Automation in Construction 112 (2020) 103078

Contents lists available at ScienceDirect

Automation in Construction

journal homepage: www.elsevier.com/locate/autcon

Complete coverage path planning using reinforcement learning for Tetromino based cleaning and maintenance robot

Anirudh Krishna Lakshmanan[a,b], Rajesh Elara Mohan[a], Balakrishnan Ramalingam[a], Anh Vu Le[c,*], Prabahar Veerajagadeshwar[a], Kamlesh Tiwari[b], Muhammad Ilyas[a]

[a] ROAR Lab, Engineering Product Development, Singapore University of Technology and Design, Singapore 487372, Singapore
[b] Computer Science Department, Birla Institute of Technology and Science, Pilani 333031, India
[c] Optoelectronics Research Group, Faculty of Electrical and Electronics Engineering, Ton Duc Thang University, Ho Chi Minh City 700000, Vietnam

ARTICLE INFO

Keywords:
Tiling robotics
Cleaning and maintenance
Inspection
Path planing
Reinforcement learning

ABSTRACT

Tiling robotics have been deployed in autonomous complete area coverage tasks such as floor cleaning, building inspection, and maintenance, surface painting. One class of tiling robotics, polyomino-based reconfigurable robots, overcome the limitation of fixed-form robots in achieving high-efficiency area coverage by adopting different morphologies to suit the needs of the current environment. Since the reconfigurable actions of these robots are produced by real-time intelligent decisions during operations, an optimal path planning algorithm is paramount to maximize the area coverage while minimizing the energy consumed by these robots. This paper proposes a complete coverage path planning (CCPP) model trained using deep blackreinforcement learning (RL) for the tetromino based reconfigurable robot platform called hTetro to simultaneously generate the optimal set of shapes for any pretrained arbitrary environment shape with a trajectory that has the least overall cost. To this end, a Convolutional Neural Network (CNN) with Long Short Term Memory (LSTM) layers is trained using Actor Critic Experience Replay (ACER) reinforcement learning algorithm. The results are compared with existing approaches which are based on the traditional tiling theory model, including zigzag, spiral, and greedy search schemes. The model is also compared with the Travelling salesman problem (TSP) based Genetic Algorithm (GA) and Ant Colony Optimization (ACO) schemes. The proposed scheme generates a path with lower cost while also requiring lesser time to generate it. The model is also highly robust and can generate a path in any pretrained arbitrary environments.

1. Introduction

Recently, autonomous devices for both home and industrial appliances have emerged due to the ever-increasing market demands. They automate the regular time-intensive and laborious tasks. Tiling robotics has quickly become an active area of research for tasks pertaining to area coverage, such as cleaning [1, 2], inspection [3], maintenance, surface painting [4] and surveillance monitoring in both indoor and outdoor spaces. There are numerous tiling robots in the market, but almost all of them are constrained by their fixed morphology design. Reconfigurable robots are able to cover a higher section of the area in any workspace compared to a fixed morphology robot. This is due to their ability to change shapes, which is extremely beneficial to tiling robots. This ability to morph into different shapes allows them to choose morphologies most suitable to the current coverage requirement dynamically. One such robot for cleaning tasks is the reconfigurable robot platform hTetro developed by Veerajagadheswar et al. [1]. The robot is able to morph into 7 different tetromino shapes using its four blocks, three hinge design. This gives the robot platform the capability to maneuver in difficult environments and around obstacles to reach hard to clean spaces.

The critical challenge of the reconfigurable tiling robot is to generate the optimal trajectory and set of shapes to cover the entire area. Typically, standard path planning algorithms are focused on trajectory planning for the robot to move from a starting point to a goal efficiently. However, when it comes to tiling tasks, completely covering the space is just as important. Hence, Complete Coverage Path Planning (CCPP) approaches are employed to plan the motion of tiling based robots. This involves planning a path which is energy efficient and completely covers the given area while avoiding any obstacles present. CCPP has been used in other fixed morphology platforms. Zeli Wang et al. solves the CCPP using a Finite State Machine (FSM) for a waste

* Corresponding author.
E-mail address: leanhvu@tdtu.edu.vn (A. Vu Le).

https://doi.org/10.1016/j.autcon.2020.103078
Received 15 May 2019; Received in revised form 2 January 2020; Accepted 6 January 2020

recycling robot [5].

All CCPP algorithms can be broadly grouped based on the decomposition strategies used to represent the environment [6]. A decomposition strategy involves breaking down the environment into smaller chunks, also referred to as sub-regions or cells. The classic method involves separating the space through the use of simple shapes like trapezoids, triangles [7]. The environment can also be separated using the morse function [8]. For sensor-based robot systems, the use of contact sensors is another strategy [8]. Some other approaches include the use of graphs [9], 3D data [10, 11] and landmarks [12]. Finally, the standard approach is through the use of grid-based decomposition as proposed by Moravec and Elfes [13] and Choset [14], which is used in this paper. Several algorithms can be used to section an environment using the grid strategy, which includes the energy aware algorithm [15], neural networks [16] and spanning trees [17]. The use of grid-based decomposition heavily reduces the complexity required in terms of computation to determine a coverage path. Most CCPP algorithms designed for the grid decomposition strategy have been for fixed-morphology robots.

The traditional modeling of CCPP for the grid-based representation involves two steps. First, a tileset that represents the shapes required to fill the space is generated using the polyomino tiling theory [18] with several lemmas. The hTetro can then move to each tile location of the chosen tileset and transform to the corresponding shape. This approach can assure that the workspaces satisfied by the tiling lemmas can be tiled by hTetro completely. However, the suggested tileset by tiling theory does not consider the optimal tileset with the least number of shape-shifts. Next, a path is generated by connecting and ordering the tileset generated. This can be done using traditional path planning algorithms such as zigzag, spiral, greedy search. As a consequence of the generated tileset, there are redundant actions generated, which cause the hTetro to transform and rotate to the desired shape within simple free spaces. However, most of these spaces can be covered more efficiently with the use of a single shape such as a square or bar. Moreover, the performance of these pathing algorithms is highly dependant on the environment and its structure.

A better approach is to model this path planning problem using the generated tileset as a TSP. This computes the least cost path (most efficient) under the pretext of connecting all waypoints, which ensures maximum coverage. However, solving this TSP is an NP-hard problem that is not feasible in real-world systems [19]. Another approach is to use Evolutionary algorithms such as GA [20] and ACO [21] to solve this TSP in a reasonable time [22]. However, these solutions are based on the tiling theory, which is heavily constrained by the landmarks extracted from the environment and cannot be adapted to any arbitrary environment. Moreover, optimization of the TSP solution also requires multiple iterations of computation to identify a semi-optimal path and can be swayed easily by the local minima during optimization.

Reinforcement learning has been used under the pretext of path planning before. Changxi et al. [23] proposed the use of reinforcement learning for trajectory planning for autonomous vehicles. Kenzo et al. [24] used the DDPG reinforcement learning algorithm to plan the motion of bipedal robots in a soccer match. Farad et al. [25] generated a path for efficient exploration in unknown environments through Actor-Critic Reinforcement learning model. A model trained using Q-Learning to generate a path from point A to point B in a grid-based decomposition of the environment has been proposed in Aleksandr et al. [26], Amit et al. [27] and Soong et al. [28]. David et al. extends this approach for multi-robot systems [29]. Yuan et al. [30] used a GRU RNN network to plan a path from point A to point B while avoiding obstacles in a grid-based environment. Richard et al. [31] discussed the application of Q-Learning for industrial robot motion planning. However, all of these works are focused on the simplistic task of trajectory generation from point A to point B and do not discuss the CCPP scenario.

This paper proposes a deep learning model trained using reinforcement learning for CCPP for the hTetro platform which is able to determine an efficient trajectory in real-time with a blackmuch lesser distance travel costweight compared to the previously discussed tiling approaches. The model also generates a more efficient path by minimizing the number of shape changes while maximizing the use of a single shape. The model is also highly versatile by being adaptable to unknown environments. The remainder of this paper is sectioned as follows. Section 2 describes the hTetro platform and the cost function used. These costs are proportional to the energy consumed and the time required by the robot for each action. Optimizing this cost leads to lower energy use, which can allow for longer operations of the robot platform. Section 3 discusses the state representation, the network architecture, training algorithm and the reward function used. Section 4 discusses and compares the performance of the algorithm in different environments.

2. hTetro reconfigurable robot platform

The hTetro platform is represented by four blocks named A, B, C and D. The three hinges which connect these four blocks provide 180-degree freedom of movement. Through this architecture, the platform can morph into seven different tetromino based morphologies - O, Z, L, T, J, S, I. Fig. 1 shows the seven different morphologies and the design of the hTetro platform. An Audrino microcontroller is used to control the hinges and wheels at each block. For running deep learning models, the Nvidia Jetson Nano is used along with the Audrino board to process the pathing algorithm. The robot uses differential drive motors for performing three different types of movements - translation, rotation, and transformation. These actions are able to be performed by issuing different velocity commands to the motor. The robot employs the use of magnets and limit-switches to ensure that transformations are completed correctly, and the robot is able to move without disrupting the current shape. Details of the mechanisms of the platform are discussed in [1].

Each action consumes a different amount of energy. For generating the path, action space is discretized, which means that a single action cannot be a combination of multiple types of movement. The cost of an action is characterized by the overall distance the robot moves for performing that particular action. The cost function for translation is determined by calculating the Euclidean distance between the start location (x_j,y_j) and the end location (x_i,y_i) of each block, which is described in Eq. (1). Rotation and transformation assume that the position

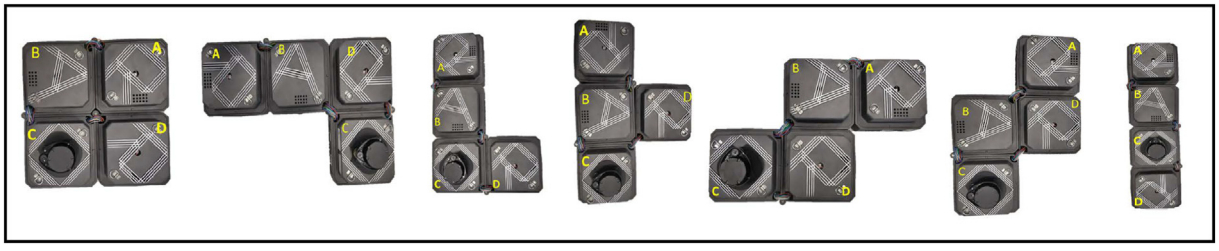

Fig. 1. Different morphologies of hTetro platform. From left to right: O, J, L, T, S, Z, I.

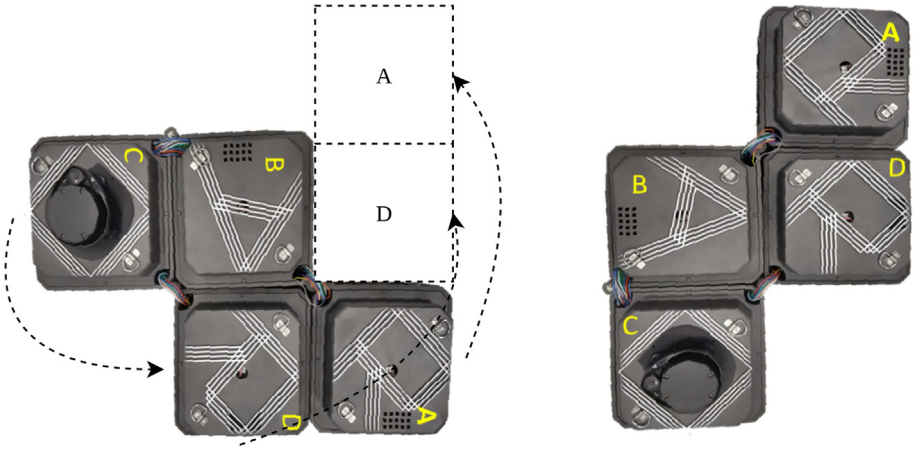

Fig. 2. hTetro rotation of Z shape.

of the center of block B of the robot is fixed. Under this assumption, the cost function for a rotation of angle π/2 (90°) is given by Eq. (2). This equation represents the length of the arc in which each block has to travel to achieve this change. An example of this is shown in Fig. 2. The blocks move according to the dotted line. The overall cost as per the given equation would be $\frac{\pi}{2}(l + l + l\sqrt{2})$, where l is the length of a block.

$$C_{translation} = \sum_{i,j \epsilon A,B,C,D} \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2} \tag{1}$$

$$C_{rotation} = \frac{pi}{2} \times \sum_{i \epsilon A,C,D} \sqrt{(x_i - x_B)^2 + (y_i - y_B)^2} \tag{2}$$

$$C_{total}(a_t) = \begin{cases} C_{transformation} & if\ a_t \in \{O, Z, L, T, J, S, I\} \\ C_{rotation} & if\ a_t \in \{clockwise, anticlockwise\} \\ C_{translation} & if\ a_t \in \{up, down, left, right\} \end{cases} \tag{3}$$

Transformation costs are dependent on the current shape and the desired shape. Table 1 shows the equations used to compute the transformation cost $C_{transformation}$. Generally, a set of block rotations are required to complete a transformation. Since these rotations are done around a hinge, the cost is in terms of L, which is $l/2$, where l is the length of the block. An example of a transformation is shown in Fig. 3. For changing the shape from O to L, the robot must first change from O to I and then I to L. The cost for transitioning from O to I is $\pi(\sqrt{5} + \sqrt{2})L$, since both C and D blocks have to move an angle of π, and they are at a distance of $L\sqrt{2}$ and $L\sqrt{5}$ respectively. The cost for changing from I to L is $\frac{pi}{2}(\sqrt{2})L$, since only D block moves this time an angle of $\frac{\pi}{2}$ at a distance of $L\sqrt{2}$ from the hinge. Hence, the overall cost turns out to be $\pi(\sqrt{5} + \frac{3}{2}\sqrt{2})L$. The overall cost of any action is given in Eq. (3).

Table 1
Transformation cost $C_{transformation}$ values. Columns headings represent the current shape. Row headings are the goal state.

	O Shape	I Shape	L Shape	J Shape
O Shape	-	$\pi(\sqrt{5} + \sqrt{2})L$	$\pi(\sqrt{5} + \frac{3}{2}\sqrt{2})L$	$\frac{\pi}{2}(\sqrt{5} + 5\sqrt{2})L$
I Shape	$\pi(\sqrt{5} + \sqrt{2})L$	-	$\pi(\sqrt{2})L$	$\frac{\pi}{2}(\sqrt{5} + 5\sqrt{2})L$
L Shape	$\pi(\sqrt{5} + \frac{3}{2}\sqrt{2})L$	$\pi(\sqrt{2})L$	-	$\frac{\pi}{2}(\sqrt{5} + 7\sqrt{2})L$
J Shape	$\frac{\pi}{2}(\sqrt{5} + 5\sqrt{2})L$	$\frac{\pi}{2}(\sqrt{5} + 5\sqrt{2})L$	$\frac{\pi}{2}(\sqrt{5} + 7\sqrt{2})L$	-
S Shape	$\frac{\pi}{2}(\sqrt{5} + 3\sqrt{2})L$	$\frac{\pi}{2}(\sqrt{5} + 3\sqrt{2})L$	$\frac{\pi}{2}(\sqrt{5} + 5\sqrt{2})L$	$\pi(\sqrt{2})L$
Z Shape	$\pi(\sqrt{2})L$	$\pi(\sqrt{5} + 2\sqrt{2})L$	$\pi(\sqrt{5} + \frac{5}{2}\sqrt{2})L$	$\frac{\pi}{2}(\sqrt{5} + 5\sqrt{2})L$
T Shape	$\frac{\pi}{2}(3\sqrt{2})L$	$\pi(\sqrt{5} + \frac{5}{2}\sqrt{2})L$	$\pi(\sqrt{5} + 3\sqrt{2})L$	$\frac{\pi}{2}(\sqrt{5} + 4\sqrt{2})L$

	S Shape	Z Shape	T Shape
O Shape	$\frac{\pi}{2}(\sqrt{5} + 3\sqrt{2})L$	$\pi(\sqrt{2})L$	$\frac{\pi}{2}(3\sqrt{2})L$
I Shape	$\frac{\pi}{2}(\sqrt{5} + 3\sqrt{2})L$	$\pi(\sqrt{5} + 2\sqrt{2})L$	$\pi(\sqrt{5} + \frac{5}{2}\sqrt{2})L$
L Shape	$\frac{\pi}{2}(\sqrt{5} + 5\sqrt{2})L$	$\pi(\sqrt{5} + \frac{5}{2}\sqrt{2})L$	$\pi(\sqrt{5} + 3\sqrt{2})L$
J Shape	$\pi(\sqrt{2})L$	$\frac{\pi}{2}(\sqrt{5} + 5\sqrt{2})L$	$\frac{\pi}{2}(\sqrt{5} + 4\sqrt{2})L$
S Shape	-	$\frac{\pi}{2}(\sqrt{5} + 5\sqrt{2})L$	$\frac{\pi}{2}(\sqrt{5} + 6\sqrt{2})L$
Z Shape	$\frac{\pi}{2}(\sqrt{5} + 5\sqrt{2})L$	-	$\frac{\pi}{2}(\sqrt{2})L$
T Shape	$\frac{\pi}{2}(\sqrt{5} + 6\sqrt{2})L$	$\frac{\pi}{2}(\sqrt{2})L$	-

Real-world environments are captured using the 2D LiDAR present on the robot. These real-world maps are converted into discrete grid maps of fixed size (11 × 11). Each grid cell has a size corresponding to one block of the hTetro platform in the real world. Larger maps are split into multiple 11 × 11 maps.

3. Proposed methodology

The problem of path planning for the reconfigurable robot platform hTetro can be represented in the form of a Markov Decision Process (MDP). The parameters of the MDP include (S,A,$P_a(s,s')$,$R_a(s,s')$). Here, S represents state information which pertains to shape, cleaning status, environment map. The set of actions represented by A involves four movements (up, down, left, right), two rotations (clockwise, anti-clockwise), and seven transformations (described in Section 2). The reward function $R_a(s,s')$ is described later in this section. The goal is to estimate a probability function $P_a(s,s')$ that maps the state s to an action a which leads to the state s'. This function can be computed through brute force, but given the infinite arrangements and configurations possible for different environments, this is not feasible. This paper uses a CNN LSTM model to approximate the probability function which is responsible for determining the movement strategy at any state.

For training the RL model, the state maps are converted into images of 88 × 88. The image represents the environmental information which is required by the network to make a decision. There are distinct representations of different features present in the environment. White squares represent grid cells that have objects or obstructions present in them. The combination of four blue squares represents the robot. Cleaned squares are blank whereas squares yet to be cleaned are represented by red diamonds. These distinct representations reduce the complexity of the model required to understand the current state. Fig. 4 shows a sample representation of the environment, which is used to train the network.

3.1. Decision network

The probability function to determine the action at any state is

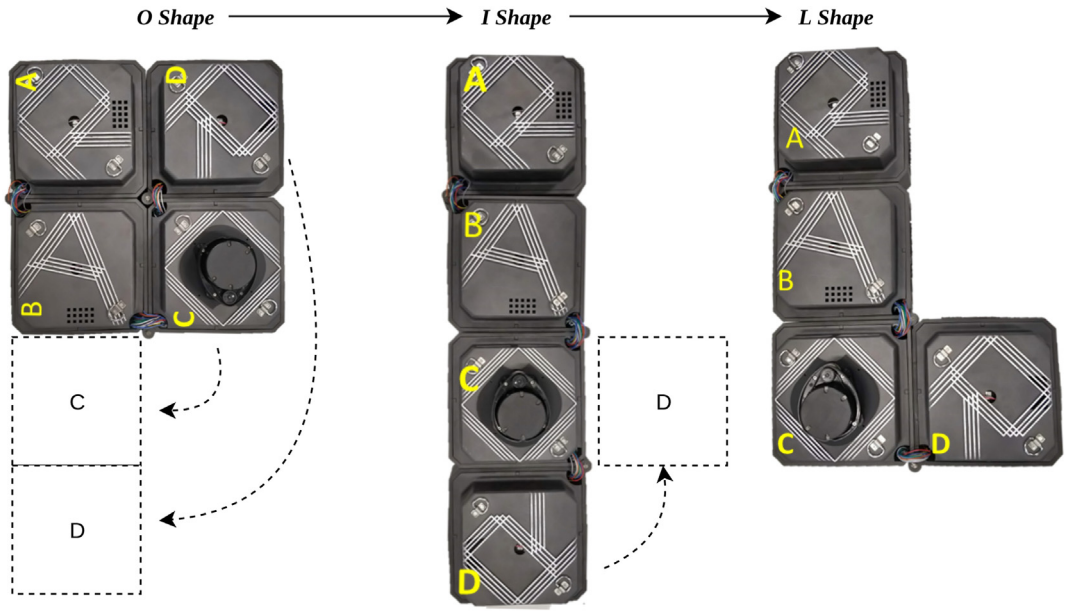

Fig. 3. hTetro shape transformation from O to L.

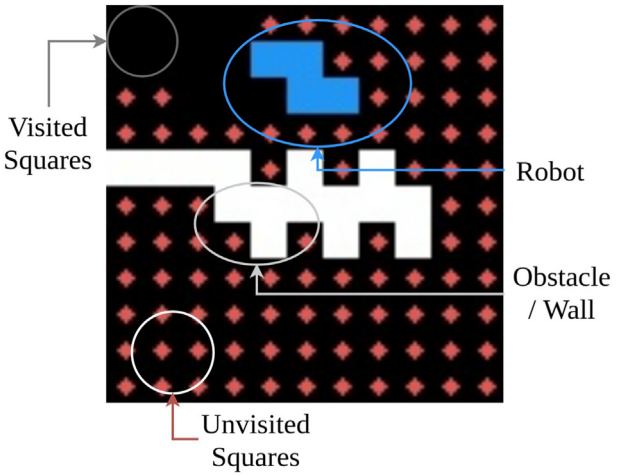

Fig. 4. State representation.

determined through a CNN LSTM network. The network architecture is shown in Fig. 5. The CNN portion has 3 convolution layers with different kernel sizes. The CNN extracts features from the environment state representation obtained above. The activation function used for these layers is the ELU activation function. ELU activation function is given by Eq. (4).

$$f(x) = \begin{cases} \alpha(e^x - 1) & for\ x \le 0 \\ x & for\ x > 0 \end{cases} \tag{4}$$

These features are then passed through a fully connected layer with ELU activation and then a LSTM with sigmoid activation. The LSTM helps learn sequences of actions that may be useful for covering a certain object pattern as well as avoid covering already covered areas. The sigmoid activation function is given by Eq. (5).

$$f(x) = \frac{1}{1 + e^{-x}} \tag{5}$$

3.2. Training the decision network

Training the decision network is done through the use of reinforcement learning (RL). Different RL algorithms exist, but only some of them are feasible for this scenario. blackModel-based RL techniques, such as Monte Carlo Tree Search (MCTS), are not feasible due to a large number of combinations of state and action pairs. Hence, model-free approaches are used. Q Learning [32] is not used here due to a large number of actions possible at each state. Policy gradient techniques estimate a policy which is able to predict the most rewarding action at any state. However, due to the nature of the complete coverage problem, it is not enough if the policy only takes into account the next action. Doing so would result in prioritizing avoidance of action with not so large reward, which will result in not achieving complete coverage. Hence, a policy gradient approach that takes into account future rewards is used, which enables the model to achieve efficient complete

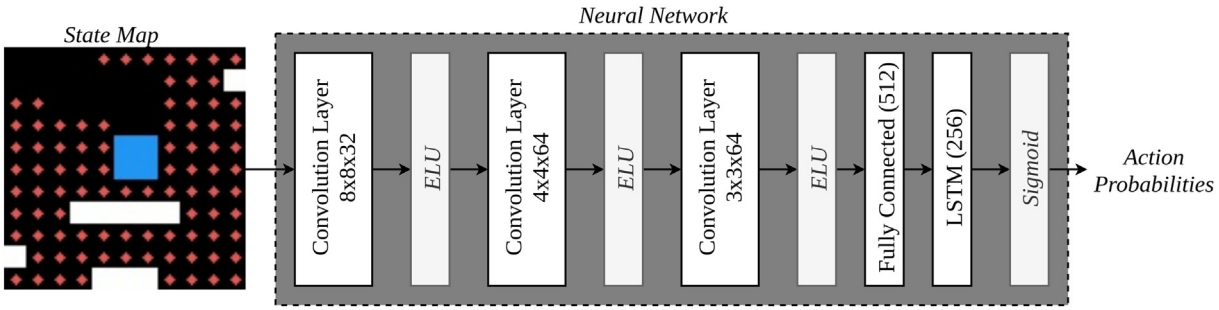

Fig. 5. Decision network model.

coverage path planning.

blackActor-critic [33] models are state-of-the-art techniques in policy gradient-based reinforcement learning. Actor-critic models comprise two sections - the actor which determines policy and a critic that evaluates the decision of the policy. The actor is trained based on the feedback from the critic, which evaluates decisions based on the reward function. The popular actor-critic model, A3C [34], is an on policy method, *i.e.* the algorithm completes a full run before evaluating their performance. Off-policy methods often have advantages over on-policy methods. This includes higher sample efficiency and better exploration. These two factors are crucial in the complex decision process for the hTetro environment, which inspires the use of the off-policy implementation of the actor-critic model, called Actor-Critic with Experience Replay (ACER) [35].

3.2.1. ACER reinforcement learning algorithm

ACER [35] is an off-policy implementation of A3C. It uses an experience replay buffer to record states, actions and rewards during training. Select experiences are chosen from this buffer and used for training the policy. ACER uses marginal value functions to approximate the policy gradient at each step. The marginal value policy gradient for any blackpolicy π with blackpolicy function θ is given by g_{mar} with state s_t and action a_t at a time t is computed as given in Eq. (6).

$$g_{mar} = \mathbb{E}_{s_t,a_t}[\rho_t \times \nabla_\theta \times log(\pi_\theta(a_t|s_t)) \times Q^\pi(s_t, a_t)] \tag{6}$$

where ρ_t is the importance weight, $\mathbb{E}$ denotes the expectation. To increase stability during the training process, ACER uses three optimizations to this function. First, ACER uses retrace q-value estimation to train the critic. Using retrace q-value instead of standard q-value function significantly reduces the bias during off-policy updates. The retrace estimator Q_{ret} is computed as shown in Eq. (7).

$$Q_{ret}(s_t, a_t) = r_t + \gamma \times \rho_{t+1} \times [Q_{ret}(s_{t+1}, a_{t+1}) - Q(s_{t+1}, a_{t+1})] + \gamma \times \mathbb{E}(s_{t+1}) \tag{7}$$

where r_t is the reward obtained given state s_t and action a_t, γ is the discount factor, Q is the standard q-value estimate and $\mathbb{E}$ is the expected Q value. The other two optimizations are aimed at reducing the variance during training. The former involves introducing a bias correction to the truncated importance weight ρ, which reduces the effect large ρ values can have on the final marginal value function. This term is added to Eq. (6). The latter is utilizing trust region policy optimization during training. Trust Region Policy Optimization (TRPO) [36] computes loss using blackKullback–Leibler (KL) divergence which is to be optimized by the optimizer. ACER modifies the KL divergence formula to make it more efficient. It uses a running average of the KL divergence to reduce huge variance caused due to large deviations from the running average.

3.2.2. Loss optimization

The loss is optimized using a Root Mean Squared Propagation (RMS Prop) optimizer [37]. RMS prop computes the weight w_t^{rms} at any time t based on the gradient g_t and exponential average v_t and the weight current weight. Eqs. (8), (9) and(10) are used to update the weights in the network. Here, η is the initial learning rate, β and ϵ are hyperparameters.

$$v_t = \beta v_{t-1} + (1 - \beta) g_t^2 \tag{8}$$

$$\Delta w^{rms} = \frac{-\eta}{\sqrt{v_t + \epsilon}} \times g_t \tag{9}$$

$$w_{t+1}^{rms} = w_t^{rms} + \Delta w^{rms} \tag{10}$$

3.3. Reward function

Reward functions are crucial for optimal training of the model. For CCPP, a reward can be given whenever the robot completely covers the accessible area. However, such a sparse reward function can result in very hard to achieve the convergence status of the model. Hence, covering each new tile is also rewarded along with complete coverage. The reward for completion is much larger compared to the covering of a single tile. Each action also has an associated cost described in Section 2. This cost is removed from the reward obtained at each step. If no new tile is covered, the overall reward is negative. This prevents the robot from backtracking or getting stuck. However, the reward of covering a tile is higher than the cost of movement, which allows the robot to make multiple actions to reach a new square. There is also a large negative reward for the suggestion of illegal action. blackIf the algorithm opts to perform an illegal action, no move will be performed. This can include collision with walls obstacles or suggestions of transformation rotations in locations with inadequate space. The overall blackreward R for a single action is given by

$$R(a_t) = n_{new} \times r_{new} + status \times r_{final} - C(a_t) - illegal \times c_{illegal} \tag{11}$$

where n_{new} is the number of new tiles covered, r_{new} is the reward for a new tile covered, *status* is a boolean that describes if the process is complete, r_{final} is the reward for completing the process, $C(a_t)$ is the cost of the given action a_t, $c_{illegal}$ is the penalty for any illegal action and *illegal* is a boolean that describes if an action is illegal. These reward values are fixed through experimentation.

4. Experimental results

4.1. Experimental setup

To evaluate path planning strategies between tiling-based methods and proposed trained model, three criteria are used in general - the efficiency, percentage of area covered and the time taken to generate the path. However, since this paper deals with CCPP, the underlying assumption is that complete coverage occurs every time. This is possible given the reconfigurable nature of the hTetro robot platform. This is also used as the criteria for termination of generation of any path. Hence, the applicable criteria for comparison is the efficiency or the cost associated with the generated path and the time taken to generate this path. The cost is computed as described by Eq. (3) in Section 2. The results are showcased as the path generated for various 11 × 11 environments along with their associated cost and execution time. The proposed approach is compared with tiling theory based algorithms in which the tetromino-based tileset is derived prior to the path using zigzag model, spiral model, greedy search and TSP models using optimization techniques like method [19], GA and ACO are used to solve the trajectory generation. One such tileset within the workspace 11 × 11 used for comparison is showed in Fig. 6b. To the best of our knowledge, excepting tiling theory-based method, the state-of-the-art CCPP methods are used to the fixed morphology platform, so further research is required to adapt them to hTetro.

The hyperparameters of the ACER algorithm and the reward function of the proposed CCPP are optimized through trials that check the convergence of the model for all environments. black120 random environments representing different real-world rooms are used to train the model. The learning rate for the RMS prop optimizer is set as 0.0001 with a β value of 0.99. The discount factor γ used for Q_{ret} estimation is set as 0.9. An experience buffer of size 20,000 is used when training on replays. In the reward function, r_{new} is set to around 20 times the maximum cost of a movement (400). The length of a block, which determines the grid size, is assumed as 2 when computing movement cost term $C(a_t)$. The penalty of an illegal action $c_{illegal}$ is set as 10 times the maximum cost of a movement (200). The final reward obtained for completely covering the workspace is 50 times the reward of a single square (20,000). All rewards are scaled down by a factor of 400. These values of hyper-parameters provide an optimal convergence of the reward function when trained with multiple environments. Changing these values by a large margin results in a lack of convergence of the

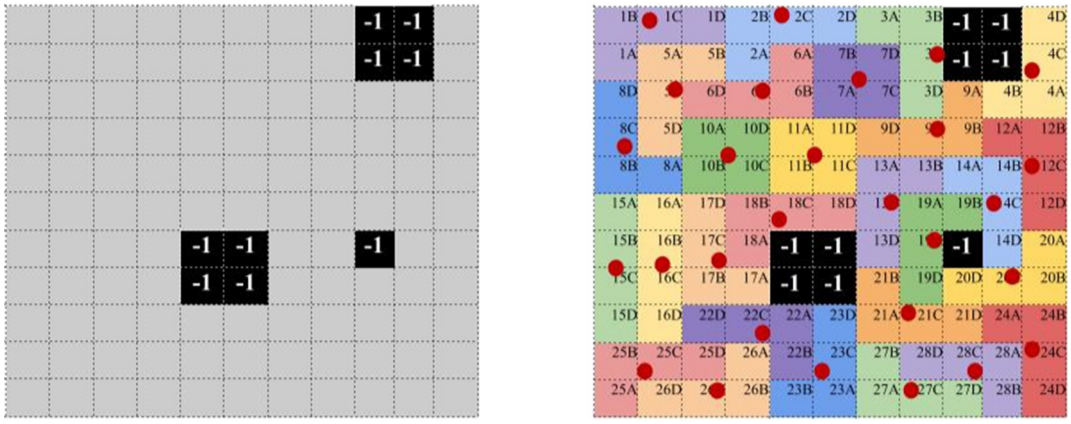

Fig. 6. Environment used for comparison. (a) Environment map used for performance measure. Red spaces with a − 1 value represents obstacles. (b) Tileset generated on the environment map by tiling based-theory method.

model during training. blackThe training time depends on the structure of the workspace. Normally it takes about black10,000 iterations to train 120 workspaces.

4.2. Experimental results simulation environment

The proposed CCPP trained model generates the same plans after three experimental trials for each workspace among 120 trained workspaces. The CCPP plans cover 100% of free spaces of tested workspaces without leaving any uncover grid cells. The averaged values of the cost associated with the generated path and the time taken to generate the path during trials are provided as 2636.59 and 0.251 *s*, respectively. The results proved the consistency in generating the CCPP plan and the feasibility for the real-time deployment of the trained model.

We selected randomly 10 workspaces among 120 workspaces which can tile completely by tiling theory based method to evaluate the optimality of the proposed CCPP framework statistically. Note that the tiling theory method is not able to generate the tetromino based tilesets for the workspace among 120 trained workspaces if the number of free spaces inside this workspace is not divisible by four. The averaged costweights and path generation time of all comparison methods are shown in Table 2. The proposed scheme with the trained model generates an optimal path with less associated costweight (2816.29 compare to 2947.59) and quicker (0.265 *s* compare to 1.159 *s*) than ACO with the second best costweight. Furthermore, this generation time is for the entire path, and decisions are made sequentially. Hence, only a fraction of this time would be required to start the movement process, and the remaining path can be generated while moving simultaneously. However, since this is a neural network model, there is a load time overhead of around 8 *s*, which can be done during the startup of the robot.

Table 2
Comparison of performance of the different CCPP methods on the simulated environments.

Method	Cost weight of path	Path generation time (*s*)
Zigzag [22]	4013.13	0.014
Spiral [22]	3924.12	0.161
Greedy Search [22]	3802.32	31.36
Method [19]	3693.41	1.192
GA [22]	2993.35	1.172
ACO [22]	2947.59	1.159
Proposed	*2816.29*	*0.265*

The comparisons between plans generated by the CCPP of tiling theory-based methods [22] and proposed CCPP method for an example workspace as Fig. 6a are given in Fig. 7 and Fig. 8, respectively. Note that the tiling theory based models first generate a tileset based on the given state. The tileset generated for this comparison workspace is shown in Fig. 6b. The sequence of actions is then optimized by the algorithm chosen. On the other hand, the proposed CCPP generates the robot shape and navigation sequence simultaneously. Specifically, the Zigzag model does a row-wise connection of tile pieces (Fig. 7a). The spiral model implements a circular or spiral search method, which prioritizes completing outer tiles before inner tiles (Fig. 7b). Although these two algorithms are computationally inexpensive, their efficiency is highly dependent on the environment configuration and is not suitable for real use. The greedy approach chooses the closest tile at each step and is executing the respective action to reach that tile (Fig. 7c). However, the closest distance may not necessarily mean the best action. This is because the total cost includes transformational and rotational costs as well, which can often add up to a large amount.

Modeling the problem as a TSP and employing method [19] (Fig. 7d), GA (Fig. 7e) or ACO (Fig. 7f) optimization models [22] reduce the cost by choosing an optimal combination of decisions which are most suitable for the given environmental configuration. These three models also take into account the transformation and rotation costs associated with each movement, thereby generating a more optimal path. GA and ACO evolutionary algorithms perform better than the method proposed in [19]. However, the performance of these techniques is capped by the performance of the tiling algorithm. Due to the nature of the tiling algorithm, a lot of different tiling pieces are stacked to fill the space. However, this can lead to a sub-optimal path to generate due to a large number of transformations required to fulfill the best path for the given tileset. In general, most transformations are as expensive or more expensive than a simple translation. Additionally, these paths can have moves that have a hidden cost, such as moves that require the robot to move around an obstacle.

Fig. 8 shows the path generated by the decision model trained using reinforcement learning for the comparison environment. Gray squares represent grid cells that have not been visited. The blue squares highlight the section, which requires a different shape to reach. White grid cells represent regions that have been visited. The path taken when the robot is in O shape is represented by the use of red arrows. Similarly, the path taken after the robot is in I shape is represented by green arrows. The dots represent the location of block B of the robot. For this environment, only one transformation occurs due to the requirement of a different shape to cover the difficult to reach squares. The model

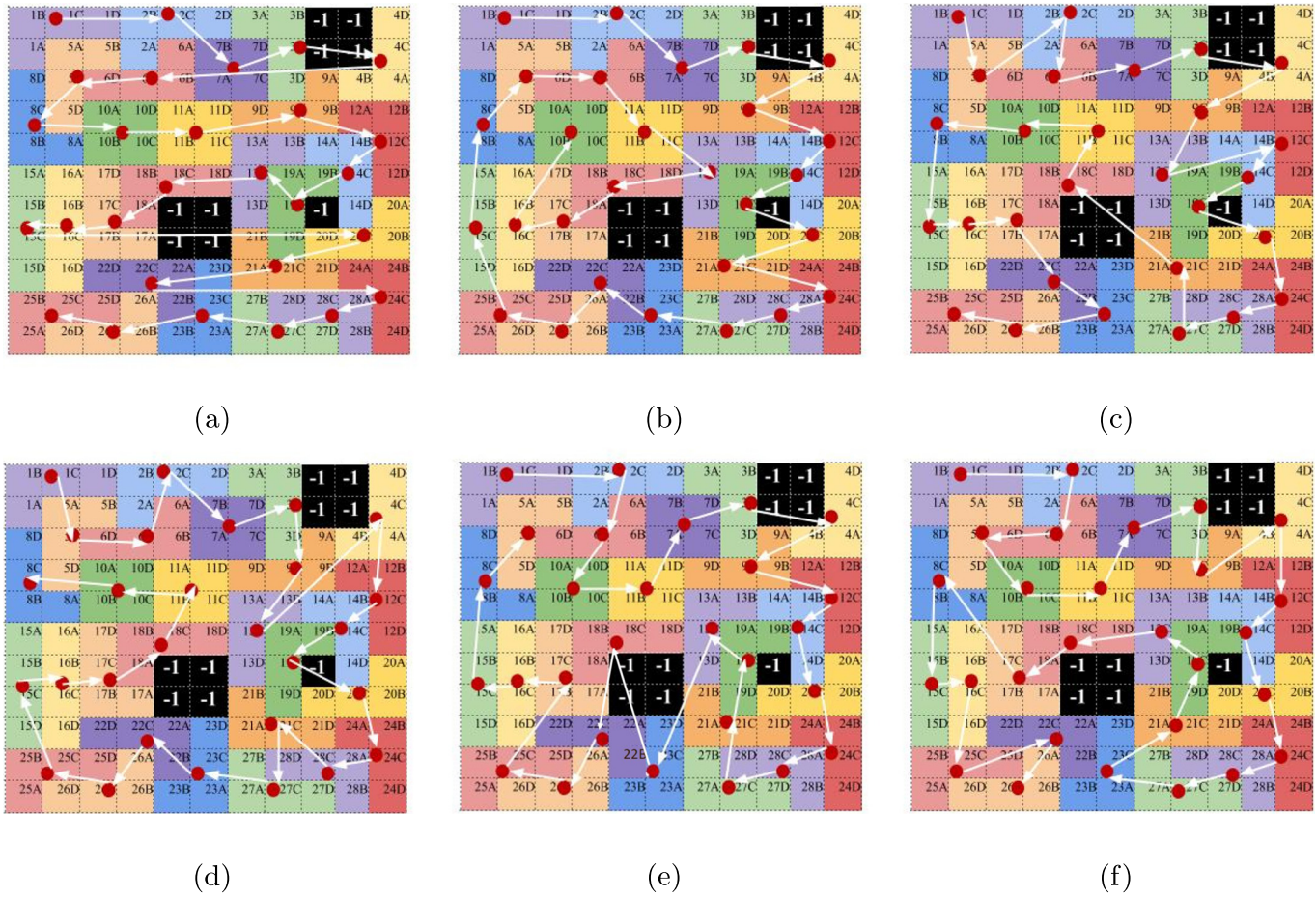

Fig. 7. Paths generated from algorithms based on the tiling theory. White arrows represent the sequence of actions the algorithm takes to completely cover the space. (a) Zigzag. Cost weight: 4072.19, (b) Spiral. Cost weight: 3821.18, (c) Greedy Search. 3791.92, (d) TSP based method [19]. Cost weight: 3688.21, (e) GA. Cost weight: 2968.95, (f) ACO. Cost weight: 2933.69.

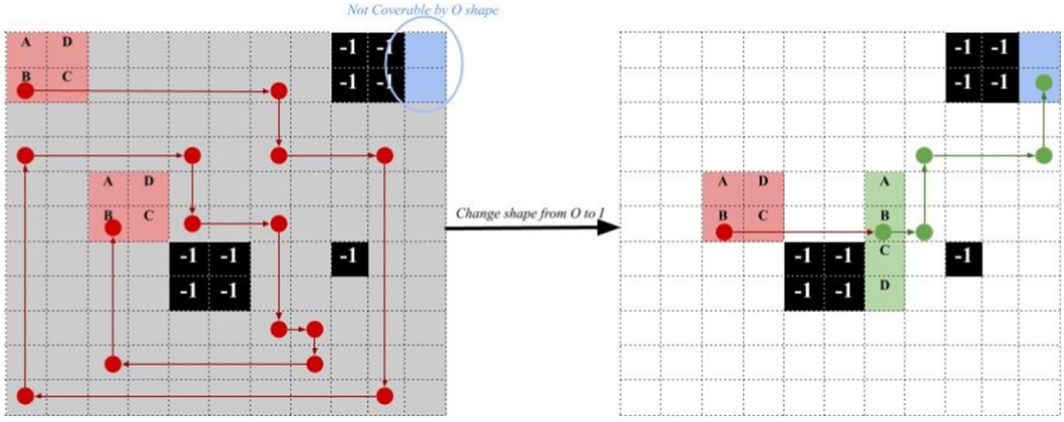

Fig. 8. Path generated by reinforcement learning in the comparison environment. Cost: 2817.31.

chooses to transform into an I shape because of it having the least transformation cost from the original O shape. On the other hand, changing to a J or L shape would have required two rotations to get into position, which would have made it a more costly maneuver. The model also tends to prioritize remaining in the original O shape.

The path generated by this model for two other environments is shown in Fig. 9. These environments cannot be tiled using the tiling theory since the number of free spaces is not divisible by four. Hence, tiling theory-based algorithms cannot be employed to generate a path. However, the proposed scheme is able to generate a path in these environments. Furthermore, the model minimizes the cost by reducing unnecessary transformation and rotation. Fig. 9a is a workspace that does not require any transformation to cover. The model generates a path with no transformation or rotation, which is the most cost-effective route. In Fig. 9b, a change of shape is necessary to cover the one block wide gaps present in the obstacle in the middle of the environment. The model suggests a change to the S shape in this case along with a rotation to align the robot with the gap in order to completely cover the workspace.

Another limitation of the tiling based approaches is that the optimal

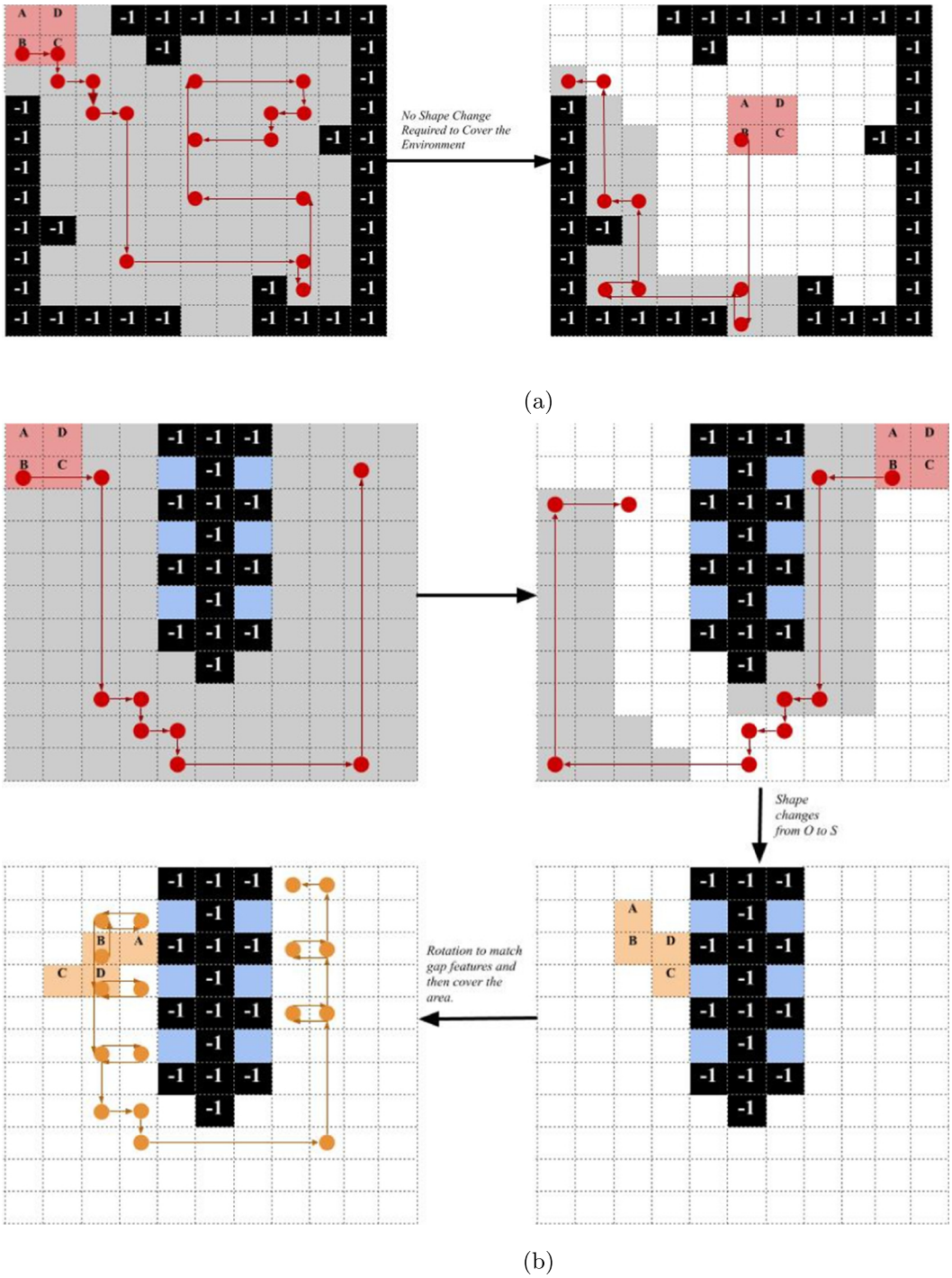

Fig. 9. Other examples of paths generated using reinforcement learning. (a) Cost: 2200, (b) Cost: 3624.45.

set of shapes for a given environment needs to be determined beforehand. This is not the case for the proposed approach, which is able to decide the optimal shapes required based on the obstacles, gaps, and spaces present in the environment. This is true for Figs. 8 and 9b where the model decides to choose shapes I and S based on the environmental structure, respectively. Even in Fig. 9a, the model decides to remain in O shape.

Visual outputs for sequences of actions in which the robot in blue color performs navigation and shapeshifting in five example simulated workspaces with various obstacle arrangements are provided in Fig. 10. Considering the simple workspaces without the need of shapeshifting as in Fig. 10a to the complex workspaces with the needs of shapeshifting as in Fig. 10b, c, e and and Maze-like environment Fig. 10d, and by following gradually the generated CCPP, one can observe that the robot can clear the free spaces (denoted as red dots) entirely. Besides, the number of shapeshifting is minimized, and the robot prefers to keep the same shape to clear the fee space area and only change to the shape with the minimal cost to clear the constraints, and narrow areas.

4.3. Experimental results in real robot

The costweights generated by all CCPP methods for simulation workspaces are validated in terms of energy consumption efficiency in the real testbed environment. To compare with the existing methods

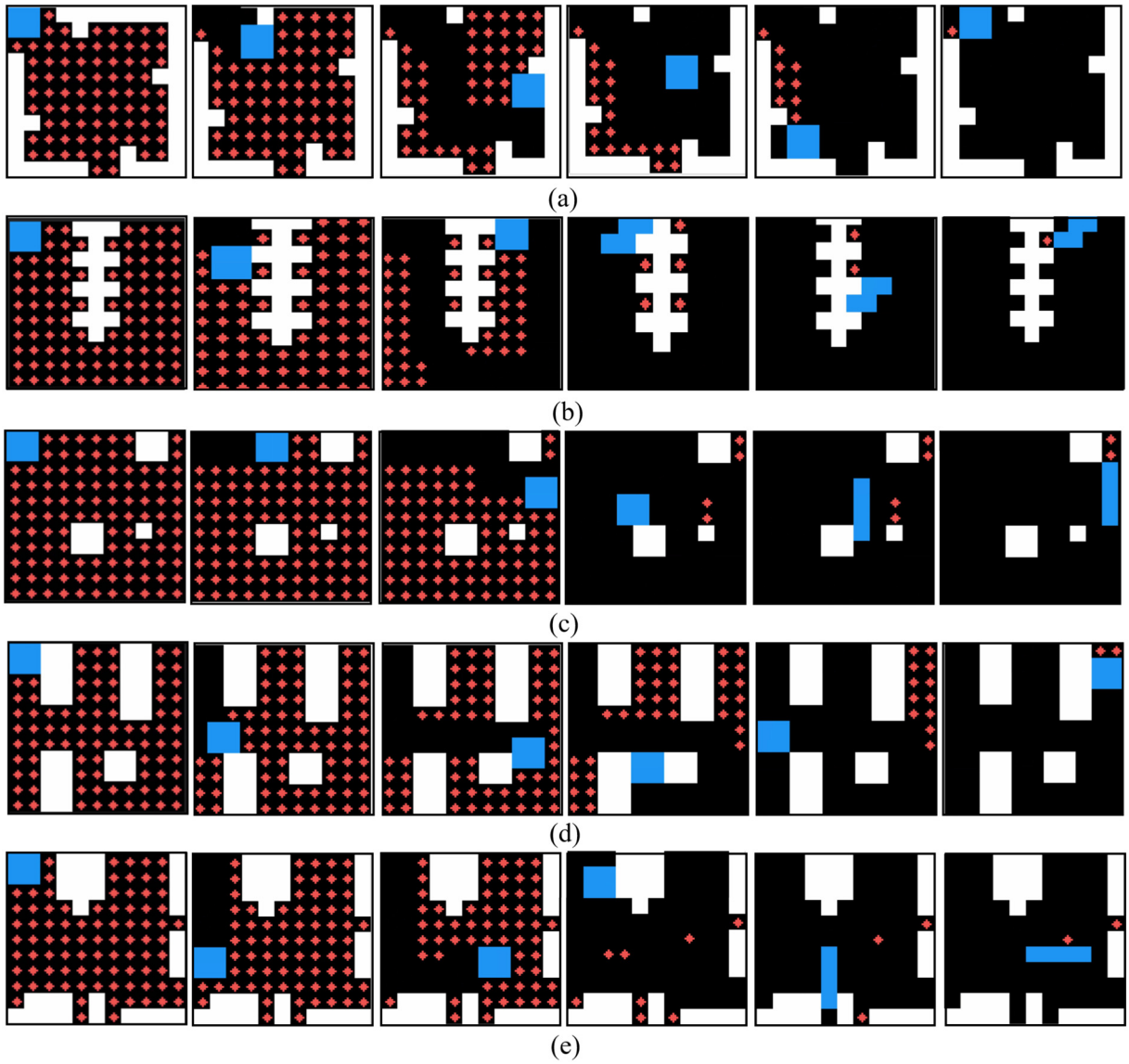

Fig. 10. Simulated navigation for different workspaces settings. (a) Only O shape, (b) shapeshift between O shape and S shape, (c) shapeshift between O shape and I shape, (d) able to backtrack in Maze-like environment, (e) shapeshift between O shape and I shape with optimal orientation.

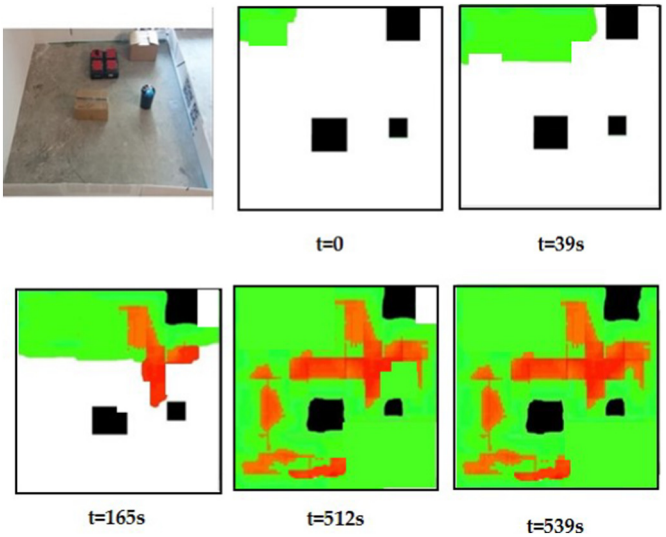

Fig. 11. Cover and re-covered areas by proposed CCPP.

using a tetromino based robot to tile the real tested environment, we use the workspace of Fig. 11 in which the obstacle layout is similar with the simulated workspace as in Fig. 7. The Hector SLAM [38] is used for building the map and localizing the robot within the map. After the map is built, the hardcode predefined waypoints indicated the locations, and the desired shapes are set then stored on the robot global navigation plan. Note that the plan was just duplicated from the simulation result as the map layouts are the same, but the exact waypoint locations are adjusted based on the prebuild map. After the waypoints are generated by all tested CCPP methods, the robot starts moving within the prebuild map, the real-time robot location is monitored, and the commands indicating the desired navigation direction and shape are issued autonomously to control the robot to clear all predefined waypoints. To calculate the power consumption of the robot during navigation, the value from the current sensor of the robot battery module was recorded. The sampling frequency of the measured current is 10 KHz at the 12 V voltage. The maximum motor speed is 150 rpm.

Coverage map, which includes area re-covered, is derived from the recorded videos by the camera mounted on the top view of the tested workspace. Fig. 11 presents the tracked map during hTetro navigation following the sequence of the proposed CCPP. In the image, the green

Table 3
Comparison of energy consumption and coverage time of the different models on the real environment.

Method	Cost	Coverage time (*s*)	Power (*W*)	Energy (*Ws*)
Zigzag [22]	4072.19	1143.79	1.78	2035.66
Spiral [22]	3821.18	1030.12	1.90	1959.06
Greedy Search [22]	3791.92	958.88	1.37	1316.49
Method [19]	3688.21	928.85	1.30	1209.14
GA [22]	2968.95	761.26	0.83	634.84
ACO [22]	2933.69	728.11	0.81	589.77
Proposed	2817.31	539.68	0.67	361.59

pixels depict the area covered by the hTetro, and the red parts present for the re-covered area. From Fig. 11, we can observe that the robot covers the map entirely with few percentages of re-covered areas.

Table 3 shows the energy consumption and averaged travel time after blackfive trials of all test methods with the real workspace as shown in Fig. 11. From the numerical values, we realize that the smaller estimated costweight is matched with the smaller energy value. Specifically, the simplest zigzag method creates the highest value of energy consumption, and close behind is the spiral. The proposed method yields both the smallest average grid coverage time and the least energy consumption. The proposed method gains about 82% lower than Zigzag the longest distance cost and about 38% lower than ACO the second-best method. These results show that the proposed CCPP with a pre-trained RL model is a possible energy-aware CCPP technique for polyomino tiling robot platforms in real environments.

5. Conclusion

This paper is the attempt to solve CCPP for the reconfigurable platform by a novel reinforcement learning-based framework. The robustness of the model is demonstrated through testing in different environments. The experimental results prove that the algorithm is able to minimize the transformational and rotational actions, thereby enabling it to reduce the cost over traditional tiling methods. The model is also able to determine the optimal set of morphologies required for each environment. In contrast to TSP based optimization schemes such as GA and ACO, the proposed method generates a path with lower cost while also taking lesser time to run. Moreover, the proposed CCPP can be extended to other polyomino-based shapeshifting robots. In future work, we will study the effectiveness of this model to generate actions in partially explored environments. We also plan to extend the proposed model to completely control the robot platform using continuous outputs that can drive motors rather than discrete action requests.

Acknowledgments

This work is financially supported by the National Robotics R&D Program Office, Singapore, under the grant no. RGAST1907, the Singapore University of Technology and Design (SUTD).

References

[1] V. Prabakaran, M.R. Elara, T. Pathmakumar, S. Nansai, Floor cleaning robot with reconfigurable mechanism, Autom. Constr. 91 (2018) 155–165, https://doi.org/10.1016/j.autcon.2018.03.015.
[2] M. Vega-Heredia, R.E. Mohan, T.Y. Wen, J. Siti'Aisyah, A. Vengadesh, S. Ghanta, S. Vinu, Design and modelling of a modular window cleaning robot, Autom. Constr. 103 (2019) 268–278, https://doi.org/10.1016/j.autcon.2019.01.025.
[3] M.D. Phung, C.H. Quach, T.H. Dinh, Q. Ha, Enhanced discrete particle swarm optimization path planning for UAV vision-based surface inspection, Autom. Constr. 81 (2017) 25–33, https://doi.org/10.1016/j.autcon.2017.04.013.
[4] S. Seriani, A. Cortellessa, S. Belfio, M. Sortino, G. Totis, P. Gallina, Automatic path-planning algorithm for realistic decorative robotic painting, Autom. Constr. 56 (2015) 67–75, https://doi.org/10.1016/j.autcon.2015.04.016.
[5] Z. Wang, H. Li, X. Zhang, Construction waste recycling robot for nails and screws: computer vision technology and neural network approach, Autom. Constr. 97 (2019) 220–228 doi.
[6] E. Galceran, M. Carreras, A survey on coverage path planning for robotics, Robot. Auton. Syst. 61 (12) (2013) 1258–1276, https://doi.org/10.1016/j.robot.2013.09.004.
[7] V. Lumelsky, S. Mukhopadhyay, K. Sun, Dynamic path planning in sensor-based terrain acquisition, IEEE Trans. Robot. Autom. 6 (4) (1990) 462–472, https://doi.org/10.1109/70.59357.
[8] E.U. Acar, H. Choset, A.A. Rizzi, P.N. Atkar, D. Hull, Morse decompositions for coverage tasks, Int. J. Robot. Res. 21 (4) (2002) 331–344, https://doi.org/10.1177/027836402320556359.
[9] K.P. Cheng, R.E. Mohan, N.H.K. Nhan, A.V. Le, Graph theory-based approach to accomplish complete coverage path planning tasks for reconfigurable robots, IEEE Access 7 (2019) 94642–94657, https://doi.org/10.1109/ACCESS.2019.29284675.
[10] T. Oksanen, A. Visala, Coverage path planning algorithms for agricultural field machines, J. Field Rob. 26 (8) (2009) 651–668, https://doi.org/10.1002/rob.20300.
[11] P. Cheng, J. Keller, V. Kumar, Time optimal UAV trajectory planning for 3D urban structure coverage, 2008 IEEE/RSJ International Conference on Intelligent Robots and Systems, IEEE, 2008, pp. 2750–2757, , https://doi.org/10.1109/IROS.2008.4650988.
[12] A. Le, V. Prabakaran, V. Sivanantham, R. Mohan, Modified a-star algorithm for efficient coverage path planning in tetris inspired self-reconfigurable robot with integrated laser sensor, Sensors 18 (8) (2018) 2585, https://doi.org/10.3390/s18082585.
[13] H. Moravec, A. Elfes, High resolution maps from wide angle sonar, 1985 IEEE International Conference on Robotics and Automation, vol. 2, IEEE, 1985, pp. 116–121, , https://doi.org/10.1109/ROBOT.1985.1087316.
[14] H. Choset, Coverage for robotics - a survey of recent results, Ann. Math. Artif. Intell. 31 (1) (2001) 113–126, https://doi.org/10.1023/A:1016639210559.
[15] A. Manimuthu, A.V. Le, R.E. Mohan, P. Veerajagadeshwar, N. Huu Khanh Nhan, K. Ping Cheng, Energy consumption estimation model for complete coverage of a Tetromino inspired reconfigurable surface tiling robot, Energies 12 (12) (2019) 2257, https://doi.org/10.3390/en12122257.
[16] S.X. Yang, C. Luo, A neural network approach to complete coverage path planning, IEEE Trans. Syst. Man Cybern. B Cybern. 34 (1) (2004) 718–724, https://doi.org/10.1109/TSMCB.2003.811769.
[17] Y. Gabriely, E. Rimon, Spiral-STC: an on-line coverage algorithm of grid environments by a mobile robot, 2002 IEEE International Conference on Robotics and Automation, vol. 1, IEEE, 2002, pp. 954–960, , https://doi.org/10.1109/ROBOT.2002.1013479.
[18] J.H. Conway, J.C. Lagarias, Tiling with polyominoes and combinatorial group theory, J. Comb. Theory, A 53 (2) (1990) 183–208, https://doi.org/10.1016/0097-3165(90)90057-4.
[19] A. Le, M. Arunmozhi, P. Veerajagadheswar, P.-C. Ku, T.H. Minh, V. Sivanantham, R. Mohan, Complete path planning for a tetris-inspired self-reconfigurable robot by the genetic algorithm of the traveling salesman problem, Electronics 7 (12) (2018) 344, https://doi.org/10.3390/electronics7120344.
[20] P. Larranaga, C.M.H. Kuijpers, R.H. Murga, I. Inza, S. Dizdarevic, Genetic algorithms for the travelling salesman problem: a review of representations and operators, Artif. Intell. Rev. 13 (2) (1999) 129–170, https://doi.org/10.1023/A:1006529012972.
[21] M. Dorigo, G. Di Caro, Ant colony optimization: a new meta-heuristic, Proceedings of The 1999 Congress on Evolutionary Computation-CEC99 (Cat. No. 99TH8406), vol. 2, IEEE, 1999, pp. 1470–1477, , https://doi.org/10.1109/CEC.1999.782657.
[22] A.V. Le, P.-C. Ku, T. Than Tun, N. Huu Khanh Nhan, Y. Shi, R.E. Mohan, Realization energy optimization of complete path planning in differential drive based self-reconfigurable floor cleaning robot, Energies 12 (6) (2019) 1136, https://doi.org/10.3390/en12061136.
[23] C. You, J. Lu, D. Filev, P. Tsiotras, Advanced planning for autonomous vehicles using reinforcement learning and deep inverse reinforcement learning, Robot. Auton. Syst. 114 (2019) 1–18, https://doi.org/10.1016/j.robot.2019.01.003.
[24] K. Lobos-Tsunekawa, F. Leiva, J. Ruiz-del Solar, Visual navigation for biped humanoid robots using deep reinforcement learning, IEEE Robot. Autom. Lett. 3 (4) (2018) 3247–3254, https://doi.org/10.1109/LRA.2018.2851148.
[25] F. Niroui, K. Zhang, Z. Kashino, G. Nejat, Deep reinforcement learning robot for search and rescue applications: exploration in unknown cluttered environments, IEEE Robot. Autom. Lett. 4 (2) (2019) 610–617, https://doi.org/10.1109/LRA.2019.2891991.
[26] A.I. Panov, K.S. Yakovlev, R. Suvorov, Grid path planning with deep reinforcement learning: preliminary results, Procedia Comput. Sci. 123 (2018) 347–353, https://doi.org/10.1016/j.procs.2018.01.0545.
[27] A. Konar, I.G. Chakraborty, S.J. Singh, L.C. Jain, A.K. Nagar, A deterministic improved Q-learning for path planning of a mobile robot, IEEE Trans. Syst. Man Cybern. Sys. 43 (5) (2013) 1141–1153, https://doi.org/10.1109/TSMCA.2012.2227719.
[28] E.S. Low, P. Ong, K.C. Cheah, Solving the optimal path planning of a mobile robot using improved Q-learning, Robot. Auton. Syst. 115 (2019) 143–161, https://doi.org/10.1016/j.robot.2019.02.013.
[29] D.L. Cruz, W. Yu, Path planning of multi-agent systems in unknown environment with neural kernel smoothing and reinforcement learning, Neurocomputing 233 (2017) 34–42, https://doi.org/10.1016/j.neucom.2016.08.108.
[30] J. Yuan, H. Wang, C. Lin, D. Liu, D. Yu, A novel GRU-RNN network model for dynamic path planning of mobile robot, IEEE Access 7 (2019) 15140–15151, https://doi.org/10.1109/ACCESS.2019.2894626.
[31] R. Meyes, H. Tercan, S. Roggendorf, T. Thiele, C. Büscher, M. Obdenbusch, C. Brecher, S. Jeschke, T. Meisen, Motion planning for industrial robots using

reinforcement learning, Procedia CIRP 63 (2017) 107–112, https://doi.org/10.1016/j.procir.2017.03.095.
[32] C.J. Watkins, P. Dayan, Q-learning, Mach. Learn. 8 (3-4) (1992) 279–292, https://doi.org/10.1023/A:1022676722315.
[33] J. Kober, J.A. Bagnell, J. Peters, Reinforcement learning in robotics: a survey, Int. J. Robot. Res. 32 (11) (2013) 1238–1274, https://doi.org/10.1177/0278364913495721.
[34] V. Mnih, A.P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, K. Kavukcuoglu, Asynchronous methods for deep reinforcement learning, Proceedings of the 33rd International Conference on International Conference on Machine Learning, 2016, pp. 1928–1937 https://dl.acm.org/citation.cfm?id=3045594 , Accessed date: 2 January 2020.
[35] Z. Wang, V. Bapst, N. Heess, V. Mnih, R. Munos, K. Kavukcuoglu, N de Freitas, Sample efficient actor-critic with experience replay (2016), arXiv preprint arXiv:1611.01224 (2016) http://arxiv.org/abs/1611.01224 , Accessed date: 2 January 2020.
[36] J. Schulman, S. Levine, P. Abbeel, M. Jordan, P. Moritz, Trust region policy optimization, Proceedings of the 32nd International Conference on International Conference on Machine Learning, 2015, pp. 1889–1897 https://dl.acm.org/citation.cfm?id=3045319.
[37] S. Ruder, An overview of gradient descent optimization algorithms (2016), arXiv preprint arXiv:609.04747 (2016) http://arxiv.org/abs/1609.04747 , Accessed date: 2 January 2020.
[38] S. Kohlbrecher, J. Meyer, T. Graber, K. Petersen, U. Klingauf, O. von Stryk, Hector open source modules for autonomous mapping and navigation with rescue robots, Robot Soccer World Cup, Springer, Berlin, Heidelberg, 2013, pp. 624–631, , https://doi.org/10.1007/978-3-662-44468-9_58.