Motion planner for a Tetris-inspired reconfigurable floor cleaning robot

Prabakaran Veerajagadheswar1, Ku Ping-Cheng1, Mohan Rajesh Elara1, Anh Vu Le2, and Masami Iwase3

Abstract
Coverage path planning technique is an essential ingredient in every floor cleaning robotic systems. Even though numerous approaches demonstrate the benefits of conventional coverage motion planning techniques, they are mostly limited to fixed morphological platforms. In this article, we put forward a novel motion planning technique for a Tetris-inspired reconfigurable floor cleaning robot named “hTetro” that can reconfigure its morphology to any of the seven one-sided Tetris pieces. The proposed motion planning technique adapts polyomino tiling theory to tile a defined space, generates reference coordinates, and produces a navigation path to traverse on the generated tile-set with an objective of maximizing the area coverage. We have summarized all these aspects and concluded with experiments in a simulated environment that benchmarks the proposed technique with conventional approaches. The results show that the proposed motion planning technique achieves significantly higher performance in terms of area recovered than the traditional methods.

Keywords
Floor cleaning robot, reconfigurable mechanism, coverage path planning, Tetris tiling theory

Date received: 1 February 2019; accepted: 2 March 2020

Introduction
Floor cleaning robots are becoming an integral part of every household that could improvise the productivity and quality of life through performing repetitive and time-consuming task. It is estimated that cleaning robots could reach a market value of USD 4.34 Billion in 2023, wherein the floor cleaning robots will hold a more significant share. Even though the dominant market player iRobot, Neato, Samsung, and Xiaomi claims that they have sold millions of units, there exists numerous research challenges and open questions to be addressed. Such research initiatives have continually pushed the performance limits of floor cleaning robots in the areas of mechanical design,2 autonomy,3 human–robot interaction,4 and performance benchmarking.5 Even though the diverse literature on floor cleaning robot demonstrates its benefits and the translation from abstract design to commercial technology, the conventional robots endure from a series of performance degradation. One primary attributes that curtail the efficacy of the floor cleaning robots is their fixed morphology
design, which restrains their accessibility in constrained spaces and weakening area coverage. We proposed an alternative design in the works of Prabakaran et al.\(^4\) and Le et al.\(^7\) to overcome these bottlenecks in the traditional cleaning platforms where we introduced a next-generation of reconfigurable floor cleaning robot named “hTetro” inspired from the game called Tetris. The developed robot is capable of reconfiguring its morphology to any of the seven one-sided Tetrino pieces which aid the robot to achieve superior area coverage performance than a traditional fixed morphology cleaning platforms.

In the process of developing a floor cleaning robot, it is critical to address its capacity to demonstrate autonomous and efficient coverage path planning (CPP). Similarly, in the case of hTetro, it is crucial to establish its CPP intelligence. CPP is an exciting field of study for robotic scientists with numerous research literature. One common technique that has been extensively utilized for CPP task is cellular decomposition. This method breaks down the workspace into small cells and applies motion planning to each cell to cover the entire region. Numerous other methods have been proposed to decompose the given space that includes trapezoidal decomposition,\(^8\) boustrophedon decomposition,\(^9\) Morse-based cellular decomposition,\(^10\) among others. Wong and MacDonald proposed a topological area coverage technique that could point landmarks as nodes to cover the area.\(^11\) In another work, a sensor-based coverage technique was proposed by Butler et al.\(^12\) The proposed method uses sensed robot data to generate paths that could reach a possible point to achieve maximum area.

Grid-based area coverage methods are in use over the last three decades, wherein the workspace is treated as uniform grid cells, and each cell contains a value which replicates the presence of an obstacle. There are numerous algorithms proposed towards realizing grid-based coverage such as wavefront algorithm,\(^13\) spanning tree technique,\(^14\) hexagonal grid decomposition,\(^15\) and graph theory-based coverage.\(^16\) Xu\(^17\) presented a graph-based CPP technique, in which he considers the mapped area as a graph and applies robot motion planning to reach every coordinate in the graph. In recent years, several 3-D area coverage techniques have been proposed and validated in the context of service robots. For instance, Cheng et al. applied 3-D coverage to inspect urban structure.\(^18\) Galceran and Carreras proposed a bathymetric 3-D map to inspect ocean floors.\(^19\) Jin and Tang proposed a 3-D coverage algorithm for agriculture purpose.\(^20\) Other CPP techniques like optimized 3-D coverage and multi-robot coverage strategies were discussed in Galceran and Carreras.\(^21\) Even though there is numerous literature that demonstrates the application of CPP techniques in robotic systems, the current approaches are mostly limited to fixed morphological platforms. Also, none of these existing techniques were realized on a reconfigurable robot in the context of a meaningful application.

Reconfigurable robots were introduced in the late 1980s and have been extensively studied since then. Transformable robots are classified into three major categories, namely, intra-reconfigurable, inter-reconfigurable, and nested-reconfigurable robots. Inter-reconfiguration is where an individual robot can change its morphology from one form to another. Some examples to this end include Scorpio,\(^22\) a bio-inspired robot capable of switching between rolling, crawling and wall climbing forms, and Panthera, a reconfigurable, could change the width for pavement cleaning.\(^23\) Inter-reconfiguration is where multiple similar robots come together to form global morphologies by undergoing assembly and disassembly process. Precedents include CEBOT, M-TRAN, ATRON, CKBot, and Molecube. The third category is nested robots, which are capable of performing both inter- and intra-reconfigurations. One such example is Hinged-Tetro, a robot that can change its morphology on its own and can form global morphology by attaching and detaching with a team of other Hinged-Tetro robots.\(^24\) With a wealth of literature covering different aspects of reconfigurable robots, connection to meaningful applications such as floor cleaning has always been weak. Also, there are minimal studies on CPP in the context of reconfigurable floor cleaning robots. To this end, a novel CPP strategy was proposed in one of our previous work\(^25\) for the hTetro robot based on the polyomino tiling theory. In that work, we validated the application of polyomino tiling theory in the context of area coverage. The work analyzes the hTetro’s system architecture, the Tetris theorems that have been validated, and concludes with the experimental results shows superior area coverage. However, the generation of the tile-set and the robot’s coverage process were executed involving manual support without any motion planning strategies. In the current work, we are proposing a motion planner framework, which can autonomously generate the tile-set and its coverage route for the hTetro robot to achieve maximum area coverage.

In most cases with CPP strategies, the commonly used motion planning algorithms are spiral motion, backtracking spiral motion, and boustrophedon (back-and-forth) motion. For instance, Jin and Tang proposed a decomposition method in order to cover the farming field using boustrophedon paths.\(^20\) Similarly, Hamede et al. proposed a genetic algorithm for CPP and used boustrophedon paths for coverage.\(^26\) A time and energy-efficient online CPP technique are proposed in Hsu et al.,\(^27\) wherein they adopt a high-resolution grid map representation and utilized spiral path motion to perform adequate coverage. In another work, Gabriely and Rimon applied the spiral motion in a cellular decomposition coverage technique.\(^14\) Gonzalez et al. utilized backtracking spiral motion in an approximate cellular decomposition area coverage method.\(^28\) Also, they proposed an improved spiral algorithm in order to consider the obstacles inside the grid space.\(^29\) Even though there is numerous literature demonstrating the advantages of existing motion techniques in the
In the context of CPP, there is a massive gap in implementing those techniques on robots with the reconfigurable base. In particular, the hTetro robot will assume different heading direction for different morphology so requires a unique motion planning algorithm in order to achieve better performance.

In this article, we are proposing a novel motion planner for the Tetris-mimicked reconfigurable floor cleaning robot that creates waypoints to navigate on the generated tile-set to cover the unified area. The proposed motion planner framework has been divided into three stages. The initial set is for planning, where the high-level global coverage planning will be performed based on the tiling theory. The second set is the generation stage, which produces the path (i.e. motion planning) to cover the area. The last stage set is the execution stage, where the robot performs navigation with appropriate morphology. In this article, we reviewed the fundamental challenges of the proposed scheme such as realizing the generation of the tile-set, motion pattern, reconfigurable ability and translating the theoretical tiling approach through the analytical process into demonstrable systems. This article discusses all these aspects and concludes with experimental to validate the efficiency of the proposed approach. We systematically benchmarked the performance of the proposed approach with few conventional motion planning techniques used in CPP (i.e. spiral, boustrophedon motion, greedy search, and random search) concerning area recovered. The motion planner herein presented is a critical effort towards designing a self-reconfigurable robot that is capable of generating global tile-set, determining correlated global and local trajectories, and generating relevant motor signals based on inverse kinematics and dynamics model.

**hTetro: A reconfigurable cleaning robot**

hTetro is a Tetris-inspired robotic floor cleaner which can change its morphology to any of the seven one-sided tetromino pieces, as shown in Figure 1. The concerned robot was developed under the principle of hinged dissection of polyominoes. The hTetro robot consists of four equivalent square blocks with each block dimension of $25 \times 25\text{ cm}^2$. Three hinged points connect the blocks with LLR (Left Left Right) configuration. The platform, as shown in Figure 1, consists of four modules connected by three passive hinges. Each of the modules has a differential-drive locomotion module driven by two 12 V DC motors, as shown in Figure 2. The locomotion modules are connected with the four blocks through the bushing on the central rod. Hence, the angle of each locomotion module is independent of its block. The angles of the locomotion modules are measured using the absolute encoders. These locomotion modules are responsible for both navigation and reconfiguration. At the end of each shape-shifting process, we lock the blocks using an electromagnet to maintain the shape while navigating. Whenever the robot receives a morphology change request, the locomotion modules will perform the shape-changing operation first, before it proceeds with navigation. For every movement, the position of all four locomotion modules is controlled separately through a feedback system. Concerning electronics, we separated it as global and local peripheries. The global components such as LIDAR, IMU, and Intel Compute stick (microprocessor with Ubuntu OS) act as a top layer which helps the robot to perform simultaneous localization and mapping (SLAM) and autonomous navigation. Since block 2 acts as an anchor point for hTetro, we housed the global peripheries on the second block. The local components such as motor driver (roboclaw), electromagnet (EM), limit switch (LS), absolute encoders (AE), motors (M), and its encoders (En) would execute the reconfiguration and locomotion of hTetro. Electromagnets are used to hold the position of the block that would maintain the morphology during navigation. Limit switches were used to ensure the completion of reconfiguration and to detect the
morphology failures. Except in block 2, we housed two sets in block 1, and four and three sets in block 3 of EM and LS components. When it comes to locomotion, we have a set of motor driver, motor, and encoder in each block. We used roboclaw motor driver to control the motors and to collect the encoder signals which will be transferred to the local controller. On top of every locomotion module, we housed an AE that could sense the angular position and send the data back to the controller. Almost every local peripheral’s data acquisition is made by the local controller for which we employed Arduino Mega that is housed in block 2. Arduino Mega (microcontroller) is the only local components that communicate with compute stick and distributes the commands to low-level peripheries to perform tasks effectively. On top of that, we housed four more micro DC motor in every block which actuates the sweepers to pull all the debris into the robot modules during locomotion. In the hTetro system, we used three cleaning modules due to the reason that it may not clean the area entirely in particular shapes. For instance, if we are using cleaning modules only in module 3 and 4, then in “O” shape the areas that segment 1 and 2 covering might not be cleaned. A specific system components of the hTetro robot is shown in Figure 3. The assumptions of modeling are (1) the robot does not perform shape changing during locomotion. In other words, it is treated a rigid platform during the locomotion. (2) The wheels roll with no slipping and no skidding during the locomotion. In this article, the local and global references of the hTetro robot were fixed on block 2. The following section details the hTetro robot’s system model setup on a workspace.

**Workspace model**

Let us consider the workspace as $W$ wherein the concerned robot $A$ is navigating. The reference frame $F$ for the workspace and the robot is expressed as $F^W$ and $F^T$. The dimension of the Cartesian workspace is denoted as $Y^W$ for $Y$ coordinate and $X^W$ for $X$ coordinate. The defined workspace is then disintegrated into square grids, wherein each grid cell size (denoted as $d_{grid}$) is identical to the size of each block of the hTetro robot. The total row and column size of the decomposed grid be like $n_{row}$ and $n_{col}$. Finally, in the grid space, the presence of the obstacles in the each cell are indicated as $O$. The figure of the model is illustrated in Figure 4.

**hTetro on a workspace**

Since the hTetro robot consist of four blocks and each of those occupies a single cell in the grid, we have given a geometric representation for all four blocks as $B_1, B_2, B_3, B_4$. Wherein $B_n(n = 1 \text{ to } 4)$ serves as a rotational angle for each block to the workspace frame. The possible rotational angles that can be achieved by each block locally are $B_n = 0, \frac{\pi}{2}, \pi$. However, the angle $B_2$ will be constant, whereas the block 2 act as a foothold for the whole hTetro platform in which the $F^A$ is tagged. The variations in $B_n$ aids the hTetro robot to achieve reconfiguration between any of the seven one-sided Tetris pieces. Unlike the traditional definition, it requires a unique representation for the hTetro to identify its position or grid coordinates $(X, Y)$ on a workspace. The pose of robot is defined: (1) The pose $q$ of an hTetro robot is a six-element array (2) the orientation of the robot $B_n$ concerning the workspace frame $F^W$

$$ q = [X, Y, B_1, B_2, B_3, B_4]^T $$

where

$$(X, Y) = \text{coordinate of hTetro Block 2}(B_2) \text{ center in workspace frame } F^W$$

$B_G = \text{global orientation of the hTetro robot which is also centered to } B_2 \text{ with respect to } F^W$

Similarly, it is critical to acquire the robot’s grid position information which provides the $i$th row and $j$th column. While considering the hTetro robot, we get four grid positions, which are the crucial block information at any point of
operation. Consider $G$ as the entire grid cells generated; each grid information represented as $G_{ij}^n$ for the workspace.

where $G_{ij}^n$ is the reference point which robot $A$ will clear while navigating.

When there is an obstacle $O$ introduced on the workspace, each $G_{ij}^n$ contains a value that replicates the obstacle presence. For instance, $O = G_{ij}^n \in \{0, 1, -1\}$, that is, free cell, obstacle cell, and undefined cell. The concerned robot can only navigate in the free grid space.

**hTetro robot navigation on the workspace**

The hTetro robot consists of navigational components in most blocks which benefit smooth locomotion. Since the concerned robot’s base is reconfigurable, achieving differential movement is very challenging. Hence, we developed linear locomotive gaits, which cause the robot to traverse forward, backward, leftward, and rightward motions. In order to achieve the mentioned motion capability, we used differential wheels in each block of the hTetro robot. The proposed motion planning strategy produces a series of waypoints $WP_{tm}$ in order to accomplish motion planning to cover the entire area. In comparison to traditional robotic pose, hTetro has its definition to represents its current stare. For instance, it requires a position $(X, Y)$, orientation $BG$ and its morphology to define its state. Each generated $WP_{tm}$ point consists of all the mentioned parameter. Once the information passed to the controller, it produces necessary motor signals to achieve action in order to complete the area coverage task. In order to reduce unblocked areas that could potentially identify as obstacle grids, the grid cell size is defined to match the size of an hTetro block $(d_{grid} = d_{block})$. Such an arrangement provides the resolution of the grid map with each grid matched to single hTetro blocks. Under this definition, the robot is capable of performing either linear motion or an angle adjustment of a block. Performing a linear motion at a time instant moves all robot blocks simultaneously in one of the four directions for a grid length $d_{grid}$ with respect to the workspace frame $FW$ neighbor block’s local frame. The hTetro robot’s action commands $q_c$ to accomplish linear and angular motion is shown in Figure 5.

**Motion planner framework for hTetro robot**

This article presents a novel motion planner framework (Figure 6) for the hTetro robot over complete coverage tasks where the choice of feasible global, local path, and morphology of the robot is critical for achieving the set performance targets. The proposed framework could be scalable across intra-, inter- and nested reconfigurable systems with minimal modifications according to its application. It allows a concerned robot to evaluate the geometry of the environment, compute desired body morphology, select associated local and global optimal trajectories, and generate appropriate motor primitives. He proposed framework consists of three stages: planning stage, generation stage, execution stage.

**Planning stage**

Precise sensing of the environment is critical to perform a better area CPP. The process of generating the map of test
space is the initial task in our proposed framework. Once the map is generated, it will automatically give out the total number of grid cells it occupies. Each cell \((G_{i,j})\) in the produced grid contains a value which replicates the mapped environment. We accumulate all the cell value in order to generate the matrix \(M\). The produced matrix is then passed to the next substage in order to plan the global tile-set. Before placing the Tetris tiles on the generated matrix, the global tile planner adopts a definite tiling theorem to reduce the number of pieces that utilized from the available one-sided Tetris set. The process of placing the tile pieces will be explained in the third section. \(M'_{i,j} = G_{i,j}^W\) (i.e. the row and column information are same in both matrix and grid).

**Generation stage**

Next, we use the tile-set matrix to generate the navigation path that achieves maximum coverage. We further divided this stage into subtask that simplifies the process of the area coverage for the concern robot. Since the hTetro robot could cover four grid cells at any point in time, it is critical to find the reference cells, which acts as a waypoint for the system to navigate. The reference generator marks the reference cells, which can be a possible waypoint for the hTetro robot. The motion path generator executes the plan of sequencing the reference cells. The accomplishment of the reference point arrangements produces navigation plan for the hTetro robot. Also, this task produces the waypoints \(WP_{tm}\) that consist of pose and morphology information of the robot and passes it to the next level. The next task is finding the current pose and morphology of the robot and generate appropriate actions to achieve the targeted state.

**Execution stage**

In the execution stage, the controller receives a series of states from the generation layer. After receiving the state information, the controller executes navigation and morphology shifts. We use a feedback controller which could generate the motor primitives for the current pose of the robot in order to achieve the received state. Similarly, the controller could perform the same set of actions for all provoked states, which ultimately leads the robot to cover the entire area. In this article, we are only discussing the planning and the generation stage.

**Strategy for generating tile-set based on tiling theory**

Implementing existing area coverage approaches on the reconfigurable platform involves several challenges due to its geometrical changes in every configuration. We adopted the polyomino tiling concept as an area coverage technique in our hTetro robot. A polyomino tiling is a process of placing tiles in a plane using any number of polyominoes pieces which is widely used in gaming and rendering purposes. Since the hTetro robot has seven different shapes, it is inefficient in terms of power and computation, when we use all seven configurations to cover the given area. So, we implemented Tetris tiling theorems to reduce the number of shapes that are used to cover the given area. Tetris tiling theorems is a branch of mathematics which provides a possible tiling set to cover a given \(A \times B\) rectangular grid space. We discussed the details of a select set of tiling theorems and its application on the hTetro robot in terms of complete coverage in Le et al. However, the details of the algorithm that generates the tile-set in a given 2-D rectangular grid space were not addressed. Next section introduces the algorithm to generate tiles in the produced matrix \(M'\).

**Choosing a theorem**

Before going to the tiling process, it is crucial to choose the possible Tetris shapes to fill the given area. In order to achieve that, we have to first choose the best tiling theorem. The best way to choose a tiling theorem is to find the size of the provoked matrix \(M'\). Let \(m\) and \(n\) be the total number of rows and columns of matrix \(M'\). Since we are considering Tetris pieces which consist of four equivalent squares, any rectangular grid that is divisible by 4 can be tiled fully with any combination of Tetris pieces. For instance, in the considered matrix, if \(m \times n\) is divisible by 4, then we can have complete tile. In order to reduce the number of tile pieces, we used the Theorem 0.1 stated below. Also, it is important to count the number of elements in the matrix \(M'\), which consist of obstacles. If the number of elements that consist of obstacles is also divisible by 4, then we can apply the below theorem.

**Theorem 0.1 (TH1)**. An \(a \times b\) gridded rectangle can be tiled by “T,” “S,” “Z” tetrominoes if and only if \(a, b \geq 4\) and either

1. One side is divisible by 4 or
2. \(a, b \equiv 2(\text{mod} 4) \land a + b > 16\).

While mapping an unknown environment, it is impractical to expect the total grid occupied could be a factor of 4. In such cases, we must use other relevant theorems to tile the defined space. If the dimensions are not exactly a factor of 4, then we will not obtain a complete tiling. We may have to leave one to two cells untiled in all possible situations. When the theorem is applied to the generated matrix \(M'\), one to two matrix element’s values are neglected to default. These elements may also consider as an obstacle. Other relevant Tetris tiling theorems that can be chosen in the condition as mentioned earlier are furnished below.

**Theorem 0.2 (TH2)**. Let \(a\) and \(b\) be integers that are strictly greater than 1. As such, the “T,” “S,” “Z” tetrominoes can tile a modified rectangle of side \(a\) and \(b\) if and only if either
Theorem 0.3 (TH3). The “O,” “L,” “J” tetrominoes can tile a deficient rectangle of side \( a \) and \( b \) only if

1. \( a, b \equiv 2 \mod 4 \) and \( b \equiv 1 \mod 2 \) or
2. \( a \equiv 1 \mod 4 \) and \( b = 2^{34} \).

Placement of tiles

The first step in the process of tiling the matrix \( M' \) is to find the possible placements of all selected pieces. When the number of selected Tetris shapes (among the available seven one-sided Tetris shapes) is more, then it requires excess time and memory to complete the tiling action. In order to obtain the appropriate tiling set, we utilized bipartite relation between the placement and the matrix elements. For instance, Figure 7 describes the typical association of the placement graph wherein we restricted to Tromino (a type of polyomino that contains three equivalent square blocks) for better understanding. The figure illustrates a \( 2 \times 3 \) matrix \( m_{e} \), and all possible placements of L-Tromino shapes. Each L-Tromino piece will occupy three matrix elements and produce eight possible placement combinations.

The next step is to search the best element to place the tile, which illustrated in Figure 8. We use a backtracking algorithm to search for the best element \( m_{e} \) in the matrix to place the selected tile. Once the placement of tile is established, the algorithm adds the implanted matrix items \( m_{e} \) in the solution set and tries to tile the rest of the matrix items. If the algorithm could not manage to place the next tile, then it tries other possible placements. Even after trying all viable placements, when the algorithm departs from tile combinations, then it backtracks to the previous matrix element and continues the same procedure with the new tile piece. The same process continues until we get a fully tiled matrix solution. The concern tile placement procedures are also applicable to the grid space with obstacles.

Generation of tile on matrix \( M' \)

The matrix \( M' \) was generated in the planning stage after the stage of occupancy grids. As explained previously, each element of \( M' \) holds a value \((0 \text{ or } -1)\) that replicates the mapped environment. When the algorithm (Algorithm 1) starts with tile placing process, it will check the element’s value \( M'_{ij} \) before placing the tile. The algorithm only considers the elements that are \( M'_{ij} = 0 \) in order to establish the placement of tile. If the element \( M'_{ij} < 0 \), then the proposed placing method will eliminate the item for tile placements and proceed to search for the next possible placement item. After placing each tile pieces on the matrix, the algorithm will number \( \{P_n\} \) the placed tiles in ascending order until the algorithm exit by ran out of placements. Random tiling for a \((6 \times 6)\) matrix was generated is shown in Figure 9.

Strategy for motion planning

In most grid-based CPP techniques, each cell represents the full robot (i.e. each cell is robot sized). So the path planning scheme needs to pass the grid coordinates to achieve the full coverage. Since the hTetro platform is unique in its reconfiguration ability and it covers four cells in a grid at any point of time, the path planning technique should generate reference coordinates for every tiled piece in order to accomplish the tiling motion strategy. The generated coordinates will be tagged with block 2 \((B_2)\) for all seven hTetro’s configurations. These coordinates will act as a waypoint \( WP \) for the robot that aids to achieve full coverage. On the other hand, the previous coverage motion planning techniques never concern the robot’s orientation because of their fixed morphology. Positioning the heading angle in every coverage platforms is treated as a control problem. Whereas with the hTetro robot, it also requires an
Algorithm 1. Placing tile in matrix $M'$.

1: function PLACE TILE ($M'$)
2: $c_t \leftarrow m \times n$; $c_n \leftarrow 0$
3: $p_n, n_t \leftarrow 1$
4: if $\text{mod}(c_t, 4) = 0 \land \text{mod}(c_t, 4) = 2$ then
5: \hspace{1em} Choose Theorem TH1
6: \hspace{2em} $T_1 \leftarrow \text{tileT}$
7: \hspace{2em} $T_2 \leftarrow \text{tileZ}$
8: \hspace{2em} $T_3 \leftarrow \text{tileS}$
9: \hspace{1em} else if $\text{mod}(c_t, 4) = 1$ then
10: \hspace{2em} Choose Theorem TH3
11: \hspace{3em} $T_1 \leftarrow \text{tileO}$
12: \hspace{3em} $T_2 \leftarrow \text{tileL}$
13: \hspace{3em} $T_3 \leftarrow \text{tileJ}$
14: \hspace{1em} else
15: \hspace{2em} Choose Theorem TH2
16: \hspace{3em} $T_1 \leftarrow \text{tileT}$
17: \hspace{3em} $T_2 \leftarrow \text{tileZ}$
18: \hspace{3em} $T_3 \leftarrow \text{tileS}$
19: \hspace{1em} end if
20: while $c_n < c_t$ do
21: \hspace{2em} for all $i \leftarrow 0$ to $n_{row}$ do
22: \hspace{3em} for all $j \leftarrow 0$ to $n_{col}$ do
23: \hspace{4em} if $M'[i][j] = 0$ then
24: \hspace{5em} while $p_n = p_n + 1$ do
25: \hspace{6em} Try place tile $T_{n_t}$
26: \hspace{7em} if (!Placed) then
27: \hspace{8em} Try place tile $T_{n_t}^T$
28: \hspace{9em} if (!Placed) then
29: \hspace{10em} $n_t = n_t + 1$
30: \hspace{5em} end if
31: \hspace{4em} else
32: \hspace{5em} Try place tile $(T^n_T) p_n$
33: \hspace{6em} $p_n = p_n + 1$
34: \hspace{7em} $c_n = c_n + 1$
35: \hspace{5em} end if
36: \hspace{4em} end if
37: \hspace{3em} end for
38: \hspace{2em} end for
39: \hspace{1em} end while
40: \hspace{1em} $V_n \leftarrow p_n$
41: \hspace{1em} end function

Sequencing of reference cells

This section discusses the strategy of sequencing the reference points generated so that the hTetro robot follows the order as a roadmap. In order to sequence the generated reference points, we are using a technique which is based on orientation reference, due to its directional shift that takes effect after every morphological movement. Acquiring the tiling matrix $M'$ and $G$ information as an input for this stage of operation to produce a series of $WP$, which consist of hTetro robot’s state $Q$, where $Q = [q_c + B_G]$.

Generation of reference cell

For better reference cell identification, we required two information that includes the tile type and its placement orientation. This two information we can acquire from the produced $M'$ matrix. As we know, each tile pieces that were placed in the matrix $M'$ holds a unique number $P_n$ in ascending order, which is also the $R$ value for each element on the particular tile. Every $R$ value should occupy four different elements $M'_{i,j}$ in the tiled matrix. Let $i$ and $j$ be the $i$th row and $j$th column of matrix $M'$. The first step of our process is to get all the four-element data that holds the value $R$ by performing a row-wise search, from which we can bring in the two basic information. At the completion of searching process, we receive a data set composed with four-element values which is denoted as $M'_{i,j}$ where $k \in \{1, 2, 3, 4\}$. In each element set $M'_{i,j}$, there will be a row and column values which are stored on another variable as $iR_k$ and $jR_k$. Using those values, we calculated the tile value $T_v$ (2) from which we can get tile type and orientation information as shown in Figure 10. All viable $T_v$ and its corresponding $Q$ values are given in Figure 11. Once the process is completed, the information will be stored in the array $WP$. Algorithm 2 continues the same procedure by increasing the $R$ value until it runs out of tiles. The generated reference cells for the random matrix, which was illustrated in the last section is shown in Figure 12.

$$T_v = \left[\frac{i_1}{r_1} - \frac{i_2}{r_2}\right] + \left[\frac{i_2}{r_2} - \frac{i_3}{r_3}\right] + \left[\frac{i_3}{r_3} - \frac{i_4}{r_4}\right] + \left[\frac{j_1}{r_1} - \frac{j_2}{r_2}\right] + \left[\frac{j_2}{r_2} - \frac{j_3}{r_3}\right] + \left[\frac{j_3}{r_3} - \frac{j_4}{r_4}\right]$$

Figure 9. The random tile-set generated.
Figure 10. The tile value and global orientation for tile “T.”

| Tile Type | Orientation | $T_v$ | Reference element | $Q$ |
|-----------|-------------|-------|-------------------|-----|
| 0         | 0           | -7    | $M_{1/4}^{1/4}$   |     |
| 90        | 0           | -7    | $M_{1/3}^{1/3}$   |     |
| 180       | 0           | -7    | $M_{1/1}^{1/1}$   |     |
| 270       | 0           | -7    | $M_{1/2}^{1/2}$   |     |
| 0         | 90          | -6    | $M_{1/3}^{1/3}$   |     |
| 180       | 90          | -6    | $M_{1/1}^{1/1}$   |     |
| 270       | 90          | -6    | $M_{1/2}^{1/2}$   |     |
| 0         | 180         | -15   | $M_{1/4}^{1/4}$   |     |
| 90        | 180         | -15   | $M_{1/3}^{1/3}$   |     |
| 270       | 180         | -15   | $M_{1/1}^{1/1}$   |     |
| 0         | 270         | -15   | $M_{1/2}^{1/2}$   |     |
| 0         | 0           | -1    | $M_{1/3}^{1/3}$   |     |
| 180       | 180         | -17   | $M_{1/1}^{1/1}$   |     |
| 90        | 180         | -17   | $M_{1/2}^{1/2}$   |     |
| 270       | 180         | -17   | $M_{1/4}^{1/4}$   |     |
| 0         | 90          | -11   | $M_{1/3}^{1/3}$   |     |
| 180       | 90          | -11   | $M_{1/1}^{1/1}$   |     |
| 270       | 90          | -11   | $M_{1/2}^{1/2}$   |     |
| 0         | 270         | -11   | $M_{1/4}^{1/4}$   |     |
| 0         | 0           | -9    | $M_{1/3}^{1/3}$   |     |
| 180       | 0           | -9    | $M_{1/1}^{1/1}$   |     |
| 270       | 0           | -9    | $M_{1/2}^{1/2}$   |     |
| 0         | 90          | -5    | $M_{1/3}^{1/3}$   |     |
| 180       | 90          | -5    | $M_{1/1}^{1/1}$   |     |
| 270       | 90          | -5    | $M_{1/2}^{1/2}$   |     |
| 0         | 270         | -5    | $M_{1/4}^{1/4}$   |     |
| 0         | 0           | -16   | $M_{1/3}^{1/3}$   |     |
| 180       | 0           | -16   | $M_{1/1}^{1/1}$   |     |
| 270       | 0           | -16   | $M_{1/2}^{1/2}$   |     |
| 0         | 90          | -12   | $M_{1/3}^{1/3}$   |     |
| 180       | 90          | -12   | $M_{1/1}^{1/1}$   |     |
| 270       | 90          | -12   | $M_{1/2}^{1/2}$   |     |

Figure 11. hTetro robot’s $T_v$ values and corresponding $Q$ values.
on the grid-based wavefront algorithm. In this algorithm, we are using the $WP$ array as an input wherein we get the initial reference point. Further, we are using the $R$ value from the matrix $M^t$ to complete this process. Altogether we are arranging the series of $R$ values, and it represents $Q$ values. The algorithm assumes that the initial value of $R$ is 1 and its corresponding $Q$ values as a default starting point for the hTetro robot. Once we have the initial reference element, concerning that item, the algorithm looks for the smallest $R$ value in the neighboring elements. So the algorithm performs a $R$ value comparison between every neighboring element. The current reference location can determine the total set of $R$ values that are compared. The proposed algorithm undergoes three different searching methods, namely middle search, corner search, and edge search, to identify the smallest neighboring $R$ value as shown in the upper-left, upper-right, and bottom-left sub-figures in Figure 13, respectively. A different number of grids will be checked for the three searching strategies: three grids during an edge search, five in a corner search, and eight while a middle search is performed. The algorithm to find the nearby grid is shown in Algorithm 3. In the algorithm, the searching method will push all possible nearby grids into a grid array $G$. Once the corresponding reference of neighbor grids are visited, the algorithm will proceed on to a row–column search which is as depicted in the bottom-right sub-figure in Figure 13. The system starts the search from the current grid location and spreads to horizontal and vertical directions until it founds a valid grid cell whose reference coordinate has yet been visited. Once the grid array $G$ has all feasible grids listed, the algorithm will proceed a simple loop function which will find the reference coordinate in the array that holds the smallest $R$ value. The chosen reference coordinate will then be pushed to the tiling motion waypoint array $WP_{tm}$, and one will be added to the current $R$ value so that the next waypoint searching process will begin until the algorithm runs out of $R$ values and all unsequenced waypoints have been sequenced in $WP_{tm}$.

**Navigation strategy**

Given a sequenced robot $WP_{nav}$, we have developed a navigation algorithm that generates the connection between waypoints and creates a fully connected navigation route for robot navigation within the entire navigation process. The navigation algorithm is described in detail in Algorithm 4. In the case of our proposed tiling motion planning and sequencing, waypoint array $WP_{tm}$ will be used; while during the simulation phase, several other sequencing algorithms and waypoint arrays will be taken as input in Algorithm 4.

In the developed navigation strategy, we refer to the connection between every waypoint pairs as “paths,” while we call the connection between all paths “route.”
Algorithm 4, we generate paths using function PATHGEN, and we connect them in specific sequences utilizing function ROUTEGEN, which eventually generates an entire route in the workspace for the hTetro to follow once the navigation begins.

In function PATHGEN, a virtual robot located at the position of \( rbtX, rbtY \) (the initial robot position before path generation) with velocity \( vel \) is created to generate the path between two waypoints. The robot initializes its position at the initial waypoint \( Q_i \) and updates its morphology according to the starting configuration as well. The algorithm first checks whether any morphology reconfiguration is required by comparing its current morphology to the morphology at the final waypoint \( Q_f \). If a different configuration is detected, the robot will prioritize the robot transformation compared to the linear movements. The platform will check whether the surrounding environment is providing enough space by executing the function CAN-TRANSFORM check. If no nearby obstacles are presented and the robot transformation command is valid, the transformation command, which is represented as “TRANSFORM \((q_1, q_2, q_3, q_4)\)” in the algorithm, will be pushed to Path. If an hTetro transformation command is not valid at the current virtual robot position, it will proceed to the second
movement command and execute the transformation check again once it reaches a new position in the workspace. The movement command is the crucial portion of the \texttt{pathGen} function, which will move the virtual robot in a small distance towards the next waypoint either in \(X\) or \(Y\) direction, whichever yields a larger distance difference between the two waypoints until the robot arrives at the position of the final waypoint. The moved distance is determined based on the robot velocity \(vel\). Once the robot path segment is selected, it will be pushed to the \texttt{Path} as well, issuing a movement command, represented as \texttt{"MOVE (x, y)"} to the robot once the navigation begins. The process will repeat until the virtual robot arrives at the position of the final waypoint with the corresponding robot morphology, and the generated path will be utilized to construct an entire robot route.

Finally, the function \texttt{routeGen} will execute a loop which search for the correct sequence that connects the generated paths from the \texttt{pathGen} function and push them into a route array (\texttt{route}) as output. During robot navigation, the robot will continuously follow the commands in \texttt{route} and will eventually cover the entire workspace area with the desired waypoint sequences.

**Evaluation of tiling motion**

As part of this study, we have evaluated the proposed motion planning on the hTetro robot in a simulated environment. In order to analyze the performance of the proposed motion planning algorithm, we will compare the results with several algorithms that tackle the waypoint sequencing as well. The simulator adopts all the algorithms to guide the robot navigation within the workspace. For algorithms, we directly adopt into the simulator without making any adjustments to the algorithm. The simulations are conducted using MATLAB Simulink software.

This section will firstly introduce the six path planning strategies that we have selected as our benchmarking algorithms. The evaluation criteria will then be introduced. Finally, the workspace setups under three scenarios, and the navigation results in these scenarios will be presented.

**Benchmarking path planning strategies**

In order to accomplish CPP tasks, many algorithms have been proposed for robots with a single fixed morphology, each of which is constructed based on a specific cellular decomposition method. Among all path planning algorithms that adopted the grid-based cellular decomposition technique, zigzag path (or boustrophedon path) and spiral path are the two popular navigation strategies for the robot to accomplish area coverage. The conventional zigzag path algorithm explores the area with a constant sweeping width and performs a 180° turn when an obstacle is encountered and move on to the next sweeping lane to avoid the swept areas. While the conventional spiral path algorithm works under a similar framework, where the robot performs 90° turns in a fixed direction when an obstacle is presented or when the robot enters areas that have already been covered, resulting in a spiral-shaped navigation path with diminishing width as time goes on. Both algorithms are capable of producing effective results in terms of minimizing the amount of area revisited by the robot platform.

Nevertheless, unlike the proposed motion planning algorithm which makes use of waypoint sequencing to navigate within the workspace, the conventional zigzag and spiral algorithms do not guarantee complete coverage of the entire area, so minor adjustments to the existing conventional zigzag and spiral path algorithms have to be made for our hTetro robot simulation model in order to make meaningful comparison between different navigation strategies. To achieve this goal, we make use of the unsequenced waypoint array \(WP\) that was created previously and produce different sequencing patterns for robot navigation. In this article, a total of eight benchmarking algorithms are introduced: conventional zigzag algorithm, conventional spiral algorithms, and six different sequencing patterns generated based on the generated waypoint array, including random path algorithm, greedy path algorithm, two zigzag-pattern-based algorithms, and two spiral-pattern-based algorithms.

Here we adopt a greedy search and random search algorithms directly into our model without making any adjustments to the algorithm. From the starting point, the greedy search algorithm will continuously explore the workspace and assign the nearest configuration like the following waypoint, while the random search assigns the next waypoint from the unsequenced waypoint set at random. The zigzag-pattern- and spiral-pattern-based algorithm are being described in Algorithms 5 and 6, respectively. These modified algorithms now take the unsequenced waypoint series \(WP\) and a sweeping width \(n_{\text{wid}}\) as input and produce sequenced waypoint series \((WP_{\text{zigzag}}\ or\ WP_{\text{spiral}})\) as output. The modified algorithms will sweep the space based on conventional zigzag and spiral path algorithm. Nevertheless, the robot will no longer follow the swept path in the adjusted algorithms, instead, it will follow the sequence of the waypoints that are being swept by either zigzag or spiral algorithm. By making this minor adjustment, we can ensure full area coverage of the generated routes in zigzag-pattern- and spiral-pattern-based algorithms. Considering that traditional zigzag and spiral algorithms are sweeping the environment in single unit grid width, we introduce the sweeping width \(n_{\text{wid}}\) so as to better describe the scenario during the navigation process of hTetro robot. In all seven feasible hTetro morphologies, the average width that is being swept by hTetro robot is around 2 unit grid width, so we have selected \(n_{\text{wid}} = 1\) and \(n_{\text{wid}} = 2\) for both zigzag-pattern- and spiral-pattern-based navigation as benchmarking algorithms.
Evaluation criteria

We propose two evaluation criteria to find the most efficient algorithm: average distance traveled and average grid coverage time. Both criteria are presented in Algorithm 7. Average distance traveled is determined based on the trajectories of all four hTetro blocks during the navigation process. In the algorithm, the distance traveled by each hTetro block within every time instance will be calculated and stored in \( \text{disttot} \) until the navigation terminates. To calculate the average grid coverage time of a navigation strategy, we assume that every single grid in the workspace requires minimal 1 s to be covered by an hTetro block. We then decompose each grid in the workspace into 10 \( \times \) 10 smaller square-shaped pieces referred to as “pixels.” These pixels will provide a better resolution of the total amount of time the robot spends within the workspace. The grid coverage time is stored in an array \( \text{Tcvg} \) in Algorithm 7. Since the hTetro robot is moving at a constant speed during the simulation, the software will calculate the time hTetro blocks used to cover each pixel by sampling the entire workspace every time interval of \( \text{tintv} \). Once the navigation terminates, the average grid coverage time will be determined accordingly by calculating the mean value of \( \text{Tcvg} \) within all accessible pixels in the workspace.

The average grid coverage time is an important criterion which determines the efficiency of the proposed path planning algorithm in terms of area recovered.

Experiments with the proposed technique

In this experiment, a total of three scenarios are being tested to analyze the performance of the algorithms. The workspace of these scenarios is being illustrated in Figure 14. Scenario 1 illustrates an empty workspace of 10 \( \times \) 10 grids. The grid width \( d_{grid} \) is set to 25 cm. Scenario 2 and Scenario 3 are workspaces with scattered obstacles. The workspace size is 11 \( \times \) 11 and 11 \( \times \) 14 grids, respectively; while the grid width remains the same.

With the grid map created, the simulator runs the global tiling set algorithm as described in the fourth section to tile the grid space. The matrix \( M^f \) is being created for each scenario according to the algorithm. Elements in these matrices will hold a negative value of \( -1 \) if an obstacle is presented at the position. The fully tiled matrices \( M^f \) are being presented in Figure 15. It is observed that in Scenario 1, the combination of “T,” “S,” “Z” tetrominoes are being used to complete the tiling task based on Theorem 0.1. Since the workspace in Scenario 2 does not fulfill the requirements in Theorem 0.1, a different combination of “O,” “L,” “J” tetrominoes is being utilized instead.
Algorithm 7. Algorithm performance evaluation criteria.

1: function DISTTRAVEL(Q_c, Q_p, d_block, dist_new)
2: dist_new ← 0
3: for n ← 1 to 4 do
4: dist_new ← dist_new + norm(getBLKCENTER(n, Q_c) - getBLKCENTER(n, Q_p)) × d_block
5: end for
6: dist_tot ← dist_new + 0.25 × dist_new
7: return dist_tot
8: end function

10: function GRIDCVGTIME(T_c, Q_c, d_block, t_h)
11: for all pixel ∈ T_c do
12: isCovered ← false
13: for n ← 1 to 4 do
14: A_blk ← getBLKAREA(n, Q_c, d_block)
15: if pixel.Position ∩ A_blk = ∅ then
16: isCovered ← true
17: end if
18: end for
19: if isCovered then
20: newpixel.Position ← pixel.Position
21: newpixel.Value ← pixel.Value + t_h
22: T_c[pixel] ← newpixel
23: end if
24: end for
25: return T_c
26: end function

according to Theorem 0.3; while Scenario 3 is being tiled based on Theorem 0.2.

The completion of the tile matrices leads the simulator to generate the reference points for each tile piece as described in the fifth section, with the results shown in Figure 16. Once the unsequenced waypoint arrays WP are generated, the proposed tiling motion and the benchmarking algorithm will be run to determine the unique sequences of the waypoints be visited. We define the upper left robot morphology as the starting robot waypoint for all seven algorithms in each scenario to eliminate the influence caused by random starting positions. The result of the generation phase of the proposed tiling motion algorithm is shown in Figure 17. The exact waypoint navigation sequence generated by the other six sequences are not presented in this article, but the efficiency performances are still presented and analyzed in the following section.

Experimental setups

As part of this research, we conducted experimental studies to validate the proposed tiling motion with the hTetro robot under the mentioned scenarios. The first experiment tested Scenario 1, wherein the test area consisted of a rectangular plot of 250 cm × 250 cm (Figure 18). Similarly, we chose an area of 275 cm × 275 cm and 350 cm × 275 cm for Scenario 2 and Scenario 3, respectively. We placed the obstacles inside the test area of Scenarios 1 and 2, as illustrated in the workspace map. The experimental setup consisted of a predefined floor area with a boundary construed by 5 mm thick foam sheet walls with a height of 40 cm. The boundaries of the defined area were adjusted according to the workspace scenarios. We housed an overhead image-acquisition device to record the area coverage process to calculate the percentage area recovered by the hTetro robot during the experimental trials. The imaging device was mounted in such a way to cover the entire workspace, as shown in Figure 18. The resulting raw video files were post-processed to analyze the performance of the robot. From the image, it can be seen that the frame contains perspective distortion. It is difficult to track the robot when the perspective of the camera is a bit slanted. So, we used a perspective distortion elimination algorithm to track the

Figure 14. Workspace map of the three evaluation scenarios. (a) Scenario 1, (b) Scenario 2, and (c) Scenario 3.
Figure 15. Tiled matrices $M^i$ in the three scenarios. (a) Scenario 1, (b) Scenario 2, and (c) Scenario 3.

Figure 16. Generated reference coordinates in the three scenarios. (a) Scenario 1, (b) Scenario 2, and (c) Scenario 3.

Figure 17. Produced navigation paths in the three scenarios. (a) Scenario 1, (b) Scenario 2, and (c) Scenario 3.
robot in every frame and generate a coverage map. We pasted red color paper on top of each block of the hTetro that aids in tracking. After we identify the red blobs in every frame, we identify the centroid of the red regions. For each identified centroids, green-colored squares are plotted in a clear reference image. Whenever the detected centroids enter the covered area, the algorithm would plot lighter red squares instead of green for that particular centroid. So in a coverage map, if there are darker red regions, means the robot visited that particular place more frequently. After generating the coverage maps on the reference images, the percentage of the area recovered was calculated using equation (3). The video post-processing and generation coverage map were executed in Matlab 2017 software

\[ P_{rc} = \frac{P_{red} \times 100}{P_{area}} \]  

where \( P_{rc} \) is percentage of area recovered, \( P_{red} \) is the total red pixel in the coverage map and \( P_{area} \) is the total pixel of the coverage map.

**Results**

**Simulated results**

During the navigation process, the average distance traveled and the grid coverage time is consistently calculated through Algorithm 7.

Based on the values calculated, bar charts are being plotted to compare the results between each algorithm, which is as shown in Figures 19 and 20. In Figure 19, the average grid coverage time in all three scenarios are presented. All algorithms demonstrate better performance in Scenario 1 since it is an obstacle-free environment, and areas are less likely to be revisited. With \( n_{row} = 2 \) chosen, the zigzag- and spiral-pattern navigation algorithms outperform the algorithms with \( n_{row} = 1 \). The performance of the presented tiling motion algorithm is highly competitive, with rather low average coverage time in all three scenarios compared to other algorithms. The average distance traveled outcome is illustrated in Figure 19. The distance traveled in all nine algorithms increases in Scenario 2 and Scenario 3 compared to Scenario 1 due to an increase of the workspace size and the extra paths taken to avoid obstacles within the environment. The proposed tiling motion algorithm has demonstrated the shortest average distance traveled in all three scenarios compared to the rest of the algorithms.

To gain a better understanding of the navigation paths taken by each algorithm and how they pose influences in the performance of the algorithm, we create robot coverage heat maps to visualize the data better. A robot coverage heat map is created once the navigation process terminates and is constructed based on the final coverage values stored in \( T_{avg} \). In the heat map, areas that are being covered by the hTetro robot are being represented in a color spectrum between yellow to red. The intensity of
the red increases when more time is being spent for the robot to cover the area, indicating that the corresponding grid is being visited several times throughout the entire navigation process, while areas that are not being covered will remain black. In this article, the coverage heat maps of Scenario 1 and Scenario 2 are being plotted, which is as shown in Figures 21 to 23.

**Experimental results**

Before each experimental run, the hTetro robot was placed in a predefined starting point within the test area. The robot uses a hector SLAM\(^4\) technique to localize inside the workspace and navigates accurately on the waypoints. The coverage motion of the hTetro robot was recorded while experimenting. At the end of each experimental run, the image-processing algorithm mentioned in the previous section was used to process the recorded videos to produce the robot coverage map, and its respected area recovered values. Figure 24 illustrates the robot tracked map with hTetro, which was generated from the video recorded from the experiment validating the first scenario. In the image, the green shaded pixels represent the area tiled by the hTetro, and the red shaded parts show area recovered. Figure 24 contains five set of images, two in each set that represents the hTetro robot’s tiling motion at the various time point of the experiment validating Scenario 1. In each set of images, the even pictures represent the hTetro robot’s position at a

![Figure 21. Coverage heat map in Scenario 1. (a) Random path. (b) Conventional zigzag pattern. (c) Conventional spiral pattern. (d) Zigzag pattern \(n\text{row} = 1\). (e) Spiral pattern \(n\text{row} = 1\). (f) Zigzag pattern \(n\text{row} = 2\). (g) Spiral pattern \(n\text{row} = 2\). (h) Greedy path. (i) Tiling motion.](image-url)
particular time point, and the odd picture indicates hTetro coverage map at the same instant. Similar set of images for the experimental Scenarios 2 and 4 are illustrated in Figures 25 and 26. Similarly, the coverage map for other algorithm under all three scenarios is shown in Figures 27 to 29.

**Discussion**

In this section, we discuss the advantage of our proposed motion planning technique for the hTetro robot by comparing with other path planning algorithms.

In the case of conventional zigzag and spiral algorithms, the hTetro robot maintains as a fixed O-shape (default Morphology) and attempts to cover the entire workspace. The coverage performance of these algorithms is similar to waypoint-based zigzag- and spiral-pattern algorithms with \( n_{row} = 1 \) and the total distance traveled rivals the ones with \( n_{row} = 2 \) in Scenario 2. However, it is important to take note that due to the fixed-morphology navigation for conventional zigzag and spiral algorithms, the robot might not be able to cover the entire workspace. For instance, the upper-right corner of both Figure 22(b) to (c) shows dark areas that an O-shaped hTetro could not arrive at, which could potentially result in a reduction in the total area covered throughout the navigation process. Such a reduction in area coverage would be more when the robot is deployed in a more obstructed real-time environment.
In both waypoints-based zigzag- and spiral-pattern algorithms, the robot follows the designated path to navigate within the environment. The advantage of these algorithms is similar to conventional zigzag and spiral algorithms, which provides a straightforward method to perform the coverage task within the designated space. These algorithms tend to perform consistent results in larger maps since the performance is less likely to be affected by one or a few unsequenced waypoints within the workspace. The introduction of the revised zigzag and spiral algorithms (based on Algorithms 5 and 6) ensures that the entire area coverage can be achieved. Figures 21(d) to (g), 22(d) to (g), and 23(d) to (g) all demonstrate that by increasing the navigation width from $n_{row} = 1$ to $n_{row} = 2$, the red-colored areas will significantly decrease. This implies that the navigation efficiency has been improved since fewer grids are being revisited considering that during the navigation process of the hTetro platform the robot tends to sweep two grids at a time instead of just a single grid. Since the hTetro system is deployed for the floor cleaning task, the platform is allowed to bump into the obstacles similar to the commercial residential floor cleaning robot. Due to this consideration, in every heat map generated, we can witness some overlaps on the obstacles.

The random path algorithm, though being able to successfully cover the entire workspace, has shown to be an overall ineffective navigation strategy since the robot is constantly revisiting several grids in the

---

**Figure 23.** Coverage heat map in Scenario 3. (a) Random path. (b) Conventional zigzag pattern. (c) Conventional spiral pattern. (d) Zigzag pattern ($n_{row} = 1$). (e) Spiral pattern ($n_{row} = 1$). (f) Zigzag pattern ($n_{row} = 2$). (g) Spiral pattern ($n_{row} = 2$). (h) Greedy path. (i) Tiling motion.
workspace due to the randomly chosen waypoint sequence, which results in the highest average coverage time and the longest distance travelled among all seven algorithms in all scenarios.

The greedy search algorithm generally performs well with a small sample size of waypoint sequences. Nevertheless, the greedy search algorithm gets punished by leaving several of the suboptimal waypoints behind during

**Figure 24.** Coverage map for tiling motion in Scenario 1.

**Figure 25.** Coverage map for tiling motion in Scenario 2.
the navigation process. If a few waypoints are being ignored during the sequencing process, the robot has to travel a much longer distance to cover these waypoints at the end of the navigation process. The greedy algorithm also faces some issues when obstacles are being introduced on the map. In Figure 23(h), the bottom-left corner has been revisited multiple times compared to the results in Figure 23(g) and (i). This is due to an unwise selection of the navigation route in the greedy algorithm that guides the robot to navigate from one side of the bottom-left obstacle in Scenario 3 to the other side (since it is the closest waypoint that can be discovered), resulting in undesirable average grid coverage time around obstacles. Also, the greedy search algorithm takes more running time to complete the path generation for the generated waypoint which is undoubtedly larger than the previous algorithms. Table 1 provided the numerical comparison for path generation of different coverage method in three scenarios. It is worth to note that zigzag and spiral methods connect the predefined waypoint orderly; on the other hand, random search and greedy search depend on the number of trials to find an optimal solution. Due to this reason, the conventional algorithms have lesser path generation time than the greedy and random search.

On the other hand, the proposed tiling motion planning algorithm has demonstrated a great performance with the shortest distance traveled and decent time grid coverage time compared to the other algorithms in the three scenarios presented as shown in both Figures 19 and 20. The grid coverage heat maps shows that the proposed algorithm might face some difficulties navigating through narrow spaces (as shown in the red-colored area in Figure 23(i)) but performs well in workspaces with large and empty areas as illustrated in Figures 21(i) and 23(i). When it comes to path generation time, the proposed motion planning technique took slightly more time than traditional zigzag and spiral with least recover area, but significantly lower than random search and greedy search.

Concerning the experimental trials, the total area recovered for the generated coverage map for all considered path planning algorithms for every scenario is shown as a trend in Figure 30. In the coverage map (Figures 27 to 29), the red regions indicate the area that was recovered by the robot, and the green region is the area with no overlaps. The percentage of area recovered was computed from the coverage map using the equation (3). The percentage of area recovered from the coverage map, and the grid coverage time from the heat map was computed differently. Due to this reason, the values from both trials will differ very much and cannot be compared. However, in the discussion below, we are going to compare the red regions generated between grid time map and coverage map. On the other hand, from the coverage map images, we can notice there are some overlapping regions on the obstacles which are caused due to the robot position mismatch during the tracking process. All most every experimental trial and its corresponding results were supporting our previous arguments on simulated outcomes. For instance, the coverage map that was generated for both conventional zigzag and spiral motion contains more than 90% of the

![Coverage map for tiling motion in Scenario 3.](image)
red region in all considered scenarios, which is almost equal to the simulations. Also, in the experimental trials, the robot cannot reach the narrow space in Scenario 2, as shown in Figure 31. The experimental trials that concern the waypoint-based zigzag and spiral motions produce more area recovery regions in the coverage map. The total area of the red region in the coverage map for zigzag and spiral pattern \( n_{\text{row}} = 1 \) was larger than 93% for Scenario 1, 92% and 93% for Scenario 2, 87% and 89% for Scenario 3, respectively. For the zigzag and spiral pattern \( n_{\text{row}} = 2 \), the area recovered are bit lower with the percentage of 86% and 82% in Scenario 1, 81% and 76% in Scenario 2, 87% and 76% in Scenario 3, respectively. Similarly, on the random path trials, the same result was reflected like in simulations, wherein most darker red regions were generated on the coverage map. The area recovered percentage for random path algorithm in the coverage map was more than 95% in every scenario. The whole red region was generated on the coverage map using greedy search algorithm was 62%, 55%, and 43% for Scenarios 1, 2, and 3, respectively, which is significantly lesser than previous algorithms. Finally, for the tiling motion, the experimental results support the simulated results by having less red regions in the generated coverage map. Interestingly, in the experimental trials, the tiling motion has more area recovered region in Scenario 1 than the Scenarios 3 and 2, which consist of more obstacles. This result is due to the real-time

![Figure 27. Coverage map for other algorithms in Scenario 1. (a) Random path. (b) Conventional zigzag. (c) Conventional spiral. (d) Zigzag pattern \( (n_{\text{row}} = 1) \). (e) Spiral pattern \( (n_{\text{row}} = 1) \). (f) Zigzag pattern \( (n_{\text{row}} = 2) \). (g) Spiral pattern \( (n_{\text{row}} = 2) \). (h) Greedy search.](image-url)
experiments, during tiling motion in Scenarios 3 and 2, the hTetro robot traveled on the same covered area many times to avoid the obstacles which result in generating more darker red regions (shown in Figures 25 and 26). At the same time in Scenario 1, the hTetro recovered more region with less repetition, which results in generating lighter red regions (shown in Figure 24). It only generates lesser than 56% of red regions in Scenario 1, 43% in Scenario 2, and 37% in Scenario 3. Thus, the proposed technique could significantly increase the area covered with a lesser recovery rate. Overall, based on the outcome of the proposed tiling motion algorithm in this experimental validation, there is significant untapped research and development opportunity related to the tiling robots within the area coverage problem.

**Conclusion**

This article introduced a motion planning technique to achieve maximum area coverage in a Tetris-inspired shape-shifting robotic floor cleaner named hTetro that was based on the polyomino tiling theory. We discussed an autonomous tile-set generation based on the tiling theory concept and introduced the path generation for the hTetro robot to navigate on the generated tiling set with an objective of maximizing the area coverage. We validated the proposed algorithm by benchmarking its performance with
We first conducted simulated trials for all considered algorithms. At the end of the trials, the waypoint navigation map and the coverage heat map are being generated, which illustrates the total area recovered by the robot under different coverage algorithms. Also, we conducted real-time experiments in a fixed setting that replicates the scenarios. In each experiment, the robot cleared the sequenced waypoints while assuming its morphology at each point. We tracked the robot motion using an overhead camera in order to generate a coverage map to compute the total area recovered for each experimental case. Both simulation and

**Figure 29.** Coverage map for other algorithms in Scenario 3. (a) Random path. (b) Conventional zigzag. (c) Conventional spiral. (d) Zigzag pattern \(n_{\text{row}} = 1\). (e) Spiral pattern \(n_{\text{row}} = 1\). (f) Zigzag pattern \(n_{\text{row}} = 2\). (g) Spiral pattern \(n_{\text{row}} = 2\). (h) Greedy search.

**Table 1.** Comparison of path generation time for all considered coverage algorithms.

| Algorithm          | Scenario 1 | Scenario 2 | Scenario 3 |
|--------------------|------------|------------|------------|
| Conventional zigzag| 0.092      | 0.102      | 0.114      |
| Conventional spiral| 0.108      | 0.125      | 0.156      |
| Zigzag pattern \(n_{\text{row}} = 1\) | 1.0906     | 1.3128     | 1.5306     |
| Spiral pattern \(n_{\text{row}} = 1\) | 1.7344     | 1.9504     | 2.1712     |
| Zigzag pattern \(n_{\text{row}} = 2\) | 1.5064     | 1.7178     | 1.9306     |
| Spiral pattern \(n_{\text{row}} = 2\) | 1.9748     | 2.2386     | 2.4284     |
| Random search      | 21.789     | 22.965     | 23.345     |
| Greedy search      | 30.233     | 32.134     | 33.987     |
| Tiling motion      | 3.0988     | 3.3774     | 3.4264     |

traditional CPP techniques under three different scenarios.

We first conducted simulated trials for all considered algorithms. At the end of the trials, the waypoint navigation map and the coverage heat map are being generated, which illustrates the total area recovered by the robot under different coverage algorithms. Also, we conducted real-time experiments in a fixed setting that replicates the scenarios. In each experiment, the robot cleared the sequenced waypoints while assuming its morphology at each point. We tracked the robot motion using an overhead camera in order to generate a coverage map to compute the total area recovered for each experimental case. Both simulation and
experimental results establish that the proposed tiling motion technique achieves maximum coverage area with the less recovered area.

Future research work is set to focus on the following areas: (1) Optimizing the reference point generation; (2) the tile-set generation with respect to energy cost in order to reduce the number of reconfiguration to complete the full area coverage; (3) studying locomotion with trajectory tracking and following while performing the tiling motion on the generated tile-set; (4) implementing the proposed approach for other polyforms reconfigurable robots; (5) applying tiling theory under the cellular decomposition technique where we apply different tiling theory for each decomposed cell in a complex environment; and (6) studying long-term autonomy with the tiling motion on a physical hTetro platform.

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

Funding
The author(s) disclosed receipt of the following financial support for the research, authorship, and/or publication of this article: This research is supported by the National Robotics R&D Programme Office, Singapore under its grant RGAST1907 and administered by the Agency for Science, Technology and Research.

ORCID iDs
Prabakaran Veerajagadheswar https://orcid.org/0000-0002-5612-2619
Anh Vu Le https://orcid.org/0000-0002-4804-7540

References
1. Marketsandmarkets.com. Cleaning robot market by type, product (floor-cleaning robot, lawn-cleaning robot, pool-cleaning robot, window-cleaning robot), application (residential, commercial, industrial, healthcare), and geography—global forecast to 2023. Report Code: SE 3777, January 2018 (Online), https://www.marketsandmarkets.com/Mar
1. Butler ZJ, Rizzi AA, and Hollis RL. Contact sensor-based coverage algorithm of rectilinear environments. In: *Proceedings of the IEEE/RSJ international conference on intelligent robots and systems (IROS 2003)*, Las Vegas, NV, 27–31 October 2003, pp. 1679–1684. IEEE.

2. Galceran E and Carreras M. Planning coverage paths on bathymetric maps for in-detail inspection of the ocean floor. In: *2013 IEEE international conference on robotics and automation (ICRA)*, Karlsruhe, 6–10 May 2013, pp. 4159–4164. IEEE.

3.  Le AV, Arunmozhi M, Veerajagadheswar P, et al. Complete coverage of an unstructured environment by a mobile robot. In: *Proceedings of the IEEE/RSJ international conference on intelligent robots and systems (IROS 2003)*, Las Vegas, NV, 27–31 October 2003, pp. 1685–1690. IEEE.

4. Butler ZJ, Rizzi AA, and Hollis RL. Contact sensor-based coverage of rectilinear environments. In: *Proceedings of the IEEE international symposium on intelligent control/semiotics*, Cambridge, MA, 17–17 September 1999, pp. 266–271.

5. Zelinsky A, Jarvis RA, Byrne JC, et al. Planning paths of complete coverage of an unstructured environment by a mobile robot. In: *Proceedings of international conference on advanced robotics*, vol. 13, Tokyo, Japan, November 1993, pp. 533–538.

6. Gabriely Y and Rimon E. An on-line coverage algorithm of grid environments by a mobile robot. In: *2002 IEEE conference on robotics & automation*, Washington, DC, 11–15 May 2002.

7. Paull L, Saeddi S, Li H, et al. An information gain based adaptive path planning method for an autonomous underwater vehicle using sidescan sonar. In: *2010 IEEE international conference on automation science and engineering (CASE 2010)*, Toronto, 21–24 August 2010, pp. 835–840.

8. Fink J, Bauwens V, Kaplan F., et al. Living with a vacuum cleaning robot. *Int J Soc Robot* 2013; 5(3): 389–408.

9. Wong SC and MacDonald BA. A topological coverage algorithm for mobile robots. In: *Proceedings of the IEEE international conference on robotics and automation (ICRA)*, Singapore, 29 May–3 June 2017, pp. 6105–6112. IEEE.

10. Le AV, Prabakaran V, Sivanantham V, et al. Modified a-star algorithm for efficient coverage path planning in Tetris inspired self-reconfigurable robot with integrated laser sensor. *Sensors* 2018; 18(8): 2585.

11. Oksanen T and Visala A. Coverage path planning algorithms for agricultural field machines. *J Field Robot* 2009; 26(8): 651–668.

12. Choset H, Acar E, Rizzi AA, et al. Exact cellular decompositions in terms of critical points of Morse functions. In: *Proceedings of the IEEE international conference on robotics and automation (ICRA ’00)*, vol. 3, San Francisco, CA, USA, 24–28 April 2000, pp. 2270–2277.

13. Cann J. Constructing roadmaps of semi-algebraic sets I: completeness. *Artif Intell* 1988; 37(1–3): 203–222.

14. Cheng P, Keller J, and Kumar V. Time-optimal UAV trajectory planning for 3D urban structure coverage. In: *IEEE/RSJ international conference on intelligent robots and systems*, Nice, 22–26 September 2008, pp. 2750–2757. IEEE.

15. Hsu PM, Lin CL, and Yang MY. On the complete coverage path planning. *IEEE Tran Syst Man Cybern Part B (Cybern)* 2004; 34(1): 718–724.

16. Cheng P, Keller J, and Kumar V. Time-optimal UAV trajectory planning for 3D urban structure coverage. In: *IEEE/RSJ international conference on intelligent robots and systems (IROS 2008)*, Nice, 22–26 September 2008, pp. 2750–2757. IEEE.

17. Xu L. (2011). *Graph planning for environmental coverage*. Ph.D. Thesis, Carnegie Mellon University, 2011.

18. Galceran E and Carreras M. Planning coverage paths on bathymetric maps for in-detail inspection of the ocean floor. In: *2013 IEEE international conference on robotics and automation (ICRA)*, Karlsruhe, 6–10 May 2013, pp. 4159–4164. IEEE.

19. Jin J and Tang L. Coverage path planning on three-dimensional terrain for arable farming. *J Field Robot* 2011; 28(3): 424–440.

20. Gonzalez E, Alarcon M, Aristizabal P, et al. BSA: a complete coverage path planning algorithm for robots. *Robot Auton Syst* 2013; 61(12): 1258–1276.

21. Tan N, Mohan RE, and Elangovan K. A bio-inspired reconfigurable robot. In: *Advances in reconfigurable mechanisms and robots II*. Cham, Switzerland: Springer, 2016, pp. 483–493.

22. Galceran E and Carreras M. A survey on coverage path planning for robotics. *Robot Auton Syst* 2013; 61(12): 1258–1276.

23. Hsu PM, Lin CL, and Yang MY. On the complete coverage path planning for mobile robots. *J Intell Robot Syst* 2014; 74(3–4): 945–963.

24. Gonzalez E, Alarcon M, Aristizabal P, et al. BSA: a coverage algorithm. In: *Proceedings 2003 IEEE/RSJ international conference on intelligent robots and systems (IROS 2003)*, Las Vegas, NV, 27–31 October 2003, pp. 1685–1690. IEEE.

25. Le AV, Arunmozhi M, Veerajagadheswar P, et al. Complete path planning for a Tetris-inspired self-reconfigurable robot by the genetic algorithm of the traveling salesman problem. *Electronics* 2018; 7(12): 344.

26. Hamed IA, Bochits DD, and Sorensen CG. Driving angle and track sequence optimization for operational path planning using genetic algorithms. *Appl Eng Agric* 2011; 27(6): 1077–1086.

27. Hsu PM, Lin CL, and Yang MY. On the complete coverage path planning for mobile robots. *J Intell Robot Syst* 2014; 74(3–4): 945–963.

28. Dornhege G, Laubichler M, and Srinivasan N. Hinged dissections: swinging and twisting. *Am Math Month* 1954; 61(10): 675–682.

29. Frederickson GN. Hinged dissections: swinging and twisting. Cambridge: Cambridge University Press, 2002.
32. Jho CW and Lee WH. Video puzzle game application of polyomino re-tiling. In: Embedded and multimedia computing technology and service. Dordrecht, Netherlands: Springer, 2012, pp. 363–369.
33. Lo KY, Fu CW, and Li H. 3D polyomino puzzle. In: ACM transactions on graphics (TOG), vol. 28(5). New York: ACM, 2009, p. 157.
34. Lester C. Tiling with T and skew tetrominoes. Querqus Linfield J Under 2012; 1(1): 3.
35. Nitica V. The tilings of deficient squares by ribbon L-tetrominoes are diagonally cracked. Open J Discrete Math 2017. arXiv preprint arXiv:1701.00419.
36. Gfredericks.com. A polyomino tiling algorithm, https://gfredericks.com/gfrlog/99 (2018, accessed 11 April 2020).
37. Englot B and Hover FS. (2012, June). Sampling-based coverage path planning for inspection of complex structures. In Icaps.
38. Choset HM, Hutchinson S, Lynch KM, et al. Principles of robot motion: theory, algorithms, and implementation. Cambridge: MIT press, 2005.
39. Wong SC, Middleton L, MacDonald BA, et al. Performance metrics for robot coverage tasks. In: Proceedings of Australasian conference on robotics and automation, Auckland, New Zealand, 27–29 November 2002, vol. 27, p. 29.
40. Kohlbrecher S, Meyer J, Graber T, et al. (2014). RoboCupRescue 2014—Robot League Team Hector Darmstadt (Germany).