Combining Reinforcement Learning and Rule-based Method to Manipulate Objects in Clutter

Yiwen Chen
School of Automation Science and Engineering
South China University of Technology
Guangzhou, China
yiwen_chen97@outlook.com

Zhaojie Ju
School of Computing
University of Portsmouth
Portsmouth, UK
zhaojie.ju@port.ac.uk

Chenguang Yang*
Bristol Robotics Laboratory
University of the West of England
Bristol, UK
cyang@ieee.org

Abstract—Picking up the clustered objects is always a challenging task in robot research field. And reinforcement learning enables robot to adapt to different tasks through plenty of attempts. To reduce the complexity of strategy learning, we propose a framework for robots to pick up the objects in clutter on table based on deep reinforcement learning and rule-based method. To manipulate the objects on table, we mainly divide the robot actions into two categories: one is pushing that uses the reinforcement learning method, while the other one is grasping that is inferred by image morphological processing. The pushing action can separate the stacking objects, create a robust grasp point for the following grasp. The grasp detect algorithm determines if there is a suitable grasp point. Judging on the result of pushing, the grasp detect algorithm will return a reward for pushing learning. Taking images as input, our framework can keep a high grasp rate with low computational complexity, which makes it achieve clutter clearing quickly.

Index Terms—reinforcement learning, grasp detection, robot, clutter clearing

I. Introduction

Deep reinforcement learning plays an important role in the strategic planning of sequential actions, and has been widely used in the robotics, enabling robot to learn various skills, such as pushing [1], grasping [2] [3], inserting [4], and manipulating deformable object [5]. However, more complex the task is, more time that needs for real robot to collect data of interacting with the environment and for neural network fitting. Especially for grasping, few positive samples and diverse objects lead to the fact that hundreds of hour for collecting data is inescapable. Although sim-to-real technique can ease this problem to some extent, learning to grasp with reinforcement learning is still time-consuming and costly. Therefore, we prefer to employ other practical algorithms to control grasp. As for pushing, there are multiple solutions to separate objects in one case in general. This kind of problem is hard to define manually and doesn't require a very precise solution, hence it is suitable for reinforcement learning to deal with this problem.

Similar to our work, in [8] pushing and grasping are both learned based on reinforcement learning. They use Q-learning to choose discrete actions on pixel wise and map the pixel coordinates to the real-world location. Compared with their work, we try to employ the reinforcement learning network with continuous output to remedy this issue. Since the grasp can be performed better with other algorithms, such as supervised learning [9] or image processing method [10], we decide to use another method instead of reinforcement learning to deal with grasp, only remaining reinforcement learning for the push action.

We find that the grasp algorithm based on supervised learning is mostly trained on Cornell Grasping Dataset or Jacquard Dataset, whose depth image is strikingly different from the depth image in simulation because of different shooting angles. Therefore, we utilize a traditional morphological method in [7], which can be easily transferred to virtual image with a little change. Compared with their work, our framework applies a policy that output continuous action to avoid large action dimensions, and the accurate grasp point detection ensures high grasp rate of graspable objects. Therefore, our framework is simple in structure but competent for the clutter clearing task.

In this work, we propose to combine pushing based on reinforcement learning and grasping based on traditional rule-based grasp detect algorithm [7]. We make use of the twin delayed deep deterministic policy gradient [6] to train our policy that determines where to start pushing and pushing direction according to current image. The pushing direction within 360 degrees is divided into two sides, and we introduce a variable to decide which side pushing to. The grasp detecting is processed with rule-based method mainly based on the recognition of minimum bounding convex hull and minimum bounding rectangle of connected regions. The grasp detecting algorithm will calculate out whether it is graspable, the grasp center and the grasp orientation. When performing the task, the pushing action is executed only when no object is graspable judging by grasp detect algorithm.

The rest of this paper is organized as follows. The related work will be introduced in Section II. And Section III describes each part of our architecture in detail. We show our experiment setting and results in Section IV. Finally, we draw the conclusion in Section V.

This work was partially supported by Engineering and Physical Sciences Research Council (EPSRC) under Grant EP/S001913
*Corrsponding author is C. Yang. Email: cyang@ieee.org

II. Related Work

Deep reinforcement learning has become popular ever since the study in the game of Go [11] and the video game [12]. The reinforcement learning algorithm can be mainly grouped into two categories depending on whether the action is continuous [13] or discrete [12]. As the algorithm continues to improve, there are more and more researches on the application of algorithms in the field of robotics. Specially the learning of push or grasp makes robot to be more intelligent and skillful. Yuan et al. learn the nonprehensile rearrangement based on deep Q-learning [1], pushing an object to the predefined goal pose in an environment with obstacles. Nair et al. utilize variational auto-encoder to encode the input image [14], calculate the reward based on the Euclidean distance of encoded vector, and verify this algorithm in the experiment of reaching and pushing. Liang et al. learn to slice the flat object to the wall to enable a grasp from the side [15].

Most application of reinforcement learning in robot is a low level control that needs long sequence to achieve the goal. The large-scale exploration space and delayed reward makes it hard to get training data of high quality, and thus lots of time is needed to collect data. As presented in [2] [3] [16] [17] , it needs more than 100k grasps to learn the grasp skill without camera calibration. This amount of data requires multiple robots to execute over a long period of time, which is costly for most of us and hard to transfer the skill to different robots.

As for deep supervised learning to grasp, Chu et al. [18] come up with a grasp proposal based on the faster r-cnn network by transferring the grasp rectangle detection to object detection, and result in high classification performance. In [19] they achieve pixel-wise grasp rectangle detection by using the fully convolutional network like U-net to predict rectangle for every pixel. Without fully connected layers, their network is significantly smaller than other networks.

In the face of cleaning clustered objects that needs to combine pushing and grasping, we are inspired by the algorithm that maps the image to the high-level actions instead of continuous actions of low level based on the mapping relation between image and workspace [8] [20]. One-to-one correspondence between discrete actions and pixels has the ability to make precise decision, but leads to large network and long infer time.

III. Pushing and Grasping

A. Pushing

We consider the task of pushing as a Markov decision: given a state s_t, the agent decides an action a_t to perform, and reach a new state s_{t+1} while getting a reward $R\left(s_t, s_{t+1}\right)$. We employ the Twined Delayed DDPG to learn the policy, which consists of one policy network, double critic networks and their own target networks. We can map the state to action with policy network π_ϕ:

$$a_t = \pi_\phi(s_t)$$

The Q value is calculated by critic network Q_{θ_i}, whose loss function is as follows:

$$\text{loss} = \left(R\left(s_t, s_{t+1}\right) + \gamma \min_{i=1,2} Q_{\theta_i'}\left(s_{t+1}, \tilde{a}\right) - Q_{\theta_i}(s_t, a_t)\right)^2$$

where $\tilde{a}$ is the action decided by target network at state s_{t+1}, $Q_{\theta_i'}$ is target critic network, and γ is a discount factor. The policy network is updated with following gradient:

$$\nabla_\phi J(\phi) = \nabla_a Q_{\theta_1}(s, a)|_{a=\pi_\phi(s)} \nabla_\phi \pi_\phi(s)$$

In this work, only depth image is used as the state that is captured by the camera over the table. The pixel plane is parallel to table surface so that pixel coordinate and table planimetric position are linearly proportional. The policy network outputs action with four dimensions (a_1, a_2, a_3, a_4) and each dimension is limited to $(-1, 1)$. They present x and y coordinate of the table surface, which side pushing to and the pushing angle, respectively. Specifically, (a_1, a_2) decides the position where to start pushing, and (a_3, a_4) decides the pushing orientation.

To avoid pushing objects out of table, we limit the length of area that can start push to 0.6 times the length of table surface. The x and y coordinate is linear to this push working range. For example, $(1, 1)$ means the top left of this area, and $(0, 0)$ means the center of the table. Although cosine-sine encoder is widely used in supervised learning [19] to represent the angle at the circumference, we found it hard to master the many-to-one mapping for reinforcement learning in the absence of direct oversight of the target. Therefore, we divide the angle into left side and right side, each with 180 degrees. If $a_3 < 0$, the robot will push its end-effector to the left side. Otherwise, it would move to the right side. The a_4 from -1 to 1 is linear mapping of 180 degrees. Given the (a_1, a_2, a_3, a_4), the robot executes the pushing action following three steps:

- The robot end-effector reaches the position that is 30cm over the pushing start point decided by (a_1, a_2).
- The robot end-effector moves straight down until it contacts with objects or it is 1.5cm above the table surface.
- The robot end-effector pushes a constant distance in a given orientation decided by (a_3, a_4).

Our reward function is simple for reinforcement learning. If a grasp can be performed after the push action, the reward $R\left(s_t, s_{t+1}\right) = 1$. If the push action results in enough change of the clustered object positions which can be judged by calculating the difference between depth images before and after pushing, the reward $R\left(s_t, s_{t+1}\right) = 0.5$. In other case the reward is zero.

The network structure is shown in Fig.3. Both the policy network and critic network have the same convolutional

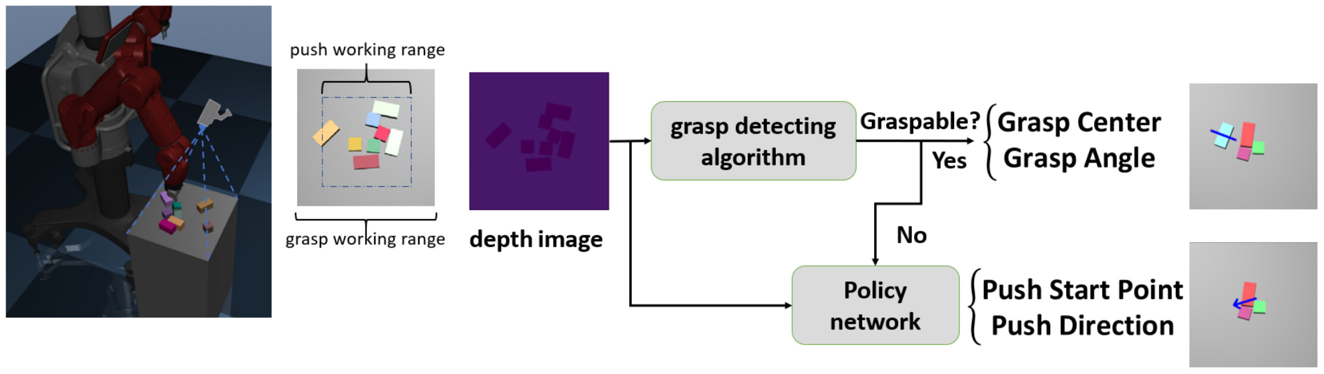

Fig. 1. The overview of our proposed system. A Baxter robot with parallel grippers is used for experiment. In the last two picture, a blue line means a grasp that will be executed with the same angle and the same center. An arrows means a push that will be executed with the same start point and the same direction. We also use this kind of method to present push and grasp in the rest of the paper.

layers to extract image feature. We prefer to add SeLU activation after the fully-connected layer. Before concatenating state and action feature in critic network, they are processed with batch normalization to guarantee the balance of gradient back propagation.

B. Grasping

The grasp detection can be defined as a problem to find the best grasp rectangle based on image [21], and convert it into the real robot grasp pose by eye to hand calibration. The grasp rectangle g is formulated as follows:

$$g = \{\mathrm{x}, \mathrm{y}, \theta, \mathrm{h}, \mathrm{w}\}$$

where (x, y) is the grasp center location, θ is grasp angle, w is open width of the gripper and h is the grasp thickness. Since our robot is equipped with stick-shaped parallel gripper, the h is ignored in this work. Therefore, we recognize the grasp line instead of grasp rectangle.

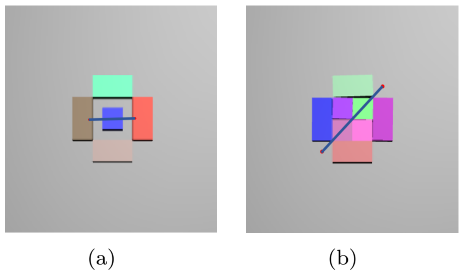

Fig. 2. Unsuccessful grasp detection that may be caused by original grasp algorithm:(a) a grasp hindered by nearby objects; (b) a grasp with overlong grasp width

Our grasping algorithm is based on the method in [7], which utilizes minimum bounding convex hull and minimum bounding rectangle of connected regions. The original algorithm is mainly applied to grasping single object with blank desktop background, and we optimized it for multiple objects grasping.

We start by making a binary image to separate the objects in the picture from the background based on depth image. Due to the ideal simulation environment, the pixel intensity of objects in an image is always greater than that of desktop background. It is simple to make binary processing with a fixed threshold. Then, we detect a grasp configuration for every connected region in binary image and make up a grasp list. Every element in this grasp list is a grasp configuration (x, y, θ, w).

This original method deals with each region separately with no consideration of nearby objects, which may result in a grasp failure because the gripper can't move to a position low enough to clamp the object(see Fig.2). Therefore, we estimate whether a grasp is valid by comparing the pixel intensity of grasp line endpoints and that of grasp center point, which is illustrated as follows:

$$is\_valid = \begin{cases} True, & I(center\ point) < I(endpoint) - \tau \\ False, & otherwise \end{cases}$$

where τ is the threshold that ensures a stable grasp.

On the other hand, this algorithm may treat multiple connected objects as one object, and detect a grasp with overlong grasp width. To deal with this problem, we find out a grasp with the shortest grasp width in the grasp list, meanwhile it needs to be smaller than a threshold δ decided by the constant width that robot gripper can open. The grasp detection process is shown in Algorithm.1. Therefore, this grasp algorithm can figure out whether it is graspable and return the best grasp configuration if graspable.

IV. Experiment

As shown in Fig.1, the pushing action is performed only when no object is graspable, which means there may be multiple grasps between two pushes. To improve the action efficiency, the grasp is stopped after two continuous grasp failures. We perform the experiment in a simulation environment called MuJoCo. The module is built with a toolkit called robosuite [22], which contains a modularized design of APIs for building new environments. We set the

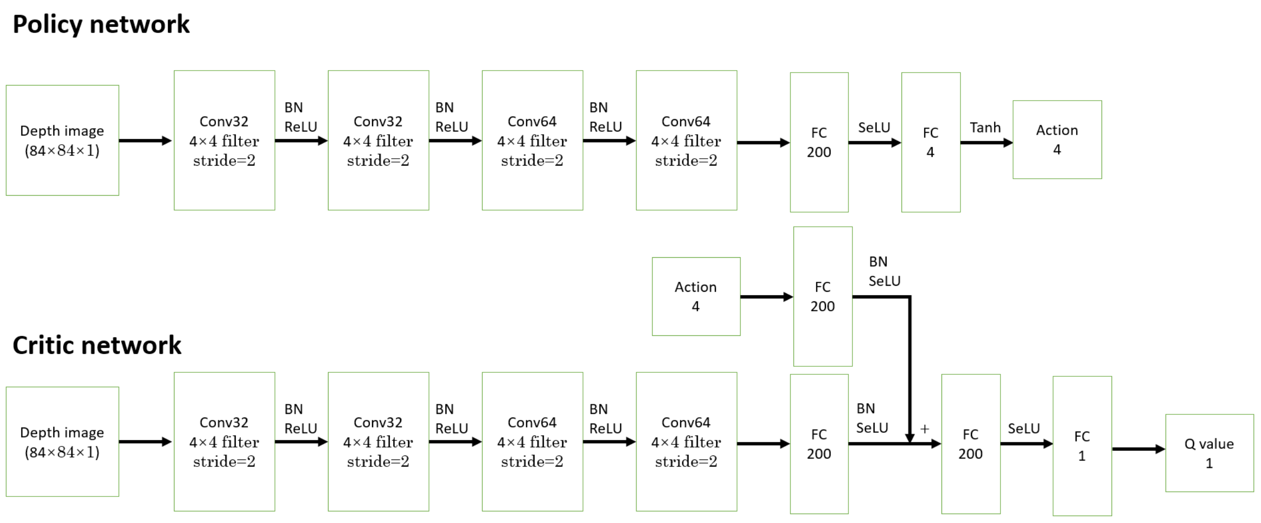
Policy network
Depth image (84×84×1)
Conv32 4×4 filter stride=2
BN ReLU
Conv32 4×4 filter stride=2
BN ReLU
Conv64 4×4 filter stride=2
BN ReLU
Conv64 4×4 filter stride=2
FC 200
SeLU
FC 4
Tanh
Action 4
Action 4
FC 200
BN SeLU
Critic network
Depth image (84×84×1)
Conv32 4×4 filter stride=2
BN ReLU
Conv32 4×4 filter stride=2
BN ReLU
Conv64 4×4 filter stride=2
BN ReLU
Conv64 4×4 filter stride=2
FC 200
BN SeLU
+
FC 200
SeLU
FC 1
Q value 1

Fig. 3. The network architecture

```
Algorithm 1 Grasp Detection Process
Input:
    a depth image
Output:
    $can_{grasp}$, $x_{grasp}$, $y_{grasp}$, $\theta_{grasp}$, $w_{grasp}$
1: Set $can_{grasp} = False$, $x_{grasp} = 0$, $y_{grasp} = 0$, $\theta_{grasp} = 0$, $w_{grasp} = \delta$
2: Make a binary image with the depth image using fixed threshold
3: Get a grasp list based on the method in [7]
4: for $(x, y, \theta, w)$ in grasp list do
5:     if $w < w_{grasp}$ and $is\_valid = True$ then
6:         $can_{grasp} = True$, $x_{grasp} = x$, $y_{grasp} = y$, $\theta_{grasp} = \theta$, $w_{grasp} = w$
7:     end if
8: end for
```

size of table surface to 0.4m×0.4m due to the robot's limited workspace range. After a grasp or a push, the robot arm is reset to a position out of camera field. Then, the camera capture an image for the next detection.

In the training phase, the objects to be manipulated are blocks with two kinds of size: 4cm×4cm×4cm and 4cm×8cm×4cm. At the beginning of training episode, up to 8 objects are randomly dropped to table. An episode is terminated if all the objects on table are taken away or push action has been performed 15 times. Depth image of 288×288 is collected for grasp detection, and the image is resized to 84×84 as input to the reinforcement learning network while pushing. All experiment is operated on the computer with NVIDIA 2080Ti and Intel Core i7-8700.

We train our network by Adam optimizer using the learning rate of 3e-4, batch size of 128, 0.01 target network update delay. Gaussian noise is added for the purpose of exploration and target value counting. The variance Gaussian noise is 0.2 for exploration while training, 0.1 for target value counting. We remain the Gaussian noise with 0.1 variance in the evaluation to prevent repetitive pushing that makes no change to the environment. Specially, we don't add Gaussian noise on a_3 while counting target value and evaluating. We have denoted that a_3 decides which side push to by determining if it's greater than 0, and thus it isn't sensitive to specific values.

TABLE I
Grasp Rate(%) of Different Algorithms

Method	original	optimized
single object	96	–
clutter	54	84

A. Grasp algorithm verification

We test our grasp algorithm under two conditions: multiple objects on table and only one object on table. Since no push is performed, it is possible that only grasp action can't clear the multiple objects. Therefore, we reset the environment if no grasp is detected in the condition of multiple objects. We make 50 trials every time, and the grasp rate is shown in Table I. We see that the grasp rate of original algorithm is very high in face of single object but hard to deal with complex situation about multiple objects. The optimized version of this algorithm is much better in the face of clutter, which is qualified for the clutter clearing task. The main reason for grasp failure presently is that two objects next to each other are recognized as one object, and the grasp center is on where they connect. But this kind of grasp failure usually separates the connected objects, and the next grasp will be successful.

TABLE II
Comparison on Results of Different Parameters

γ	0.3	0.5	0.7	0.9
reward per step	0.54	**0.57**	0.53	0.37
completion	0.92	**0.94**	0.86	0.74

TABLE III
Comparison on Results of Different Input Types

input image	RGB	depth	RGB + depth
reward per step	0.32	0.57	**0.58**
completion	0.76	**0.94**	0.92

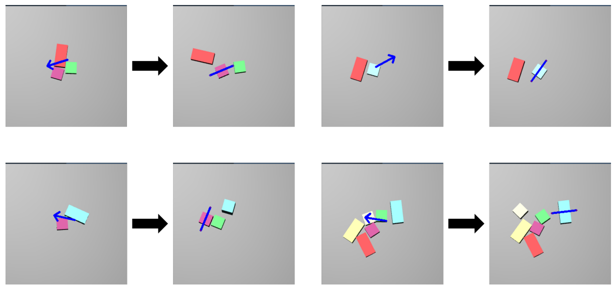
Fig. 4. The visualization of push decision and result

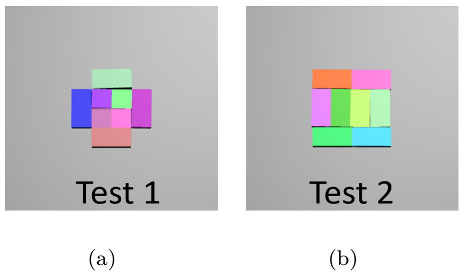

Fig. 5. The environment for testing

B. Reinforcement Learning Training

In this section, we study effects of different setting on performance. It is hard to evaluate the performance during training because the placement of objects is varied a lot in each episode. Therefore, we evaluate pushing performance for 50 episodes after 300 episodes of training. As shown in the Table II, this method can achieve a completion rate of 94% with high action efficiency. And we can see that the discount factor γ has a great impact on pushing performance. Unlike the reinforcement learning in other tasks whose γ is usually more than 0.9, the pushing task prefers to have smaller γ around 0.5. In this kind of task, the pushing action should have a positive immediate effect to help grasp, and the relationship between two pushes is little. Therefore, small γ reduces the consideration of future state and have greater performance.

We also compare the algorithm based on different inputs and the result is shown in Table III. We found that the pushing performance is poor if just RGB image is used as input. The concat of depth image and RGB image has similar performance as depth image. From the perspective of input type, depth image is the main factor affecting performance. Maybe depth image involves enough information for detecting pushing. Therefore, only depth image is used as input for reinforcement learning in this framework to accelerate speed.

We show some examples of pushing in Fig.4. We see that the pushing action can effectively separate adjacent objects, create a good grasp point and enough space for next grasp.

C. Clutter clearing

Besides evaluating in the case of random objects, we set two more testing situation by manual placement of objects. As shown in Fig.5, these objects are closely connected without any space to give a grasp. It is impossible to clear these objects without the combination of grasp and push. We ran 20 episodes in each situation, and the data is shown in Table IV. We see that the grasp rate is lower than that in the random situation. The objects are more crowded and it is more possible to treat multiple objects as single object. But this algorithm made a completion for 17 times and 18 times respectively in the evaluation of 20 times. The main reason for unsuccessful clearing is that double objects stay in a corner, and the robot can't divide them by pushing because of the limitation of push working range. Several full episodes are shown in Fig.6. We see that our algorithm has the ability to manipulate objects in clutter quickly. In our framework, it only takes 1ms to calculate from image to push action, and 5ms to detect the best grasp. Therefore, our framework is hopefully used in real world to perform the task in real time.

V. Conclusion

In this paper, we present a framework that combines pushing and grasping to manipulate objects in clutter. The pushing decision is based on reinforcement learning method that has continuous output, and the traditional rule-based method is used to make grasp detection. Thanks to the simple network architecture and practical

TABLE IV
Results of Test Environment

	test 1	test 2
Grasp Rate(%)	74.2	72.8
reward per step	0.58	0.62
completion	0.85	0.90

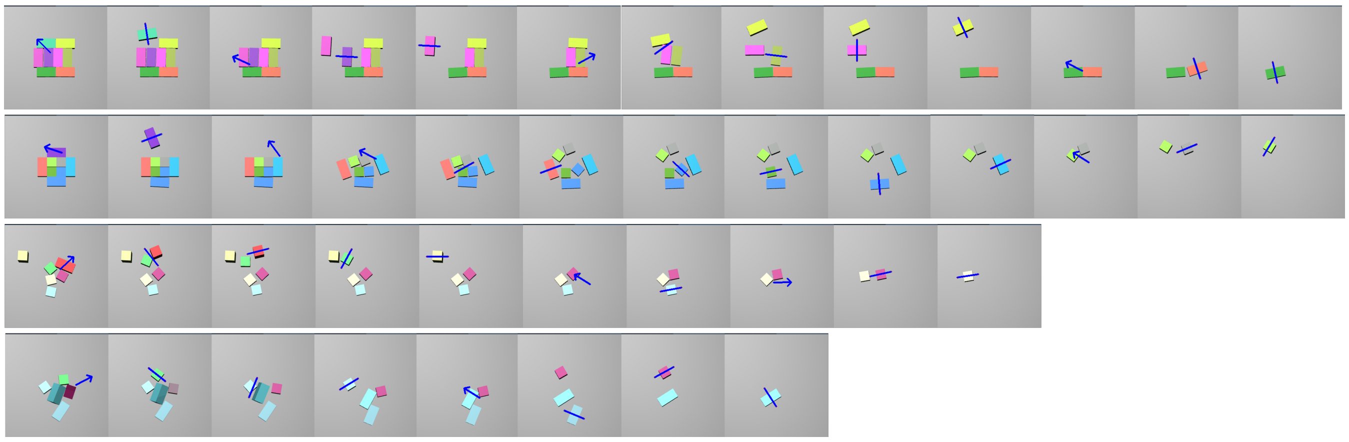

Fig. 6. Four full episodes including two tests and two random situations. Each row shows an episode.

grasp detecting algorithm, the action can be decided based on depth image within 5ms. The experiment results show that our algorithm is qualified to clear multiple objects with high efficiency and success rate. In the future, we will try to further improve the grasp rate, transfer this framework to real Baxter robot, and test it with more objects of different shapes.

References

[1] Yuan W, Hang K, Kragic D, et al. End-to-end nonprehensile rearrangement with deep reinforcement learning and simulation-to-reality transfer[J]. Robotics and Autonomous Systems, 2019, 119: 119-134.

[2] Jang E, Devin C, Vanhoucke V, et al. Grasp2vec: Learning object representations from self-supervised grasping[J]. arXiv preprint arXiv:1811.06964, 2018.

[3] Kalashnikov D, Irpan A, Pastor P, et al. Qt-opt: Scalable deep reinforcement learning for vision-based robotic manipulation[J]. arXiv preprint arXiv:1806.10293, 2018.

[4] Lee M A, Zhu Y, Srinivasan K, et al. Making sense of vision and touch: Self-supervised learning of multimodal representations for contact-rich tasks[C]//2019 International Conference on Robotics and Automation (ICRA). IEEE, 2019: 8943-8950.

[5] Matas J, James S, Davison A J. Sim-to-real reinforcement learning for deformable object manipulation[J]. arXiv preprint arXiv:1806.07851, 2018.

[6] Fujimoto S, van Hoof H, Meger D. Addressing function approximation error in actor-critic methods[J]. arXiv preprint arXiv:1802.09477, 2018.

[7] Zhang J, Yang C, Li M, et al. Grasping Novel Objects with Real-Time Obstacle Avoidance[C]//International Conference on Social Robotics. Springer, Cham, 2018: 160-169.

[8] Zeng A, Song S, Welker S, et al. Learning synergies between pushing and grasping with self-supervised deep reinforcement learning[C]//2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). IEEE, 2018: 4238-4245.

[9] Mahler J, Liang J, Niyaz S, et al. Dex-net 2.0: Deep learning to plan robust grasps with synthetic point clouds and analytic grasp metrics[J]. arXiv preprint arXiv:1703.09312, 2017.

[10] Lin H, Zhang T, Chen Z, et al. Adaptive fuzzy Gaussian mixture models for shape approximation in Robot Grasping[J]. International Journal of Fuzzy Systems, 2019, 21(4): 1026-1037.

[11] Silver D, Huang A, Maddison C J, et al. Mastering the game of Go with deep neural networks and tree search[J]. Nature, 2016, 529(7587):484-489.

[12] Human-level control through deep reinforcement learning[J]. Nature, 2015, 518(7540):529-533.

[13] Lillicrap T P, Hunt J J, Pritzel A, et al. Continuous control with deep reinforcement learning[J]. arXiv preprint arXiv:1509.02971, 2015.

[14] Nair A V, Pong V, Dalal M, et al. Visual reinforcement learning with imagined goals[C]//Advances in Neural Information Processing Systems. 2018: 9191-9200.

[15] Liang H, Lou X, Choi C. Knowledge Induced Deep Q-Network for a Slide-to-Wall Object Grasping[J]. arXiv preprint arXiv:1910.03781, 2019.

[16] Quillen D, Jang E, Nachum O, et al. Deep reinforcement learning for vision-based robotic grasping: A simulated comparative evaluation of off-policy methods[C]//2018 IEEE International Conference on Robotics and Automation (ICRA). IEEE, 2018: 6284-6291.

[17] Levine S, Pastor P, Krizhevsky A, et al. Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection[J]. The International Journal of Robotics Research, 2018, 37(4-5): 421-436.

[18] Chu F J, Xu R, Vela P A. Real-world multiobject, multigrasp detection[J]. IEEE Robotics and Automation Letters, 2018, 3(4): 3355-3362.

[19] Morrison D, Corke P, Leitner J. Closing the loop for robotic grasping: A real-time, generative grasp synthesis approach[J]. arXiv preprint arXiv:1804.05172, 2018.

[20] Berscheid L, Meißner P, Kröger T. Robot Learning of Shifting Objects for Grasping in Cluttered Environments[J]. arXiv preprint arXiv:1907.11035, 2019.

[21] Lenz I, Lee H, Saxena A. Deep learning for detecting robotic grasps[J]. The International Journal of Robotics Research, 2015, 34(4-5): 705-724.

[22] Fan L, Zhu Y, Zhu J, et al. Surreal: Open-source reinforcement learning framework and robot manipulation benchmark[C]//Conference on Robot Learning. 2018: 767-782.