Intelligent Service Robotics (2021) 14:549–561
https://doi.org/10.1007/s11370-021-00377-4

ORIGINAL RESEARCH PAPER

Obstacle rearrangement for robotic manipulation in clutter using a deep Q-network

Sanghun Cheong[1] · Brian Y. Cho[2] · Jinhwi Lee[3] · Jeongho Lee[3] · Dong Hwan Kim[3] · Changjoo Nam[4] · Chang-hwan Kim[3] · Sung-kee Park[3]

Received: 4 December 2019 / Accepted: 8 July 2021 / Published online: 11 August 2021

Abstract
We propose a learning-based method to solve the problem of rearranging objects in clutter to obtain a collision-free path for retrieving a target object by a robotic manipulator. The method provides the solutions for *what obstacles to remove* and *where to place them* in what orders to rearrange the obstacles. The proposed method uses a deep Q-network to learn the optimal policies for robot's rearranging actions. To apply the network, it is assumed that the configurations of objects and environment are known and the environment is considered as a grid space. Two types of structures for a deep Q-network are proposed according to action characteristics for this problem. From extensive simulation experiments, we showed that our algorithm could reduce the number of rearranged obstacles and the total execution time significantly (up to 35%) compared to a baseline method. The experiments were performed by a real robot with a vision system and showed the feasibility of the proposed method on real world.

Keywords Object rearrangement · Manipulation planning · Deep Q-network

1 Introduction

A robot has been requested to provide useful services in human environments, which may not be suitably structured to facilitate the perception and functional capabilities of robot. A robot often needs to manipulate objects in such human environments, where many objects block a target to manipulate. The clutter of objects may make it difficult for the robot to perform perception, task and motion planning, control and execution.

In cluttered environments where multiple obstacles surround a target object, rearranging obstacles is necessary to secure a collision-free path for the grasping of a robotic manipulator. Planning for grasping the target should solve such two problems as *what to rearrange* and *where to place it* in *what order*. In some configurations, what to rearrange may mainly be considered when there is an empty space enough for placing the obstacles to be relocated. In other configurations, the robot may have to manipulate the objects only within given bounded workspace such that where to place the objects becomes important, too. This kind of problem is known to be NP-hard even in a simple case where only one obstacle is movable [5]. We are going to deal with the two problems simultaneously by employing a deep learning skill.

In this paper, we aim to solve the two problems, (i) selecting what to remove and (ii) determining where to place, simultaneously. We introduce the methods to rearrange objects and grasp a target in clutter, which employ a *Deep Q-Network* (DQN) [13]. Due to the intractability of the problem, conventional topological or geometrical planning approaches would take prohibitively long time to solve a moderate-sized problem. Unlike them, our approach based

✉ Chang-hwan Kim
ckim@kist.re.kr

Sanghun Cheong
welovehun91@gmail.com

Changjoo Nam
cjnam@inha.ac.kr

[1] SK Telecom, SK T-Tower, 65, Eulji-ro, Jung-gu, Seoul 04539, Korea

[2] University of Utah, 201 presidents Circle, Salt Lake city, UT 84112, USA

[3] Korea Institute of Science and Technology, 5 Hwarang-ro 14-gil, Seongbuk-gu, Seoul 02792, Republic of Korea

[4] Inha University, 100 Inharo, Michuhol-gu, Incheon 22212, Korea

Fig. 1 A dense clutter of 10 objects (the light green cuboid in the back is the target) in the 36 cm × 72 cm space marked by the red lines on the table. A subset of obstacles needs to be rearranged to retrieve the target. This is one of the instances that our method can provide a solution for

on the deep reinforcement learning technique will provide an answer to the problem quickly even in a dense clutter shown in Fig. 1, where the objects should reside in the small space marked by the red lines[1].

The following are the contributions of this work:

- We propose an algorithm determining what to remove and where to place in what order in cluttered configurations.
- We provide extensive experiments in simulated configurations and with a real robot integrated with a vision system.

2 Related work

Several planning methods have been introduced to solve the rearrangement problem of obstacles. In [20], obstacles around a target object were removed until the shortest path from a robot to the target had no obstacle on it. It was assumed that the configuration had an enough space for the robot to place the removed obstacles so that the problem of where to place obstacles was not considered. Dogar et al. [6] defined a *Negative Goal Region* (NGR), which denoted the region the end-effector of a robot should go through to grasp a target. Accordingly, all the obstacles inside the NGR should be removed to avoid collisions. However, the locations to place the obstacles were not dealt with and the configuration of objects were less cluttered. Other works [7,9,14,15] also focused on determining what to remove in what order only, while not considering where to place removed obstacles. Recently, a method considering where to place was proposed [4] but it is missing what to remove.

Several learning methods have been introduced to solve the grasping problem in a cluttered environment.

Laskey et al. [10] proposed the deep learning networks to grasp a target object surrounded by other objects. In the networks, the policies for the accessible direction of a planar robot arm to the target were trained with a hierarchy of three types of supervisory data: (1) no obstacles reaching, (2) non-expert crowd-sourced humans, and (3) a human expert. In aspect of the use of reinforcement learning, their methods are similar to ours at some points but they dealt with a grasping problem only. Instead our method considered "what to move" as well as "where to place", which may request a huge number of human demonstrations.

Bejjani et al. [3] introduced a receding horizon planner (RHP) for pushing (nonprehensile) manipulation in clutter. Deep neural network for the planner could be learned from the task instances generated by the kino-dynamic planner to solve the physics-based manipulation. The planner could make it possible for a planar manipulator to locate a target object at a given spot by pushing it and other objects, simultaneously. Their problem was all to solve a planar manipulation and push a target object, not to grasp and place it, such that their planner may be inapplicable to our problem (what to pick and where to place it).

Bejjani et al. [2] extended the reinforcement learning guided receding horizon planner (RHP) to such variant features of objects as types, shapes, and number of objects in clutter. In addition to non-prehensile (e. g. pushing or pulling), their planner was able to perform prehensile (e.g. grasping) manipulation actions such that the robot could grasp and push (or pull) an object to a given location. Meanwhile, other objects on the path might be pushed by unexpected contacts with a robot griper. Even considering grasp actions, their planner dealt with moving a target object to a given location, not considering what to pick and where to place like ours.

Instead of using physics-based simulation data, Hasan et al. [8] used human participants demonstrations in a virtual environment of reaching a target object on a table cluttered with obstacles. Using the human demonstration data, they proposed the support vector machine (SVM) based classifiers to abstract necessary geometric information for reaching like gap, empty region, object direction etc. Their problem dealt with determining the possible directions to reach a target and move an obstacle by finding proper gaps and regions. Conceptually their approach looks similar to our method in the sense of using the geometric information (i.e. gaps or regions) but it seems to be modified to find what object to rearrange and move it to what location. Especially, it may need additional human demonstrations to search rearranging objects and locations rather than directions for moving a robot arm.

[1] We restrict the robot to manipulate the objects only inside the space although there are empty spots around the space.

The studies described above focused on searching an accessible path to a target object using deep learning or machine learning with human or simulation data. Their methods could make a robot to reach the target object by pushing or grasping-and-pushing obstacles around the target object, once the target object to grasp is given and the place to relocate it is also given for certain problems. They did not consider how far or much the obstacles are pushed, that means, where the obstacles were rearranged. On the other hand, our problem tried to obtain the rearranging objects and their new places without collisions with other objects.

3 Background: deep Q-network

Our goal is to train a rearrangement problem-solving model (so-called agent) using the *Deep Q-Network* (DQN) method proposed by Mnih et al. [13]. It is known that the DQN method uses a deep neural network to estimate a Q-function accurately even if the size of a problem is large. Before applying the DQN method, we follow a *Markov Decision Process* (MDP) to define a problem. An MDP is a decision-making process of an agent, where the actions chosen by the agent change the environment. An MDP is represented as a tuple $\langle S, A, R, T, \gamma \rangle$, where S denotes a finite set of states, A denotes a finite set of actions, R is a reward function, T is a state-transition function, and γ is a discount factor.

Watkins et al. [21] developed *Q-learning* to enable agents (given through MDP) to act optimally against environments. They define the Q-function at time t as an expected result when an agent takes an action a in a given state s as follows,

$$Q(s, a) = \mathbb{E}\,[R_t \mid S_{\mathrm{t}} = s, A_{\mathrm{t}} = a]. \tag{1}$$

In Q-learning, a policy $\pi(s)$ specifies an action chosen at a state s by an agent. The optimal policy $\pi^*(s)$ for the agent maximizes a cumulative expected reward function (Q-function) for a given finite state s and every possible finite action a. The optimal policy is then obtained by maximizing the cumulative reward as

$$\pi^*(s) = \underset{a}{\operatorname{argmax}}\ Q(s, a). \tag{2}$$

DQN proposed by Mnih et al. [13] uses two techniques: experience replay and target network. The technique of experience replay stores sample transitions, i.e. experiences of an agent $((s, a, r, s')$ through a number of episodes. s' denotes the state updated by applying the action a into the state s. After each episode, random batches from the stored experience are used to update the network. In the learning iterations, i.e. while updating the network, DQN minimizes a loss function $L(\theta)$ to update the network parameters θ, which is defined as the difference between the predicted Q-values and the target Q-values as

$$L(\theta) = \mathbb{E}\left[\left(r + \gamma \max_{a'} Q(s', a'; \theta^-) - Q(s, a; \theta)\right)^2\right], \tag{3}$$

where θ^- and θ are the parameters of neural networks. The target values of $r + \gamma \max_{a'} Q(s', a'; \theta^-)$ are used from some previous iterations. After updating all the network parameters through the learning, the optimal policy is given as

$$\pi^*(s) = \underset{a}{\operatorname{argmax}}\ Q(s, a; \theta). \tag{4}$$

4 Problem description: state and action

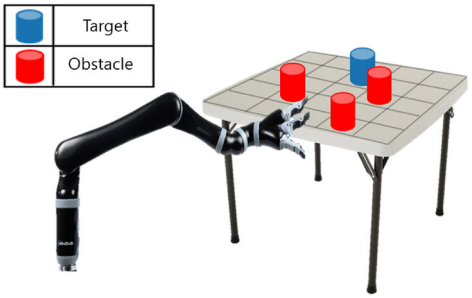
We consider the problem of rearranging obstacles in a cluttered environment as seen in Fig. 1 to retrieve a target object. In this environment, we aim to secure a collision-free path to the target by rearranging obstacles under the minimum number of manipulation actions. As we have observed that the number of rearrangements (i.e., the number of actions performed for rearranging obstacles) has a significant impact on the overall execution time, the objective is to minimize the number of rearrangements so that the execution time of manipulation will be expected to be minimized.

In the early stage of development, we assume that the configurations of objects and environment are known. We consider an environment with Z obstacles and one target object as seen in Fig. 2. In the figure, the environment (actually a table) is discretized into G grids. Although the table is evenly divided into the G grids, the grid cells are not necessarily symmetric and identical. In addition, the entire table may not be discretized into a grid space so that the part of table can be considered if the workspace of a robot is less than the table. We define a grid space with two simple rules: First, the size of a grid should be bigger than that of the largest object in the workspace. Second, one grid should have only one object on it.

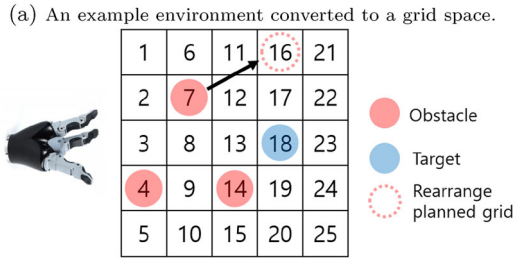
The state s represents an environment with G grids including objects (obstacles and a target) in it so that s is given in a $G \times 1$ vector. A rearrangement action a consists of two parts: a_p is the action of picking the pth obstacle and a_l the action of placing that obstacle at the lth grid as

$$a = [\, a_p, a_l \,]. \tag{5}$$

We assume a deterministic environment meaning that all transition probabilities are 1 in the entire work.



(a) An example environment converted to a grid space.



(b) The grid space (G=25) with grid IDs (1-25) and the objects on the grids. The red circles are the obstacles and the blue circle is the target.

Fig. 2 Discretization of an environment

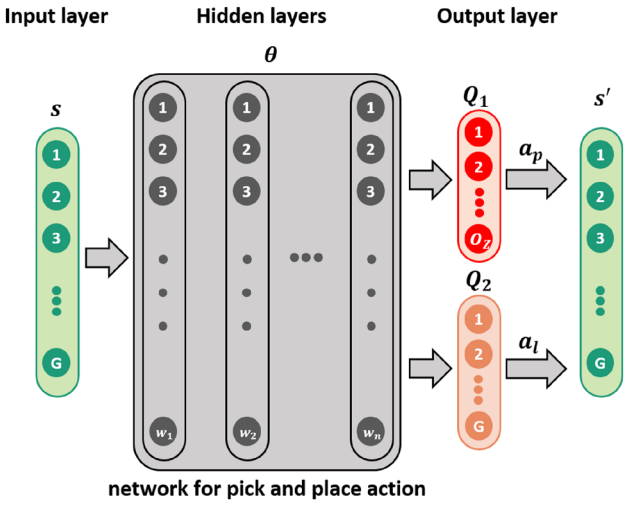


Fig. 3 An illustration of the generalized single DQN network model. We use the vanilla DQN method for training the network. The input is the state s describing the environment (the locations of objects in a grid space). The output is the action $a = [\, a_p, a_l \,]$ that a robot performs. From the output layer Q_1 (upper in red), we select the action for picking a_p through Eq. (8). From output layer Q_2 (lower in pink), we select the action for placing a_l through Eq. (9). According to a_p and a_l, the agent makes a transition from s to the next state s'

5 DORE: DQN-based obstacle rearrangement algorithm

In this section, we proposed two methods for training the DQN-based models: single DQN and sequentially separated DQN. For each method, we described the architectures of the neural network and the elements of the MDP. Then, we propose the DQN-based Obstacle Rearrangement algorithm (DORE) to solve an obstacle rearrangement problem.

5.1 Single DQN

The input of model is an environment represented in a single state, which includes the grids, obstacles and a target in that state. The output is the actions that the agent will perform. The environment consists of Z obstacles, $O = \{o_1, o_2, ..., o_Z\}$, one target object $T = \{o_t\}$, $G \in \mathbb{R}^+$ grids for a table, and the position information of obstacles and target as shown in Fig. 2b. When the jth element in O has a value i, i.e. $o_j = i$, it denotes that the jth obstacle is located at the ith grid. Similarly, $o_t = k$ in T means that the target stands at the kth grid. The state s is represented by $G \times 1$ vector as

$$s = \{g_1, g_2, ..., g_G\}. \tag{6}$$

The elements (for i=1 to G) of s are given as

$$g_i = \begin{cases} 0 \text{ if the } i\text{th grid is empty,} \\ 1 \text{ if the } i\text{th grid is occupied by an obstacle,} \\ 2 \text{ if the } i\text{th grid is occupied by the target.} \end{cases} \tag{7}$$

For example, Fig. 2a shows a table of 5 by 5 grids, three obstacles (in red), and one target (in blue). The three obstacles are located at the 4th, 7th, 14th grid and the target is at the 18th. For this example, the state s is then given as s = {0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 2, 0, 0, 0, 0, 0, 0, 0 }.

Based on the states and actions in Eqs. (5)–(7), we construct a network shown in Fig. 3 using TensorFlow interface [1]. The network consists of three types of layers: an input layer, hidden layers, and two output layers. The network receives a state s vector representing the occupancy of the grids (7). The hidden layers enable the calculation of the optimal action-value function through node updates. We have two output layers to obtain $a = [\, a_p, a_l \,]$. Output layer 1, Q_1, is a $Z \times 1$ vector and contains the Q-values for determining the picking action a_p. Output layer 2, Q_2, is a $G \times 1$ vector and contains the Q-values for determining the placing action a_l. Thus, the two output layers produce the action for determining what to pick and where to place at the same time. The action $a = [\, a_p, a_l \,]$ is obtained from the network through

$$a_p = \underset{a}{\text{argmax}}\ Q_1(s, a; \theta) \quad \text{and} \tag{8}$$

$$a_l = \underset{a}{\text{argmax}}\ Q_2(s, a; \theta). \tag{9}$$

Suppose we have a state shown in Fig. 2a. The environment has $O = \{4, 7, 14\}$ with $T = \{18\}$. If the network gives $a_p = 2$ and $a_l = 16$, the second obstacle in O (i.e. the obstacle at grid 7) is chosen to pick and grid 16 is chosen to place that obstacle as shown in Fig. 2b.

In each learning iteration, the reward r is determined according to the next state s' that is obtained by applying an action a to the current state s. The next state s' could yield three types of rewards r: (i) positive reward $r_p \in \mathbb{R}^+$, (ii) negative reward $r_n \in \mathbb{R}^-$, and (iii) unfinished reward $r_u \in \mathbb{R}^{\geq 0}$. Determination of these types depends on if the next state is successful in generating a collision-free path to retrieve the target. The positive and negative rewards are used as termination conditions for the iterations. The positive termination condition occurs if the agent in the state s' has a collision-free path so that the reward r becomes r_p and is stored in the experience replay as $e = (s, a, r, s')$. The negative termination condition occurs if the agent fails to execute the selected action $a = [\, a_l, a_p \,]$ so that $r = r_n$. This failure is defined as four cases: pick fail, place fail, pick and place fail, and go back fail.

- *Pick fail*: the obstacle to be picked (through a_p) is obstructed by other obstacles.
- *Place fail*: the grid for placing an object (through a_l) is obstructed by other obstacles.
- Pick and Place fail: the agent fails to do both a_p and a_l.
- *Go back fail*: s' is the same with one of the previous states (duplicated visits). No action is performed.

Note that obstructions and feasibility of collision-free paths are checked using the modified VFH+ as mentioned in "Appendix A". If s' does not meet any of the termination conditions, learning process will continue further (no termination of the episode). In this case, $r = r_u$.

We consider if the model trains stably while increasing the number of grids for variable environments. We increased the number of grids to 6 by 6 ($G = 36$), 7 by 7 ($G = 49$), and 8 by 8 ($G = 64$). In each environment, we compared two cases: one case has one target and three to five obstacles and another case has one target and three to 20% of the number of grids.

As the number of grids increases over 6 by 6, the success rate of learning up to five obstacles was 95% but it gradually decreased in the cases of more than five obstacles. This may be caused by learning two different types of actions in a single DQN. To compensate for the loss of success rate, we propose another DQN structure to divide the network according to action types so that learning can be done in consideration of two different characteristics of actions, respectively. This is followed in the next section.

5.2 Sequentially separated DQN

Split deep Q-learning proposed in [17] showed that actions in different characteristics could reflect their features better when they were learned from the separate networks than from a single DQN. Similarly in our work, sequentially separated DQN is structured to learn picking and placing actions independently and sequentially through two networks. We construct the networks shown in Fig. 4 using TensorFlow interface [1].

The first network θ_p, which is for pick actions, learns *what to remove* in clutter. A pick action a_p denotes the action to pick the obstacle on the a_pth grid. According to the pick action a_p, the agent makes a transition from the state s_p for picking (so called pick state) to the next state $s_p{}'$. In the next pick state the obstacle on the a_pth grid is considered to be removed.

In the network for pick actions, we expand the state s defined for the single DQN into the pick state s_p by adding accessibility of obstacles. The pick state s_p consists of the state s and the masked pick state $m_p(s)$, which is given in $2 * G \times 1$ vector as

$$s_p = [\, s,\ m_p(s) \,], \tag{10}$$

where the state s is same as for the single DQN by Eqs. (6) and (7). The masked pick state $m_p(s)$ represents whether each grid in the state is accessible for a robot or there is no obstacle on that grid. $m_p(s)$ is then given in a $G \times 1$ vector as

$$m_p(s) = \{p_1,\ p_2,\ \ldots,\ p_G\}. \tag{11}$$

The elements (for i=1 to G) of $m_p(s)$ are given as

$$p_i = \begin{cases} 0 \text{ if } g_i \neq 1 \text{ or } AccessibleCheck(g_i) = 0 \\ 1 \text{ if } g_i = 1 \text{ and } AccessibleCheck(g_i) = 1, \end{cases} \tag{12}$$

where g_i is given in Eq. (7) and $AccessibleCheck$ function is given as

$$AccessibleCheck(g_i) = \begin{cases} 0: \text{ no collision-free path to the } i\text{th grid}, \\ 1: \text{ collision-free path to the } i\text{th grid}. \end{cases} \tag{13}$$

The value of ith element p_i is 1 if there is an obstacle on the ith grid and the robot has a collision-free path to that obstacle. p_i is 0 if there is no obstacle on the ith grid or if it has no collision-free path to an object or a target. Note that obstructions and collision-free paths are checked using the modified VFH+ in "Appendix A".

The pick action a_p is selected from the output layer Q_p in a $G \times 1$ vector as seen in Fig. 4. The output layer contains the Q-values for all the possible actions. We select the pick action a_p that maximizes the Q-value among the possible actions through

$$a_p = \underset{a}{\text{argmax}}\, \{Q_p(s_p, a; \theta_p) \times m_p(s)\}. \tag{14}$$

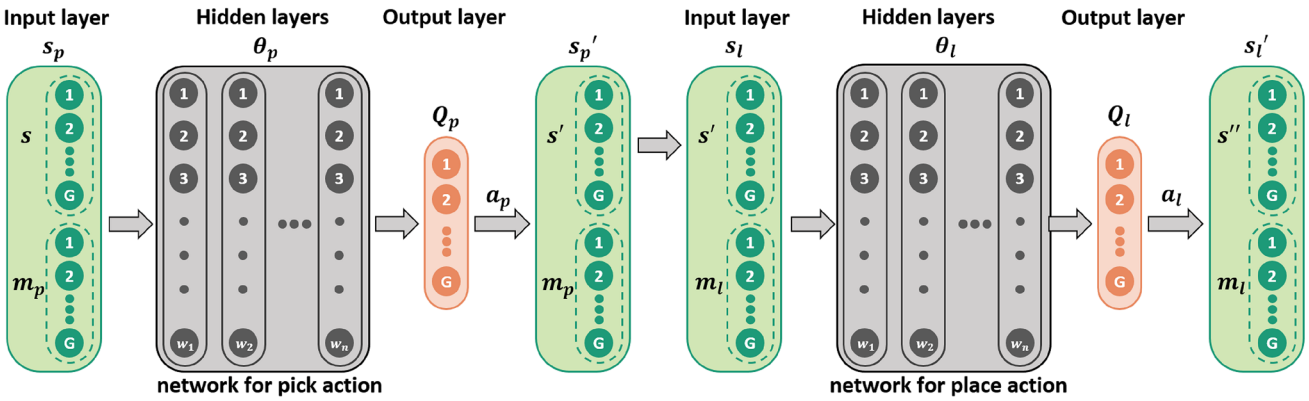


Fig. 4 An illustration of the generalized sequentially separated DQN network model. We use a vanilla DQN method for training each networks. In the pick network θ_p, the input is the pick state s_p describing the environment s and the accessibility of obstacles $m_p(s)$. From the output layer Q_p, we select the pick action a_p through Eq. (14). According to a_p, the agent makes a transition from s_p to the next state $s_p{}'$. Sequentially, the place network θ_l, the input is the place state s_l describing the environment s' and the accessibility of grids $m_l(s')$. From the output layer Q_l, we select the place action a_l through Eq. (18). According to a_l, the agent makes a transition from s_l to the next state $s_l{}'$

The next state $s_p{}'$ could yield three types of rewards r: (i) positive reward $r_p \in \mathbb{R}^+$, (ii) negative reward $r_n \in \mathbb{R}^-$, and (iii) unfinished reward $r_u \in \mathbb{R}^{\geq 0}$. Determination of these types depends on if the next state is successful in generating a collision-free path to retrieve the target. The negative reward is used as termination conditions for the iterations. The positive condition occurs if the agent in the state $s_p{}'$ has a collision-free path so that the reward r becomes r_p and is stored in the experience replay for the pick network θ_p as $e_p = (s_p, a_p, r, s_p{}')$. The negative termination condition occurs if the agent fails to execute the selected action a_p, so that $r = r_n$. If $s_p{}'$ does not meet the termination condition, learning process will continue further (no termination of the episode). In this case, $r = r_u$.

Sequentially, the second network θ_l for placing actions determines *where to place it* from the place state s_l which is the next state from the first network. Place action a_l is an action that places the obstacle on the a_lth grid. The agent makes a transition from s_l to the next place state $s_l{}'$. We redefined the state, action, and reward function for the pick network θ_p and the place network θ_l.

In the network for place actions, we expanded s into the place state s_l to include additional information of accessibility as for pick actions. The place state s_l consists of two parts, the state s' and the masked place state $m_l(s')$, and is given in a $2 * G \times 1$ vector as

$$s_l = [\, s',\ m_l(s') \,]. \tag{15}$$

The masked place state $m_l(s')$ is determined according to if each grid in the state s' is accessible and there is no object on that grid, which means that the agent can place the obstacle on that grid. The masked place state $m_l(s')$ is defined in a $G \times 1$ vector as

$$m_l(s') = \{l_1,\ l_2,\ \ldots,\ l_G\}. \tag{16}$$

The elements (for i=1 to G) of $m_l(s')$ are given as

$$l_i = \begin{cases} 0 \text{ if } g_i \neq 0 \text{ or } AccessibleCheck(g_i) = 0, \\ 1 \text{ if } g_i = 0 \text{ and } AccessibleCheck(g_i) = 1, , \end{cases} \tag{17}$$

where g_i is given in Eq. (7) and $AccessibleCheck$ function is Eq. (13). The value of ith element l_i is 1 if there is nothing on the ith grid and the robot has a collision-free path to the grid, and 0 if there is objects on the ith grid or if it has no collision-free path to the grid.

The place action a_l is selected from the output layer Q_l in a $G \times 1$ vector as seen in Fig. 4. We select the place action a_l that maximizes the Q-value among possible actions through

$$a_l = \underset{a}{\text{argmax}}\, \{Q_l(s_l, a; \theta_l) \times m_l(s')\}. \tag{18}$$

The next state $s_l{}'$ could yield three types of rewards r: (i) positive reward $r_p \in \mathbb{R}^+$, (ii) negative reward $r_n \in \mathbb{R}^-$, and (iii) unfinished reward $r_u \in \mathbb{R}^{\geq 0}$. Determination of these types depends on if the next state is successful in generating a collision-free path to retrieve the target. The positive and negative rewards are used as termination conditions for the iterations. The positive termination condition occurs if the agent in the state $s_l{}'$ has a collision-free path so that the reward r becomes r_p and is stored in the experience replay for the place network θ_l as $e_l = (s_l, a_l, r, s_l{}')$. The negative termination condition occurs if the agent fails to execute the selected action a_l, so that $r = r_n$. If $s_l{}'$ does not meet the

termination condition, learning process will continue further (no termination of the episode). In this case, $r = r_u$.

5.3 Algorithm implementation

We propose the DQN-based Obstacle Rearrangement (DORE) algorithm (described in Algorithm 1). We train a model using DQN [12,13] and check the success rates by intermediate performance tests (IPTs) during training. Once the training is done, we test the algorithm (line 3 in Algorithm 1). In case of failure, the algorithm conducts a transfer learning method (line 4–16), which will be described later in detail.

Algorithm 1 DORE

Input: Model M, training episodes E_r, test episodes E_t, target success rate S_T, training limit T_{lim}, IPT test size T_{try}, IPT test criterion T_{epo}
Output: Success or Fail
1: $M = \text{TRAINMODEL}(M, E_r, E_t, S_T, T_{\text{lim}}, T_{\text{try}}, T_{\text{epo}})$
2: Randomly choose e_u from E_t
3: Test M with e_u
* start transfer learning*\
4: **if** e_u fails with M **then**
5: $n_t = 0$
6: $e = \text{MAKEEXPERIENCEMEMORY}(e_u)$
7: **while** $n_t < T_{\text{lim}}$ **do**
8: n_t++
9: $M = \text{TRAINDQN}(M, e_u, e)$
10: Test M with e_u
11: **if** e_u succeeds with M **then**
12: **return** Success
13: **end if**
14: **end while**
15: **return** Fail
16: **else**
17: **return** Success
18: **end if**

Algorithm 2 TRAINMODEL

Input: Model M, training episodes E_r, test episodes E_t, target success rate S_T, training limit T_{lim}, IPT test size T_{try}, IPT test criterion T_{epo}, Number of IPT batches N_b
Output: A trained model M
1: $e = \text{MAKEEXPERIENCEMEMORY}(E_r)$ ..... // generate an experience memory for DQN
2: $n_t = 0$ ..... // n_t is the number of episodes used for training
3: **while** $n_t < T_{\text{lim}}$ **do**
4: $M = \text{TRAINDQN}(M, e_r, e)$ ..... // $e_r \in E_r$ is one of the training episodes
5: n_t++
6: **if** $n_t \mod T_{\text{epo}} == 0$ **then**
7: $S_{\text{counter}} = 0$
8: **for** i=1 to N_b **do**
9: $S_r = \text{IPT}(M, E_t, T_{\text{try}})$ .... // run an IPT after training every T_{epo} episodes
10: **if** $S_r \geq S_T$ **then**
11: $S_{\text{counter}}++$
12: **end if**
13: **end for**
14: **if** $S_{counter} == N_b$ **then**
15: **return** M
16: **end if**
17: **end if**
18: **end while**
19: **return** M

The model training called in line 1 of Algorithm 1 is described in Algorithm 2. For training, we create random instances to have Z obstacles and one target object in G grids. While generating the random instances, we discard those instances that do not require any rearrangement. Each of these instances is called an episode. We aggregate a number of episodes for training and test. The set of training episodes E_{r} includes 70% of the total instances. The test set E_{t} includes the rest 30%.

We implement experience replay e with E_{r} (line 1 in Algorithm 2). We train the model by two methods (single DQN and sequentially separated DQN) using e and E_{r} (line 4 in Algorithm 2) until the model reaches the target success rate S_{T} (= 95% in this work) or the number of training episodes reaches to a predetermined limit T_{lim}. The IPT measures the success rate S_{r} of the model while training is being performed (line 7 in Algorithm 2). We run an IPT every time $T_{\text{epo}} \in \mathbb{Z}^+$ number of episodes are terminated. The number of instances used for each IPT is $T_{\text{try}} \in \mathbb{Z}^+$ (we set $T_{\text{epo}} = T_{\text{try}} = 1000$ in this work).

After each IPT, we calculate the success rate,

$$S_{\text{r}} = \frac{T_{\text{try}} - F_{\text{t}}}{T_{\text{try}}} \times 100 \tag{19}$$

where F_{t} is the total number of failure cases that a batch of IPTs returns. For increasing reliability of training, we complete training if S_{r} reaches S_{T} in five consecutive IPT batches (line 8–16 in Algorithm 2).

Although the model finishes training, it does not solve any problem instance with 100% of success rate. In order to deal with the cases of failing in tests, we implement a transfer learning method [18] to learn extra episodes (line 4–16 in Algorithm 1). Transfer Learning (TL) is a technique taking additional training for a trained model. We train the model additionally using a new e generated by the problem instance that the model failed (line 6 in Algorithm 1). The algorithm finishes the additional training if the positive termination condition is met (i.e., the robot has a collision-free path to the target) or the number of training episodes exceeds T_{lim}. If TL finishes successfully, it means that the model can solve the problem instance that it had failed before running TL (line 11–13 in Algorithm 1).

6 Experiments

In this section, the single and sequentially separated DQNs were tested in various environments, where multiple objects and different sizes of workspace were considered. First, the number of rearrangements and the total execution time between the baseline method and the single DQN are compared. In addition, the planning time was compared between a graph-based search and the single DQN. Second, some sample results planned by the single DQN were validated with a real robot manipulator integrated with a vision system. Lastly, the two networks of single and sequentially separated DQNs are compared in the success rate. In the experiments, we build both the networks fully connected hidden layers, which use swish function as the activation function in both cases. The hyperparameters used in training are given in Table 5.

6.1 Experiments with single DQN in simulated environments

The single DQN was conducted in several sizes of grid environment and its results were compared with those of a graph-based search method and the baseline method.

A graph-based search is one of the well known methods to deal with the kind of our problem (i.e. it is to find the best node out of a graph consisting of actions). To implement it, we need to define a node and branching. The node denotes a state (s) representing the configuration of a grid environment. For the rearrangement problem, we define two types of branching (b_p and b_l in Fig. 5): selecting what obstacles to pick for rearranging and where to place it. The method builds a graph structure by branching from the initial node (the initial configuration) until no more branching is necessary to rearrange the obstacles. The method checks if there is a possible collision-free path to grasp or release obstacles in both branching processes through the modified VFH+ in "Appendix A". The method also checked whether the robot end-effector can reach a target object by using the modified VFH+, when a new node is created.

To compare with the graph-based search method, a 6 by 6 grid environment is given and the side length of grid is 12 cm. For training the single DQN method, we generate random episodes with three to nine obstacles (not including

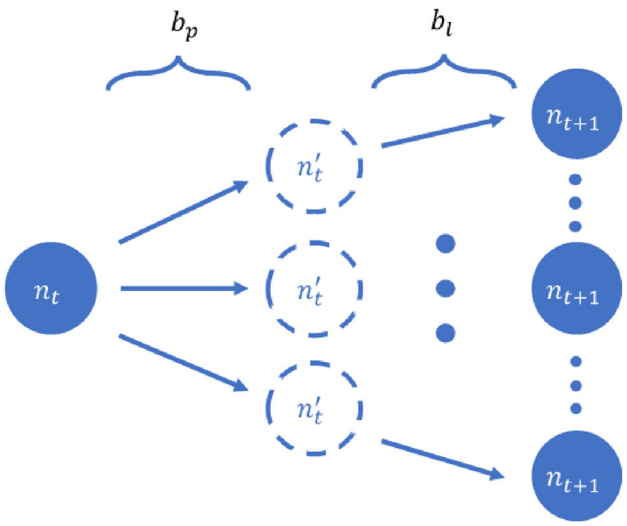


Fig. 5 A graph-based search: n_t and n_{t+1} are the nodes denoting the current and next steps, respectively. The branching, b_p, denotes the picking action (i.e. selecting a rearranging obstacle), which yields the intermediate node, n'_t. The next branching, b_l, is then made to find where to place and builds the next step

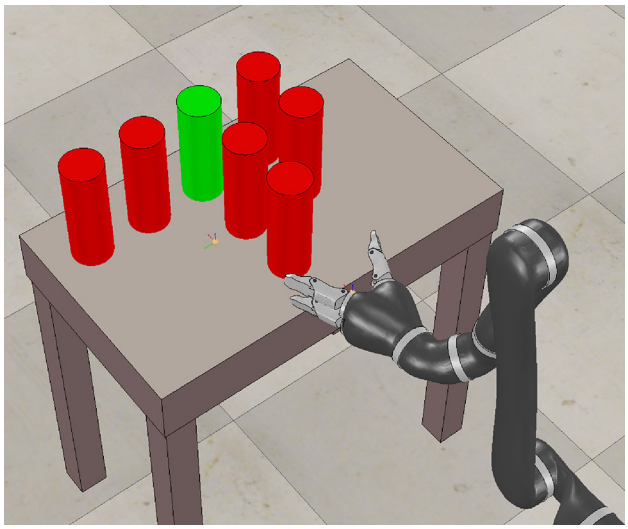


Fig. 6 The environment for a realistic simulation: We use a virtual model of Kinova Jaco1 in the simulator, V-REP. This is an example of the 3 by 6 grid environment with 6 obstacles in red and a target object in green

Table 1 The results from simulations for the 6 by 6 grid space (20 repetitions)

Metric	Method	Number of obstacles						
		3	4	5	6	7	8	9
Planning time (s)	Graph-based search	1.3 (0.35)	2.0 (0.71)	3.0 (0.71)	3.7 (0.75)	4.3 (1.75)	6.5 (2.03)	8.3 (1.78)
	Proposed (single DQN)	0.8 (0.08)	1.1 (0.41)	1.2 (0.41)	1.3 (0.40)	1.5 (0.74)	1.9 (0.80)	2.2 (0.70)

The numbers in parentheses represent standard deviations

Table 2 The results from simulated experiments for the 3 by 6 grid space (10 repetitions)

Metric	Method	Number of obstacles						
		3	4	5	6	7	8	9
Number of	Baseline	2.2 (0.78)	2.6 (0.84)	3.3 (0.67)	3.7 (0.48)	4.9 (1.19)	5.9 (0.99)	6.7 (0.82)
Rearrangements	Proposed (single DQN)	1.4 (0.51)	1.9 (0.73)	2.2 (0.63)	2.4 (0.51)	3.1 (0.99)	3.6 (0.96)	4.3 (0.94)
Total execution	Baseline	169.9 (48.17)	193.9 (50.52)	235.9 (43.84)	259.9 (30.89)	331.9 (70.40)	391.9 (56.66)	439.9 (52.98)
Time (s)	Proposed (single DQN)	120.7 (32.06)	150.7 (42.36)	168.7 (38.90)	180.7 (32.27)	222.7 (61.99)	252.7 (57.24)	294.7 (55.23)

The numbers in parentheses represent standard deviations

Table 3 The results from simulations for the 6 by 6 grid space (10 repetitions)

Metric	Method	Number of obstacles						
		3	4	5	6	7	8	9
Number of	Baseline	2.2 (0.78)	2.6 (0.84)	3.2 (0.63)	3.5 (0.52)	4.3 (0.67)	5.2 (1.03)	5.8 (0.91)
Rearrangements	Proposed (single DQN)	1.4 (0.51)	1.9 (0.73)	2.1 (0.56)	2.2 (0.42)	2.5 (0.84)	3.1 (0.73)	3.6 (0.69)

The numbers in parentheses represent standard deviations

the target, $Z = 3$ to 9) as seen in Table 1. Twenty random instances for each case of Z were tested. The planning time of the graph-based search and the single DQN method for those instances are compared in Table 1. The graph-search took longer time for planning than the single DQN method. The planning time difference between the two methods increases, as more obstacles exist. The largest difference (i.e. the single DQN method is 73% less than the graph-based search) was obtained, when $Z = 9$. This observation is expected, since the single DQN method takes much time for learning but less time for execution such that it worked faster than the graph-search. However, there is still a tread-off between both the methods. For example, the single DQN method may need to learn a new environment that is very different from the learned environments.

The baseline method developed by Dogar et al. [6] was implemented to compare the rearranging performance in clutter. The method removes all obstacles in the straight path from the end-effector to the target. The method did not find the places to put the removed obstacles but placed them in any empty space. For comparing with the single DQN method, we added a rule that an empty grid, which was not obstructed and furthest from the target, was chosen to place a picked obstacle. For checking obstruction, we used the modified VFH+ in "Appendix A".

We design the grid space considering the robot's workspace and the end-effector size. The space has 3 by 6 square cells whose side length is 12 cm. Note that the environment reflects the characteristics of the real Jaco1 robot and the same with the one shown in Fig. 7b. For training, we generate random episodes where the number of obstacles Z (not including the target) ranges from three to nine. We used the single DQN method for the training the model. We tested 10 random instances for each Z. We measured the number of rearrangements and the total execution time for the baseline and the proposed method. The total execution time includes the rearrangement planning time and the running time by a virtual robot as seen in Fig. 6. The results in Table 2 show that the single DQN method reduced both the number of rearrangements and the total execution time significantly compared to the baseline method. In average, the number of rearrangements was reduced by 34% and the total execution time was reduced by 30%. As the single DQN was trained up to $Z = 7$, the algorithm failed with some instances of $Z > 7$ (e.g., it failed seven times out of ten in average). If it fails, the DORE algorithm runs the transfer learning (TL) so that a solution can be provided. It is noticed that no TL is called for the cases of $Z \leq 7$ in our experiments. During TL, the algorithm takes extra time for additional training. TL took 143 s (standard deviation = 42.8) on average, until it solved the previously failed instance. The computing system used is composed of Intel Core i7-6700 CPU, GeForce GTX 1060 6GB/PCIe/SSE2, and 16GB RAM.

Another grid environment with 6 by 6 grid was considered as seen in Table 3. We trained the single DQN with the number of grids $G = 36$, where the number of obstacles Z ranges from three to nine. We tested 10 random instances by increasing Z from three to nine. The number of rearrangements was only compared, since the total execution time by a virtual robot is tightly dependent on and proportional to the number of rearrangements. The results in Table 3 show that the single DQN method reduced the number of rearrangements by 36% compared to the baseline method. We figured out why the baseline method planned more rearrangements and took longer execution time. First, the baseline method rearranges all the objects on the straight path from a robot end-effector to a target object, which causes unnecessary rearrangements. Second, the rearrangements planned by the baseline method

sometimes blocked some part of the empty space so that the rest of the empty space might not be reachable. As the single DQN was trained up to $Z = 7$, similarly to the case of 3 by 6 grid environment the transfer learning (TL) often ran for the instances with $Z > 7$. TL took 261 s on average (standard deviation = 51.4) using a computing system with Intel Xeon(R) Gold 6130 CPU, four GeForce GTX 2080 Ti/PCIe/SSE2, and 128GB RAM.

6.2 Experiment with a real robot manipulator

We validated the single-DQN in the DORE algorithm by using a real robot system integrated with a vision module. Both the robot and the vision module were implemented on the Robot Operating System (ROS). Kinova Jaco1, a 6-DOF manipulator fixed at the base frame, was used in the experiments. Figure 7b shows some snapshots from the experiment with seven objects. The 3 by 6 grid environment used is same as one of the simulated environments described in Sect. 6.1.

To extract the grasping information of objects, an RGB-D sensor is installed at a fixed position. The sensor is able to observe all objects in the robot workspace. The sensor first obtains the 2D coordinates of object locations using point clouds and then computes the grasping centers of objects. The Faster R-CNN in [16] was employed to obtain object labels and the bounding boxes as seen in the first snapshot of Fig. 7b.

As mentioned before, the 3 by 6 grid environment was build with seven objects including a target object. For this environment, the single-DQN planned the rearrangements of three obstacles and the collision-free path to the target object was obtained. It took 402.3 s to complete the given task of grasping the target object, which includes both of planning and execution time. From the real robot experiments, it was observed that the single-DQN could work in the real world as done in the simulated environments.

6.3 Comparison of the two training networks

In addition to the single DQN, we proposed the sequentially separated DQN structure to train the network stably for the environments with the large number of grids like 6 by 6 ($G = 36$), 7 by 7 ($G = 49$), and 8 by 8 ($G = 64$). In each of these environments, we compared two cases: One case has one target and three to five obstacles and another case has one target and three to 20% of the number of grids. We measured the success rates for the single DQN and the sequentially separated DQN methods. The results are shown in Table 4.

As the number of grids increases over 6 by 6, the success rate of learning up to five obstacles was 95% but it gradually decreased in the cases of more than five obstacles. We did not use the sequentially separated DQN network structure in simulation experiments and real robot experiments. If we had used the method, there would have been better results.

7 Conclusion

In this work, we propose a learning-based method to solve the problem of rearranging objects in clutter, which can provide a collision-free path for a robotic manipulator to grasp a target. We employs a deep Q-network to learn the optimal policy of the robot performing actions for rearranging objects. The proposed method then produces the solutions for *what obstacles to remove* and *where to place them* in what orders. Two networks, single DQN and sequentially separated DQN, are constructed by considering the characteristics of actions.

The experimental results show that our algorithm reduces the number of rearranged obstacles compared to a baseline method. Accordingly, the execution time until grasping the target object is reduced. The experiment with a real robot integrated with a vision system shows that our method works in the real world as well. In addition, comparison on the performance of two networks shows that sequentially separated DQN works better than single DQN, as more obstacles exist in more complicated clutter. From this, it is observed that structures of networks may carefully be designed according to features of actions. One of the interesting future directions is to apply the proposed method to the instances that cannot be modeled by regularly-spaced identical grids, where we would use a graph representation instead of a grid space.

Furthermore, We believe that research including the uncertainty of perception that can occur in clutter environments is also possible. There is a problem in an environment with multiple objects, where invisible space occurs due to being obscured by objects. To this end, we also plan to apply the method of expressing and learning invisible spaces in the state to future research.

A Modified VFH+

Traditionally, Vector Field Histogram (VFH+) [19] is used to find a collision-free direction of a mobile robot. VFH+ calculates a histogram representing the density of obstacle around the robot. In [11], VFH+ is modified to find a direction where the lowest number of obstacles need to be relocated. If there is a direction with zero density in the histogram, it means that the target can be transported toward that direction without making any collision (i.e., no obstacle in that direction). If only non-zero density values exist in the histogram, the direction with the lowest density is chosen since the number of obstacles to be removed is the smallest if the robot decides to clear obstacles in that direction.

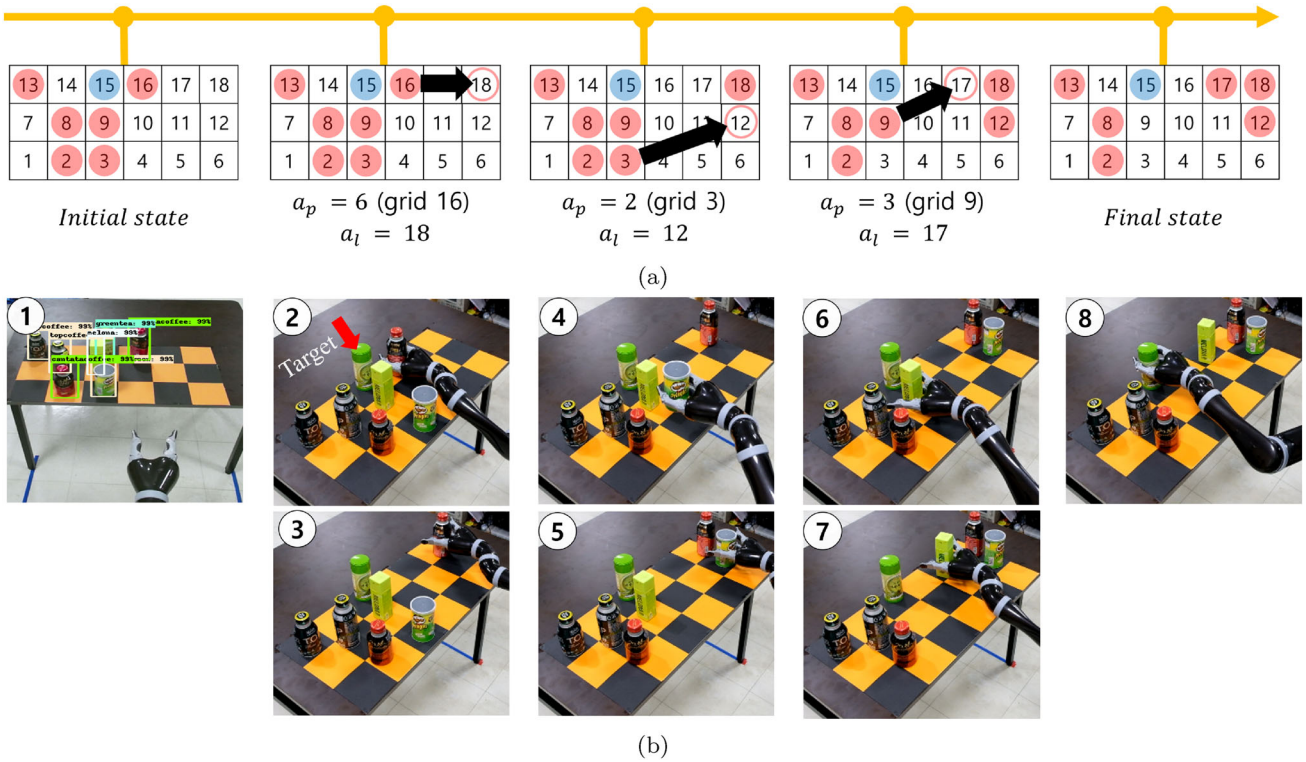


Fig. 7 **a** Rearrangement planning result for the 3 by 6 grid environment with 7 objects (reds: 6 obstacles, blue: 1 target). The resultant sequence of rearrangement planning: the obstacle on grid 16 moves to grid 18. The obstacle on grid 3 then moves to grid 12. The obstacle on grid 9 goes to grid 17 and the target on grid 15 can be finally retrieved. **b** Snapshots of the real robot experiment. The 2D bounding boxes in the first picture shows the object information found by the vision system. The rest of pictures show the rearranging movements until the robot retrieves the target object (the green cuboid in the back)

Table 4 The results from comparing the success rate with single DQN (S-DQN in the table) and sequentially separated DQN (SS-DQN in the table)

Grid space		6 by 6 ($G = 36$)		7 by 7 ($G = 49$)		8 by 8 ($G = 64$)	
No. of obstacles (obstacle density in %)		3–5 (8–14)	3–8 (8–22)	3–5 (6–10)	3–10 (6–20)	3–5 (4–8)	3–13 (4–20)
Success rate (%)	S-DQN	95	78	95	64	95	53
	SS-DQN	95	95	95	81	95	78

We use the modified VFH+ while training the model. Our proposed method finds *what to relocate* and *where to place* the obstacles. During the training, the modified VFH+ checks if the obstacle chosen to be relocated is accessible to the end-effector (i.e., graspable without collisions). If the obstacle is accessible, the modified VFH+ checks if the obstacle can be placed to the location found by our method without collisions. Depending on whether the rearranging actions are successful or not, our method determines how to reward the actions.

B Hyperparameters

Hyperparameters used in training the networks are shown in Table 5. Algorithms 1 and 2 used six hyperparameters: number of training episode, number of test episode, target success rate, training limit, intermediate performance test size, and intermediate performance test criterion. Training the networks by DQN methods [13], we used nine hyperparameters: minibatch size, experience replay size, target network update frequency, discount factor, learning rate, initial exploration, final exploration, replay start size, and number of nodes in hidden layer.

Table 5 The values of hyperparameters used for training the networks in DORE

	Single DQN		Sequentially separated DQN
	3 by 6 grid space	6 by 6 grid space	n by n grid space ($n = 6, 7, 8$)
Number of training episode ($\|E_r\|$)	119,000	595,000	70,000
Number of test episode ($\|E_t\|$)	51,000	255,000	30,000
Target success rate (S_T)	95%	95%	95%
Training limit (T_{lim})	$\|E_r\| \times 15$	$\|E_r\| \times 15$	$\|E_r\| \times 15$
Intermediate performance test size (T_{try})	1000	1000	1000
Intermediate performance test criterion (T_{epo})	1000	1000	1000
Minibatch size	32	32	32
Experience replay size	1,000,000	1,000,000	1,000,000
Target network update frequency	10,000	10,000	10,000
Discount factor	0.99	0.99	0.99
Learning rate	0.0025	0.0025	0.0025
Initial exploration	1	1	1
Final exploration	0.1	0.1	0.1
Replay start size	$\|E_r\| \times 5$	$\|E_r\| \times 5$	$\|E_r\| \times 5$
Number of nodes in hidden layer (w_1, w_2, ..., w_n)	400, 300, 300	600, 500, 400, 300	400, 300, 300
Reward set (r_p, r_u, r_n)	10, 0, −1	10, 0, −3	10, 0, −3

References

1. Abadi M, Agarwal A, Barham P, Brevdo E, Chen Z, Citro C, Corrado GS, Davis A, Dean J, Devin M, Ghemawat S, Goodfellow I, Harp A, Irving G, Isard M, Jia Y, Jozefowicz R, Kaiser L, Kudlur M, Levenberg J, Mané D, Monga R, Moore S, Murray D, Olah C, Schuster M, Shlens J, Steiner B, Sutskever I, Talwar K, Tucker P, Vanhoucke V, Vasudevan V, Viégas F, Vinyals O, Warden P, Wattenberg M, Wicke M, Yu Y, Zheng X (2015) TensorFlow: large-scale machine learning on heterogeneous systems. http://tensorflow.org/. Software available from tensorflow.org
2. Bejjani W, Dogar MR, Leonetti M (2019) Learning physics-based manipulation in clutter: combining image-based generalization and look-ahead planning. arXiv preprint arXiv:1904.02223
3. Bejjani W, Papallas R, Leonetti M, Dogar MR (2018) Planning with a receding horizon for manipulation in clutter using a learned value function. In: IEEE-RAS 18th international conference on humanoid robots (humanoids). IEEE, pp 1–9
4. Cheong SH, Cho BY, Lee J, Kim C, Nam C (2020) Where to relocate?: Object rearrangement inside cluttered and confined environments for robotic manipulation. In: Accepted for presentation at IEEE international conference on robotics and automation (ICRA)
5. Demaine ED, Demaine ML, Hoffmann M, O'Rourke J (2003) Pushing blocks is hard. Comput Geom 26(1):21–36
6. Dogar MR, Srinivasa SS (2012) A planning framework for non-prehensile manipulation under clutter and uncertainty. Auton Robots 33(3):217–236
7. Garrett CR, Lozano-Pérez T, Kaelbling LP (2015) Backward-forward search for manipulation planning. In: Proceedings of IEEE/RSJ international conference on intelligent robots and systems (IROS), pp 6366–6373
8. Hasan M, Warburton M, Agboh WC, Dogar MR, Leonetti M, Wang H, Mushtaq F, Mon-Williams M, Cohn AG (2020) Human-like planning for reaching in cluttered environments. In: IEEE international conference on robotics and automation (ICRA). IEEE, pp 7784–7790
9. Haustein J, King J, Srinivasa S, Asfour T (2015) Kinodynamic randomized rearrangement planning via dynamic transitions between statically stable states. In: Proceedings of IEEE international conference on robotics and automation (ICRA), pp 3075–3082
10. Laskey M, Lee J, Chuck C, Gealy D, Hsieh W, Pokorny FT, Dragan AD, Goldberg K (2016) Robot grasping in clutter: using a hierarchy of supervisors for learning from demonstrations. In: IEEE international conference on automation science and engineering (CASE). IEEE, pp 827–834
11. Lee J, Cho Y, Nam C, Park J, Kim C (2019) Efficient obstacle rearrangement for object manipulation tasks in cluttered environments. In: Proceedings of IEEE international conference on robotics and automation (ICRA), pp 2917–2924
12. Mnih V, Kavukcuoglu K, Silver D, Graves A, Antonoglou I, Wierstra D, Riedmiller M (2013) Playing atari with deep reinforcement learning. arXiv preprint arXiv:1312.5602
13. Mnih V, Kavukcuoglu K, Silver D, Rusu AA, Veness J, Bellemare MG, Graves A, Riedmiller M, Fidjeland AK, Ostrovski G et al (2015) Human-level control through deep reinforcement learning. Nature 518(7540):529
14. Moll M, Kavraki L, Rosell J et al (2018) Randomized physics-based motion planning for grasping in cluttered and uncertain environments. IEEE Robot Autom Lett 3:712–719
15. Nam C, Lee J, Cheong SH, Cho BY, Kim C (2020) Fast and resilient manipulation planning for target retrieval in clutter. In: Accepted for presentation at IEEE international conference on robotics and automation (ICRA)
16. Ren S, He K, Girshick R, Sun J (2015) Faster R-CNN: towards real-time object detection with region proposal networks. In: Advances in neural information processing systems, pp 91–99
17. Sarantopoulos I, Kiatos M, Doulgeri Z, Malassiotis S (2019) Split deep q-learning for robust object singulation. arXiv preprint arXiv:1909.08105
18. Shin H, Roth HR, Gao M, Lu L, Xu Z, Nogues I, Yao J, Mollura D, Summers RM (2016) Deep convolutional neural networks for computer-aided detection: CNN architectures, dataset

characteristics and transfer learning. IEEE Trans Med Imaging 35(5):1285–1298
19. Ulrich I, Borenstein J (1998) VFH+: reliable obstacle avoidance for fast mobile robots. In: Proceedings of IEEE international conference on robotics and automation (ICRA), pp 1572–1577
20. Van Den Berg J, Stilman M, Kuffner J, Lin M, Manocha D (2009) Path planning among movable obstacles: a probabilistically complete approach. In: Algorithmic foundation of robotics VIII. Springer, pp 599–614
21. Watkins CJCH (1989) Learning from delayed rewards. Ph.D. thesis, King's College, Cambridge