End-to-End Nonprehensile Rearrangement with Deep Reinforcement Learning and Simulation-to-Reality Transfer

Weihao Yuan[a], Kaiyu Hang[b], Danica Kragic[c], Michael Y. Wang[a], Johannes A. Stork[d,*]

[a]*Robotics Institute, ECE, Hong Kong University of Science and Technology, Hong Kong*
[b]*Mechanical Engineering and Material Science, Yale University, New Haven, Connecticut, USA*
[c]*Centre for Autonomous Systems, EECS, KTH Royal Institute of Technology, Stockholm, Sweden*
[d]*Centre for Applied Autonomous Sensor Systems, rebro University, rebro, Sweden*

Abstract

Nonprehensile rearrangement is the problem of controlling a robot to interact with objects through pushing actions in order to reconfigure the objects into a predefined goal pose. In this work, we rearrange one object at a time in an environment with obstacles using an end-to-end policy that maps raw pixels as visual input to control actions without any form of engineered feature extraction. To reduce the amount of training data that needs to be collected using a real robot, we propose a simulation-to-reality transfer approach. In the first step, we model the nonprehensile rearrangement task in simulation and use deep reinforcement learning to learn a suitable rearrangement policy, which requires in the order of hundreds of thousands of example actions for training. Thereafter, we collect a small dataset of only 70 episodes of real-world actions as supervised examples for adapting the learned rearrangement policy to real-world input data. In this process, we make use of newly proposed strategies for improving the reinforcement learning process, such as heuristic exploration and the curation of a balanced set of experiences. We evaluate our method in both simulation and real setting using a Baxter robot to show that the proposed approach can effectively improve the training process in simulation, as well as efficiently adapt the learned policy to the real world application, even when the camera pose is different from simulation. Additionally, we show that the learned system not only can provide adaptive behavior to handle unforeseen events during executions, such as distraction objects, sudden changes in positions of the objects, and obstacles, but also can deal with obstacle shapes that were not present in the training process.

Keywords: Nonprehensile Rearrangement, Deep Reinforcement Learning, Transfer Learning

1. INTRODUCTION

The skill of object rearrangement under the presence of obstacles is essential for object manipulation in cluttered and unstructured environments [1, 2, 3, 4]. However, the classical pick-and-place approaches to object rearrangement fundamentally rely on three further and nontrivial skills: grasping objects [5, 6, 7, 8], motion planning [9, 10], and placing objects [11]. For successful rearrangement, the object has to be secured in the robot's manipulator considering mechanical grasp stability, then it has to be moved around the obstacles considering the forces acting on the object, and finally, it has to be released at the target location making sure that it remains standing upright. This not only requires that the object, for its weight and size, is graspable but also depends on small uncertainties about the object's shape and pose for synthesizing grasps and motions, which renders the pick-and-place approach to object rearrangement into a challenging problem.

In many applications, it is however more prudent to assume that perception is uncertain and incomplete, and that reliable grasping for transport is virtually unattainable. In such a scenario, it is more meaningful to instead leverage nonprehensile actions in the form of pushing and sweeping actions [1, 3, 12, 13, 14, 15]. Such multi-contact interactions are often observed in human manipulation behavior because they are more robust and reliable under uncertainty [16, 12, 17]. Subduing the difficult problems of grasp and motion synthesis in this way, however, raises the problem of creating complex robot-object interactions which are difficult to model and plan. For this reason nonprehensile rearrangement is usually addressed under simplifying assumptions such as complete observability or known object geometry [18, 15].

Although classical planning-based approaches can address this problem via careful modeling of interactions, estimating physical properties, and reconstructing states from perception data, we propose to consider nonprehensile rearrangement as a model-free, end-to-end learning

*Corresponding author
Email addresses: wyuanaa@connect.ust.hk (Weihao Yuan), kaiyu.hang@yale.edu (Kaiyu Hang), dani@kth.se (Danica Kragic), mywang@ust.hk (Michael Y. Wang), johannes.stork@oru.se (Johannes A. Stork)

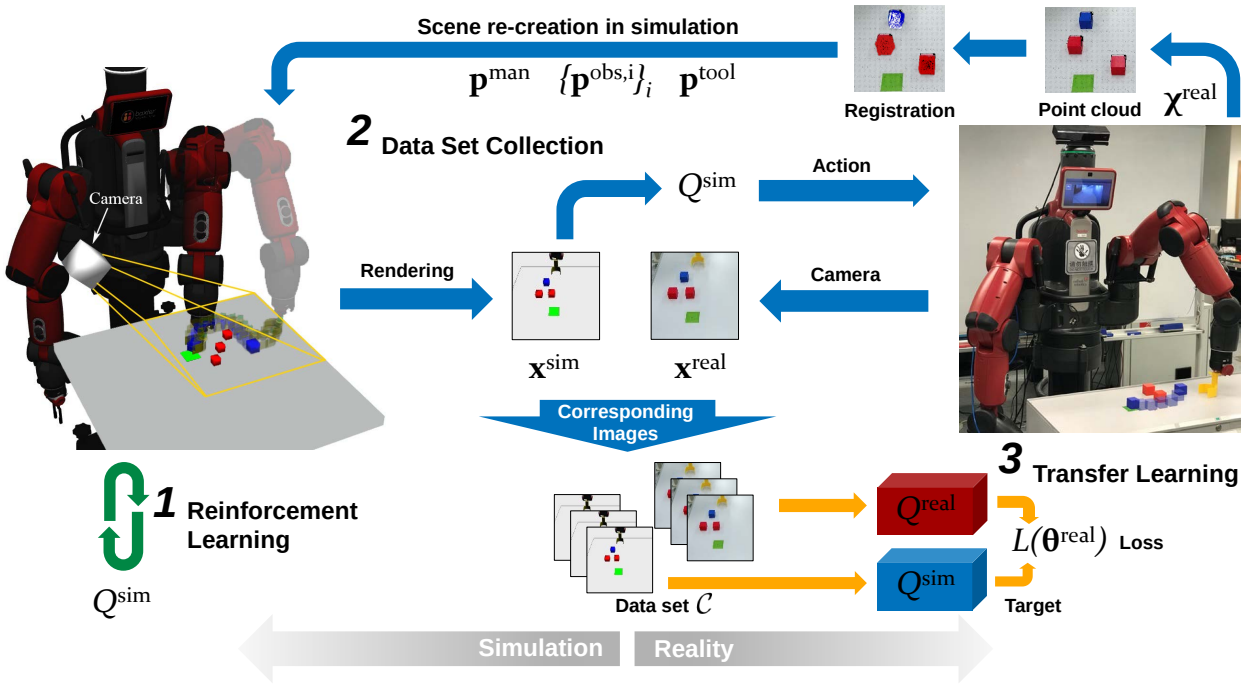


Figure 1: The real robot is tasked to first find and then push an object (blue) around obstacles (red) to a goal region (green) relying on only visual feedback. Our approach achieves this in three steps: (1) We first train the robot in simulation with reinforcement learning (green arrows). (2) Then, we collect a data set of corresponding images by commanding the real robot using a re-created scene in simulation (blue arrows). (3) In the last step, we use the corresponding images in supervised learning to transfer the policy from simulation to reality (orange arrows).

problem. Using reinforcement learning, we can leverage the robot's own experience. Similarly to other learning-based manipulation systems [19], this approach removes extensive engineering of perception and reasoning components and allows adapting the system to changing conditions by learning without re-modeling of components.

Deep reinforcement learning policies have recently shown a high level of performance on end-to-end visual control problems, however, this required in the order of hundreds of thousands of data points to succeed [20]. The collection of such an amount of training data in real-world settings usually depends on simple automatic reset of the task, additional instrumentation for feedback signals, and a highly parallelized training setup [21, 22]. For this reason end-to-end reinforcement learning is often only demonstrated in virtual tasks such as computer games and simulated systems [20, 23, 24, 25].

In our task of nonprehensile rearrangement, automated training data collection would have to place obstacles at random positions and transport the manipulation object back to the starting location for every reset. This would require to automate a problem of the type we want to solve in the first place, which is the reason why we deem fully automated data collection impractical for our purpose. Since we are still interested in a robot performing nonprehensile rearrangement in reality, we instead draw on the concept of transfer learning [26, 27]. Concretely, we train the rearranging policy in simulation where training experience can be obtained at low cost and then transfer the learned policy to real-world input data.

To this end, we model the nonprehensile rearrangement task in a general purpose robot simulator [28] with close resemblance to reality in terms of visual data and physical behavior of robot and objects. After training the rearrangement policy in simulation, we collect a comparatively small amount of real-world data in a calibrated robot setup. In this process, we use additional sensors and computer vision techniques to extract all relevant information such as object and obstacle positions from real-world data and recreate the perceived situation in simulation. This provides a set of real-world perception data for which we can, with a detour via their simulated recreation, provide the corresponding actions from the previously learned policy as supervision labels. Finally, we use pairs of perception and action labels in supervised learning to adapt the previously learned policy to real-world input data. The different steps of this approach are shown in Figure 1. In our experiments, we explore different ways of adapting and modifying the policy model in the last step.

When learning the end-to-end rearrangement policy in simulation, we employ a reinforcement learning method that is based on fitting the state-action value function with a deep neural network [20]. This is known to be unstable and prone to diverge because of, among others, temporal correlation in training data and the stability-plasticity dilemma. A common remedy is randomizing over the data

[20, 29, 30, 31] combined with periodically updating target values [20]. In our task, however, learning is confronted with another hampering bias in the training data distribution: Random exploration again and again leads to collision with obstacles and predominantly provides experiences of failure in terms of the task. In our experience, this constitutes an impediment for learning which leads to the robot never obtaining a policy that solves the task.

To address this issue, we proposed two methods to achieve a balanced training data set in our previous work [32]. First, we define a task-specific behavior policy for the off-policy reinforcement learning algorithm. This policy uses an environment-based potential field [33, 34] to increase the probability of high-reward actions by reducing the number of collisions but does not preclude collisions completely. Second, we introduce active curation of the training data set which adjusts the data insertion probabilities according to the current ratio of success and failure experiences in the data set. This makes sure that randomized access to the training data set retrieves success and failure experience most equally and results in informative gradients for learning the state-action value function.

For evaluation, we consider a table top rearrangement task where object and obstacles are colored cubes and the target location is indicated by a colored marker as seen in Figure 1. Even though the robot is trained with only two randomly placed obstacles, the learned policy can deal with up to four obstacles at the same time. When the camera angle is similar, the transferred policy shows similar performance as in simulation. When the camera pose is altered in reality, the transferred policy can still execute the task with a small decrease in performance. Different from classical open-loop planning, our approach produces a closed-loop policy that can react to unpredictable changes such as moving and appearing obstacles or variable friction without the need of computationally expensive re-planning.

In this paper, we integrate our past work on learning the nonprehensile rearrangement task in simulation [32] (described in Section 4 and Section 6) as well as our additional efforts with regard to transferring the learned policy to reality (Section 5, Section 7). Our contributions are thus

1. modeling the nonprehensile rearrangement task as a reinforcement learning problem,
2. enabling learning the task in simulation with
   (a) heuristic action sampling for exploration and
   (b) active training data set curation,
3. transferring the learned policy to a real robot with supervised examples, and
4. evaluation of different ways to transfer the learned policy by adapting and modifying the policy model.

After presenting related literature in Section 2, we formulate the problem and state our assumptions in Section 3. In Section 4 we describe how we learn the policy in simulation and explain in Section 5 how we transfer the policy to reality. Section 6 and 7 contain experimental evaluation in simulation and reality, respectively. Our concluding comment are found in Section 8.

2. RELATED WORK

In this section, we first review the literature about nonprehensile manipulation for rearrangement using classical methods and analyze their limitations. Then, we survey learning-based solutions for robotic manipulation without transfer between simulation and reality. Finally, we discuss works with real-world robots that use domain transfer learning and compare them to our transfer approach.

Most works based on nonprehensile manipulation for rearrangement problems employ classic methods. For instance, Haustein *et al.* [15, 35, 36] used sampling-based planning. In their work, they use a free-floating end-effector to avoid expensive planning in configuration space and rely on a black-box physics model for planning with kinodynamic-RRT [35]. The algorithm searches for dynamic transitions between statically stable states and can exploit interaction with the environment. Similar to our approach, they assume quasi-static dynamics for planar pushing and project the actions to a constraint manifold parallel to the support surface [15]. Different to our approach, the actions space is separated in so-called object-centric and robot-centric actions [36].

Different from the closed-loop planning approaches, King *et al.* [37] proposed a Monte Carlo Tree Search-based method to create open-loop trajectories. Their algorithm uses learning from demonstration and plans in a simplified subspace in order to produce more useful trajectories as compared to random sampling which accelerates the search process.

However, nearly all these methods [15, 35, 36] assume complete observability of the state from visual perception for planning, which is impractical or even impossible in most of applications. In addition, to deal with the complex dynamics of interacting with physical bodies, physical properties are often assumed to be known to keep planning of action sequences tractable. Similar to our work, dynamics are often simplified by assuming a quasi-static model [38, 39] to reduce complexity, which conveniently allows solutions based on motion primitives [40, 18]. Moreover, these planning-based methods are time-consuming, such that they are practically only used for static environment and are unable to do real-time re-planning when the scene is changed from external influences.

To address the stated issues, some researchers applied learning methods to robotic manipulation [19, 41, 42, 43, 32, 44, 45, 46, 47]. For instance, [19, 41, 42] used supervised and unsupervised learning methods to learn grasping from a large number of data, while [43] learned tasks from demonstrations. However, their tasks were relatively simple and the environment was static. For pushing tasks, [46, 47] used visual model predictive control but did not

automatically identify and consider the obstacles. If training is directly done on a real robot, examples are difficult and expensive to collect. This was addressed by training with multiple robots in parallel [22] and demonstrations of the task [48]. The first option requires that the task is easy to reset and the second option can introduce human bias. When learning is done in simulation to have easier access to more data, the resulting system is not applied to reality [32], or the system has to forgo access to raw sensor data to avoid the gap between simulation and reality [44, 45].

A better solution to this problem is training in simulation and then transferring to reality. There are two dominant ways to do domain transfer from simulation to reality: One is reducing the discrepancy between simulation and reality, and the other is learning to extract features that are invariant for different domain such that they work in both domains. For the latter concept, some works tried to introduce variance into the simulation environment by employing randomization [49, 27, 50]. This increases the robustness of the learned policy, which then allows the robot to still perform in reality. We evaluate such approach as a baseline in our experiments. Other works translated simulated perception data to realistic sensor data using Generative Adversarial Networks (GANs) [51, 52, 53], to make the gap between training data in simulation and real-world measurements small. For the latter approach, some works introduced confusion loss to confuse the network when training [54], while others transmitted the hidden activations of one network to another [55]. The work of [56] is most similar to ours: Their approach uses weak pair-wise matching to learn a domain-invariant representation from vision to action space.

Most domain transfer works require that the position and viewing angle of the camera in reality is the same as in simulation. This assumes that the discrepancy is mainly manifested in color features rather than in position or scale of the observed scene. Thus, the transfer can be done in a pixel-to-pixel manner [51, 52, 53, 54, 55, 56]. Our method, however, can work with a distinct camera position, with few aligned data points, and requires only a short training time. Different from [54, 55, 56], our simulation-to-reality transfer operates on state-action values while their approaches transfer in action space. Overall, our work treats perception and planning together without the knowledge of the explicit physical model and dynamics. We leverage on convolution neural network to recognize the scene and understand the task. In our case, deciding on actions is a very fast end-to-end process, which allows adaptive behavior when the environment is changing unexpectedly. To make our system operate in reality, we train the network in simulation, which is tractable and efficient, and then do the transfer from simulation domain to reality. To our knowledge, this is the first time of addressing a real-world nonprehensile rearrangement problem with obstacles utilizing deep reinforcement learning.

3. PROBLEM STATEMENT

In this section, we describe our version of the nonprehensile rearrangement problem, state relevant assumptions, and introduce notation and terminology for the reminder of the paper. At the end of this section, we formalize the policy learning objective in simulation and the transfer learning objective.

3.1. Task and Assumptions

Since we are mainly interested in studying the learning and transfer of a nonprehensile rearrangement policy, we restrict ourselves to a particular instance of the task such that variations in size, shape, weight and other properties of the object, obstacles, and robot do not introduce additional difficulties. Therefore, we consider the situation depicted in Figure 1 where a robot with a nonprehensile manipulation tool is tasked with moving a cube-shaped *manipulation object* on a table top past a set of cube-shaped *obstacles* to a squared visual indicator for the *target location*. Initially, the target location and the manipulation object are situated in the half-space in front of the tool.

In this setting, we assume that the robot can move the manipulation tool parallel to the table top to any location on the surface but keeps the tool always oriented along the table's Y-axis. This removes the need for considering kinematic constraints. We also assume that the robot is capable of pushing the manipulation object effortless across the table and that friction is large enough to render the interaction quasi-static. This removes the need for considering momentum in robot or object motion and allows us to consider rearrangement in discrete time steps. To make the task solvable, we further require that during tests there exists at least one path to the target position that is not fully blocked by obstacles.

Despite the simplifying assumptions stated above, the task is still challenging: First, the physical properties of the environment are unknown, which makes it hard to predict future states. Additionally, the robot's execution of actions in the real world is unreliable and perception of exact poses of objects and obstacles might be occluded. Given that the robot can only perceive the scene from camera images as seen in Figure 1, it is clear that a trivial hard-coded approach, for example a rule-based system, would be hard to design. We also demonstrate in Section 6 that a trivial approach such as moving the gripper with the object on a straight line to the target location is not successful because it leads to collision in over 90% of all cases.

3.2. Definitions and Notations

Solving the task is structured as a sequential decision-making problem, where in each time step t, the robot perceives one *observation* and has to decide on an *action* until the task terminates with *success* or *failure*.

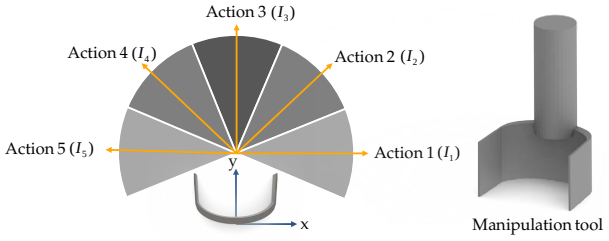


Figure 2: This figure illustrates the 5 predefined motion directions for the manipulation tool. Action 3 is aligned with the front direction Y of the manipulation tool. The sectors depict the ranges in which the potential integrals for heuristic action sampling are calculated, as explained in Sec. 4.3.

Observations. An observation $\mathbf{x} \in \mathbf{X}$ is a 128×128 RGB image (49152 dimensions) taken from a camera pointing at the table top. Examples of observation in simulation and reality are seen in Figure 1. In the images, the manipulation object is seen in *blue*, the obstacles in *red*, the target location in *green*. The robot arm with the attached manipulation tool can occlude parts of the scene. In notation, we differentiate between simulated observations $\mathbf{x}^{\text{sim}}$ and $\mathbf{x}^{\text{real}}$, if necessary.

Actions. An action $a \in \mathcal{A}$ translates the manipulation tool parallel to the table top by a constant offset $d_a \in \mathbb{R}^+$. There are five predefined motion directions as depicted in Figure 2 which allow a large set of trajectories but no backward movement.

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

Success and Failure. An episode only terminates with *success* if the manipulation object reaches the target location. In any other case termination is caused by *failure*. The latter can be caused by *timeout* (i.e. $T_{\max}$ steps are reached), *collision* with an obstacle, or *bound violation* (i.e. moving the tool outside of the work-surface). Bound violation as a failure mode implies that the robot has to learn to keep the tool in a kinematically feasible space.

Grounding Labels. During the learning process in simulation, the algorithm has access to the following 2D positions relative to the table top'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. Based on these we derive predicates for *success* and *failure*.

3.3. Policy Learning Objective

Since our action space is finite, we pose learning the rearrangement policy as a model-free Q-learning problem [57]. The goal of Q-learning is to recover the optimal state-action value function, called Q-function. The optimal Q-function is defined as the discounted cumulative sum of future rewards after making observation $\mathbf{x}$ and taking action a,

$$Q^*(\mathbf{x}, a) = \max_{\pi} \mathbb{E}\left[\sum_{i \geq 0} \gamma^i r_{t+i} \mid \mathbf{x}_t = \mathbf{x}, a_t = a, \right], \quad (1)$$

where the symbol π stands for a policy $\pi: \mathbf{X} \rightarrow \mathcal{A}$. For this formulation, we have to provide a feedback signal r_t that models the nonprehensile rearrangement task.

In Section 4.1 we define the feedback signal r_t in terms of grounding labels and in Section 4.2, we explain how we use the Deep Q-learning algorithm [20] to learn the optimal Q-function in simulation, Q^{sim}.

3.4. Transfer Learning Objective

The optimal Q-function implicitly represents the optimal policy as a greedy action choice [58],

$$a^* = \arg\max_{a \in \mathcal{A}} Q^{\text{sim}}(\mathbf{x}^{\text{sim}}, a). \quad (2)$$

As a consequence, we can transfer the optimal policy by directly transferring Q^{sim} from simulation to Q^{real} in reality. Given pairs $(\mathbf{x}^{\text{sim}}, \mathbf{x}^{\text{real}})$ of observations of the same situation in simulation and reality, this is a supervised learning problem.

In Section 5, we explain how we collect the pairs of corresponding images $(\mathbf{x}^{\text{sim}}, \mathbf{x}^{\text{real}})$ and how we optimize Q^{real} to map to the same value as Q^{sim}. Thereafter, we also discuss different initializations and models of Q^{real} based on Q^{sim}.

4. POLICY LEARNING

In this section, we explain how we learn the nonprehensile rearrangement policy in simulation with the Deep Q-learning algorithm [20]. First, we model nonprehensile rearrangement as a reinforcement learning problem by defining a task-dependent feedback signal and explain our model for the Q-function Q^{sim}. After that, we describe our behavior policy that samples high-reward exploration actions and explain how we obtain informative gradients by curating a balanced training data set. The complete algorithm is shown in Listing 1 below.

4.1. Feedback Signal

In the formalism of reinforcement learning, the transition feedback signal (also called reward) implicitly specifies the task. In our nonprehensile rearrangement problem, we want to use the manipulation tool to move the manipulation object to the target location without colliding with obstacles or leaving the work-surface with the tool within a limited mount of time, as described in Section 3.1. This task can be modeled by a feedback signal r_t^{task} that is positive when the target location is reached and negative in cases where there is a collision, where the tool leaves the

Algorithm 1 Deep Q-Learning

Randomly initialize primary and target networks with $\boldsymbol{\theta}^p = \boldsymbol{\theta}^t$
Initialize experience buffer D
for episode $k = 1, 2 \ldots, K$ **do**
  for time step $t = 1, 2, \ldots$ **until** termination **do**
    if with probability P_{exploit} **then**
      $a_t \leftarrow \arg\max_{a \in \mathcal{A}} Q^{\text{sim}}(\mathbf{x}^{\text{sim}}, a; \boldsymbol{\theta}^t)$
    else
      Sample action $a_t \sim P_{\mathcal{A}}(a \mid \mathbf{p}^{\text{tool}})$ ▷ Sec. 4.3
    end if
    Execute a_t
    Get experience $e_t = (\mathbf{x}_t^{\text{sim}}, a_t, r_t, \mathbf{x}_{t+1}^{\text{sim}})$
  end for
  Curate D with episode E ▷ Sec. 4.4
  Sample experiences $e \sim \text{Uniform}(D)$
  Update $\boldsymbol{\theta}^p$ and $\boldsymbol{\theta}^t$ according to policy ▷ Sec. 4.2.2
end for

work-surface, or where the maximum number of steps is reached.

We define this feedback signal using four different case-based signals r_t^{goal}, r_t^{coll}, r_t^{bound}, and r_t^{time} that only take two different values. For signals r_t^{coll}, r_t^{bound}, and r_t^{time} the value is -1 when the condition is fulfilled while the default value is ∞. In contrast, the signal r_t^{goal} is set to 1 if the condition is fulfilled and 0 otherwise. These definitions allow us to write the feedback signal r_t^{task} as

$$r_t^{\text{task}} = \min \left\{ r_t^{\text{goal}}, r_t^{\text{coll}}, r_t^{\text{bound}}, r_t^{\text{time}} \right\}, \tag{3}$$

such that we result with reward 1 if the robot succeeds in the task, -1 if it fails, and 0 otherwise.

In more detail: We provide positive feedback when the object is close enough to the target location,

$$r_t^{\text{goal}} = 1 \iff \|\mathbf{p}_t^{\text{man}} - \mathbf{p}_t^{\text{target}}\| < \epsilon^{\text{suc}}, \tag{4}$$

and give negative feedback when any obstacle i has been moved,

$$r_t^{\text{coll}} = -1 \iff \exists i \colon \|\mathbf{p}_t^{\text{obs},i} - \mathbf{p}_0^{\text{obs},i}\| > \epsilon^{\text{fail}}, \tag{5}$$

the tool leaves the work-surface (bound violation),

$$r_t^{\text{bound}} = -1 \iff \mathbf{p}_t^{\text{tool}} \notin \text{work-surface}, \tag{6}$$

or the maximum number of steps is reached (timeout)

$$r_t^{\text{time}} = -1 \iff t = T_{\max}. \tag{7}$$

In our nonprehensile rearrangement task the above definitions result in a delayed and sparse feedback signal at the end of each episode, which makes temporal-difference learning difficult. Especially at the beginning of the learning process, the feedback signal r_t^{task} will predominantly be negative, which in our experience leads to a slow learning process. While reward shaping (i.e. the modeling of sub-tasks in the feedback signal) can have adverse effects [58], we found that adding a signal r_t^{tool} that keeps the tool close to the manipulation object and a signal that rewards moving the object closer to the target location r_t^{target} is pertinent to learning the task. The overall feedback signal r_t is then defined as

$$r_t = \alpha_1 r_t^{\text{tool}} + \alpha_2 r_t^{\text{target}} + \alpha_3 r_t^{\text{task}}, \tag{8}$$

with weights $\alpha_1 = 0.1$, $\alpha_2 = 0.2$ and $\alpha_3 = 1$.

Concretely, the sub-task feedback signal, r_t^{tool}, increases when tool and manipulation object get closer to each other during the transition,

$$r_t^{\text{tool}} = \frac{1}{d_a} \left(\left\| \mathbf{p}_{t-1}^{\text{tool}} - \mathbf{p}_{t-1}^{\text{man}} \right\| - \left\| \mathbf{p}_t^{\text{tool}} - \mathbf{p}_t^{\text{man}} \right\| \right). \tag{9}$$

The sub-task feedback signal, r_t^{target}, increases when the manipulation object and the target location get closer during the transition,

$$r_t^{\text{target}} = \frac{1}{d_a} \left(\left\| \mathbf{p}_{t-1}^{\text{man}} - \mathbf{p}_{t-1}^{\text{target}} \right\| - \left\| \mathbf{p}_t^{\text{man}} - \mathbf{p}_t^{\text{target}} \right\| \right). \tag{10}$$

We note that in our task, the sub-task signal r_t^{tool} is never in conflict with the overall task since the tool has to be close to the manipulation object to move it. Differently, the sub-task signal r_t^{target} can conflict with the overall task when the tool has to move strictly sidewards (using action 1 and 5) to avoid an obstacle. However, both signals are scaled such that their range is small compared to r_t^{task} and their influence becomes minor after the initial learning phase.

4.2. Deep Q-Learning

Our input space of RGB images is continuous and high-dimensional. For this reason, we employ the Deep Q-learning algorithm [20] which models the Q-function Q^{sim} as a deep convolutional neural network that maps to the state-action value of each action $a \in \mathcal{A}$ in parallel. We reflect this in notation by writing $Q^{\text{sim}}(\mathbf{x}^{\text{sim}}, a; \boldsymbol{\theta})$ where $\boldsymbol{\theta}$ is the parameters of the network. Below we first explain our model for Q^{sim} and then detail the learning strategy.

4.2.1. Function Model

We follow a common approach in modeling Q-functions [20] and represent Q^{sim} with a sequence of convolution and pooling steps followed by several fully connected layers. The convolutional and pooling steps take one observation $\mathbf{x}^{\text{sim}}$ with 128×128 RGB pixels and map to a low-dimensional representation which is mapped to state-action values by the fully connected layers. The convolutional part consists of six convolutional layers with Rectified Linear Units (ReLU) as activation function to extract the feature map, and four max-pooling layers to reduce the size of the output. The model is visualized as the top network in Figure 3.

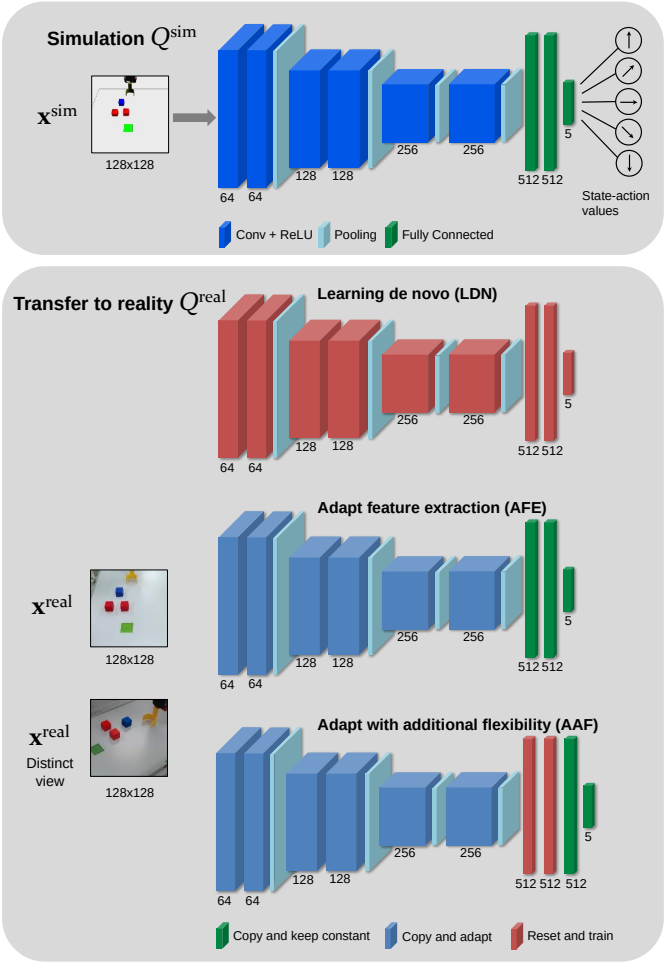


Figure 3: We model the function Q^{sim} with the deep convolutional neural network depicted in the top. The network computes Q-values for each action in parallel. For transfer to reality with pairs of corresponding images, we use three different strategies: We learn the whole model de novo (LDN), we copy Q^{sim} and adapt the feature extraction layer (AFE), and we copy Q^{sim}, add two additional layers for flexibility (AAF).

4.2.2. Learning the Q-Function

Q-learning is based on bootstrapping with single state transitions $(\mathbf{x}, a, r, \mathbf{x}')$. This means that the next target value for $Q^{\text{sim}}(\mathbf{x}, a)$ is defined by the transition reward r and the so-far learned state-action value of the following state $\mathbf{x}'$. However, representing Q^{sim} by a nonlinear function approximator, as explained in Section 4.2.1 above, can lead to instability and divergence during bootstrapping because of temporal correlation of transitions [59]. This problem is commonly addressed by *experience replay* [20, 29, 30, 31] and by training separate *target* and *primary networks*. The separate networks have parameters $\boldsymbol{\theta}^t$ and $\boldsymbol{\theta}^p$ respectively, and are updated by optimizing the following loss function:

$$\mathcal{L}(\boldsymbol{\theta}^p) = \mathop{\mathbb{E}}_{(\mathbf{x},a,r,\mathbf{x}')} \left[\left(r + \gamma \max_{a' \in \mathcal{A}} Q(\mathbf{x}', a'; \boldsymbol{\theta}^t) - Q(\mathbf{x}, a; \boldsymbol{\theta}^p) \right)^2 \right]. \tag{11}$$

Experience replay stores single transitions $(\mathbf{x}, a, r, \mathbf{x}')$ in a training data set D called replay buffer and uses them to optimize the primary network. In practice the loss function $\mathcal{L}(\boldsymbol{\theta}^p)$ is approximated by sampling transitions $(\mathbf{x}, a, r, \mathbf{x}')$ from the replay buffer D according to some distribution. It is therefore important that the sampled transitions are representative and cover the whole state-actions space. Otherwise, the estimated gradients are uninformative and worse in the worse case previously learning progress can be lost [20]. The target network parameters $\boldsymbol{\theta}^t$ are updated towards the primary network parameters $\boldsymbol{\theta}^p$ based on a certain schedule with a small learning rate η^t,

$$\boldsymbol{\theta}^t \leftarrow (1 - \eta^t)\, \boldsymbol{\theta}^t + \eta^t\, \boldsymbol{\theta}^p, \tag{12}$$

which leads to slow adaptation but increases learning stability.

Reinforcement learning algorithms have to tradeoff between *exploring* new areas of the state space and *exploiting* the so-far learned policy. Since in our case exploiting with a poor initial training policy rarely leads to successful episodes, we employ a training schedule with three different phases [20]. For the first K_1 episodes, we employ *only exploration* to train state perception. After that, we proportionally increase the exploration probability P_{exploit} until episode K_2. After that, we only train with exploitation for learning the state-action value function.

4.3. Behavior Policy with Informed Action Sampling

Q-learning is an off-policy reinforcement learning algorithm which means, that during training episodes, actions can be selected ad libitum. This allows us to collect transitions for learning with a behavior policy that is different from the learned policy to exploit background and domain knowledge. Usually, the behavior policy is an ϵ-greedy policy which most of the times follows the so-far learned Q-function and less often selects a random action. It can, however, be useful to assist an off-policy learning algorithm by exploring high-reward regions [60]. In our case, taking random exploration actions quickly fills the replay buffer $\mathcal{D}$ with transitions that lead to failure in the task. This is because random tool movements easily cause collisions. As a result, the replay buffer is biased towards experience with negative outcome which leads to slow progress towards learning in the task.

For our nonprehensile rearrangement task, we are particularly interested in a complete exploration heuristic that samples actions such that collisions are infrequent but that does not preclude certain types of experiences a priori.

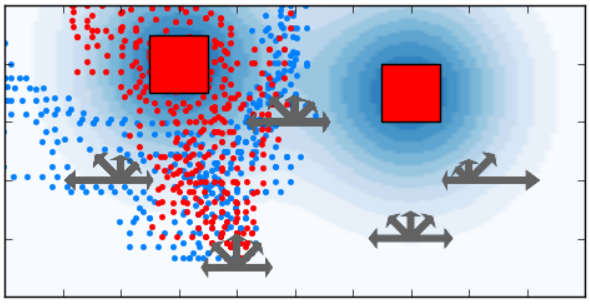


Figure 4: For exploration, we model the environment with obstacles (red squares) as a Gaussian potential field and sample actions according to local potential changes. This process selects actions leading away from obstacles more frequently than actions leading towards obstacles. Arrow length indicates 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.

One way to create an informed sampling heuristic is to model the environment by a potential field U and to derive from it an action distribution $P_{\mathcal{A}}$ which depends on local potential change. Such a distribution has a lower frequency of actions moving the tool close to obstacles, because it increases potential, and higher frequency of actions moving the tool into an obstacle-free region, because it decreases potential as illustrated by blue dots in Figure 4.

Specific to our nonprehensile rearrangement task, we model the table top environment by a mixture of two-dimensional and smooth object potential function $U(\mathbf{p})$. The potential function for each obstacle i is factorized according to dimensions and centered at the obstacles position $\mathbf{p}^{\text{obs},i}$,

$$U_i(\mathbf{p}) = \varphi(\mathbf{p}_x; \mathbf{p}_x^{\text{obs},i}, \sigma_i)\, \varphi_\alpha(\mathbf{p}_y; \mathbf{p}_y^{\text{obs},i}, \sigma_i), \tag{13}$$

where subscript x and y are used to refer to dimensions one and two respectively. The individual factors are defined by the normal distribution function φ and the skew-normal distribution function φ_α with shape parameter α for modeling obstacles. Since we assume that the tool is oriented along the Y-axis, this means that the potential field is steeper in front of an obstacle which puts stronger emphasize on avoiding collisions.

Similar to a common approach in planar navigation [61], we identify locally optimal motion directions $\theta \in [0, 2\pi]$ at a point $\mathbf{p} \in \mathbb{R}^2$ by considering instantaneous potential change in the field with respect to a motion direction. To this end, we integrate the instantaneous potential change in U at position $\mathbf{p}$ over angle intervals I_a for each action $a \in \mathcal{A}$,

$$\Delta(a, \mathbf{p}) = \int_{\theta \in I_a} \nabla_{\mathbf{v}_\theta} U(\mathbf{p})\, \mathrm{d}\theta. \tag{14}$$

In this integral, we use the directional derivative of the potential field with respect to a vector $\mathbf{v}_\theta = [\sin\theta, \cos\theta]^\intercal$,

$$\nabla_{\mathbf{v}_\theta} U(\mathbf{p}) = \mathbf{v}_\theta \cdot \frac{\partial U(\mathbf{p})}{\partial \mathbf{p}}, \tag{15}$$

which is defined by a dot product of the direction $\mathbf{v}_\theta$ and the gradient vector where $||\mathbf{v}_\theta|| = 1$. We define the angle intervals I_a by splitting the interval θ between $-\frac{1}{8}\pi$ and $\frac{9}{8}\pi$ into five equally sized segments centered at each action's motion direction, as shown in Figure 2.

We model the distribution $P_{\mathcal{A}}$ for sampling actions by relating the probability of an action to the instantaneous potential changes $\Delta(a, \mathbf{p})$. This means that higher change, which indicates increased required effort in the direction of an action, results in lower action probability. We therefore formulate the distribution $P_{\mathcal{A}}$ using a normalized exponential function,

$$P_{\mathcal{A}}(a \mid \mathbf{p}) = \frac{\exp(-\Delta(a, \mathbf{p}))}{\sum_{a' \in \mathcal{A}} \exp(-\Delta(a', \mathbf{p}))}, \tag{16}$$

which assigns higher probability to larger instantaneous reduction of potential.

4.4. Estimating Informative Gradient

As explained in Section 4.3 before, at the beginning of learning our nonprehensile rearrangement task, episodes will predominantly terminate with failure. This means that the common approach of inserting every transition into the replay buffer $\mathcal{D}$ and uniformly sampling to approximate $\mathcal{L}(\boldsymbol{\theta}^p)$ [20, 62, 63, 64] will be heavily biased towards transitions that lead to failure. In our experience, this hampers learning to the extend that it can completely fail to solve the task. Instead, for successful training, experiences sampled from D should be informative and representative. Such issues can be addressed by prioritizing certain types of experience [63].

In our experience, this means that sampled experiences should come from both successful *and* failing episodes in equal shares to provide informative gradients for optimizing the parameters $\boldsymbol{\theta}^p$. To avoid over-representing either failed or successful episodes in training data, we selectively store experiences in $\mathcal{D}$. If the ratio of successful experiences ρ in the replay buffer is below the threshold ξ_{min}, we store failing experience only with the probability $P_{\text{store}} = 0.1$. If, in contrast, ρ is above the threshold ξ_{max}, we store success experience only with the probability $P_{\text{store}} = 0.1$. This algorithm is expressed in listing 2. In any case, if the replay buffer is filled to maximal capacity, the oldest experience is displaced by the new experience. In this way, we can reduce redundant data while still taking in new experience. In experiments we set $\xi_{\text{min}} = 0.3$ and $\xi_{\text{max}} = 0.7$.

5. POLICY TRANSFER

In this section, we explain how we transfer the policy that we learned in simulation to a real-world robot. As

Algorithm 2 Training Data Set Curation

Analyze episode E for failure or success
if E was success **then**
  $P_{\text{store}} \leftarrow \begin{cases} 0.1, & \rho > \xi_{\max} \\ 1, & \text{otherwise} \end{cases}$
else
  $P_{\text{store}} \leftarrow \begin{cases} 0.1, & \rho < \xi_{\min} \\ 1, & \text{otherwise} \end{cases}$
end if
for $e \in E$ **do**
  Insert e into $\mathcal{D}$ (with displacement)
end for

described in Section 4, the policy is encoded by the Q-function which is modeled as a deep convolutional neural network Q^{sim} and takes simulated camera images $\mathbf{x}^{\text{sim}}$ as input. Therefore, we pose the problem of transferring the policy as learning a new function Q^{real} on real camera images which outputs the same values as Q^{sim} for real images $\mathbf{x}^{\text{real}}$ that correspond to simulated $\mathbf{x}^{\text{sim}}$ of the same scene. Conceptually, this means that we operate in three different data regimes: The training regime where we use simulation data $\mathbf{x}^{\text{sim}}$ for learning the policy, the transfer regime where we use both, simulation $\mathbf{x}^{\text{sim}}$ and real-world data $\mathbf{x}^{\text{real}}$ with additional instrumentation for ground truth labeling, and the test regime where we only use real-world data $\mathbf{x}^{\text{real}}$. Below, we first explain why this transfer is needed by describing the difference between the simulation and real-world. We then explain how we collect a data set of corresponding images $\mathcal{C}$ for transfer learning and finally formulate the transfer as a supervised learning problem.

5.1. Difference between Simulation and Reality

Although robot simulators such as Gazebo [28] allow us to collect experience in a couple of hours that would take months to collect in reality, for end-to-end reinforcement learning, they have the drawback that they do not perfectly capture reality in terms of perception data and physical properties. Therefore, models that are trained only with synthetic data from a simulator cannot be expected to generalize well to real robots. For instance, in our case, the Q-function Q^{sim} trained in Section 4 has a success rate of 0% when employed for controlling a real robot with real-world input data. This is exemplified in Figure 5, where Q^{sim} moves the robot into a collision even though the direct path to the target location is unobstructed.

In our nonprehensile rearrangement problem, the main difference between the two domains lies in perception data. A simulated image and real camera image of identical situations show significant differences in color and reflection of light, as well as the relative and absolute size of objects, robot parts, and the table top surface. We argue that no matter how hard we try to model the real scene in simulation, the light and the camera position would not

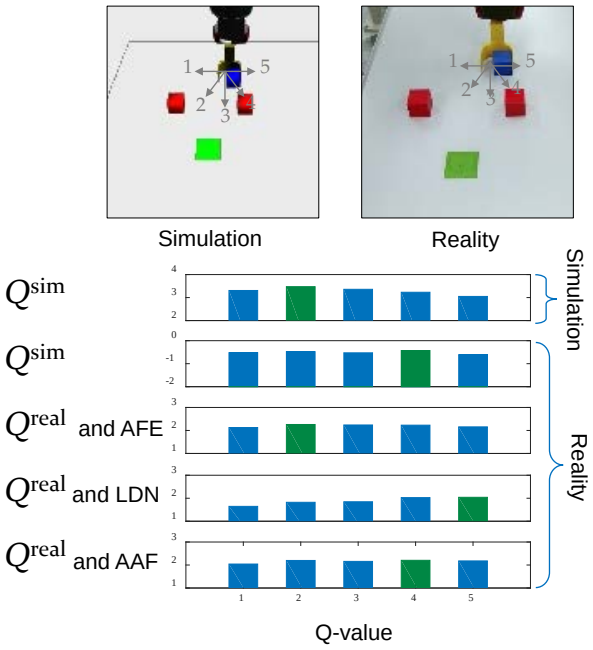


Figure 5: Q-value outputs of different Q-functions for simulated and real images. Q^{sim} selects the correct action in simulation but provided with a camera image of the real scene, it selects an action that leads to collision. After transfer learning of Q^{real}, only the AFE approach leads to the correct policy and selects action 2.

be exactly the same. Additionally, the predictable and repeatable nature of the simulation images and the wide, unpredictable diversity of real-world images makes bridging the gap even more difficult.

In contrast to the difference in visual perception, the difference in the relevant physical properties between simulation and reality is comparatively minor. As explained in Section 3.1, we assume that our nonprehensile rearrangement problem exhibits quasi-static dynamics. For this reason, interactions that occur in the nonprehensile rearrangement problem, are modeled with sufficient accuracy and we do not observe obvious difference.

5.2. Transfer Learning Data Set Collection

We collect a data set of pairs of corresponding images $\mathcal{C} = \{(\mathbf{x}_i^{\text{real}}, \mathbf{x}_i^{\text{sim}})\}_i$ that show the same scene as a camera image $\mathbf{x}_i^{\text{real}}$ and as a simulation rendering $\mathbf{x}_i^{\text{sim}}$. Since re-creating a simulated scene in reality is difficult and involves precise manual adjustments, we decide to instead manually create random scenes in reality and automize re-creation of the scene in simulation. For this, we introduce an additional RGB-D sensor with a calibrated pose relative to the robot. With this sensor, we collect point cloud data $\boldsymbol{\chi}^{\text{real}}$ that allows us to obtain all relevant information to re-create the scene virtually. The RGB-D sensor is placed at a different location than the robot's camera and can see the whole table top without obstruction. This additional sensor is only available during data collection

and therefore the algorithm in Listing 3 cannot be used to control the robot.

We begin by re-setting the real robot to one of its random starting states $\mathbf{j} \in \mathbf{J}_0$ at the end of the table top and placing the manipulation object some distance in front of that region. Then, we randomly arrange the obstacles in the area between the target location and the manipulation object. In the next step, we collect the camera image $\mathbf{x}^{\text{real}}$ and point cloud data $\boldsymbol{\chi}^{\text{real}}$. Using computer vision methods we extract the obstacle positions $\{\mathbf{p}^{\text{obs},i}\}_i$ and the position of the manipulation object $\mathbf{p}^{\text{man}}$ from $\boldsymbol{\chi}^{\text{real}}$. For this, we match the cube models with the 3D point cloud obtained form the depth sensor, using the Iterative Closest Point (ICP) [65, 66] algorithm. Point cloud data $\boldsymbol{\chi}^{\text{real}}$ and ICP results are exemplified in Figure 1. For practical reasons, the target location $\mathbf{p}^{\text{target}}$ is kept constant for transfer learning and evaluation in this paper and does not have to be extracted.

After initializing the simulation with the values of $\mathbf{p}^{\text{man}}$, $\{\mathbf{p}^{\text{obs},i}\}_i$, and $\mathbf{j}$, we move the simulated and real robot in parallel to collect corresponding images using either random actions or optimal actions. For optimal actions, we use the rendered image $\mathbf{x}^{\text{sim}}$ and the learned Q-function Q^{sim} in Equation (2). In this way, we get the corresponding images $(\mathbf{x}^{\text{real}}, \mathbf{x}^{\text{sim}})$, which we collect in the data set $\mathcal{C}$. This process is summarized in Figure 1 and example pairs of images can be seen in Figure 1(center) and Figure 5. Positions of manipulation tool, manipulation object, and obstacles in training data can be seen in Figure 8 in the experiment section.

Algorithm 3 Transfer Data Set Collection

```
Initialize transfer data set C
for all transfer episodes do
  Re-set the real robot to initial joint state j ∈ J_0
  Manually create random scene in reality
  Perceive image x^real and point cloud χ^real
  Extract {p^obs,i}_i and p^man from χ^real
  Reset simulation with {p^obs,i}_i, p^man, and j_0
  Render image x^sim in simulation
  Add pair (x^real, x^sim) to C
  while episode not terminated do
    Select optimal a ← arg max_{a'∈A} Q^sim(x^sim, a')
    Execute action a in simulation and reality
    Perceive image x^real
    Render image x^sim in simulation
    Add pair (x^real, x^sim) to C
  end while
end for
```

5.3. Supervised Policy Transfer

Using pairs of corresponding images $(\mathbf{x}^{\text{real}}, \mathbf{x}^{\text{sim}}) \in \mathcal{C}$ from the transfer learning data set and the already learned Q-function Q^{sim}, we learn a new Q-function Q^{real} with supervised learning. Our goal is that Q^{sim} and Q^{real} map to the same values when provided with corresponding input images $\mathbf{x}^{\text{sim}}$ and $\mathbf{x}^{\text{sim}}$. If this is the case, the same action is optimal in both simulation and reality which means that the same policy is encoded by Q^{sim} and Q^{real} for their respective domain.

To achieve this result, we have to minimize the difference in Q-value that is predicted by Q^{sim} and Q^{real} for each of the actions $a \in \mathcal{A}$. Our loss function for this learning problem therefore consists of the squared difference in Q-value summed over all actions,

$$\mathcal{L}(\boldsymbol{\theta}^{\text{real}}) = \sum_{i=1}^{|\mathcal{C}|} \sum_{a \in \mathcal{A}} \left(Q^{\text{real}}(\mathbf{x}_i^{\text{real}}, a; \boldsymbol{\theta}^{\text{real}}) - Q^{\text{sim}}(\mathbf{x}_i^{\text{sim}}, a; \boldsymbol{\theta}^{\text{sim}}) \right)^2 . \tag{17}$$

In the equation above, $\boldsymbol{\theta}^{\text{real}}$ stand for the parameters of the Q-function on real images Q^{real} and $\boldsymbol{\theta}^{\text{sim}}$ stands for the parameters of the Q-function learned in simulation Q^{sim}. The values of the latter, i.e. $Q^{\text{sim}}(\mathbf{x}_i^{\text{sim}}, a; \boldsymbol{\theta}^{\text{sim}})$, serve as the supervised targets for the former, $Q^{\text{real}}(\mathbf{x}_i^{\text{real}}, a; \boldsymbol{\theta}^{\text{real}})$. First we considered learning the parameters of Q^{real} de novo utilizing only the comparatively small number of target values provided by $\mathcal{C}$ but this does not generalize well to other scenes, as discussed in experiments in Section 7. Similarly, we evaluated the naive approach of fine-tuning Q^{sim} starting with the parameters $\boldsymbol{\theta}^{\text{sim}}$ with real-world data, but the performance was not as good as in simulation, as reported in Section 7. For this reason, we decide to design the model for Q^{real} based on the model of the already learned function Q^{sim}. Concretely, we initialize the model for Q^{real} with the structure and parameter values of Q^{sim} and then optimize the loss function in Equation (17) by modifying the structure and adapting the parameters. Our approach is motivated by the fact that structure and parameter values of Q^{sim} already contain task-relevant knowledge and only need to be adapted to the new domain.

In our experiments, we consider three different strategies of modifying and adapting the function model of Q^{sim} in order to learn Q^{real}.

1. We retain the structure of Q^{sim} but re-initialize all parameters and train Q^{real} de novo as a baseline.
2. Since the difference between the two domains is primarily visual, we copy Q^{sim} exactly and then adapt only the parameters of the feature extraction part consisting of the convolution layers.
3. To provide more flexibility, we introduce two new fully connected layers after the convolutional part as shown in Figure 3 and adapt the convolutional layers and the two new fully connected layers.

For brevity we refer to these as learning de novo (LDN), adapting feature extraction (AFE), and adapting with additional flexibility (AAF).

Table 1: System Parameters

Parameter	Notation	Value
Primary-Net Learning Rate	η^p	10^{-4}
Target-Net Learning Rate	η^t	10^{-3}
Replay Buffer Size	$\lvert\mathcal{D}\rvert$	$200,000$
Discount Factor	γ	0.99
Episode Limit	$T_{\max}$	150
Reward Weights	$\alpha_1,\ \alpha_2,\ \alpha_3$	$0.1, 0.2, 1$
Buffer Policy	$\xi_{\min}, \xi_{\max}$	$70\%, 30\%$
ε-greedy	$K_1,\ K_2$	$200, 5000$
Action Scale	d_a	1cm

6. POLICY LEARNING EXPERIMENTS

In this section, we evaluate our approach to learning a nonprehensile rearrangement policy in simulation. For this, we first present our simulation setup, describe data collection, and provide detail on model training. We employ the simulation setup to quantitatively evaluate the learned policy. Additionally, we provide qualitative examples that demonstrate how our approach reacts to sudden changes from external influences and what the effects of slight changes of physical properties are.

6.1. Simulation Platform and Setup

We use Gazebo [28] to simulate a Baxter robot. 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 Figure 2. The robot only controls its left arm to interact with the environment and we use joint space motion planning [67] to generate the five actions $a \in \mathcal{A}$. The manipulation object and obstacles are represented by $4 \times 4 \times 4$ cm cubes. The target region is indicated by a 6×6 cm square. An execution is successful if the horizontal distance between the centers of manipulation object and the target region is less than 2 cm. For perception, we simulate a fixed camera beside the robot as shown in Figure 1(left). We define a work-surface of 30 by 50 cm on the table top in which the manipulation tool can reach all positions.

6.2. Data Collection and Learning

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, however in evaluation, we will use more obstacles. The obstacles are placed randomly while at least one obstacle is directly placed between the manipulation object and the target location making obstacle avoidance necessary. We set the maximal episode length to $T_{\max}$ and execute the learning algorithm from Listing 1 to select actions and update the network.

For comparison and insight into the effects of our proposed methods, we train Q^{sim} de novo in three different regimes:

1. Without any of our newly proposed modifications as a baseline (BL).
2. With training data set curation (DC).
3. With training data set curation and heuristic exploration (DCHE).

In each of the three regimes we use approximately 600k transitions from about 10k episodes.

The training parameters listed in Table 1 are empirically determined with regard to performance and computation resources. Training is performed with a single Nvidia GeForce GTX 1080 Ti GPU using Adam [68] with mini-batch size set to 32.

6.3. Quantitative Evaluation

For quantitative evaluation of the learned policies, we consider the ratio of successfully terminated episodes (*success rate*) and compare their efficiency in solving the task by the *number of actions* they require to reach successful termination. Both quantities are estimated using a large set of random nonprehensile rearrangement problems.

6.3.1. Influence of Curation and Heuristic Exploration

In section 4.3 and 4.4, we argued that an unbiased training data set $\mathcal{D}$ is beneficial and proposed training data set curation and heuristic exploration. We evaluate the influence of the two newly proposed methods by inspecting the training data set at different stages during the training process. Since $\xi_{\min} = 0.3$ and $\xi_{\max} = 0.7$, the ratio of success transitions should quickly reach 30% and not rise above 70% when both methods are used.

As shown in Figure 6(a), the 30% mark is quickly reached when both methods are used (DCHE). Only data set curation reaches the mark after 1,500 episodes (DC) and the training regime without modifications requires about 5,000 episodes (BL). The training regime without modifications does not reach 50% of success transitions in $\mathcal{D}$ until about episode $7,000$ and stays below 20% till episode $4,000$. Adding training data set curation improves this to 50% at episode $5,000$. By additionally applying heuristic exploration, 50% is already reached at episode $3,000$. Both DC and DCHE finally reach and level out at 70% as intended by setting $\xi_{\max} = 0.7$.

6.3.2. Success Rate

During training, we evaluate each model every $1,000$ episodes and using 300 random scenes. As shown in Figure 6(b), the BL approach improves slower than the other two approaches and takes $5,000$ episodes until reaching 50% success rate. The DC approach starts better than BL and reaches 50% success rate already between $3,000$ and $4,000$ episodes. For the DCHE approach, the success rate reaches 50% before $2,000$ episodes and is stable and converged around episode $10,000$. We can observe rapid improvement until episode $6,000$ where success experience in the reply buffer reaches the upper limit of 70%. All approaches finally reach similar success rates. This indicates

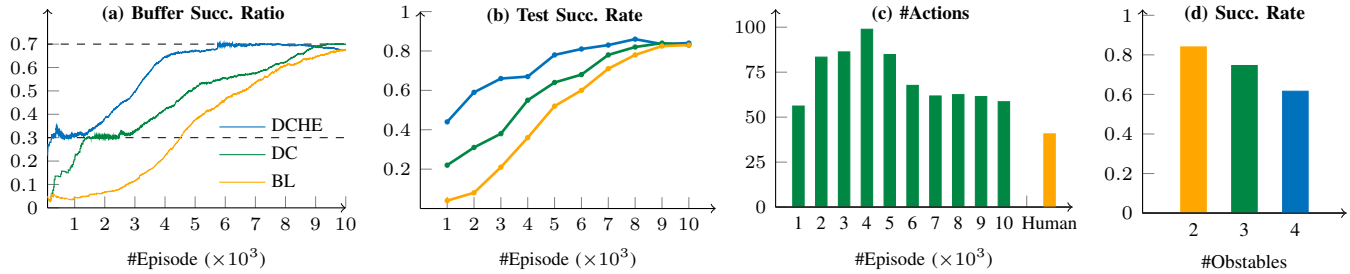


Figure 6: (a) The ratio of success episodes in the training data set $\mathcal{D}$ when $\xi_{\min} = 0.3$ and $\xi_{\max} = 0.7$. BL: Standard replay memory. DC: With data set curation. DCHE: With data set curation and heuristic exploration. (b) The success rates against the number of experienced episodes. (c) The average number of actions taken to accomplish a random task for DCHE versus a human operating the robot with a keyboard. (d) The success rate in test scenes with 2, 3 and 4 random obstacles for DCHE.

that sufficient success experiences in the reply buffer is crucial for increasing performance fast. Finally, we achieve a success rate of 85% indicating that the learned network can effectively handle the task of nonprehensile rearrangement. The same-level success rate is also achieved by DC and BL but much slower.

For comparison, we also test the success rate of a trivial strategy that moves the tool on a straight line to the target location. We obtain the manipulation object position from the simulator and move the robot gripper manually to collect the object. Then, we program the gripper to move straight to the target location. With this strategy, collision with the obstacles is the only reason for a failure. The success rate of this strategy is 6.67%, estimated from 300 random scenes. This indicates that perceiving the obstacles and actively avoiding them is necessary for success.

6.3.3. Number of Actions

The average number of actions that are needed to solve a nonprehensile rearrangement task indicates how efficient the policy is. Therefore, we compare the average number of actions during different stages of training with both training data curation and heuristic exploration (DCHE) to human performance using also 300 random scenes. The data about human performance is collected from a human subject who can only see the same input as the robot and presses arrow keys on a computer keyboard to control the manipulation tool in the 2d workspace. The results are shown in Figure 6(c).

During the initial phase of training, the policy can only solve trivial instances of the problem and therefore has a low average of used actions. After experiencing $4,000$ episodes, the number of actions decreases rapidly from 100 to below 70 until $6,000$ episodes from where on the number only decreases slowly to below 60. Compared to this, the human subject used on average below 40 actions. We attribute the larger number of actions required by the learned policy to its more conservative collision avoidance behavior since we observe it tends to keep away from obstacles.

6.3.4. Different Number of Obstacles

The training environment only contains 2 randomly placed obstacles but we are interested in whether the learned policy is limited to this number. For this reason, we evaluate the policy trained with training data set curation and heuristic exploration (DCHE) using 2, 3 and 4 obstacles in 300 random scenes for comparison. If the policy can solve the task with more than two obstacles the learning process must have generalizable knowledge about the task.

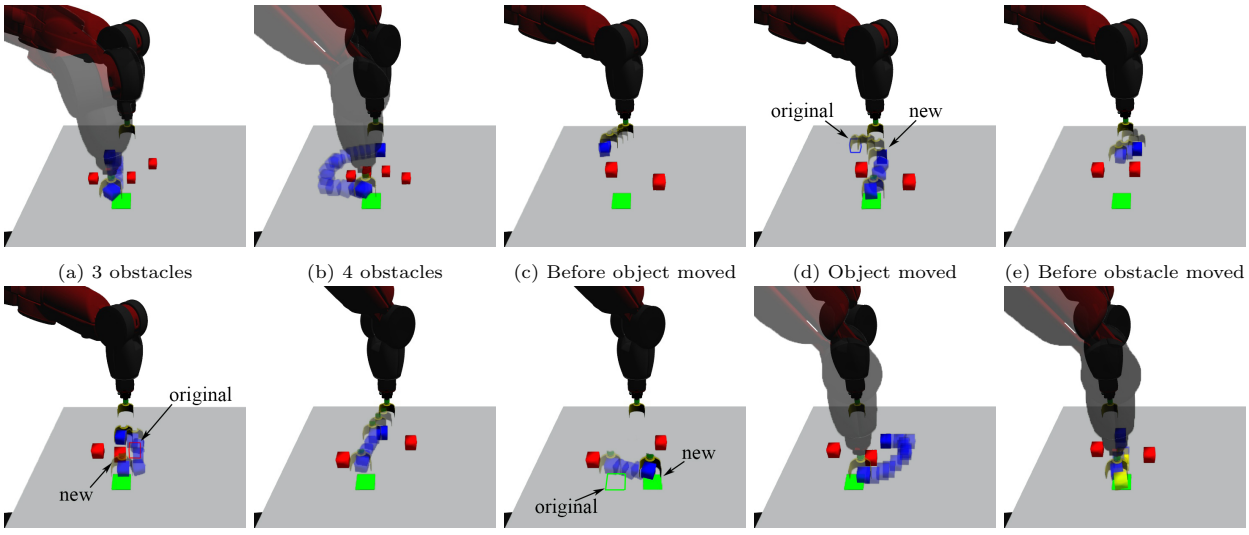
The resulting success rates are shown in Figure 6(d). The highest performance is with 2 obstacles and decreases from over 80% to below 80% for 3 obstacles. For 4 obstacles the success rate is about 60%. However, when more obstacles are added, the task also becomes more difficult. This can be observed in the example solutions shown in Figure 7(a-b). Also, in some cases, the target location is fully blocked by the randomly placed obstacles.

6.4. Qualitative Evaluation

One advantage of a learned policy as compared to a planning-based approach to nonprehensile rearrangement is that no computationally expensive re-planning is needed if an unforeseen event happened. In our case, the Q-function is computed in about 3ms which allows on-line reactive behavior. Below, we test the robustness of our policy (learned with DCHE) by moving objects and adding distractors during execution, as well as setting the friction coefficient to a different value from training.

6.4.1. Modifying the Environment

First, we test the effects of changing the position of the manipulation object. As seen in Figure 7(c-d), the robot adjusts and still collects the manipulation object. Next, we change the position of one of the obstacles to block the direct path to the target location. Figure 7(e-f), shows that the robot adapts and still solves the task. Finally, we change the target position. As seen in Figure 7(g-h), the robot also adjusts to this and moves into the direction of the new target.



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

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

Figure 7: Qualitative experiments in simulation 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.

6.4.2. Changing Friction

During training of the policy, we assumed that the manipulation object only moves because of interaction with the manipulation tool. In this test, we significantly decrease the friction coefficient between the manipulation object and the table top such that the object will begin to slide after each action in order to see whether the policy is limited to the dynamics of the training phase. In Figure 7(i), we can see that although the object path is jittering during the execution, the robot still solves the task. This example is also presented in the complementary video.

6.4.3. Adding Distraction

When the policy is trained, there are no other objects placed on the table top besides the obstacles, the manipulation object, and the visual marker for the target region. In this test, we introduce a yellow distraction object on the table top to see whether the policy will discover the new obstacle and react appropriately. As shown in Figure 7(j), the robot still completes the task. However, it is worthwhile to note that the distraction object is pushed which means that it is not considered as an obstacle by the policy.

7. TRANSFER LEARNING EXPERIMENTS

In this section, we use the learned Q-function Q^{sim} from the previous experiments and transfer it to a real-world robot. For this, we use the three different methods of supervised transfer AFE, AAF, and LDN that are discussed in Section 5.3. Below, we first describe the robot platform and sensor setup, then we provide details about data collection, and document the training process. We evaluate the transferred policy quantitatively based on success rate and qualitatively by modifying the environment during execution. Besides this, we conduct experiments where the real-world camera is placed in a different position from the simulated camera.

7.1. System Setup

Our experiments are conducted with a real Baxter robot placed against a table. The program is run on a computer with Intel i7-6850K CPU and a single Nvidia GeForce GTX 1080 Ti GPU. The network forward time for all architectures is around 4 milliseconds, which is negligible compared to action time. The executing time of each step is limited by the ability of the robot. In our case each step is around 0.6 second for Baxter. In average, 60 steps are needed for one episode as reported in Figure 6(c). Thus there is around half minute for finishing the task.

The images $\mathbf{x}^{\text{real}}$ are collected with a Logitech C920 webcam while the point cloud data $\boldsymbol{\chi}^{\text{real}}$ is collected with a Kinect One RGB-D sensor. The camera is first placed in approximately the same position as in simulation, but without exact calibration. The RGB-D sensor is mounted to have an unobstructed view of the table top. The manipulation object and the obstacles are 3d-printed in the same relative size as in simulation.

7.2. Data Collection and Learning

We follow the training data collection algorithm from Listing 3 and collect data from 70 episodes which results in 4,000 pairs of corresponding images in the data set $\mathcal{C}$. We remark that this process takes about 5 hours and that

collecting similar amounts of training data as for learning Q^{sim} is therefore not practical. For instance, it would take approximately 715 hours to collect the 10,000 training episodes used in Section 6 for policy learning. The episodes are randomly initialized with a constant target location. The distribution of collected data can be inspected in Figure 8. It includes a large variety of positions and covers most areas on the table top within the work-surface. One example of corresponding images is shown in Figure 5.

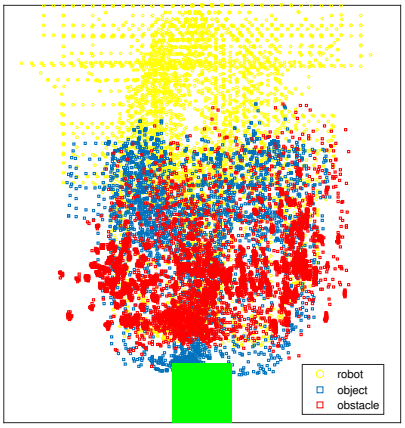


Figure 8: When collecting corresponding images for the transfer learning data set $\mathcal{C}$ it is important to cover many different situations and is not biased towards certain scenarios. This plot visualizes the distribution of positions of the tool, manipulation object, and obstacles on the table top that are contained in our data set. A wide range of starting positions for the tool and obstacles are contained. The plot also shows that the manipulation object is moved to many positions on the table.

The models for Q^{real} are trained with Adam [68], L2 regularization, and batch size 8. The learning rate η^{real} for the supervised learning is set to 1×10^{-4}.

7.3. Quantitative Evaluation

For qualitative evaluation, we first consider training loss and policy agreement and then investigate success rate in real robot experiments. The loss is according to Equation (17) while policy agreement is computed as the ratio of image pairs $(\mathbf{x}^{\text{sim}}, \mathbf{x}^{\text{real}}) \in \mathcal{C}$ for which Q^{sim} and Q^{real} are maximized for the same action.

7.3.1. Training Results

The loss curves against training epoch for transfer methods LDN, AAF, AFE are shown in Figure 9(left) and policy agreement is shown in Figure 9(right). If the loss is small, the models Q^{sim} and Q^{real} output similar Q-values and the policies agree the same actions are optimal.

In all three cases, the loss declines rapidly until 200 epochs, when the loss reaches about 0.01, which is after about 3 hours of training. The loss decreases first for strategies AFE and AAF which are both initialized with parameter values from Q^{sim}. But the final loss of AFE is higher than AAF and LDN, which allow the evolution of fully connected layers. Policy agreement increases with the decrease of loss where the same level ratio are achieved by strategies AAF and LDN.

Table 2: Success rate in reality.

Camera position	Q^{sim}	Q^{real} AFE	Q^{real} AAF	Q^{real} LDN	Domain random
Similar	0%	81%	65%	60%	43%
Distinct	0%	72%	66%	31%	0%

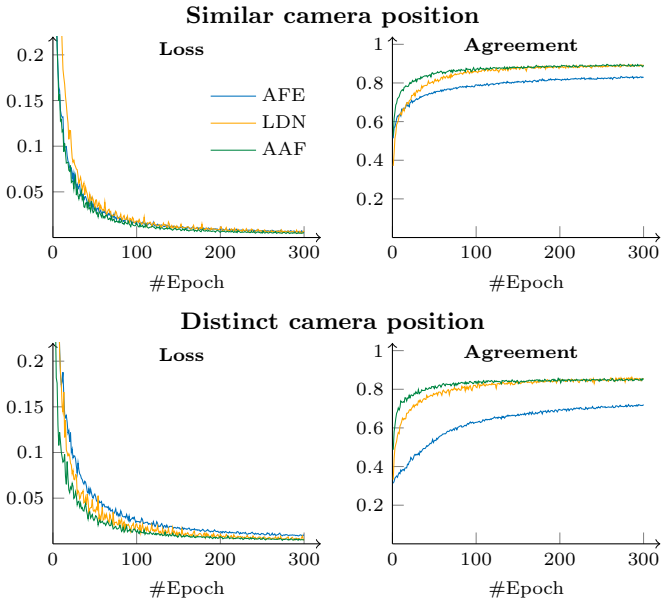
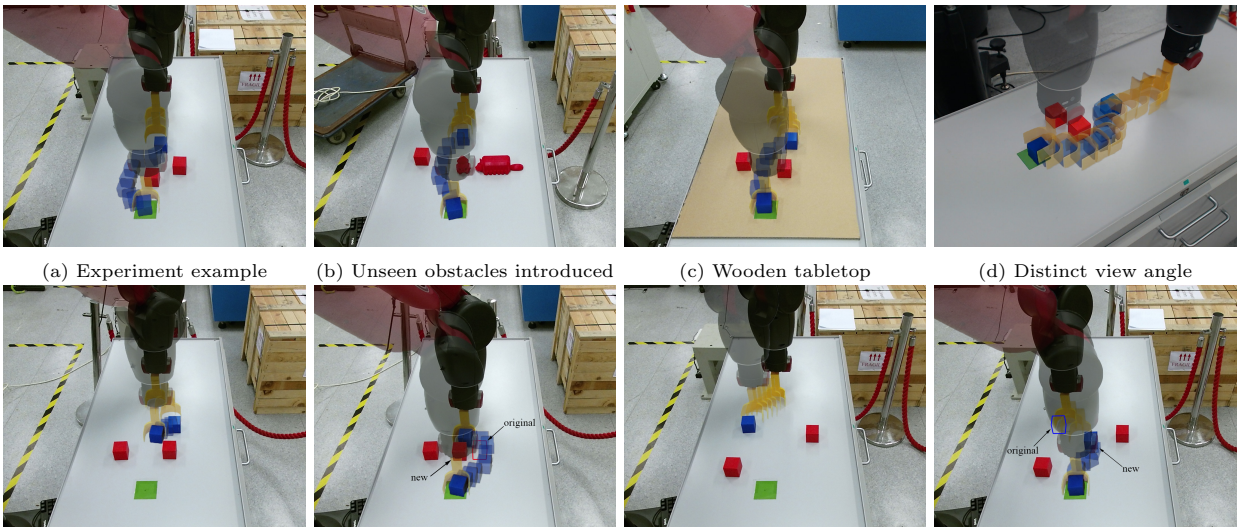


Figure 9: Training processes when camera in similar position and distinct position. Loss in training and policy agreement are presented to observe the behaviors of 3 different strategies: *AFE*, *LDN* and *AAF*.

7.3.2. Success Rate

We evaluate the three models learned by the transfer strategies after 200 epochs of learning and Q^{sim} on 100 random real-world rearrangement problems each. Success rates are shown in Table 2. For comparison, we also include a domain randomization baseline [49]. For this, simulation is randomized with different light intensities, the camera position is changed randomly, and Gaussian noise is added to the captured images. The success rate is obtained by directly applying the learned policy in the real-world.

Applying the Q^{sim} to control the real robot with real camera images results in 0% success rate. As exemplified in Figure 5, the selected actions are completely useless. The best performance is achieved by AFE with 81% success rate, which is at the same level as Q^{sim} in simulation. One example execution of this model is as shown in Figure 10(a). In contrast to the training losses, strategies AAF and LDN perform worse than AFE in real-world tests



(a) Experiment example (b) Unseen obstacles introduced (c) Wooden tabletop (d) Distinct view angle

(e) Before obstacle moved (f) Obstacle suddenly moved (g) Before object moved (h) Object suddenly moved

Figure 10: Experiments in reality. (a-b) Example executions in normal scene and when unseen obstacles were introduced. (c) Example execution on a wooden tabletop. (d) Example execution when the position of the camera was placed distinct from the original one. (e-f) Reactive path re-planning when the manipulation object or the obstacles were suddenly moved.

and result in success rate of 65% and 60% respectively. We assume that this indicates that these two models are overfitting to the training data. The success rate of fine-tuning Q^{sim} is the same as AAF, 65%.

The success rate of domain randomization is 43%. It is successful in the most simple scenarios where the obstacles do not block the direct path but it is prone to fail when moving close to obstacles is necessary. The policy often collides with obstacles when an accurate motion is needed, which indicates that domain randomization is not sufficient to handle accurate transfer. Besides that, we also observed failure modes where the policy moved the manipulation object away from the work-region for no apparent reason. This shows the difficulty of domain randomization.

7.4. Generalization Test

Similar to the evaluation in simulation above, we are interested in how the policy reacts to different types of unforeseen events. For this reason, we move obstacles and manipulation object during execution and introduce obstacles with different shapes from the ones observed in training data. Also, we test the transfer performance to a wooden tabletop. All experiments are conducted with the AFE trained model of Q^{real}.

7.4.1. Moving Manipulation Object and Obstacles

First, we move the obstacle to block the direct path to the target location. As seen in Figure 10(e-f), the robot reacts by moving around the obstacle. When we move the manipulation object as seen in Figure 10(g-h), the robot changes its behavior and reacts by collecting the objects from its new positions. In both cases, the robot shows enough flexibility to still solve the task.

7.4.2. Obstacles with Different Shape

In training we used two red cube-shaped obstacles but now we introduce obstacles of different shapes as seen in Figure 10(b), where it shows that the robot pushes the manipulation object around the new type of obstacle and can still solve the task. Because one of the novel obstacles is longer and more difficult for collision avoidance, the success rate with these two unseen obstacles is 78%, which is just a little worse than when tested with cube objects only. This indicates that the learned obstacle avoidance behavior is not restricted to only the training objects and can generalize to objects of different shapes and sizes.

7.4.3. Different Tabletop

After the experiments on a table which is similar to the one in simulation, we collect the data on a wooden table surface and test our transfer method. The data size and experiment setup is the same. As presented in Figure 10(c), the policy can still solve the task. The success rate is 80%, which is nearly the same as the previous environment.

7.5. Distinct Camera Position

In the experiments above, we placed the camera at a similar position as simulation. This means that the pairs of corresponding images $(\mathbf{x}^{\text{sim}}, \mathbf{x}^{\text{real}})$ show the scene from a similar angle. In this experiment, we place the camera position in an angle that differs significantly from simulation. In Figure 5, we can see how the image differs from the simulated image. Again, we transfer the policy with the three different strategies LDN, AFE, and AAF and perform the same quantitative evaluation as above. The results are shown in Figure 9 and Table 2 next to those from similar camera position.

Figure 9(bottom) shows that the losses are larger than with similar camera position and that the ratio of identical actions is lower. As seen in Table 2, the success rates for strategies AFE and LDN decrease from 81% to 72% and from 60% to 31% while the value for strategy AAF remains similar. We remark that strategy AFE with distinct camera position still performs better than strategies AAF and LDN with similar camera pose even though they have a better ratio of identical actions. Interestingly, the performance of strategy AAF which introduces two new fully connected layers does not decrease. This could mean that introducing these layers allows for better adaption to change of viewing angle.

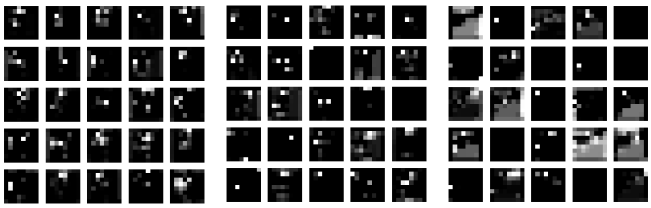
We also tested the the policy trained with domain randomization for this distinct camera angle. Since the camera angle in this experiment is significantly different from the one in the simulation, it is very difficult for domain randomization to generalize over. As expected, we have seen that the network cannot solve the problem for even very simple scenarios.

An example of the experiments is as shown in Figure 10(d). In this viewing angle, parts of the obstacles are occluded by the manipulation tool, which may cause the localization of the objects failed.

7.6. Feature Maps

Finally, we inspect the models' feature maps to see if similar features are relevant in both domains. In Figure 11, we show feature maps of Q^{sim} and two version of Q^{real} for corresponding images of the same scene in simulation, with similar camera position, and with distinct camera position. The feature maps are extracted at the last layer before the fully connected layers and both version of Q^{real} are trained with the AFE transfer strategy.

The feature maps of Q^{sim} and Q^{real} for similar camera pose mostly show activations in similar locations. Since AFE initializes Q^{real} with Q^{sim} and the fully connected layers are kept constant, this is not surprising. However, it shows that in Q^{real}, the layers before the depicted feature map are now adapted to provide similar activations as in Q^{sim} from real-world images of the scene. Compared to that, the feature maps of Q^{sim} and Q^{real} for distinct camera pose show less similarities. In many cases, the activations in Q^{real} are less unique and often in different locations. Since the AFE transfer strategy keeps the fully connected layers constant, this explains the drop in performance from similar to distinct camera pose observed in Table 2.



(a) (b) (c)

Figure 11: Feature maps of Q^{sim} and Q^{real} for corresponding images $\mathbf{x}^{\text{sim}}$ and $\mathbf{x}^{\text{real}}$ of the same scene. Feature maps are extracted at the last layer before fully connected layers and the transfer strategy is AFE such that the fully connected layers are identical. (a) Q^{sim}. (b) Q^{real} with similar camera position. (c) Q^{real} with distinct camera position. When comparing (a) and (b) similarities can be found that can be interpreted as detecting objects at similar locations. Compared to that, (c) has less unique activation and is more distinct from (a).

8. CONCLUSION

In this paper, we have formulated nonprehensile manipulation as an end-to-end learning problem. For this, we modeled the task as a reinforcement learning problem and used a transfer learning method based on a data set of corresponding simulated and real-world data to transfer learning results obtained in simulation to real-world input data. Unlike classic, planning-based methods, this approach does not rely on explicit perfect perception of the state or an accurate physical model. While training our approach relies on a large amount of virtual experience, as typical for deep reinforcement learning, only a small number of aligned real-world data is required for learning. To enable successful reinforcement learning in this problem domain, we additionally proposed a potential-field-based heuristic exploration method and active data set curation to provide the learning algorithm with a balanced data set.

We quantitatively evaluated the performance in simulation and reality and obtained success rates of 85% and 81% in random scenes. Since we recorded worse results, including complete failure to solve the task when removing the suggested components, this implies that our contributions are essential in solving this task. Qualitatively, we showed our system's reactive, adaptive, and robust behavior with regard to environment change between training and execution as well as during execution, such as the sudden change of object positions, changing low-friction coefficient, and previously unseen obstacles.

In the future, we will consider integrating more types of sensors, such as tactile and depth sensors, into our system to supply more information and enable cross-modal sensing ability to better perceive the manipulation environment and better understand the task space. Furthermore, our method is limited by the need for collecting a small amount of aligned cross domain training data for transfer. In future work, we plan to remove the step of recording aligned data and instead want to generate aligned data using generative adversarial models [51].

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), Knut and Alice Wallenberg Foundation and Foundation for Strategic Research.

References

[1] A. Cosgun, T. Hermans, V. Emeli, 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, S. Srinivasa, Physics-based grasp planning through clutter, in: Robotics: Science and Systems, 2012.

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

[4] M. Stilman, 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, D. Kragic, A framework for optimal grasp contact planning, IEEE Robotics and Automation Letters 2 (2) (2017) 704–711.

[6] K. Hang, M. Li, J. A. Stork, Y. Bekiroglu, F. Pokorny, A. Billard, D. Kragic, Hierarchical fingertip space: A unified framework for grasp planning and in-hand grasp adaptation, IEEE Transactions on Robotics 32 (4) (2016) 960–972.

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

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

[9] T. Siméon, J.-P. Laumond, J. Cortés, A. Sahbani, Manipulation planning with probabilistic roadmaps, The International Journal of Robotics Research 23 (7-8) (2004) 729–746.

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

[11] Y. Jiang, C. Zheng, M. Lim, A. Saxena, Learning to place new objects, in: Proc. IEEE Int. Conf. Robotics and Automation, IEEE, 2012, pp. 3088–3095.

[12] M. Dogar, S. Srinivasa, A framework for push-grasping in clutter, Robotics: Science and systems VII 1.

[13] M. Horowitz, J. Burdick, Interactive non-prehensile manipulation for grasping via pomdps, in: Proc. IEEE Int. Conf. Robotics and Automation, IEEE, 2013, pp. 3257–3264.

[14] J. Barry, L. P. Kaelbling, T. Lozano-Pérez, A hierarchical approach to manipulation with diverse actions, in: Proc. IEEE Int. Conf. Robotics and Automation, IEEE, 2013, pp. 1799–1806.

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

[16] I. M. Bullock, A. M. Dollar, Classifying human manipulation behavior, in: Rehabilitation Robotics (ICORR), 2011 IEEE International Conference on, IEEE, 2011, pp. 1–6.

[17] J. Liu, F. Feng, Y. C. Nakamura, N. S. Pollard, Annotating everyday grasps in action, in: Dance notations and robot motion, Springer, 2016, pp. 263–282.

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

[19] J. Mahler, J. Liang, S. Niyaz, M. Laskey, R. Doan, X. Liu, J. A. Ojea, K. Goldberg, Dex-net 2.0: Deep learning to plan robust grasps with synthetic point clouds and analytic grasp metrics, arXiv preprint arXiv:1703.09312.

[20] 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 518 (7540) (2015) 529–533.

[21] S. Levine, P. Pastor, A. Krizhevsky, D. Quillen, Learning hand-eye coordination for robotic grasping with large-scale data collection, in: International Symposium on Experimental Robotics, Springer, 2016, pp. 173–184.

[22] S. Gu, E. Holly, T. Lillicrap, S. Levine, Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates, in: Proc. IEEE Int. Conf. Robotics and Automation, IEEE, 2017, pp. 3389–3396.

[23] Y. Duan, X. Chen, R. Houthooft, J. Schulman, P. Abbeel, Benchmarking deep reinforcement learning for continuous control, in: International Conference on Machine Learning, 2016, pp. 1329–1338.

[24] G. Lample, D. S. Chaplot, Playing fps games with deep reinforcement learning., in: AAAI, 2017, pp. 2140–2146.

[25] H. Van Hasselt, A. Guez, D. Silver, Deep reinforcement learning with double q-learning., in: AAAI, Vol. 16, 2016, pp. 2094–2100.

[26] S. J. Pan, Q. Yang, A survey on transfer learning, IEEE Transactions on knowledge and data engineering 22 (10) (2010) 1345–1359.

[27] S. James, A. J. Davison, E. Johns, Transferring end-to-end visuomotor control from simulation to real world for a multi-stage task, in: Conference on Robot Learning, 2017, pp. 334–343.

[28] N. Koenig, 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.

[29] J. L. McClelland, B. L. McNaughton, 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 102 (3) (1995) 419.

[30] J. ONeill, B. Pleydell-Bouverie, D. Dupret, J. Csicsvari, Play it again: reactivation of waking experience and memory, Trends in neurosciences 33 (5) (2010) 220–229.

[31] L.-J. Lin, Reinforcement learning for robots using neural networks, Tech. rep., Carnegie-Mellon Univ Pittsburgh PA School of Computer Science (1993).

[32] W. Yuan, J. A. Stork, D. Kragic, M. Y. Wang, K. Hang, Rearrangement with nonprehensile manipulation using deep reinforcement learning, in: Proc. IEEE Int. Conf. Robotics and Automation, IEEE, 2018, pp. 270–277.

[33] Q. Zhu, Y. Yan, 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.

[34] A. Varava, K. Hang, D. Kragic, F. Pokorny, Herding by caging: a topological approach towards guiding moving agents via mobile robots, in: Proceedings of Robotics: Science and Systems, 2017. doi:10.15607/RSS.2017.XIII.074.

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

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

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

[38] J. Zhou, R. Paolini, A. M. Johnson, J. A. Bagnell, M. T. Mason, A probabilistic planning framework for planar grasping under uncertainty, IEEE Robotics and Automation Letters 2 (4) (2017) 2111–2118.

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

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

[41] L. Pinto, A. Gupta, Supersizing self-supervision: Learning to grasp from 50k tries and 700 robot hours, in: Proc. IEEE Int. Conf. Robotics and Automation, IEEE, 2016, pp. 3406–3413.

[42] A. Yahya, A. Li, M. Kalakrishnan, Y. Chebotar, S. Levine, Collective robot reinforcement learning with distributed asynchronous guided policy search, in: Proc. IEEE/RSJ Int. Conf. Intelligent Robots and Systems, IEEE, 2017, pp. 79–86.

[43] Y. Duan, M. Andrychowicz, B. Stadie, O. J. Ho, J. Schneider, I. Sutskever, P. Abbeel, W. Zaremba, One-shot imitation

learning, in: Advances in neural information processing systems, 2017, pp. 1087–1098.
[44] L. Tai, G. Paolo, M. Liu, Virtual-to-real deep reinforcement learning: Continuous control of mobile robots for mapless navigation, in: Proc. IEEE/RSJ Int. Conf. Intelligent Robots and Systems, IEEE, 2017, pp. 31–36.
[45] W. Yuan, K. Hang, H. Song, D. Kragic, M. Y. Wang, J. A. Stork, Reinforcement learning in topology-based representation for human body movement with whole arm manipulation, in: Proc. IEEE Int. Conf. Robotics and Automation, IEEE, 2019.
[46] C. Finn, S. Levine, Deep visual foresight for planning robot motion, in: Proc. IEEE Int. Conf. Robotics and Automation, IEEE, 2017, pp. 2786–2793.
[47] F. Ebert, C. Finn, A. X. Lee, S. Levine, Self-supervised visual planning with temporal skip connections, arXiv preprint arXiv:1710.05268.
[48] M. Večerík, T. Hester, J. Scholz, F. Wang, O. Pietquin, B. Piot, N. Heess, T. Rothörl, T. Lampe, M. Riedmiller, Leveraging demonstrations for deep reinforcement learning on robotics problems with sparse rewards, arXiv preprint arXiv:1707.08817.
[49] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, P. Abbeel, Domain randomization for transferring deep neural networks from simulation to the real world, in: Proc. IEEE/RSJ Int. Conf. Intelligent Robots and Systems, IEEE, 2017, pp. 23–30.
[50] F. Sadeghi, S. Levine, Cad2rl: Real single-image flight without a single real image, arXiv preprint arXiv:1611.04201.
[51] K. Bousmalis, N. Silberman, D. Dohan, D. Erhan, D. Krishnan, Unsupervised pixel-level domain adaptation with generative adversarial networks, in: Proceedings of the IEEE conference on computer vision and pattern recognition, 2017, pp. 3722–3731.
[52] A. Shrivastava, T. Pfister, O. Tuzel, J. Susskind, W. Wang, R. Webb, Learning from simulated and unsupervised images through adversarial training, in: The IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Vol. 3, 2017, p. 6.
[53] K. Bousmalis, A. Irpan, P. Wohlhart, Y. Bai, M. Kelcey, M. Kalakrishnan, L. Downs, J. Ibarz, P. Pastor, K. Konolige, et al., Using simulation and domain adaptation to improve efficiency of deep robotic grasping, arXiv preprint arXiv:1709.07857.
[54] Y. Ganin, E. Ustinova, H. Ajakan, P. Germain, H. Larochelle, F. Laviolette, M. Marchand, V. Lempitsky, Domain-adversarial training of neural networks, The Journal of Machine Learning Research 17 (1) (2016) 2096–2030.
[55] A. A. Rusu, M. Vecerik, T. Rothörl, N. Heess, R. Pascanu, R. Hadsell, Sim-to-real robot learning from pixels with progressive nets, arXiv preprint arXiv:1610.04286.
[56] E. Tzeng, C. Devin, J. Hoffman, C. Finn, P. Abbeel, S. Levine, K. Saenko, T. Darrell, Adapting deep visuomotor representations with weak pairwise constraints, arXiv preprint arXiv:1511.07111.
[57] R. S. Sutton, A. G. Barto, Introduction to Reinforcement Learning, 1st Edition, MIT Press, 1998.
[58] R. S. Sutton, A. G. Barto, Reinforcement learning: An introduction, MIT press Cambridge, 2018.
[59] J. N. Tsitsiklis, B. Van Roy, Analysis of temporal-diffference learning with function approximation, in: Advances in neural information processing systems, 1997, pp. 1075–1081.
[60] S. Levine, V. Koltun, Guided policy search, in: International Conference on Machine Learning, 2013, pp. 1–9.
[61] H. M. Choset, Principles of robot motion: theory, algorithms, and implementation, MIT press, 2005.
[62] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, D. Wierstra, Continuous control with deep reinforcement learning, arXiv preprint arXiv:1509.02971.
[63] T. Schaul, J. Quan, I. Antonoglou, D. Silver, Prioritized experience replay, arXiv preprint arXiv:1511.05952.
[64] S. Adam, L. Busoniu, R. Babuska, Experience replay for real-time reinforcement learning control, IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews) 42 (2) (2012) 201–212.
[65] P. J. Besl, N. D. McKay, Method for registration of 3-d shapes, in: Sensor Fusion IV: Control Paradigms and Data Structures, Vol. 1611, International Society for Optics and Photonics, 1992, pp. 586–607.
[66] R. B. Rusu, S. Cousins, 3D is here: Point Cloud Library (PCL), in: Proc. IEEE Int. Conf. Robotics and Automation, Shanghai, China, 2011.
[67] S. Chitta, I. Sucan, S. Cousins, Moveit!, IEEE Robotics & Automation Magazine 19 (1) (2012) 18–19.
[68] D. Kingma, J. Ba, Adam: A method for stochastic optimization, arXiv preprint arXiv:1412.6980.