EXI-Net: EXplicitly/Implicitly Conditioned Network for Multiple Environment Sim-to-Real Transfer

Takayuki Murooka
OMRON SINIC X Corporation
The University of Tokyo
t-murooka@jsk.imi.i.u-tokyo.ac.jp

Masashi Hamaya
OMRON SINIC X Corporation
hamaya@sinicx.com

Felix von Drigalski
OMRON SINIC X Corporation
f.drigalski@sinicx.com

Kazutoshi Tanaka
OMRON SINIC X Corporation
kazutoshi.tanaka@sinicx.com

Yoshihisa Ijiri
OMRON SINIC X Corporation
OMRON Corporation
yoshihisa.ijiri@omron.com

Abstract: Sim-to-real transfer is attractive for robot learning, as it avoids the high cost of collecting data with real robots, but transferring agents from simulation to the real world is challenging. Previous studies have presented promising methods to solve this problem, but they may fail when a wider range of dynamics has to be considered. In this study, we propose a network architecture with explicit and implicit dynamics parameters for sim-to-real transfer from multiple environments. Using this method, we can estimate the dynamics of the real world and optimize the action in various kinds of environments. The core novelty lies in the dynamics estimation and action optimization, as well as the use of explicit (physically quantifiable) and implicit (latent) dynamics parameters to condition the network input. We apply our method to the object pushing task and verify its effectiveness by comparing it with previous methods and real-world experiments.

Keywords: Sim-to-Real Transfer, Multiple Environments, Robotic Manipulation

1 Introduction

While learning-based approaches have shown promising performance on complex robotic tasks, collecting the large amounts of training data they require with real robots is often prohibitive. As training agents in simulation would significantly reduce the cost of data collection, the "sim-to-real" problem has become a focus of research.

A promising approach to bridge the gap between simulation and real world is domain randomization, which randomizes parameters such as mass, friction, or appearance during training [1, 2, 3, 4, 5], typically using recurrent neural networks to memorize and manipulate the differences in dynamics. However, the previous works have generally been tested in few different environments, and their scalability to multiple environments has been hardly investigated. Nachum et al. [6] indicated that dealing with a wider range of randomization would make the optimal policy too conservative. This results in poor performance, especially in multi-environment sim-to-real transfer.

In this study, we propose **EXplicitly and Implicitly conditioned Network (EXI-Net)** that can capture different dynamics in the network's input parameters so that multiple environments can be represented. Our key idea is adding two groups of dynamics parameters obtained explicitly and calculated implicitly to the network input (see Fig. 1). We call these parameters **explicit dynamics parameters** (ones which are easily quantified physically, e.g. mass, friction or center of gravity (CoG)) and **implicit dynamics parameters** (ones which are hard to quantify physically, e.g. object shape, uneven ground, or other unmodeled effects). During training, we randomly sample these parameters and directly add the explicit parameters to the network input while we learn the implicit parameters in a latent space using the parametric biases [7] whose effectiveness was shown in other robotic tasks [8, 9, 10]. For testing, we optimize the actions and estimate these dynamics parameters

4th Conference on Robot Learning (CoRL 2020), Cambridge MA, USA.

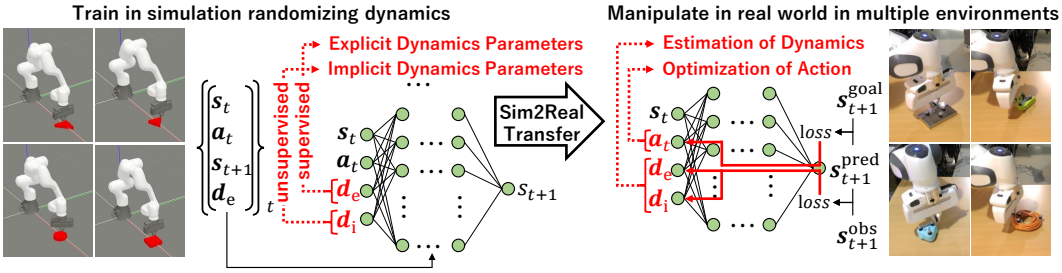


Figure 1: Overview of proposed method for multi-environment sim-to-real transfer. $\boldsymbol{d}_e$ and $\boldsymbol{d}_i$ are explicit and implicit dynamics parameters, which are learned in a supervised and unsupervised fashion respectively.

via online iterative forward and back propagation using the learned predictive model, which predicts the next state from the current state and action. While most previous studies encoded the differences of dynamics implicitly in the weights of recurrent neural networks, we propose conditioning the network on these explicit and implicit dynamics parameters. This allows the model to express a wider range of information of different dynamics parameters with a single network and without degrading the scalability to manipulate in multiple environments.

Our main contributions for multi-environment sim-to-real transfer are summarized as follows: 1) We propose EXI-Net: a deep neural network with two categories of dynamics parameters called explicit and implicit dynamics parameters, and a formalized calculation process for both parameters. 2) We exploit the learned predictive model with explicit and implicit dynamics parameters to perform online action optimization and dynamics estimation. The expressiveness of explicit and implicit dynamics parameters and their capability to memorize and estimate a wide range of dynamics significantly improves the method's performance in multi-environment sim-to-real transfer. 3) We apply the proposed method to object pushing tasks in various dynamics conditions, and confirm that our method outperforms other straightforward approaches with recurrent structures. Our method can also be applied to various situations such as locomotion or assembly tasks, where a slight difference in dynamics parameters can affect the behavior significantly.

2 Related Works

Sim-to-Real Transfer. Focusing on the gap in the system between simulation and real world, there are many studies on system identification to close its gap. Ajay et al. used data augmentation to identify the system [11]. TuneNet [12] dealt with the system identification by making use of one-shot residual tuning.

In contrast, there are various kinds of studies which focus on learning-based manipulation in the real world making use of sim-to-real techniques. Domain adaptation [13], which achieves the adaptation of the network among several domains, has been applied to a range of robotic manipulation tasks by leveraging generative adversarial networks or data augmentation [14, 15, 16, 17, 18]. This method is effective at adapting from simulation to the real world, but it requires real-world manipulation data for training.

Domain randomization was also developed for robust manipulation without data in the real world by using training data from randomized environmental simulation. Tobin et al. [1] proposed robust multiple objects grasping with domain randomization by randomizing rendering. Dynamics randomization, which uses environments with randomized dynamics, was also developed [2, 3, 4, 5]. Recent methods also proposed how to deal with the uncertainty of parameters to be randomized [19] or sample them efficiently [20]. As the wider range of randomization would make the policies more conservative, Nachum et al. [6] presented a hierarchical sim-to-real transfer method, which performed two step randomization on low-level and high-level policies. However, while these methods have used one or several domains during training, they have not investigated in-depth how they perform in multiple domains In this study, we aim at multi-environment sim-to-real transfer and propose network conditioning to handle the variety of dynamics by setting explicit and implicit

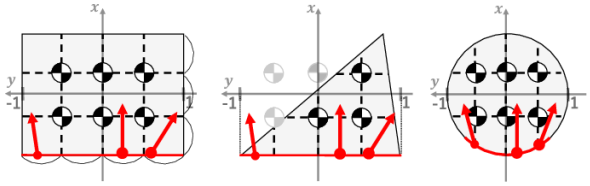
dynamics parameters directly as the input of the single network. Moreover, unlike the previous methods using recurrent neural networks for domain randomization to encode the differences of dynamics implicitly, we propose explicit and implicit dynamics parameters conditioning for the better expressiveness in multiple environments.

Pushing Manipulation. In this study, we tackle an object pushing task since it is a fundamental manipulation task. This topic has been studied widely [21], and recently data-driven approaches as well as a large amount of datasets have been studied and published [22, 23]. In data-driven approach, some previous studies can deal with only one dynamics condition [4, 11, 24, 25]. To deal with the variety of the object shape, other studies utilized visual information [26, 27, 28, 29] by changing the object shape during training, typically with recurrent neural networks. However, previous studies has not investigated a wide variety of dynamics, but focused mainly on different object shapes. We aim to realize the object pushing task in a wide range of various dynamics such as mass, friction coefficient, CoG position, and object shape by leveraging multi-environment sim-to-real without vision.

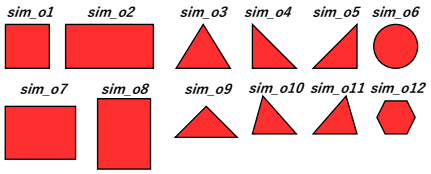
3 Proposed Method

In this section, we introduce our proposed method as applied to the object pushing task which we use to validate our method. This task is suitable for our problem setting as it includes a wide range of explicit and implicit dynamics parameters. First, we present a system expression for the pushing task, sim-to-real transfer, and how to optimize the action and estimate the dynamics.

3.1 State and Action for Object Pushing Task



(a) Object pushing action. Each red point shows a pushing position, and each arrow a direction. The object's edge is parameterized from -1 to 1. CoG positions, described as a black and white circle, that lie outside of the object are ignored.



(b) Object shapes used in simulation. The upper row is for training and the lower row for testing.

Figure 2: Detail of pushing action and objects.

We define the state and action in a similar manner as previous studies [25]. The state is represented as an object pose in a two-dimensional plane, and the action as a pushing position and a direction. The pushing position is a number between -1 to 1, which describes a position on the parameterized object's edge, and the pushing direction is the angle from the edge normal, which lies from $-\frac{\pi}{4}$ to $\frac{\pi}{4}$. A red arrow represents examples of the action in Fig. 2 (a).

3.2 Network Structure Conditioned by Explicit/Implicit Dynamics Parameters

In this study, we propose a learning-based method conditioned by dynamics parameters which is based on a deep predictive model [26, 30, 31, 32]. We designed a network to represent the predictive model of manipulation, which means the network input is the current state and action, and the output is the next state. Let $\boldsymbol{s}_t$ and $\boldsymbol{a}_t$ be the state and action at time step t, then the network of the predictive model $\boldsymbol{f}$ is formulated as:

$$\boldsymbol{s}_{t+1} = \boldsymbol{f}(\boldsymbol{s}_t, \boldsymbol{a}_t). \tag{1}$$

We extend this predictive model by two types of dynamics parameters which condition the network input. *Explicit dynamics parameters* represent physical quantities such as object mass, the object's CoG or friction coefficients. *Implicit dynamics parameters* are other physical parameters that affect the dynamics but are difficult to quantify or model, such as the object shape. During training,

explicit dynamics parameters are given explicitly ("embedded supervised"), while implicit dynamics parameters are learned from the system dynamics ("embedded unsupervised") using the idea of parametric biases [7], which is explained in Sec. 3.4. Let $\boldsymbol{d}_{\mathrm{e}}$ and $\boldsymbol{d}_{\mathrm{i}}$ be explicit and implicit dynamics parameters, then the network is formulated as follows:

$$\boldsymbol{s}_{t+1} = \boldsymbol{f}(\boldsymbol{s}_t, \boldsymbol{a}_t, \boldsymbol{d}_{\mathrm{e}}, \boldsymbol{d}_{\mathrm{i}}). \tag{2}$$

3.3 Sim-to-Real Transfer

We train the predictive model conditioned by explicit and implicit dynamics parameters in various dynamics conditions. We first collect the dataset in simulation by changing explicit dynamics parameters within the range from minimum to maximum with the step size at regular intervals described in Table 1. The CoG is represented in the coordinate system shown in Fig. 2 (a). Apart from the explicit dynamics parameters, we also change the object shape. We prepare six objects from *sim_o1* to *sim_o6* for training in the simulation shown in Fig. 2 (b).

Table 1: The range of the four explicit dynamics parameters in simulation

	Mass [kg]	Friction coefficient	$\mathrm{CoG_x}$	$\mathrm{CoG_y}$
Minimum	0.1	0.1	-0.5	-0.3
Maximum	1.0	1.0	0.5	0.3
Step Size	0.3	0.3	0.5	0.6

It is not important to obtain the "true" values of the dynamics parameters while testing (= executing) in the real world. In fact, we expect the dynamics parameters of the predictive model trained in simulation to differ from the real world. What is important is that the training data contains a range of experiences to which the data obtained in the real world can be correlated. By estimating the dynamics parameters during execution in the real world, our system can close the gap quickly. This is also discussed in Sec. 5 with the experimental data.

3.4 Action Optimization and Dynamics Estimation

Some previous studies also used deep predictive models [32] to optimize the actions continuously using iterative forward propagation and back propagation. In this study, we extend this calculation method to not only action optimization but also dynamics estimation. The action optimization and dynamics estimation could be calculated together in the same gradient-based way. Meanwhile, we calculate them separately to make the optimization convergence easier by the lower dimensionality of the variables. To perform action optimization, we want to calculate the action which causes the next state to be closest to the goal state. We can formulate this optimization problem as follows:

$$\boldsymbol{a}_t^{\mathrm{opt}} = \underset{\boldsymbol{a}_t}{\mathrm{argmin}}\ J(\boldsymbol{s}_{t+1}^{\mathrm{goal}}, \boldsymbol{s}_{t+1}^{\mathrm{pred}}), \tag{3a}$$

$$J(\boldsymbol{s}_{t+1}^{\mathrm{goal}}, \boldsymbol{s}_{t+1}^{\mathrm{pred}}) = \frac{1}{2}\|\boldsymbol{s}_{t+1}^{\mathrm{goal}} - \boldsymbol{s}_{t+1}^{\mathrm{pred}}\|^2, \tag{3b}$$

where J is the cost function to obtain the optimal action $\boldsymbol{a}^{\mathrm{opt}}$, and $\boldsymbol{s}^{\mathrm{pred}}$ and $\boldsymbol{s}^{\mathrm{goal}}$ are the predicted and goal states, respectively. This is a nonlinear optimization problem, which requires a prohibitive amount of time to solve exactly. Therefore, we solve this problem approximately, by using sequential optimization with an update rate $\epsilon_{\boldsymbol{a}}$ as follows:

$$\boldsymbol{g}_{\boldsymbol{a}} = \partial J(\boldsymbol{s}_{t+1}^{\mathrm{goal}}, \boldsymbol{s}_{t+1}^{\mathrm{pred}})/\partial \boldsymbol{a}_t, \tag{4a}$$

$$\boldsymbol{a}_t \leftarrow \boldsymbol{a}_t - \epsilon_{\boldsymbol{a}}\, \boldsymbol{g}_{\boldsymbol{a}}/\|\boldsymbol{g}_{\boldsymbol{a}}\|_2, \tag{4b}$$

where $\|*\|_2$ represents the L2 norm. $\partial J(\boldsymbol{s}_{t+1}^{\mathrm{goal}}, \boldsymbol{s}_{t+1}^{\mathrm{pred}})/\partial \boldsymbol{a}_t$ can be calculated by back propagation of the network with the loss $J(\boldsymbol{s}_{t+1}^{\mathrm{goal}}, \boldsymbol{s}_{t+1}^{\mathrm{pred}})$. The calculation process of forward propagation to predict $\boldsymbol{s}_{t+1}^{\mathrm{pred}}$ and the update of the action with the gradient of the cost function utilizing back propagation in Eq. (4) is repeated several times for convergence of the action. As for dynamics estimation, if $\boldsymbol{d} = (\boldsymbol{d}_{\mathrm{e}}^T\ \boldsymbol{d}_{\mathrm{i}}^T)^T$, we can calculate it in the same way as the action optimization with an update rate $\epsilon_{\boldsymbol{d}}$ as follows:

$$\boldsymbol{g}_{\boldsymbol{d}} = \partial J(\boldsymbol{s}_{\tau+1}^{\mathrm{obs}}, \boldsymbol{s}_{\tau+1}^{\mathrm{pred}})/\partial \boldsymbol{d}, \tag{5a}$$

$$\boldsymbol{d} \leftarrow \boldsymbol{d} - \epsilon_{\boldsymbol{d}}\, \boldsymbol{g}_{\boldsymbol{d}}/\|\boldsymbol{g}_{\boldsymbol{d}}\|_2, \tag{5b}$$

where $\boldsymbol{s}^{\mathrm{obs}}$ is the past information of pushing, which was observed in the current environment and stored. In Algorithm. 1, we describe the algorithm for action optimization and dynamics estimation which is executed every cycle, where L represents a buffer to store the dataset of $(\boldsymbol{s}_t, \boldsymbol{s}_{t-1}, \boldsymbol{a}_{t-1})$ for the current environment. We set both $N^{\boldsymbol{a}}_{\mathrm{update}}$ and $N^{\boldsymbol{d}}_{\mathrm{update}}$ to 100, and finally arrive at the optimized action $\boldsymbol{a}_t$ and the estimated dynamics $\boldsymbol{d}$, then command $\boldsymbol{a}_t$ to the robot.

Algorithm 1 Calculation of Action and Dynamics

$\boldsymbol{s}_t^{\mathrm{obs}} \leftarrow observe$
Store $(\boldsymbol{s}_t^{\mathrm{obs}}, \boldsymbol{s}_{t-1}^{\mathrm{obs}}, \boldsymbol{a}_{t-1})$ in L
for $k = 1, 2, \ldots, N^{\boldsymbol{a}}_{update}$ **do**
  $\boldsymbol{s}_{t+1}^{\mathrm{pred}} \leftarrow f(\boldsymbol{s}_t^{\mathrm{obs}}, \boldsymbol{a}_t, \boldsymbol{d}_{\mathrm{e}}, \boldsymbol{d}_{\mathrm{i}})$
  $\boldsymbol{a}_t \leftarrow \textsc{UpdateAction}(\boldsymbol{s}_{t+1}^{\mathrm{goal}}, \boldsymbol{s}_{t+1}^{\mathrm{pred}}, \boldsymbol{a}_t)$ ▷ Eq. 4
end for
for $k = 1, 2, \ldots, N^{\boldsymbol{d}}_{update}$ **do**
  Sample $(\boldsymbol{s}_{\tau+1}^{\mathrm{obs}}, \boldsymbol{s}_{\tau}^{\mathrm{obs}}, \boldsymbol{a}_{\tau})$ from L
  $\boldsymbol{s}_{\tau+1}^{\mathrm{pred}} \leftarrow f(\boldsymbol{s}_{\tau}^{\mathrm{obs}}, \boldsymbol{a}_{\tau}, \boldsymbol{d}_{\mathrm{e}}, \boldsymbol{d}_{\mathrm{i}})$
  $\boldsymbol{d} \leftarrow \textsc{UpdateDynamics}(\boldsymbol{s}_{\tau+1}^{\mathrm{obs}}, \boldsymbol{s}_{\tau+1}^{\mathrm{pred}}, \boldsymbol{d})$ ▷ Eq. 5
end for

4 Experiments

The general formulation of the predictive model is shown in Eq. (2). In our case, we can simplify the equation since we can always move our reference frame to the center of the object. Consequently, the state does not affect the prediction of the model and can be omitted from the equation, so that the predictive model is expressed as follows:

$$\Delta \boldsymbol{s}_{t+1} = \boldsymbol{f}(\boldsymbol{a}_t, \boldsymbol{d}_{\mathrm{e}}, \boldsymbol{d}_{\mathrm{i}}), \tag{6a}$$

$$\Delta \boldsymbol{s}_{t+1} \triangleq \boldsymbol{s}_{t+1} - \boldsymbol{s}_t. \tag{6b}$$

To verify the effectiveness of explicit and implicit dynamics parameters, we conducted comparative experiments with six types of methods: 1) a direct model that outputs the action directly from the difference between the next and current state as in Eq. (7), 2) EXI-net without explicit and implicit dynamics parameters, 3) EXI-net without explicit dynamics parameters, 4) EXI-net without explicit and implicit dynamics parameters and with memory using the recurrent structure, 5) EXI-net with memory using the recurrent structure and 6) EXI-net. We refer to these methods as *Direct*, *EXI w/o* $\boldsymbol{d}_{\mathrm{e}}$, $\boldsymbol{d}_{\mathrm{i}}$, *EXI w/o* $\boldsymbol{d}_{\mathrm{e}}$, *EXI w/o* $\boldsymbol{d}_{\mathrm{e}}$, $\boldsymbol{d}_{\mathrm{i}}$ *w/ mem.*, *EXI w/ mem.*, *EXI* respectively.

$$\boldsymbol{a}_t = \boldsymbol{f}(\Delta \boldsymbol{s}_t, \boldsymbol{d}_{\mathrm{e}}, \boldsymbol{d}_{\mathrm{i}}) \tag{7}$$

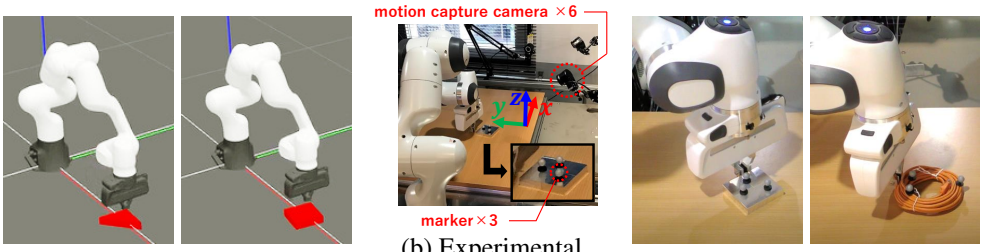
The network representing the predictive model consisted of three hidden layers of 100 fully connected units with ReLU activations. The explicit dynamics parameters had four dimensions (mass, friction coefficient, and the CoG position). The implicit dynamics parameters had five dimensions for *EXI* and nine for *EXI w/o* $\boldsymbol{d}_{\mathrm{e}}$ (to replace the four missing explicit parameters). The recurrent structure was implemented with 100 LSTM [33] units. We implemented these networks using the deep learning library PyTorch [34]. The loss function was the mean squared error, and we trained each network for 100 epochs with batch size 128 using the ADAM optimizer [35].

4.1 Data Preparation and Training

We used Gazebo [36] as the simulator, and a Franka Emika Panda robot for both simulation and real world experiments. To generate a dataset, we discretized the possible push positions and directions and performed pushing. The pushing position on the object edge was parameterized from -1 to 1, so the discretized push positions ranged from -1 to 1 with a step of 0.2, and the pushing direction from $-\frac{\pi}{4}$ to $\frac{\pi}{4}$ with a step of $\frac{\pi}{8}$. Using Gazebo, we generated datasets in the various dynamics conditions described in Sec. 3.3. The total number of explicit dynamics parameters including mass, friction coefficient and the CoG position for each six objects for training was 120, 120, 80, 60, 60, and 120, respectively, considering the restriction of the CoG position shown in Fig. 2 (a). To learn the parametric biases representing the implicit dynamics parameters, we trained a different set of biases for each object shape, as we expected the implicit parameters to express the object shape.

4.2 Pushing Objects in Simulation

To verify the trained predictive model, we first tested pushing objects in simulation. We put the objects in front of the robot in Gazebo. The goal state, (a goal object pose (Δx, Δy, $\Delta \theta$)), was



(a) Pushing objects in Gazebo (b) Experimental setup in real world (c) Pushing objects in real world

Figure 3: Experiments of pushing objects

sampled randomly from $U(0.1m, 0.3m)$, $U(-0.2m, 0.2m)$, $U(-60°, 60°)$ respectively. The robot pushed the object by optimizing the pushing action and estimating explicit and implicit dynamics parameters using the predictive model. After pushing the object from the initial object pose to the goal, we calculated the error of the object pose between the goal and current object pose. Each trial was considered successful if it brought the object within 2 cm of the goal position and 5° of the goal orientation in at most 15 steps. We used twelve objects designated as *sim_o1* to *sim_o12*, the first 6 objects of which were used for training, shown in Fig. 2 (b). For each test object, we conducted 100 trials by changing the mass, friction coefficient and CoG position randomly within the maximum and minimum values shown in Table 1. We summarized the average success rate, the number of steps and the error of estimated explicit dynamics parameters when estimated for each object in Table 2. We calculated the mean absolute error between the estimated and true explicit dynamics parameters, and normalized it by the minimum and maximum values. From this result, in most cases we observed that *EXI* and *EXI w/ mem.* performed favorably compared to other methods including *EXI w/o* $\boldsymbol{d}_e$,$\boldsymbol{d}_i$ *w/ mem.* typically used in previous studies.

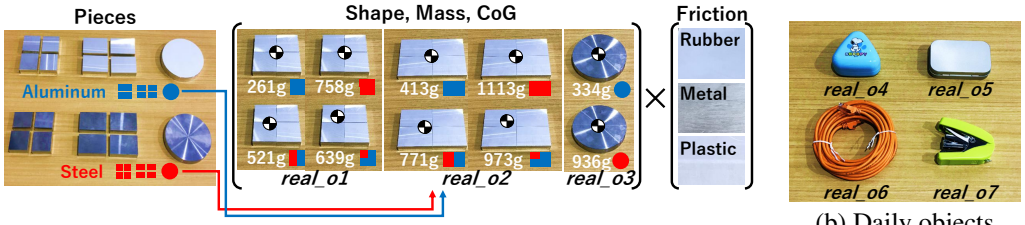
Table 2: Test in simulation (Success rate / Steps / Error of explicit dynamics parameters).

object	*Direct*	*EXI w/o* $\boldsymbol{d}_{\mathrm{e}}$, $\boldsymbol{d}_{\mathrm{i}}$	*EXI w/o* $\boldsymbol{d}_{\mathrm{e}}$	*EXI w/o* $\boldsymbol{d}_{\mathrm{e}}$, $\boldsymbol{d}_{\mathrm{i}}$ *w/ mem.*	*EXI w/ mem.*	*EXI*
sim_o1	0.51 / 13.7 / -	0.41 / 13.0 / -	**1.00** / 9.4 / -	0.83 / 12.2 / -	**1.0** / 6.2 / 0.21	**1.0** / 6.7 / 0.14
sim_o2	0.34 / 13.6 / -	0.54 / 14.3/ -	0.94 / 8.8 / -	0.82 / 12.3 / -	**1.0** / 6.8 / 0.15	**1.0** / 7.3 / 0.21
sim_o3	0.40 / 14.3 / -	0.55 / 13.8 / -	0.86 / 9.3 / -	0.88 / 13.4 / -	**1.0** / 9.1 / 0.13	**1.0** / 7.2 / 0.11
sim_o4	0.41 / 13.8 / -	0.61 / 12.1 / -	**1.00** / 8.9 / -	0.79 / 13.1 / -	**1.0** / 8.8 / 0.17	**1.0** / 8.4 / 0.28
sim_o5	0.54 / 13.9 / -	0.73 / 13.2 / -	0.78 / 10.0 / -	0.93 / 13.3 / -	**1.0** / 6.3 / 0.24	**1.0** / 6.1 / 0.10
sim_o6	0.33 / 12.4 / -	0.52 / 12.6 / -	0.88 / 10.3 / -	0.72 / 12.7 / -	**1.0** / 6.4 / 0.19	**1.0** / 4.5 / 0.23
sim_o7	0.39 / 13.2 / -	0.53 / 13.8 / -	0.91 / 10.2 / -	0.78 / 12.0 / -	**1.0** / 7.8 / 0.14	**1.0** / 8.1 / 0.09
sim_o8	0.47 / 13.5 / -	0.48 / 14.2 / -	**1.00** / 7.6 / -	0.86 / 11.4 / -	**1.0** / 6.9 / 0.16	**1.0** / 7.8 / 0.17
sim_o9	0.39 / 12.6 / -	0.50 / 12.4 / -	0.90 / 8.1 / -	0.75 / 10.3 / -	**1.0** / 8.9 / 0.22	**1.0** / 8.1 / 0.09
sim_o10	0.51 / 14.1 / -	0.51 / 13.5 / -	0.83 / 9.0 / -	0.83 / 13.7 / -	**1.0** / 9.1 / 0.12	**1.0** / 7.5 / 0.15
sim_o11	0.38 / 13.6 / -	0.68 / 13.9 / -	0.79 / 11.9 / -	0.71 / 10.8 / -	**1.0** / 7.2 / 0.08	**1.0** / 8.8 / 0.11
sim_o12	0.42 / 13.4 / -	0.58 / 14.3 / -	0.77 / 13.6 / -	0.65 / 12.9 / -	**1.0** / 8.5/ 0.13	**1.0** / 8.3 / 0.09

4.3 Pushing Objects in Real World

To verify the effectiveness of the proposed method for multi-environment sim-to-real transfer, we conducted two types of experiments with the real robot shown in Fig. 3 (c). We compared the proposed method with *EXI w/ true* $\boldsymbol{d}_{\mathrm{e}}$, *EXI w/o* $\boldsymbol{d}_{\mathrm{e}}$, $\boldsymbol{d}_{\mathrm{i}}$ *w/ mem.* and *EXI w/ mem.*. *EXI w/ true* $\boldsymbol{d}_{\mathrm{e}}$ was performed with explicit and implicit dynamics parameters, but the explicit dynamics parameters were not estimated in real time but measured physically before pushing. We used a motion-capture system to obtain the object pose using six motion-capture cameras and three markers on the object shown in Fig. 3 (b).

By assembling pieces of steel and aluminium (which have different density), we create square (*real_o1*) and rectangular (*real_o2*) objects with differing masses and CoGs as shown in Fig. 4 (a). As we could easily determine the dynamics parameters for these objects as well as the metal discs (*real_o3*), we classify them as "regularly shaped" objects. To change the friction parameters, we attached rubber, plastic or nothing to the objects' underside, so that we obtained twelve different



(a) Regularly shaped objects (b) Daily objects

Figure 4: Detail of objects used in real world. Blue and red figures describe pieces made of aluminum and steel respectively.

sets of dynamics parameters for shaped and for rectangular objects, and six for circular objects. The condition of the goal state and the success condition was the same as in the simulation experiments. For each object and dynamics parameter, we conducted five trials. We summarized the average success rate, the number of steps, and the error of the estimated explicit dynamics parameters when estimated for each object in Table 3.

Second, we prepared four daily life objects: a triangular plastic box, a metal case, a wound-up LAN cable, and a stapler, ($real\_o4$ to $real\_o7$) as shown in Fig. 4 (b). These experiments show that the proposed method can be applied to novel objects. By approximating these objects to a rectangle, triangle, or circle, we could discretize the object edge. In these experiments, we extended the proposed method to all the edges of the object to push toward any goal pose. We sampled the goal object pose (Δx, Δy, $\Delta\theta$) randomly from $U(-0.2m, 0.2m)$, $U(-0.2m, 0.2m)$, $U(-90°, 90°)$ respectively, which was a wider range of possible goals, resulting in a general pushing problem. The success evaluation was the same as the experiments in the simulation. For each object, we conducted ten trials and summarized the results in Table 3. From the results of both experiments, in all cases we observed that *EXI* and *EXI w/ mem.* showed the best performance.

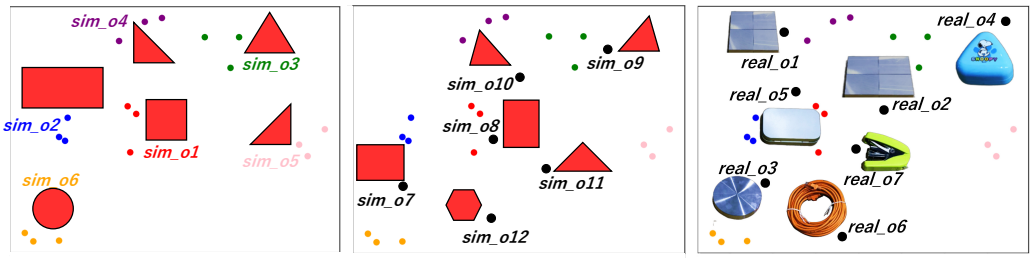
Table 3: Test in real world (Success rate / Steps / Error of explicit dynamics parameters).

object	*EXI w/ true* $\boldsymbol{d}_{\mathrm{e}}$	*EXI w/o* $\boldsymbol{d}_{\mathrm{e}}$, $\boldsymbol{d}_{\mathrm{i}}$ *w/ mem.*	*EXI w/ mem.*	*EXI*
real_o1	0.89 / 11.3 / 0.00	0.72 / 12.8 / -	0.99 / 7.2 / 0.39	**1.0** / 8.3 / 0.52
real_o2	0.95 / 13.2 / 0.00	0.69 / 13.1 / -	**1.0** / 6.8 / 0.41	0.98 / 6.2 / 0.33
real_o3	0.98 / 10.8 / 0.00	0.92 / 13.4 / -	**1.0** / 8.1 / 0.28	**1.0** / 7.9 / 0.38
real_o4	- / - / -	0.5 / 13.8 / -	**1.0** / 7.9 / -	**1.0** / 7.0 / -
real_o5	- / - / -	0.6 / 14.5 / -	**1.0** / 9.3 / -	0.9 / 9.5 / -
real_o6	- / - / -	0.5 / 13.3 / -	**1.0** / 7.2 / -	**1.0** / 7.2 / -
real_o7	- / - / -	0.5 / 13.6 / -	0.9 / 10.9 / -	**1.0** / 6.8 / -

5 Discussion

In this study, we considered a variety of dynamics by adding explicit and implicit dynamics parameters to the network input, although most previous studies [4, 5, 26] considered it using the recurrent structure. By comparing *EXI w/o* $\boldsymbol{d}_{\mathrm{e}}$, $\boldsymbol{d}_{\mathrm{i}}$ *w/ mem.* and *EXI* in Table 2 or Table 3, we can see that adding explicit and implicit dynamics parameters to the network input outperformed recurrent neural networks in this case. In addition, the performance of *EXI w/o mem.* and *EXI* was very similar. In summary, our method performed better than a recurrent network without dynamics parameters, and adding the recurrent structure to our method had little positive impact on performance.

During training, we randomly separated the dataset of each shape into three sub-datasets to acquire three parametric biases for each shape and checked that implicit dynamics parameters could learn the feature of the object shape well. To visualize the representation of implicit dynamics parameters, we decomposed them into two-dimensional space by applying principal component analysis (PCA); the results are shown in Fig. 5 (a). Besides, we show one example of the placement of parametric biases estimated by *EXI* for each object in the experiments in Fig. 5 (b) (c). The parametric biases were grouped closely together when training with the same object shape, and clustered in different regions for different shapes, implying that the biases indeed implicitly represent different object



(a) When training. (b) When testing in simulation. (c) When testing in real world.

Figure 5: Parametric biases after PCA transformation (first two components plotted)

shapes. When we tested an object shape that was not contained in the training dataset, parametric biases also converged to a certain feature. One can see in Fig. 5 (b) (c) that the results for *sim_o7*, *sim_o1* and *sim_o2*, as well as for *sim_o3* and *sim_o9* show particularly good congruence. Similarly, the implicit dynamics parameters of *real_o3* and *sim_o6* are close and the real-world test succeeded in all trials of this object. The overall performance in the real-world experiments imply that implicit dynamics parameters were able to composite object shape information without the visual information commonly used in previous studies [26, 27, 28, 29].

Furthermore, we can see the effect of explicit dynamics parameters, which were not used in previous studies [8, 9, 10], by comparing *EXI w/o* $\boldsymbol{d}_{\mathrm{e}}$ and *EXI* in Table 2. The results show that separating dynamics parameters into explicit and implict dynamics parameters in the network has a large effect on the method's performance. The reason is most likely that learning parametric biases (implicit dynamics parameters) from scratch is more difficult for higher numbers of parameters. The explicit parameters could relax the learning parameter problem by separating the parameters that can be given as apriori knowledge (explicit) and those that need to be learned (implicit).

In simulation, we estimated explicit dynamics parameters mostly correctly as shown in Table 2. By contrast, in the real world, most of the estimated explicit dynamics parameters were far from the true values. This is because of the gap between simulation and real world. However, as shown by the results for *EXI w/ true* $\boldsymbol{d}_{\mathrm{e}}$ and *EXI* in Table 3, the estimated dynamics parameters do not need to match the true values for good performance, as we can still leverage the experiences made in simulation. In fact, when we used the measured values for the explicit dynamics parameters in the real world, the robot performed worse than with the estimated parameters. Indeed, this is the core feature of our method: using parameters that represent knowledge about the world, changing the parameters in a wide range during training, and estimating the parameters online to allow the system to adjust to the real world. In this way, our system achieved multi-environment sim-to-real transfer.

The reason that the dynamics parameters have such a profound effect is that our method learns coherent responses under a variety of dynamics. In a sense, the network learns the behavior of different "versions" of the world, and estimating the dynamics parameters online "shifts" the current world to a version it has experienced before, so its simulated experiences still apply.

We proposed the concept of conditioning the predictive model with explicit and implicit dynamics parameters for multi-environment sim-to-real transfer, and applied it to a supervised-learning method. Since we simply extend the predictive model, this can be applied to other methods that use a predictive model, e.g. model-based reinforcement learning. This concept will enable those learning-based approaches to explore in multiple environments and accelerate the spread of those researches in various robotic tasks, which is part of our future work.

6 Conclusion

In this study, we proposed a method for multi-environment sim-to-real transfer by separating dynamics into explicit and implicit dynamics parameters, and optimizing the action and estimating dynamics parameters. We demonstrated that our method outperforms state-of-the-art methods in an object pushing task. Explicit and implicit dynamics parameters were proven to bridge the gap between simulation and real world more precisely in various dynamics conditions than other methods, such as recurrent neural networks.

References

[1] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 23–30. IEEE/RSJ, 2017.

[2] J. Tan, T. Zhang, E. Coumans, A. Iscen, Y. Bai, D. Hafner, S. Bohez, and V. Vanhoucke. Sim-to-real: Learning agile locomotion for quadruped robots. *arXiv preprint arXiv:1804.10332*, 2018.

[3] J. Matas, S. James, and A. J. Davison. Sim-to-real reinforcement learning for deformable object manipulation. *arXiv preprint arXiv:1806.07851*, 2018.

[4] X. B. Peng, M. Andrychowicz, W. Zaremba, and P. Abbeel. Sim-to-real transfer of robotic control with dynamics randomization. In *IEEE international conference on robotics and automation*, pages 1–8. IEEE, 2018.

[5] 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*, 39(1):3–20, 2020.

[6] O. Nachum, M. Ahn, H. Ponte, S. Gu, and V. Kumar. Multi-agent manipulation via locomotion using hierarchical sim2real. *arXiv preprint arXiv:1908.05224*, 2019.

[7] J. Tani and M. Ito. Self-organization of behavioral primitives as multiple attractor dynamics: A robot experiment. *Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans*, 33(4):481–488, 2003.

[8] T. Ogata, H. Ohba, J. Tani, K. Komatani, and H. G. Okuno. Extracting multi-modal dynamics of objects using rnnpb. In *International Conference on Intelligent Robots and Systems*, pages 966–971. IEEE/RSJ, 2005.

[9] T. Chen, A. Murali, and A. Gupta. Hardware conditioned policies for multi-robot transfer learning. In *Advances in Neural Information Processing Systems*, pages 9333–9344, 2018.

[10] K. Kawaharazuka, K. Tsuzuki, M. Onitsuka, Y. Asano, K. Okada, K. Kawasaki, and M. Inaba. Object recognition, dynamic contact simulation, detection, and control of the flexible musculoskeletal hand using a recurrent neural network with parametric bias. *Robotics and Automation Letters*, 5(3):4580–4587, 2020.

[11] 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 *International Conference on Intelligent Robots and Systems*, pages 3066–3073. IEEE/RSJ, 2018.

[12] A. Allevato, E. S. Short, M. Pryor, and A. Thomaz. Tunenet: One-shot residual tuning for system identification and sim-to-real robot task transfer. In *Conference on Robot Learning*, pages 445–455, 2020.

[13] V. M. Patel, R. Gopalan, R. Li, and R. Chellappa. Visual domain adaptation: A survey of recent advances. *Signal Processing Magazine*, 32(3):53–69, 2015.

[14] A. A. Rusu, M. Večerík, T. Rothörl, N. Heess, R. Pascanu, and R. Hadsell. Sim-to-real robot learning from pixels with progressive nets. In *Conference on Robot Learning*, pages 262–270, 2017.

[15] A. Gupta, C. Devin, Y. Liu, P. Abbeel, and S. Levine. Learning invariant feature spaces to transfer skills with reinforcement learning. *arXiv preprint arXiv:1703.02949*, 2017.

[16] K. Bousmalis, A. Irpan, P. Wohlhart, Y. Bai, M. Kelcey, M. Kalakrishnan, L. Downs, J. Ibarz, P. Pastor, K. Konolige, et al. Using simulation and domain adaptation to improve efficiency of deep robotic grasping. In *International Conference on Robotics and Automation*, pages 4243–4250. IEEE, 2018.

[17] F. Golemo, A. A. Taiga, A. Courville, and P.-Y. Oudeyer. Sim-to-real transfer with neural-augmented robot simulation. In *Conference on Robot Learning*, pages 817–828, 2018.

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

[19] F. Ramos, R. C. Possas, and D. Fox. Bayessim: adaptive domain randomization via probabilistic inference for robotics simulators. *arXiv preprint arXiv:1906.01728*, 2019.

[20] B. Mehta, M. Diaz, F. Golemo, C. J. Pal, and L. Paull. Active domain randomization. In *Conference on Robot Learning*, pages 1162–1176, 2020.

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

[22] 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 *International Conference on Intelligent Robots and Systems*, pages 30–37. IEEE/RSJ, 2016.

[23] M. Bauza, F. Alet, Y.-C. Lin, T. Lozano-Pérez, L. P. Kaelbling, P. Isola, and A. Rodriguez. Omnipush: accurate, diverse, real-world dataset of pushing dynamics with rgb-d video. In *International Conference on Intelligent Robots and Systems*, pages 4265–4272. IEEE/RSJ, 2019.

[24] M. Bauza and A. Rodriguez. A probabilistic data-driven model for planar pushing. In *International Conference on Robotics and Automation*, pages 3008–3015. IEEE, 2017.

[25] M. Bauza, F. R. Hogan, and A. Rodriguez. A data-efficient approach to precise and controlled pushing. In *Conference on Robot Learning*, pages 336–345, 2018.

[26] 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*, volume 14, pages 1–9, 2018.

[27] A. Kloss, M. Bauza, J. Wu, J. B. Tenenbaum, A. Rodriguez, and J. Bohg. Accurate vision-based manipulation through contact reasoning. *arXiv preprint arXiv:1911.03112*, 2019.

[28] L. Yen-Chen, M. Bauza, and P. Isola. Experience-embedded visual foresight. In *Conference on Robot Learning*, pages 1015–1024, 2020.

[29] M. Björkman, Y. Bekiroglu, V. Högman, and D. Kragic. Enhancing visual perception of shape through tactile glances. In *International Conference on Intelligent Robots and Systems*, pages 3180–3186. IEEE/RSJ, 2013.

[30] I. Lenz, R. A. Knepper, and A. Saxena. Deepmpc: Learning deep latent features for model predictive control. In *Robotics: Science and Systems*. Rome, Italy, 2015.

[31] W. Yan, A. Vangipuram, P. Abbeel, and L. Pinto. Learning predictive representations for deformable objects using contrastive estimation. *arXiv preprint arXiv:2003.05436*, 2020.

[32] D. Tanaka, S. Arnold, and K. Yamazaki. Emd net: An encode–manipulate–decode network for cloth manipulation. *Robotics and Automation Letters*, 3(3):1771–1778, 2018.

[33] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.

[34] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. In *Advances in neural information processing systems*, pages 8026–8037, 2019.

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

[36] N. Koenig and A. Howard. Design and use paradigms for gazebo, an open-source multi-robot simulator. In *International Conference on Intelligent Robots and Systems*, pages 2149–2154. IEEE/RSJ, 2004.