Policy Pretraining through Imitation of Expert Demonstrations and Object Pose Estimation for Sim2Real Pushing

Ji HUANG, Muhammad Reza Ar RAZI, Yankun WEI

***Abstract*—The manipulation of objects by robots, particularly in the context of pushing tasks, remains a challenging problem due to the inherent uncertainties in the environment and the dynamic nature of the interactions involved. This article presents a method for training policies for pushing tasks through imitation learning of expert demonstrations in a simulation environment. In addition, a *singleshotpose* network is trained with simulation data and applied to both to sim and real environment for object pose estimation, with an effort to reduce the reality gap through domain randomization.**

***Index Terms*—robotics, imitation learning, pose estimation, reality gap, domain randomization.**

I. Introduction

In recent years, the integration of deep learning techniques with robotics has seen remarkable advancements, opening avenues for more adaptive and efficient robotic systems. However, the policy training process usually requires a large number of data. While the data could be acquired directly on a real robot [1], this solution is too expensive in terms of time and resources to scale up. Alternatively, the data can be generated in simulation faster, cheaper, safer, and with unmatched diversity [2]. One major issue about simulation-based training is the inherent mismatches with real-world settings. Bridging the gap between simulation and reality is of great importance.

The task of this topic is training a policy to push a randomly-located box to a defined target position with a kinematically-redundant 7-link robotic manipulator. Model-free Reinforcement Learning approaches typically suffer from high sample complexities, which require a lot of iterations to train a policy for challenging tasks. The aim of this topic is to solve the pushing task in randomized scenarios by using a demonstrative custom-designed expert policy. For the vision task, this article focuses on the application of the network ***singleshotpose*** [3] for estimating object poses.

Given the challenges associated with data collection in real environments, the pushing task and the policy training of the robotic manipulator is based on simulation platform ***Virtual Training Platform for Robot Learning (VTPRL)*** as shown in Fig 1. Built on unity, VTPRL can provide simulation of physical effects as well as image rendering. Although the object's pose can be acquired directly in the simulator and used for policy training, vision-based object pose estimation, alongside the utilization of domain randomization techniques, is still indispensable for sim-to-real transfer and training of robust and adaptable systems.

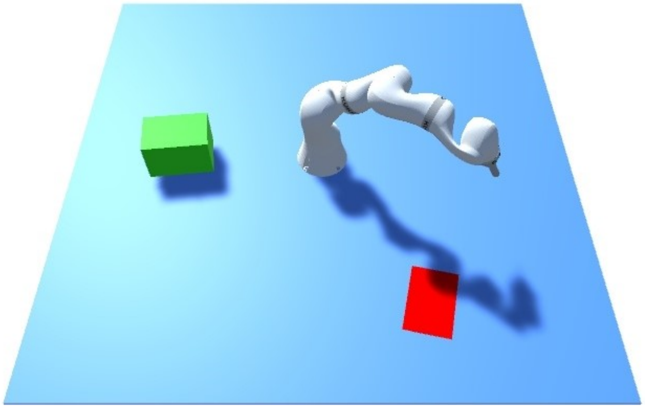

Fig. 1. Simulation environment. The task of the KUKA LBR iiwa 7 R800 manipulator is to push the green box to the red target

II. Related Work

A. Nonprehensile Manipulation

In general, robot manipulators are very efficient in pick-and-place operations. This is reasonable since once the robot has firmly grasped the object, it can rely upon its position accuracy capability to move the object without the need of continuously sensing the state thereof [4]. However, if the object is subject only to unilateral constraints, the manipulation task is said to be nonprehensile [5]. In this scenario, the object can still be manipulated by the end effector, but it is possible neither to prevent any infinitesimal motions of the object nor to resist all external wrenches applied to it [6], leading the manipulation to be non-trivial.

Pushing is the one of the nonprehensile motion primitives. For the robots without grippers, prehensile manipulation is not an option. Even in the presence of grippers, objects may be too large or too heavy to grasp. As described in [7], while pushing can solve these problems, the control problem is made difficult by some indeterminacy about the presence of friction forces, also causing an unpredictability of the precise object's motion. These factors are crucial for the design of the control policy.

At the early stage of research, the pushing problem was simplified to planar pushing and investigated under ***quasi-static assumption*** with analytical approaches [8]. However, the assumptions do not hold in real applications. More recently, ***physics engines*** have been utilized in a large body work related to pushing. They can take dynamic interaction and 3D objects into consideration. Zhu et al. [9] utilized a physics engine for motion prediction, learning the physical parameters through black-box Bayesian optimization. With the advances of deep learning, SE3-NETS [10] was introduced. The network takes depth image, action vectors and associations between points in subsequent images as inputs to predict 3D rigid body motions. Besides from modeling the dynamics of physical systems, deep reinforcement learning has been used to learn control policies. For example, Lowrey et al. [11] used a modified form of the natural policy gradient algorithm to train a non-prehensile manipulation policy in simulation, and transferred it to a physical system.

B. Vision Based Object Pose Estimation

There exists a wide range of pose estimation approaches. Traditional methods rely on handcrafted features and geometric principles, using local keypoints and feature matching to recognize and estimate pose of RGB object instance [12]. Other high-performing classical methods rely on depth information and use point-pair based voting [13].

With the emergence of deep learning, learning-based approaches for object pose estimation have gained prominence. These methods leverage the expressive power of neural networks to directly learn the mapping between image features and object poses. Convolutional Neural Networks (CNNs), in particular, have shown remarkable performance in various vision tasks, including object detection, segmentation, and pose estimation.

Some learning based approaches regress representations of the pose directly. PoseCNN [14] obtain regions-of-interest by a custom detection pipeline and regresses depth, the projected 2D center postion and a quaternion for each region-of-interest. DeepIM [15] proposes an iterative refinement method. Given pose hypothesis, their network regresses a pose difference between pose hypothesis and ground truth, using rendered image from pose hypothesis and original image as input.

Other learning based approaches are based on establishing 2D-3D correspondences followed by a variant of PnP-RANSAC algorithm. BB8 [16] presents a 6D object detection pipeline comprising two distinct CNN modules. The first CNN coarsely segments the object of interest from the background, while the second predicts the 2D locations of the projections of the object's 3D bounding box based on the segmentation results, which are then used to compute the 6D pose using PnP algorithm. The method is effective but slow due to its multi-stage nature. *Singleshotpose* propose a deep CNN architecture that takes the image as input and directly detects the 2D projections of the 3D bounding box vertices [3], which are more suitable for real-time applications.

C. Sim-to-Real Transfer

Simulation-based training provides data at low-cost, but involves inherent mismatches with real-world settings [17]. Several methods have been investigated to reduce this reality gap.

System identification [18] entails the construction of a precise mathematical model to represent a physical system accurately. However, challenges still exist in creating sufficiently realistic simulators. One such obstacle is the difficulty in generating high-quality rendered images that faithfully simulate real-world vision.

Domain adaptation [19] refers to a set of transfer learning techniques developed to update the data distribution in sim to match the real one through a mapping or regularization enforced by the task model.

Domain randomization [20] highly randomize the simulation in order to cover the real distribution of the real-world data despite the bias between the model and real world. By create a variety of simulated environments with randomized properties and train a model that works across all of them, model can adapt to the real-world environment, as the real system is expected to be one sample in that rich distribution of training variations.

Both domain adaptation and domain randomization are unsupervised. Compared to domain adaptation which requires a decent amount of real data samples to capture the distribution, domain randomization may need only a little or no real data, which is the reason we use domain randomization in this project.

III. Methodology

A. Simulator Environment

1) Action Space and Observation: In our quest to streamline the task complexity, the action space was deliberately confined to a 2D plane. This simplification allowed for a focused manipulation strategy leveraging the x and y axes. Correspondingly, the observation space was meticulously structured to encapsulate a comprehensive snapshot of the task environment, comprising:

- A 6-dimensional representation of the box pose, detailing its position and orientation in the environment.
- A 6-dimensional final target pose, specifying the desired endpoint location and orientation.
- A 3-dimensional end effector position, highlighting the current location of the robot's manipulative tool.

2) Initial Position: To circumvent the complexities associated with implementing intricate path planning algorithms for navigating the end effector to the box's rear without initiating contact, we opted for a scenario-based approach to initial positioning. This approach predicates two distinct initial positions for the end effector: at the leftmost and rightmost sides of the table. Consequently, the box's initial position is invariably set to be directly in front of the end effector, ensuring alignment on the same table side. This setup strategically places the final target on the opposite side of the table, simplifying the task's directional dynamics.

3) Reward: The reward function, designed to evaluate policy performance, is encapsulated by the following equation:

$$\text{reward} = \max\left(d_m, 1.0 - \frac{d_c}{d_{init}}\right) \tag{1}$$

where:

- d_c is the current distance from the box to the final target,
- d_{init} is the initial distance at the start of the episode,

- d_m tracks the best (maximum) fraction of goal distance reduction achieved throughout the episode.

This function rewards the policy for reducing the distance between the box and the target, effectively encouraging progress towards the goal.

B. Expert Demonstration

To achieve the task of pushing a box to a designated target, we developed a hardcoded expert policy that guides the robot hand through a series of calculated actions. The policy breaks down the complex task into manageable steps, leveraging the robot's 2D action space for precise control. This subsection outlines the logic and steps implemented in the expert policy.

- **Move to Pre-Block Position:** This phase is a critical initial step in the expert policy, designed to position the robot hand optimally for the subsequent action of pushing the box towards the target. This phase involves precise calculations and movements to ensure the robot is correctly aligned for an effective push.
  The goal in this phase is to maneuver the robot hand to a strategic pre-block position, precisely 0.05 meters away from the nearest surface of the box. This positioning is calculated based on the formula:
  $$\text{Pre-Block Distance} = \frac{\text{Box Length}}{2} + 0.05 \tag{2}$$
  This ensures the robot hand is optimally placed for initiating the push, allowing for an approach that factors in the momentum and alignment necessary to direct the box towards the target.
- **Alignment and Reorientation:** Alignment accuracy is critical for the success of the pushing task. The alignment is evaluated using the theta error, which quantifies the angular discrepancy between the robot's current orientation and the optimal pushing direction. An alignment correction is initiated if:
  $$|\theta_{\text{error}}| > \theta_{\text{threshold\_to\_orient}} = 0.2618 \text{ rad (15 degrees)} \tag{3}$$
  Reorientation actions are undertaken to adjust the robot hand's position and orientation until the theta error falls below this threshold, ensuring the robot is correctly aligned to push the box straight towards the target without inducing unintended rotations.
  To adjust the box's trajectory or orientation, a rotation is applied to $\vec{xy}_{\text{dir\_block\_to\_target}}$ by an angle of 0.15 radians, determined by the direction (left or right) necessary for optimal alignment towards the target. This rotational adjustment allows for fine-tuning the box's path, ensuring it remains aligned with the target through the pushing process.
- **Push Block Phase:** Upon successfully aligning with the pre-block position, the robot hand initiates the *Push Block* phase. The aim is to direct the center of the box towards the target by calculating the movement direction as follows:
  $$\vec{xy}_{\text{dir\_block\_to\_target}} = \frac{\vec{xy}_{\text{block\_to\_target}}}{||\vec{xy}_{\text{block\_to\_target}}||} \tag{4}$$
  where $\vec{xy}_{\text{block\_to\_target}}$ represents the vector from the box to the target, and $||\cdot||$ denotes the norm of the vector, ensuring the direction is normalized.
- **Smooth Movement Control:** To achieve a controlled approach to the pre-block position, a maximum step velocity of 0.01 meters per step is maintained, facilitating smooth and precise movements. This velocity constraint is crucial for preventing any premature disturbance to the box, allowing for real-time trajectory adjustments as the robot hand nears the target position. The implementation detail is as follows:
  $$\text{Max Step Velocity} = 0.01 \text{ meters/step} \tag{5}$$
  This constraint ensures the robot hand approaches the pre-block position with the necessary caution and precision, laying the groundwork for a successful push action.
- **Feedback Loop:** The expert policy continuously evaluates the robot hand's proximity to the calculated position, adjusting its movement based on real-time feedback to ensure alignment and adherence to the maximum step velocity. This careful orchestration of movement and alignment is instrumental in positioning the robot hand for an effective and efficient execution of the subsequent pushing task.

Overall, the robot is optimally initialized at a calculated distance from the box. During the pushing, the robot's orientation is meticulously aligned with the goal pushing direction. The *Smooth Movement Control* phase ensures that these actions are carried out with a level of precision and control that minimizes any potential disturbance to the box, highlighting the policy's commitment to smooth and precise execution. The continuous *Feedback Loop* represents the policy's dynamic adaptability, making real-time adjustments based on immediate feedback to ensure sustained alignment and adherence to the strategic approach outlined in the preceding phases.

C. Behavior Cloning

Behavioral cloning is a simple yet effective approach of imitation learning. It directly learns a policy by using supervised learning on observation-action pairs from expert demonstrations.

Given a set of state-action tuples $\{(s_i, a_i)\}$, we seek for the imitation policy π_ϕ. The goal of learning is to find the parameter ϕ for which π_ϕ best matches this set of provided state-action pairs. We can find this parameter using maximum-likelihood estimation:

$$\phi^* = \arg\max_{\phi} \prod_{i=0}^{N} \pi_\phi(a_i|s_i) \tag{6}$$

To enable the behavior cloning algorithm [21] to learn from our expert demonstrations, we first accumulate the trajectories generated by the expert during certain episodes, and then flatten them as input. We then solve for ϕ^* using Adaptive Moment Estimation, where it seeks to find changes in ϕ that increase the probability of each inferred demonstrator action, $\tilde{a}_i$, in the imitation policy's distribution $\pi_\phi(\cdot|s_i)$.

D. Box Pose Estimation

1) Singleshotpose Model: Inspired by single shot 2D object detectors such as YOLO [22], *singleshotpose* uses similar architecture. The 3D model of each object is parameterized with 9 control points, consist of one center point and 8 corners of the tight 3D bounding box fitted to the 3D model, similar to BB8. This parameterization is general and can be used for any rigid 3D object with arbitrary shape and topology [3].

The model takes as input a single full color image, processes it with a fully-convolutional architecture and divides the image into a 2D regular grid containing S × S cells. Each grid location in the 3D output tensor will be associated with a vector, consisting of predicted 2D image locations of the 9 control points, the class probabilities of the object and an overall confidence value.

The model predicts offsets for the 2D coordinates with respect to (c_x, c_y), the top-left corner of the associated grid cell. The predicted control point (g_x, g_y) is defined as:

$$g_x = f(x) + c_x \tag{7}$$

$$g_y = f(x) + c_y \tag{8}$$

where $f()$ is chosen to be a 1D sigmoid function in case of the center point and the identity function in case of the eight corner points. This has the effect of forcing the network to first find the approximate cell location for the object and later refine its eight corner locations.

During training, we set the confidence value of cells containing the object's center point to 1 and other cells to 0. At test time, predictions at cells with low confidence values, i.e. where the objects of interest are not present, will be pruned.

Since we only have one type of object in this project, the loss function of training are set as:

$$L = \lambda_{pt} L_{pt} + \lambda_{conf} L_{conf} \tag{9}$$

where L_{pt} and L_{conf} are the coordinate and confidence loss using mean-squared error. In our implementation, both λ_{pt} and λ_{conf} are set to 1.

At last, 6D pose estimation are calculated from the correspondences between the 2D and 3D points using a Perspective-n-Point (PnP) pose estimation method [23].

2) Datasets: To train and evaluate this method we created three datasets: Sim Dataset (2500 images), Random Dataset (4200 images) and Real Dataset (108 images) as shown in Fig 2.

3) Vision Domain Randomization: In this project we use approach similar to [20] for domain randomization, randomizing appearances of the simulation environment including:

- Color and texture of floor
- Lighting condition
- Shadow Appearance

IV. Experiments

A. Expert Demonstration

The cornerstone of our approach lies in the generation of robust training data derived from our hardcoded expert policy. The initial step in this process involved generating 300 episodes of robot interaction trajectories. This dataset was intended to capture a comprehensive range of movements and strategies employed by the expert policy in pushing the box to the target. Critical to ensuring the quality of our training data, we filtered out any episodes where the task was not successfully completed, focusing only on successful outcomes for training the BC model.

The criterion for deeming an episode successful hinges on the proximity of the box to the final target. Specifically, an episode is considered successfully concluded when the distance between the box and the final target is less than 0.06 meters. Moreover, to maintain operational boundaries, each episode is capped at a maximum of 800 time steps, ensuring task completion within a defined temporal framework.

1) Initial Training and Challenges: Upon training the behavior cloning model with this initial dataset for 2500 epochs, we observed a success rate of merely 75%. This performance was below expectations, primarily because the model tended to stop the pushing action prematurely, failing to reach the desired target position for the box consistently. A thorough investigation into this issue revealed that the data for the last time step of each episode was inaccurately recorded. Specifically, due to the environment simulator resetting before passing the final observation, the data for the last time step was erroneously reflecting the observation post-reset, rather than the terminal state of the completed task.

2) Refined Trajectories Generation: After fixing the data accuracy issue, we proceeded to generate a new set of trajectories, maintaining a consistent length of 800 time steps per episode across 300 episodes. This uniformity in episode length and the inclusion of accurate terminal observations significantly enhanced the quality of our training data, leading the success rate to escalate to 94%.

3) Dead Zone Problem: Observing the expert demonstration in simulation, we notice that when the robot goes near the base or edge, the behavior becomes unstable, leading the end effector to drift farther and farther away from the target. And that is the reason why we cannot achieve 100% success with the current expert policy. This problem is due to the manipulator's own limitation. We can avoid some ill-positioned configurations by limiting the initial position of the box to a smaller random area. However, we argue that more advanced algorithm should be developed to address this challenge, since the box could be pushed into "dead zone" in the process.

B. BC Evaluation

To assess the effectiveness of our behavior cloning (BC) model in replicating the expert policy's success, we conducted an evaluation comparing the performance of the expert policy against the BC-derived policy. The evaluation focused on the success rate of episodes in achieving the task, with success criteria based on the position error between the box and the final target.

The success of an episode was determined through two distinct criteria: a strict success criterion where the position error of the box relative to the final target was less than 0.06

TABLE I
SUCCESS EPISODES OF DIFFERENT POLICIES

Policy	n_trajectories	n_success	less-precise
Expert	100	95	-
BC-Policy	100	90	95

TABLE II
TIMESTEPS AND REWARD COMPARISON

Policy	avg. timestep	avg. reward
Expert	507.82	0.872
BC-Policy	589.69	0.864

meters, and a less-precise criterion with a threshold of less than 0.1 meters. The results of this evaluation are summarized in Table I, providing a comparative overview of the success episodes across different policies.

The data presents a compelling narrative on the capabilities of the BC model. Despite the inherently complex nature of the box-pushing task, the BC-Policy demonstrates a notable success rate, closely mirroring that of the expert policy under both success criteria. Specifically, while the BC-Policy falls slightly short of the expert's performance in the strict success criterion (n_success), it achieves parity under the less-precise criterion, indicating a high degree of efficacy in general task execution.

This evaluation underscores the BC model's potential as a robust tool for learning and replicating expert behavior in robotic tasks, albeit with a slight margin for improvement in precision. The close alignment in success rates between the BC-Policy and the Expert policy, particularly under the less-precise criterion, highlights the BC model's effectiveness in capturing the fundamental aspects of the expert's task execution strategy.

C. Box Pose Estimation

We evaluate the network both in simulated- and real environment, also the transfer of the model trained in simulated- to real environment.

1) Evaluation Metrics: We use three standard metrics to evaluate 6D pose accuracy, namely – 2D reprojection error, 3D vertices error and pose error.

When using reprojection error, a pose estimate is considered to be correct when the mean distance between the 2D projections of the object's 3D mesh vertices using the estimate and the ground truth pose is less than 5 pixels [24].

When using 3D vertices error, a pose estimate is considered to be correct when the mean distance between the true coordinates of 3D mesh vertices and those estimated given the pose is less than 10% of the object's diameter [25].

When using pose error, a pose estimate is considered to be correct when the translation and rotation difference between true pose and estimated pose is less than 5cm and 5° respectively.

2) Results: In our experiments, we trained the network on each of the three datasets using 85% of the dataset as the training set, denoted by Sim-, Random-, and Real model respectively.

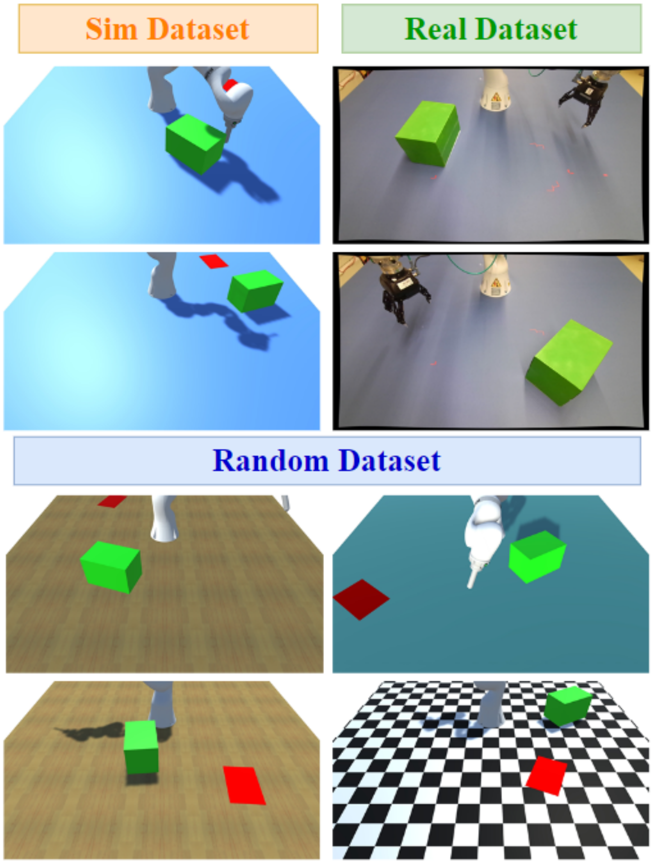

Fig. 2. Datasets generated by VTPRL and collected in real world

TABLE III
ACCURACY ON SIM DATASET OF SIM- AND RANDOM MODEL

Criteria	Sim Model	Random Model
Reprojection Error <5px	97.33%	94.93%
Vertices Error <10% diameter	98.93%	97.07%
Pose Error <5cm and 5°	99.20%	97.60%

We test the Sim- and Random model on Sim Dataset as well as a generated real-time pushing sequence from VTPRL. As shown in Table III, IV and V, Random model performed slightly less well than Sim model. Both models performed worse on pushing sequence than on Sim Dataset, which is analyzed to be due to the fact that the push sequence contains some cases where the objects overlapping with the robotic arm are occluded, while these are avoided in the dataset.

The larger error caused by occlusion for both models in the pushing sequence leads to a significant drop in reporjection error accuracy, while both were able to achieve greater than 95% accuracy on the other two evaluation criteria in both test cases. This is because the original work of *singleshotpose* used input image size of 640×480 while we use 1920×1080, which makes the requirement of reprojection error of less than 5 pixels more difficult to achieve for our model, while the other two criterion are independent of image size.

We also test all three models in Real Dataset while Real model is trained on a training set of 98 images. As shown in Table VI and VII, both Sim- and Random model transferred

TABLE IV
ACCURACY ON PUSHING SEQUENCE OF SIM- AND RANDOM MODEL

Criteria	Sim Model	Random Model
Reprojection Error <5px	78.34%	70.03%
Vertices Error <10% diameter	100.00%	97.03%
Pose Error <5cm and 5°	100.00%	99.11%

TABLE V
AVERAGE ERROR IN TWO TEST CASES
(S: SIM DATASET, P: PUSHING SEQUENCE)

Error	Sim Model	Random Model
S: Reprojection Error	2.60663 px	3.86935 px
P: Reprojection Error	3.87947 px	4.92249 px
S: Vertices Error	1.3196 cm	1.3809 cm
P: Vertices Error	1.4855 cm	1.3808 cm
S: Translation Error	1.2685 cm	1.1197 cm
P: Translation Error	1.4608 cm	1.1843 cm
S: Rotation Error	0.850657°	2.048857°
P: Rotation Error	1.270422°	2.064575°

TABLE VI
ACCURACY ON ENTIRE REAL DATASET OF ALL THREE MODELS

Criteria	Sim	Random	Real
Reprojection Error <5px	0.00%	0.00%	80.56%
Reprojection Error <10px	6.48%	12.04%	97.22%
Vertices Error <10% diameter	38.89%	40.74%	100.00%
Pose Error <5cm and 5°	11.11%	14.81%	100.00%

TABLE VII
ACCURACY ON 10 IMAGES TEST SET OF ALL THREE MODELS

Criteria	Sim	Random	Real
Reprojection Error <5px	0.00%	0.00%	0.00%
Reprojection Error <10px	0.00%	0.00%	70.00%
Vertices Error <10% diameter	30.00%	10.00%	100.00%
Pose Error <5cm and 5°	10.00%	10.00%	100.00%

to Real Dataset performed very poorly, although domain randomization demonstrated some improvement in the results. Also the Real model does not perform very well due to the fact that the number of data in the real set is too small to represent the full distribution of the real environment.

D. Pipeline Evaluation

To form a pipeline for the sim2real pushing task, we integrated the box pose estimation model into the control model. For the evaluation, we retrieve both numerical and image observation from the simulation environment. The image observation of each time step is re-scaled and transformed to tensor representation as input for the best Random box pose estimation model we achieved in previous experiments. The model's box pose outputs x and y are then used to substitute the corresponding numerical observations.

We compared the numerical observation based BC policy to the image based BC policy. As shown in Table VIII, only 3 out of 10 episodes of image based policy succeed, which means minor inaccuracy of the box position estimation can have huge negative impact to the final outcome. The results reflect the sensibility of the pushing task to disturbances, it also shows that BC policy is not robust against sensor noise.

TABLE VIII
SUCCESS EPISODES OF DIFFERENT POLICIES

Policy	n_trajectories	n_success	avg. time steps	avg. reward
BC_num	10	10	529.6	0.907
BC_image	10	3	779.9	0.613

TABLE IX
ITERATION TIME OF DIFFERENT SETUP

Policy	avg. iteration time (s)
BC_num	0.0008
BC_image	0.0523

The average time steps indicate that the image based BC policy tend to exceed the maximum 800 time steps in each episode, leading the pushing task to fail. One possible reason could be the deviation of the estimated box pose from the true pose causing the calculated pushing direction to change randomly, wasting a lot of time to redirect to the true target direction.

The average reward of the image based BC policy is not optimal but comparable to the numerical one, showing that the box pose estimation is accurate enough to be applied. If we loose up the success criterion and increase maximum time steps, the image based BC policy may achieve better results. This requires further investigation in the future.

Table IX shows the average iteration time of each policy in second. We can see that with the image based box pose estimation, the pipeline takes significantly more time to run. However, on our experiment device (GPU: RTX-3080), the model still achieved about 19 frames per second. We believe that when deployed on a better GPU, this model has adequate real-time capability.

V. CONCLUSION

This article has detailed the development, implementation, and evaluation of a hardcoded expert policy designed for a robotic hand tasked with pushing a box to a specified target, the subsequent training of a Behavior Cloning (BC) policy based on the expert's demonstrated actions and the incorporation of visual perception.

The expert policy demonstrated a high degree of effectiveness, achieving a success rate of 95% under defined success criteria, except in scenarios involving the ”dead zone”. The BC policy, trained on trajectories generated by the expert policy, showed commendable performance with a 90% success rate after rectifying initial data inaccuracies and refining the training process. These results highlight the potential of BC models to closely replicate expert behavior, achieving substantial success in complex tasks with nuanced execution requirements.

The *singleshotpose* network for box pose estimation demonstrated adequate performance, both in terms of pose estimation accuracy and in terms of real-time performance. The domain randomization method used for sim-to-real transfer does improved the model performance when deployed to Real Dataset.

The pipeline implementation and evaluation highlight the viability of behavior cloning for robotic tasks, but the results also show its instability to noise caused by sensing.

Future Directions: Looking forward, enhancing the robustness of the BC policy to handle edge cases, such as the ”dead zone”, presents a valuable avenue for research. Further studies could explore the integration of reinforcement learning to fine-tune the BC policy post-training, potentially improving its adaptability and success rate in varied scenarios. Additionally, expanding the action and observation spaces could introduce new dimensions of complexity and realism into the task, offering deeper insights into the capabilities of behavior cloning in robotic applications.

For the visual perception model, more complete randomization, adjustment of the network structure, and other sim-to-real methods such as domain adaptation can be attempted.

Appendix A
Model Analysis Based on Saliency Map

To demonstrate the effect of domain randomization and to explain the reason for the poor performance of the models when transferred to the Real Dataset, we visualized the saliency map for analysis.

On simulated images, as shown in Fig 3, benefiting from domain randomization, the Random model does focus less on the floor other than the object compared to the Sim model. However, we can also observe that the Random model puts a significant portion of its attention on the robotic manipulator, whose appearance is not being randomized in the domain randomization.

We believe that the more important the pixels in the object region are in the prediction process of the model, the better the prediction results will be. This can be seen in the saliency test Fig 4 that uses the three models on real images, in which the Real model focuses entirely on the object and its boundaries, while the other two models capture many undesired features.

Despite the fact that domain randomization succeeds in making the model focus less on areas outside of the object, i.e., the floor and shadows that get randomized, it also made the model focus more on stuff like the robotic manipulator that are not randomized but are not desired. Also, not randomizing the object’s appearance such as color may result in the model not being forced to focus more on the object’s region. This insufficient randomization is very likely the reason why the model performs poorly on Real Datasets.

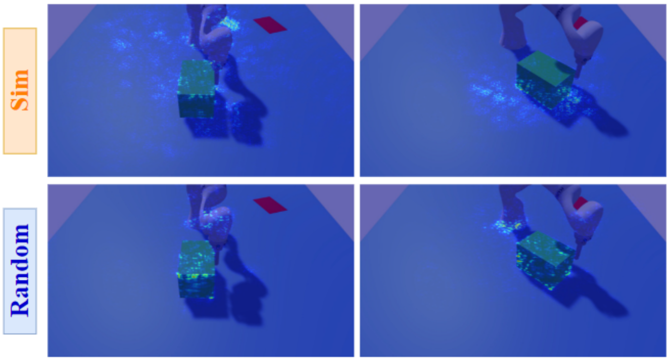

Fig. 3. Saliency map on simulated images of Sim- and Random model

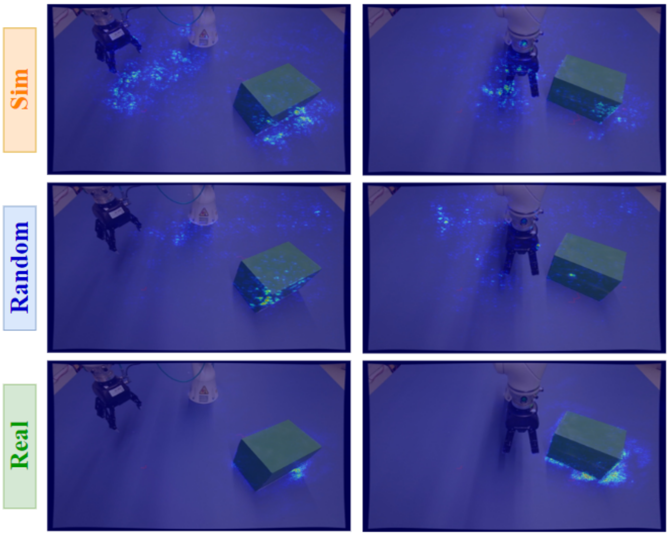

Fig. 4. Saliency map on real images of Sim-, Random- and Real model

Appendix B
Data Distribution of Datasets

The x,y position distribution and Rotation around z-axis distribution of datasets used in this project are shown in Fig 5 and Fig 6.

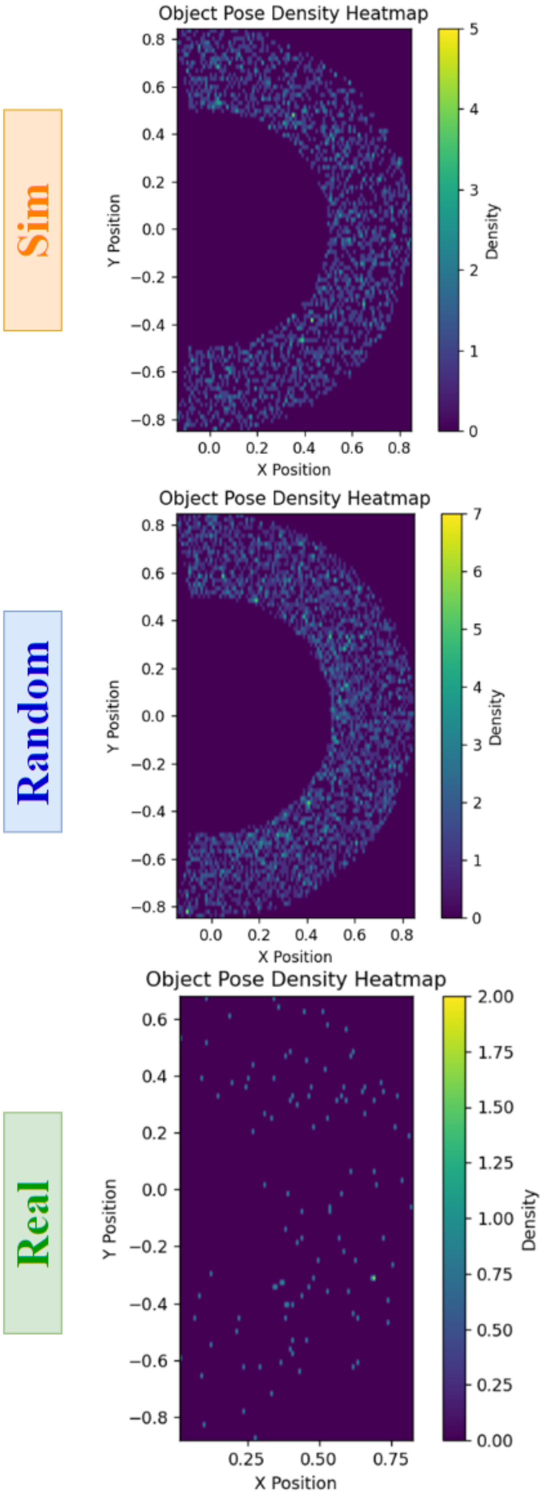

Fig. 5. Position distribution of Sim-, Random- and Real Dataset

.

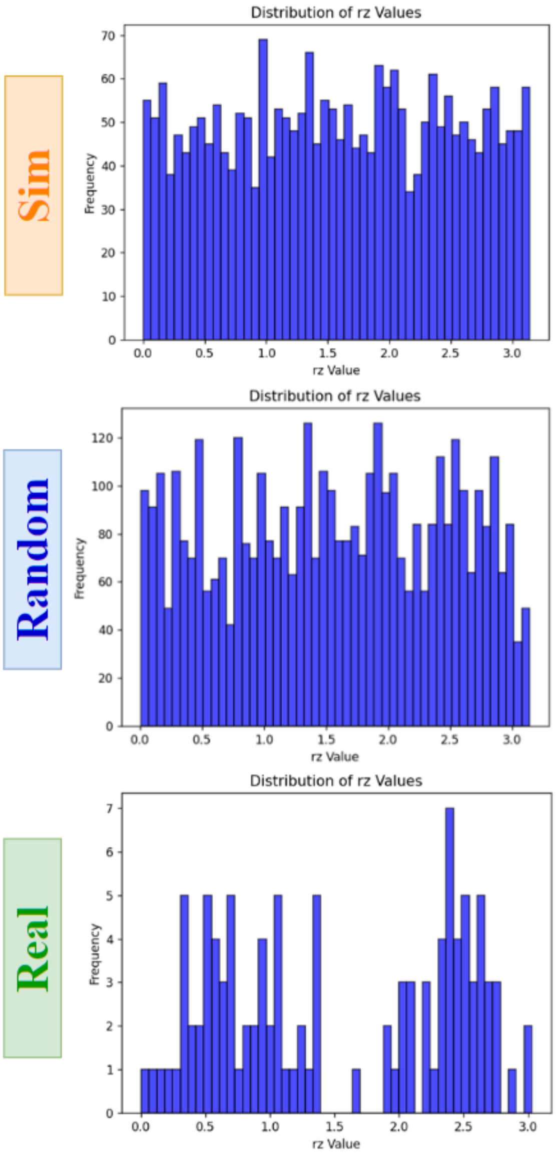

Fig. 6. Rotation distribution of Sim-, Random- and Real Dataset

.

References

[1] S. Levine, P. Pastor, A. Krizhevsky, J. Ibarz, and D. Quillen, "Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection," *The International Journal of Robotics Research*, vol. 37, no. 4-5, pp. 421–436, 2018. [Online]. Available: https://doi.org/10.1177/0278364917710318

[2] F. Muratore, F. Ramos, G. Turk, W. Yu, M. Gienger, and J. Peters, "Robot learning from randomized simulations: A review," *Frontiers in Robotics and AI*, vol. 9, 2022. [Online]. Available: https://www.frontiersin.org/articles/10.3389/frobt.2022.799893

[3] B. Tekin, S. N. Sinha, and P. Fua, "Real-time seamless single shot 6d object pose prediction," in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2018, pp. 292–301.

[4] J. Z. Woodruff and K. M. Lynch, "Planning and control for dynamic, nonprehensile, and hybrid manipulation tasks," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2017, pp. 4066–4073.

[5] M. T. Mason and K. M. Lynch, "Dynamic manipulation," in *Proceedings of 1993 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'93)*, vol. 1. IEEE, 1993, pp. 152–159.

[6] F. Ruggiero, V. Lippiello, and B. Siciliano, "Nonprehensile dynamic manipulation: A survey," *IEEE Robotics and Automation Letters*, vol. 3, no. 3, pp. 1711–1718, 2018.

[7] K. M. Lynch and M. T. Mason, "Stable pushing: Mechanics, controllability, and planning," *The international journal of robotics research*, vol. 15, no. 6, pp. 533–556, 1996.

[8] M. T. Mason, "On the scope of quasi-static pushing," in *International Symposium on Robotics Research, 1986*, 1986, pp. 229–233.

[9] S. Zhu, A. Kimmel, and A. Boularias, "Information-theoretic model identification and policy search using physics engines with application to robotic manipulation," *arXiv preprint arXiv:1703.07822*, 2017.

[10] A. Byravan and D. Fox, "Se3-nets: Learning rigid body motion using deep neural networks," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2017, pp. 173–180.

[11] K. Lowrey, S. Kolev, J. Dao, A. Rajeswaran, and E. Todorov, "Reinforcement learning for non-prehensile manipulation: Transfer from simulation to physical system," in *2018 IEEE International Conference on Simulation, Modeling, and Programming for Autonomous Robots (SIMPAR)*. IEEE, 2018, pp. 35–42.

[12] D. Lowe, "Object recognition from local scale-invariant features," in *Proceedings of the Seventh IEEE International Conference on Computer Vision*, vol. 2, 1999, pp. 1150–1157 vol.2.

[13] B. Drost, M. Ulrich, N. Navab, and S. Ilic, "Model globally, match locally: Efficient and robust 3d object recognition," in *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2010, pp. 998–1005.

[14] Y. Xiang, T. Schmidt, V. Narayanan, and D. Fox, "Posecnn: A convolutional neural network for 6d object pose estimation in cluttered scenes," 2018.

[15] Y. Li, G. Wang, X. Ji, Y. Xiang, and D. Fox, "Deepim: Deep iterative matching for 6d pose estimation," *International Journal of Computer Vision*, vol. 128, no. 3, p. 657–678, Nov. 2019. [Online]. Available: http://dx.doi.org/10.1007/s11263-019-01250-9

[16] M. Rad and V. Lepetit, "Bb8: A scalable, accurate, robust to partial occlusion method for predicting the 3d poses of challenging objects without using depth," in *2017 IEEE International Conference on Computer Vision (ICCV)*, 2017, pp. 3848–3856.

[17] W. Zhao, J. P. Queralta, and T. Westerlund, "Sim-to-real transfer in deep reinforcement learning for robotics: a survey," in *2020 IEEE Symposium Series on Computational Intelligence (SSCI)*, 2020, pp. 737–744.

[18] K. Kristinsson and G. Dumont, "System identification and control using genetic algorithms," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 22, no. 5, pp. 1033–1046, 1992.

[19] M. Wang and W. Deng, "Deep visual domain adaptation: A survey," *Neurocomputing*, vol. 312, p. 135–153, Oct. 2018. [Online]. Available: http://dx.doi.org/10.1016/j.neucom.2018.05.083

[20] J. Tobin, "Real-world robotic perception and control using synthetic data," Ph.D. dissertation, EECS Department, University of California, Berkeley, Jun 2019. [Online]. Available: http://www2.eecs.berkeley.edu/Pubs/TechRpts/2019/EECS-2019-104.html

[21] A. Gleave, M. Taufeeque, J. Rocamonde, E. Jenner, S. H. Wang, S. Toyer, M. Ernestus, N. Belrose, S. Emmons, and S. Russell, "imitation: Clean imitation learning implementations," arXiv:2211.11972v1 [cs.LG], 2022. [Online]. Available: https://arxiv.org/abs/2211.11972

[22] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," 2016.

[23] V. Lepetit, F. Moreno-Noguer, and P. Fua, "Epnp: An accurate o(n) solution to the pnp problem," *International Journal of Computer Vision*, vol. 81, 02 2009.

[24] E. Brachmann, F. Michel, A. Krull, M. Y. Yang, S. Gumhold, and C. Rother, "Uncertainty-driven 6d pose estimation of objects and scenes from a single rgb image," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 3364–3372.

[25] S. Hinterstoisser, V. Lepetit, S. Ilic, S. Holzer, G. Bradski, K. Konolige, and N. Navab, "Model based training, detection and pose estimation of texture-less 3d objects in heavily cluttered scenes," vol. 7724, 10 2012.