Pre- and post-contact policy decomposition for non-prehensile manipulation with zero-shot sim-to-real transfer

Minchan Kim[1], Junhyek Han[1], Jaehyung Kim[1], and Beomjoon Kim[1]

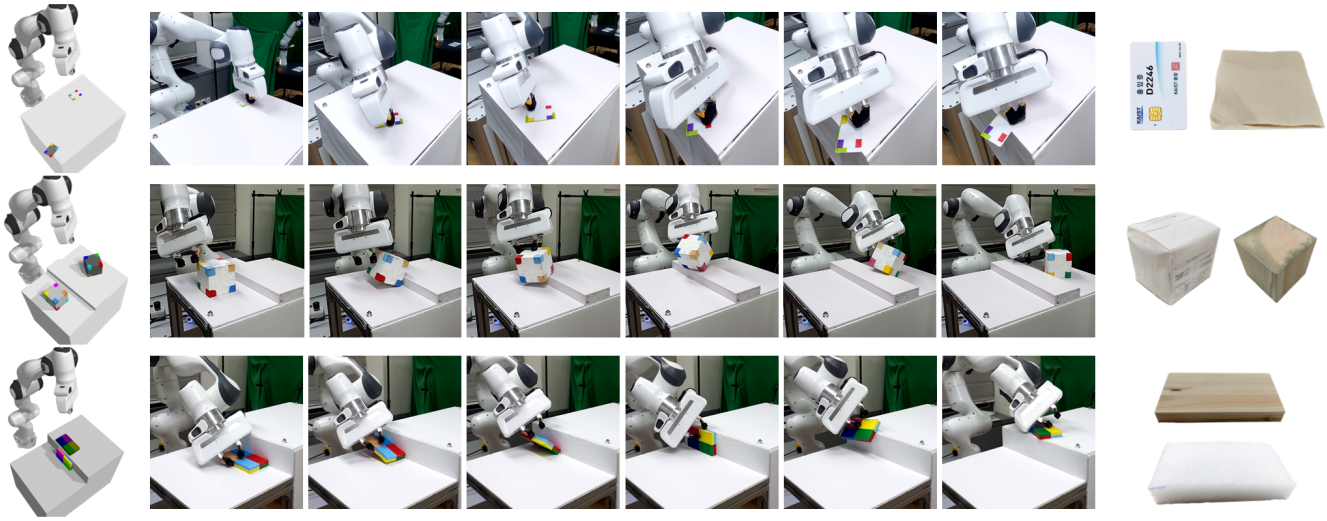


Fig. 1: Illustration of our tasks. The first column shows the initial robot and object poses and the desired object poses in the dark. The third column shows a subset of objects we manipulate in each domain. The middle column shows the manipulation motions. **First row**: the card is too flat to be grasped, so the robot must use dragging and re-orientation. **Second row**: the box is too large to be grasped, so the robot must push it to the bump, tumble it over, and re-orient it. **Third row**: the wall to the object's right is blocking all feasible grasps. The robot must first lift it up against the wall, drag it to the top, and then finally give a little push to move it to the target location. In the last two tasks, notice how the robot must both overcome and exploit the environmental contact to manipulate the object.

***Abstract*— We present a system for non-prehensile manipulation that require a significant number of contact mode transitions and the use of environmental contacts to successfully manipulate an object to a target location. Our method is based on deep reinforcement learning which, unlike state-of-the-art planning algorithms, does not require apriori knowledge of the physical parameters of the object or environment such as friction coefficients or centers of mass. The planning time is reduced to the simple feed-forward prediction time on a neural network. We propose a computational structure, action space design, and curriculum learning scheme that facilitates efficient exploration and sim-to-real transfer. In challenging real-world non-prehensile manipulation tasks, we show that our method can generalize over different objects, and succeed even for novel objects not seen during training. Project website: `https://sites.google.com/view/nonprenehsile-decomposition`**

I. Introduction

Humans possess remarkable skills in non-prehensile manipulation techniques such as pulling, pushing, dragging, and tumbling, which enables them to handle objects that are hard or impossible to grasp. In contrast, most robots are only capable of performing pick-and-place and fail in situations where grasping is not an option due to physical or geometric limitations. Our objective is to equip robots with the ability to manipulate objects even in such circumstances, using a basic gripper and an RGB camera as shown in Figure 1. This is a challenging problem that involves planning a sequence of combined robot and object movements while accounting for the contact interactions among an object, robot, and environment. Since the robot is equipped with a simple gripper, it must leverage the gravity and environment's geometry to perform the task effectively.

The state-of-the-art algorithms for such problems are planning algorithms that use tree search or trajectory optimization based on an analytical physics model [1–6]. However, their applicability in the real world is limited by several factors. Firstly, their search space is extremely complex, involving both discrete contact activations and continuous robot and object motions, with different sub-manifolds defined by different motion constraints. This makes planners too slow to use in practice. Secondly, they require accurate analytical contact models of the world, which is non-trivial to define

[1] Kim Jaechul Graduate School of AI at KAIST, {minchan21, junhyek.han, kimjaehyung, beomjoon.kim}@kaist.ac.kr

as rigid-body dynamics are often insufficient [7]. Moreover, even if we *can* define the governing equation of contacts, extracting the physical parameters for instantiating the equation, such as the center of mass or friction coefficient, from high-dimensional RGB images is a challenging problem.

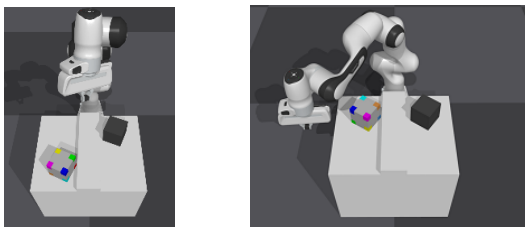
Because of these limitations, they have only been applied in a controlled setup where all necessary information about the object, such as inertial parameters or pose, is known. To achieve reasonable planning time, they often rely on strong assumptions about contact interactions, robot motions, and transition dynamics, such as quasi-static dynamics, predefined contact modes (slipping, static, rolling, etc.), and predefined contact locations in the environment, object, and robot [1, 3, 5, 8].

We propose an alternative solution for non-prehensile manipulation that utilizes deep reinforcement learning (RL). By directly learning the representation that implicitly encodes the necessary information from sensory data, our approach eliminates the need to extract difficult-to-observe object inertial parameters. At the expense of offline training, the online computation is reduced to feed-forward predictions from a neural network rather than searching in a complex space. Notably, our approach does not make assumptions about contact interactions and robot or object motions, making it more versatile than traditional planning algorithms.

Like the recent approaches for contact-rich tasks such as in-hand manipulation [9–11] and locomotion [12, 13], we use a simulator to train a policy and then transfer it to the real world. In this scheme, there are two primary challenges that we need to address for non-prehensile manipulation: exploration and the sim-to-real gap. Exploration in non-prehensile manipulation is challenging because unlike locomotion or in-hand manipulation where the object or the ground is in close proximity to make contact, the object is not in a position where we can directly make contact, especially in the initial state. So, if we randomly explore, there is zero chance of sampling an action that contacts the object [14]. The sim-to-real gap arises from the inaccuracies in the physics engine and modeling. We found that the hardware of the widely-used collaborative robots such as Franka Emika Panda is based on industrial robots and has a large gear ratio with intricate joint friction that is difficult to model accurately.

Our primary contribution is the design of a computational structure, action space, and learning curriculum that solves both of these challenges. For the exploration problem, one naive approach would be to define a contact-inducing reward that rewards the robot if its hand gets closer to the object. However, for non-prehensile manipulation, not all contacts are equal: some initial contacts are more crucial as it leads to more promising state space in the post-contact phase, as shown in Figure 2. Unfortunately, such a reward would make the robot contact the closest point on the object, and we empirically found this approach too sensitive to the initial system state.

Instead, we make a key observation that in general, any non-prehensile manipulation can be divided into two stages: the *pre-contact* and *post-contact* stages. In the pre-contact stage, the robot primarily focuses on finding a promising initial contact on the object that would lead to successful manipulation. In the post-contact phase, the robot finds a sequence of forces to apply to the object that successfully moves it to the target location. In this latter stage, the robot will mostly be in contact with the object — however, it might purposefully break a contact to exploit gravity and environmental contact or simply change its contact point to exert a force from a different direction.



(a) EE-ABOVE (b) EE-AT-RIGHT

Fig. 2: (a) EE-ABOVE: The end-effector is placed above the object. (b) EE-AT-RIGHT: The end-effector is placed on the right side of the object. The black cube denotes the goal pose. The robot should contact the object on the right to manipulate it to the goal, and initially contacting it on top of the object makes exploration extremely challenging as it would have to break the contract entirely and re-make it on the right.

Based on this observation, we propose a computational structure that facilitates effective exploration, where we use two different policies for each stage: pre-contact and post-contact policies, each having different action spaces. The *pre-contact policy*'s action is defined by the point on the object and the gripper that together defines the contact position, along with the gripper's orientation and gripper width at the contact. This guarantees contact with every action and enables the robot to explore the space of contacts more efficiently than say, using a joint torque or SE(3) end-effector pose. We train this policy first in a simulator where we have accurate object shape information and then transfer it to the real world by using a student-teacher training scheme [15].

Because the pre-contact policy involves just position control, there is a negligible sim-to-real gap. The design of the post-contact policy's action space, on the other hand, requires more consideration as it involves contact interaction with the object. In locomotion and in-hand manipulation, the typical approach is to learn a policy to generate the next target *joint position*, and use an analytical joint position controller [9, 11, 12]. On the other hand, for our tasks with natural motion constraints[1] and continual contacts, learning a policy that generates the target *end-effector pose* along with control gain parameters for an operational space controller (OSC) has shown to have the best sample efficiency among different action spaces [16].

Unfortunately, OSC requires an accurate robot model [17], but there are several factors hindering accurate modeling;

[1]In the context of hybrid position-force control

using a system identification helps, but we found it to be insufficient. For instance, the wrist joint on Panda has large friction in one direction but not so much in the opposite direction, and such intricacy cannot be captured with the parameters available in a typical physics simulator. Furthermore, since OSC uses a Cartesian space error to compute joint torques unless the end-effector moves exactly the same way for the given torque in simulation and the real world, there would be a large joint position gap.

We instead propose a different action space. Just like in [16], we predict the end-effector target pose. However, instead of predicting the gains for an OSC, we predict the gains for each joint and use inverse kinematics to solve for the joint position target. Then, we use a joint position controller to realize that joint position using the predicted controller gains. We found that using the joint position controller transfers much more effectively to the real world than OSC because the controller errors are now in the joint space rather than Cartesian space which you can correct by moving the respective joint.

Another factor we need to consider when transferring a policy to the real world is the violation of joint velocity and torque limits. The real robot has safety measures that prevent the robot to move above a certain speed and contact force. To account for this, we can terminate the episode if the robot violates them during learning, as in [10]. However, we found this makes learning difficult as it explores too conservatively, and fails to learn a policy in some of our domains. So, we introduce an action scaling curriculum that allows the robot to explore aggressively at the beginning of the learning but slowly decrease the maximum magnitude of the action so that the velocity and torque limits are met. This facilitates both efficient exploration and sim-to-real transfer.

As in previous works [3, 9, 11, 12], we make significant use of domain randomization to close the sim-to-real gap. We found that, with an end-to-end system, it is difficult to determine where the sim-to-real gap lies since the system is not modularized. So we design a modular system where the perception module takes a single RGB and outputs the 2D object key points. The policy takes the 2D key points and the proprioceptive sensor data to produce an action. We train our key point detector, pre-contact, and post-contact policies entirely in simulation and transfer them to the real world

We apply our method to three challenging non-prehensile manipulation problems where environment or object geometry renders grasping impossible, as shown in Figure 1. The robot needs to make use of both intrinsic and extrinsic contacts to manipulate objects that are not directly graspable. We show that not only can our method handle objects seen in training, but also new objects with different friction coefficients, centers of mass, and densities that are outside of the domain randomization range.

II. Related work

A. Planning through contacts

There are several methods for planning through contacts that can be applied to our tasks. The first class of algorithms is optimization-based methods [4, 5, 18] which analytically models the robot kinematic and dynamics constraints, such as joint limits, and contact constraints, such as friction cone constraints, as constraints in an optimization problem, and uses gradient-based techniques to find a trajectory. To handle the hybrid search space and discontinuous dynamics, some methods smoothen the contact mode decisions [5] or use complementarity constraints [4, 18]. While these algorithms can compute impressive motions with many contact mode changes, they also tend to output motions that are unrealistic because it is difficult to satisfy the constraints and to accurately model contacts. To our knowledge, these methods have scarcely been demonstrated on a real robot, if not never.

Another class of planning algorithms is based on graph search [1, 3, 8, 19, 20]. To handle discontinuous dynamics and hybrid search space, these methods typically construct a graph where each node encodes the contact mode and states of the object and robot, and each edge encodes the motion between two states if the transition is feasible. To compute the motion on each edge, either a pre-defined motion primitive [19] or an additional planner is used that accounts for dynamics constraints imposed by the contact mode and state [1, 3, 8, 20]. In general, these methods output more physically realistic motions than optimization-based algorithms and have been demonstrated on real robots [3, 18, 20]. However, they make strong assumptions such as quasi-static assumption or a pre-defined set of primitives or contact modes, and are limited to tasks that can be solved using simple motion with very few or no contact mode changes.

Because all these algorithms must compute a plan in a hybrid search space with discontinuous dynamics, they are too slow to use in practice. Furthermore, except [20], which predicts whether a motion primitive and its parameters will succeed from a segmented point cloud of the scene, all methods depend on the assumption that physical parameters such as mass and friction coefficient are known. In contrast, our approach suffers from neither of these problems. We can compute the next action by making a feed-forward prediction using a neural network, which usually takes on the order of milliseconds, and operate directly using a high dimensional camera and proprioceptive sensor rather than relying on accurate physical parameter knowledge.

B. Reinforcement learning for contact-rich tasks

RL algorithms have been successfully demonstrated for in-hand manipulation [9–11, 21–23] tasks. One key difference between these and our work is that in in-hand manipulation, the object typically starts in close proximity to the robot, whereas in our domain, we must solve the additional problem of reaching and making contact with the object. With an exception to [9], which has the object right beneath the hand, all systems begin with the object in hand. Like in [9], we encourage contact between the object and the finger using a reward. However, we found this to be insufficient as the initial contact is more crucial in our domain, and use a pre-contact policy to facilitate efficient exploration. Following [22], we use object key points to represent their pose.

TABLE I: The components of state space $\mathcal{S}$ of π_{post}.

Component	Description
$q[t] \in \mathbb{R}^9$	joint and finger positions at time step t
$\dot{q}[t] \in \mathbb{R}^9$	joint and finger velocities at time step t
$u_o[t] \in \mathbb{R}^{2\times 8}$	2D object key points at time step t
$u_g \in \mathbb{R}^{2\times 8}$	2D goal object key points
$T_E[t] \in SE(3)$	end-effector pose at time step t
$a_{post}[t-1]$	post-contact policy's action at time step $t-1$

There are few works on RL-based non-prehensile manipulation [24–27]. Unlike planning-based algorithms, these methods can handle high-dimensional input data such as images, do not explicitly depend on physical parameters, and have been demonstrated in the real world. However, these methods are limited to planar pushing, whereas planning algorithms could synthesize complex motions which incorporate multiple contact mode changes. Our method on the other hand not only generates complex movements, but also operates on high-dimensional sensory data without physical parameters.

The work by Zhou and Held [28] is close to ours in that it trains an agent to grasp an initially ungraspable object, and shows generalization over a variety of objects. One key difference from our work is that we focus on moving the object to the goal pose, whereas their objective is to grasp the object. Moreover, since they reward the agent to approach the target grasp, the motion typically involves only a few mode changes. Finally, they use an OSC and must operate the policy at a very low frequency to close the sim-to-real gap. However, our method controls joint position directly and controls the robot at a much higher frequency, enabling more dexterous motions.

III. Method

We are given a single RGB camera, a manipulator with a simple gripper, and a proprioceptive sensor. The target location of the object is defined using a relative transform with respect to its initial pose. We use Franka Emika Panda in this paper, but our method can be applied to other manipulators. Figure 3 shows how different modules in our system get trained and used in simulation and the real world.

A. Training π_{pre} and π_{post} in a simulator

The state space of π_{pre} consists of $(T_o, T_g) \in SE(3) \times SE(3)$ where T_o and T_g denote initial and goal object poses respectively. Its action space consists of four components: the orientation of the gripper, $R_E \in SO(3)$, its width, $l \in [0, 0.04]$, a point on the gripper, c_f, and a point on the object, c_o. The space of c_f is defined as the pre-defined points on the robot gripper, and the space of c_o is defined as points uniformly sampled from the surface of the object. The desired end-effector position, denoted p_c, is defined by matching c_f and c_o. See Figure 4 for an illustration. We compute the joint position at (p_c, R_E) using inverse kinematics (IK) solver.

We define the 3D key points as the vertices of the 3D bounding box for the target object. The state space of the post-contact policy π_{post} consists of variables listed in Table I. Its action space consists of the end-effector *residual* denoted with $\Delta T_E[t] \in SE(3)$, defined as the difference between the target and the current end-effector pose, the controller gain $k_p[t]$ and the damping ratio $\rho[t]$ for the joint controller. The controller gain $k_d[t]$ is calculated by $k_d[t] = \rho[t] \cdot \sqrt{k_p[t]}$. To execute an action, we use IK to solve for the joint position target in the next step, and then run the joint position controller with the controller gains.

We train both π_{pre} and π_{post} using Proximal Policy Optimization (PPO) [29] using the reward function defined as

$$
\begin{aligned}
r(s[t], a_{post}[t]) = \sum_{i=1}^{N_k} \frac{C_1}{\|x_o^i[t] - x_g^i\|_2 + C_1} - \|k_p[t]\|_2 \\
+ C_2 \mathbb{1}\left(d[t] < \bar{d} \text{ and } \theta[t] < \bar{\theta}\right) \\
+ \frac{C_3}{\|(p_{\text{lf}}[t] + p_{\text{rf}}[t])/2 - p_{\text{obj}}[t]\|_2} \ , \qquad (1)
\end{aligned}
$$

where C_1, C_2, and C_3 are positive constants, $x_o^i[t] \in \mathbb{R}^3$ and $x_g^i \in \mathbb{R}^3$ are 3D current and goal object key points, and N_k is the number of key points on the object. The first term increases as the object get closer to the goal, where the distance is computed based on the 3D key points. The second term regularizes the magnitude of k_p to induce a compliant motion. The third term is an extra reward for completing the task, in addition to matching the key points for completing the job. Here, $d[t]$ is the distance between the goal and current object position, and $\theta[t]$ is the distance between the goal and current object orientation, measured using a quaternion difference $\alpha[t]\alpha_g^{-1}$ between the current and goal object orientation, denoted $\alpha[t]$ and α_g respectively. Typically we set $C_2 \gg C_1$. The last term induces the end-effector to be close to the object. Here, p_{obj} is the position of the center of mass of the collision mesh of the object, and p_{lf} and p_{rf} are at the left and right finger joints respectively.

We train π_{pre} and π_{post} jointly. Given an environment, we first create a problem by sampling a $(T_o[0], T_g)$ pair. We then use π_{pre} to obtain its action, a_{pre}. If a_{pre} yields an end-effector pose that is in a collision or is kinematically infeasible, we terminate the episode, receive $C_4 < 0$ as a reward, and update π_{pre} with this reward. Otherwise, we execute a_{pre} by setting the robot at joint configuration q_E given by the IK solver for a_{pre}. Then, we execute π_{post}, updating it at every H time steps until the episode terminates. In this phase, an episode can terminate because (1) the maximum episode length, L, has been reached, (2) the robot drops the object, or (3) the robot succeeds in the task. Once the episode terminates, we take the sum of the rewards that we have obtained during the episode and update π_{pre}. The process repeats for a desired number of iterations.

B. Sim-to-real transfer

1) Transferring π_{pre}: Since the state and action of π_{pre} depend on information unavailable in the real world, we cannot use π_{pre} directly. So, we train a student policy, π_{pre}^S, using π_{pre} as the teacher policy [15]. The student policy's action consists of $q_E \in SE(3)$, the end-effector pose, and

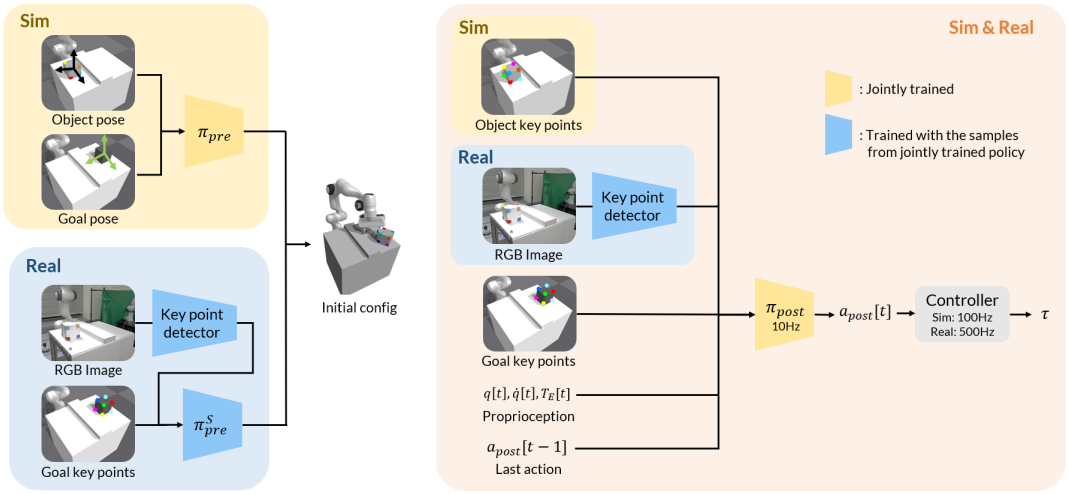


Fig. 3: Modules in our system. (Top left) In the simulation, we use the ground-truth object and goal pose as inputs to π_{pre}, which defines the initial robot configuration for the post-contact policy. (Bottom left) In the real world, we use the output of the key point detector and π^S_{pre}, the student pre-contact policy. (Right) π_{post} in simulation uses the ground-truth 2D key points along with other inputs, while in the real world, it uses the output of the key point detector.

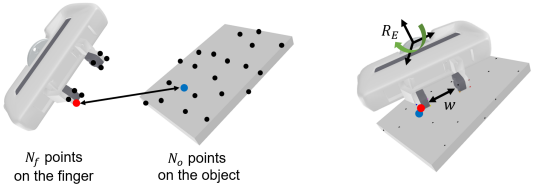


Fig. 4: Illustration of how π_{pre} makes the initial contact.

l. Its state consists of the 2D current and goal object key points, $u_o[t] \in \mathbb{R}^{2\times 8}$, and $u_g \in \mathbb{R}^{2\times 8}$. We generate the imitation learning data for π^S_{pre} by running a trained π_{pre} in the simulator. We convert actions of π_{pre} to q_E and l pair and use them as labels for π^S_{pre}. We also convert T_o and T_g into $u_o[t]$ and u_g by projecting the vertices of the bounding box of the object to the image plane. We train π^S_{pre} by minimizing the mean squared error (MSE). To ensure the MSE loss makes sense on the orientation, we represent the orientation of q_E in the form of 6D representation used in [30], which is just the first two columns of a rotation matrix.

2) Training and transferring a perception module: We train a key point detector using synthetic images in simulation and then use domain randomization to transfer it to the real world. Our detector takes a single RGB image as an input and outputs 2D object key points. We generate training data for the key point detector by running trained π_{pre} and π_{post} in the simulator. At each time step, we take an RGB image, a segmentation mask indicating whether each pixel belongs to the robot, the object, the table, or the background, and ground-truth 2D key points projected onto an image plane.

We adopt the key point detector network architecture from [31], which generates N_k heatmaps, one for each 2D key point, with the same size as the input image. These heatmaps represent the probability of each 2D key point's existence at each pixel. To train the network, we use ground-truth Gaussian heatmaps centered at the projected location on the image plane for each key point, as in [31]. We train the key point detector by minimizing the KL divergence between the predicted heatmaps and the ground-truth heatmaps. To make the key point detector robust to changes in the background and the environment when we transfer to the real world, we use momentum contrast learning [32] which enables the key point detector to focus on the object rather than its surroundings. We add InfoNCE term [33] to the KL divergence loss. To augment the images, we replaced the RGB values of the pixels belonging to the background and the table by referring to the segmentation mask. With probability 0.5, we replaced the pixel RGB values with random RGB colors, and with probability 0.5, we replaced them with textures used in [34].

3) Correcting the errors in a robot model: To close the sim-to-real gap in the robot model, we perform system identification on the joint dynamics parameters available in the simulator, denoted α, which includes joint friction, damping, and armature. To do this, we first collect N_{traj} trajectories with the length of $\mathcal{T}$ from the real robot by setting one of the seven joints to follow a sinusoidal joint position target of the form $q^{real}[t] = A\sin(\beta\gamma t)$, while the remaining joints maintain their position. The amplitude A and frequency γ of the trajectory are selected randomly from the hand-tuned range where its maximum is proportional to the position and velocity limits of each joint. We then optimize

$$\min_{\alpha} \sum_{i=0}^{N_{traj}} \Big(\sum_{t\in\mathcal{T}} (q_i^{real}[t] - q_i^{sim}[t;\alpha])^2\Big)^{\frac{1}{2}}$$

for each joint using CMA-ES [35], where $q_i^{sim}[t;\alpha]$ is the joint position that would result in the simulator when we

TABLE II: Domain randomization noise. $\mathcal{U}[min, max]$ denotes uniform distribution, and $\mathcal{N}[\mu, \sigma]$ denotes Normal distribution.

Parameter	Range
Table friction	$\times\mathcal{U}[0.7, 1.3]$
Robot end-effector surface friction	$\times\mathcal{U}[0.9, 1.1]$
Object mass	$\times\mathcal{U}[0.7, 1.3]$
Torque noise	$+\mathcal{N}[0.0, 0.03]$
2D key point noise	$+\mathcal{N}[0.0, 0.03]$
Sensor noise	$+\mathcal{N}[0.0, 0.01]$

use a joint position controller to follow the same sinusoidal trajectory with dynamics parameters set to α. Since only one joint moves in each trajectory, we can treat all joints separately, which simplifies the optimization problem.

4) Meeting the joint limits: The real robot has joint velocity and torque limits. To satisfy them, we must limit the magnitude of the end-effector residual at each time step, but this hinders efficient exploration. We use a curriculum for the maximum end-effector residual magnitude to handle this problem. Denote the target end-effector residual magnitude that satisfies the joint limits as ζ^*, which is obtained from the real robot. We begin with a large initial limit $\zeta_o > \zeta^*$, which gets reduced whenever the policy success rate reaches 80%. The scale is reduced in a geometric sequence with ratio $(\zeta^*/\zeta_o)^{\frac{1}{N_s}}$ where N_s is a hyperparameter that determines the number of steps to reach the target residual magnitude from the initial magnitude.

5) Domain randomization: To make the policy more robust to domain difference and the noise when it is transferred to the real robot, we further train our policy with domain randomization (DR) after we train our policy with an action scaling curriculum. Below is the list of properties we applied and the detailed ranges are noted in Table II.

- Physics properties: The friction of the table and robot and the mass of the object are randomized. We also add random noise to the commanded torque from the controller, to reflect the effects of real-world noise.
- Robot model: We adaptively reduce the gap between minimum and maximum values for the joint position range whenever the policy reached a success rate over a threshold.
- Perception: We add noise to the input of the policy to mimic the sensor noise and error of the key point detector.

IV. Experiments

A. Domain description and experiment setup

We have designed three domains to evaluate the performance of our method as shown in Figure 1. The first domain is the *card domain*, where for each training episode, we uniformly sample the initial and the goal object positions on the table while its orientation along the z-axis is uniformly sampled in the range of $[0, 2\pi)$. The second domain is the *bump domain*. The object is initially placed on the right side of the bump, and the robot's task is to push the object over the bump and reorient it to match a goal position and orientation, which is uniformly sampled on top of the bump or the opposite side of the table. The initial and the goal orientations along the roll and pitch axis are uniformly sampled from the set [0, $\pi/2$, π, $3\pi/2$], while that along the yaw axis is uniformly sampled from $[0, 2\pi)$. The third domain is the *wall domain*. The initial pose of the object is fixed to the up-right pose near the wall. The goal pose of the object is uniformly sampled on the left side of the top of the wall. We do not try to match the orientation in this domain. During training, we fix the density of the object. The default density of the object is $457.1\mathrm{kg/m^3}$, $200\mathrm{kg/m^3}$, and $150\mathrm{kg/m^3}$ for the card, bump, and wall domain respectively.

We use IssacGym [36] to train the policies, and use different policies for different domains. In the simulator, we run $24,576$ environments simultaneously. We use $C_1 = 0.02$, $C_2 = 1000$, $C_3 = 0.03$, $C_4 = -100$, $\bar{d} = 0.01\mathrm{m}$ and $\bar{\theta} = 0.1\mathrm{rad}$ for the reward function. We run π_{post} to output actions at a frequency of 10Hz, while the controller runs at a frequency of 100Hz. For the action scale schedule, we use $N_s = 10$, $\zeta_o = (0.06, 0.1)$, and $\zeta^* = (0.02, 0.03)$.

For the real-world experiment, we use a RealSense D435 camera to obtain RGB images. We use RRT* [37] to plan a trajectory from the initial robot configuration given by π^S_{pre}, and use Polymetis [38] to control the robot. We run π_{post} outputs its action at 10Hz, and the controller runs at 500Hz. To address safety issues, we set the minimum damping ratio for $\rho[t]$ at 0.5. We wrapped the gripper of the robot with a glove that has a higher friction coefficient to match the setting of the simulation.

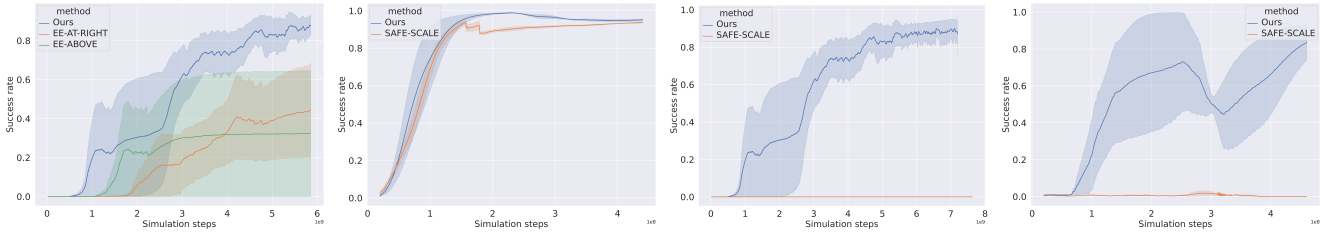
B. Results

In this paper, we make the following claims.

- Claim 1: Jointly training π_{pre} with π_{post} facilitates more efficient exploration than using a single policy with contact-inducing reward.
- Claim 2: Residual curriculum enables efficient exploration and learning of a policy that meets joint limits.
- Claim 3: Inverse differential kinematics and joint position controller is better suited for sim-to-real transfer on robots with large gear ratios, such as Panda.
- Claim 4: Our system can transfer to the real world and generalize to objects that were not seen during training.

To support claim 1, we compare our method against the benchmark that does not use π_{pre} in the bump domain where we initialize the joint position of the robot in two different ways, as shown in Figure 2a. In both setups, the robot needs to learn that it needs to contact the object on the right side to push it over the bump. Figure 5a shows the results. We can see the effect of using π_{pre}: our method reaches 95% success rate at around 6 billion steps, while the one without π_{pre} and π_{post} decomposition fails to reach the same level of success rate. The policy in EE-AT-RIGHT setup does a little better than the one in EE-ABOVE, because the robot begins at a configuration near the right side of the object. However, it is much less data-efficient than our method.

To support claim 2, We compare methods with and without end-effector residual scaling during training. For the one



(a) Bump domain (b) Card domain (c) Bump domain (d) Wall domain

Fig. 5: Learning curves for (a) comparing our method with the one that does not use pre-contact policy in the bump domain, (b, c, d) comparing our method to the one without scheduling in card, bump, and wall domains, respectively. The sudden dips or temporary plateaus in the graphs are due to domain randomization and residual scaling curriculum.

Domain	Controller	Object	Mass (g)	Surface	scenario 1	scenario 2	scenario 3	Success rate
Card	Ours	3D printed (default)	8	plastic	5/5	3/5	5/5	0.87
	OSC				1/5	1/5	0/5	0.13
	ζ^*				5/5	2/5	4/5	0.73
	Ours	3D printed (wrapped)	8	vinyl	4/5	1/5	2/5	0.47
		Acrylic	**16***	acrylic	3/5	3/5	5/5	0.73
		Wood	8	wood	5/5	4/5	5/5	0.93
		Credit card	5	plastic	5/5	5/5	4/5	0.93
		Tissue†	**1***	tissue	4/5	5/5	5/5	0.93
		Chocolate chip†	9	vinyl	3/5	4/5	5/5	0.87
Bump	Ours	3D printed (default)	146	paper	5/5	3/5	5/5	0.87
	OSC				0/5	0/5	1/5	0.07
	Ours	3D printed (wrapped)	146	vinyl	4/5	2/5	5/5	0.73
		3D printed (high density)	**223***	paper	5/5	5/5	5/5	1.00
		Wood	**381***	paper	5/5	3/5	4/5	0.80
		Painted wood	**381***	wood	2/5	5/5	3/5	0.67
		Sponge†	**30***	paper	2/5	5/5	3/5	0.67
		Diaper box†	102	paper	2/5	4/5	2/5	0.53
Wall	Ours	3D printed (default)	38	paper	5/5	3/5	5/5	0.87
	OSC				0/5	0/5	0/5	0.00
	Ours	3D printed (wrapped)	38	vinyl	5/5	3/5	5/5	0.87
		3D printed (high density)	**62***	paper	5/5	4/5	4/5	0.87
		Wood	**97***	paper	5/5	0/5	4/5	0.60
		Painted wood	**97***	wood	5/5	1/5	3/5	0.60
		Sponge†	**5***	paper	5/5	5/5	5/5	1.00
		Water tissue box†	**138***	paper	0/5	1/5	2/5	0.20

TABLE III: The real-world results in diverse objects for three domains. † means the object is non-rigid, and * means the mass of the object is out of DR range. ζ^* means the policy is trained with the same controller and architecture as our system except that it uses fixed residual magnitude limit ζ^*. Each scenario refers to different initial and target object poses.

without the end-effector residual, we fix the residual magnitude limit to ζ^* and train the policy. Figures 5b, 5c, and 5d show the result in three domains. As the plot shows, except for the card domain, the one without scheduling fails to learn the task, demonstrating the importance of our scheduling scheme. For the card domain, We further compare our policy with the policy without scheduling in the real world. The policy without scheduling achieved 73% success rate, and ours achieved 87% as shown in Table III. This shows that action scaling not only helps with exploration, but also in closing the sim-to-real gap.

To support claim 3, we compare our method against the method in [16] where the policy outputs the next end-effector pose and the gains for an OSC in the real world. We use the same action scheduling and policy architecture as our system but use a different action space and controller. In the simulation, the OSC-based method achieves over 90% success rates in the card and bump domains and over 80% in the wall domain in simulation. We then compare them in the real world with the object that has the same physical parameters as the one used in training. Table III shows the result. It shows that the OSC-based approach does much worse in the real world and achieves 13%, 7%, and 0% success rates in card, bump, and wall domains respectively, while ours achieve 87% success rates in all three. This indicates that OSC-based action space has a larger sim-to-real gap than our action space.

To support claim 4, we test our policy on 7 different objects in each domain. The objects vary in their mass, surface friction, scale, and deformation. We paint the objects so that it has the same visual appearance as the training data. The list of the objects and the results are shown in Table III. For the default object that has the same physical properties as the one we used in training, we achieve 87% success rate, indicating that our system can transfer to the real world. Furthermore, our method succeeds even for some objects whose physical properties (ex. non-rigid, heavy, etc.) are outside of our domain randomization range. The object with the lowest success rate is the water tissue box, which is highly non-rigid.

V. Limitation

Our current work is limited to fixed environments and objects, and must train new policies if they change. The fu-

ture work would involve generalizing across different object shapes by incorporating shape information.

VI. Acknowledgement

This work was supported by Institute of Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea government(MSIT) (No.2019-0-00075, Artificial Intelligence Graduate School Program(KAIST)), (No.2022-0-00311, Development of Goal-Oriented Reinforcement Learning Techniques for Contact-Rich Robotic Manipulation of Everyday Objects), (No. 2022-0-00612, Geometric and Physical Commonsense Reasoning based Behavior Intelligence for Embodied AI), and Samsung Electronics.

References

[1] G. Lee, T. Lozano-Perez, and L. P. Kaelbling, "Hierarchical Planning for Multi-Contact Non-Prehensile Manipulation," in *International Conference on Intelligent Robots and Systems*, 2015.

[2] J. Barry, T. Lozano-Peréz, and L. P. Kaelbling, "A Hierarchical Approach to Manipulation with Diverse Actions," in *International Conference on Robotics and Automation*, 2013.

[3] X. Cheng, E. Huang, Y. Hou, and M. T. Mason, "Contact Mode Guided Motion Planning for Quasidynamic Dexterous Manipulation in 3D," in *International Conference on Robotics and Automation*, 2022.

[4] M. Posa, C. Cantu, and R. Tedrake, "A Direct Method for Trajectory Optimization of Rigid Bodies Through Contact," *The International Journal of Robotics Research*, 2014.

[5] I. Mordatch, Z. Popović, and E. Todorov, "Contact-Invariant Optimization for Hand Manipulation," in *ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, 2012.

[6] B. Aceituno-Cabezas and A. Rodriguez, "A Global Quasi-Dynamic Model for Contact-Trajectory Optimization," in *Robotics: Science and Systems*, 2020.

[7] I. Kao, K. M. Lynch, and J. W. Burdick, "Contact Modeling and Manipulation," in *Springer Handbook of Robotics*, 2016.

[8] K. Miyazawa, Y. Maeda, and T. Arai, "Planning of Graspless Manipulation Based on Rapidly-Exploring Random Trees," in *Assembly and Task Planning: From Nano to Macro Assembly and Manufacturing*, 2005.

[9] T. Chen, J. Xu, and P. Agrawal, "A System for General In-Hand Object Re-Orientation," in *Conference on Robot Learning*, 2022.

[10] T. Chen, M. Tippur, S. Wu, V. Kumar, E. Adelson, and P. Agrawal, "Visual Dexterity: In-Hand Dexterous Manipulation from Depth," *arXiv*, 2022.

[11] O. M. Andrychowicz, B. Baker, M. Chociej, R. Jozefowicz, B. McGrew, J. Pachocki, A. Petron, M. Plappert, G. Powell, A. Ray, *et al.*, "Learning Dexterous In-Hand Manipulation," *The International Journal of Robotics Research*, 2020.

[12] J. Hwangbo, J. Lee, A. Dosovitskiy, D. Bellicoso, V. Tsounis, V. Koltun, and M. Hutter, "Learning Agile and Dynamic Motor Skills for Legged Robots," *Science Robotics*, 2019.

[13] J. Lee, J. Hwangbo, L. Wellhausen, V. Koltun, and M. Hutter, "Learning Quadrupedal Locomotion over Challenging Terrain," *Science Robotics*, 2020.

[14] C. R. Garrett, R. Chitnis, R. Holladay, B. Kim, T. Silver, L. P. Kaelbling, and T. Lozano-Perez, "Integrated Task and Motion Planning," *Annual Review of Control, Robotics, and Autonomous Systems*, 2021.

[15] D. Chen, B. Zhou, V. Koltun, and P. Krähenbühl, "Learning by Cheating," in *Conference on Robot Learning*, 2020.

[16] R. Martín-Martín, M. A. Lee, R. Gardner, S. Savarese, J. Bohg, and A. Garg, "Variable Impedance Control in End-Effector Space: An Action Space for Reinforcement Learning in Contact-Rich Tasks," in *International Conference on Intelligent Robots and Systems*, 2019.

[17] J. Nakanishi, R. Cory, M. Mistry, J. Peters, and S. Schaal, "Operational Space Control: A Theoretical and Empirical Comparison," *The International Journal of Robotics Research*, 2008.

[18] J. Moura, T. Stouraitis, and S. Vijayakumar, "Non-Prehensile Planar Manipulation via Trajectory Optimization with Complementarity Constraints," in *International Conference on Robotics and Automation*, 2022.

[19] C. Zito, R. Stolkin, M. Kopicki, and J. L. Wyatt, "Two-Level RRT Planning for Robotic Push Manipulation," in *International Conference on Intelligent Robots and Systems*, 2012.

[20] J. Liang, X. Cheng, and O. Kroemer, "Learning Preconditions of Hybrid Force-Velocity Controllers for Contact-Rich Manipulation," in *Conference on Robot Learning*, 2022.

[21] I. Akkaya, M. Andrychowicz, M. Chociej, M. Litwin, B. McGrew, A. Petron, A. Paino, M. Plappert, G. Powell, R. Ribas, *et al.*, "Solving Rubik's Cube with a Robot Hand," *arXiv*, 2019.

[22] A. Allshire, M. Mittal, V. Lodaya, V. Makoviychuk, D. Makoviichuk, F. Widmaier, M. Wüthrich, S. Bauer, A. Handa, and A. Garg, "Transferring Dexterous Manipulation from GPU Simulation to a Remote Real-World Trifinger," *arXiv*, 2021.

[23] A. Handa, A. Allshire, V. Makoviychuk, A. Petrenko, R. Singh, J. Liu, D. Makoviichuk, K. Van Wyk, A. Zhurkevich, B. Sundaralingam, *et al.*, "DeXtreme: Transfer of Agile In-Hand Manipulation from Simulation to Reality," *arXiv*, 2022.

[24] W. Yuan, J. A. Stork, D. Kragic, M. Y. Wang, and K. Hang, "Rearrangement with Nonprehensile Manipulation Using Deep Reinforcement Learning," in *International Conference on Robotics and Automation*, 2018.

[25] W. Yuan, K. Hang, D. Kragic, M. Y. Wang, and J. A. Stork, "End-to-End Nonprehensile Rearrangement with Deep Reinforcement Learning and Simulation-to-Reality Transfer," *Robotics and Autonomous Systems*, 2019.

[26] K. Lowrey, S. Kolev, J. Dao, A. Rajeswaran, and E. Todorov, "Reinforcement Learning for Non-Prehensile Manipulation: Transfer from Simulation to Physical System," in *International Conference on Simulation, Modeling, and Programming for Autonomous Robots*, 2018.

[27] X. B. Peng, M. Andrychowicz, W. Zaremba, and P. Abbeel, "Sim-to-Real Transfer of Robotic Control with Dynamics Randomization," in *International Conference on Robotics and Automation*, 2018.

[28] W. Zhou and D. Held, "Learning to Grasp the Ungraspable with Emergent Extrinsic Dexterity," in *Conference on Robot Learning*, 2022.

[29] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal Policy Optimization Algorithms," *arXiv*, 2017.

[30] Y. Zhou, C. Barnes, J. Lu, J. Yang, and H. Li, "On the Continuity of Rotation Representations in Neural Networks," in *Conference on Computer Vision and Pattern Recognition*, 2019.

[31] M. Vecerik, J.-B. Regli, O. Sushkov, D. Barker, R. Pevceviciute, T. Rothörl, R. Hadsell, L. Agapito, and J. Scholz, "S3K: Self-Supervised Semantic Keypoints for Robotic Manipulation via Multi-View Consistency," in *Conference on Robot Learning*, 2021.

[32] K. He, H. Fan, Y. Wu, S. Xie, and R. Girshick, "Momentum Contrast for Unsupervised Visual Representation Learning," in *Conference on Computer Vision and Pattern Recognition*, 2020.

[33] A. v. d. Oord, Y. Li, and O. Vinyals, "Representation Learning with Contrastive Predictive Coding," *arXiv*, 2018.

[34] Y. Jiang, A. Gupta, Z. Zhang, G. Wang, Y. Dou, Y. Chen, L. Fei-Fei, A. Anandkumar, Y. Zhu, and L. Fan, "VIMA: General Robot Manipulation with Multimodal Prompts," *arXiv*, 2022.

[35] N. Hansen, S. D. Müller, and P. Koumoutsakos, "Reducing the Time Complexity of the Derandomized Evolution Strategy with Covariance Matrix Adaptation (CMA-ES)," *Evolutionary Computation*, 2003.

[36] V. Makoviychuk, L. Wawrzyniak, Y. Guo, M. Lu, K. Storey, M. Macklin, D. Hoeller, N. Rudin, A. Allshire, A. Handa, *et al.*, "Isaac Gym: High Performance GPU-Based Physics Simulation for Robot Learning," *arXiv*, 2021.

[37] S. Karaman and E. Frazzoli, "Sampling-Based Algorithms for Optimal Motion Planning," *The International Journal of Robotics Research*, 2011.

[38] Lin, Yixin and Wang, Austin S. and Sutanto, Giovanni and Rai, Akshara and Meier, Franziska, "Polymetis," 2021.