involves platforms that are capable of performing inter-reconfigurations with its individual modules being intra-reconfigurable. The high dexterity and dynamic flexibility reconfigurable robots allow them to accomplish a wide variety of tasks under controlled environments.

Nevertheless, few currently existing reconfigurable robots are designed to tackle complete area coverage tasks. The primary concern to develop an intra-reconfigurable platform suitable to accomplish CCPP tasks is regarding the robustness of the robot. A large portion of CCPP tasks, such as lawn mowing, harvesting, and demining, usually involve a larger workspace with few obstacles. A heavy-duty robot with a fixed shape is considered a better candidate compared to reconfigurable robots in those environments. The planned paths in these scenarios are simple, so implementing reconfigurability in the robots by trading off robustness for motion dexterity does not yield better outcome in these scenarios. On the other hand, CCPP tasks that emphasize area coverage in complicated areas, like indoor cleaning missions, would require robot platforms with higher dexterity to avoid obstacles scattered within the environment and to provide precise motion and direction control. Hinged-tetro (or hTetro) developed by Prabakaran *et al.* [20] is an example of a cleaning robot equipped with the reconfigurability to shape-shift into several transformations. The team presented a tiling theory-based algorithm [21] to demonstrate the feasibility of covering an area by utilizing several morphologies of a similar reconfigurable robot platform, hTetro. With the shape-shifting ability, the hTetro robot is capable of accessing narrow areas within the workspace. High levels of area coverage performance can be observed in the experimental results demonstrated in the aforementioned paper. This paper expands on the previous work of hTetro and focuses on the implementation of coverage path planning algorithms of the platform.

To construct a valid path planning strategy for reconfigurable robots, we put the emphasis on graph theory-based CCPP algorithms. Extensive graph theory-based searching algorithms have been developed for robot platforms, most of which focus on shortest path problems and make use of heuristics like Randomized Search [22], A* Algorithms [23], D* Lite [24], etc. To achieve maximum area coverage, however, would require fundamentally different approaches. A valid strategy is to formulate the problem as the longest path problem (LPP). Solutions generated by LPP simple path algorithm can achieve maximum area coverage provided that every edge in the graph has a positive weight. However, it is worth mentioning that LPP is an NP-Complete class problem as it is trivially a generalization of the Hamiltonian path problem. LPP has been proven to be unsolvable in polynomial time unless $P = NP$ [25], [26]. Due to the computational complexity that lies in the nature of LPP problems, it is not the main concern of this paper to propose an algorithm that outperforms current existing LPP algorithms while yielding an optimal solution. Instead, the goal is to propose a reasonable fast path searching approach that can successfully navigate the robot from the start configuration to the end configuration while ensuring maximum area coverage of the space. To achieve this goal, approaches including graph partitioning and dynamic programming are being implemented to simplify the problem and to speed up the computational time.

Rest of the paper is organized as follows. Section II describes the reconfigurable robot platform that is being modeled. Section III develops the graph theory-based model and formulates the complete coverage problem. In Section IV, the proposed complete coverage path planning algorithm for reconfigurable robots is being presented. Section V shows the simulations and results of the proposed algorithm. Finally, Section VI presents the conclusions along with a note of future developments.

II. HINGED-TETRO PLATFORM

This section introduces the robot platform selected for the system and presents the system model setup of the proposed complete coverage path planning algorithm.

A. HTETRO ROBOT PLATFORM

The workspace $\mathcal{W} \subset \mathbb{R}^2$ is the environment in 2-D Cartesian space where robot agent $\mathcal{A}$ navigates. The reference frames of $\mathcal{W}$ and $\mathcal{A}$ are denoted as $\mathcal{F}^{\mathcal{W}}$ and $\mathcal{F}^{\mathcal{A}}$ [27].

This paper considers hTetro [20] as the selected robot agent, which is a chain-type modular self-reconfigurable (MSR) floor cleaning robot that consists of four blocks connected by three active hinges. The geometries of the four hTetro blocks are denoted as $\mathcal{B}_1, \mathcal{B}_2, \mathcal{B}_3, \mathcal{B}_4$. All hTetro blocks are in a square shape of width d_{block}.The hinge relative connections between hTetro blocks are shown in Figure 1, which results in the following mechanical movement constraints:

$$0 \leq \theta^{\mathcal{B}_1} - \theta^{\mathcal{B}_2} \leq \pi$$
$$0 \leq \theta^{\mathcal{B}_2} - \theta^{\mathcal{B}_3} \leq \pi$$
$$0 \leq \theta^{\mathcal{B}_4} - \theta^{\mathcal{B}_3} \leq \pi$$

where $\theta^{\mathcal{B}_n}(n = 1, \ldots, 4)$ represents the angle rotated from workspace frame to local frame of $\mathcal{B}_n$, with the convention of counterclockwise rotation as the positive direction.

In the proposed model, robot local frame $\mathcal{F}^{\mathcal{A}}$ is being attached to the center of the second block in hTetro ($\mathcal{B}_2$). Consider all possible angle combinations of $\theta^{\mathcal{B}_n}$ that fulfills hinge constraints of $\theta^{\mathcal{B}_n} \in \{0, \frac{pi}{2}, \pi, \frac{3\pi}{2}\}$ while $\mathcal{F}^{\mathcal{A}}$ is fixed, a total of seven robot shapes can be configured. These shapes form the seven basic morphologies of hTetro, as shown in Figure 2. The ability to shape-shift into any of the seven tetromino morphologies allows the hTetro robot to efficiently navigate with the ideal shape according to the perceived terrain and obstacles.

Each hTetro Block is equipped with four omnidirectional wheels, with a pair of wheels being placed perpendicularly to the other pair. This mechanical design allows a hTetro block to instantly change its direction of motion by 90. Differential wheeled robots, on the other hand, are required to perform a U-turn to conduct a direction change in robot motion.

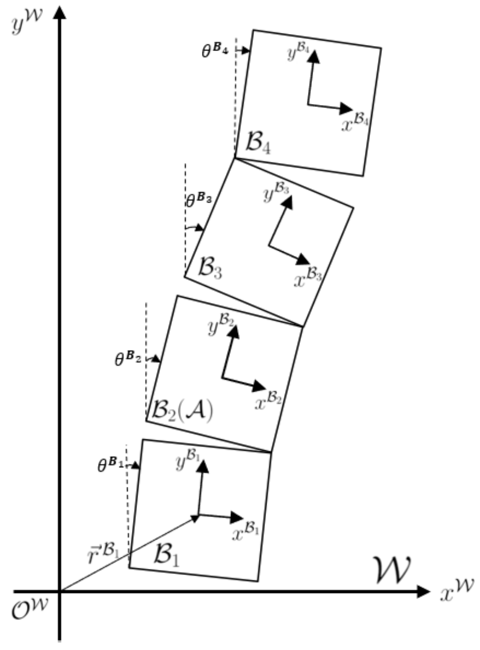


FIGURE 1. hTetro system model.

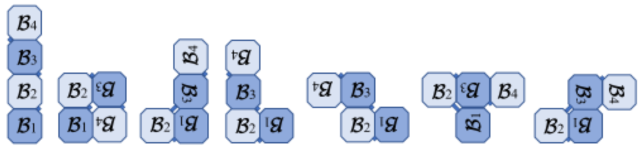


FIGURE 2. 7 basic morphologies of hTetro.

Therefore, instead of controlling the revolutions per minute (RPM) values of each motor in hTetro blocks, the commands implemented to control hTetro block linear motions are simply 'Forward', 'Backward', 'Left', 'Right'. Since a hTetro robot is composed of four different blocks, the combinations of the four commands sent to each block allow them to perform motions with high complexity cooperatively, such as making pivot turns or performing orientation auto-correction when a deviation in hTetro block heading is detected.

B. ROBOT CONFIGURATION

Most robots with fixed shapes describe their configurations with three parameters: the x and y coordinates in the workspace, and a heading angle. Nevertheless, due to the reconfigurable nature of hTetro robots, this representation is insufficient to describe shape-shifting motions and the different morphologies of an hTetro robot in the environment. Therefore, the revised definition of robot configuration is being presented as follow.

Definition 1 (Robot Configuration): The configuration $\mathbf{q}$ of a hTetro robot is a six-element array

$$\mathbf{q} = [x, y, \theta^{\mathcal{B}_1}, \theta^{\mathcal{B}_2}, \theta^{\mathcal{B}_3}, \theta^{\mathcal{B}_4}]^T \quad (1)$$

where:

$$
\begin{aligned}
(x, y) &= \text{coordinate of hTetro Block 2 } (\mathcal{B}_2) \text{ center in workspace frame } \mathcal{F}^{\mathcal{W}} \\
\theta^{\mathcal{B}_k} &= \text{angle rotation of local frame in } \mathcal{B}_n (n = 1, \dots, 4) \text{ with respect to the global frame}
\end{aligned}
$$

C. WORKSPACE MODEL

In this paper, the grid-based method is being implemented to formulate the mathematical system model for our algorithm. Through approximate cellular decomposition technique proposed by Choset [7], a collection of uniform grid cells in the workspace can be determined, where each grid contains variables stating whether space is free or being occupied by obstacles [28]. This section introduces the construction of the grid map of the workspace and the variable that stores grid information.

This work considers a rectangular-shaped workspace $\mathcal{W}$ that could be fully decomposed into square-shaped grids with grid width d_{grid}. Let n_{row} and n_{col} be the total number of rows and columns after the cellular decomposition. Grid position is then defined as follow.

Definition 2 (Grid Position): A grid position represents the coordinate vector of the grid which locates at i-th row and j-th column. $\mathbf{g}_{i,j}^{\mathcal{W}}$ is the grid position with respect to the workspace frame $\mathcal{F}^{\mathcal{W}}$; whereas $\mathbf{g}_{i,j}^{\mathcal{B}_n}$ represents the grid position with respect to the frame where hTetro block $\mathcal{B}_n$ locates.

$$\mathbf{g}_{i,j}^{\mathcal{W}} = \left[x_{i,j}^{\mathcal{W}} \; y_{i,j}^{\mathcal{W}} \right]^T \tag{2}$$

$$\mathbf{g}_{i,j}^{\mathcal{B}_k} = \left[x_{i,j}^{\mathcal{B}_k} \; y_{i,j}^{\mathcal{B}_k} \right]^T = R(\theta^{\mathcal{B}_k}) T(\mathbf{v}^{\mathcal{B}_k}) \mathbf{g}_{i,j}^{\mathcal{W}} \tag{3}$$

where:

$$
\begin{aligned}
R(\theta) &= \text{a 2-dimensional rotation matrix rotating through angle } \theta \text{ counterclockwise about the origin.} \\
T(\mathbf{v}) &= \text{a 2-dimensional translation matrix along vector } \mathbf{v}.
\end{aligned}
$$

Since the workspace frame is fixed throughout the experiment, $\mathbf{g}_{i,j}^{\mathcal{W}}$ will remain constant whereas the value of $\mathbf{g}_{i,j}^{\mathcal{B}_k}$ will constantly be changing.

In the proposed model, obstacles are being introduced in the workspace $\mathcal{W}$. Through approximate cellular decomposition, all grids with overlapping areas with the interior of obstacles within the workspace form an obstacle set $\mathbf{O}$.

With the grid model being constructed, a variable is being introduced to store the grid information at each time instance, which is called "grid activity". The activity of a grid keeps track of whether the obstacle is and the coverage of the grid at each time instance, which is defined as follow.

Definition 3 (Grid Activity): The activity of a grid located at i-th row and j-th column at time t is represented by $a_{i,j}(t)$. A grid activity set $\mathbf{A}$ consists of all grid activities within the workspace. The grid activity is updated at each time instance based on the previous grid activity value, which is defined by:

$$a_{i,j}(t+1) = \begin{cases} 1 & , \text{ if } a_{i,j}(t) = 1 \text{ or } \exists \mathbf{g}_{i,j}^{\mathcal{B}_k}, k \in \{1, \dots, 4\} \\ & \quad \text{s.t. } |x_{i,j}^{\mathcal{B}_k}| \leq \frac{d_{grid}}{2} \wedge |y_{i,j}^{\mathcal{B}_k}| \leq \frac{d_{grid}}{2} \\ -1 & , \text{ if } \mathbf{g}_{i,j}^{\mathcal{W}} \in \mathbf{O} \\ 0 & , \text{ otherwise} \end{cases} \tag{4}$$

Based on Definition 3, all grids with obstacle presenting will have grid activity of -1; whereas the grid activity of other grids remains 0 until it is being covered by any of the hTetro blocks. Once a grid is being covered, the grid activity will be a constant number of 1 throughout the entire navigation process. The workspace $\mathcal{W}$ is considered fully covered at time t if all grids have a grid activity of 1 or -1, providing that no unaccessible grids are presented in the workspace.

D. HTETRO ROBOT NAVIGATION STRATEGY

This paper proposes a robot navigation strategy based on roadmap method. A roadmap $\mathcal{R}$ consists of a series of ideal robot configurations $\mathbf{q}$, which specifies the desired position, heading, and shape of hTetro robot in a particular sequence that results in maximum area coverage of the workspace. The calculation and optimization of the maximum area covered are achieved by utilizing graph theory-based path planner, which will be introduced in section III and IV.

Once a roadmap is being constructed, the navigation system will estimate the positions and heading angles of each hTetro block based on onboard sensor readings. A series of commands which include linear motions in four directions and adjustments in hinge angles will then be sent to each hTetro block in order to clear the configurations assigned to the robot. A configuration is considered cleared once the robot arrives at the coordinate with exact hTetro block angles specified. A new configuration will then be assigned to the robot, and the navigation process will continue until all configurations in the roadmap are being cleared.

In order to reduce unblocked areas that could potentially be identified as obstacle grids during the grid decomposition process, the minimal grid size is defined to match the size of a hTetro block ($d_{grid} = d_{block}$), which provides the highest resolution of the grid map with each grid being geometrically coverable by hTetro blocks. The hTetro robot can perform either linear motion or angle adjustment within each time step. Performing a linear motion moves all hTetro blocks simultaneously in one of the four directions for a grid length d_{grid} with respect to the workspace frame $\mathcal{F}_{\mathcal{W}}$. An angle adjustment of a block changes the orientation of the block by 90 degrees with respect to the blocks' reference frames. In the hTetro model, the robot frame $\mathcal{F}_{\mathcal{A}}$ is attached to the second block, indicating that block 1 and block 3 will be taking block 2 as reference frame and rotate 90 degrees within a time step, block 4 will be taking block 3 as its reference frame, and

TABLE 1. hTetro configuration command table.

Actions	String command	Configuration command $\mathbf{q}_c$
Stop	'S'(stop)	$[0\ 0\ 0\ 0\ 0\ 0]^T$
Linear Motion	'F'($\mathcal{F}^{\mathcal{A}}$ forward translation)	$[0\ d_{grid}\ 0\ 0\ 0\ 0]^T$
	'B'($\mathcal{F}^{\mathcal{A}}$ backward translation)	$[0\ -d_{grid}\ 0\ 0\ 0\ 0]^T$
	'L'($\mathcal{F}^{\mathcal{A}}$ left translation)	$[-d_{grid}\ 0\ 0\ 0\ 0\ 0]^T$
	'R'($\mathcal{F}^{\mathcal{A}}$ right translation)	$[d_{grid}Fd\ 0\ 0\ 0\ 0\ 0]^T$
Rotation / Shape-shift	'1r'($\mathcal{F}^{\mathcal{B}_1}$ -90 rotation),	$[0\ 0\ -\pi/2\ 0\ 0\ 0]^T$
	'1l'($\mathcal{F}^{\mathcal{B}_1}$ 90 rotation),	$[0\ 0\ \pi/2\ 0\ 0\ 0]^T$
	'2r'($\mathcal{F}^{\mathcal{B}_2}$ -90 rotation),	$[0\ 0\ 0\ -\pi/2\ 0\ 0]^T$
	'2l'($\mathcal{F}^{\mathcal{B}_2}$ 90 rotation),	$[0\ 0\ 0\ \pi/2\ 0\ 0]^T$
	'3r'($\mathcal{F}^{\mathcal{B}_3}$ -90 rotation),	$[0\ 0\ 0\ 0\ -\pi/2\ 0]^T$
	'3l'($\mathcal{F}^{\mathcal{B}_3}$ 90 rotation),	$[0\ 0\ 0\ 0\ \pi/2\ 0]^T$
	'4r'($\mathcal{F}^{\mathcal{B}_4}$ -90 rotation),	$[0\ 0\ 0\ 0\ 0\ -\pi/2]^T$
	'4l'($\mathcal{F}^{\mathcal{B}_4}$ 90 rotation),	$[0\ 0\ 0\ 0\ 0\ \pi/2]^T$

block 2 will be taking the workspace frame as reference frame instead. For angle adjustments that cannot be finished within a single time step, the process will continue until the heading angles of all blocks are identical to those specified in the next robot configuration. Combining the linear motions and orientation adjustments hTetro can perform, a total of 13 commands are being defined to control the movements of the platform, which forms a configuration command table as shown in Table 1. A configuration command array set $\mathbf{Q}_c$ consists of several configurations commands $\mathbf{q}_c$ that correspond to specific input string commands. The robot configuration at the next time instance can be updated based on the command received and the current configuration through the equation below:

$$\mathbf{q}(t+1) = \mathbf{q}(t) + \mathbf{q}_c \tag{5}$$

III. GRAPH MODEL OF COMPLETE COVERAGE PROBLEM

Searching algorithms have been extensively studied in graph theory. By modeling the system as a graph, an optimal solution is guaranteed to be found, though the problem might not be solvable within polynomial time [26]. Applying heuristics to the problem has been a common approach to tackle graph-based problems; however, to reduce the time consumed to solve the problem, some of these approaches might make compromises on the accuracy of the solution [29]. The approach this paper proposed creates partitions of the workspace graph based on heuristics and performs exhaustive searches within the partitioned subgraphs to ensure accuracy and to accomplish full coverage of the area.

This section formulates the graph model of the workspace based on the hTetro model developed in section II. The definitions of morphology layer sets and stripe layer sets that are used to construct the graph model is then introduced. An auxiliary graph will then be constructed based on the graph partitioning results, and searching algorithms will be implemented to generate the optimal path to traverse within the stripe layer sets. The details of the navigation

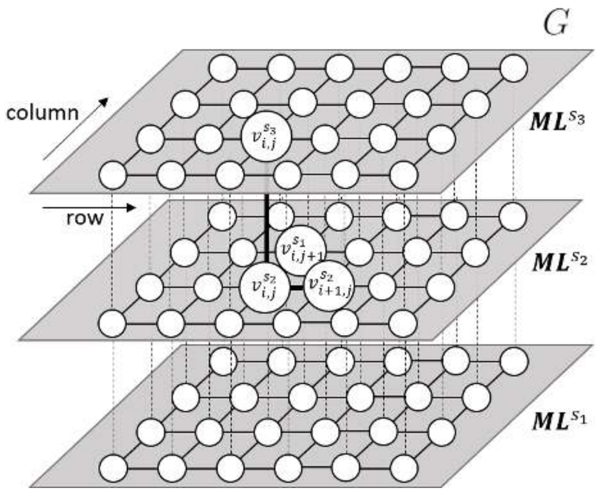


FIGURE 3. An illustration of hTetro workspace graph *G*.

algorithms within stripe layer sets are further introduced in section IV.

A. CONSTRUCTION OF HTETRO WORKSPACE GRAPH

A morphology layer set is being defined to construct a graph theory based representation of the workspace. A morphology layer set specifies the vertices where the hTetro robot with certain morphology can navigate. The hTetro workspace graph, consisting of several morphology layer sets, in then defined. The definitions are shown as follow.

Definition 4 (Morphology Layer Set): A morphology layer set $\mathbf{ML}^s$ is a set that consists of a total of $n_{row} \times n_{col}$ elements:

$$\mathbf{ML}^s = \{v^s_{i,j} | i, j \in \mathbb{N}, i \leq n_{row}, j \leq n_{col}\} \tag{6}$$

where:

$s = (\theta^{\mathcal{B}_1}, \theta^{\mathcal{B}_2}, \theta^{\mathcal{B}_3}, \theta^{\mathcal{B}_4})$ a tuple with four heading angles which represents a specific hTetro morphology.

$v^s_{i,j} =$ vertex that correspond to the grid at i-th row and j-th column in morphology s

Definition 5 (hTetro Workspace Graph): A hTetro workspace graph G is defined as a weighted graph with vertex set $\mathbf{V}$, edge set $\mathbf{E}$, and morphology set $\mathbf{S}$. A morphology set $\mathbf{S}$ represents the set of all hTetro morphologies s that are allowed throughout the navigation. $\mathbf{V}$ and $\mathbf{E}$ can be written as:

$$3\mathbf{V} = \Big(\bigcup_{s \in S} \mathbf{ML}^s\Big)$$

$$\begin{aligned}\mathbf{E} &= \{(v^s_{i,j}, v^{s'}_{i,j}) | s, s' \in \mathbf{S}, s \neq s', i, j \in \mathbb{N}, i \leq n_{row}, j \leq n_{col}\} \\ &\cup \{(v^s_{i,j}, v^s_{i+1,j}) | s \in \mathbf{S}, i, j \in \mathbb{N}, i \leq n_{row} - 1, j \leq n_{col}\} \\ &\cup \{(v^s_{i,j}, v^s_{i,j+1}) | s \in \mathbf{S}, i, j \in \mathbb{N}, i \leq n_{row}, j \leq n_{col} - 1\}\end{aligned}$$

Figure 3 illustrates the definition of a workspace, which consists of three morphology layer sets ($\mathbf{ML}^{s_1}$, $\mathbf{ML}^{s_2}$, $\mathbf{ML}^{s_3}$).

The vertices of the workspace graph are the union of vertices in all available morphology layer sets. The edge connections within a morphology layer set represent possible robot linear movements within the workspace with fixed robot morphology. In Figure 3, the edge connectivity of vertex $v_{i,j}^{s2}$ is being highlighted. The linear motion is limited to four different directions, while a connection between two morphology layers represents a shift in robot morphology with the grid position being fixed.

Assuming that the total number of morphologies in $\mathbf{S}$ is set to n_{shape}, the workspace graph will consist of a total of $n_{row} \times n_{col} \times n_{shape}$ vertices, which makes exhaustive search approach throughout the entire graph unpractical; therefore, graph partitioning and dynamic programming techniques are introduced to simplify the problem.

B. GRAPH PARTITIONING AND DYNAMIC PROGRAMMING

This subsection introduces the attempt to utilize graph partitioning method to separate the graph into several stripes. Graph partitioning is a commonly used algorithmic operation that significantly reduces the time complexity of a graph [30] and is a crucial prerequisite for efficient large-scale parallel graph algorithms [31].

The core idea to implement graph partitioning method to the hTetro workspace graph is to divide it into several "stripe layer subgraphs". Each stripe layer subgraph covers a small portion of the workspace area, where the recursive backtracking algorithm is implemented to search for a path that covers the subgraph area. The details of the recursive backtracking algorithm are introduced in section IV. Assuming that all grids in a striped layer subgraph are covered and the action costs of all stripe layers have been calculated, one remaining task of the algorithm is to provide an optimized path between the stripe layer subgraphs such that the total action cost for the overall area coverage mission is minimized. The optimization problem is being solved by implementing Dijkstra searching algorithm in our algorithm.

The stripe layer set and stripe layer graph in the graph partition model is defined as follow.

Definition 6 (Stripe Layer Set): A stripe layer set $\mathbf{SL}_k$ is a set that consists of all vertices in the k-th stripe layer set, with a total of $n_{col} \times n_{str_k}$ elements:

$$\mathbf{SL}_k = \{v_{i,j} | i, j \in \mathbb{N}, i \leq n_{col}, \sum_{n=1}^{k-1} n_{str_k} < j \leq \sum_{n=1}^{k} n_{str_k}\} \tag{7}$$

where:

n_{str_k} = number of columns within stripe layer k. ($n_{str_k} \in \mathbb{N}$, $\sum_k n_{str_k} = n_{col}$)

Definition 7 (Stripe Layer Graph): A stripe layer graph SLG_k is a vertex-induced subgraph of G, which shares the vertex set $\mathbf{SL}_k$ and all corresponding edges in workspace graph G.

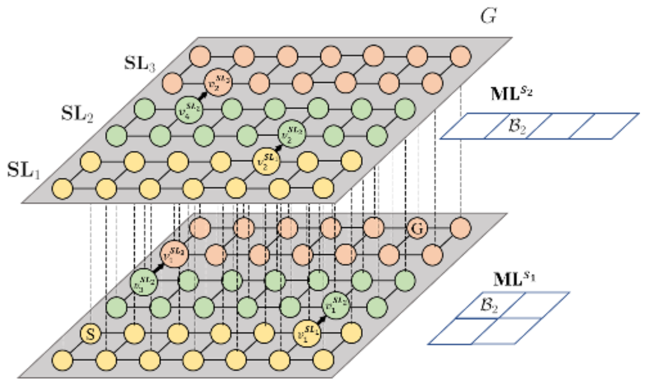


FIGURE 4. **hTetro workspace graph *G* with O- and horizontal I-shape morphology layer sets. The cut-edges between stripe layer sets are being highlighted.**

In the proposed algorithm, reasonable values chosen for stripe column widths n_{str} are between 2 to 4, which creates stripes with similar size with the hTetro robot. When the robot navigates within in a stripe layer subgraph, due to the constrained column direction movement, the robot will generally be moving in either $+x^{\mathcal{W}}$ or $-x^{\mathcal{W}}$ direction. The shortest path that connects all stripe paths can be found when the positive direction and negative direction stripes are placed alternately, forming a boustrophedon-patterned motion [32]. The robot will navigate to the next stripe layer subgraph once all accessible grids in the current subgraph are being covered. The stripe layers are being connected by one or several directed cut-edges. Within a morphology layer set, each transition between stripe layer subgraphs will be assigned a cut-edge to provide the robot with the flexibility to navigate to the following stripe layer with different robot morphologies.

Take the scenario demonstrated in Figure 4 for example, which illustrates a simple workspace graph partitioned into three stripe layer sets $\mathbf{SL}_k (k = 1, \ldots, 3)$ with different colors. In the example, graph morphology set $\mathbf{S}$ consists of two elements: O-shape ($s_1 = (0, 0, -\pi, -\pi)$) and horizontal I-shape ($s_2 = (-\pi/2, -\pi/2, -\pi/2, -\pi/2)$). During the initialization of the algorithm, the start configuration (denoted as $\mathbf{q}_S$), goal configuration (denoted as $\mathbf{q}_G$), and the partitions of the stripes have to be determined. In Figure 4, the start and goal configurations are being marked as the letter 'S' and 'G' with O-shaped starting morphology. The algorithm will generate a path from $\mathbf{q}_S$ that covers the area under the first stripe layer before entering one of the cut-edges that connects to the second stripe layer, which is either $(v_1^{\mathbf{SL}_1}, v_1^{\mathbf{SL}_2})$ or $(v_2^{\mathbf{SL}_1}, v_2^{\mathbf{SL}_2})$ in Figure 4. The row positions of cut-edges might differ between morphologies based on the position of the second block $\mathcal{B}_2$ of hTetro. For instance, in the first morphology layer (O-shape) of the presented scenario, the cut-edge is located at the second-to-last row since placing it at the last row will result in several hTetro blocks going beyond the workspace boundary. The idea also applies to all other morphology layers. Within a morphology layer, only a single cut-edge will be formed to connect two different stripes layers.

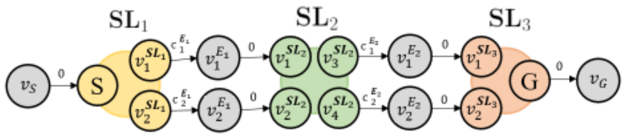


FIGURE 5. A quotient graph of hTetro workspace graph in Figure 4 after graph partitioning.

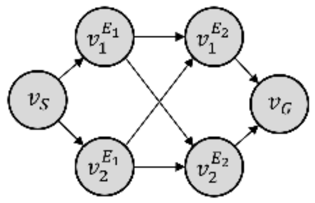


FIGURE 6. Auxiliary graph of Figure 4. Each vertex v corresponds to the auxiliary vertex in Figure 5 which stands for a connection between two subgraphs.

With the workspace graph defined, we now focus on the simplification of the computational complexity of the problem through dynamic programming, which is a commonly used algorithmic paradigm for approaching a complex problem by breaking it into several subproblems and make use of the memoization technique to cache the results of subproblems and directly reuse them when the same computation is required again [33]. In the proposed model, the coverage tasks within each stripe layer subgraphs are the subproblems; while the minimization of the number of actions required is the main problem. To prevent re-calculation of the cost required to cover each stripe layer, the total action cost of each stripe layer will be memorized by the dynamic programming scheme. It is essential to take note that the total action cost is not guaranteed to be constant within each stripe layer due to the combinations of selected start and goal vertices. Different start and goal vertex not only represent different hTetro morphologies but might also suggest different geometry positions of the second block $\mathcal{B}_2$ of hTetro in the workspace. To tackle the aforementioned issue, the concept of the auxiliary vertex is introduced and defined as follow.

Definition 8 (Auxiliary Vertex): An auxiliary vertex $v_n^{E_k}$ is a vertex that is introduced between the $n-$th cut-edge $(v_n^{\mathbf{SL}_k}, v_n^{\mathbf{SL}_{k+1}})$ that runs between stripe layer sets $\mathbf{SL}_k$ and $\mathbf{SL}_{k+1}$. The newly formed edges that replace the original cut-edge will become $(v_n^{\mathbf{SL}_k}, v_n^{E_k})$ and $(v_n^{E_k}, v_n^{\mathbf{SL}_{k+1}})$.

Figure 5 demonstrates a quotient graph which illustrates the auxiliary vertices and their connections between each stripe layer sets. The three different colors represent the three stripe layer sets in Figure 4. To minimize the total cost of the entire navigation process, the costs of all stripe layer sets and cut-edges have to be known. The definition of the costs is as follow.

Definition 9 (Cut-Edge Cost): A cut-edge cost $c_n^{E_k}$ is the weight of cut-edge $(v_n^{\mathbf{SL}_k}, v_n'^{\mathbf{SL}_{k+1}})$.

Definition 10 (Stripe Layer Set Cost): A stripe layer set cost $c^{\mathbf{SL}_k}(A \rightarrow B)$ is the total action cost for the robot to travel from vertex A to vertex B in stripe layer set $\mathbf{SL}_k$. For the first stripe layer set, the starting vertex is considered as 'S', and the goal vertex in the last stripe layer is denoted as 'G'. The value of the stripe layer set cost is calculated by function GetStripeLayerCost in Algorithm 5, which will be introduced in section IV-C.

Since a cut-edge is separated into two edges after auxiliary vertices are introduced, the original cut-edge cost $c_n^{E_k}$ will be shared by the two edges, with one of them inheriting the original cut-edge cost while the cost of the other edge is set to 0. In Figure 5, two extra vertices, v_S and v_G, are being introduced. Vertex v_S is attached to the original starting vertex; while vertex v_G is attached to the goal vertex, both with an edge cost of 0. All newly introduced vertices in Figure 5 form a directed acyclic graph (DAG) in a higher hierarchy level, which is referred to as an auxiliary graph as shown in Figure 6. The edge weight w of a directed edge $\mathbf{e}$ in the auxiliary graph is the sum of all edge costs that run between the two vertices in Figure 5. The calculation of w takes the path costs of stripe layer sets defined in Definition 10 into account, and the equation can be written as follow.

$$w(\mathbf{e}) = \begin{cases} c^{\mathbf{SL}_1}(S \rightarrow n') + c_n^{E_1} & , \text{ if } \mathbf{e} = (v_S, v_{n'}^{E_1}) \\ c^{\mathbf{SL}_k}(n \rightarrow n') + c_n^{E_k} & , \text{ if } \mathbf{e} = (v_n^{E_k}, v_{n'}^{E_{k+1}}) \\ c^{\mathbf{SL}_k}(n \rightarrow G) & , \text{ if } \mathbf{e} = (v_n^{E_k}, v_G) \end{cases} \tag{8}$$

With the hTetro auxiliary graph model constructed and weights known, Dijkstra algorithm with priority queue [34] is being implemented to calculate the shortest path of the graph as shown in Algorithm 1. The best solution found will determine the shape morphologies and positions for the robot to traverse between stripes that result in the minimum overall action cost. Dynamic programming is introduced to memorize the calculated stripe layer set costs to prevent re-calculations from speeding up the computation process.

With the start and goal hTetro configurations known and the auxiliary graph fully defined, Algorithm 1 is being implemented, and it will return the optimal path, and the corresponding path cost throughout the entire auxiliary graph, which shows the best hTetro morphologies to traverse between the stripe layer sets. However, to construct a full roadmap R for the robot to follow, the paths within the stripe layer graphs are still required.

IV. STRIPE LAYER SUBGRAPH COMPLETE COVERAGE PATH PLANNING

This section introduces a recursive backtracking algorithm that solves the coverage problem within a stripe layer graph SLG_k with start and goal configurations given. During the recursive backtracking searching process, the validity of robot action is checked continuously, and optimization techniques are being implemented to speed up the computation time. Coverage checking criteria are being introduced to ensure maximum coverage of the workspace after the navigation process terminates. The proposed CCPP algorithm

Algorithm 1 Dijkstra Algorithm With Priority Queue and Memoization

```
1: function DijkstraPQ($G$, $v_S$, $v_G$)
2:     Create edge weight table WT for memoization, distance array dist, and path array path
3:     Create priority queue $PQ$, add every vertex $v$ as elements and dist[$v$] as keys to $PQ$.
4:     WT[$v_i$, $v_j$] $\leftarrow \infty$ for all $v_i, v_j \in \mathbb{N}, v_i \leq n_{row}, v_j \leq n_{col}$
5:     dist[$v$] $\leftarrow \infty$ for all vertex $v \in V^G$; dist[$v_S$] $\leftarrow 0$
6:     path[$v$] $\leftarrow [v_S]$ for all vertex $v \in V^G$
7:     while $PQ$ not empty do
8:         $u \leftarrow PQ$.ExtractMin() // Remove and returns the element with smallest key in $PQ$.
9:         for all edges $\mathbf{e} = (v, v')$, $\mathbf{e} \in E^G$ do
10:            $w_e \leftarrow$ LookupWeightTable(WT, $v$, $v'$)
11:            if dist[$v'$] $>$ dist[$v$] $+ w_e$ then
12:                dist[$v'$] $=$ dist[$v$] $+ w_e$
13:                path[$v'$] $=$ path[$v$].Append($v'$)
14:                $PQ$.DecreaseKey($v'$, dist[$v'$]) // Decrease the value of $v'$.key to dist[$v'$].
15:            end if
16:        end for
17:    end while
18:    return {dist[$v_G$], path[$v_G$]}
19: end function
20:
21: function LookupWeightTable(WT, $v$, $v'$)
22:    $v_i, v_j \leftarrow$ row, column value of vertex $v$ in grid map.
23:    if WT[$v_i$, $v_j$] $< \infty$ then
24:        return WT[$v_i$, $v_j$]
25:    else
26:        WT[$v_i$, $v_j$] $\leftarrow w(\mathbf{e})$,where $\mathbf{e} = (v, v')$
27:        return WT[$v_i$, $v_j$]
28:    end if
29: end function
```

Algorithm 2 Complete Coverage Path Planning Algorithm

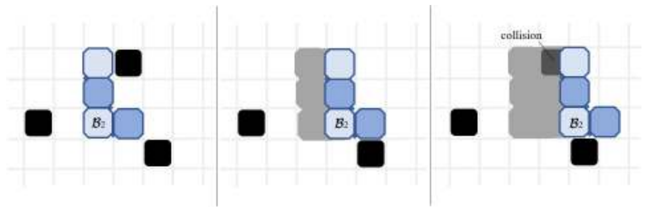
```
Input: Workspace grid map of size $n_{row} \times n_{col}$, stripe column widths $\mathbf{N}_{str}$, viable morphologies set $S$, starting and goal configurations ($\mathbf{q}_S$, $\mathbf{q}_G$).
Output: Roadmap $\mathcal{R}$ that stores all configurations in a path

1. Generate a valid angle adjustment table AAT between all morphologies in $S$ which stores all invalid relative grid positions ($x_{rel}$, $y_{rel}$).
2. Create stripes based on $\mathbf{N}^c$, identify the cut-edges between all stripes, and generate hTetro auxiliary graph $G$.
3. Create following tables for memoization: i) valid action table (VAT), ii) action cost table (ACT), iii) stripe subgraph path table (SPT)
4. Determine start and goal vertices ($v_S$, $v_G$) from start and goal configurations ($\mathbf{q}_S$, $\mathbf{q}_G$).
5. Calculate auxiliary graph {dist, path} $\leftarrow$ DijkstraPQ ($G$, $v_S$, $v_G$) from Algorithm 1.
6. Create empty roadmap $\mathcal{R}$
for all $v \in$ path do
    $\mathcal{R}$.Append(SPT[$v$])
end for
```

will determine the value of stripe layer set cost as defined in Definition 10 while being invoked in Algorithm 1 and will save vertices in the optimal paths into a stripe subgraph path table (**SPT**) for memorization. The roadmap $\mathcal{R}$ will eventually be generated based on the stored path in **SPT**. The pseudocode for the proposed complete coverage path planning algorithm is shown in Algorithm 2.

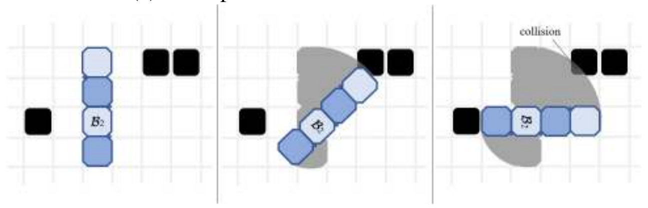
A. ACTION VALIDITY FOR RECONFIGURABLE ROBOTS

A major challenge that lies in the implementation of CCPP algorithms for reconfigurable robots is the modeling process of motion constraints based on different robot morphologies. CCPP algorithms developed for fixed-morphology robots consider only the fixed geometry of the robot modules and their orientations. A common approach for these algorithms to simplify obstacle avoidance tasks in the path planner is to decompose the workspace with the grid size matching the robot size. Assuming that a robot action is considered as "valid" if the assigned action does not result in any collision between the robot and the terrain, the path planning algorithms for fixed-morphology robots would only require the system to examine the clearance of the following grids based on the robot's direction of motion. However, for a reconfigurable robot to perform a valid action, it is crucial to ensure that the geometries of all robot modules during the transition phase of robot actions are not colliding or intersecting with obstacles in the workspace. In the proposed algorithm, the validity of each action that can be performed by the robot is being evaluated and modeled independently and serves as an essential constraint in the recursive backtracking function.

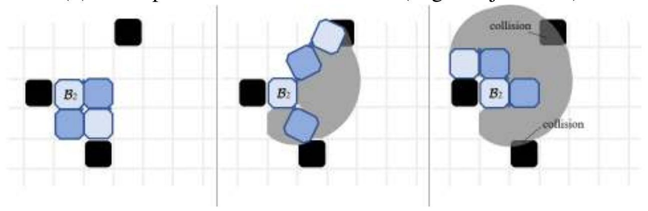
In the case of hTetro, the task is to model the validity of every action listed in the configuration command table (Table 1). The three types of actions in the configuration command table include stop, linear motions, and rotations or shape-shifting. The platform will not be performing any motion once a stop command is being received, so apparently, no extra constraints would be added. Linear motion would translate the four blocks in a specific direction regardless of their individual headings. Since a hTtro block size is identical to the grid size in our model, the algorithm has to check whether any of the four blocks collide with obstacles during the translation process, which can be achieved by simply checking the clearance of the four grids at the goal configuration. For the hTetro platform to accomplish a rotation or shape-shifting, however, will require clearance for extra grids as some of them are being covered by rotating hTetro blocks due to the change in angles of the hinges.



(a) Example of an invalid linear motion



(b) Example of an invalid rotation (angle adjustment).



(c) Example of an invalid shape-shifting (angle adjustment).

FIGURE 7. Examples of invalid hTetro motions. In the figures, the black-colored areas represent obstacles, while gray-colored areas represent the space being swept during the robot motion. Collision occurs whenever the two different colored areas overlap during the transition phase of the action.

Algorithm 3 hTetro Waypoint Navigation Strategy

```
1: function zigzagSeq($WP[], n_{row}, n_{col}, n_{wid}$)
2:     $curX, seq \leftarrow 1$
3:     $dir \leftarrow 1$
4:     $WP_{zigzag}[] \leftarrow (1, 1)$
5:     while $curX < n_{row}$ do
6:         if $dir = 1$ then
7:             $col_i \leftarrow 1; col_f \leftarrow n_{col}$
8:         else
9:             $col_i \leftarrow n_{col}; col_f \leftarrow 1$
10:        end if
11:        for $curY \leftarrow col_i$ to $col_f$ do
12:            for $curW \leftarrow 0$ to $n_{wid} - 1$ do
13:                for all $(R, Q) \in Wp[]$ do
14:                    if $(curX + curW = Q.X \wedge curY = Q.Y)$ then
15:                        $WP_{zigzag}[].push(seq, Q)$
16:                        $seq \leftarrow seq + 1$
17:                    end if
18:                end for
19:            end for
20:        end for
21:        $dir \leftarrow -1 \times dir$
22:        $curX \leftarrow curX + n_{wid}$
23:    end while
24: end function
```

An illustration of the validity of a hTetro action is shown in Figure 7. During the initialization process of the algorithm, the relative positions of all swept grids when these actions are being performed will be calculated and saved in a valid action table **VAT** for memoization; during the path searching process, the **VAT** will be constantly utilized to determine the validity of robot angle adjustment commands.

B. GRAPH PARTITIONING IN STRIPE LAYER GRAPH

The k-th stripe layer subgraph SLG_k is an undirected graph with $n_{row} \times n_{str_k} \times n_{shape}$ vertices, which is a relatively small number compared to number of vertices in hTetro workspace graph. Nevertheless, conducting exhaustive searches in an undirected graph like a stripe layer graph SLG would potentially be time-consuming once the number of rows n_{row} or number of morphologies allowed n_{shape} increases. In order to reduce the time complexity in the algorithm, we propose an approach similar to section III-B, where graph partition and dynamic programming techniques are used to simplify the problem.

When the program initializes, all grids in the workspace are being evaluated utilizing the grid validity check technique introduced previously to find grids that can be only able to be cleared by one or few morphology layers. These configurations are considered as "intermediate configurations" throughout the navigation process within a stripe layer subgraph. By identifying the intermediate configurations set $\mathbf{Q}_I$ in the stripe layer, an auxiliary graph will be constructed similar to Section III-B. Algorithm 1 will then be used to calculate the best path within the stripe layer, which will significantly speed up the computation process since we are only required to focus on optimization problems in graphs with much smaller size. These graphs between the intermediate configurations are referred to as "regions". The realization of the idea is demonstrated in Algorithm 4, where we introduce n_{ver} that represents the total number of vertices that are covered by all morphologies at the same grid. A new intermediate configuration will only be added to $\mathbf{Q}_I$ if the n_{ver} is equal to or smaller than a predetermined number, which is denoted as $n_{ver,max}$. In this paper, $n_{ver,max} = 1$ is chosen, which implies that whenever the best route is found within a region, it will directly become a portion of the navigation path within a stripe layer.

C. RECURSIVE BACKTRACKING

With the navigation setup within a stripe layer determined, a simple backtracking algorithm is being implemented to find a valid path. Backtracking is a modified depth-first search (DFS) algorithm which will perform DFS traversal of the tree and incrementally build candidates. If a non-promising candidate is reached, the candidate will be abandoned while the system backtracks to its state before the decision is made [35]. The proposed recursive backtracking algorithm is demonstrated in Algorithm 5.

Algorithm 4 Determination of Intermediate Configurations

```
1:  function GetIntermediateConfig(S, A)
2:      $\mathbf{Q}_I$, $\mathbf{G}_{vis}$, $\mathbf{G}_{cov}$ $\leftarrow$ [ ]
3:      for all $\{(s, row, col) | s \in \mathbf{S}, row, col \in \mathbb{N}, row \le n_{row}, col \le n_{col}\}$ do
4:          if isGridValid($row, col$, A) then
5:              $\mathbf{G}_{vis}[s, row, col] \leftarrow 1$
6:              for all $g^{\mathcal{B}_k}_{row,col}$, $k = 1, \ldots, 4$ do
7:                  for all $a_{i,j} \in \mathbf{A}$ do
8:                      if $a_{i,j} = 0$ AND (area of hTetro block at $g^{\mathcal{B}_k}_{row,col}$) $\cap$ (area of grid at $\mathbf{g}^{\mathcal{W}}_{i,j}$) $\neq \varnothing$ then
9:                          $\mathbf{G}_{cov}[s, row, col] \leftarrow (i, j)$
10:                     end if
11:                 end for
12:             end for
13:         else
14:             $\mathbf{G}_{vis}[s, row, col] \leftarrow \emptyset$
15:         end if
16:     end for
17:     for all $\{(row, col) | row, col \in \mathbb{N}, row \le n_{row}, col \le n_{col}\}$ do
18:         $n_{ver} \leftarrow 0$
19:         for all $s \in$ S do
20:             if $\mathbf{G}_{cov}[s, row, col] \neq \emptyset$ then
21:                 $n_{ver} \leftarrow n_{ver} + 1$
22:             end if
23:         end for
24:         if $n_{ver} \le n_{ver,max}$ then
25:             $(i, j) \leftarrow \mathbf{G}_{cov}[s, row, col]$
26:             $\mathbf{Q}_I$.Append($(i, j, s)$)
27:         end if
28:     end for
29: end function
```

The recursive backtracking function RBT in the proposed algorithm will loop through all elements within the configuration command table set $\mathbf{Q}_c$ as defined in section II-D. The algorithm will visit nearby nodes based on the commands in $\mathbf{Q}_c$. The weights of the edges are determined based on the number of commands required for the hTetro platform to perform the assigned action. The cost of a linear action is one as only one command is required. However, for the robot to visit vertices in different morphology layer sets, the number of commands in Table 1 required might differ. For instance, for the robot to visit horizontal I-shape ($s = (0, 0, -\pi, -\pi)$) layer set from vertical I-shape ($s = (0, 0, 0, 0)$), a single command '2r' would be sufficient; while to visit O-shape ($s = (0, 0, -\pi, -\pi)$) layer set from vertical I-shape, two consecutive '3r' commands are required for the action to be accomplished. Since the action costs for angle adjustments, which includes rotations and shape-shifting, are determined based on the total number of commands required, formulating the cost function based on robot commands will reflect on the total time taken of the entire navigation process. It is worth noting that defining the cost function based on the energy consumption of actions or other strategies might yield different optimization results.

Algorithm 5 Stripe Layer Cost Determination

```
1:  function GetStripeLayerCost($SLG$, $\mathbf{q}_S$, $\mathbf{q}_G$)
2:      $\mathbf{Q}_I \leftarrow [\mathbf{q}_S, \mathbf{q}_G]$ ; $\mathbf{Q}_{tot} \leftarrow [\,]$ ; $cost_{tot} = 0$ ; $m \leftarrow 1$
3:      $G \leftarrow SLG$
4:      $\mathbf{A} \leftarrow$ grid activities of all vertices in $G$
5:      while $m \neq size(\mathbf{Q}_I) - 1$ do
6:          $c_{cur} \leftarrow 0$; $c_{opt} \leftarrow \infty$; $\mathbf{Q}_{cur}, \mathbf{Q}_{opt} \leftarrow [\,]$
7:          Run RBT function between $\mathbf{Q}_I[m]$ and $\mathbf{Q}_I[m+1]$.
8:          if $\mathbf{Q}_{opt} \neq [\,]$ then
9:              $\mathbf{Q}_{tot}$. Append($\mathbf{Q}_{opt}$) ; $cost_{tot} \leftarrow cost_{tot} + c_{opt}$
10:             $m \leftarrow m + 1$.
11:         else
12:             Check all $a_{i,j} = 0\, \forall a \in \mathbf{A} \wedge g_{i,j} \in G$.
13:             Determine valid $\mathbf{q}$ at grid $(i, j)$ and insert
14:             it to $(m + 1)$-th position of $\mathbf{Q}_I$.
15:             If no valid grid found, $n_{str_k} \leftarrow n_{str_k} + 1$
16:             $G \leftarrow$ subgraph of current region
17:             $\mathbf{A} \leftarrow$ grid activities of all vertices in $G$
18:         end if
19:     end while
20:     $SPT[SLG] \leftarrow \mathbf{Q}_{tot}$ ; $ACT[SLG] \leftarrow c_{tot}$
21:     return $c_{tot}$
22: end function
23:
24: function RBT(G, $\mathbf{q}$, $\mathbf{q}_G$, $\mathbf{A}$, $c_{cur}$, $c_{opt}$, $\mathbf{Q}_{cur}$, $\mathbf{Q}_{opt}$)
25:     for all $\mathbf{q}_c \in \mathbf{Q}_c$ do
26:         if isValidAction($\mathbf{q}$, $\mathbf{q}_c$, $\mathbf{A}$) then
27:             Cache value of $\mathbf{A}$, $c_{cur}$, and $\mathbf{Q}_{cur}$
28:             $t \leftarrow t + 1$ ; $\mathbf{q}' \leftarrow \mathbf{q} + \mathbf{q}_c$
29:             Update grid activity $\mathbf{A}$ with Equation 4.
30:             $\mathbf{Q}_{cur} \leftarrow \mathbf{Q}_{cur}$.Append($\mathbf{q}'$)
31:             $c_{action} \leftarrow$ action cost between $\mathbf{q}$, $\mathbf{q}'$ in Table 1.
32:             $c_{prev} \leftarrow c_{cur} + c_{action}$
33:             if ($\mathbf{q}' = \mathbf{q}_G$) AND ($c_{cur} < c_{opt}$) AND ($a_{i,j} =$
34:             $1 \wedge$ IsGridValid($i, j$, A) $\forall a \in \mathbf{A}$)
35:                 $\mathbf{Q}_{opt} \leftarrow \mathbf{Q}_{cur}$ ; $c_{opt} \leftarrow c_{cur}$
36:             else
37:                 Run RBT function between $\mathbf{q}'$ and $\mathbf{q}_G$.
38:             end if
39:             Restore cached value of $\mathbf{A}$, $c_{cur}$, and $\mathbf{Q}_{cur}$
40:             $t \leftarrow t - 1$
41:         end if
42:     end for
43:     return $\mathbf{A}$, $c_{cur}$, $c_{opt}$, $\mathbf{Q}_{cur}$, $\mathbf{Q}_{opt}$
44: end function
```

The overview of the recursive function that achieves maximum region coverage while minimizing total action cost is demonstrated as follow. The function first iterates through all feasible actions $\mathbf{Q}_c$ that can be taken by the robot. If the selected action is considered as valid in Algorithm 6 and the

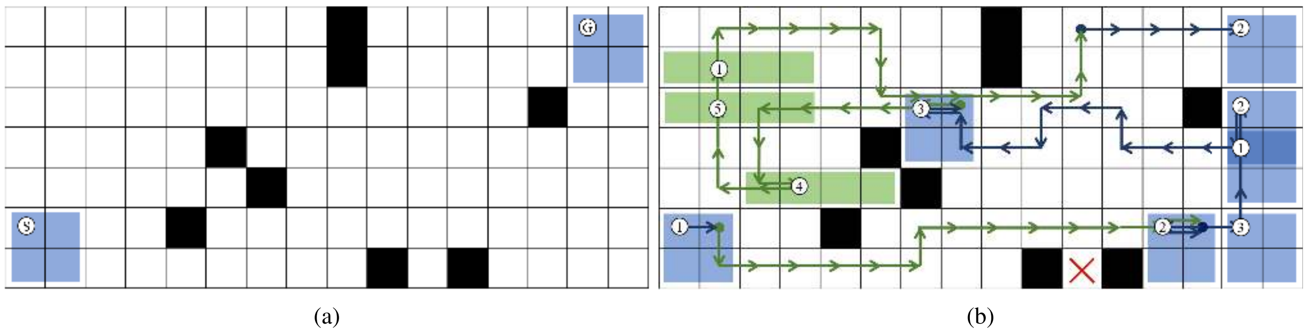


FIGURE 8. A workspace example with the area separated into a 16 × 7 grid map. Figure 8a shows the initial workspace and the start and goal hTetro configurations; while Figure 8b demonstrates the final path generated by the proposed complete coverage algorithm. (Green arrows: horizontal I-shape; blue arrows: O-shape).

next vertex has not yet been visited, the path and action cost will be cached before a new recursive function is called at the next vertex. When all feasible actions are being explored by a vertex, it will backtrack to its parent vertices while undoing the previous actions. If the goal vertex is reached, the coverage of the stripe layer will be calculated to check whether the candidate path fulfills maximum coverage and outperforms previously cached optimal path by having a smaller action cost. If the candidate path fulfills the criteria above, the optimal will be updated, and the algorithm will continue to explore the remaining portion of the tree.

D. ASSURANCE OF COMPLETE AREA COVERAGE

In Algorithm 6, we deliberately introduce the constraint which prevents the robot from revisiting any vertices. The idea is to create a simple path which does not contain repeating vertices so that the searching algorithm will not be stuck in indefinite loops during the search. The disadvantage of the constraint, however, is that several vertices might be unvisitable due to the action limitations. Consider a scenario in which the robot has to move in to cover a narrow area and move out afterward. The robot is unable to perform the simple action mentioned in this scenario due to the restriction of revisiting the same nodes. To tackle this issue, a check will be conducted once the tree in the region has been fully explored. If none of the paths suggests complete area coverage while the areas are reachable by any of the allowed morphologies, an extra intermediate vertex will be added. The vertex will separate the region into two smaller ones in which the searching algorithm resumes. Several vertices will be shared by both regions, which allow a small portion of grids being re-visited throughout the process. In situations where no path exists between the start and goal configurations due to obstacle placements, the stripe column width n_{str} will be gradually increased until a valid path is found.

Algorithm 6 Grid Validity Check

```
1:  function isValidAction($\mathbf{q}$, $\mathbf{q}_c$, $\mathbf{A}$)
2:      if VAT[$\mathbf{q}$, $\mathbf{q}_c$] $\neq \varnothing$ then
3:          return VAT[$\mathbf{q}$, $\mathbf{q}_c$]
4:      end if
5:      $\mathbf{q}' \leftarrow \mathbf{q} + q_c$
6:      if $\mathbf{q}_c[0] \neq 0$ OR $\mathbf{q}_c[1] \neq 0$ then
7:          $(row, col) \leftarrow (\mathbf{q}'[0], \mathbf{q}'[1])$
8:          VAT[$\mathbf{q}$, $\mathbf{q}_c$] $\leftarrow$ isGridValid($row$, $col$, $\mathbf{A}$)
9:          return VAT[$\mathbf{q}$, $\mathbf{q}_c$]
10:     else
11:         for all $(x_{rel}, y_{rel}) \in$ AAT do
12:             $(row, col) \leftarrow (\mathbf{q}[0] + x_{rel}, \mathbf{q}[1] + y_{rel})$
13:             if ! isGridValid($row$, $col$, $\mathbf{A}$) then
14:                 VAT[$\mathbf{q}$, $\mathbf{q}_c$] $\leftarrow$ $false$
15:                 return VAT[$\mathbf{q}$, $\mathbf{q}_c$]
16:             end if
17:         end for
18:         VAT[$\mathbf{q}$, $\mathbf{q}_c$] $\leftarrow$ $true$
19:         return VAT[$\mathbf{q}$, $\mathbf{q}_c$]
20:     end if
21: end function
22:
23: function isGridValid($row$, $col$, $\mathbf{A}$)
24:     for all $g^{\mathcal{B}_k}_{row,col}$, $k = 1, \ldots, 4$ do
25:         for all $a_{i,j} \in \mathbf{A}$ do
26:             if $a_{i,j} = -1$ AND (area of hTetro block at $g^{\mathcal{B}_k}_{row,col}$) $\cap$ (area of grid at $\mathbf{g}^{\mathcal{W}}_{i,j}$) $\neq \varnothing$ then
27:                 return $false$
28:             end if
29:         end for
30:     end for
31:     return $true$
32: end function
```

V. SIMULATIONS RESULTS

In this section, the simulated path planning results of the proposed algorithm are being presented. The simulations are being conducted using MATLAB Simulink software.

Figure 8 to Figure 12 illustrate an example which demonstrates the core working principles of the proposed complete coverage path planning algorithm for reconfigurable

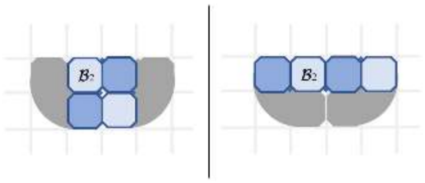


FIGURE 9. Transition validity check between O-shape and horizontal I-shape hTetro morphologies.

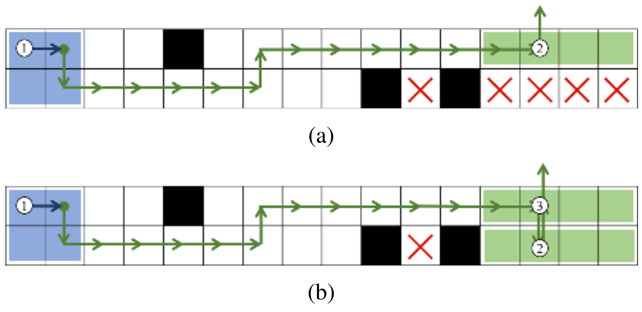


FIGURE 10. Coverage path obtained within the first stripe area in Figure 8a, which demonstrates how maximum area coverage is reached by introducing intermediate vertices.

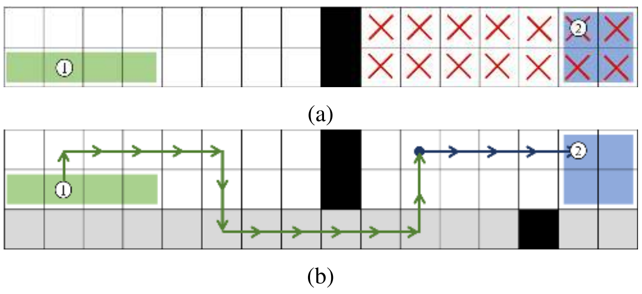


FIGURE 11. Coverage path obtained within the last stripe area in Figure 8a, which demonstrates how maximum area coverage is achieved by increasing stripe column width.

robots. Figure 8a shows the empty workspace before navigation and Figure 8b demonstrates the final CCPP result. The simulated workspace is decomposed into a grid map with 16 rows and 7 columns, with grids that are occupied by obstacles shaded in black in the figure. Two morphologies, O-shape ($s = (0, 0, -\pi, -\pi)$) and horizontal I-shape ($s = (-\pi/2, -\pi/2, -\pi/2, -\pi/2)$) are being allowed in this simulation. Figure 9 demonstrates the swept area when shape-shifting between the two morphologies occurs. The figure indicates that the clearance of four nearby grids are required for a valid shape-shifting action. These grids are being checked in Algorithm 6 during the robot reconfiguration. In this simulation, the columns are separated into 3 stripes with $n_{str_1} = 2, n_{str_2} = 3, n_{str_3} = 2$.

A. DEMONSTRATION OF STRIPE LAYER COVERAGE

Figure 10 shows the coverage path planning algorithm of the first stripe ($k = 1$) with stripe column width n_{str} of 2 and demonstrates the strategy to ensure maximum stripe area coverage by introducing intermediate vertices. The numbers in the figure represent the sequence of the configuration queue $\mathbf{q}_{queue}$, which the robot will attempt to clear accordingly once the navigation process begins. The grid that a number locates at indicates the position of the second hTetro block, and the shape of the configuration at the position is being directly illustrated in the figure, with O-shape morphology colored in blue and horizontal I-shape colored in green. The generated path is represented in arrows, where blue and green colored arrows represent movement paths of O-shaped and horizontal I-shaped hTetro, respectively. In Figure 10a, the simple path that is first generated by Algorithm 5 has five grids that are unvisited since re-visiting the same vertex is prohibited. The unvisited grids are marked with 'X' in the figures. Since the stripe is not fully covered by the path generated, Algorithm 4 will search for configurations that cover the unvisited grids and set them as intermediate configurations. In the presented scenario, the four unvisited grids at the right side of the figure can be covered by a horizontal I-shaped hTetro, so an intermediate configuration is being inserted into $\mathbf{q}_{queue}$. The single unvisited grid at the left, however, cannot be covered by any of the allowed configurations. These grids are not in the scope of the complete coverage task we aim to accomplish, so once no viable configurations can be found that covers the grids, the grids will be left unvisited while the algorithm moves on to clear the next stripe layer set. Figure 10b demonstrates the path generated after the horizontal I-shaped intermediate configuration is added as the second configuration to be cleared during the navigation process. As shown in this example, by introducing intermediate configurations that separate the stripe into different regions, the algorithm is able to search for the simple path independently and ensure maximum area coverage within the stripe layer by allowing several grids with overlapping paths.

Figure 11 demonstrates the last stripe layer ($k = 3$) at the top with stripe column width n_{str} of 2. As shown in Figure 11a, the stripe is being blocked by obstacles and neither hTetro morphologies are able to navigate to the ending configuration of the stripe. If no valid path is found during the process, the stripe width will be gradually increased until a valid path is found. In Figure 11b, a column (gray colored grids) that has been fully covered previously is being borrowed by the current stripe, and traveling to vertices on this column is being permitted. With the extra column introduced, as shown in the figure, the algorithm is now able to generate a path that avoids the obstacle and reaches the goal configuration.

B. DEMONSTRATION OF ROADMAP CONSTRUCTION

After all edge costs in the auxiliary graph are cached in the memoization table in Algorithm 1, Dijkstra algorithm is being implemented to determine the optimal cut-edge vertices between all stripe layer sets that results in minimal overall action cost. The process is being illustrated in Figure 12, where the costs of all stripe layer sets and cut-edges are being listed. In this example, the generated

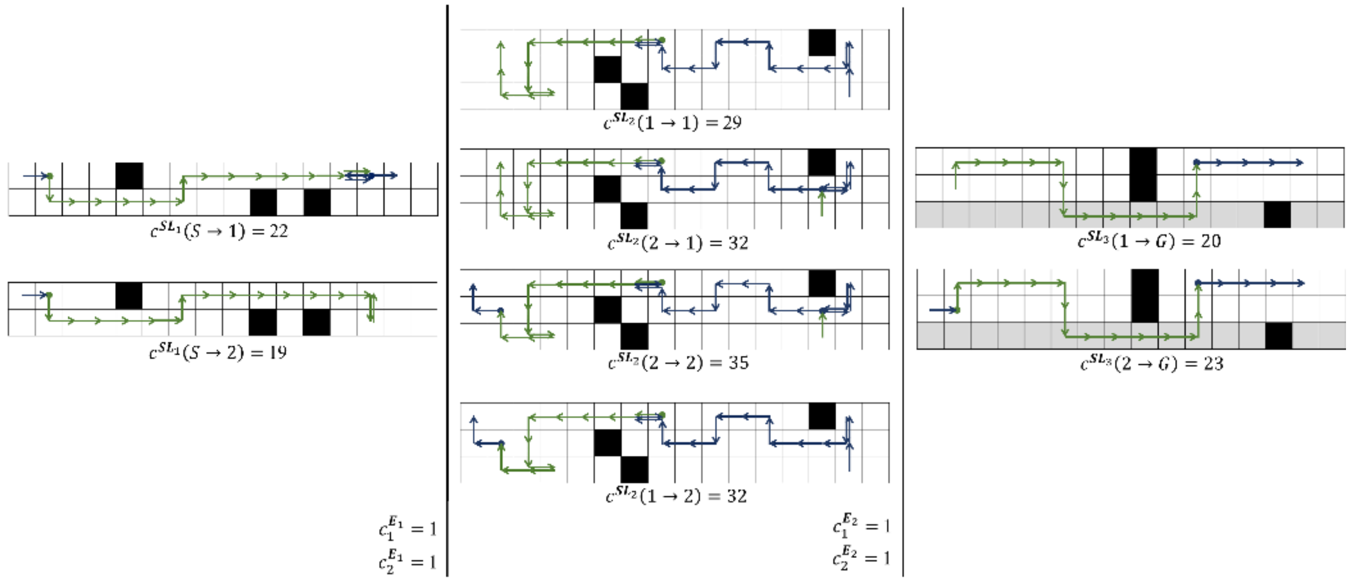


FIGURE 12. Dijkstra searching process of auxiliary graph of workspace example shown in Figure 8a. The generated optimal path is shown in Figure 8b with a total cost of 73.

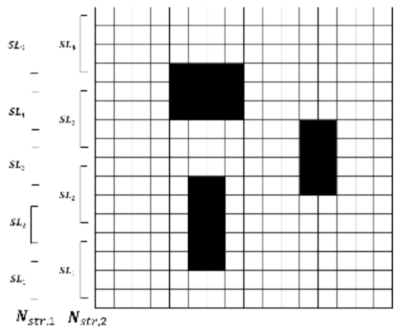


FIGURE 13. A workspace example with the area separated into a 16 × 16 grid map. The two stripe separating methods ($N_{str,1}$, $N_{str,2}$) are shown in the figure.

optimal path takes path with the following vertices in the auxiliary graph: ($v_S \to v_1^{E_1} \to v_1^{E_2} \to v_G$). The total action cost is calculated by summing up all the stripe layer set costs and cut-edge costs the path passes by, which equals to 73.

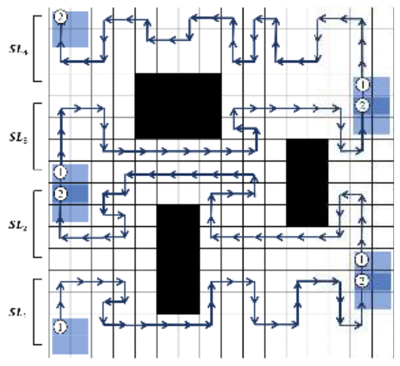
The last step in Algorithm 5 attempts to generate the final roadmap $\mathcal{R}$ which stores all robot configurations within the workspace according to the action sequence. The roadmap is generated by concatenating the paths of all stripe layer sets which is stored in the stripe path table (**SPT**). Once the roadmap stores all robot configurations within the workspace in sequence, the CCPP task is completed and the robot is ready to start its navigation in the workspace. By connecting the paths in the generated roadmap, the final result of the algorithm shown in Figure 8b. In this figure, all intermediate vertices are being drawn to better illustrate the moving patterns and the morphologies of the hTetro robot.

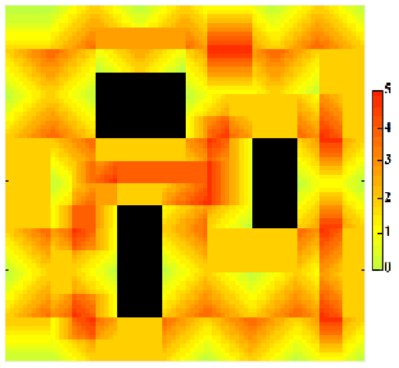
C. ALGORITHM STARTING VARIABLES

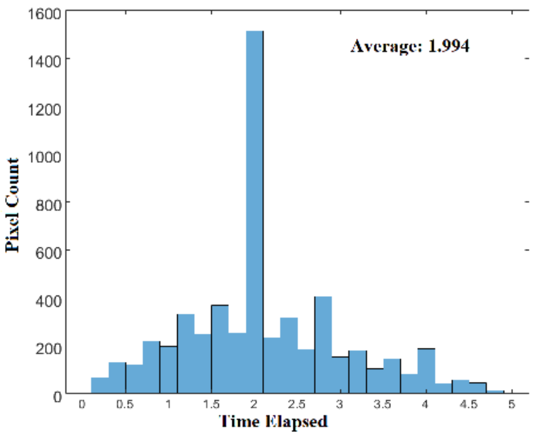
In the proposed algorithm, adjustable variables that determine algorithm performance include stripe column widths n_{str} and the allowed hTetro morphologies **S**. Since the final path outcome is affected by the starting variables, the main objective of this subsection is to propose promising starting variables so that the algorithm will function properly regardless of the size of the workspace and the quantity of obstacles. The efficiency of predetermined starting variables will be evaluated based on following criteria: i) whether the setup yields a path that achieves complete coverage ii) total action cost iii) total overlapping area. A new workspace example is being introduced as shown in Figure 13 with grid size of 16 × 16 for the analysis, and the proposed CCPP algorithm will be implemented to calculate the optimal path for different starting variables setups.

The first criterion being checked is the capability of the algorithm to generate a complete coverage path with the assigned starting variables. Several combinations of starting variables might not work well in workspaces with complicated obstacle layout or with inadequate stripe column widths chosen. For instance, the workspace demonstrated in Figure 8a has n_{str} of several stripe layers set to 2, which significantly restricts the possible morphologies that can be utilized within the layers. The placement of obstacles creates narrow regions which limit the transformation space of the robot, resulting in shape-shifting into certain morphologies becoming a sub-optimal strategy in the algorithm.

Once a valid path is generated, the total action cost is calculated based on the optimization result of the proposed algorithm, and the overlapped area is being evaluated based on the time elapsed for a robot to cover the workspace. As defined in section II-D, a linearly moving hTetro block


(a)


(b)



(c)

FIGURE 14. Navigation result of workspace in Figure 13 with stripe column width set $\mathbf{N}_{str,2}$ and allowed morphology set to O-shape.

will cover an entire grid area within a single time instance. The overlapped area analysis decomposes a single grid into 10 × 10 pixels and calculates the time each pixel is being covered by hTetro blocks throughout the entire navigation process. The time step is being set to 0.1 in this analysis to provide coverage estimation with higher accuracy for robot motions. The average coverage time will then be calculated based on the average time spent on each pixel.

The starting variables chosen for the analysis make use of three different hTetro morphologies, namely, O-shape, vertical I-shape, and horizontal I-shape. Simulations have been conducted which experiment with different combinations of hTetro morphologies and column stripe width S. In the analysis, two different profiles of stripe column widths are tested, which are $\mathbf{N}_{str,1}$ and $\mathbf{N}_{str,2}$. They are defined as follow:

$$\mathbf{N}_{str,1} = \begin{cases} 3 & ,k=1,\ldots,4 \\ 4 & ,k=5 \end{cases} ;\ \mathbf{N}_{str,2} = 4,\ k=1,\ldots,4 \qquad (9)$$

A visualization of the two stripe column sets is shown in Figure 13, where $\mathbf{N}_{str,1}$ generally makes use of stripes with column width of 3 and $\mathbf{N}_{str,2}$ with column width of 4.

After the proposed CCPP algorithm is being implemented on different hTetro morphology sets and stripe column width sets, the results including the successes of complete coverage, action cost, and average coverage time of different morphology sets are recorded as shown in Table 2, where hTetro O-shape morphology is simplified as 'O'; vertical I-shape simplified as 'vI'; and horizontal I-shape simplified as 'hI'. According to the table, with all three morphologies selected and $\mathbf{N}_{str,2}$ chosen, the algorithm demonstrates the best performance with the lowest action cost and lowest average coverage time. Scenarios where $\mathbf{N}_{str,1}$ is selected generally do not perform well compared to $\mathbf{N}_{str,2}$ since most hTetro morphologies struggle to fully cover the entire workspace with small stripe column widths. The table also suggests that by increasing the number of morphologies allowed, the performance of the path planning algorithm is improved accordingly. A comparison between starting variables with different allowed morphology sets is shown in Figure 14 and Figure 15. Figure 14 demonstrates an example with stripe column width set $\mathbf{N}_{str,2}$ and with only O-shaped allowed; whereas in the example of Figure 15, all three morphologies are allowed. Figure 14a and Figure 15b show the path generated by the proposed algorithm. Figure 14b and Figure 15b illustrate the coverage result of the path, showing the time spent for the robot to cover each pixel. A yellow colored pixel shows that the total coverage time is around 1 unit time, while longer coverage time yields a darker color at the pixel. The distribution histograms of the time spent on each pixel are shown in Figure 14c and Figure 15c. Even though both examples yield a rather close total action cost according to Table 2, with more hTetro morphologies allowed, the average grid coverage time is greatly reduced from 1.994 to 1.770. The reason being that the robot can easily switch to morphologies that effectively cover areas at large and open spaces, where the upper and lower right area yields covering the time of nearly one as shown in Figure 15b.

Therefore, for practical implementations of the proposed algorithm in the real world, which generally consists of a larger workspace and obstacles with regular shapes, the following starting variable setup for the hTetro workspace graph is suggested:

$$\begin{aligned} 3n_{str_k} &= 4\,,\ \forall n_{str_k} \in \mathbf{N}_{str} \\ \mathbf{S} &= \{(0,0,0,0),(0,0,-\pi,-\pi), \\ &\qquad (-\pi/2,-\pi/2,-\pi/2,-\pi/2)\} \end{aligned}$$

This starting variable setup is capable of efficiently clearing most unobstructed areas with vertical I-shaped hTetro morphology and makes use of O-shaped and horizontal I-shaped morphologies to achieve obstacle avoidance. The proposed CCPP algorithm with this setup yields an optimal navigation strategy that minimizes the total number of vertices being revisited while achieving full area coverage and shows its

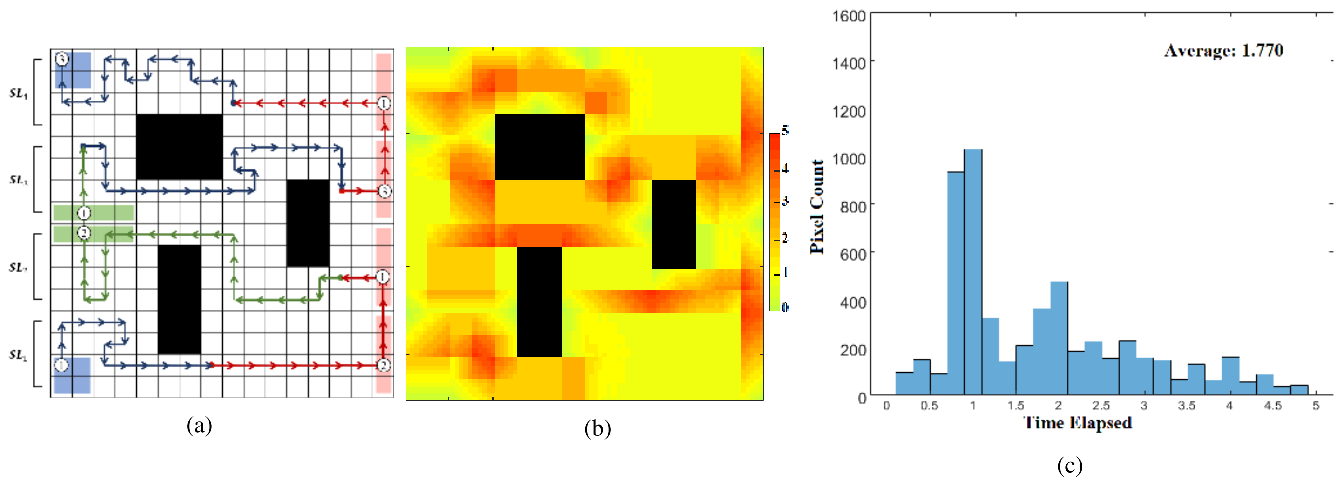


FIGURE 15. Navigation result of workspace in Figure 13 with stripe column width set $N_{str,2}$ and allowed morphology set to O-, vertical I-, and horizontal I-shape. (Red arrows: vertical I-shape; green arrows: horizontal I-shape; blue arrows: O-shape).

TABLE 2. Starting variables performance table.

Morphology Set	Stripe column Width Set	Complete coverage	Action cost	Average coverage time
O	$\mathbf{N}_{str,1}$	Yes	137	2.104
Any config with vI,hI	$\mathbf{N}_{str,1}$	No	-	-
O	$\mathbf{N}_{str,2}$	Yes	120	1.994
vI	$\mathbf{N}_{str,2}$	No	-	-
hI	$\mathbf{N}_{str,2}$	No	-	-
O, vI	$\mathbf{N}_{str,2}$	Yes	118	1.864
O, hI	$\mathbf{N}_{str,2}$	Yes	120	1.994
vI, hI	$\mathbf{N}_{str,2}$	Yes	119	1.837
O, vI, hI	$\mathbf{N}_{str,2}$	Yes	112	1.770

strong potential to be implemented in real-world reconfigurable robots to tackle complete coverage tasks with high dexterity and efficiency.

VI. CONCLUSIONS

This paper presents a novel off-line approach that focuses on complete coverage path planning tasks for self-reconfigurable robots using graph theory based searching algorithms and optimization techniques. In the presented algorithm, the workspace is modeled as a graph with multiple morphology layers sets and can be decomposed into several stripe layer sets. The navigation strategy focuses on full coverage within each individual stripe layer set, where the algorithm takes the cost of all robot actions into account and generates a path with minimal action cost through recursive backtracking. The stripe layer costs will then be calculated and memorized. With the stripe layer costs and cut-edge costs identified, an auxiliary graph is created where the Dijkstra algorithm is being implemented to determine the final path with the optimal configuration sequence between stripe layer sets. Finally, this paper analyzes the performance of different algorithm starting variables and proposes an ideal setup for real-world reconfigurable robot implementation.

Potential future research areas are as follow. (1) Improvement of graph partitioning strategies. By splitting the original workspace graph into subgraphs of wisely designed shapes instead of simple stripes may reduce revisited vertices and yield better solutions. (2) Alternative optimization goals, such as minimum energy cost or the minimum number of grids that are covered by hTetro blocks multiple times. (3) Extension of current work to different self-configurable robots and adaption of the algorithm to the new platforms accordingly.

REFERENCES

[1] M. Waanders, "Coverage path planning for mobile cleaning robots," in *Proc. 15th 20th Student Conf. IT*, Enschede, The Netherlands, 2011, pp. 1–10.

[2] Z. Bo, F. Fang, S. Zhenhua, M. Zhengda, and D. Xianzhong, "Fast and templatable path planning of spray painting robots for regular surfaces," in *Proc. 34th Chin. Control Conf. (CCC)*, Jul. 2015, pp. 5925–5930. doi: 10.1109/chicc.2015.7260567.

[3] M. Đakulovic and I. Petrovic, "Complete coverage path planning of mobile robots for humanitarian demining," *Ind. Robot, Int. J.*, vol. 39, no. 5, pp. 484–493, 2012. doi: 10.1108/01439911211249779.

[4] R. N. De Carvalho, H. A. Vidal, P. Vieira, and M. I. Ribeiro, "Complete coverage path planning and guidance for cleaning robots," in *Proc. IEEE Int. Symp. Ind. Electron. (ISIE)*, Jul. 1997, pp. 677–682. doi: 10.1109/isie.1997.649051.

[5] B.-M. Shiu and C.-L. Lin, "Design of an autonomous lawn mower with optimal route planning," in *Proc. IEEE Int. Conf. Ind. Technol.*, Apr. 2008, pp. 1–6. doi: 10.1109/icit.2008.4608497.

[6] P.-M. Hsu and C.-L. Lin, "Optimal planner for lawn mowers," in *Proc. IEEE 9th Int. Conf. Cybernetic Intell. Syst.*, Sep. 2010, pp. 1–7. doi: 10.1109/ukricis.2010.5898126.

[7] H. Choset, "Coverage for robotics—A survey of recent results," *Ann. Math. Artif. Intell.*, vol. 31, nos. 1–4, pp. 113–126, Oct. 2001. doi: 10.1023/A:1016639210559.

[8] Y. Gabriely and E. Rimon, "Spanning-tree based coverage of continuous areas by a mobile robot," in *Proc. IEEE Int. Conf. Robot. Automat. (ICRA)*, May 2001, pp. 1927–1933. doi: 10.1109/robot.2001.932890.

[9] Y. Gabriely and E. Rimon, "Competitive on-line coverage of grid environments by a mobile robot," *Comput. Geometry*, vol. 24, no. 3, pp. 197–224, 2003. doi: 10.1016/s0925-7721(02)00110-4.

[10] 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. Robots Syst.*, Oct. 2009, pp. 5788–5793. doi: 10.1109/iros.2009.5354499.

[11] E. Gonzalez, O. Alvarez, Y. Diaz, C. Parra, and C. Bustacara, "Bsa: A complete coverage algorithm," in *Proc. IEEE Int. Conf. Robot. Automat.*, Apr. 2005, pp. 2040–2044. doi: 10.1109/robot.2005.1570413.

[12] S. X. Yang and C. Luo, "A neural network approach to complete coverage path planning," *IEEE Trans. Syst., Man, Cybern., B (Cybern.)*, vol. 34, no. 1, pp. 718–724, Feb. 2004. doi: 10.1109/tsmcb.2003.811769.

[13] 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, 2016. doi: 10.1016/j.jides.2016.05.004.

[14] T. R. Schäfle, S. Mohamed, N. Uchiyama, and O. Sawodny, "Coverage path planning for mobile robots using genetic algorithm with energy optimization," in *Proc. Int. Electron. Symp. (IES)*, Sep. 2016, pp. 99–104. doi: 10.1109/elecsym.2016.7860983.

[15] A. Janchiv, D. Batsaikhan, B. Kim, W. G. Lee, and S.-G. Lee, "Time-efficient and complete coverage path planning based on flow networks for multi-robots," *Int. J. Control, Autom. Syst.*, vol. 11, no. 2, pp. 369–376, 2013. doi: 10.1007/s12555-011-0184-5.

[16] S. Dogru and L. Marques, "Towards fully autonomous energy efficient coverage path planning for autonomous mobile robots on 3D terrain," in *Proc. Eur. Conf. Mobile Robots (ECMR)*, Sep. 2015, pp. 1–6. doi: 10.1109/ecmr.2015.7324206.

[17] 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, 2012. doi: 10.1007/s10489-012-0406-4.

[18] 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. doi: 10.1109/tie.2004.825197.

[19] N. Tan, N. Rojas, R. E. Mohan, V. Kee, and R. Sosa, "Nested reconfigurable robots: Theory, design, and realization," *Int. J. Adv. Robot. Syst.*, vol. 12, no. 7, p. 110, 2015. doi: 10.5772/60507.

[20] V. Prabakaran, M. R. Elara, T. Pathmakumar, and S. Nansai, "hTetro: A tetris inspired shape shifting floor cleaning robot," in *Proc. IEEE Int. Conf. Robot. Automat. (ICRA)*, May/Jun. 2017, pp. 6105–6112. doi: 10.1109/icra.2017.7989725.

[21] V. Prabakaran, R. E. Mohan, V. Sivanantham, T. Pathmakumar, and S. Kumar, "Tackling area coverage problems in a reconfigurable floor cleaning robot based on polyomino tiling theory," *Appl. Sci.*, vol. 8, no. 3, p. 342, 2018. doi: 10.3390/app8030342.

[22] J. Bruce and M. Veloso, "Real-time randomized path planning for robot navigation," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, Sep./Oct. 2002, pp. 2383–2388. doi: 10.1109/irds.2002.1041624.

[23] F. Duchoň, A. Babinec, M. Kajan, P. Beňo, M. Florek, T. Fico, and L. Jurišica, "Path planning with modified a star algorithm for a mobile robot," *Procedia Eng.*, vol. 96, pp. 59–69, 2014. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S187770581403149X. doi: 10.1016/j.proeng.2014.12.098.

[24] K. Al-Mutib, M. Alsulaiman, M. Emaduddin, H. Ramdane, and E. Mattar, "D* lite based real-time multi-agent path planning in dynamic environments," in *Proc. 3rd Int. Conf. Comput. Intell., Modelling Simulation*, Sep. 2011, pp. 170–174. doi: 10.1109/cimsim.2011.38.

[25] D. Portugal, C. H. Antunes, and R. Rocha, "A study of genetic algorithms for approximating the longest path in generic graphs," in *Proc. IEEE Int. Conf. Syst., Man Cybern.*, Oct. 2010, pp. 2539–2544. doi: 10.1109/icsmc.2010.5641920.

[26] A. Björklund, T. Husfeldt, and S. Khanna, "Approximating longest directed paths and cycles," in *Automata, Languages and Programming* (Lecture Notes in Computer Science). Berlin, Germany: Springer, 2004, pp. 222–233. doi: 10.1007/978-3-540-27836-8_21.

[27] J.-C. Latombe, *Robot Motion Planning*. Norwell, MA, USA: Kluwer, 2010.

[28] E. Galceran and M. Carreras, "A survey on coverage path planning for robotics," *Robot. Auton. Syst.*, vol. 61, no. 12, pp. 1258–1276, 2013. doi: 10.1016/j.robot.2013.09.004.

[29] E. Ippoliti, *Heuristic Reasoning*. London, U.K.: Springer, 2015.

[30] L. Kuvcera, "Expected complexity of graph partitioning problems," *Discrete Appl. Math.*, vol. 57, nos. 2–3, pp. 193–212, 1995. doi: 10.1016/0166-218x(94)00103-k.

[31] H. Meyerhenke, P. Sanders, and C. Schulz, "Parallel graph partitioning for complex networks," in *Proc. IEEE Int. Parallel Distrib. Process. Symp.*, May 2015, pp. 1055–1064. doi: 10.1109/ipdps.2015.18.

[32] H. Choset and P. Pignon, "Coverage path planning: The boustrophedon cellular decomposition," in *Field and Service Robotics*. London, U.K.: Springer, 1998, pp. 203–209. doi: 10.1007/978-1-4471-1273-0_32.

[33] F. S. Hillier and G. J. Lieberman, *Introduction to Mathematical Programming*. New York, NY, USA: McGraw-Hill, 1996.

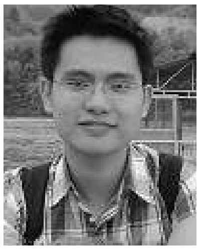
[34] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*. Cambridge, MA, USA: MIT Press, 2009.

[35] J. Edmonds, "Recursive backtracking," in *How to Think About Algorithms*. New York, NY, USA: Cambridge Univ. Press, 2008, pp. 251–266. doi: 10.1017/cbo9780511808241.019.

KU PING CHENG received the B.Sc. degree in computer engineering from the Singapore University of Technology and Design, in 2017, where he is currently a Research Officer in autonomous robotics with the Engineering Product Development Pillar. His research interests include robotics and automation, intelligent robots, control systems, and computer vision.

RAJESH ELARA MOHAN received the B.E. degree from Bharathiar University, India, in 2003, and the M.Sc. and Ph.D. degrees from Nanyang Technological University, in 2005 and 2012, respectively. He is currently an Assistant Professor with the Engineering Product Development Pillar, Singapore University of Technology and Design. He is also a Visiting Faculty Member with the International Design Institute, Zhejiang University, China. He has published over 80 papers in leading journals, books, and conferences. His research interest includes robotics with an emphasis on self-reconfigurable platforms as well as research problems related to robot ergonomics and autonomous systems. He was a recipient of the SG Mark Design Award, in 2016 and 2017, the ASEE Best of Design in Engineering Award, in 2012, and the Tan Kah Kee Young Inventors' Award, in 2010.

NGUYEN HUU KHANH NHAN defended his Ph.D. thesis at the Institute of Research and Experiments for Electrical and Electronic Equipment, Moscow, Russian. He is currently a Lecturer with the Faculty of Electrical and Electronic Engineering, Ton Duc Thang University, Ho Chi Minh City, Vietnam. His research interests include VLSI, MEMS and LED driver chips, robotics vision, robot navigation, and 3D video processing.



ANH VU LE received the B.S. degree in electronics and telecommunications from the Ha Noi University of Technology, Vietnam, in 2007, and the Ph.D. degree in electronics and electrical engineering from Dongguk University, South Korea, in 2015. He is currently with the Optoelectronics Research Group, Faculty of Electrical and Electronics Engineering, Ton Duc Thang University, Ho Chi Minh City, Vietnam. He is also a Postdoctoral Research Fellow with the ROAR Laboratory, Singapore University of Technology and Design. His current research interests include robotics vision, robot navigation, human detection, action recognition, feature matching, and 3D video processing.

• • •