Rearrangement with Nonprehensile Manipulation Using Deep Reinforcement Learning

Weihao Yuan[1], Johannes A. Stork[2], Danica Kragic[2], Michael Y. Wang[1] and Kaiyu Hang[1]

Abstract— **Rearranging objects on a tabletop surface by means of nonprehensile manipulation is a task which requires skillful interaction with the physical world. Usually, this is achieved by precisely modeling physical properties of the objects, robot, and the environment for explicit planning. In contrast, as explicitly modeling the physical environment is not always feasible and involves various uncertainties, we learn a nonprehensile rearrangement strategy with deep reinforcement learning based on only visual feedback. For this, we model the task with rewards and train a deep Q-network. Our potential field-based heuristic exploration strategy reduces the amount of collisions which lead to suboptimal outcomes and we actively balance the training set to avoid bias towards poor examples. Our training process leads to quicker learning and better performance on the task as compared to uniform exploration and standard experience replay. We demonstrate empirical evidence from simulation that our method leads to a success rate of 85%, show that our system can cope with sudden changes of the environment, and compare our performance with human level performance.**

I. INTRODUCTION

The skill of *rearrangement planning* is essential for robots for manipulating objects in cluttered and unstructured environments [1–4]. Classic approaches to object rearrangement use so-called *pick-and-place actions* and rely on grasp [5–8] and motion planning [9, 10]. Assuming that the robot's workspace is constrained to a tabletop, more recent works try to leverage on *nonprehensile actions* for more efficient solutions [11–14], however, exchanging complex grasp planning for planning of complex robot-object interactions.

Besides the fact that the general problem is *NP-hard* [15], rearrangement planning poses many other challenges which are often addressed under simplified assumptions. Due to occlusions caused by clutter in a single camera setup, a robot often suffers from incomplete knowledge of the environment's state [16]. Therefore, a number of recent works assume complete observability of the state from perfect visual perception for planning [11–14]. Often, the complex dynamics of nonprehensile interaction are reduced to a quasi-static model [17, 18] which conveniently allows solutions based on motion primitives [19, 20]. Moreover, for keeping planning of action sequences tractable, physical

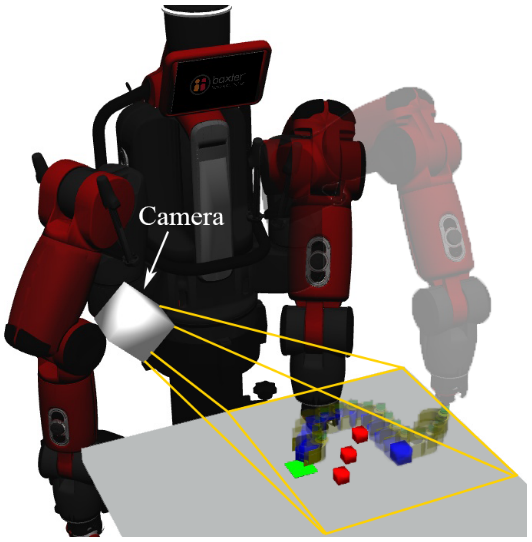


Fig. 1: The robot is tasked to first find and then push an object (blue) around obstacles (red) to a goal region (green) relying on only visual feedback.

properties are often assumed to be known such that robot-object interactions can be simulated [12]. In some cases, a free-floating end-effector is assumed to avoid expensive planning in configuration space and to allow physics-aware planning with kinodynamic-RRT [11]. All these approaches treat perception, action planning, and motion planning separately.

In this work, we design a learning system that treats perception, action planning, and motion planning in an end-to-end process. Different from whole-arm interaction as studied by King *et al.* [12], our task consists of pushing a manipulation object to a target position while avoiding collisions, as illustrated in Fig. 1. Perceptions are single-view RGB images and the actions move a manipulation tool in five different directions. Different to model or simulation-based approaches [11, 12], we assume no prior knowledge of any physical properties such as mass, inertia matrices, friction coefficients, and so on.

Instead of a classic planning framework which requires an explicit physical model, we use model-free Q-learning [21] to find an *optimal* policy directly from visual input. Since our workspace consists of many objects located at arbitrary positions, the state space is infeasible for classic Q-learning. However, based on only visual input, latest research on deep Q-network (DQN) successfully shows the power of deep convolutional neural networks in playing Atari games with human-level performance [22]. Therefore, we employ DQN for our tabletop rearrangement problem, which bears

[1] These authors are with the Hong Kong University of Science and Technology and HKUST Robotics Institute. W. Yuan is with the Department of Electronic and Computer Engineering. M. Y. Wang is with the Department of Mechanical and Aerospace Engineering and the Department of Electronic and Computer Engineering. K. Hang is with the Department of Computer Science and Engineering and HKUST Institute for Advanced Study.

[2] J. A. Stork and D. Kragic are with the Robotics, Perception and Learning Lab, Centre for Autonomous Systems, KTH Royal Institute of Technology, Sweden

similarities to Atari games, both in perception and state transitions. Similar to the games, our robot operates in a stochastic world where obstacles can move at any time and friction varies, requiring reactive behavior. This can be addressed since a DQN determines actions based on only the current input as opposed to a classic planning framework.

Our contributions concern both the rearrangement task and the learning process and consist of:

1) modeling the rearrangement task as a reinforcement learning problem with task-specific reward functions,
2) improving the training process by active control of the replay dataset to avoid bias towards suboptimal examples,
3) devising an informed exploration process based on a Gaussian potential field to reduce the amount of suboptimal examples caused by collisions.

In our simulation-based evaluation, the DQN trained with only 2 random obstacles can achieve high success rates when presented with 2 to 4 randomly positioned obstacles. We interpret this as evidence that the network learns both global features for path planning and local features for collision avoidance. Our comparison against the performance of a human expert player indicates that the DQN plans more conservative to avoid collisions. Furthermore, we qualitatively show that our system can react to sudden changes in the positions of the object, obstacles or the target, as well as the randomly altered friction coefficient, and a distracting novel object introduced to the scene.

This paper is structured by formally defining the problem in Sec. II, and then introducing the necessary preliminaries in Sec. III. In Sec. IV we explain the details of our design of DQN-based learning architecture. Finally, we evaluate our system in Sec. V and conclude in Sec. VI.

II. PROBLEM STATEMENT

In this section, we formally define the task and the necessary assumptions.

A. Task and Assumptions

We assume that a robot is equipped with a non-prehensile manipulation tool, which can move along the planar work-surface to reach all required positions. As shown in Fig. 1, on the work-surface there is a cube-shaped manipulation object, a few cube-shaped obstacle objects, and a squared visual indicator for the target location. The manipulation tool has a fixed orientation, while the target location and the manipulation object on the work-surface are initially situated in the half-space in front of the tool. Mass and friction of the object and obstacles are not known but allow for effortless manipulation. We assume that the target position is not fully blocked by obstacles and that there exists at least one path allowing the manipulation tool to push the object into the target area.

The work-surface is observed by a static single-view RGB camera perceiving the manipulation object in *blue*, the obstacles in *red*, the target location in *green*, and the robot arm with the attached manipulation tool with possible

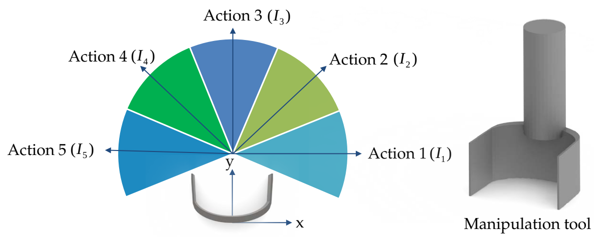


Fig. 2: This figure illustrates the 5 predefined motion directions for the manipulation tool. Action 3 is aligned with the front direction Y of the manipulation tool. The colored sectors depict the ranges in which the potential integrals are calculated, as required in Sec. IV-C for informed action sampling.

occlusions. Manipulation is done in discrete time steps such that a camera image is recorded at time step t and then an action is executed leading to the next time step $t+1$.

The task is to find a sequence of predefined actions, as depicted in Fig. 2, to push the object from a random initial position to the target area while avoiding collisions with any randomly positioned obstacles. Note that with these 5 actions, the manipulation tool can achieve a large set of trajectories but it cannot move backwards.

B. Definitions and Notations

Observations. An observation $\mathbf{x}$ is a 128×128 RGB image (49152 dimensions) taken from the camera pointing at the work-surface. An example of an observation is seen in Fig. 3. In the images, the robot and objects can occlude parts of the scene.

Actions. An action $a \in \mathcal{A}$ translates the manipulation tool parallel to the work-surface using one of the predefined motion directions for a fixed step size $d_a \in \mathbb{R}^+$.

Episodes. An episode E is a sequence of actions that is terminated by either success or failure. We index the set of episodes by k and use time steps $t = 1, 2, \ldots, T$ within episodes where T might be different from episode to episode.

Success and Failure. An episode terminates with *success* iff. the manipulation object reaches the target location. Otherwise, it terminates with *failure* in cases when too many time steps have passed, obstacles are moved (collision), or the tool is moved outside of the work-surface.

Grounding Labels. During the learning process, the algorithm has access to the following 2D positions relative to the work-surface's frame: Manipulation object position $\mathbf{p}^{\text{man}}$, tool position $\mathbf{p}^{\text{tool}}$, target location $\mathbf{p}^{\text{target}}$, and for each obstacle i the position $\mathbf{p}^{\text{obs},i}$. The positions are all measured in centimeters. From these we can derive predicates for *success* and *failure*.

C. Objective

Our goal is to learn a robust function $Q(\mathbf{x}, a)$ over all relevant camera images $\mathbf{x}$ and actions $a \in \mathcal{A}$, such that repeatedly taking the best actions $a^* = \arg\max_{a \in \mathcal{A}} Q(\mathbf{x}, a)$ in subsequent situations moves the manipulation object to the target location. It must be possible to start in any situation where the manipulation object and target location are situated in front of the manipulator as described above. Learning this function alleviates the problems of explicitly modeling the

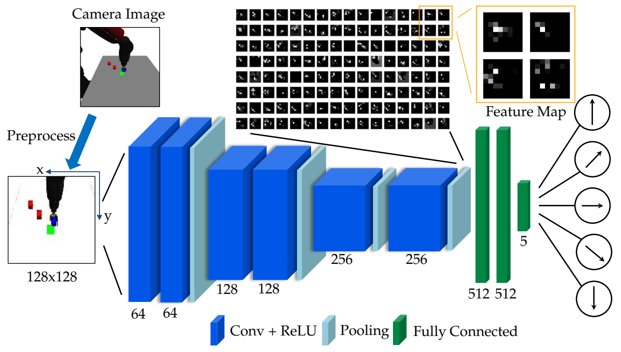


Fig. 3: We represent the action-value function with the deep convolutional neural network structure depicted here. The network computes Q-values for each action in parallel. The picture captured by the camera, the 128x128 image to be fed to the network and the feature map output by the convolutional part are shown.

environment with its dynamics, tracking the manipulation object, or executing a planning algorithm.

III. PRELIMINARIES

Our method is based on learning a deep Q-network [22] from experiences while using Gaussian potential fields [23, 24] to generate pertinent and informative examples during exploration.

A. Deep Q-Learning

Deep Q-learning considers tasks in which the agent interacts with the environment through a sequence of observations $\mathbf{x}_t$, actions a_t, and rewards r_t. The goal is to select actions that maximize cumulative reward. For this, the optimal state-action function (Q-function [22]),

$$Q^*(\mathbf{x}, a) = \max_{\pi} \mathbb{E}\left[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \dots \mid \mathbf{x}_t = \mathbf{x}, a_t = a, \pi\right], \tag{1}$$

is approximated by a deep convolutional neural network $Q(\mathbf{x}, a; \theta)$ with parameters θ. This is the maximum sum of rewards discounted by γ achieved by the policy π after making observation $\mathbf{x}$ and taking action a.

Representing the state-action function by a nonlinear function approximator can lead to instability and divergence [25]. These problems are usually addressed by *experience replay* [22, 26–28] and by training separate *target* and *primary networks*, with parameters θ^t and θ^p respectively, which are updated in different frequencies [22]. For this, previous experiences $e_t = (\mathbf{x}_t, a_t, r_t, \mathbf{x}_{t+1})$ from time steps t are stored in a replay buffer D to optimize the loss function,

$$\mathcal{L}(\theta^p) = \mathop{\mathbb{E}}_{(\mathbf{x},a,r,\mathbf{x}')}\left[\left(r + \gamma \max_{a' \in \mathcal{A}} Q(\mathbf{x}', a'; \theta^t) - Q(\mathbf{x}, a; \theta^p)\right)^2\right], \tag{2}$$

for which $(\mathbf{x}, a, r, \mathbf{x}')$ is sampled from D according to some distribution. The target network parameters are updated towards the primary network parameters upon a certain schedule.

Once the network is successfully trained, the greedy policy which selects the action with the maximal Q-value,

$$\pi(\mathbf{x}) = \arg\max_{a \in \mathcal{A}} Q(\mathbf{x}, a; \theta), \tag{3}$$

can be used to select actions to solve the task.

B. Potential Fields

For planar navigation tasks, it is a common practice to model the effort or cost of passing through a point $\mathbf{p} \in \mathbb{R}^2$ by a potential field $U\colon \mathbb{R}^2 \to \mathbb{R}$ where higher potential means more effort required [29]. For identifying locally optimal motion directions $\theta \in [0, 2\pi]$ at a point $\mathbf{p} \in \mathbb{R}^2$, we can consider the directional derivative $\nabla_{\mathbf{v}_\theta} U(\mathbf{p})$ along the vector $\mathbf{v}_\theta = [\sin\theta, \cos\theta]^\intercal$.

For simplicity, the potential field U is often defined as a mixture of potential functions U_i, representing individual features of the environment,

$$U(\mathbf{p}) = \frac{1}{N}\sum_{i=0}^{N} U_i(\mathbf{p}). \tag{4}$$

In *Gaussian potential fields*, obstacles are modeled by the normal distribution function, $U_i(\mathbf{p}) = \varphi(\mathbf{p}; \boldsymbol{\mu}, \boldsymbol{\Sigma})$, leading to a smooth potential surface. If the potential is independent for each dimension, i.e. the covariance matrix is diagonal, $\boldsymbol{\Sigma} = \operatorname{diag}(\sigma_x, \sigma_y)$, the potentials U_i can be factorized,

$$U_i(\mathbf{p}) = \varphi(\mathbf{p}_x; \boldsymbol{\mu}_x, \sigma_x)\, \varphi(\mathbf{p}_y; \boldsymbol{\mu}_y, \sigma_y), \tag{5}$$

where subscript x and y are used to refer to dimensions one and two respectively. We use both, the normal distribution function φ and the skew-normal distribution function φ_α with shape parameter α for modeling obstacles.

IV. LEARNING NONPREHENSILE REARRANGEMENT

We learn nonprehensile rearrangement using Q-learning where the Q-function is approximated by a deep convolutional neural network. To train this network, we define rewards that model the task and alternate between collecting episodes of experiences and updating network parameters. Effective deep Q-learning requires both, informative and task-relevant experiences, and adequate utilization of past experiences. Below, we explain how we collect informative experiences by informed action sampling and how we utilize both failure and success in learning by sampling the replay buffer. The process is summarized in Alg. 1.

A. Network Structure

We define a deep convolutional neural network that computes the action-value function for each action $a \in \mathcal{A}$ in parallel. The input of the network is one observation $\mathbf{x}$ with 128×128 RGB pixels and the output is the Q-values for actions. As seen in Fig. 3, there is a convolutional part for learning a low-dimensional representation followed by a fully connected part for mapping to action values. The

Algorithm 1 Learning Architecture

```
1:  Randomly initialize primary and target networks $\theta^p = \theta^t$
2:  Initialize experience buffer $D$
3:  for episode $k = 1, 2 \ldots, K$ do
4:    for time step $t = 1, 2, \ldots$ until termination do
5:      if with probability $P_{\text{exploit}}$ then
6:        $a_t \leftarrow \arg\max_{a \in \mathcal{A}} Q(\mathbf{x}, a; \theta^t)$
7:      else
8:        Sample action $a_t \sim P_{\mathcal{A}}(a \mid \mathbf{p}^{\text{tool}})$        ▷ Sec. IV-C
9:      end if
10:     Execute $a_t$
11:     Get experience $e_t = (\mathbf{x}_t, a_t, r_t, \mathbf{x}_{t+1})$
12:   end for
13:   Update $D$ according to policy        ▷ Sec. IV-D.1
14:   Sample experiences $e \sim \text{Uniform}(D)$
15:   Update $\theta^p$ and $\theta^t$ according to policy        ▷ Sec. IV-D.2
16: end for
```

convolutional part consists of six convolutional layers with Rectified Linear Unit (ReLU) as activation function to extract the feature map, and four max pooling layers to reduce the size of the output. This network structure is instantiated twice for the target network and the primary network respectively.

B. Reward

In reinforcement learning, the reward implicitly specifies what the agent is encouraged to do. Therefore, it is important that the reward models the task correctly. We want to relocate the manipulation object to the target location by moving the manipulation tool but without obstacle collision. For this, we define the reward given an episode of experiences of length T using three components r_t^{tool}, r_t^{man}, and r_t^{term}. The first component,

$$r_t^{\text{tool}} = (\|\mathbf{p}_{t-1}^{\text{tool}} - \mathbf{p}_{t-1}^{\text{man}}\| - \|\mathbf{p}_t^{\text{tool}} - \mathbf{p}_t^{\text{man}}\|)/d_a, \quad (6)$$

increases when the tool and the manipulation object get closer. The second component,

$$r_t^{\text{target}} = (\|\mathbf{p}_{t-1}^{\text{man}} - \mathbf{p}_{t-1}^{\text{target}}\| - \|\mathbf{p}_t^{\text{man}} - \mathbf{p}_t^{\text{target}}\|)/d_a, \quad (7)$$

increases when the manipulation object and the target location get closer. Finally, we have to capture success or failure which only occurs at the end of the episode at time step T. In case of success, the manipulation object reaches the target location, $\|\mathbf{p}_T^{\text{man}} - \mathbf{p}_T^{\text{target}}\| < \epsilon^{\text{suc}}$. The episode is terminated with failure when too many steps have been taken, obstacles are moved (collision), or the tool moves out of the work-surface. We model these conditions by the following terminal reward,

$$r_T^{\text{term}} = \begin{cases} 1, & \|\mathbf{p}_T^{\text{man}} - \mathbf{p}_T^{\text{target}}\| < \epsilon^{\text{suc}} \\ -1, & \|\mathbf{p}_T^{\text{obs},i} - \mathbf{p}_0^{\text{obs},i}\| > \epsilon^{\text{fail}} \\ -1, & \mathbf{p}_T^{\text{tool}} \text{ out of work-surface} \\ -1, & T = n^{\text{steps}} \quad \text{(timeout)} \end{cases}, \quad (8)$$

which is 0 for all steps $t < T$. The last reward captures the main essence of the task but is an infrequent experience.

All three rewards defined above are combined in a weighted sum with α_1, α_2 and α_3 being the weighing factor:

$$r_t = \alpha_1 r_t^{\text{tool}} + \alpha_2 r_t^{\text{target}} + \alpha_3 r_t^{\text{term}}, \quad (9)$$

to form our reward feedback.

C. Heuristic Exploration with Informed Action Sampling

Reinforcement learning for our rearrangement task is challenging because exploration can lead to preemptive termination of an episode. For example, when the manipulation tool is close to obstacles, uniformly sampling the next action is often a poor choice leading to collisions as seen as red dots in Fig. 4. Doing so nevertheless ultimately results in an unbalanced training set dominated by unsuccessful episodes. When exploring, we therefore aim at selecting actions that are unlikely to prematurely terminate the episode due to obstacle collisions as a means to collect informative samples for the dataset similar to [23, 24].

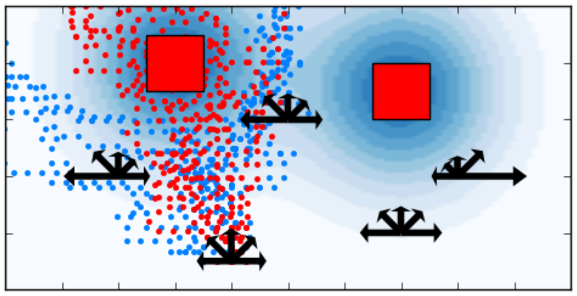


Fig. 4: For exploration, we model the environment with obstacles (red squares) as a Gaussian potential field and sample actions according to local potential changes. This process selects actions leading away from obstacles more frequently than actions leading towards obstacles. Arrow length indicates action probability. For illustration, we sample actions uniformly (red) and according to our distribution (blue) starting from the same position. Red paths lead to collisions more often than blue paths.

1) Action Sampling: We are interested in a complete heuristic for exploration that does not preclude certain types of experiences but want to sample actions such that collisions are infrequent. For this reason, we model the environment by a potential field U, as described in Sec. III-B, and sample exploration actions from a distribution, $a \sim P_{\mathcal{A}}$, which depends on local potential change. This results in a lower frequency of actions moving the tool close to obstacles, which increases potential, and higher frequency of actions moving the tool into an obstacle-free region, which decreases potential as illustrated by blue dots in Fig. 4.

We model $P_{\mathcal{A}}$ by discretizing the space of motion directions θ between $-\frac{1}{8}\pi$ and $\frac{9}{8}\pi$ into five intervals, as shown in Fig. 2, resulting in sectors I_a centered around each action's motion direction. To compute an action's probability $P_{\mathcal{A}}(a \mid \mathbf{p})$ at a point in the environment $\mathbf{p} \in \mathbb{R}^2$, we first integrate potential change in U at position $\mathbf{p}$ over the angle interval I_a,

$$\Delta(a, \mathbf{p}) = \int_{\theta \in I_a} \nabla_{\mathbf{v}_\theta} U(\mathbf{p}) \mathrm{d}\theta, \quad (10)$$

for each action $a \in \mathcal{A}$. Based on the potential changes $\Delta(a, \mathbf{p})$, we formulate the distribution $P_{\mathcal{A}}$ using a normalized exponential function,

$$P_{\mathcal{A}}(a \mid \mathbf{p}) = \frac{\exp(-\Delta(a, \mathbf{p}))}{\sum_{a' \in \mathcal{A}} \exp(-\Delta(a', \mathbf{p}))}, \tag{11}$$

which assigns higher probability to larger instantaneous reduction of potential.

2) Environment Model: For sampling actions according to Eq. (11), we assume that the tool is oriented along the Y-axis and define obstacle potentials U_i consisting of two factors for each obstacle with position $\mathbf{p}^{\text{obs},i}$,

$$U_i(\mathbf{p}) = \varphi(\mathbf{p}_x; \mathbf{p}_x^{\text{obs},i}, \sigma_i)\, \varphi_\alpha(\mathbf{p}_y; \mathbf{p}_y^{\text{obs},i}, \sigma_i), \tag{12}$$

where we use the notation from Sec. III-B. Skewing the potential along the Y-axis makes the potential steeper when the tool is before the obstacle leading to stronger emphasize on avoiding collisions.

D. Experience Replay and Network Updates

The stability-plasticity dilemma and correlation of experience [22] in deep Q-learning are usually addressed by uniformly sampling experiences from the replay buffer D of previous experiences [22, 30–32] for training. However, until the task has been sufficiently learned the majority of experiences would come from failed episodes, e.g., the manipulation tool did not catch the object, the motion caused collisions or the motion did not lead to the goal region. In our experience, this leads to slow learning in our task.

For effective training, sampled experiences need to be informative and representative, which in our experience means that they should come from successful *and* failing episodes in equal shares. Additionally, when learning a task with high-dimensional observations, not all experiences can be collected in the buffer D and adding new experiences displaces older ones. Therefore, we propose a policy for data sampling and storing and a policy for network updates.

1) Replay Buffer Policy: The overall goal is to avoid over-representing failed or successful episodes in training data. For this, we store experiences in D according to variable probabilities P_{store}. If the ratio of successful experiences in the buffer is far away from 50%, e.g. less than 30%, we use a higher storing probability P_{store} for successful experiences than for failing experiences. If ratio is more than 70%, we do the opposite. If the buffer is full, the oldest experience is displaced by the newly added experience.

2) Network Update Policy: Updating the network parameters with experiences from a dataset biased towards failing episodes leads to poor performance on the task. Therefore, we update the network according to the dataset's condition. If the ratio of success experiences deviates into any direction from 50%, we slow down the network update in terms of the deviation magnitude. The schedule based on the ratio of success experiences r^{succ} shown below realizes this concept:

$$\text{Update action:}$$
$$\begin{cases} \text{Update with small probability,} & \epsilon_1 < |r^{\text{succ}} - 0.5| \\ \text{Update once,} & \epsilon_2 \leq |r^{\text{succ}} - 0.5| < \epsilon_1 \\ \text{Update multiple times,} & |r^{\text{succ}} - 0.5| < \epsilon_2 \end{cases}$$

where ϵ_i are the update control points. Whenever we update the primary network, we update the target network parameters θ^t according to the primary network's parameters θ^p using a low learning rate of 0.001,

$$\theta^t \leftarrow 0.999\, \theta^t + 0.001\, \theta^p, \tag{13}$$

which leads to slow adaptation but increases learning stability.

V. EXPERIMENTS

In this section, we present experiment setup, data collection, model training and evaluation. We quantitatively evaluate the DQN trained using our approach to show that it can handle the given task with high success rate. Additionally, we provide qualitatively examples that demonstrate how our approach reacts to sudden changes from external influences and how it generalizes to slight changes of physical properties.

A. Experiment Platform and Setup

The experiments are conducted with a Baxter robot in a simulated virtual environment using Gazebo [33]. The simulation considers physical properties such as mass, friction, and velocities, but these are not known to the robot. A customized manipulation tool is mounted on the left hand of Baxter as seen in Fig. 2. The robot only controls its left arm to interact with the environment. The manipulation object and obstacle objects are represented by cube-shaped objects. For perception, we simulate a fixed camera beside the robot as shown in Fig. 1. We define the work-surface to be 30 by 50 cm. The system parameters are empirically determined in terms of both the performance and our computation resource limits as listed in Table I.

TABLE I: System Parameters

Parameter	Notation	Value
Primary-Net Learning Rate	η	10^{-4}
Replay Buffer Size	$\lvert D\rvert$	$200,000$
Discount Factor	γ	0.99
Episode Limit	n^{steps}	150
Reward Weights	$\alpha_1,\ \alpha_2,\ \alpha_3$	$0.1, 0.2, 1$
Update Policy	$\epsilon_1,\ \epsilon_2$	$0.4, 0.1$
ε-greedy	$K_1,\ K_2$	$200, 5000$
Action Scale	d_a	1cm

B. Data Collection

For each training episode, we initialize the robot in the starting pose and randomly place the manipulation object in front of the manipulation tool. The number of obstacles is fixed to 2 for data collection. The obstacles are placed randomly while at least one obstacle is directly placed

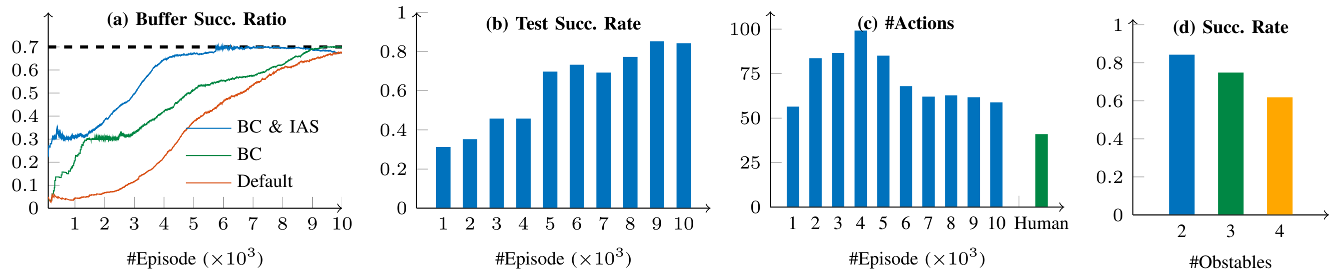


Fig. 5: (a) The ratio of success episodes in the replay buffer. BC: Replay Buffer Control. IAS: Informed Action Sampling. (b) The success rate against the number of experienced episodes. (c) The average number of actions taken to accomplish a random task. (d) The success rate in test scenes with 2, 3 and 4 random obstacles.

between the manipulation object and the target location making obstacle avoidance necessary. We set the maximal episode length to n^{steps} and proceed according to Alg. 1 to select actions and update the network.

Exploiting with a poor initial training policy rarely leads to successful episodes. Therefore, we tradeoff between *exploration* and *exploitation* using an ϵ-greedy training schedule with three phases [22]: At the beginning of training, the convolutional part of the network is not well trained, so 1) we employ *only exploration* (Sec. IV-C) for K_1 episodes to train state perception; 2) after this phase, we increase the exploitation probability for each episode until episode K_2 and 3) thereafter, we only train with exploitation for learning the state-action function. This is summarized below as the the exploration probability P_{exploit} for episode number k,

$$P_{\text{exploit}} = \begin{cases} 0, & 0 < k \leq K_1 \\ \tau k, & K_1 < k \leq K_2 \\ 1, & k > K_2 \end{cases}, \tag{14}$$

where $\tau \in \mathbb{R}^+$ is a factor which controls the probability P_{exploit} to linearly increase from 0 to 1 in the corresponding range.

C. Network Training

While collecting experiences as aforementioned, we train the deep Q-network *de novo* in terms of objective function Eq. (2) with the Adam optimizer [34]. The mini-batch size is set to 32. In order to evaluate the proposed approach, we train the network using 3 different configurations: 1) The network is trained using the replay buffer control (Sec. IV-D) and the informed action sampling (Sec IV-C). 2) The network is trained using only buffer control. 3) The network is trained without any of proposed methods. The training process took approximately 600k actions during which 10k episodes were collected for each of the 3 configurations. The training hardware is a single Nvidia GeForce GTX 1080 Ti GPU. More than 90% of training time is spent on simulation.

D. Quantitative Experiments

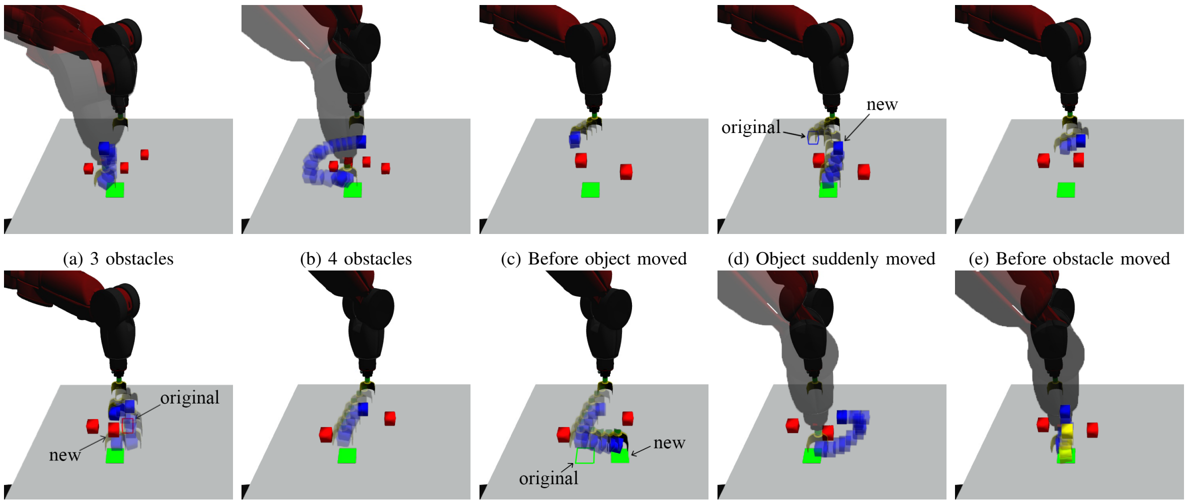
1) Reply Buffer Control and Informed Action Sampling: For evaluating the effectiveness of these two techniques, we record the ratio of success episodes in the reply buffer during the training process of the aforementioned 3 training configurations. As shown in Fig. 5(a), the default configuration performs poorly until $7,000$ episodes to achieve 50%, which has been achieved at the episode $5,000$ by adding the buffer control. By additionally applying informed action sampling, the buffer achieves a balanced share already at the episode $3,000$. This result clearly shows the effectiveness of our proposed methods. Furthermore, as explained below, it is crucial to collect sufficient success experiences for training, since it significantly affects the training results.

2) General Performance: During training, we save the network parameters once every $1,000$ episodes and evaluate its performance using 300 random scenes. As shown in Fig. 5(b), the success rate increases while the network experienced more episodes and is stable and converged around the episode $10,000$. We can observe a rapid increment at the episode $5,000$ where the success episodes in the reply buffer reached the upper limit of 70%. This implies that sufficient success experiences in the reply buffer is crucial for increasing the network's performance. Finally, we achieve a success rate of 85% indicating that the learned network can effectively handle the task of nonprehensile rearrangement.

3) Action Effectiveness: For the training process of the paragraph above, Fig. 5(c) shows the number of actions needed to complete a random task. It can be observed that less actions are needed in the beginning of training. This is because in the beginning the network only succeeds in very simple scenes which do not require many actions. After experiencing $4,000$ episodes, the number of actions starts to decrease since the network has further optimized the reward outputs to make the actions more effective.

Additionally, we involve a human subject to be tested with the same input as the robot to make action decisions by pressing arrow keys to control the end-effector in the 2d workspace. The result in Fig. 5(c) shows that the human performed a little better in terms of the number of actions. This is because our network is more conservative than the human in collision avoidance and tends to keep away from obstacles. However, this also shows that the effectiveness of our network is comparable to a human as it does not take many more actions to achieve the same tasks.

4) Number of Obstacles: Although we train the network using only 2 random obstacles, we test it using also 3 and 4 obstacles in 300 random scenes. As shown in Fig. 5(d), the performance deteriorates when more obstacles are involved. However, the network is still able to handle most of the



(a) 3 obstacles (b) 4 obstacles (c) Before object moved (d) Object suddenly moved (e) Before obstacle moved

(f) Obstacle suddenly moved (g) Before target moved (h) Target suddenly moved (i) Low-friction (j) Distraction object

Fig. 6: Qualitative experiments to investigate the robustness of the network. (a-b) Example executions when 3 or 4 obstacles were randomly positioned. (c-h) Reactive path re-planning when the manipulation object, obstacles or the target positions were suddenly moved. (i) Reactive action planning in a low-friction environment. (j) Example execution when a distraction object (yellow) was involved.

scenes. Example solutions generated by our network are shown in Fig. 6(a-b). We interpret this as that the network learns not only global features to find the path of moving from the start position to the target position, but also local features to avoid collisions. We note that the failures in scenes with 3 and 4 obstacles can happen sometimes due to the target being fully blocked by randomly placed obstacles which do not allow for completing the task.

E. *Qualitative Experiments*

One of the most important advantages of a learned policy over a classic physics-based planning algorithm is that the final behavior naturally reacts to unexpected changes in the environment without the need of explicit re-planning. Below, we test robustness of our approach by moving objects and adding distractors during execution, as well as setting the friction coefficient different from training.

1) Object Sliding, Obstacles Moving and Target Moving: As shown in Fig. 6(c-d), while the manipulation tool is approaching the manipulation object, we suddenly change the position of the manipulation object. Still, our approach can finish the task. Additionally, in Fig. 6(e-f), we suddenly change the position of one of the obstacles to block the direct path. Again, our approach completes the task. Moreover, as seen in Fig. 6(g-h), we suddenly change the target position when the robot is just about to complete the task. Here, our approach reaches the new target position.

2) Low-friction Environment: As another test, we significantly decrease the friction coefficient between the manipulation object and the table surface, such that the object will slide to some direction after each action. In Fig. 6(i), we can see that although the object path is jittering during the execution, the approach still completes the task. This example is also presented in the complementary video.

3) Distraction: As shown in Fig. 6(j), when there is an distraction object (yellow) in the environment, the behavior is not affected by it and can still complete the task. However, it is worthwhile to note that the distraction object is pushed. This indicates that the network focuses on the relevant information from the inputs, but that it does not guarantee collision-free manipulation with new, unknown objects.

VI. CONCLUSION

In this work, we have formulated nonprehensile manipulation planning as a reinforcement learning problem. Concretely, we modeled the task with relevant rewards and trained a deep Q-network to generate actions based on the learned policy. Additionally, we proposed *replay buffer control* as well as potential field-based *informed action sampling* for efficient training data collection to facilitate the network convergence.

We quantitatively evaluated the trained network by testing its success rate at different training stages, the results showed that the performance of the network was steadily improved, and that the network training was significantly affected by the ratio of success episodes in the reply buffer. After the network is converged, it achieved a success rate of 85% implying that it has learned how to handle the task. The average number of actions needed to complete a task has shown that the network was able to optimize its reward outputs to improve the action effectiveness. In comparison to a human subject, we can conclude that the network has achieved comparable performance to the human while it is more conservative in path planning for collision avoidance. Additionally, we have qualitatively shown that the network is reactive and adaptive to uncertainties, such as the sudden changes of objects and target positions, low-friction coefficients and distraction objects.

In future work, we plan to adapt the behaviors learned in simulation to a real environment, where we have no knowledge about physical properties of the objects and the lighting conditions. For this, we also need to enable the network to learn how to adapt the image input from a real camera to an image that can be used by the deep Q-network trained in simulation. Additionally, we would like to integrate more sensors, such as tactile and depth sensors, into our system to enable the cross-modal sensing ability for the system to better understand the task space, so as to more robustly handle the uncertainties in the real world.

ACKNOWLEDGEMENT

This work was supported by the HKUST SSTSP project RoMRO (FP802), HKUST IGN project IGN16EG09, HKUST PGS Fund of Office of Vice-President (Research & Graduate Studies) and Knut and Alice Wallenberg Foundation & Foundation for Strategic Research.

REFERENCES

[1] A. Cosgun, T. Hermans, V. Emeli, and M. Stilman, "Push planning for object placement on cluttered table surfaces," in *Proc. IEEE/RSJ Int. Conf. Intelligent Robots and Systems*, 2011.

[2] M. Dogar, K. Hsiao, M. Ciocarlie, and S. Srinivasa, "Physics-based grasp planning through clutter," in *Robotics: Science and Systems*, 2012.

[3] M. Gupta and G. Sukhatme, "Using manipulation primitives for brick sorting in clutter," in *Proc. IEEE Int. Conf. Robotics and Automation*, 2012.

[4] M. Stilman and J. Kuffner, "Navigation among movable obstacles: Real-time reasoning in complex environments," in *Proc. IEEE-RAS Int. Conf. Humanoid Robots*, 2004.

[5] K. Hang, J. A. Stork, N. S. Pollard, and D. Kragic, "A framework for optimal grasp contact planning," *IEEE Robotics and Automation Letters*, vol. 2, no. 2, pp. 704–711, 2017.

[6] K. Hang, M. Li, J. A. Stork, Y. Bekiroglu, F. Pokorny, A. Billard, and D. Kragic, "Hierarchical fingertip space: A unified framework for grasp planning and in-hand grasp adaptation," *IEEE Transactions on Robotics*, vol. 32, no. 4, pp. 960–972, 2016.

[7] K. Hang, J. A. Stork, F. Pokorny, and D. Kragic, "Combinatorial optimization for hierarchical contact-level grasping," in *Proc. IEEE Int. Conf. Robotics and Automation*, pp. 381–388, IEEE, 2014.

[8] K. Hang, J. A. Stork, and D. Kragic, "Hierarchical fingertip space for multi-fingered precision grasping," in *Proc. IEEE/RSJ Int. Conf. Intelligent Robots and Systems*, pp. 1641–1648, IEEE, 2014.

[9] T. Siméon, J.-P. Laumond, J. Cortés, and A. Sahbani, "Manipulation planning with probabilistic roadmaps," *The International Journal of Robotics Research*, vol. 23, no. 7-8, pp. 729–746, 2004.

[10] M. Stilman, J.-U. Schamburek, J. Kuffner, and T. Asfour, "Manipulation planning among movable obstacles," in *Proc. IEEE Int. Conf. Robotics and Automation*, 2007.

[11] J. A. Haustein, J. King, S. S. Srinivasa, and T. Asfour, "Kinodynamic randomized rearrangement planning via dynamic transitions between statically stable states," in *Proc. IEEE Int. Conf. Robotics and Automation*, 2015.

[12] J. E. King, J. A. Haustein, S. S. Srinivasa, and T. Asfour, "Nonprehensile whole arm rearrangement planning on physics manifolds," in *Proc. IEEE Int. Conf. Robotics and Automation*, 2015.

[13] J. E. King, M. Cognetti, and S. S. Srinivasa, "Rearrangement planning using object-centric and robot-centric action spaces," in *Proc. IEEE Int. Conf. Robotics and Automation*, 2016.

[14] J. E. King, V. Ranganeni, and S. S. Srinivasa, "Unobservable monte carlo planning for nonprehensile rearrangement tasks," in *Proc. IEEE Int. Conf. Robotics and Automation*, 2017.

[15] G. Wilfong, "Motion planning in the presence of movable obstacles," *Annals of Mathematics and Artificial Intelligence*, vol. 3, no. 1, pp. 131–150, 1991.

[16] D. Schiebener, J. Morimoto, T. Asfour, and A. Ude, "Integrating visual perception and manipulation for autonomous learning of object representations," *Adaptive Behavior*, vol. 21, no. 5, pp. 328–345, 2013.

[17] J. Zhou, R. Paolini, A. M. Johnson, J. A. Bagnell, and M. T. Mason, "A probabilistic planning framework for planar grasping under uncertainty," *IEEE Robotics and Automation Letters*, vol. 2, no. 4, pp. 2111–2118, 2017.

[18] N. Fazeli, R. Tedrake, and A. Rodriguez, "Identifiability analysis of planar rigid-body frictional contact," in *International Symposium on Robotics Research (ISRR)*, 2015.

[19] A. Cosgun, T. Hermans, V. Emeli, and M. Stilman, "Push planning for object placement on cluttered table surfaces," in *Proc. IEEE/RSJ Int. Conf. Intelligent Robots and Systems*, 2011.

[20] M. Dogar and S. Srinivasa, "A framework for push-grasping in clutter," in *Robotics: Science and Systems*, 2011.

[21] R. S. Sutton and A. G. Barto, *Introduction to Reinforcement Learning*. MIT Press, 1st ed., 1998.

[22] 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, pp. 529–533, 2015.

[23] Q. Zhu, Y. Yan, and Z. Xing, "Robot path planning based on artificial potential field approach with simulated annealing," in *Sixth International Conference on Intelligent Systems Design and Applications*, 2006.

[24] A. Varava, K. Hang, D. Kragic, and F. Pokorny, "Herding by caging: a topological approach towards guiding moving agents via mobile robots," in *Proceedings of Robotics: Science and Systems*, 2017.

[25] J. N. Tsitsiklis and B. Van Roy, "Analysis of temporal-diffference learning with function approximation," in *Advances in neural information processing systems*, pp. 1075–1081, 1997.

[26] J. L. McClelland, B. L. McNaughton, and R. C. O'reilly, "Why there are complementary learning systems in the hippocampus and neocortex: insights from the successes and failures of connectionist models of learning and memory.," *Psychological review*, vol. 102, no. 3, p. 419, 1995.

[27] J. ONeill, B. Pleydell-Bouverie, D. Dupret, and J. Csicsvari, "Play it again: reactivation of waking experience and memory," *Trends in neurosciences*, vol. 33, no. 5, pp. 220–229, 2010.

[28] L.-J. Lin, "Reinforcement learning for robots using neural networks," tech. rep., Carnegie-Mellon Univ Pittsburgh PA School of Computer Science, 1993.

[29] H. M. Choset, *Principles of robot motion: theory, algorithms, and implementation*. MIT press, 2005.

[30] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *arXiv preprint arXiv:1509.02971*, 2015.

[31] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized experience replay," *arXiv preprint arXiv:1511.05952*, 2015.

[32] S. Adam, L. Busoniu, and R. Babuska, "Experience replay for real-time reinforcement learning control," *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 42, no. 2, pp. 201–212, 2012.

[33] N. Koenig and A. Howard, "Design and use paradigms for gazebo, an open-source multi-robot simulator," in *Proc. IEEE/RSJ Int. Conf. Intelligent Robots and Systems*, 2004.

[34] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.