

Knowledge Induced Deep Q-Network for a Slide-to-Wall Object Grasping

Hengyue Liang, Xibai Lou and Changhyun Choi

Abstract—In robotic grasping tasks, robots usually avoid any collisions with the environment and exclusively interact with the target objects. However, the environment can facilitate grasping rather than being obstacles. Indeed, interacting with the environment sometimes provides an alternative strategy when it is not possible to grasp from the top. One example of such tasks is the Slide-to-Wall grasping, where the target object needs to be pushed towards a wall before a feasible grasp can be applied. In this paper, we propose an approach that actively exploits the environment to grasp objects. We formulate the Slide-to-Wall grasping problem as a Markov Decision Process and propose a reinforcement learning approach. Though a standard Deep Q-Network (DQN) method is capable of solving MDP problems, it does not effectively generalize to unseen environment settings that are different from training. To tackle the generalization challenge, we propose a Knowledge Induced DQN (KI-DQN) that not only trains more effectively, but also outperforms the standard DQN significantly in testing cases with unseen walls, and can be directly tested on real robots without fine-tuning while DQN cannot.

Index Terms—Deep Learning in Robotics and Automation, RGB-D Perception, Perception for Grasping and Manipulation

I. INTRODUCTION

Object grasping has been one of the actively studied topics in the robotic literature. In fact, it is an important prerequisite for advanced robotic manipulation tasks. One branch of recent researches has been focusing on task level object grasping through data driven methods, such as learning to pick and classify objects in a sequential manner [1], learning simultaneous object detection and grasping pose estimate [2] and learning to employ push actions into grasping tasks to promote grasping behavior in object clutter [3].

However, the above approaches do not consider the existence of any collision objects to be avoided in the workspace. It is more challenging to grasp a target object with of a static collision object in the workspace, and interacting with the collision object provides the only applicable way, e.g., a Slide-to-Wall style grasp introduced in [4]. As demonstrated in Fig. 1, a Slide-to-Wall grasping strategy for a robot consists of the following steps: First, the gripper moves the target object towards a wall; second, it pushes the object against the wall to tilt the object; a side grasp is applied when one finger of the gripper is under the object. An example of such grasping in our life is that human will push a poker card placed on a table towards other objects to trigger a side grasping possibility.

*This work was in part supported by the MnDRIVE Initiative on Robotics, Sensors, and Advanced Manufacturing.

Authors are with the Department of Electrical and Computer Engineering, University of Minnesota, Twin Cities, Minneapolis, MN 55455, USA. {liang656, lou00015, cchoi}@umn.edu

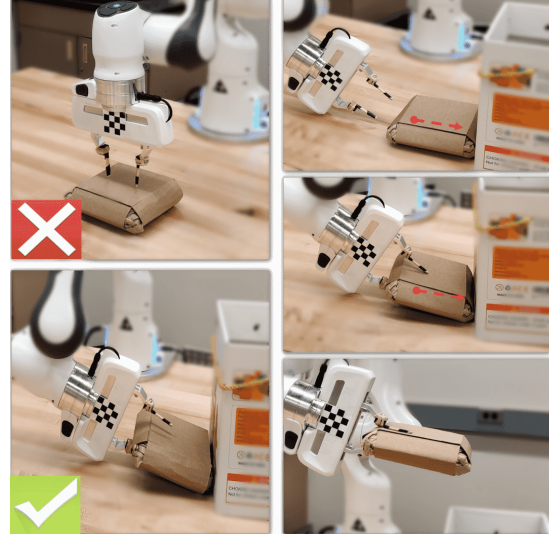


Fig. 1: Example of Slide-to-Wall grasp task (left-bottom) and a demonstration of the Shovel-and-Grasp (SaG) motion primitive (right). When the robot is given a task to grasp the oversized object, the common top grasping strategy may not work (left-top). An alternative strategy is to perform a Slide-to-Wall grasping that exploits an environmental structure, for which the Shovel-and-Grasp (SaG) motion primitive is employed. The red arrows indicate the predicted the SaG action point and direction.

As the final grasping depends on a sequence of previous actions in the Slide-to-Wall grasping task, we can formulate the problem as a Markov Decision Process (MDP). For general MDP problems, Reinforcement Learning (RL) is a powerful tool to find solutions. E.g., it has successfully demonstrated superhuman performance in finding policies to play complicated games [5], [6]. Despite the fact that RL has also gained much attention these days in robotic manipulation topics, it has overfitting problems that limit its use in those areas. The way how RL algorithms is designed to function makes themselves task specific and they must be trained in the exact environment supposed to be applied. When the workspace is not free from collisions, self exploration of RL is risky and unacceptable. Hence, applying RL algorithm will be problematic as it would potentially damage both the robot and the environment.

In order to solve the Slide-to-Wall grasping problem effectively and avoid collisions for real robots, we need to simultaneously tackle the generalization challenge of the RL methods. More specifically, we would like a trained policy to be able to predict both effective and safe actions even

when the testing environment is unseen from training. In this work, we propose a variant of the DQN method, namely Knowledge Induced DQN (KI-DQN), with a single motion primitive, Shovel-and-Grasp (SaG), to achieve the goal. The significance of our approach is two-fold:

- 1) **A compound motion primitive SaG that utilizes push and the collaborative grasp between the gripper and the environment** — First, the gripper tip point retreats 10 *cm* away from the predicted action point along the predicted direction. Then the gripper moves forward to the action point and closes the gripper when the tip is at the predicted point, as shown in Fig. 1. When the target object is at free space, this motion primitive is equivalent to a push that moves the object toward the wall; when the target object is near the wall, it becomes a shovel motion that one finger of the gripper goes underneath the object and is able to result into a successful side grasp.
- 2) **An end-to-end trained KI-DQN** that is embedded with the knowledge of the target object during training and reflects a detection of the target object from its predicted Q function. Even the network is purely trained in *one single set of simulation configurations*, it can be directly transfer to different, unseen environment setups of simulation testing cases and even real world experiments, without necessitating harmful actions on the walls with unseen shapes.

II. RELATED WORKS

There are many works focusing on task level manipulations for robotic grasping. E.g., [7] studies object grasping in multi-object environment using a handcrafted sweeping to move the none-graspable objects. In addition, it uses pushing as a pre-grasp behavior on the target object so that grasping is more robust with less uncertainty. However, as the problem gets more complex, e.g., grasping in densely cluttered environment, or combining other motions with grasping, the handcrafted policies start to lose its competitiveness.

To find better solutions for complex tasks, many researches turn to RL methods. Not only showing superhuman behavior by self-learning via massive try-and-error in many games [5] [8], RL algorithms have been successfully applied to robotics in many research areas, such as robot local motions [9] [10], motor skills of limb behavior [11] [12], force control of compliant manipulation [13] and autonomous aerobatic helicopter flight control [14]. RL has also been used to explore push and grasp synergies for robotic grasping tasks. E.g., [15] shows the possibility of utilizing an RL algorithm to train a control policy of selecting push and grasp proposals for densely cluttered objects. However, their entire pipeline is not end-to-end trained, as its RL algorithm excludes object segmentation, action proposals and extracting handcrafted features of the object that are used to complete the grasp task. Moreover, the method depends on model-based simulation to provide predictions on the proposed push and use the prediction as an inference to guide the RL selection. The handcrafted features can potentially hurt the ability of RL

method in finding good policies as they can limit the power, or even mislead the self exploration knowledge of RL algorithms.

[3] presents an interesting work on robotic grasping tasks, where an end-to-end DQN framework is proposed to solve a push and grasp synergy problem of cleaning a clutter of objects, showing that a Convolutional Neural Network(CNN) based Q function can be mapped to a discretized workspace and regulate pre-defined action primitives. Unlike other RL methods, this method does not require massive training samples. However, it only applies to a risk-free environment to apply on real robots. Otherwise, the exploration of the action primitives during training is dangerous for the robot.

Having a risk-free environment is not always true for robotic grasping, e.g., [4] introduces a Slide-to-Wall grasping strategy that uses the wall as a helper to grasp a target object, though the work itself mainly focuses on designing a compliant robotic gripper. In the Slide-to-Wall grasping task, the challenge to apply DQN methods is that it is likely to crush the robot from collisions first before an effective policy is learned if they are directly applied; even a policy is learned from training, it may not be able to generalize to other environment settings where the wall has only a small change from the training case.

Inspired by [3], our work utilizes the CNN based DQN method to solve the Slide-to-Wall grasping task. We propose a new training strategy for DQN to achieve good generalization abilities to unseen environment settings, while being effective and sample efficient in finding good policies.

To the best of our knowledge, the DQN formulation for the object grasping strategy exploiting the environment and the improvement of the generalization capability and sample complexity of it have not been studied yet.

III. PROBLEM FORMULATION

As introduced in Section I, when the object is not graspable with a single grasping trial, a sequence of the SaG actions should be applied to move the object close to the wall, and finally an effective grasp is performed in a proper approaching direction with the help of the wall. This sequential decision making problem can be formulated as a discrete Markov Decision Process (MDP). An MDP is formally defined as a tuple $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma)$, where \mathcal{S} is the set of environment states, \mathcal{A} is a finite set of actions, \mathcal{P} is the transition probability matrix, \mathcal{R} is the reward function and $\gamma \in (0, 1)$ is the discount factor that balances the immediate reward by an action and future expected reward [16].

In this problem, the environment state $s_t \in \mathcal{S}$ (e.g., the positions and orientations of the wall and object) is assumed unknown to the robot. The robot estimates the environment through a fixed, mounted RGB-D camera. For convenience, we treat s_t , with some abuse of notation, equivalent to the height-map image that is captured by the RGB-D camera and transformed via the known extrinsic of the camera w.r.t. the robot coordinate frame. The workspace is a 40 *cm* \times 40 *cm* square on the table, and is divided equally into a 40 \times 40 discrete action grids. The SaG thus has a resolution of 1 *cm*²

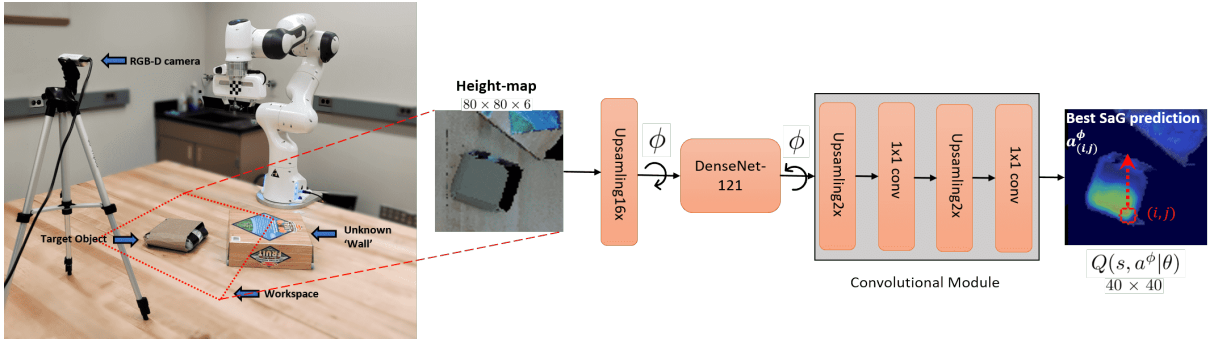


Fig. 2: **System Overview.** Before each robot action, the camera captures an RGB-D image of the workspace (dashed red square) and forms a $80 \times 80 \times 6$ height-map representation by concatenating ‘RGBDDD’ channels in order to take advantage of pre-trained DenseNet models (a pre-trained DenseNet takes a 3 channel image as input). The height-map is rotated with an orientation ϕ (in our case $0^\circ, 45^\circ, 90^\circ$) and fed into a DenseNet-121 structure to get $10 \times 10 \times 1000$ intermediate feature. Then the feature is rotated back ($-\phi$) and passed through a Convolutional Module to get a 40×40 Q map representing the score SaGs acting at point (i, j) on the workspace with rotation ϕ .

on the workspace. The SaG can be executed within each grid with a set of rotation angles ϕ . Though the rotation angles can be arbitrarily defined, we choose three rotation angles, $0^\circ, 45^\circ$, and 90° , due to the limited view of the (single) camera, the robot and workspace configurations. Therefore, the entire action space \mathcal{A} consists of $40 \times 40 \times 3 = 480$ actions. \mathcal{P} is the intrinsic transitions of the environment and is unknown to the robot. As the goal of the task is to successfully grasp the object, the immediate reward $r_t = 1$ is assigned only when the applied SaG successfully grasps an object at time t , otherwise $r_t = 0$ even if the SaG moves the object closer to the wall.

IV. METHOD

A. Deep Q-Network (DQN)

DQN is a model-free, policy based RL algorithm where the agent directly learns a policy to pick actions given the current state. \mathcal{P} is implicitly learned within the policy, making DQN a good candidate to solve the problem.

For general Q learning methods, an action-value function (Q-function) Q^π subject to a policy π is defined as

$$Q^\pi(s, a) := \mathbf{E}_{s, a, \pi} \left[\sum_{t=1}^{\infty} \gamma^t r_t \right] \quad (1)$$

where $\gamma \in (0, 1)$ is the discount factor. The optimal Q function is $Q^*(s, a) = \max_{\pi} Q^\pi(s, a)$ and the optimal policy is proven to be greedy [5]. That is, the agent picks the action a by $a^* = \pi^*(Q^*(s, a)) = \arg \max_a Q^*(s, a)$.

As for DQN, a parameterized function $Q^\pi(s, a; \theta)$ is used to approximate the actual Q-function. A deep Neural Network (NN) is typically used as a function approximator and the weights of the network θ is updated through gradient descent as follows:

$$\theta_{t+1} \leftarrow \theta_t + \alpha (y_t^Q - Q(s_t, a_t; \theta_t)) \nabla_{\theta} Q(s_t, a_t; \theta) \quad (2)$$

where α is the learning rate scalar, $y_t^Q = r_t + \gamma \max_a Q(s_{t+1}, a; \theta^-)$ is the update target value. $\theta^- = \theta^{t_0}$ for some $t_0 < t$ as a fixed value and updated by $\theta^- = \theta_t$ once

every predefined training steps to ensure network stability during training [17]. In [3], a CNN based Q function estimator is used to solve the push and grasp synergy problem and showed effectiveness. We use this method as a baseline which is compared with our approach in Section V.

B. Model Structure

We use a CNN as a Q-function approximator, which can be found in Fig. 2. The network takes an RGB-D image as input and outputs a 40×40 Q function map $Q(s, a_{i,j}^\phi; \theta)$, where ϕ denotes the SaG rotation angle. $a_{i,j}^\phi$ denotes the action on i^{th} column and j^{th} row on the ϕ rotated Q-function map and can be mapped to the actual workspace with 1 cm^2 resolution.

The input height-map of the network is first upsampled 16 times and rotated with the corresponding angle ϕ . The network then draws $10 \times 10 \times 1000$ intermediate features by passing through a DenseNet-121 [18] without the last classification module, and rotates the features back with angle $-\phi$ and passes the rotated features through a Convolutional Module (CM) to get the final output Q function map. The CM is a repeated sequence of Upsampling2x layer followed by a 1×1 Convolutional layer.

The complete Q function is given as

$$Q(s, a; \theta) = \bigcup_{i, j, \phi} Q(s, a_{i,j}^\phi; \theta) \quad (3)$$

by rotating the input image with the different rotation angles ϕ and feed into the network multiple times.

The network model has the following advantages: First, it learns *orientation invariant intermediate features* that can be more easily trained and extended to different orientations simply by rotating the input images. Second, the final CM module also acts as a buffer zone to eliminate the aliasing effect caused by the rotation of the discrete action grid.

C. Knowledge Induced Deep Q-Network

There are concerns for RL methods to directly apply to robotic manipulations. How to ensure safe explorations

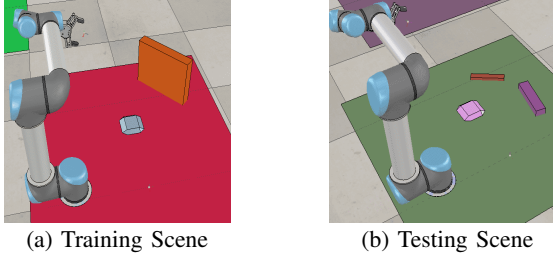


Fig. 3: **Example of training and testing scenes in simulation.** Training scene (a) uses a random colored object with a high static wall on a random colored background. Testing scene (b) uses the same random colored object with low static wall (or walls).

during real implementations while the environment can be different from training is one of them. For DQN methods regulating action primitives based on predicted Q function map, one of the solutions is to develop a perceptual recognition of the object reflected in the predicted Q-function. However, DQN by its own update (2) is not able to get such perception (See V-A). We thus need to induce the perceptual knowledge into DQN and its predicted Q function through extra mechanisms.

One useful knowledge in this problem is that *the predicted action point should always be located on the object*—a SaG on the background floor results no changes on the states and a SaG on the wall is strictly forbidden.

Let \mathcal{A}^ϕ denote the set of SaG actions that locates on the area where the target object is projected onto the Q map with rotation ϕ . A knowledge induced Q-function with target object detection is thus given as:

$$Q^\pi(s, a_{i,j}^\phi) := \begin{cases} 0, & \text{if } a_{i,j}^\phi \notin \mathcal{A}^\phi \\ \mathbf{E}_{s,a,\pi}[\sum_{t=1}^{\infty} \gamma^t r_t], & \text{otherwise} \end{cases} \quad (4)$$

and the gradient update rule (2) becomes

$$\theta_{t+1} \leftarrow \theta_t + \sum_{a_{i,j}^\phi \in \mathcal{A}} \alpha(y_t^{\phi,i,j} - Q(s_t, a_{i,j}^\phi; \theta_t)) \nabla_{\theta} Q(s_t, a_{i,j}^\phi; \theta) \quad (5)$$

where

$$y_t^{\phi,i,j} = \begin{cases} 0, & \text{if } a_{i,j}^\phi \notin \mathcal{A}^\phi \\ Q(s_t, a_{i,j}^\phi; \theta_t), & \text{if } a_{i,j}^\phi \in \mathcal{A}^\phi \text{ and } a_{i,j}^\phi \neq a_t \\ r_t + \gamma \max_a Q(s_{t+1}, a; \theta^-), & \text{otherwise} \end{cases} \quad (6)$$

The update gradient at each training step tries to:

- Press the predicted value of the actions outside the object area towards zero.
- Keep the predicted value of the actions on the object area but not executed on the previous step as it was.
- Drive the executed action that is on the object area towards the vanilla DQN update value.

Intuitively by (5), the network will be able to explore and develop good policies on the area where the object is located, and at the same time it is enforced to *learn* the position of the object, thus developing the detection of the target object.

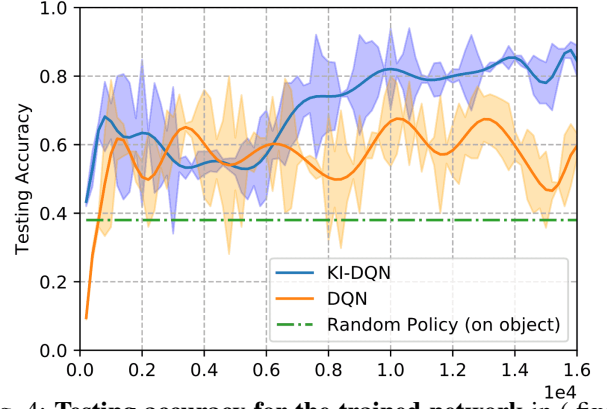


Fig. 4: **Testing accuracy for the trained network in (fixed and randomly generated) training scenes.** The x-axis is the iteration number that the network parameter is saved from training.

When the network predictions on the off-object area are zero, (5) reduces to vanilla DQN update (2). To get gradient in (5), we only need to provide the extra information of where the object is located on the Q-function map.

V. EXPERIMENT AND RESULT

We use V-REP with bullet 2.83 engine as the simulation environment for training. Our neural network model is trained by PyTorch version 0.3.1. The policies for both DQN and KI-DQN are trained purely with one target object and one wall only in the simulation environments, as shown Fig. 3a. We test the trained model in the following scenarios:

- 1) The same set of simulation scenes as the training scene.
- 2) Simulation scenes that walls have different shape and height from the training scene as Fig. 3b.
- 3) Real experiment scenarios as Fig. 2.

We aim to validate from the experiments whether:

- 1) KI-DQN and DQN methods can learn a effective policy that solve this problem?
- 2) KI-DQN can learn and keep the perceptual knowledge of the object from training and generalize it to unseen scenarios, while having an effective policy?
- 3) It is possible to directly apply the KI-DQN model trained in simulation only to real environment and yet it maintains safe actions?

A. Training

In each training episode (scene), we place a 25 cm high, 5 cm thick static wall near one corner of a 40 cm × 40 cm square workspace with random position, orientation, and color. We drop a target object (15 cm × 15 cm × 3 cm box) as shown in Fig. 3a with a random color and position on the workspace. The floor of the workspace is also set with random color each scene.

Both DQN and KI-DQN use the same model structure as in IV-B, optimized by Stochastic Gradient Descent (SGD) with 0.9 momentum and 2×10^{-5} weight decay. The learning rate is set to be 10^{-5} for the DenseNet-121 layers and 10^{-4} for the CM module. The network does an online update every

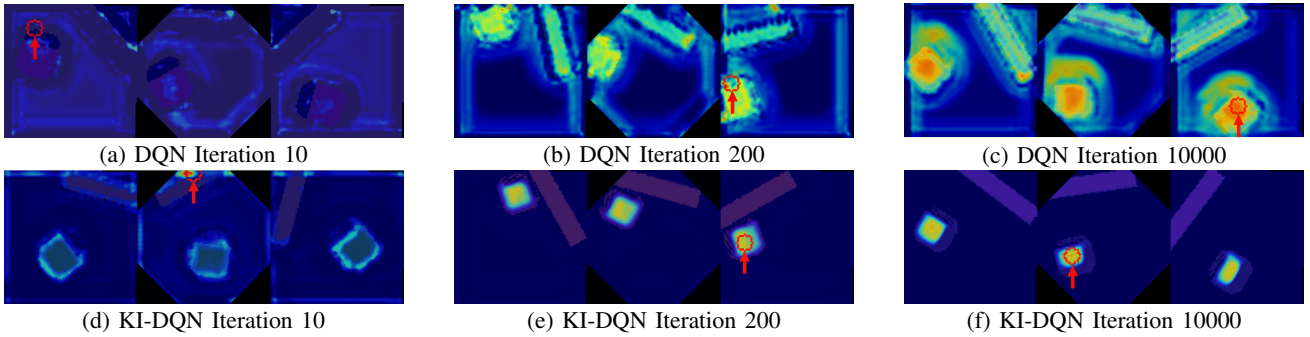


Fig. 5: **Visualized predicted Q function for SaG during training.** From left to right in each subfigure: 0° , 45° , 90° rotated Q maps. The red circle indicates the highest Q value in each prediction and the red arrow indicate the SaG direction. (a), (b), (c) are the predicted Q function map for DQN method; (d), (e), (f) are the predicted Q function map for KI-DQN method. The brighter and more reddish area, the higher predicted Q value.

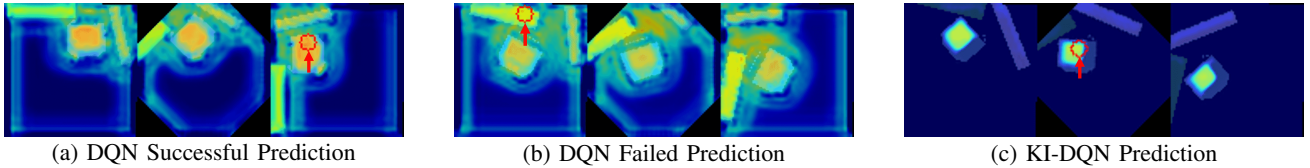


Fig. 6: **Examples of Visualized predicted Q function for SaG under testing case in simulation (low wall scenes).** From left to right in each subfigure: 0° , 45° , 90° rotated Q maps. The red circle indicates the highest Q value in each prediction and the red arrow indicate the SaG direction. Though 6a is a successful SaG prediction by DQN, there are also high predictions on the wall. This indifferent prediction about the object and wall causes typical failures for DQN as is in 6b. 6c is the prediction from KI-DQN, where all high valued prediction are focused on the object even the walls are unseen in training.

iteration (one robot action) and one prioritized experience replay [19] on one sampled transition. y_t in both (2) and (5) is updated every 200 iterations. The reward discount factor $\gamma = 0.95$ is used for both methods.

As the reward is binary at the end of each episode, it is not effective to evaluate the training process by plotting the gained reward. Thus we reserve 50 sample cases (generated in the same random manner as the training scenes) and evaluate the model every 200 iterations. The result can be found in Fig. 4. A random policy is also evaluated as a baseline, where a SaG is executed at a random action point on the object with a random angle.

Fig. 4 shows that both methods start to form an effective policy quickly and achieve better than random behavior. As the training goes, KI-DQN continuously improves the policy while DQN seems to stuck around 0.7 or less accuracy. This may due to the fact that KI-DQN provides more informative gradients to provide consistently efficient updates, while DQN does not.

Fig. 5 visualizes the evolution of $Q(s, a; \theta)$ for both methods as the training goes. Though DQN manages to find effective policies to solve the task, the update itself cannot manage to learn to recognize the target object. KI-DQN, on the other hand, gradually improves the policy and absorbs the induced knowledge at the same time (high Q values on the object only from Fig. 5e to Fig. 5f).

| Method | Task Success Rate (%) |
|---------------------------|------------------------------------|
| KI-DQN (ours) | 93.45 ± 2.71 |
| DQN | 60.00 ± 5.78 |
| Random Policy (on object) | 37.00 ± 3.00 |

TABLE I: Testing results in simulation scenes.

B. Evaluating Generalization Ability

Similar to Section V-A, we prepare 50 randomly generated scenes with the same object but unseen wall settings to test the ability to generalize. As is shown in Fig. 3b, we place one or two smaller walls with random orientations to simulate different wall shapes. The height of the wall(s) is randomly sampled from the uniform distribution $U[0.5, 3]$ cm, which is less or equal to the object height. During evaluation, the network parameters are fixed and the policy is greedy without exploration. We pick 10 best models each from both DQN and KI-DQN methods to evaluate, and the maximal number of trial actions is set as 15 per scene.

Fig. 6 illustrates the visualized Q predictions during this evaluation for both DQN and KI-DQN. We can see from Fig. 6c that even when the wall setting is unseen during training, KI-DQN still manages to recognize the target object successfully and the predicted high Q value concentrates on the object. However, DQN cannot guarantee the same condition. Fig. 6a illustrates one example of a successful DQN prediction on the testing scenario and Fig. 6b displays one failed case where DQN predicted the best SaG point is on

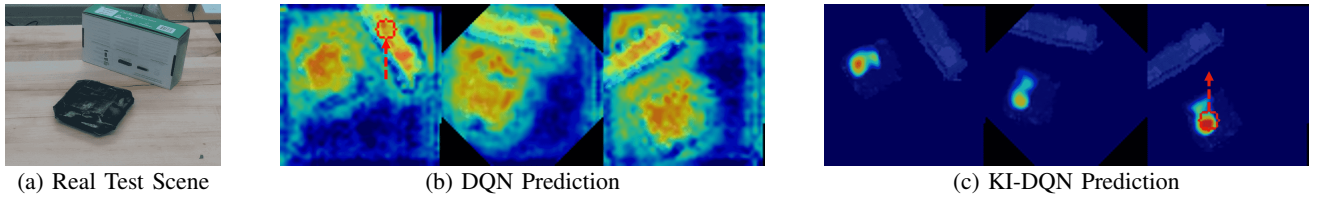


Fig. 7: **Visualized predicted Q function in a real experiment similar to the train scene.** The red circle indicates greedy predicted action. We use the flat black box as the target object and another box as the wall. Though it is geometrically similar to the scene where the networks are trained in simulation, DQN generates actions on the wall while KI-DQN correctly detects the target object and predicts a effective action.

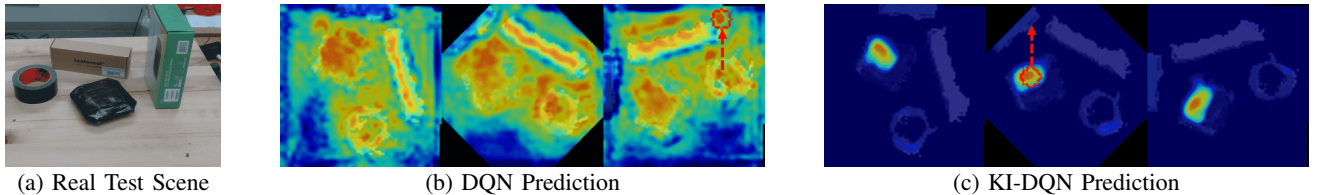


Fig. 8: **Visualized predicted Q function in a real experiment with very different wall setups.** We use the flat black box as the target object and all other objects as the walls. Even when the environment is significantly differs from training, KI-DQN manages to give effective predictions while DQN cannot.

the wall. Table I summarizes the quantitative result of the evaluation. From Table I, we see that KI-DQN outperforms the DQN and Random Policy by a large margin in unseen wall settings. Note that in simulation testing scenes, robot collision on the wall is allowed and the task is only considered to fail by using up all 15 trials. Thus this result reflects a consistent prediction failure risk for DQN method when the environment settings are changed.

C. Real Experiment

We also apply the trained network for both methods in real experiments. A flat box is used as the target object and the other objects with different shapes to simulate walls. Unlike testing in simulation, the task is considered as failure if the network predicts one action on the wall.

Fig. 7 shows a test setting where the scene is similar to the training scene in simulation, and the corresponding predicted Q map for both methods. While DQN cannot guarantee similar behaviors as in training and predicts action on the wall, KI-DQN manages to give reliable and effective predictions on the object.

Table II shows the quantitative result of both DQN and KI-DQN methods in real robot settings as shown Fig. 7. We can see that DQN method successful rate drops significantly compared to its testing result in simulated scenes. This gap is mostly due to the fact that a collision with the wall is considered a direct failure in real experiments. However, our KI-DQN still gives a good successful rate in completing the task, indicating its reliable generalization ability.

We also set the real environment to be very different from training, as is shown in Fig. 8, and KI-DQN is still able to show accurate perceptions of the target object and predicts effective actions while DQN cannot.

| Method | Success Rate (%) |
|---------------|------------------|
| KI-DQN (ours) | 71.42 |
| DQN | 21.43 |

TABLE II: Testing results in real robot experiments.

VI. DISCUSSION AND FUTURE WORK

In this work we present a variant of DQN method that allows a CNN based Q function to be able to detect the target object of interest and at the same time developing an effective policy to regulate action primitives to solve a Slide-to-Wall robot grasping task. Though the network is purely trained under one single set of scenes in simulation without any supervision, the network is able to generalize its trained policy to unseen testing environments and reliably transfer to the real scene.

The limitation of this work is that the settings of the Slide-to-Wall grasping task require a relatively simple policy due to the workspace and robot limitations. It only requires one action primitive to complete the task. One of the future works can be investigating whether this method can be applied to a more complicated task requiring a collaboration of different motion primitives. Another future work would be studying whether a more dynamic control of robots that learned from RL algorithms, rather than pre-defined discrete action primitives, can also achieve similar generalization behavior to unseen environments.

REFERENCES

- [1] A. Zeng, S. Song, K.-T. Yu, E. Donlon, F. R. Hogan, M. Bauza, D. Ma, O. Taylor, M. Liu, E. Romo *et al.*, “Robotic pick-and-place of novel objects in clutter with multi-affordance grasping and cross-domain image matching,” in *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2018, pp. 1–8.

- [2] G. Wu, W. Chen, H. Cheng, W. Zuo, D. Zhang, and J. You, "Multi-object grasping detection with hierarchical feature fusion," *IEEE Access*, vol. 7, pp. 43 884–43 894, 2019.
- [3] A. Zeng, S. Song, S. Welker, J. Lee, A. Rodriguez, and T. Funkhouser, "Learning synergies between pushing and grasping with self-supervised deep reinforcement learning," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2018, pp. 4238–4245.
- [4] C. Eppner, R. Deimel, J. Alvarez-Ruiz, M. Maertens, and O. Brock, "Exploitation of environmental constraints in human and robotic grasping," *The International Journal of Robotics Research*, vol. 34, no. 7, pp. 1021–1038, 2015.
- [5] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, p. 529, 2015.
- [6] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel *et al.*, "A general reinforcement learning algorithm that masters chess, shogi, and go through self-play," *Science*, vol. 362, no. 6419, pp. 1140–1144, 2018.
- [7] M. R. Dogar and S. S. Srinivasa, "A planning framework for non-prehensile manipulation under clutter and uncertainty," *Autonomous Robots*, vol. 33, no. 3, pp. 217–236, 2012.
- [8] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot *et al.*, "Mastering the game of go with deep neural networks and tree search," *nature*, vol. 529, no. 7587, p. 484, 2016.
- [9] N. Kohl and P. Stone, "Policy gradient reinforcement learning for fast quadrupedal locomotion," in *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA'04. 2004*, vol. 3. IEEE, 2004, pp. 2619–2624.
- [10] G. Endo, J. Morimoto, T. Matsubara, J. Nakanishi, and G. Cheng, "Learning cpb-based biped locomotion with a policy gradient method: Application to a humanoid robot," *The International Journal of Robotics Research*, vol. 27, no. 2, pp. 213–228, 2008.
- [11] E. Theodorou, J. Buchli, and S. Schaal, "Reinforcement learning of motor skills in high dimensions: A path integral approach," in *2010 IEEE International Conference on Robotics and Automation*. IEEE, 2010, pp. 2397–2403.
- [12] J. Peters and S. Schaal, "Reinforcement learning of motor skills with policy gradients," *Neural networks*, vol. 21, no. 4, pp. 682–697, 2008.
- [13] M. Kalakrishnan, L. Righetti, P. Pastor, and S. Schaal, "Learning force control policies for compliant manipulation," in *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2011, pp. 4639–4644.
- [14] P. Abbeel, A. Coates, M. Quigley, and A. Y. Ng, "An application of reinforcement learning to aerobatic helicopter flight," in *Advances in neural information processing systems*, 2007, pp. 1–8.
- [15] A. Boularias, J. A. Bagnell, and A. Stentz, "Learning to manipulate unknown objects in clutter by reinforcement," in *Twenty-Ninth AAAI Conference on Artificial Intelligence*, 2015.
- [16] C. White, *Markov decision processes*. Springer, 2001.
- [17] H. Van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double q-learning," in *Thirtieth AAAI conference on artificial intelligence*, 2016.
- [18] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 4700–4708.
- [19] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized experience replay," *arXiv preprint arXiv:1511.05952*, 2015.