COCOI: Contact-aware Online Context Inference for Generalizable Non-planar Pushing

Zhuo Xu[1,2*], Wenhao Yu[3], Alexander Herzog[1], Wenlong Lu[1], Chuyuan Fu[1], Masayoshi Tomizuka[2], Yunfei Bai[1], C. Karen Liu[4], Daniel Ho[1]

***Abstract*— General contact-rich manipulation problems are long-standing challenges in robotics due to the difficulty of understanding complicated contact physics. Deep reinforcement learning (RL) has shown great potential in solving robot manipulation tasks. However, existing RL policies have limited adaptability to environments with diverse dynamics properties, which is pivotal in solving many contact-rich manipulation tasks. In this work, we propose Contact-aware Online COntext Inference (COCOI), a deep RL method that encodes a context embedding of dynamics properties online using contact-rich interactions. We study this method based on a novel and challenging non-planar pushing task, where the robot uses a monocular camera image and wrist force torque sensor reading to push an object to a goal location while keeping it upright. We run extensive experiments to demonstrate the capability of COCOI in a wide range of settings and dynamics properties in simulation, and also in a sim-to-real transfer scenario on a real robot (Video: `https://youtu.be/nrmJYksh1Kc`).**

I. Introduction

Contact rich manipulation problems are ubiquitous in the physical world. In millions of years of evolution, humans have developed the remarkable capability to understand environment physics, so as to achieve general contact rich manipulation skills. Combining visual and tactile perception with end-effectors like fingers and palms, humans effortlessly manipulate objects with various shapes and dynamics properties in complex environments. Robots, on the other hand, lack this capability – due to the difficulty of understanding high dimensional perception and complicated contact physics. Recent development in deep reinforcement learning (RL) has shown great potential towards solving manipulation problems [1]–[3] by leveraging two key advantages. First, the representative capability of a deep neural network structure can capture complicated dynamics models. Second, control policy optimization explores vast contact interactions. However, contact-rich manipulation tasks are generally dynamics-dependent; since the RL policies are trained in a specific dynamics setting, they specialize within the training scenario and are vulnerable to variations of dynamics. Learning a policy that is robust to dynamics variations is pivotal for deployment to scenarios with diverse object dynamics properties.

In this work, we design a deep RL method that takes multi-modal perception input and uses deep representative structure to capture contact-rich dynamics properties. The proposed method, Contact-aware Online COntext Inference (COCOI), uses prior camera frames and force readings in a contact-aware way to encode dynamics information into a latent context representation. This allows the RL policy to plan with dynamics-awareness and improves in robustness against domain variations.

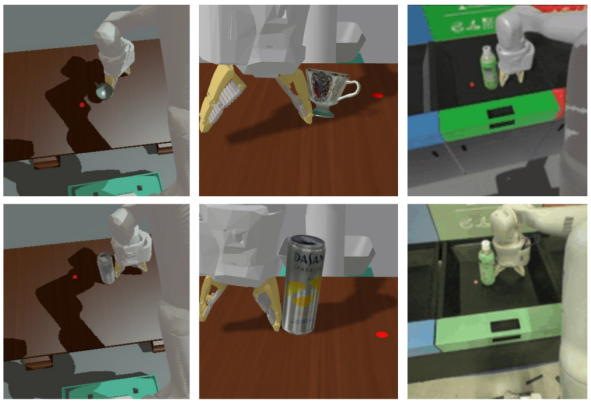

Fig. 1: Our method, COCOI, achieves dynamic-aware, non-planar pushing of an upright 3D object. The method is robust against domain variations, including various objects and environments, in both simulation and the real world. The first and second columns show the table simulation setting in the robot's perspective and the third party perspective, respectively. The third column shows the simulated and real world trash bin settings in the robot perspective.

We apply COCOI to a novel pushing task where dynamics property reasoning plays a vital role: the robot needs to push an object to a target location while avoiding knocking it over (Fig. 1). Prior work in pushing mostly focus on objects inherently stable when pushed on a flat surface. This essentially reduces the task to a 2D planar problem [4] [5]. As a result, they cannot handle our proposed class of "non-planar pushing tasks" where real-world 3D objects can move with the full six degrees of freedom during pushing. Despite being commonly seen in everyday life, these tasks have the following challenges:

1) Visual perception: unlike in planar pushing, where concrete features can be retrieved from a top down view, in non-planar pushing, key information can not be easily extracted from the third angle perspective image.
2) Contact-rich dynamics: the task dynamics properties

[1]Everyday Robots, X The Moonshot Factory, Mountain View, CA, USA
[2]University of California, Berkeley, Berkeley, CA, USA
[3]Robotics at Google, Mountain View, CA, USA
[4]Stanford University, Stanford, CA, USA
*Work done as an AI Resident at Everyday Robots.

are not directly observable from raw sensor information. Furthermore, in our non-planar pushing task, dynamics property reasoning is vital to avoid knocking the object over.

3) Generalizable across domain variations: the policy needs to be effective for objects with different appearances, shapes, masses, and friction properties.

The paper is structured as follows: In Section II, we review and compare with related works. We formalize the problem of interest in Section III and describe the RL pushing controller in Section IV. We explain COCOI in Section V and the handling of the sim-to-real visual gap in Section VI. The capability of COCOI is validated with the experiments in Section VII, before we make the conclusions in Section VIII.

II. Related Work

In order to adapt a learning based policy to different task settings, variation of dynamics, and unknown disturbances, previous researchers train policies on randomized environments with possible variations to gain robustness. Rajeswaran et al. [6]. and Peng et al. [7] learn RL policies in randomized simulated environments and directly apply them to different domains. Chebotar et al. leverage real world experiences to adapt the simulation randomization [8]. Other work aim to derive a representation of the task context for domain specific planning. Rakelly et al. take a meta learning direction to learn patterns within the task to help plan [9]. Yu et al. learn a universal policy and use an online system identification module to learn the dynamics parameters [10]. Beyond learning based methods, planning-control framework can also overcome the domain gap. Harrison et al. use model predictive control to track a reference trajectory derived from a learning policy [11], and Xu et al. use a robust controller to reject dynamics variation and external disturbances [12] [13]. COCOI is different from previous methods in that we focus on the contact-rich manipulation problem, and use a contact-aware structure to infer the contact dynamics context online.

Another thread of related research is the pushing task [14]. Earlier studies on pushing are based on analytical approaches and consider quasi-static planar pushing [15]. Later, researchers have introduced data driven methods to model pushing physics. Zhou et al. develop a dynamics model for planar friction and design force control method for planar sliding [4]. Yu, Bauza et al created a planar pushing dataset for data-driven modeling [16], [17]. More recent works involve using deep learning to learn object properties [5], [18], [19], but they only focus on planar pushing problems. Stuber et al. [20] and Byravan et al. [21] learn motion models for pushing simple blocks.

On non-planar pushing, the majority of works are still in the exploration stage. Ridge et al. propose an object image fragment matching method for 3D objects [22], albeit for a limited library of objects. Zhu et al. use a simulator as a predictive model [23]. Kopicki et al. learn a 3D dynamics model in the PhysX simulator [24]. These works all rely on carefully selected objects and precise detection and localization; our model takes monocular camera image input and solves a generalized non-planar pushing task for diverse objects.

III. Problem Statement

In this section, we formally define the proposed non-planar pushing task and formulate the problem using a Partially Observable Markov Decision Process (POMDP). We focus on the class of pushing tasks where maintaining the upright pose of the object is critical. For example, when pushing a glass of water on the table, the glass should not tilt and spill. In our work, we assume that an object is randomly placed on a flat surface with an upright initial pose. The task objective is to use the robot end effector to push the object to a random target position on the surface, while maintaining its upright orientation. The object can have irregular shape and mass distribution, and the robot may push at any point on the object, making the contact dynamics physics complicated.

We use state $\boldsymbol{s} \in S$ to represent the full task state. In a POMDP, the state is not directly available and can only be inferred using observations. Concretely, we use the image captured using the robot's monocular camera as the high dimensional observation (Fig. 1, first and third columns), in which the target location is rendered using a red dot. We also include a low dimensional proprioception observation of the gripper height and open/close status. We use state and observation interchangeably in the following sections. The action $\boldsymbol{a} \in A$ contains the designated position and orientation of the gripper, the gripper open/close command, and a termination boolean. The system transition function $T : S \times A \to S$ follows the physical model, and we use a sparse binary reward function $R : S \times A \to \mathbb{R}$ with value 1 when the distance between the object and the target is smaller than a threshold and the object is upright. The task objective is to maximize the expectation of the discounted cumulative reward, or the return

$$R = \mathop{\mathbb{E}}_{\{(\boldsymbol{s_t},\boldsymbol{a_t})\}} \left[\sum_{t=0}^{t=\infty} \gamma^t r_t \right] \tag{1}$$

where γ is the discount coefficient, and $r_t = R(\boldsymbol{s_t}, \boldsymbol{a_t})$ is the reward obtained at time t.

IV. Learning Basic Pushing Controller

We use Q-learning to train object pushing control policy. The definition of the Q function is

$$Q(\boldsymbol{s}, \boldsymbol{a}) = \mathop{\mathbb{E}}_{\{(\boldsymbol{s_t},\boldsymbol{a_t})\}} \left[\sum_{\boldsymbol{s},\boldsymbol{a}}^{t=\infty} \gamma^t r_t \right], \tag{2}$$

the expected return starting from state $\boldsymbol{s}$ and taking $\boldsymbol{a}$. The Q function satisfies the Bellman function:

$$Q(\boldsymbol{s_t}, \boldsymbol{a_t}) = \mathbb{E}_{\boldsymbol{s_{t+1}}}[R(\boldsymbol{s_t}, \boldsymbol{a_t}) + \gamma \cdot Q(\boldsymbol{s_{t+1}}, \pi(\boldsymbol{s_{t+1}}))] \tag{3}$$

where $\pi(\boldsymbol{s_{t+1}})$ represents the action selected by the optimal policy at the current iteration and state $\boldsymbol{s_{t+1}}$. The optimal

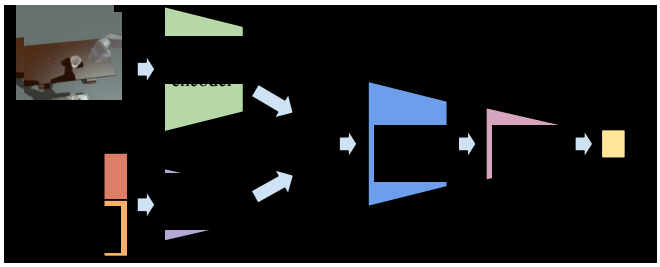

Fig. 2: The feed forward neural network Q function for object pushing. The stacked current and initial images are fed into a convolutional neural network (CNN) encoder, and the low dimensional input is fed into a fully connected network (FCN) encoder. The output of these two streams are added together and then fed into another CNN-followed-by-FCN structure, the Q value prediction head.

policy is defined by:

$$\pi(\boldsymbol{s_t}) = \arg\max_{\boldsymbol{a}\in A} Q(\boldsymbol{s_t}, \boldsymbol{a}) \tag{4}$$

One Q-learning iteration uses the collected data tuples $(\boldsymbol{s_t}, \boldsymbol{a_t}, \boldsymbol{s_{t+1}})$ in two steps:

1) Estimate the optimal policy output

$$\pi(\boldsymbol{s_{t+1}}) = \arg\max_{\boldsymbol{a}\in A} Q(\boldsymbol{s_{t+1}}, \boldsymbol{a}) \tag{5}$$

2) Minimize the Bellman error

$$\|Q(\boldsymbol{s_t}, \boldsymbol{a_t}) - [R(\boldsymbol{s_t}, \boldsymbol{a_t}) + \gamma \cdot Q(\boldsymbol{s_{t+1}}, \pi(\boldsymbol{s_{t+1}}))]\| \tag{6}$$

For the object pushing task, we represent the Q function with a two stream deep neural network $Q_\theta(\boldsymbol{s_t}, \boldsymbol{a_t})$ parameterized by θ, as shown in Fig. 2, similiar to [2]. The high dimensional stream feeds stacked current and initial images into a convolutional neural network (CNN) encoder. The low dimensional stream feeds stacked low dimensional state (gripper height and open/close status) and action (designated gripper pose, the gripper open/close command, and a termination boolean) into a fully connected network (FCN) encoder. The outputs of the two streams are added together and fed into another CNN-followed-by-FCN structure to predict the Q function value for the pushing task.

For Q function network training, we adopt QT-Opt, a distributional variant of the Q learning framework for continuous state-action tasks [2], [25]. Concretely, we use distributed workers to collect data tuples $(\boldsymbol{s_t}, \boldsymbol{a_t}, \boldsymbol{s_{t+1}})$, and we store them into a replay buffer. Each optimization iteration samples a batch of data tuples. For equation (5), the optimal policy output is estimated using an online sampling-based cross entropy method (CEM) based on the current Q function. For equation (6), the Q function network weights are updated using gradient descent with the following loss function:

$$l(\theta) = \|Q_\theta(\boldsymbol{s_t}, \boldsymbol{a_t}) - [R(\boldsymbol{s_t}, \boldsymbol{a_t}) + \gamma \cdot Q_\theta(\boldsymbol{s_{t+1}}, \pi(\boldsymbol{s_{t+1}}))]\| \tag{7}$$

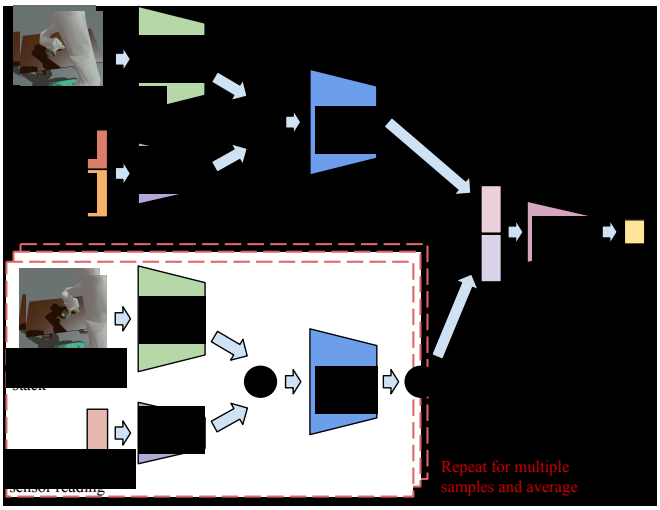

Fig. 3: The proposed COntext Inference (COI) module, which works in parallel with the state-action stream. The COI takes as input a history sample consisting of the stacked pre-impact and post-impact images and the force reading, to infer the dynamics representations. The representations of different samples are averaged and concatenated to derive the state-action representation, which is then fed into a final Q value prediction network. Networks with same color share architectures, but not necessarily network weights.

V. Contact-aware Online Context Inference

A. Online Context Inference

The architecture in Fig. 2 has been demonstrated on challenging tasks like grasping [2]. However, given it only has access to a single sensory input, it is not able to infer the dynamics properties of the object, which is necessary for our non-planar pushing task. In this section, we describe online COntext Inference (COI), a module that takes history observation samples and encodes them into a dynamics context representation – thus equipping the control policy with the ability to infer dynamics of the object.

As shown in Figure 3, COI consists of a set of additional streams in the policy network that encode history sensor observations into a dynamics context representation. Each stream of COI takes a pair of consecutive sensory inputs separated in time by 0.5s (the sensor update interval in our robot system). We denote each sensory input pair as a tuple $H_\tau = (I_\tau, I_{\tau+1}, f_\tau)$, where I and f refer to the camera image and force reading respectively, and τ represents the time at which the sensor inputs are retrieved. The encoded sensory input for each stream is then averaged to obtain the final dynamics context representation of COI, which is concatenated with the state-action representation to estimate the Q value.

B. The Contact-aware Sampling Strategy

Given an architecture that can process multiple history sensor observation pairs, key questions are:

1) How many history samples should we include?

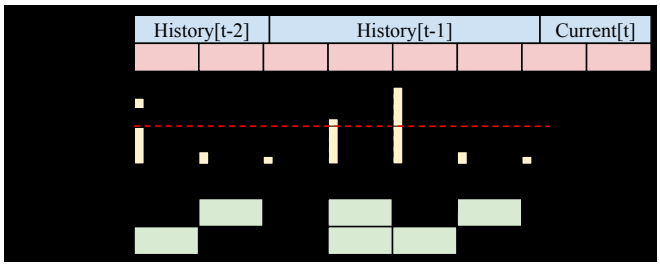

Fig. 4: Illustrations of the sampling strategies. For VCOI, the samples are retrieved with a uniform sampling interval. CO-COI takes a contact-aware sampling method which actively checks the contact force, and only retrieve samples when the force magnitude is larger than 1 Newton.

2) At what time should we retrieve the sensor observations?

With a higher number of history samples, the policy has more information to infer a potentially less noisy object dynamics representation, at the cost of higher computation time and memory. In our experiments, we found three history samples to be a reasonable number to achieve good inference performance with manageable computation cost.

The timings at which sensor observations are sampled is vital for the performance of COI. An arbitrarily sampled sensor observation pair may contain limited information about the object dynamics (e.g. when gripper is far away from the object) and contribute little to the learning performance. To ensure that each history sample contains useful information, we propose a contact-aware sampling strategy which actively checks the force torque sensor mounted at the robot gripper and only collects a sample when the contact force magnitude is considerably large (> 1 N), as shown in Fig. 4. This strategy guarantees the samples to be representative, in that the gripper and the object are in contact. We call this sampling strategy COntact-aware-COI, or COCOI.

In our experiments, we validate the performance of CO-COI by comparing it to a naïve strategy: vanilla COI, or VCOI, where history samples are retrieved with a uniform sampling interval, as shown in Figure 4.

VI. GAN for Visual Gap Bridging

In order to adapt the RL policy trained in simulation to the real world, we also need to overcome the discrepancy between the rendered, simulated image and the image captured by a real-camera. We adopt RetinaGAN, a generative adversarial network (GAN) approach to generate synthetic images that look realistic with object-detection consistency [26]. See the concurrent submission[1] for further details; this work focuses on contact inference independently of the visual discrepancy. Qualitative performance is shown in Fig. 5. We train the RL policy with simulation data only and directly deploy it on the real robot.

[1] https://retinagan.github.io

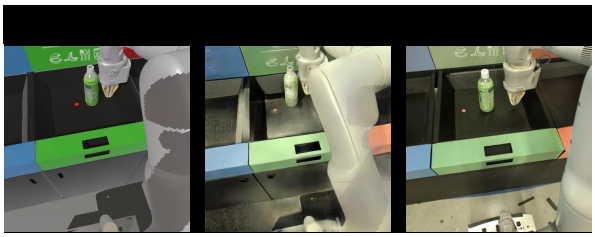

Fig. 5: Images of the pushing-in-station setting in simulation, with RetinaGAN visual adaptation, and in the real world.

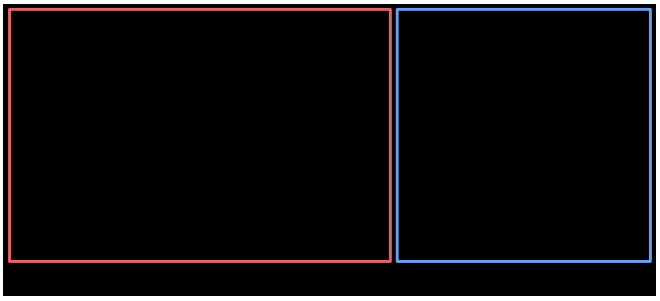

Fig. 6: A subset of the 75 different 3D objects used for training and testing.

VII. Experiments

A. Setup

We train the control policies in PyBullet simulation [27]. We first define a flat surface randomly placed in front of the robot. The surface can be either a desk surface or a designated flat area inside a trash bin, as shown in Fig. 1. We use a 3D model set containing 75 different objects, such as cups, bottles, cans, mugs, etc. (Fig. 6). We divide the objects into a training set containing 64 objects and an unseen testing set of 11 objects including a stack of cups. Note that the upper cups in the stack have the degrees of freedom to tilt when pushed, which makes the pushing task more challenging. In PyBullet, the contact physics between the robot gripper and the object is modelled using a point contact model with an elliptic friction cone.

At the beginning of each episode, the object and the target position are randomly placed within a rectangular area. We set the object upright on the surface, and the target position is rendered using a red dot. The robot gripper is initialized at a randomized position beside the object. The push policy controls the robot gripper to push the object to the red dot. An episode is considered successful if the object is pushed to within 5 cm of the target, and the object tilting angle remains smaller than 0.1 rad. We also apply a small penalty at each timestep to encourage faster execution. The discount factor γ of the POMDP is 0.9.

B. Policy Models

The deep neural networks for the Q functions are constructed as in Fig. 2 and Fig. 3, and the blocks with the same names share the same network architecture. The detailed architecture is shown in Table I.

TABLE I: Architecture of each module in the Q networks

Block name	Architecture
Images input	Tensor with shape (472, 472, 6)
Low dim state input	2 dim vector
Low dim action input	7 dim vector
Force torque sensor reading	3 dim vector
CNN encoder	Conv(64, 6, 2) MaxPool(3) Repeat x6: Conv(64, 5, 1) MaxPool(3)
FCN encoder	FC(256) FC(64) Reshape(1, 1, 64)
CNN	Conv(64, 3, 1) MaxPool(2) Repeat x3: Conv(64, 3, 1)
state-action representation	(8, 8, 64) dim matrix
dynamics context representation	(8, 8, 64) dim matrix
FCN	FC(64) FC(64) Sigmoid
Q function	1 dim vector

We train and compare four models:

1) The baseline model: the basic feed forward Q function network as described in Section IV and shown in Fig. 2. This model is the most straightforward and shows the capability of the baseline RL method.
2) The VCOI model: the VCOI method as shown in Figure 3. The history observations are sampled using uniform sampling.
3) The oracle: the architecture of this model is the same as the baseline, but we expand the low dimensional state input with 2 key, unobservable dynamics parameters: the object mass and friction coefficient. We directly extract the ground truth parameter values from the simulator to obtain an oracle model.
4) The COCOI: the proposed COntact-aware Online COntext inference model as shown in Figure 3. The active contact-aware sampling strategy is applied.

C. Policy Learning and Comparison

We adopt the QT-Opt framework [2] to train the policies. In the training setting, the objects and goal locations are randomly sampled in a 0.5m $\times$ 0.3m area, the object mass is randomized from 0.05 kg to 0.5 kg, and the friction coefficient is randomized from 0.5 to 1.0. We train the models using stochastic gradient descent, using a learning rate of 0.0001 and momentum of 0.9. We train the models with a batch size of 2048 for 80k steps.

In the early stage of training, we use a rule-based scripted policy, which moves the gripper along the line connects the object and the target, to generate successful episodes and improve the exploration efficiency. This rule-based method only achieves less than 5% success rate, illustrating the difficulty of the non-planar pushing task. During training, we observe that the policy first obtains the capability to solve easier scenarios where the object-goal distance is short and then gradually learns to push objects that are initialized far from the goal. Fig. 7 shows a comparison of training performance for the four models.

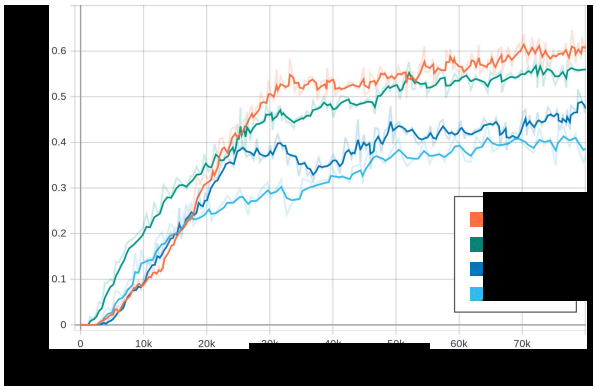

Fig. 7: Training success rate as a function of training steps for the four RL models in Section VII-B

.

As shown in Fig. 7, COCOI and the oracle model perform significantly better than VCOI and the baseline model. COCOI reaches even higher success rate than the oracle, which indicates that COCOI is capable to capture more than just the oracle information (object mass and friction coefficient) - there are other factors such as the object shape and contact point that affect the dynamics properties.

D. Performance under Domain Variations

Overall, the RL policy succeeds in pushing the upright object and learns to perform smart behaviors. For example, the robot can break contact with object when the object leans, and it can open its gripper to use its finger to make subtle impacts. We evaluate pushing performance for the four models on a large variety of settings.

First, the initial range of the object and the target are varied and the results are shown in Table II. COCOI consistently outperforms VCOI and baseline and achieves a similar performance to the oracle. Specifically, COCOI shows an average relative improvement of 50% and 20% success rate compared to the baseline and VCOI, respectively.

Second, we fix the initial object placement to 0.4m $\times$ 0.3m and vary dynamics properties. We change the friction coefficient and the object mass to be outside the training range. We also test performance of the models on unseen objects and a stack of cups whose inertia can change during pushing. Evaluation results with these setting variations are reported in Table III.

Across different domains, COCOI consistently outperforms other methods. The performance of the oracle model is especially poor in cases where the real friction parameter is not in the range of the training set. This could be due to the policy overfitting to the input dynamics parameters.

TABLE II: Comparison of success rate for models evaluated with different initial placement settings.

Initial range	0.3m x 0.6m	0.3m x 0.5m	0.3m x 0.4m	0.3m x 0.3m	0.25m x 0.5m	0.25m x 0.5m
Baseline	26.5%	34.1%	51.1%	59.2%	40.0%	42.5%
COI	36.0%	43.2%	63.4%	72.2%	49.0%	50.8%
Oracle	**43.8%**	53.9%	**73.8%**	**78.2%**	60.5%	62.4%
COCOI	43.0%	**59.3%**	73.2%	75.7%	**64.7%**	**62.6%**

TABLE III: Comparison of success rate for models evaluated with different dynamics properties settings.

Model	default setting	0.0-0.5 friction	1.0-1.5 friction	0.5-1.0kg mass	Unseen objects	Cup stack
Baseline	51.1%	53.8%	32.1%	32.6%	42.5%	26.2%
COI	63.4%	59.6%	44.2%	44.7%	48.3%	38.8%
Oracle	**73.8%**	39.6%	38.2%	41.4%	**60.9%**	36.5%
COCOI	73.2%	**72.6%**	**47.2%**	**49.9%**	58.9%	**43.7%**

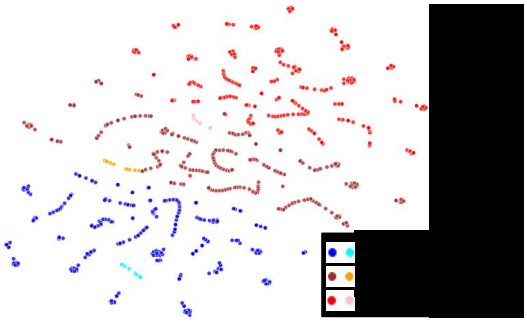

Fig. 8: t-SNE visulization of the inferred context representation for three different dynamics parameters settings. For each setting, the context representations from a randomly chosen episode is highlighted with a brighter color. The clusters show clear separation and are distributed with an order consistent with the friction magnitude.

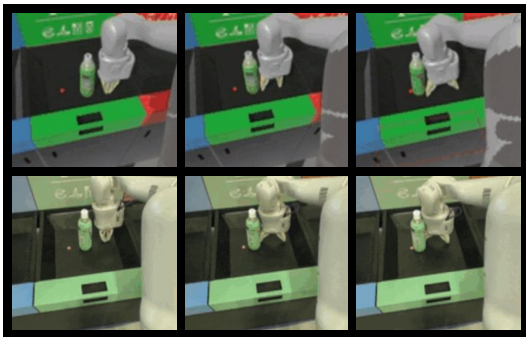

Fig. 9: Visualization of sim-to-real pushing policy transfer.

E. Interpretation of Context Representations

To inspect the dynamics context learned by COCOI, we visualize the inferred representations for three settings with different dynamics parameters. For each setting, we run our controller for 20-30 episodes to fill a buffer of 256 dynamics context representations. We then visualize these representations using a combination of principle component analysis (PCA) and t-distributed stochastic neighbor embedding (t-SNE) [28] (Fig. 8). The visualization shows clear separation between settings, which indicates COCOI learns to infer the dynamics properties. Also, representations within one episode are grouped closer to each other than to other episodes, suggesting that learned representations are consistent and structured. Moreover, we observe the order of the clusters is consistent with the dynamics properties: the clusters with the largest mass and friction and the smallest mass and friction are farthest apart, while the cluster with intermediate parameters is in the center.

F. Real World Deployment

To test real world deployment, we design a push-in-bin task in both the simulator and in the real world, as shown in Fig. 1. We adopt the method described in Section VI and train a RetinaGAN model to adapt the simulation images to synthetic images with realistic appearance. We train the pushing policy with COCOI based on synthetic images and run 10 real world pushing episodes. We achieve 90% success, demonstrating the capability of our 3D pushing policy to overcome both the visual and dynamics domain gap. Fig. 9 shows example sequences in simulation and the real world.

VIII. Conclusions

We propose COCOI, a deep RL method that uses history robot-object interaction samples to infer contact dynamics context, and we show it outperforms baseline in contact-rich manipulation tasks with domain variations. We design and study COCOI on a non-planar pushing task commonly seen in everyday life. Extensive experiments demonstrate the capability of COCOI in a wide range of settings, dynamics properties, and sim-to-real transfer scenarios.

There are many promising future work directions to pursue based on our approach. For example, we study the non-planar pushing task with a single object on the surface. It would be interesting to train manipulation controllers that can perform pushing with multiple objects or in a cluttered environment. In addition, extending our approach to push non-rigid objects such as a piece of cloth is another important direction that can further expand the capability of our controller.

REFERENCES

[1] S. Levine, C. Finn, T. Darrell, and P. Abbeel, "End-to-end training of deep visuomotor policies," *The Journal of Machine Learning Research*, vol. 17, no. 1, pp. 1334–1373, 2016.

[2] D. Kalashnikov, A. Irpan, P. Pastor, J. Ibarz, A. Herzog, E. Jang, D. Quillen, E. Holly, M. Kalakrishnan, V. Vanhoucke, *et al.*, "Qt-opt: Scalable deep reinforcement learning for vision-based robotic manipulation," *arXiv preprint arXiv:1806.10293*, 2018.

[3] J. Lee, J. Hwangbo, L. Wellhausen, V. Koltun, and M. Hutter, "Learning quadrupedal locomotion over challenging terrain," *Science Robotics*, vol. 5, no. 47, 2020.

[4] J. Zhou, R. Paolini, J. A. Bagnell, and M. T. Mason, "A convex polynomial force-motion model for planar sliding: Identification and application," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 372–377, IEEE, 2016.

[5] J. K. Li, W. S. Lee, and D. Hsu, "Push-net: Deep planar pushing for objects with unknown physical properties.," in *Robotics: Science and Systems*, vol. 14, pp. 1–9, 2018.

[6] A. Rajeswaran, S. Ghotra, B. Ravindran, and S. Levine, "Epopt: Learning robust neural network policies using model ensembles," *arXiv preprint arXiv:1610.01283*, 2016.

[7] X. B. Peng, M. Andrychowicz, W. Zaremba, and P. Abbeel, "Sim-to-real transfer of robotic control with dynamics randomization," in *2018 IEEE international conference on robotics and automation (ICRA)*, pp. 1–8, IEEE, 2018.

[8] Y. Chebotar, A. Handa, V. Makoviychuk, M. Macklin, J. Issac, N. Ratliff, and D. Fox, "Closing the sim-to-real loop: Adapting simulation randomization with real world experience," in *2019 International Conference on Robotics and Automation (ICRA)*, pp. 8973–8979, IEEE, 2019.

[9] K. Rakelly, A. Zhou, C. Finn, S. Levine, and D. Quillen, "Efficient off-policy meta-reinforcement learning via probabilistic context variables," in *International conference on machine learning*, pp. 5331–5340, 2019.

[10] W. Yu, J. Tan, C. K. Liu, and G. Turk, "Preparing for the unknown: Learning a universal policy with online system identification," *arXiv preprint arXiv:1702.02453*, 2017.

[11] J. Harrison, A. Garg, B. Ivanovic, Y. Zhu, S. Savarese, L. Fei-Fei, and M. Pavone, "Adapt: zero-shot adaptive policy transfer for stochastic dynamical systems," in *Robotics Research*, pp. 437–453, Springer, 2020.

[12] Z. Xu, C. Tang, and M. Tomizuka, "Zero-shot deep reinforcement learning driving policy transfer for autonomous vehicles based on robust control," in *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*, pp. 2865–2871, IEEE, 2018.

[13] C. Tang, Z. Xu, and M. Tomizuka, "Disturbance-observer-based tracking controller for neural network driving policy transfer," *IEEE Transactions on Intelligent Transportation Systems*, 2019.

[14] J. Stüber, C. Zito, and R. Stolkin, "Let's push things forward: A survey on robot pushing," *Frontiers in Robotics and AI*, vol. 7, p. 8, 2020.

[15] M. T. Mason, "Mechanics and planning of manipulator pushing operations," *The International Journal of Robotics Research*, vol. 5, no. 3, pp. 53–71, 1986.

[16] K.-T. Yu, M. Bauza, N. Fazeli, and A. Rodriguez, "More than a million ways to be pushed. a high-fidelity experimental dataset of planar pushing," in *2016 IEEE/RSJ international conference on intelligent robots and systems (IROS)*, pp. 30–37, IEEE, 2016.

[17] M. Bauza and A. Rodriguez, "A probabilistic data-driven model for planar pushing," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 3008–3015, IEEE, 2017.

[18] A. Ajay, J. Wu, N. Fazeli, M. Bauza, L. P. Kaelbling, J. B. Tenenbaum, and A. Rodriguez, "Augmenting physical simulators with stochastic neural networks: Case study of planar pushing and bouncing," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 3066–3073, IEEE, 2018.

[19] Z. Xu, J. Wu, A. Zeng, J. B. Tenenbaum, and S. Song, "Densephysnet: Learning dense physical object representations via multi-step dynamic interactions," *arXiv preprint arXiv:1906.03853*, 2019.

[20] J. Stüber, M. Kopicki, and C. Zito, "Feature-based transfer learning for robotic push manipulation," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1–5, IEEE, 2018.

[21] 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)*, pp. 173–180, IEEE, 2017.

[22] B. Ridge, A. Leonardis, A. Ude, M. Deniša, and D. Skočaj, "Self-supervised online learning of basic object push affordances," *International Journal of Advanced Robotic Systems*, vol. 12, no. 3, p. 24, 2015.

[23] 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.

[24] M. Kopicki, S. Zurek, R. Stolkin, T. Moerwald, and J. L. Wyatt, "Learning modular and transferable forward models of the motions of push manipulated objects," *Autonomous Robots*, vol. 41, no. 5, pp. 1061–1082, 2017.

[25] C. Bodnar, A. Li, K. Hausman, P. Pastor, and M. Kalakrishnan, "Quantile qt-opt for risk-aware vision-based robotic grasping," *arXiv preprint arXiv:1910.02787*, 2019.

[26] D. Ho, K. Rao, Z. Xu, E. Jang, M. Khansari, and Y. Bai, "Retinagan: An object-aware approach to sim-to-real transfer," 2020.

[27] E. Coumans and Y. Bai, "Pybullet, a python module for physics simulation for games, robotics and machine learning," 2016.

[28] L. van der Maaten and G. Hinton, "Visualizing data using t-sne," *Journal of Machine Learning Research*, vol. 9, no. 86, pp. 2579–2605, 2008.