Received September 17, 2021, accepted October 4, 2021, date of publication October 5, 2021, date of current version October 13, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3118109

Automatic Curriculum Design for Object Transportation Based on Deep Reinforcement Learning

GYUHO EOH AND TAE-HYOUNG PARK, (Member, IEEE)
Industrial AI Research Center, Chungbuk National University, Cheongju 28116, South Korea

Corresponding author: Tae-Hyoung Park (taehpark@cbnu.ac.kr)

This work was supported by the Grand Information Technology Research Center Support Program supervised by the Institute for Information and Communications Technology Planning and Evaluation (IITP) through the Ministry of Science and ICT (MSIT), South Korea, under Grant IITP-2021-2020-0-01462.

ABSTRACT This paper presents an automatic curriculum learning (ACL) method for object transportation based on deep reinforcement learning (DRL). Previous studies on object transportation using DRL have a sparse reward problem that an agent receives a rare reward for only the transportation completion of an object. Generally, curriculum learning (CL) has been used to solve the sparse reward problem. However, the conventional CL methods should be manually designed by users, which is difficult and tedious work. Moreover, there were no standard CL methods for object transportation. Therefore, we propose an ACL method for object transportation in which human intervention is unnecessary at the training step. A robot automatically designs curricula itself and iteratively trains according to the curricula. First, we define the difficult level of object transportation using a map, which is determined by the predicted travelling distance of an object and the existence of obstacles and walls. In the beginning, a robot learns the object transportation at an easy level (i.e., travelling distance is short and there are less obstacles around), then learns a difficult task (i.e., the long travelling distance of an object is required and there are many obstacles around). Second, training time also affects the performance of object transportation, and thus, we suggest an adaptive determining method of the number of training episodes. The number of episodes for training is adaptively determined based on the current success rate of object transportation. We verified the proposed method in simulation environments, and the success rate of the proposed method was 14% higher than no-curriculum. Also, the proposed method showed 63% (maximum) and 14% (minimum) higher success rates compared with the manual curriculum methods. Additionally, we conducted real experiments to verify the gap between simulation and practical results.

INDEX TERMS Curriculum learning, object transportation, deep reinforcement learning, difficulty level.

I. INTRODUCTION

Recently, object transportation has come to occupy an important position in logistics [1], exploration [2], and service robotics fields [3]. Traditionally, many researchers have tried to study object transportation techniques by imitating nature's behavior models. A dung beetle rolls cow dung to build a house using it or eat it [4], and ants transport prey to their nest together [5]. They can transport an object using prearranged behavior patterns such as pulling, pushing, and enclosing actions. Inspired by these actions, four primary object transportation methods have been suggested in the robotics field [6]: *grasping*, *pushing*, *caging*, and *tool-using* methods. Each method has its own advantage, but it cannot be generally applied to practical environments due to restrictions of high control complexity or tool-using. To overcome these restrictions, deep reinforcement learning (DRL)-based object transportation methods have been presented. The DRL algorithm is an effective solution for addressing complicated object transportation problems by reducing control complexity.

In DRL-based object transportation, it is important that proper rewards should be given at the right time during the training process. However, one-time reward is given only

The associate editor coordinating the review of this manuscript and approving it for publication was Yingxiang Liu.

when the object reaches a goal; it is difficult to achieve the goal by the random actions of a robot, which is known as a sparse reward problem [7]. Therefore, the effective design of training procedures is necessary for receiving frequent success rewards by step-wise learning; if an agent sufficiently learns an easy object transportation task, then the agent moves on to a difficult task. This is referred to as curriculum learning (CL). The CL enables a robot to receive more rewards and learn effectively. However, despite these advantages, previous CL methods required to be manually designed by a user [8], [9]; the CL should be frequently modified according to the training results, and diverse curricula should be tested to find a proper method as well. This trial-and-error approach does not guarantee robust performance, and it is difficult to find an effective CL method. Thus, many researchers have suggested automatic curriculum learning (ACL) methods which design curricula automatically without human's intervention [10], [11]. In the ACL, an agent determines the proper difficulty level for efficient learning and assigns a task for each episode. The whole learning procedures are automatically designed, and the agent learns tasks according to the curricula.

In this paper, we propose a novel ACL for object transportation considering spatial and temporal information. First, we define the difficulty level of object transportation according to the predicted travelling distance of an object and the existence of surrounding obstacles and walls; the success rate of object transportation highly depends on the status of travelling routes. The difficulty level can be described as a probability between 0 and 1, and it can be converted into a grayscale image; this image is exploited for selecting pose-initialization region at the training step. An agent can know the difficulty level of object transportation by referring to the image. Second, an agent adjusts the number of episodes in real time to secure a predefined success rate. The number of training episodes is not fixed, and the agent modifies the number while learning. Guaranteeing sufficient training time is an important factor that affects performance.

The contributions of this paper are summarized as follows:

1) We present an automatic curriculum learning framework for object transportation based on deep reinforcement learning.
2) We present an automatic region selection method for pose-initialization of an object by defining the difficulty level of object transportation from a spatial point of view.
3) An adaptive determining method of the number of training episodes is suggested to guarantee a predefined success rate.
4) We verified the proposed method by conducting practical experiments as well as simulations.

The proposed method appears to be partially similar to our previous paper [8]. However, there are fundamentally different parts as follows. First, this paper focuses on an automatic curriculum method, not the manually built curriculum that was presented in the previous paper. The previous study required a manual design, but the proposed ACL method enables a robot to make a training plan itself. Second, the proposed curricula are automatically designed based on temporal information. Guaranteeing sufficient training time to satisfy the desired success rate is a key factor in the proposed ACL. Third, we consider the feasible travelling distance by reflecting surrounding obstacles and walls in this paper; the travelling distance was calculated by a simple Euclidean norm assuming the shortest distance without obstacles in the previous paper. Finally, we conducted practical experiments as well as simulations to verify the proposed method.

This paper is organized as follows. Section II describes related works for object transportation. Section III explains the basic concepts of reinforcement learning, deep Q-network, and curriculum learning. Section IV defines the problem of this paper, and section V shows the system overview of the proposed ACL framework. Two ACL methods are presented in section VI: the generation method of difficulty level map (section VI-A) and an adaptive determining method of the number of training episodes (section VI-B). Simulation and practical experiments are suggested in section VII and VIII, respectively. We discuss the importance of this paper and the future work in section IX. Finally, the conclusion is given in section X.

II. RELATED WORK

A. OBJECT TRANSPORTATION METHODS

Object transportation methods are divided into four categories, such as grasping, pushing, caging, and tool-using methods.

First, in the grasping method, a single or more robots grasp an object with a robotic arm and transport it toward a goal in 2D [12], [15], [16] or 3D environment [13], [14], [30]. Robots do not have to consider the motion of an object during transportation because the object is tightly coupled with the robotic arm; the object can be easily transported if it is successfully grasped once by a gripper. However, a gripping technique requires complicated preliminary steps [31] such as 1) recognizing the object, 2) controlling the motion of a robot, 3) determining where the gripper grasps, and 4) grasping control of a manipulator. These consecutive actions are intractable and easy to fail. Additionally, a large object, which is larger than the size of a gripper, cannot be transported.

Second, a single or more robots push an object in the pushing method [17]–[19], [21], [32]; the number of hired robots is determined by the weight of a target object. In contrast to the grasping method, preliminary actions are not necessary for pushing the object. The movement of robots is free and controllable because robots are not held tightly to the object. However, external physical information such as ground friction, the stiffness of the object, or the geometrical shapes of the object and robot should be known in advance for precise control. The perfect prediction of the whole information is almost impossible, which is the primary reason for inapplicability in practical applications. Also, many studies assumed

TABLE 1. Diverse object transportation methods.

Method	Characteristics	Pros	Cons	Reference
Grasping	A single or more robots grasp an object with a manipulator and transport it toward a goal.	- There is no need to consider the uncertainty of the object's motion.	- Preliminary actions (e.g., detecting and grasping an object) are necessary for object transportation. - A large object cannot be transported.	[12–16]
Pushing	A single or more robots push an object toward a goal using their bodies.	- Any preliminary actions are not necessary. - A large object can be transported by cooperative pushing behavior between multiple robots.	- The movement of an object should be precisely predicted. - External environment variables, such as ground friction or stiffness of an object, should be shared in advance to robots.	[17–21]
Caging	Multiple robots approach and enclose an object together, then transport it toward a goal while maintaining the formation.	- Robots do not have to consider the uncertain movement of an object during transportation. - Large or multiple objects can be transported.	- An excessive number of robots is necessary to enclose an object. - A lot of space should be secured for object transportation.	[22–25]
Tool-using	A robot manipulates an object using additional equipment such as a lifter, stick, spring, or rope.	- A robot can transport an object easily using tools.	- This method cannot be applied to general transportation applications.	[26–29]

a quasi-static environment, which is an inappropriate assumption when a slippery object is transported.

Third, multiple robots approach and enclose an object using a predefined caging formation (e.g., circle [22], [23]), then transport the object to a goal while maintaining the formation [24], [33]. In the caging method, the robots do not have to predict the movement of an object because the object cannot escape from the caging formation during the transportation process; robots only need to maintain the formation. Also, large or multiple objects can be transported because objects are guaranteed to be inside the formation. However, it is difficult to maintain a specific formation all the time while moving. For robust and stable transportation, multiple robots should be synchronized in real-time, which is difficult in practical applications. Also, the excessive space for enclosing the target object is necessary, which makes object transportation more difficult in the environment where narrow passages exist.

Finally, a single or more robots can transport an object with the help of tools such as a lifter [26], handcart [34], stick [29], spring [27], or rope [28]. Tools facilitate object transportation by restricting the object's motion in several ways. However, the movement of robots is also restricted by tools, like the grasping method. For example, two robots connected with a rope cannot move freely and they should move together by taking into account the connection between them. In addition, a specific preparation step is needed for object transportation in advance. For example, the action of lifting an object should be completed before the beginning of object transportation. The grasping method can be seen as one of the tool-using methods because robots use manipulators. Thus, the disadvantages of the tool-using method are almost the same as those of the grasping method.

The descriptions, advantages, and disadvantages of the above-mentioned object transportation methods are summarized in Table 1. It is possible to select one of the methods according to the purpose of use because each object transportation method has different pros and cons. Initially, many researchers studied the grasping method because of its robust transportation ability [35]. After that, the pushing method became the mainstream because of the advantage of not requiring preliminary actions [36]. The caging method combined the advantages of grasping and pushing methods; this method does not have to consider the object's complicated motion but also the requirement of preliminary actions [37]. Recently, the pushing method has become applicable in practical environments again because of the rapid progress of reinforcement learning (RL) [38]–[40]. Therefore, we focus on RL-based object transportation using pushing behavior in this paper.

B. OBJECT TRANSPORTATION TECHNIQUES USING REINFORCEMENT LEARNING

Previous object transportation studies described in section II-A have their own advantage, but there is a common disadvantage that controlling the movement of an object is difficult. In the real environment, unexpected motion errors due to slippery ground's material or inaccurate localization can deteriorate transportation ability; the previously presented object transportation studies were difficult to cope with such a situation. Recently, the DRL algorithm, combining conventional RL with deep neural network, has risen to prominence, and it has been known as an appropriate solution for optimal decision and robust control. A representative example of DRL applications was presented by Google DeepMind Lab [41], [42]; the DRL algorithm showed overwhelming performance compared with humans in the Atari 2600 and Go games. Since then, many researchers have tried to apply the DRL algorithm to the object transportation field because of its positive aspects [43]–[45].

Zhang *et al.* [43] presented a decentralized cooperative object transportation based on deep Q-network. Two robots carry a large rod through a doorway using cooperative behavior. Complicated motion prediction was not necessary, but the rod should be attached to robots in advance; this method cannot be applied to the general purpose of object transportation. Li *et al.* [46] introduced a deep recurrent neural network-based pushing method using visual images as input. This method can be used to transport an object that has unknown physical properties, and Manko *et al.* [47] also uses a 2D-image map for learning. Xiao *et al.* [48] suggested pushing-based object transportation which trains decentralized Q-network with the help of a centralized Q-network for action selection. A variant caging method using DRL was presented by considering event-triggered communication and control [45]. Some researchers have suggested the RL-based box-manipulation method for animation generation [39], [44].

III. BACKGROUND

A. REINFORCEMENT LEARNING

Reinforcement learning is a kind of machine learning that enables an agent to learn how to act in an interactive environment to maximize accumulated rewards. An agent (i.e., robot) can learn a policy itself using feedback on its actions. The RL can be applied to solve sequential decision-making under uncertainty, and this is formulated as the Markov Decision Process (MDP).

The MDP consists of 5 tuples such as the state space ($\mathcal{S}$), a set of actions ($\mathcal{A}$), the reward function ($\mathcal{R}$), the state transition probability ($\mathcal{T}$), and the discount factor (γ). The state space $\mathcal{S}$ is an observable set, and an agent observes a state $s_t \in \mathcal{S}$ at timestamp t. The agent selects an action $a_t \in \mathcal{A}$ using a policy function $\pi(a|s)$; the policy is a mapping function from $\mathcal{S}$ to $\mathcal{A}$. The reward function is an expected return when an agent takes an action a in the state s: $\mathcal{R}(s, a) = \mathbb{E}[r_{t+1}|s_t = s, a_t = a]$. The state transition probability $\mathcal{T}$ is the probability of moving from s to s' when an agent takes action a: $\mathcal{T}(s, s', a) = \mathbb{P}[s_{t+1} = s'|s_t = s, a_t = a]$. Finally, the discount factor $\gamma \in [0, 1]$ affects how much weight it gives to future rewards in the value function. If the discount factor is zero ($\gamma = 0$), immediate reward is given only by current state and action. In contrast, if the discount factor is one ($\gamma = 1$), the future rewards are calculated without loss.

B. DEEP Q-NETWORK

Q-learning is a model-free RL and it trains in a way that maximizes expected accumulated rewards [49]. In Q-learning, the state-action value function $Q_\pi(s, a)$ is the expected return when taking an action a given state s following a policy π:

$$\begin{aligned} Q_\pi(s, a) &= \mathbb{E}_\pi(R_t|s_t = s, a_t = a) \\ &= \mathbb{E}_\pi\left(\sum_{k=0}^{\infty} \gamma^k r_{t+k+1}|s_t = s, a_t = a\right), \end{aligned} \tag{1}$$

where R_t is discounted return at timestamp t, which is the sum of total rewards obtained by the agent while exploring an environment. In this case, the reward is proportionally reduced in the far-off future by the discount factor $\gamma \in [0, 1]$.

The Q-learning algorithm updates the Q-value using the weighted sum method by adding the old Q-value and the learned value as follows:

$$\begin{aligned} Q(s_t, a_t) \leftarrow (1-\alpha)\, &\overbrace{Q(s_t, a_t)}^{\text{old value}} \\ &+ \underbrace{\alpha\, [r_t + \gamma \max_a Q(s_{t+1}, a)]}_{\text{learned value}}, \end{aligned} \tag{2}$$

where $\alpha \in [0, 1]$ is the learning rate which controls how fast an agent learns. If the learning rate is high, the Q-value is modified quickly. On the contrary, if the learning rate is low, the Q-value will be updated slowly. For example, if the learning rate is zero ($\alpha = 0$), an agent will not learn anything.

The Q-table is a lookup table for recording the maximum expected future rewards of actions in each state. Q-values are recorded in a finite-sized table, and these values are updated by a Q-learning algorithm using (2). This tabular method can be applied in a finite small-sized environment such as the grid world [50] because the information amount for describing the small environment is not large. In the real environment, however, it is difficult to describe the whole environment using the Q-table because the state and action space are large.

Deep Q-network (DQN), therefore, was presented to overcome the limited dimension representation of the Q-table [41], [51]. A Q-table was replaced with a neural network (NN) using function approximation; the NN can return all Q-values of possible actions given the state. If the NN has multiple hidden layers, we call it the DQN. The weights of DQN are described by θ, and thus, (2) is modified as follows:

$$\begin{aligned} Q(s_t, a_t; \theta) \leftarrow &(1-\alpha)Q(s_t, a_t; \theta) \\ &+ \alpha[r_t + \gamma \max_a Q(s_{t+1}, a; \theta^-)], \end{aligned} \tag{3}$$

where θ and θ^- are the weight parameters of the local and target network, respectively. The loss function of DQN is represented as follows:

$$\begin{aligned} \mathcal{L}_t(\theta_t) = \mathbb{E}_{s_t,a_t,r_t,s_{t+1}}\Big[&\overbrace{r_t + \gamma \max_{a_{t+1}} Q(s_{t+1}, a_{t+1}; \theta_t^-)}^{\text{target}} \\ &- \underbrace{Q(s_t, a_t; \theta_t)}_{\text{prediction}} \Big]^2, \end{aligned} \tag{4}$$

where θ_t and θ_t^- are the parameters of the local and target Q-network at timestamp t, respectively. An agent trains the network to minimize the loss function $\mathcal{L}_t$.

Meanwhile, the target Q-network is updated at regular intervals to reduce unstable learning. In addition, an *experience replay memory* was introduced to disconnect the time-correlation between training steps [52]. Instead, an agent samples a small batch of tuples from a replay buffer and trains it. The periodical update of the target network and the experience replay were key factors in applying the DRL algorithm to real applications [41].

C. CURRICULUM LEARNING FOR OBJECT TRANSPORTATION

Curriculum learning is a step-wise training method for high performance and fast learning completion [53]. An agent learns how to achieve a goal according to a curriculum which consists of gradual steps from easy to difficult tasks. The objective of object transportation is transporting an object to a goal, and two sequential processes, i.e., approaching and pushing an object, should be executed in order to accomplish the objective. Generally, a reward is given once when the object reaches a goal at the learning step; reaching a specific goal position by random actions of a robot is a sparse situation, which is a *sparse reward problem*. For solving the sparse reward problem, the CL can be applied to the DRL-based object transportation method. The definitions of easy and difficult tasks could differ for each application; we define the difficulty level of object transportation according to the following criteria: 1) estimated travelling distance of an object and 2) the number of training episodes. First, it is more difficult to transport an object to a goal as the estimated travelling distance increases. If an object travels a long distance, it is easily stuck or loses its way due to surrounding obstacles or walls. Second, learning will be easier as the training time increases. This means that the number of training episodes affects learning performance. Therefore, the CL design of object transportation should be considered by these two factors.

Meanwhile, the manual CL design is tedious and trial-and-error prone; thus, many studies have proposed automatic curriculum learning (ACL) methods [7], [11], [54]. In ACL methods, an agent automatically selects the range of training tasks, syllabus, or sub-goals; humans do not have to intervene in the training procedures. Using ACL, an agent easily learns how to manipulate an object without the complicated manual design of CL. In this paper, a novel ACL design is presented by considering estimated travelling distance and the number of training episodes.

IV. PROBLEM DEFINITION

The problem of this paper is finding the optimal ACL method for maximizing the success rate of object transportation. An example of object transportation is presented in Fig. 1; a robot pushes an object toward a goal by avoiding obstacles. If the center of an object is located within $d_{success}$ from the goal, the robot succeeds in object transportation. On the contrary, if the object cannot be transported within a given time, object transportation fails. Thus, the object transportation addressed in this paper is an episodic task.

Meanwhile, the ACL design $\mathcal{D}(i)$ consists of the free configuration space $\mathcal{Q}_{free}(i)$ at i^{th} episode and the number of training episodes $\mathcal{N}_{episode}(i)$ as follows:

$$\mathcal{D}(i) = \{\mathcal{Q}_{free}(i), \mathcal{N}_{episode}(i)\}, \tag{5}$$

where $\mathcal{Q}_{free}(i) = \mathcal{Q}(i) \setminus \bigcup_j \mathcal{QO}_j$, $\mathcal{Q}(i)$ is the permissible pose initialization region of an object at i^{th} episode, and $\mathcal{QO}_j$ is the configuration of a robot that intersect an obstacle j.

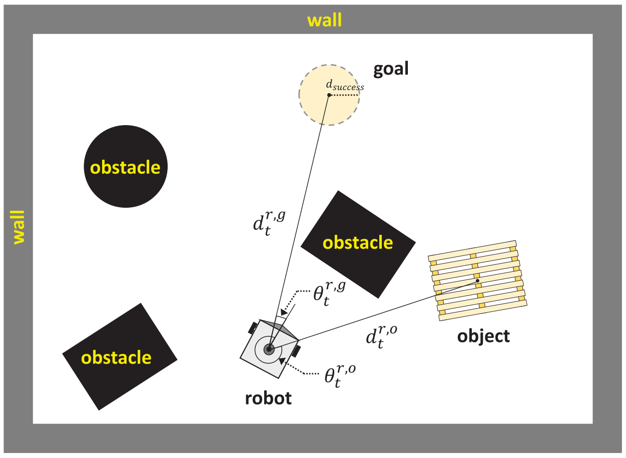

FIGURE 1. The example of problem description. A single robot transports an object to a goal by pushing behavior. There are multiple (three in this figure) static convex-shaped obstacles in an environment.

Therefore, the problem is formulated as follows:

$$\begin{aligned} &\arg\max_{\mathcal{D}} \mathbb{E}\left[\sum_{k=1}^{N_f} \left(p(\tau_k)|\pi_\theta^{\mathcal{D}}\right)\right], \\ &\text{s.t } p(\tau_k) = 1 \ \ if \ \ d_t^{o,g} \leq d_{success}, \quad \forall t \leq \mathrm{T}_{max} \\ &\quad p(\tau_k) = 0 \ if \ d_t^{o,g} > d_{success}, \quad \forall t \leq \mathrm{T}_{max} \end{aligned} \tag{6}$$

where τ_k is a k^{th} task, $p(\tau_k)$ is a success flag represented by a boolean value, N_f is the total number of tasks, and $\pi_\theta^{\mathcal{D}}$ is a policy if an agent is trained according to the curriculum $\mathcal{D}$. The value T_{max} is the maximum steps for each episode and the time t should be lesser than or equal to T_{max}.

Five assumptions are made in this paper. First, a two-wheeled mobile robot is used in a 2D-Euclidean plane. In a 3D-environment, we cannot apply the pushing method. Second, we assume that the shape of a robot is a circle. Third, we assume that a robot can detect and recognize the surrounding environments, such as a goal, an object, and obstacles. Diverse tracking methods can be exploited to localize them, but these are not the main focus of this paper. Thus, we omit the detailed tracking method. Fourth, there is a feasible transporting path of an object when the positions of a robot and the object are initialized at the learning stage. If there is no feasible transport route, the problem of object transportation cannot be solved. Finally, all obstacles are assumed to be convex shapes. If the shape of obstacles is concave, an object is easily stuck due to the concave structure of obstacles. A robot cannot transport an object when it is stuck once.

V. SYSTEM OVERVIEW

The proposed object transportation learning consists of three components such as an automatic curriculum learning (ACL) module, simulator, and training modules.

First, the ACL module generates a difficulty level map (DLM) by considering the predicted travelled distance and surrounding obstacles. In addition, the number of training episodes is determined in the ACL module. The detailed explanations of the ACL module will be described in section VI.

Second, the simulator receives the pose initialization information of an object and the number of training episodes from the ACL module, then builds simulations. An agent (a robot) takes an action a_t at the state s_t and receives a new state s_{t+1} according to state transition probability $\mathcal{T}$ and a reward r_t. The probability $\mathcal{T}$ is reflected by the physical model in the simulator. The tuple $(s_t, a_t, r_{t+1}, s_{t+1})$ is transmitted to the training module.

Finally, the agent learns how to transport an object using the deep Q-network with an experience replay in the training module. The state of an agent is described as follows:

$$s_t = [d_t^{r,o}, \cos\theta_t^{r,o}, \sin\theta_t^{r,o}, d_t^{r,g}, \cos\theta_t^{r,g}, \sin\theta_t^{r,g}], \quad (7)$$

where $\theta_t^{i,j}$ and $d_t^{i,j}$ are the angle and distance between i and j $(i, j \in \{\mathbf{r}obot, \mathbf{o}bject, \mathbf{g}oal\})$, as shown in Fig. 1. Three consecutive states $\{s_t, s_{t-1}, s_{t-2}\}(t \geq 3)$ are concatenated to form a unified state $(\tilde{s}_t)$. The action space consists of 6 actions: $\{\mathbf{F}, \mathbf{B}, \mathbf{FL}, \mathbf{FR}, \mathbf{BL}, \mathbf{BR}\}$, where **F**, **B**, **L**, and **R** represent forward, backward, left, and right, respectively. The specific values of actions are determined by considering the size of a target object and the environment. For example, we set a large translational velocity to forward action if the environment is large. An agent receives +1 and -0.01 rewards if an object reaches a goal and collides with obstacles or walls, respectively. In the working process without collisions, the agent receives $0.1 \times \Delta d_t^{r,o} + 0.5 \times \Delta d_t^{o,g}$, where $\Delta d_t^{r,o}$ is the distance difference between a robot and an object, and $\Delta d_t^{o,g}$ is the distance difference between an object and a goal. A robot can learn the obstacle avoidance method using the reward function without referring to the obstacles' state because all positions of obstacles and a goal are fixed; a robot can infer the existence of walls and obstacles from the state s_t using the reward function related to collision. Deep Q-network consists of three dense layers with rectified linear unit (ReLU) and each layer has 256 nodes. The ϵ-greedy algorithm is used for selecting an action, which is helpful for balancing the ratio between exploration and exploitation [55]. In addition, the experience replay memory is adopted to break the temporal correlation between training data [41].

VI. AUTOMATIC CURRICULUM LEARNING

The proposed ACL method uses spatial and temporal approaches to adjust the difficulty level of object transportation; the spatial approach is the generation of difficulty level map (DLM), and the temporal approach is the adaptive determination of the number of training episodes (ADE). A robot determines the region itself where an object is initialized using the DLM. The difficult level of object transportation is affected by the initialized position of an object at the beginning of each episode. The DLM provides a basis for determining where the position of an object is initialized while training, and thus, the DLM is related to the effective learning method. In addition, a robot can set up the number of training episodes to become an expert at a current task. If a robot does not show the desired performance for the current task, then the robot learns the task continuously until the robot is familiar with it; the ADE can adjust the learning speed by limiting the desired performance. Two above-mentioned methods have in common that they are both intended to solve a sparse reward problem. Efficient and robust learning is possible by allowing more frequent success for a robot. The detailed explanations of DLM and ADE will be described in the next sections.

A. THE GENERATION OF DIFFICULTY LEVEL MAP

The DLM is generated using the predicted travelling distances of an object and the existence of obstacles and walls. If an object is transported a long distance, it is easy to fail transportation due to unexpected motion errors. In addition, the closer an object is to obstacles or walls, the more difficult it is to transport due to their route disturbance. Therefore, a robot should reflect these factors to build a DLM.

The overall procedures of a DLM generation example are shown in Fig. 2. We assume that three obstacles (two rectangular and one circular shapes) exist in an environment, as shown in Fig. 2(a). The travelling distances, starting from an initial position to a goal, are calculated by the A* algorithm [56] with normalization under obstacles' configuration space, as shown in Fig. 2(b): distance DLM (DDLM). Multiple levels of configuration space with obstacle's variant radii and walls are stacked up, as shown in Fig. 2(c): boundary DLM (BDLM). Finally, Fig. 2(d) shows the final DLM summing up the multiple DLMs, Fig. 2(b) and Fig. 2(c).

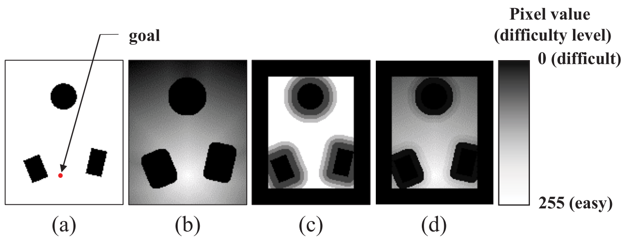

FIGURE 2. The generation process of difficulty level map of object transportation. (a) Original map (b) Difficulty level map according to predicted travelling distance of an object using A* algorithm, i.e., distance DLM (DDLM). (c) Difficulty level map according to obstacles and walls, i.e., boundary DLM (BDLM) (d) Final difficulty level map.

For describing above-mentioned steps as mathematical expressions, we define a configuration space i^{th} obstacle, $\mathcal{QO}_i(j)$, with an extended radius r_j of j^{th} obstacle as follows:

$$\mathcal{QO}_i(j) = \{q \in \mathcal{Q} | R(q, r_j) \cap \mathcal{WO}_i \neq \emptyset\}, \quad (8)$$

where $q \in \mathbb{R}^2$ is a robot pose, $\mathcal{Q}$ is the configuration space, and $\mathcal{WO}_i$ is the workspace of i^{th} obstacles. The function $R(q, r_j)$ is written as

$$R(q, r_j) = \{(x', y') | (x - x')^2 + (y - y')^2 \leq (r + r_j)^2\}, \quad (9)$$

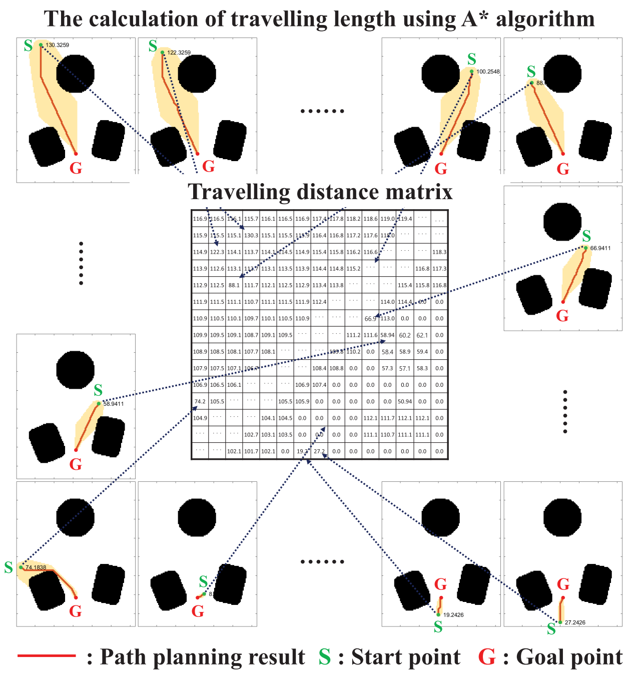
The calculation of travelling length using A* algorithm
Travelling distance matrix
S
G
: Path planning result
S : Start point
G : Goal point

FIGURE 3. The generation process of travelling distance matrix for building a distance difficulty level map (DDLM). First, divide the whole environment into grids with a specific resolution; it can be expressed as a matrix form if the whole environment is assumed to be a rectangular shape. Second, calculate the travelling distances for all starting points using the path planning algorithm. In this case, any planning algorithm can be used to calculate distances; we select the A* algorithm in this paper [56]. Finally, generate a travelling distance matrix by recording the path planning results of an object for all start points.

where r is the radius of a robot, and r_j is the extended radius of an obstacle at j^{th} step.

The distance DLM (DDLM) generation steps are as follows. First, generate the free configuration space $\mathcal{Q}_{free} = \mathcal{QO}_i(0)$ ignoring the extended radius of obstacles ($r_0 = 0$). Second, divide the whole region into multiple grids with a specific resolution. If we assume the world has a rectangular shape, the whole region can be described as a matrix form. In the case of irregular-shaped environments, we can approximate the environment as a rectangular shape based on the outer-most contour line. In this case, the regions that do not really exist are colored black. Third, calculate the travelling distance from all starting points to a goal using the A* algorithm. In this case, the available path planning algorithm is not limited to the A*. Fourth, make the calculated distances as a matrix form. The above steps from the second to fourth are shown in Fig. 3. Finally, normalize the travelling distance matrix as a gray-scale figure from 0 to 255 level. For example, the near region from a goal is covered in white, and the far region from a goal is covered in black. The final output of above processes is illustrated in Fig. 2(b). The DLM generation processes considering obstacles and walls are shown in Fig. 4. First, generate multiple configuration spaces obstacles $\mathcal{QO}_i(j)$ by changing the extended radius of obstacles $r_j (j \in \{1, 2, \ldots, N_{er}\})$. Second, assign different weight ratios ω_j according to the proximity of obstacles

Algorithm 1: The Difficulty Level Map Generation

```
   Input: $\mathcal{M}$, $r$, $\mathbf{p}_G$
   Output: Difficulty level map, $\mathcal{NT}$
 1 Load 2D-environmental map, $\mathcal{M}$;
 2 Generate the free configuration space, $\mathcal{Q}_{free}$;
 3 for $p_r$ in $\mathcal{Q}_{free}$ do
 4     $Path_{A*}(\mathbf{p}_r, \mathbf{p}_G) \leftarrow$ A*-planning($\mathcal{M}, \mathbf{p}_r, \mathbf{p}_G$);
 5     $dist(\mathbf{p}_r) \leftarrow \sum \|\Delta Path_{A*}(\mathbf{p}_r, \mathbf{p}_G)\|_2$;
 6 end
 7 $\mathcal{QO} \leftarrow$ Generate an obstacle DLM using (10);
 8 $\mathcal{QW}(\mathcal{M}) \leftarrow$ Generate a wall DLM using (11);
 9 $\mathcal{ND} \leftarrow$ Normalize($dist$);
10 $\mathcal{NOW} \leftarrow$ Normalize($\mathcal{QO} + \mathcal{QW}(\mathcal{M})$);
11 $\mathcal{NT} \leftarrow$ Normalize($\mathcal{ND} + \mathcal{NOW}$);
```

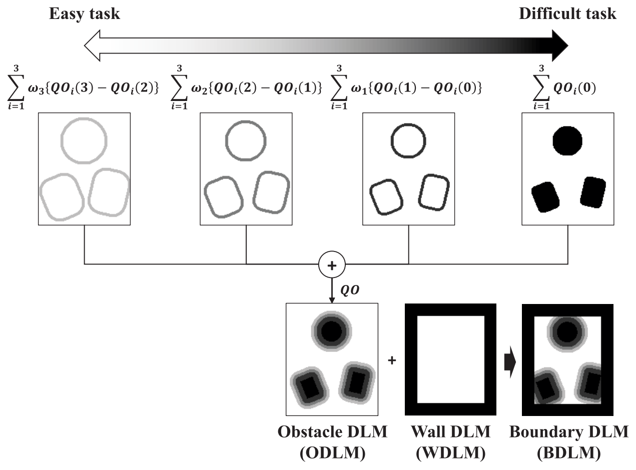
Easy task
Difficult task
$\sum_{i=1}^{3} \omega_3\{QO_i(3) - QO_i(2)\}$
$\sum_{i=1}^{3} \omega_2\{QO_i(2) - QO_i(1)\}$
$\sum_{i=1}^{3} \omega_1\{QO_i(1) - QO_i(0)\}$
$\sum_{i=1}^{3} QO_i(0)$
+
QO
+
Obstacle DLM (ODLM)
Wall DLM (WDLM)
Boundary DLM (BDLM)

FIGURE 4. Example of the boundary difficulty level map (BDLM) generation process. An ODLM is the weighted sum of the different configuration spaces defined by the extended radii of obstacles. In this figure, the larger the index, the larger the radius: $r_1 < r_2 < r_3$. The value of r_0 is zero. WDLM is generated by following the outer-most boundary. A BDLM is the sum of ODLM and WDLM.

or walls. Third, summing-up the whole configuration space obstacles, and build an obstacle DLM (ODLM) as follows:

$$\mathcal{QO} = \sum_{i=1}^{M_{obs}} \sum_{j=1}^{N_{er}} \omega_j \{\mathcal{QO}_i(j) - \mathcal{QO}_i(j-1)\}, \tag{10}$$

where M_{obs} is the number of obstacles. Fourth, assign the same thickness value to walls because it is easy to be stuck if an object approaches the walls closely; We eliminate the possibility of pose-initialization near the walls. This wall DLM (WDLM), $\mathcal{QW}(m)$, is generated by the following equation:

$$\mathcal{QW}(m) = \{q \in \mathcal{Q} | R(q) \cap \mathcal{W}(m) \neq \emptyset\}, \tag{11}$$

where m is an environmental map in the grid world. Finally, a BDLM including obstacles and walls is generated by normalizing the sum of the ODLM and WDLM.

The summary of the total DLM generation is described in Algorithm 1. First, load a 2D-map, $\mathcal{M}$, and generate the free configuration space, $\mathcal{Q}_{free}$ (line 1–2). Second, generate a path $Path_{A*}(\mathbf{p}_r, \mathbf{p}_G)$ using the A* algorithm for all $\mathbf{p}_r \in \mathcal{Q}_{free}$, and the total travelling distances of each path are calculated (line 3–6). Third, an obstacle and a wall DLM ($\mathcal{QO}$ and $\mathcal{QW}$) are generated by (10) and (11), respectively (line 7–8). Finally, normalize the DDLM and BDLM, and the total DLM ($\mathcal{NT}$) is calculated by normalizing the sum of DDLM and BDLM (line 9–11).

The proposed DLM provides comparable indices only according to the spatial difficulty level of object transportation, which means that the problem of selecting the pose-initialization region using the DLM remains. Generally, the uniform partitioning of the region is appropriate because we already consider feasible travelling distances using path planning and surrounding obstacles. For example, we divided the whole area into two regions in the bisection partitioning with [0, 0.5) and [0.5, 1.0) ratios of the DLM. Meanwhile, we should also determine how many regions are divided into, and it depends on the size of the whole environment. If the environment is small, we partition the whole area into 2 or 3 regions. On the contrary, if the environment is large, we have to divide the whole region into many regions, such as 5 or more regions. As a result, all we have to do is determine the size of the region partitioning; a manual curriculum design is not necessary.

B. ADAPTIVE DETERMINING THE NUMBER OF TRAINING EPISODES

If a robot does not fully learn a task, then it is necessary to learn until the robot is familiar with the task. In other words, learning a difficult task is almost impossible in RL if there is not secured learning time for easier tasks. Therefore, we set a target success probability, $\mathbf{P}_{success}$, and move on to the more difficult task only if the success probability of a current task exceeds the $\mathbf{P}_{success}$; the robot learns the current task continuously until the desired success rate is reached. In other words, we can determine the trade-off relationship between the training time and success rate by adjusting the target success probability. For example, if the $\mathbf{P}_{success}$ is high, the training takes a long time, but the success rate of object transportation is high. On the contrary, if the $\mathbf{P}_{success}$ is low, the training takes a short time, but the success rate will be low.

Algorithm 2 describes the ADE procedure including the DQN training method. First, initialize related variables such as the number of unit episode (N_{unit}), success counting array (S_n), replay memory ($\mathcal{D}$), a local and target DQNs, and total DLM ($\mathcal{NT}$) (line 1–5). Second, record the success of the recent N_c trials to S_n (line 7–11). Third, increase the episode index and save the previous and current episode indices (line 12–18). In this case, if the average success rate is lower than the predefined success probability ($\mathbf{P}_{success}$), then an agent learns again in the current region. Fourth, calculate the minimum and maximum difficulty levels according to an episode index from the DLM, respectively (line 19–20). These levels are uniformly partitioned by the size of the whole environment, as already described in section VI-A. If the maximum difficulty level reaches 1, then the training process finishes (line 21–23). Fifth, reset an environment within the difficulty levels $[\rho_{min}, \rho_{max}]$. Finally, the conventional DQN training process is executed using an ϵ-greedy algorithm, an experience replay, and a target network (line 25–32) [41].

Algorithm 2: Adaptive Determining Method of Training Episodes

```
   Input: $\mathbf{P}_{success}, N_{unit}, N_c, p_{ratio}(\cdot)$
   Output: DQN training
1  Initialize the unit episode indices: $n_{pstep}, n_{cstep} \leftarrow 0$;
2  Initialize the success counting array $S_n$ with size $N_c$;
3  Initialize a replay memory $\mathcal{D}$;
4  Initialize a local and target network;
5  Generate the total DLM ($\mathcal{NT}$) using the Algorithm 1;
6  while True do
7      if object transportation is success then
8          $S_n(i_{ep}$ MOD $N_c) \leftarrow 1$;
9      else
10         $S_n(i_{ep}$ MOD $N_c) \leftarrow 0$;
11     end
12     $i_{ep} \leftarrow i_{ep} + 1$;
13     $n_{cstep} \leftarrow (i_{ep}$ MOD $N_{unit}) + 1$;
14     if $n_{cstep} \neq n_{pstep}$ and $\frac{\sum S_n}{N_c} < \mathbf{P}_{success}$ then
15         $i_{ep} \leftarrow i_{ep} - N_{unit}$;
16         $n_{cstep} \leftarrow (i_{ep}$ MOD $N_{unit}) + 1$;
17     end
18     $n_{pstep} \leftarrow n_{cstep}$;
19     $\rho_{min} \leftarrow p_{ratio}(n_{cstep})$;
20     $\rho_{max} \leftarrow p_{ratio}(n_{cstep} + 1)$;
21     if $\rho_{max} = 1$ then
22         break;
23     end
24     Reset an environment using the $\mathcal{NT}$ within
         $[\rho_{min}, \rho_{max}]$.;
25     Set $\epsilon$ according to $\epsilon$-greedy algorithm;
26     for iteration = 1 to $L$ do
27         Take an action $a_t$ with a probability $\epsilon$;
28         Receive next state $s_{t+1}$;
29         Store a MDP tuple to replay memory $\mathcal{D}$;
30         Train the DQN network by minimizing (4).;
31     end
32     Update a target network with the local Q-network
         every $K$ episodes.;
33 end
```

VII. SIMULATIONS

A. SIMULATION ENVIRONMENT

We constituted a simulation environment assuming a warehouse scenario, as shown in Fig. 5; a robot transports a target object (i.e., pallet) to a desired goal. The size of the total environment was 6.0 m (W) $\times$ 7.5 m (H), and the coordinate of the goal was (5.0 m, 0.0 m). The sizes, locations, and weights of a robot, pallet, and obstacles are presented in Table 2. We defined the proper velocities of a robot by considering

TABLE 2. The information of robot, pallet, and obstacles.

Name		Size	Location	Weight	Fixed
Robot		0.3 m (diameter)	Random	1.8 kg	No
Pallet		1.2 m (W) × 0.8 m (H)	Random	0.1 kg	No
Obstacle	Rectangle 1	1.13 m (W) × 0.54 m (H)	(4.25 m, 1.65 m)	Heavy	Yes
	Rectangle 2	1.00 m (W) × 0.56 m (H)	(4.6 m, -1.55 m)	Heavy	Yes
	Circle	0.76 m (diameter)	(0.7 m, -0.05 m)	Heavy	Yes

TABLE 3. Distance ratio intervals according to different function types when the total size of total episodes is 1000.

Episode index (i_{ep})	Bisection ($N_{max}=2$)			Quadrisection ($N_{max}=4$)		
	Uniform	Convex	Concave	Uniform	Convex	Concave
$[0, 250)$	[0, 0.50)	[0, 0.25)	[0, 0.71)	[0, 0.25)	[0, 0.06)	[0, 0.50)
$[250, 500)$				[0.25, 0.50)	[0.06, 0.25)	[0.50, 0.71)
$[500, 750)$	[0.50, 1.0)	[0.25, 1.0)	[0.71, 1.0)	[0.50, 0.75)	[0.25, 0.56)	[0.71, 0.87)
$[750, 1000)$				[0.75, 1.0)	[0.56, 1.0)	[0.87, 1.0)

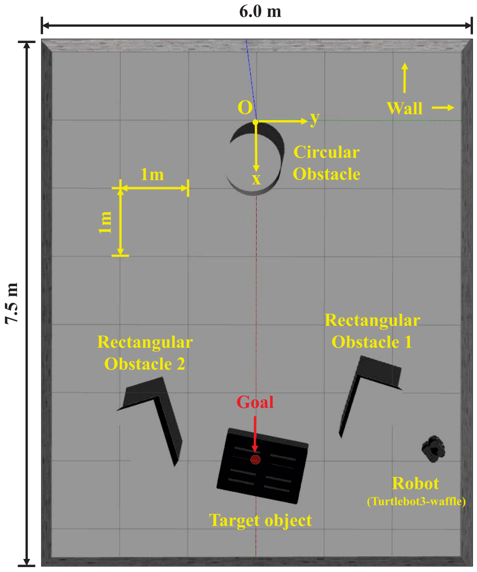

FIGURE 5. Simulation environment assuming a warehouse scenario. Two rectangular and one circular obstacles are placed in a virtual warehouse. The target object is a pallet, and the motion model of a robot follows that of a real robot (turtlebot3-waffle) in a Gazebo simulator. The total size of the environment is 6.0 m (W) × 7.5 m (H), and the sizes of each component are described in Table 2.

the total size of an environment; the forward and backward transitional velocities of a robot were 0.3 m/s and −0.3 m/s, respectively, and the left and right rotational velocities were 1.0 rad/s and -1.0 rad/s, respectively. We assumed that a robot could not push obstacles; their positions were fixed regardless of external forces. The resolution of the DLM was 5 cm.

A Gazebo simulator with Robot Operating System (ROS) was exploited to build a simulation environment [57]. An open dynamics engine (ODE) was applied to the Gazebo simulator, and thus, the motion of a robot was analogous to real experiments as we will describe in section VIII-B. For fast training, we played the Gazebo simulator 70 times faster; total training time was about 6 hours for 1000 episodes on Geforce GTX-1650 with Intel i7-9700, 3.00 Ghz.

The hyper parameters of the proposed method are as follows. The number of obstacles M_{obs} is 3, and that of the extended radii N_{er} is 4 in (10). The weight ratio ω_i is proportional to each step: $\omega_i = 1.0 \times i$ for $i \in \{1, 2, \ldots, N_{er}\}$. The size for evaluating current performance (N_c) was 30, and the unit episode size for ϵ-decay (N_{unit}) was 500. To analyze the performance according to the criterion of success probability ($\mathbf{P}_{success}$), we selected two criteria as 0.8 and 0.9, respectively. The discount factor (γ) and learning rate (α) were 0.999 and 0.001, respectively. In the Algorithm 2, the size of the replay memory ($\mathcal{D}$) was 10 million, and the maximum step size (L) was 1000. The update period of the target network was 20: $K = 20$. The batch size was 512, and the value of ϵ linearly decreased from 1.0 to 0.1.

B. SIMULATION RESULTS

We compared the proposed and manually-designed CL methods, as shown in Table 4. For testing the manually-designed CL methods in diverse environments, we introduced the distance ratio concept. The distance ratio is simply assigned by the distance between an initial position and a goal, which is different from the calculation method using the proposed DLM. For example, the distance ratio of the maximum distance from the goal is 1.0 in the environment, and that of the minimum distance from the goal is 0.0; the distance ratio of the middle is 0.5. We divided the whole region into several areas according to the distance ratio. The distance ratios are defined by three different functions: uniform, convex, and concave. Equation (12) presents diverse function types:

$$f_{partition}(n_{step}) = \begin{cases} \dfrac{n_{step}}{N_{max}}, & \textit{if uniform} \\ \left(\dfrac{n_{step}}{N_{max}}\right)^2, & \textit{if convex} \\ \sqrt{\dfrac{n_{step}}{N_{max}}}, & \textit{if concave} \end{cases} \tag{12}$$

where N_{max} is the maximum step size. This size is determined by the number of total training episodes $N_{episodes}$ and unit step size, N_{unit}: $N_{max} = \frac{N_{episodes}}{N_{unit}}$. The value n_{step} is the quotient of the current episode index i_{ep} divided by N_{unit}: $n_{step} \leftarrow (i_{ep}\ \textbf{MOD}\ N_{unit})$. Table 3 shows the distance ratio intervals according to bisection and quadrisection region

TABLE 4. Success rates when the maximum number of training episodes was 1000 except for PP2.

Curriculum	Region partition	Episode		Success / Total (success ratio)	Average travelling distance (m)		Index
		Type	Length		Robot	Pallet	
No curriculum	Random	Whole	1000	146 / 200 (0.73)	208.15	82.62	NC
Manual	Uniform	Uniform	500 / 500	136 / 200 (0.68)	128.27	44.43	UU1
			250 / 250 / 250 / 250	104 / 200 (0.52)	328.04	134.90	UU2
		Increasing	300 / 700	145 / 200 (0.73)	168.77	87.33	UI1
			50 / 150 / 300 / 500	125 / 200 (0.63)	155.62	63.41	UI2
			100 / 200 / 300 / 400	106 / 200 (0.53)	177.20	66.45	UI3
		Decreasing	700 / 300	141 / 200 (0.71)	185.27	57.67	UD1
			500 / 300 / 150 / 50	109 / 200 (0.55)	300.51	32.09	UD2
			400 / 300 / 200 / 100	125 / 200 (0.63)	226.08	48.79	UD3
	Convex	Uniform	500 / 500	105 / 200 (0.53)	159.10	53.11	CVU1
			250 / 250 / 250 / 250	123 / 200 (0.62)	318.35	58.07	CVU2
		Increasing	300 / 700	129 / 200 (0.65)	229.2	75.65	CVI1
			50 / 150 / 300 / 500	107 / 200 (0.54)	157.75	38.49	CVI2
			100 / 200 / 300 / 400	131 / 200 (0.66)	194.11	68.71	CVI3
		Decreasing	700 / 300	127 / 200 (0.64)	169.23	73.69	CVD1
			500 / 300 / 150 / 50	115 / 200 (0.58)	168.46	59.99	CVD2
			400 / 300 / 200 / 100	92 / 200 (0.46)	174.59	54.05	CVD3
	Concave	Uniform	500 / 500	100 / 200 (0.50)	222.44	80.62	CCU1
			250 / 250 / 250 / 250	92 / 200 (0.46)	317.27	88.45	CCU2
		Increasing	300 / 700	112 / 200 (0.56)	141.94	53.76	CCI1
			50 / 150 / 300 / 500	138 / 200 (0.69)	178.49	73.99	CCI2
			100 / 200 / 300 / 400	47 / 200 (0.24)	527.42	78.39	CCI3
		Decreasing	700 / 300	109 / 200 (0.55)	189.22	60.53	CCD1
			500 / 300 / 150 / 50	125 / 200 (0.63)	174.21	41.03	CCD2
			400 / 300 / 200 / 100	125 / 200 (0.63)	178.21	88.47	CCD3
Automatic (Proposed)	DLM	Adaptive ($\mathbf{P}_{success} = 0.8$)		**174** / 200 (**0.87**)	169.68	80.83	PP1
		Adaptive ($\mathbf{P}_{success} = 0.9$)		**181** / 200 (**0.91**)	160.35	89.46	PP2

partitioning. For example, in the convex region partition at quadrisection, a pallet was initialized only in a specific region according to the distance ratios from a goal for each episode: $\{[0, 0.06), [0.06, 0.25), [0.25, 0.56), [0.56, 1.0)\}$ distance ratios for $\{[0, 250), [250, 500), [500, 750), [750, 1000)\}$ episode indices, respectively. This strategy implies following an assumption; if a robot learns how to transport in a small-sized region first, then the robot can easily learn how to transport an object in a large region.

We tested various manual curricula using distance ratios, as shown in Table 4. We can derive two main conclusions from the results. First, manual curricula show unstable and inconsistent performance. In the manual curricula, the performances are affected by how to partition regions and set the number of training episodes. Among them, the number of episodes highly affected the performance. For example, the third concave-increasing combination (CCI3) is much lower than the second one (CCI2). The only difference between them was the number of episodes. Meanwhile, uniform region partitioning methods seem better than others (i.e., convex and concave) but not always; the artificial manipulation of region partitioning sometimes has an adverse effect on performance. Moreover, most cases of manual curriculum show worse performance than even the no-curriculum case. These results show the limitations of manual curriculum learning. Second, the proposed methods (i.e., PP1 and PP2) showed better performance than manual curricula and no-curriculum cases. We considered the difficulty level of object transportation for the automatic curriculum. Critical factors affecting performance are automatically designed in the proposed methods. We restricted the number of total training episodes to 1000 by defining $\mathbf{P}_{success} = 0.8$ (PP1) for fair comparison with other cases; the that of all other cases was 1000. If we lift the restricted number of training episodes, the proposed method (PP2) showed a better success ratio: 0.91. In this case, the number of total episode was 2000.

Fig. 6 shows the average reward curves of each method. The reward of the proposed method gradually increased during the training process although it was not the highest. The learning curves of manual curricula were various because the numbers of training episodes differed for each design. For example, the average reward of the second convex-decreasing curriculum (CVD2) increased and decreased at sharp points such as (500, 800, 950, 1000) episodes. The second proposed method (PP2) was not illustrated in the figure because the total episode length of the PP2 was 2000.

VIII. PRACTICAL EXPERIMENTS

A. EXPERIMENTAL ENVIRONMENT

We constituted a practical environment for verifying proposed method, as shown in Fig. 8 and 9. We tested the proposed object transportation method using a DQN which was already learned in the Gazebo simulator. Fig. 7 shows the picture of a robot, an object, and obstacles in the practical experiment. A turtlebot3-waffle robot was used for the experiment because we assumed this robot as a training model in the simulator. The forward and backward transitional velocities are 0.15 m/s and -0.15 m/s, respectively, and the left and right rotational velocities are -0.2 rad/s and 0.2 rad/s, respectively. We set the velocities of a robot as low compared with the

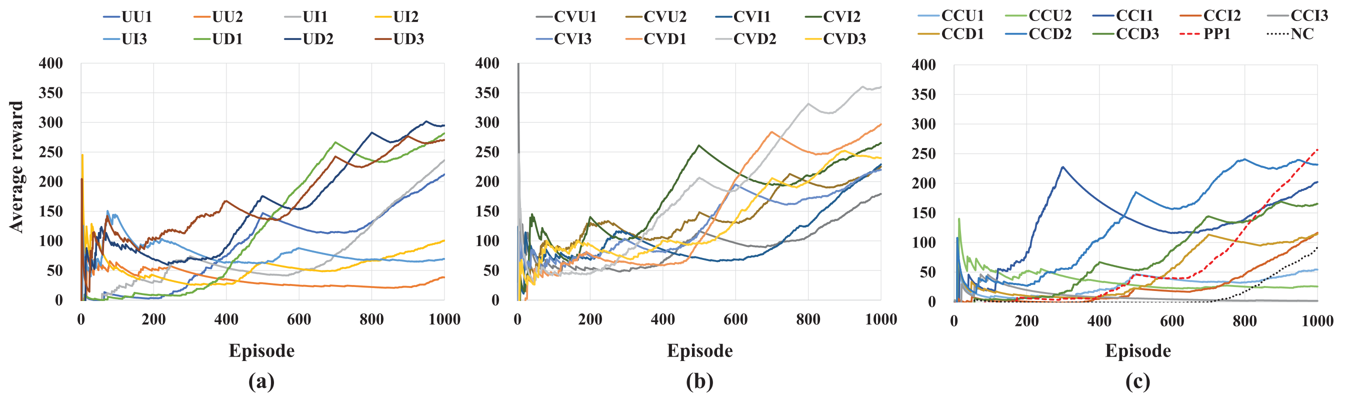

FIGURE 6. Average reward of each method in Table 4. (a) Average reward graphs using the manually-designed curricula with uniform region partitioning. (b) Average rewards using manual curricula with convex region partitioning. (c) Average rewards of the manually-designed curricula with concave region partitioning. Average rewards of no-curriculum and the proposed methods are also drawn.

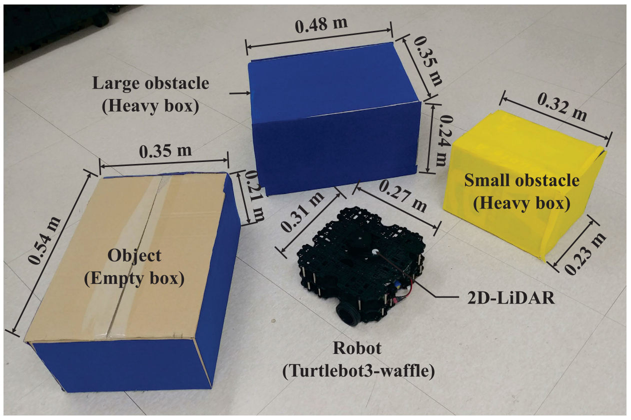

FIGURE 7. A robot (turtlebot3-waffle), an object (empty box), and two large and small obstacles (heavy boxes) were used in practical experiments. The robot could push the object because of the light weight of the object. In contrast, the robot could not push any obstacles because they were heavy.

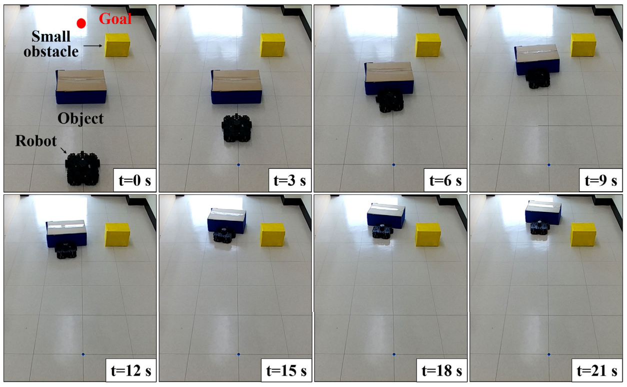

FIGURE 8. Object transportation process in a one-obstacle environment. A robot approached a target object straightly for the first 6 seconds. The robot pushed the object to the left because there was a small obstacle on the right side (t = 6–12 s). The robot pushed the object toward a goal on the right after the robot avoided the obstacle (t = 12–18 s). Finally, the robot succeeded in transporting the object to the goal (t = 21 s).

simulation due to the restrictions of the environmental size. The total environmental size is 2.3 m (W) × 4.0 m (H). The size of the object is 0.48 m (W) × 0.36 m (H) and that of obstacles is 0.54 m (W) × 0.35 m (H) (large) and 0.32 m (W) × 0.23 m (H) (small), respectively. The positions of an object and obstacles are detected by 2D-LiDAR equipped in the turtlebot3-waffle. The main objective of practical experiments is to verify how similar the performances of object transportation between simulation and real environments are.

B. RESULTS

We conducted two experiments in an environment where there were one or two obstacles. Fig. 8 shows the object transportation process in a one-obstacle environment. Initially, a robot approached an object with a straight line (0–6 s). The robot began to push the object when the robot arrived in the position of the object (6 s). The robot pushed an object toward the left direction to avoid an obstacle (i.e., yellow obstacle to the right of the robot) (6–12 s). Once the robot passes the obstacle, then it pushes the object toward the direction of the goal (12–18 s). Finally, the object arrived at the desired goal at 21 s. This process was analogous to the Gazebo simulation result, and the success rate of object transportation was 60% (6 out of 10).

The result in the two-obstacle environment was analogous to that in the one-obstacle case, as shown in Fig. 9. A robot approached an object at first (0–4 s). Then, the robot pushed the object toward a narrow passage between obstacles for transportation (4–10 s). Then, the robot moved backward to detect obstacles and an object, and readjusted its pushing direction (10–22 s). After that, the robot carried pushing the object and readjusting its heading repeatedly until the object reached a goal (22–38 s). Finally, the object was transported successfully at 40 s. Iterative backward and pushing motions seem like ineffective methods, but the robot learned this method in the simulation. The success rate was 40% (4 out of 10).

The success rates of object transportation in the above two experiments were lower than those of simulation. There are three main reasons. First, the pushing power of a robot was

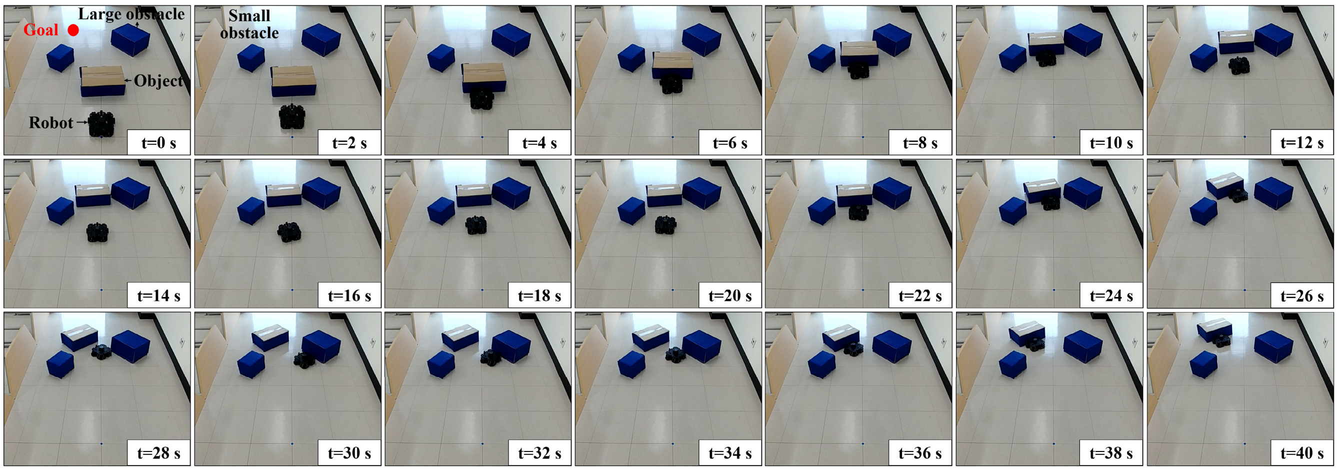

FIGURE 9. **Object transportation process in a two-obstacle environment. First, a robot approached an object for the first 4 seconds. Then, the robot pushed the object toward a narrow passage between a small and large obstacles (t = 4–10 s). The robot moved slightly backward and forward to change its heading direction when the object was located between obstacles (t = 10–22 s). After that, the robot pushed the object to the left side because the direction of the goal was left (t = 22–26 s). The robot moved backward and forward again (t = 26–36 s), and the object was successfully transported in 38 seconds. Moving motion forward and backward of the robot is the RL result of object transportation for adjusting transport's direction.**

not sufficient to push an object. In the simulation, we assumed that an object was light enough to be pushed by a single robot. However, a real robot could not push an object freely because the robot had insufficient pushing power due to hardware constraints. Second, a robot slips over a floor, especially when it pushes an object. In this case, a robot easily loses its pose, and the object cannot be controlled by the robot. Finally, there were unexpected errors due to communication delays or inaccurate positioning. In the simulation, these errors were not considered at all or these were negligible. However, these errors highly affected the performance of object transportation in the real environment.

IX. DISCUSSION

Curriculum learning is an effective training guide, and a sparse reward problem can be solved using it. Thus, many researchers have tried to adopt curriculum learning for their research fields such as collision avoidance and object transportation. However, there were several difficulties in introducing curriculum learning. First, it is difficult to design an effective curriculum for achieving a desired goal. Curriculum learning requires a manual design, which totally differs according to the type of task. For example, diverse factors affect the learning results in object transportation, such as the determination of pose-initialization region and the number of training episodes. Determining these factors properly is difficult and tedious. Second, users should design a new curriculum each time because there are no specific design rules. Based on training results, users should redesign the curriculum each time to improve performance. Finally, there is no standard curriculum learning method for object transportation. In some cases, the performance of object transportation without the curriculum showed a better result than that with the curriculum. A curriculum highly affects not only the positive but also the negative, and thus, it is difficult to design an effective hand-crafted curriculum.

To overcome the limitations of the conventional CL, an ACL method for object transportation was suggested in this paper. First, we defined the difficulty level of object transportation from the spatial point of view. The core assumption for defining the difficulty level is that the following: object transportation will be more difficult if an object should be transported for a long distance or in a complex environment where many obstacles exist. We could generate the DLM by this assumption, and the pose-initialization region of an object for each episode was automatically determined. Moreover, we adaptively adjusted the number of training episodes during the training process because it is an important factor that affects performance. This means that we determine the training procedure from the temporal point of view.

However, there are some limitations to our work. First, we considered static obstacles only. In the real environment, there are many dynamic obstacles and multiple agents (e.g., ground vehicles and UAVs [58]), and thus, we will study effective object transportation methods in dynamic environments in the future. Second, we did not consider path-planning methods for complex environments where there are many obstacles. Precise planning is sometimes needed for transporting an object to a goal in a cluttered environment. In extreme cases, there may be no feasible planning. A robot should recognize these situations for successful transportation. In this paper, we considered only the limited number of obstacles (e.g., two or three static obstacles) by reflecting their motions to the reward function. This is insufficient to be generalized to actual fields. Third, object transportation can be executed by multiple robots, which is called cooperative object transportation. An assumption should be changed if multiple robots are used.

An environment has non-stationary characteristics because robots affect each other during the training process. We should consider this characteristic to apply to multi-robot case. Finally, the action of robots can be continuous. If a robot takes an action with continuous transitional and rotational velocities, then the robot can transport an object more precisely. In this case, the proximal policy optimization (PPO) algorithm can replace the function of DQN.

X. CONCLUSION

This paper proposed an automatic curriculum design for object transportation based on deep reinforcement learning. The proposed curricula for effective training of object transportation are automatically determined from two points of views: spatial and temporal difficulty levels. From a spatial point of view, the difficulty level of object transportation grows as the predicted travelling distance of an object increases. Object transportation is also difficult in environments where many obstacles and walls are near an object. These spatial constraints affect the success rate of object transportation, and thus, we suggested the generation method of the difficulty level map for automatic curriculum design. Meanwhile, from a temporal point of view, the difficulty level of object transportation is related to the number of training episodes. It is difficult to determine how long a robot should learn to complete the current task. We introduced the criterion of success probability, and trained the robot until the current performance was satisfied with the predefined success rate. The success rate of the proposed automatic curriculum design was 14% (min)$\sim$63% (max) higher than that of the manually-designed curriculum. The proposed method can be applied to the applications of DRL-based object transportation, such as plenary exploration, warehouse, foraging, and waste retrieval.

In the future work, we will investigate more effective ACL methods in a cluttered environment where multiple obstacles exist. We expect that a robot can find and learn an effective path-finding of object transportation by helping with the previous path-planning algorithms. Moreover, cooperative object transportation based on the ACL method will be studied.

REFERENCES

[1] C. Rizzo, A. Lagraña, and D. Serrano, "GEOMOVE: Detached AGVs for cooperative transportation of large and heavy loads in the aeronautic industry," in *Proc. IEEE Int. Conf. Auton. Robot Syst. Competitions (ICARSC)*, Apr. 2020, pp. 126–133.

[2] P. S. Schenker, T. L. Huntsberger, P. Pirjanian, E. T. Baumgartner, and E. Tunstel, "Planetary rover developments supporting Mars exploration, sample return and future human-robotic colonization," *Auton. Robots*, vol. 14, no. 2, pp. 103–126, 2003.

[3] J. Hooks, M. S. Ahn, J. Yu, X. Zhang, T. Zhu, H. Chae, and D. Hong, "ALPHRED: A multi-modal operations quadruped robot for package delivery applications," *IEEE Robot. Autom. Lett.*, vol. 5, no. 4, pp. 5409–5416, Oct. 2020.

[4] I. Hanski and Y. Cambefort, *Dung Beetle Ecology*, vol. 1195. Princeton, NJ, USA: Princeton Univ. Press, 2014.

[5] O. Feinerman, I. Pinkoviezky, A. Gelblum, E. Fonio, and N. S. Gov, "The physics of cooperative transport in groups of ants," *Nature Phys.*, vol. 14, no. 7, pp. 683–693, Jul. 2018.

[6] E. Tuci, M. H. M. Alkilabi, and O. Akanyeti, "Cooperative object transport in multi-robot systems: A review of the state-of-the-art," *Frontiers Robot. AI*, vol. 5, p. 59, May 2018.

[7] M. Riedmiller, R. Hafner, T. Lampe, M. Neunert, J. Degrave, T. Wiele, V. Mnih, N. Heess, and J. T. Springenberg, "Learning by playing solving sparse reward tasks from scratch," in *Proc. Int. Conf. Mach. Learn.*, 2018, pp. 4344–4353.

[8] G. Eoh and T.-H. Park, "Cooperative object transportation using curriculum-based deep reinforcement learning," *Sensors*, vol. 21, no. 14, p. 4780, Jul. 2021.

[9] Y. Tsvetkov, M. Faruqui, W. Ling, B. MacWhinney, and C. Dyer, "Learning the curriculum with Bayesian optimization for task-specific word representation learning," 2016, *arXiv:1605.03852*. [Online]. Available: http://arxiv.org/abs/1605.03852

[10] N. Jiang, S. Jin, and C. Zhang, "Hierarchical automatic curriculum learning: Converting a sparse reward navigation task into dense reward," *Neurocomputing*, vol. 360, pp. 265–278, Sep. 2019.

[11] A. Graves, M. G. Bellemare, J. Menick, R. Munos, and K. Kavukcuoglu, "Automated curriculum learning for neural networks," in *Proc. Int. Conf. Mach. Learn.*, 2017, pp. 1311–1320.

[12] Y. Gombo, A. Tiwari, and S. Devasia, "Accelerated-gradient-based flexible-object transport with decentralized robot teams," *IEEE Robot. Autom. Lett.*, vol. 6, no. 1, pp. 151–158, Jan. 2021.

[13] H. Yang, M.-S. Kim, and D. Lee, "Distributed rotor-based vibration suppression for flexible object transport and manipulation," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, May 2020, pp. 1230–1236.

[14] G. Loianno, V. Spurny, J. Thomas, T. Baca, D. Thakur, D. Hert, R. Penicka, T. Krajnik, A. Zhou, A. Cho, M. Saska, and V. Kumar, "Localization, grasping, and transportation of magnetic objects by a team of MAVs in challenging desert-like environments," *IEEE Robot. Autom. Lett.*, vol. 3, no. 3, pp. 1576–1583, Jul. 2018.

[15] J. Alonso-Mora, S. Baker, and D. Rus, "Multi-robot formation control and object transport in dynamic environments via constrained optimization," *Int. J. Robot. Res.*, vol. 36, no. 9, pp. 1000–1021, 2017.

[16] Z. Wang and M. Schwager, "Kinematic multi-robot manipulation with no communication using force feedback," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, May 2016, pp. 427–432.

[17] M. De Berg and D. H. P. Gerrits, "Computing push plans for disk-shaped robots," *Int. J. Comput. Geometry Appl.*, vol. 23, no. 1, pp. 29–48, Feb. 2013.

[18] D. Nieuwenhuisen, A. F. van der Stappen, and M. H. Overmars, "Path planning for pushing a disk using compliance," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, Aug. 2005, pp. 714–720.

[19] J. Chen, M. Gauci, W. Li, A. Kolling, and R. Groß, "Occlusion-based cooperative transport with a swarm of miniature mobile robots," *IEEE Trans. Robot.*, vol. 31, no. 2, pp. 307–321, Apr. 2015.

[20] J. Stüber, C. Zito, and R. Stolkin, "Let's push things forward: A survey on robot pushing," *Frontiers Robot. AI*, vol. 7, p. 8, Feb. 2020.

[21] S. Krivic and J. Piater, "Pushing corridors for delivering unknown objects with a mobile robot," *Auton. Robots*, vol. 43, no. 6, pp. 1435–1452, Aug. 2019.

[22] J. Hu, P. Bhowmick, and A. Lanzon, "Group coordinated control of networked mobile robots with applications to object transportation," *IEEE Trans. Veh. Technol.*, vol. 70, no. 8, pp. 8269–8274, Aug. 2021.

[23] W. Wan, B. Shi, Z. Wang, and R. Fukui, "Multirobot object transport via robust caging," *IEEE Trans. Syst., Man, Cybern., Syst.*, vol. 50, no. 1, pp. 270–280, Jan. 2020.

[24] S. Makita and W. Wan, "A survey of robotic caging and its applications," *Adv. Robot.*, vol. 31, nos. 19–20, pp. 1071–1085, Oct. 2017.

[25] A. Rodriguez, M. T. Mason, and S. Ferry, "From caging to grasping," *Int. J. Robot. Res.*, vol. 31, no. 7, pp. 886–900, 2012.

[26] Z. Liu, H. Kamogawa, and J. Ota, "Fast grasping of unknown objects through automatic determination of the required number of mobile robots," *Adv. Robot.*, vol. 27, no. 6, pp. 445–458, Apr. 2013.

[27] S. Berman, Q. Lindsey, M. S. Sakar, V. Kumar, and S. Pratt, "Study of group food retrieval by ants as a model for multi-robot collective transport strategies," in *Proc. Robot., Sci. Syst. VI*, 2011, p. 259.

[28] B. Donald, L. Gariepy, and D. Rus, "Distributed manipulation of multiple objects using ropes," in *Proc. Millennium Conf., IEEE Int. Conf. Robot. Automat. Symp. (ICRA)*, vol. 1, Apr. 2000, pp. 450–457.

[29] A. Yamashita, T. Arai, J. Ota, and H. Asama, "Motion planning of multiple mobile robots for cooperative manipulation and transportation," *IEEE Trans. Robot. Autom.*, vol. 19, no. 2, pp. 223–237, Apr. 2003.

[30] H. Kim, H. Lee, S. Choi, Y.-K. Noh, and H. J. Kim, ''Motion planning with movement primitives for cooperative aerial transportation in obstacle environment,'' in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, May 2017, pp. 2328–2334.

[31] V. Lippiello, F. Ruggiero, B. Siciliano, and L. Villani, ''Visual grasp planning for unknown objects using a multifingered robotic hand,'' *IEEE/ASME Trans. Mechatronics*, vol. 18, no. 3, pp. 1050–1059, Jun. 2013.

[32] S. Rodriguez, M. Morales, and N. M. Amato, ''Multi-agent push behaviors for large sets of passive objects,'' in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, Oct. 2016, pp. 4437–4442.

[33] W. Wan, R. Fukui, M. Shimosaka, T. Sato, and Y. Kuniyoshi, ''Grasping by caging: A promising tool to deal with uncertainty,'' in *Proc. IEEE Int. Conf. Robot. Autom.*, May 2012, pp. 5142–5149.

[34] F. Ohashi, K. Kaminishi, J. D. F. Heredia, H. Kato, T. Ogata, T. Hara, and J. Ota, ''Realization of heavy object transportation by mobile robots using handcarts and outrigger,'' *Robomech J.*, vol. 3, no. 1, pp. 1–12, Dec. 2016.

[35] T. Yoshikawa and K. Nagai, ''Manipulating and grasping forces in manipulation by multifingered robot hands,'' *IEEE Trans. Robot. Autom.*, vol. 7, no. 1, pp. 67–77, Feb. 1991.

[36] K. M. Lynch, ''Locally controllable manipulation by stable pushing,'' *IEEE Trans. Robot. Autom.*, vol. 15, no. 2, pp. 318–327, Apr. 1999.

[37] G. A. Pereira, M. F. Campos, and V. Kumar, ''Decentralized algorithms for multi-robot manipulation via caging,'' *Int. J. Robot. Res.*, vol. 23, nos. 7–8, pp. 783–795, 2004.

[38] Y. Deng, X. Guo, Y. Wei, K. Lu, B. Fang, D. Guo, H. Liu, and F. Sun, ''Deep reinforcement learning for robotic pushing and picking in cluttered environment,'' in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, Nov. 2019, pp. 619–626.

[39] H.-Y. Yang and S.-K. Wong, ''Agent-based cooperative animation for box-manipulation using reinforcement learning,'' *Proc. ACM Comput. Graph. Interact. Techn.*, vol. 2, no. 1, pp. 1–18, Jun. 2019.

[40] M. Rahimi, S. Gibb, Y. Shen, and H. M. La, ''A comparison of various approaches to reinforcement learning algorithms for multi-robot box pushing,'' in *Proc. Int. Conf. Eng. Res. Appl.* Cham, Switzerland: Springer, 2018, pp. 16–30.

[41] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, ''Human-level control through deep reinforcement learning,'' *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[42] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, ''Mastering the game of go with deep neural networks and tree search,'' *Nature*, vol. 529, no. 7587, pp. 484–489, 2016.

[43] L. Zhang, Y. Sun, A. Barth, and O. Ma, ''Decentralized control of multi-robot system in cooperative object transportation using deep reinforcement learning,'' *IEEE Access*, vol. 8, pp. 184109–184119, 2020.

[44] S. Chen, G. Liu, and S. Wong, ''Generation of multiagent animation for object transportation using deep reinforcement learning and blend-trees,'' *Comput. Animation Virtual Worlds*, vol. 32, nos. 3–4, p. e2017, Jun. 2021.

[45] K. Shibata, T. Jimbo, and T. Matsubara, ''Deep reinforcement learning of event-triggered communication and control for multi-agent cooperative transport,'' 2021, *arXiv:2103.15260*. [Online]. Available: http://arxiv.org/abs/2103.15260

[46] J. K. Li, W. S. Lee, and D. Hsu, ''Push-Net: Deep planar pushing for objects with unknown physical properties,'' in *Proc. Robot., Sci. Syst.*, vol. 14, 2018, pp. 1–9.

[47] S. V. Manko, S. A. K. Diane, A. E. Krivoshatskiy, I. D. Margolin, and E. A. Slepynina, ''Adaptive control of a multi-robot system for transportation of large-sized objects based on reinforcement learning,'' in *Proc. IEEE Conf. Russian Young Res. Electr. Electron. Eng. (EIConRus)*, Jan. 2018, pp. 923–927.

[48] Y. Xiao, J. Hoffman, T. Xia, and C. Amato, ''Learning multi-robot decentralized macro-action-based policies via a centralized Q-Net,'' in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, May/Aug. 2020, pp. 10695–10701.

[49] C. J. C. H. Watkins and P. Dayan, ''Q-learning,'' *Mach. Learn.*, vol. 8, nos. 3–4, pp. 279–292, 1992.

[50] H. R. Tizhoosh, ''Reinforcement learning based on actions and opposite actions,'' in *Proc. Int. Conf. Artif. Intell. Mach. Learn.*, vol. 414, 2005.

[51] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, ''Playing atari with deep reinforcement learning,'' 2013, *arXiv:1312.5602*. [Online]. Available: http://arxiv.org/abs/1312.5602

[52] L.-J. Lin, ''Self-improving reactive agents based on reinforcement learning, planning and teaching,'' *Mach. learn.*, vol. 8, nos. 3–4, pp. 293–321, 1992.

[53] Y. Bengio, J. Louradour, R. Collobert, and J. Weston, ''Curriculum learning,'' in *Proc. 26th Annu. Int. Conf. Mach. Learn.*, 2009, pp. 41–48.

[54] C. Florensa, D. Held, X. Geng, and P. Abbeel, ''Automatic goal generation for reinforcement learning agents,'' in *Proc. Int. Conf. Mach. Learn.*, 2018, pp. 1515–1528.

[55] M. Tokic, ''Adaptive ε-greedy exploration in reinforcement learning based on value differences,'' in *Proc. Annu. Conf. Artif. Intell.* Berlin, Germany: Springer, 2010, pp. 203–210.

[56] F. Duchoň, A. Babinec, M. Kajan, P. Beňo, M. Florek, T. Fico, and L. Jurišica, ''Path planning with modified a star algorithm for a mobile robot,'' *Proc. Eng.*, vol. 96, pp. 59–69, Jan. 2014.

[57] K. Takaya, T. Asai, V. Kroumov, and F. Smarandache, ''Simulation environment for mobile robots testing using ROS and gazebo,'' in *Proc. 20th Int. Conf. Syst. Theory, Control Comput. (ICSTCC)*, Oct. 2016, pp. 96–101.

[58] F. Jiang, K. Wang, L. Dong, C. Pan, W. Xu, and K. Yang, ''Deep-learning-based joint resource scheduling algorithms for hybrid MEC networks,'' *IEEE Internet Things J.*, vol. 7, no. 7, pp. 6252–6265, Jul. 2020.

GYUHO EOH received the B.S., M.S., and Ph.D. degrees in electrical engineering and computer science from Seoul National University, South Korea, in 2009, 2011, and 2016, respectively.

From March 2016 to December 2020, he was a Senior Engineer at LG Electronics CTO Division (advanced robotics laboratory), where his research focused on fusion SLAM. He succeeded in commercializing various robotics products, such as a robotic vacuum cleaner, building cleaning robot, airport guide robot, and serving robot. He is currently a Research Professor at Chungbuk National University. His current research interests include deep reinforcement learning-based applications, such as multi-robot cooperation, coordination, and job shop scheduling.

TAE-HYOUNG PARK (Member, IEEE) received the B.S., M.S., and Ph.D. degrees from the Department of Control and Measurement Engineering, Seoul National University, South Korea, in 1988, 1990, and 1994, respectively. From 1994 to 1997, he worked as a Senior Researcher with the Precision Instrument Institute, Samsung Techwin Company Ltd. Since 1997, he has been a Professor with the School of Electronics Engineering, Chungbuk National University. From 2000 to 2001, he was a Visiting Scholar with the University of Toronto. His main research interests include electronic assembly and testing systems, robot motion planning, and optimization algorithms.