Automatic Prefabricated Construction Simulation using Deep Reinforcement Learning

Aiyu Zhu, a.zhu@tue.nl
Eindhoven University of Technology, Eindhoven, Netherlands

Pieter Pauwels, p.pauwels@tue.nl
Eindhoven University of Technology, Eindhoven, Netherlands

Bauke de Vries, b.d.vries@tue.nl
Eindhoven University of Technology, Eindhoven, Netherlands

Abstract
The automation of prefabricated construction imposes higher demands on construction process planning. To enable the construction behavior to be reliably executed by construction robots, it is necessary to plan the construction process more precisely. Simultaneously, to ensure construction safety, the construction policy also needs to be updated in real-time according to the construction environment changes to guarantee an optimized and collision-free construction process. We propose a deep reinforcement learning-based near real-time construction process planning framework for dynamic construction environments. As a test, we create a simple construction simulated environment and design a single component simulation in a dynamic construction environment.

Keywords: Construction automation, Construction planning, Deep reinforcement learning

1 Introduction
As prefabricated buildings are increasingly being used in various building types, such as factories, housing, office, etc (Generalova, Generalov, & Kuznetsova, 2016), the construction method has also changed from traditional cast-in-place construction to prefabricated construction, where the components are produced in a factory and assembled on site. Due to prefabricated construction's modular and industrial features, robot-based construction has shown potential for the AEC industry (Bock, 2015).

In contrast to the factory environment, the on-site construction environment is complex and variable. Therefore, one of the essential challenges in realizing automated robotic construction is to allow the robots to independently adapt their construction tasks to the changing construction environment (Davila Delgado et al., 2019). As the traditional construction is mostly executed manually, on-site management is frequently done in terms of days or floors, with construction operations planned at a macro level, while the project manager adjusts specific construction tasks according to the specific construction circumstances (Jeong, Chang, Son, & Yi, 2016). Consequently, on-site construction management relies on more coarse of (digital) control than manufacturing assembly lines, which have much more fine-grained and automated construction actions (e.g., transport paths and assembly methods for each component). Furthermore, the precision required for robotic work is poorly satisfied by conventional on-site management. There is a demand for a method that can plan construction processes in response to site changes, thus providing construction robots with well-defined construction tasks.

Currently, the most relevant research for on-site construction action is crane-related lift planning (Lin, Wu, Wang, Wang, & Gao, 2014). As one of the most important pieces of equipment in on-site construction, when the crane can be unmanned, it can be considered as a construction robot. Therefore, lift planning and optimization can be regarded as a problem of construction planning for construction robots. The current leading research for on-site lift planning (Zhang & Pan, 2020) is based on optimizing site layout, collision analysis in path planning and path planning optimization, etc.

However, the construction process often changes according to multiple factors (e.g., site layout changes, site environment changes). These changes often occur simultaneously, and it is difficult to achieve automation, because that requires a reasonable combination in real time according to the changes in the site environment with different component layouts (Zhang & Pan, 2020). So we need a method that can flexiable adjust the robot's construction strategies (paths, construction methods, etc.) in real time depending on the changes in the environment.

As a type of AI-based method, reinforcement learning (RL) algorithms allow the agent to explore and exploit the environment and, through the feedback (rewards) received from the exploration and exploitation, accumulate experience to gain an optimal action policy. Deep reinforcement learning (DRL), as an algorithm that combines RL and deep learning, has greatly improved reinforcement learning algorithm's performance by computing the reinforcement learning output through neural networks and has displayed surpassing human performance in dealing with specific complex problems (Mnih et al., 2013). Thus, utilizing the features of DRL, we can attempt to introduce the DRL into the dynamic construction processes, and as a result make the robot better able to respond to its dynamic environment(s).

This paper implements deep reinforcement learning algorithms into construction process planning to achieve a framework for components' self-organization with different site layouts in a changing construction environment for prefabricated construction. The purpose of this paper is to address the problem of the automatic construction process planning (path planning and collision analysis currently) for automated construction equipment (construction robots, crane, and etc.) in a dynamic site environment. The path obtained by self-organization of the components can be used as a result to control the construction robot (e.g. crane). Meanwhile, we designed a simple 3D construction simulation environment to test the framework's near real-time adaptation in a changing environment for a single component.

2 Background

In this section, we briefly review the related work to lift planning and introduce the Q-learning algorithm which is the core algorithm in deep q networks (DQN) (Mnih et al., 2013).

2.1 Lift planning and optimization

Cranes, as one of the most important pieces of equipment on-site, achieves component transportation. So, most research of on-site construction planning for construction action is based on crane lift path planning.

In 2002, Soltani (Soltani, Tawfik, Goulermas, & Fernando, 2002) evaluated the feasibility of heuristic search algorithms in construction path planning. In the same year, based on the heuristic depth method, Reddy (Safouhi, Mouattamid, Hermann, & Hendi, 2011) proposed obstacle-free lift path planning. With the development of artificial intelligence techniques, genetic algorithm-based methods (Cai, Cai, Chandrasekaran, & Zheng, 2016; Dutta, Cai, Huang, & Zheng, 2020), simulated annealing (K. Wu, García de Soto, & Zhang, 2020) were applied in lift path planning. Heuristic-based algorithms can optimize the motion path by exploring the global environment, but when the layout is changed, the path needs to be re-planned every time.

Zhang (Zhang, Pan, & Zheng, 2019) proposed the construction process planning of tower cranes based on transfer learning in 2019. By creating a recognition model for the lifting process and training the neural networks, the corresponding lift planning can be derived according to the actual construction situation. However, the main limitation is that it requires a large amount of historical data for learning. Similarly, as a machine learning method, reinforcement learning has

been attempted to introduce component-based robot construction with a simple 2D environment (Zhu, Pauwels, & de Vries, 2020).

The main limitation of the above described commonly-used intelligent algorithms in the field of construction path planning is that when the layout of the yard or the construction environment changes, it is necessary to re-plan based on global information, and the previous planning information cannot be reused, which reduces the efficiency of planning.

2.2 Value-based method reinforcement learning – Q learning

Reinforcement learning is one of the machine learning methods, often expressed as Markov Decision Process (MDP) (Vrabie & Lewis, 2010), to make decisions partially under the control of a decision-maker in a partially random situation. As an extension of the MDP approach, reinforcement learning relies on reward and punishment values for the agent to learn and improve its actions in a defined environment. In reinforcement learning, the agent that needs to learn a policy is set in a given environment and it learns the policy by exploring and exploiting the environment through a certain number of episodes. At the beginning of each episode, the current environment is observed as a state, after which the agent performs an action based on the current environment that causes the agent to continue exploring or to fail. At the end of each episode, the agent updates the action set policy for the previous episodes based on the feedback from the reward, thus converging on the policy that makes the reward optimal.

As a value-based model-free reinforcement learning method, Q-learning optimizes the combination of actions that achieve the maximum reward by the method that updates the q-value of the action set for each episode (Bertsekas & Yu, 2012). The q-value is used to evaluate the value of each action under the current decision, where a larger q-value represents a higher value of the action. As an off-policy algorithm, Q-learning is used to explore the environment by adapting for the greedy algorithm. The agent is allowed to try actions without the maximum q-value while learning the policy; when the policy is updated, the agent must perform the action with the maximum q-value.

3 Methods

In prefabricated construction processes, we need the robot to accomplish a sequence work that involves transportation, positioning, and assembly of the components. To achieve automated construction, we need to provide the robot with explicit action instructions to execute. Thus, the construction requirements and path planning of the components can be regarded as action instructions for the robots. Here, we consider the automated robot construction as the self-organization of the components. To realize the self-organization of components in a dynamic environment, we have developed a framework based on the DRL algorithm to generate the construction process of the components. Furthermore, we designed one experimental environment to test the feasibility of the framework for a number of scenarios, which are documented in Section 4.

3.1 Simulated environment for construction processes

To realize the simulation and visualization of the construction process, we built a simplified 3D construction simulated environment using pygame and pyOpenGL (Figure 1), based on the 2D version in Zhu et al. (2020). The environment consists of a site, components and obstructions, allowing to customize the size of the site, the length of the obstructions and components, and the number of components. At the same time, we have established simple construction rules that the construction process must obey. The following two sections describe the setup of the simulated environment and the simplified construction rules.

3.1.1 Setup of the Simulation Environment

The created simulation environment is shown in Figure 1. The basic units of this environment are voxels with length (X), width (Y) and height (Z) of one unit. Voxels can be combined to create three types of construction objects: sites, obstructions, and components. We also need to set the boundary of the environment and the construction objects are given spatial coordinates in the environment within the set boundary.

For the construction site itself, we defined a number of voxels in a horizontal slab with height one. Their type properties are set to "ground" and shown in gray to represent the site. The type properties of the other non-construction objects voxels are set to "air" and shown in wireframe (white) boxes (optionally not displayed) to represent the free building space. To distinguish the construction area from the yard area, we set the construction area in the site to 'construction area', which is displayed in blue. The obstructions and components must be placed above the site.

We set the obstruction to black, which consists of a combination of multiple voxels. Obstructions are used to abstract all objects other than non-components that need to be avoided during component movement (e.g., equipment in the field, vehicles, etc.). Obstructions are dynamic, which means that they can appear at any of the initial and target positions of the non-objects within the boundaries of the site.

The component is a combination of multiple voxels that are connected in an unidirectional manner (linear sequence of voxels). So in the environment, the length of the component can be customized and the section size is always 1x1 at the moment. The attributes of the component are id, type, initial position, target position, and construction state. The id is used to identify the uniqueness of the component. The type is used to describe the type of the component (e.g. site, obstruction, etc.). The initial position is the position of the component in the yard and is displayed as an orange line box. The target position is the final position of the component in the structure and is displayed as a pink wireframe box. The construction states are 'unbuilt', 'in transit', 'arrived', and 'assembled'. In this environment, we simplify arrival and assembly as continuous actions; assembly is executed immediately after arrival. Thus, arrival and assembly are considered as one state. The 'unbuilt' state is represented by orange, 'in transit' by blue, 'arrived' by green, and 'assembled' by pink (see Figure 1(a-c).

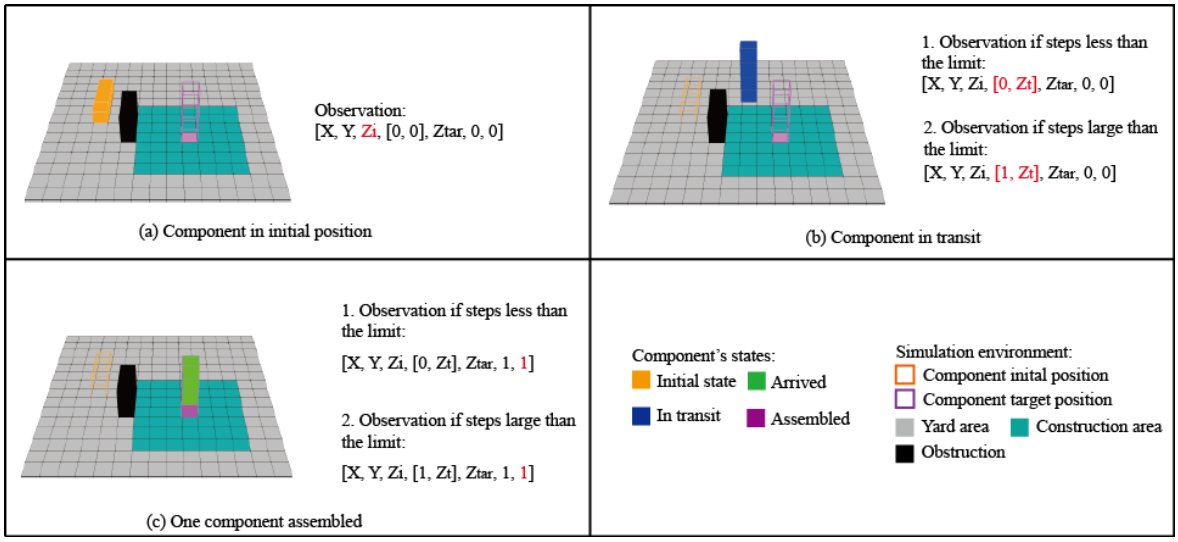


Figure 1. The simulated construction environment

3.1.2 Simplified Construction Rules for Prefabrication

To simulate the prefabricated construction, we defined the simulation environment as simplified construction rules in three aspects: construction safety, construction behavior and construction process. We propose three rules for construction safety: 1. construction objects can not collide with each other, 2. components cannot move out of the construction area, 3. the initial position of components must be set in the yard area.

During the construction process, we assign six actions to the component: up, down, left, right, front, and back. The component can execute only one of the six actions at one time, and each action moves the component by one base unit (one voxel in the environment) based on assigned action. Meanwhile, before the components enter 'in transit', to simulate the movements of the crane during construction, we will initialize the component positions. For example, the component will be erected for the column and moved up one basic unit in the Z direction.

Once each component is in 'assembled' state, the construction of that component is finished; when all components are in 'assembled' state, the construction is finished.

3.2 Link Simulated environment to DRL

In order to make the DRL algorithm embeddable in our simulation environment, we need to design the interface between prefabricated construction simulations and deep reinforcement learning. In this interface, the agent will learn the construction strategies by simulating the effective features in the environment. Therefore, we will design observations, actions, rewards and signals based on construction rules.

3.2.1 Observation Space

To transform the simulated environment information into valid features that can be used in DRL, we divide the original simulated environment information into two types: spatial information and object information and assign them to each basic unit (voxel). The spatial information includes the spatial coordinates (X, Y, Z) of each voxel in the environment. When the basic unit is 'air', the spatial coordinates are only (X, Y), and when construction objects appear, Z values are given to describe their height in space with states in object information. The object's information is a tuple with five elements, which contains the features of the construction object. When the base unit is of type 'air', there are no construction objects at that position, and the object's information is a tuple of zeros. When the base unit is defined as 'site' or 'obstruction', only the initial position of the five features is used to describe its height in space. When the base unit is of type 'component', it contains all five features in its tuple. The elements are as follows:

- Initial position: A scalar indicates the height (Zi) of the construction object in the environment. If the construction object is 'site' or 'obstruction', the position is fixed. If the construction object is 'component', the scalar includes the initial position for the 'unbuilt' state (Figure 1(a)).
- Position in transit: A binary tensor represents the height (Z_t) changes during the component in transportation. To avoid unnecessary over-exploration and to allow the policy to converge faster, we limited the actions. We set a feature to distinguish the number of actions of a component. The component must reach the target location in the environment with several steps less than X+Y+Z. Therefore, the maximum number of restricted steps of the component is X+Y+Z. When the number of steps of the component is less than this number, the tensor is [0, (Z_t)] (Figure 1(b)(1)). When the number of steps of the component is greater than this number, the component is [1, (Z_t)] (Figure 1(b)(2)).
- Target position: A scalar indicates the height (Z_{tar}) of the component's target position of height.
- Arrived index: A booelan scalar is used to determine if the component has reached the target position or not. The scalar has value zero if the component has not reached the target position, and one if the component has reached the target position.
- Assembled count index: This global index is used as a scalar that counts how many components are finalized and 'assembled' (at target position). Every time a component is 'assembled' (arrived index = 1), this count index of assembled components is incremented with value one and re-assigned to the scalar (Figure 1(c)).

3.2.2 Action Space

As in the simulation environment, the component can perform six actions. Here we consider these six actions as six discrete choices, using six sequential independent integers (0-5) in an array to define these six actions for the agent. Correspondingly, 0, 1, 2, 3, 4, 5 represents the six actions: forward, backward, left, right, up, and down. Each action moves a basic unit (voxel) in the direction ordered by the action.

3.2.3 Reward and Done signal Design

Generally, we use the dense reward policy (Z. Wu, Lian, Unhelkar, Tomizuka, & Schaal, 2020) to design our reward structure which means the agent will get a reward in each step. Only when the agent completes the task, it can get a positive reward; otherwise, any other action will get a

negative reward, and the agent's assignment is to find the action policy that can get the maximum reward.

To converge the policy, we count the components so that the later the component is built, the bigger the reward will get when 'assembled' and the lower the penalty in the movement. To avoid over-exploration by the agent, we limit the maximum number of steps (s) for the agent. When the agent's exploration exceeds the maximum number of steps, a 'done' signal is transmitted with value 'True', indicating that the episode is finished and the final policy is not learned (no final reward earned). When all components are assembled, the same 'done' signal is also set to True, indicating that the episode is in the end and the final policy is learned (the final reward is earned). The reward design (R) is provided in equation (1).

$$R_{(r0)} = \begin{cases} \sum_0^s \frac{r_0}{S_i} + \sum_{i=1}^n nS_i \,, s < limit \\ \sum_0^s \frac{2r_0}{S_i} + \sum_{i=1}^n nS_i \,, s < limit \end{cases} \quad (1)$$

Here, r_0 represents the negative reward for each step and $r_0 = -1$ in practice; s represents the steps of the current action of the component; S represents the number of components; S_i indicates the serial number of the current component being built (e.g., S_3 for the third component being built, S_3=3); n indicates the number of components, and the limit is the set limit number of steps.

3.3 Deep Q Network (DQN)-based framework in a dynamic construction

Based on the algorithm of DQN, we design a framework that can be used for near real-time construction simulation of dynamic construction environments.

3.3.1 Deep Q Network

DQN is a value-based deep reinforcement learning algorithm that combines deep neural networks and Q-learning. Q-learning algorithm is based on the markov decision process (MDP), which can select the action with a large q-value as the output according to the reward of the decision made by the agent to achieve the optimization of sequential actions. By combining with the deep neural network, instead of the traditional Q-learning method of querying the table to obtain the output of action, the q-value of each action is fitted by the neural network, which greatly improves the decision efficiency of Q-learning. The representation of the Q-value ($Q(s,a)$) is shown in equation (2):

$$Q(s,a) = Es^t \sim \varepsilon[r + \gamma maxQ(s^t, a^t)|s,a] \quad (2)$$

where s^t denotes the agent's state of the environment at time t; ε is the emulator which means that when the agent makes a decision, there is a small positive number ε of probability to select an unknown action randomly, leaving 1 - ε probability to select the action with the highest action value among the existing actions; γ is the learning rate which is used to control the rate of gradient descent; r is the reward obtained for that action, and $Q(s^t, a^t)$ is the value of q for all possible combinations of situations in the next step.

3.3.2 Structure of Framework

Our framework has two layers, as Figure 2 displays, the initialization layer and the application layer, both of which use the same construction environment. We first enable the agent to learn the construction policy in the initialization layer, and then implement the update of the construction policy in the dynamic environment with obstructions through the application layer.

In the initialization layer, the initial location of the components is randomly set at any yard location and no obstructions are arranged. This environment is used to train the components to plan the construction process in any layout automatically. When the initialization layer is completed, we save the trained neural network as the basic construction policy of the construction environment. In the learning process, we save the policy learned in the current episode every certain number of learning times, and test the success rate of the policy that can

accomplish the construction, run the current policy in the current environment a certain number of times, and end the learning when the success rate reaches the set requirement.

In the application layer, we assume that the site layout has been completed, so the initial position of the components is designed. In order to simulate the possible site changes in the construction environment, we add obstructions to the route of the components. When the construction policy is not affected by the environment change, the results of the trained neural network are maintained for simulation, and when the obstructions affect the construction policy (collision occurred), the simulation is stopped and the trained neural network is re-trained with the practice environment as an observation—training until the agent learns a construction policy that meets the requirements and saves the updated neural network as the new policy. When the trained neural network is run in the practice environment for the first time, we need a small trial (10 times in practice) without obstructions to confirm the best policy for the practice layout. So, we will record the rewards of each episode and use the policy of the episodes with the highest reward to the simulation.

4 Experiments

We use Deep Q Network (DQN) with OpenAI's stable-baselines library as our DRL policy and design one experimental environment to test the construction simulation of one single column under our framework.

4.1 Experimental Environment

In our experimental environment, the site environment is X=15, Y=15, Z=6, the construction area is the light blue area as displayed in Fig. 3, and the column's length is 4.

In the initial environment (initialization layer), the initial positions of the components in each episode are set randomly in an arbitrary non-construction area environment. As shown in Figure 3(a), the orange voxels represent the initial position of the column, and the pink wireframe voxels represent the target position of the column.

In the application environment, the initial positions of the components are set in a determined non-construction area environment and one or multiple obstructions of height 2 and length 3 will appear in the action path of the component. As shown in Figure 3(b), the column is set at a fixed initial position and the trained neural network is imported into the current environment to get the construction path planned by the agent. After that, the obstruction is set according to that

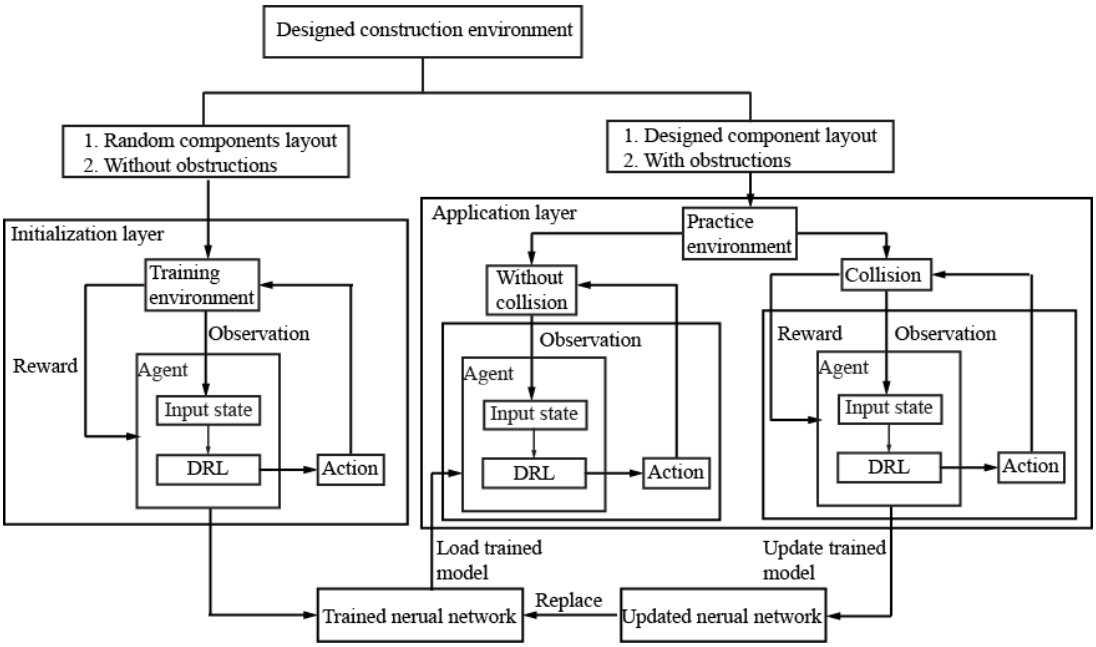


Figure 2. The framework of DQN-based construction process planning

path. Figure 3(b)(1) displays the unchanged construction environment, Figure 3(b)(2) shows the occurrence of one obstruction in the environment, Figure 3(b)(3) shows the occurrence of two horizontal obstructions, Figure 3(b)(4) shows the occurrence of one obstruction close to the target position. So, in short, 4 scenarios are tested within one and the same overall environment.

4.2 Setup DQN

For our experiments, we use the DQN policy and set the γ to 0.0005, the ε to 0.99. According to the experimental setting, we assign a step limit of 15 (site width) for the component, and the maximum number of steps for the agent in experiment 1 is 36. When the agent learning success rate reaches 100% in the initialization layer, we output this policy as our trained neural network. In the application layer, we select the episode with the largest reward in the 80% success rate (based on this policy, eight times successful construction for every ten tests) sample as the final construction policy. We test the success rate of the components every 2000 steps to determine whether the construction policy achieves an 80% success rate.

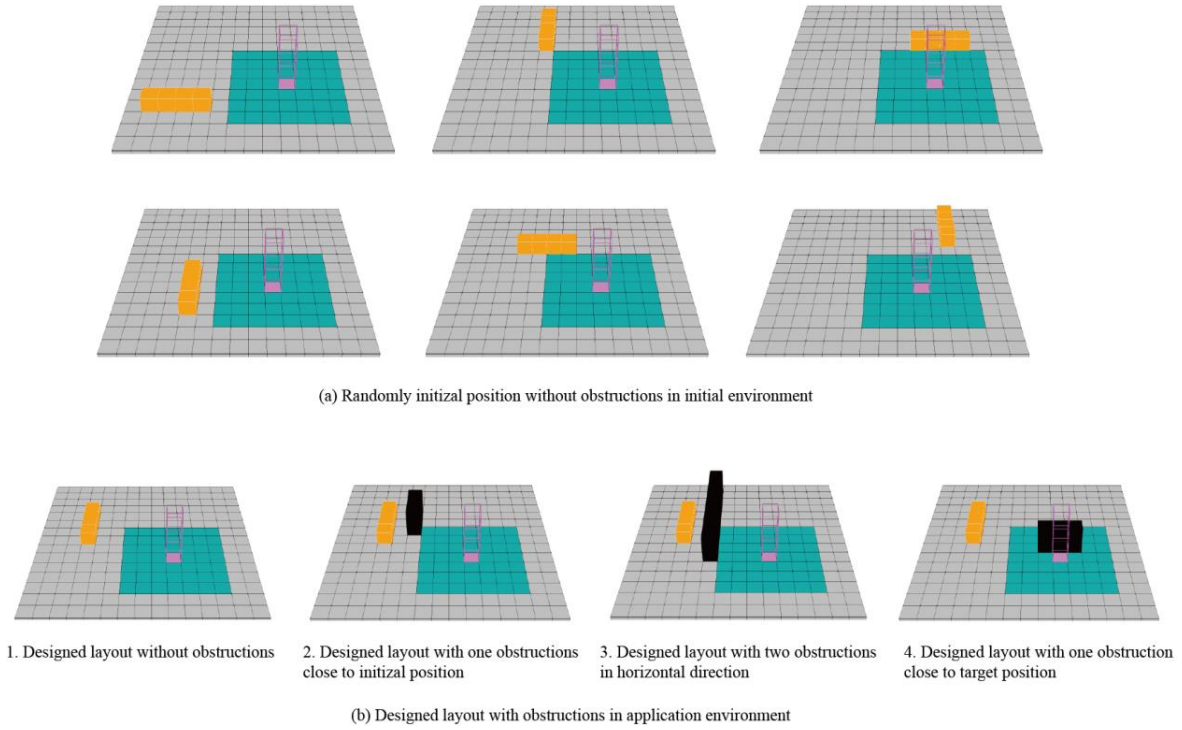


Figure 3. The experimental environment for a single column

4.3 Findings

Figure 4 shows the different construction policies given after the DQN algorithm update in the changing environment. In deep reinforcement learning, the learning of policy by the agent is usually monitored by recording the rewards and success rate during the learning process. Here, we use the success rate to validate: 1. whether the agent can learn and update the construction

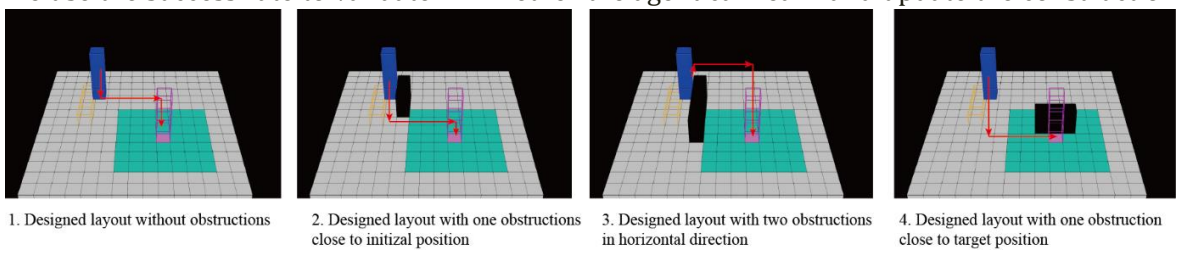


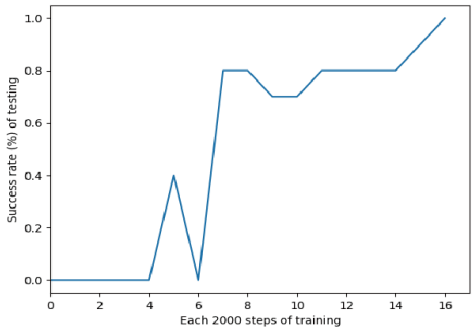
Figure 4. Different construction policies in changing construction environments

policy, 2. compare the required steps for policy updates in different obstruction scenarios.

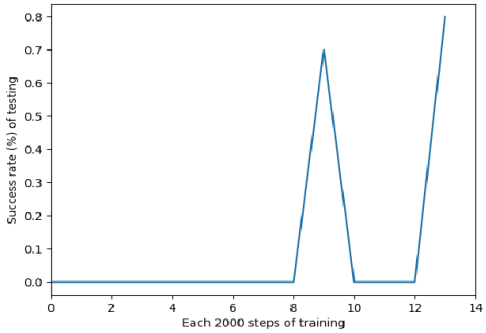
4.4 Results

The results of the success rate of the agent by testing every 2000 steps are shown in Figure 5. According to the success rate in Figure 5, we have the following results:

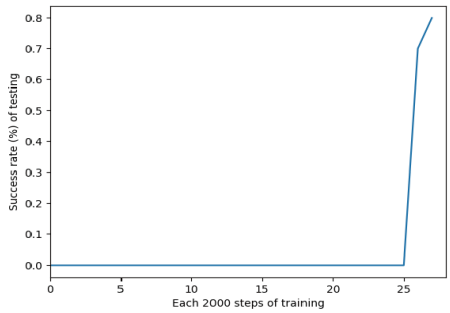
1. We find that based on the DQN method, the agent learned an appropriate construction policy in all scenarios,
2. By comparing Figure 5(b) and Figure 5(d), we observe that the agent can learn the construction policy faster when the obstruction is closer to the target position,
3. Through Figure 5(c), we notice that the learning steps of the agent become longer when the number of obstructions increases.



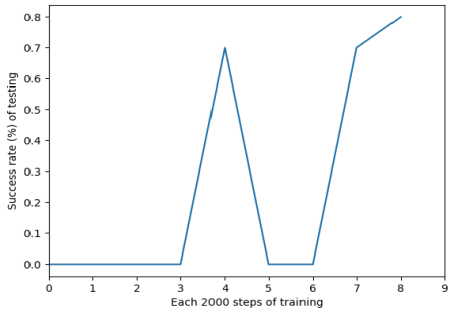
(a) Trained result in the initial environment



(b) Trained result in environment with one obstructions close to initizal position



(b) Trained result in the environment with two obstructions in horizontal direction



(d) Trained result in the environment with one obstructions close to target position

Figure 5. The success rate of different environments

4.5 Discussion

Our framework implements near real-time construction process planning in the experimental environment. For a single component, the trained neural network can achieve automatic construction process planning under any layout within the environment, and when the environment changes, the agent can still update the construction policy and complete the construction autonomously.

5 Conclusion

In this paper, we propose an AI-based framework for construction process planning in dynamic environments, and we validate our framework in a simple 3D environment to achieve near real-time construction process planning for a single component in a dynamic environment. We propose a framework based on the DRL algorithm that is introduced into the construction process. In a simplified simulation environment, we train a neural network oriented to the construction process of a single component. With this trained neural network, the component's transport path can be planned under any initial site layouts. Meanwhile, during the construction process, when

the change of site environment affects the component's transport path, the component's trained neural network will be updated according to the change of environment and a new transport path will be planned in real time.

While the research shows valuable preliminary results, the study suffers from the following limitations: 1. currently, we have tested the framework in a simple 3D simulation environment based on voxels, and further research is needed to extend the approach to BIM models (voxelization); 2. this experiment was performed only for a single component and the number of components needs to be increased; 3. the framework has high requirements for computational power, since in realistic construction scenarios, the framework needs to perform real-time calculations (within robot) on the changing environment and feedback the results. Therefore, computing power within robotic devices is one of the primary considerations affecting the efficiency of the framework.

Our future research is based on the above limitations: 1. to extend the framework to the BIM model; 2. to realize the construction process planning of multiple components and apply it to small-scale prefabricated structures.

References

Bertsekas, D. P., & Yu, H. (2012). Q-learning and enhanced policy iteration in discounted dynamic programming. *Mathematics of Operations Research*, *37*(1), 66–94. https://doi.org/10.1287/moor.1110.0532

Bock, T. (2015). The future of construction automation: Technological disruption and the upcoming ubiquity of robotics. *Automation in Construction*, *59*, 113–121. https://doi.org/10.1016/j.autcon.2015.07.022

Cai, P., Cai, Y., Chandrasekaran, I., & Zheng, J. (2016, February 1). Parallel genetic algorithm based automatic path planning for crane lifting in complex environments. *Automation in Construction*, Vol. 62, pp. 133–147. https://doi.org/10.1016/j.autcon.2015.09.007

Davila Delgado, J. M., Oyedele, L., Ajayi, A., Akanbi, L., Akinade, O., Bilal, M., & Owolabi, H. (2019). Robotics and automated systems in construction: Understanding industry-specific challenges for adoption. *Journal of Building Engineering*, *26*, 100868. https://doi.org/10.1016/j.jobe.2019.100868

Dutta, S., Cai, Y., Huang, L., & Zheng, J. (2020). Automatic re-planning of lifting paths for robotized tower cranes in dynamic BIM environments. *Automation in Construction*, *110*, 102998. https://doi.org/10.1016/j.autcon.2019.102998

Generalova, E. M., Generalov, V. P., & Kuznetsova, A. A. (2016). Modular Buildings in Modern Construction. *Procedia Engineering*, *153*, 167–172. https://doi.org/10.1016/j.proeng.2016.08.098

Jeong, W. S., Chang, S., Son, J. W., & Yi, J. S. (2016). BIM-integrated construction operation simulation for just-in-time production management. *Sustainability (Switzerland)*, *8*(11), 1–25. https://doi.org/10.3390/su8111106

Lin, Y., Wu, D., Wang, X., Wang, X., & Gao, S. (2014). Lift path planning for a nonholonomic crawler crane. *Automation in Construction*, *44*, 12–24. https://doi.org/10.1016/j.autcon.2014.03.007

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., & Riedmiller, M. (2013). *Playing Atari with Deep Reinforcement Learning*. 1–9. Retrieved from http://arxiv.org/abs/1312.5602

Safouhi, H., Mouattamid, M., Hermann, U., & Hendi, A. (2011). An algorithm for the calculation of feasible mobile crane position areas. *Automation in Construction*, *20*(4), 360–367. https://doi.org/10.1016/j.autcon.2010.11.006

Soltani, A. R., Tawfik, H., Goulermas, J. Y., & Fernando, T. (2002). Path planning in construction sites: Performance evaluation of the dijkstra, a*, and GA search algorithms. *Advanced Engineering Informatics*, *16*(4), 291–303. https://doi.org/10.1016/S1474-0346(03)00018-1

Vrabie, D., & Lewis, F. L. (2010). Approximate dynamic programming. *The Control Systems Handbook: Control System Advanced Methods, Second Edition*, *II*, 1385–1418. https://doi.org/10.1201/b10384

Wu, K., García de Soto, B., & Zhang, F. (2020). Spatio-temporal planning for tower cranes in construction projects with simulated annealing. *Automation in Construction*, *111*, 103060. https://doi.org/10.1016/j.autcon.2019.103060

Wu, Z., Lian, W., Unhelkar, V., Tomizuka, M., & Schaal, S. (2020). Learning dense rewards for contact-rich manipulation tasks. *ArXiv*.

Zhang, Z., & Pan, W. (2020). Lift planning and optimization in construction: A thirty-year review. *Automation in Construction*, *118*(May), 103271. https://doi.org/10.1016/j.autcon.2020.103271

Zhang, Z., Pan, W., & Zheng, Z.-J. (2019). Transfer Learning Enabled Process Recognition for Module Installation of High-rise Modular Buildings. *Modular and Offsite Construction (MOC) Summit Proceedings*, 268–275. https://doi.org/10.29173/mocs103

Zhu, A., Pauwels, P., & de Vries, B. (2020). Robot construction simulation using deep reinforcement learning. *EG-ICE 2020 Workshop on Intelligent Computing in Engineering, Proceedings*, 472–480.