2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)
October 25-29, 2020, Las Vegas, NV, USA (Virtual)

Adaptability Preserving Domain Decomposition for Stabilizing Sim2Real Reinforcement Learning

Haichuan Gao[1*], Zhile Yang[1*], Xin Su[1], Tian Tan[2] and Feng Chen[1]

***Abstract*— In sim-to-real transfer of Reinforcement Learning (RL) policies for robot tasks, Domain Randomization (DR) is a widely used technique for improving adaptability. However, in DR there is a conflict between adaptability and training stability, and heavy DR tends to result in instability or even failure in training. To relieve this conflict, we propose a new algorithm named *Domain Decomposition* (DD) that decomposes the randomized domain according to environments and trains a separate RL policy for each part. This decomposition stabilizes the training of each RL policy, and as we prove theoretically, the adaptability of the overall policy can be preserved. Our simulation results verify that DD really improves stability in training while preserving ideal adaptability. Further, we complete a complex real-world vision-based patrolling task using DD, which demonstrates DD's practicality. A video is attached as supplementary material.**

I. INTRODUCTION

Sim-to-real (sim2real) Reinforcement Learning (RL) is a promising approach for real-world robot-control tasks due to its flexibility in task description and ease in data sampling [1]–[6]. In sim2real RL, reality gap (i.e. the impossibility of constructing simulation environments the same as real-world ones), is a critical problem that hinders the adaptability of RL policies. For this problem, Domain Randomization (DR), which randomizes the domain of simulation environments to cover probable real-world ones and trains agents in samples from the randomized domain (Fig. 1a, 1b), is widely adopted [7]–[9] to strengthen the adaptability of RL policy.

Real-world tasks generally contain considerable reality gaps, e.g. textures, lighting and friction coefficients. Therefore, a heavy DR (i.e. DR that complexly randomizes the domain and samples a large number of environments), is commonly used to provide sufficient adaptability [10]–[12]. However, in DR there is a conflict between adaptability and training stability. This results in that, a heavy DR requires a large scale of paralleled data-sampling to stabilize training. Problem is that, when the scale is limited by the amount of computation resources, training of standard RL algorithms tends to be unstable and fail [13].

This problem becomes amplified when dealing with vision-based tasks with uncertain dynamics, where reality gaps in both visual styles and dynamics need to be considered. The most advanced researches that handle this type of tasks include [12] and [14]. More specifically, [12] compresses visual observations into lower-dimensional sensor-level states to make training easier (Fig. 1c). Although this method is effective for reality gaps in both visual styles and dynamics, the design of sensor-level states is task-specific, thus, it does not generalize to other tasks. [14] proposes Random2canonical (R2C) that relieves instability through mapping different visual styles of observations to one single style before RL training. The limitation is that different dynamics cannot be mapped together to further improve stability.

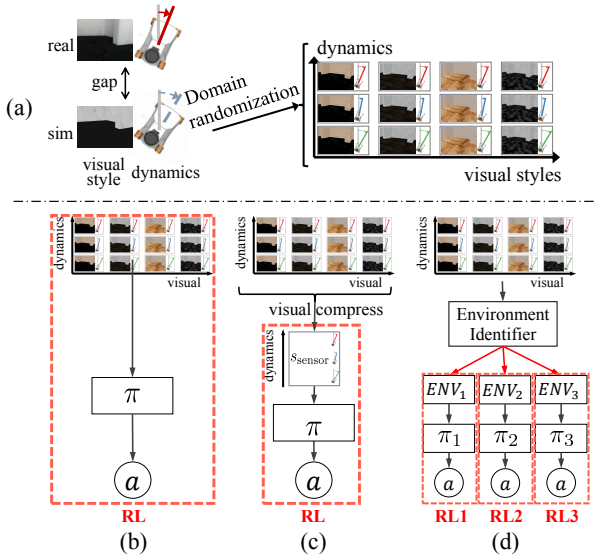


Fig. 1. (a) Reality gap and DR; (b, c) representative methods train one RL policy in the original randomized domain; (d) DD decomposes the randomized domain and trains a separate RL policy for each part.

We argue that the instability in training of RL policies with DR is due to the complexity in the domain of environments: a stable training of RL policy generally requires unbiased evaluations, which further requires data samples to be sampled independently and identically across all the environments [15], [16], but the complexity of the domain makes this infeasible to achieve practically. Therefore, we propose *Domain Decomposition* (DD) that performs decomposition to the randomized domain and handles each part separately. In this way data samples from each part come from a simpler distribution, and independent identical sampling becomes feasible, i.e. does not require an unaffordably large batch size, making the training of RL policies stable. Here we

† This work was supported in part by the National Natural Science Foundation of China under Grant 61671266 and Grant 61836004, in part by the Tsinghua-Guoqiang research program under Grant 2019GQG0006, and in part by Qualcomm Technologies, Inc.

*Equal contribution

[1]Haichuan Gao, Zhile Yang, Xin Su and Feng Chen are with Tsinghua University, Beijing, China. {ghc18,yzl18,suxin16}@tsinghua.edu.cn, chenfeng@tsinghua.edu.cn

[2]Tian Tan is with Stanford University, Stanford, USA. tiantan@stanford.edu

need to solve two key problems: 1) how to decompose the randomized domain to ensure stability in training, and 2) whether the adaptability of RL policy that can be brought by DR is preserved in DD so that the conflict is indeed relieved.

To address the first problem, we notice that the training of RL policies in a single environment is generally stable and that different simulation environments can be uniquely identified by their parameters in the simulator. Therefore, we propose to decompose the randomized domain according to environments. Concretely, we train an environment identifier to classify environments with their parameters, and then train a set of RL policies respectively for each class of environments (Fig. 1d). Note that this design is not task-specific, because the parameters of simulation environments are generally available in simulators.

For the second problem, because the environment identifier and the corresponding RL policies are trained separately, the overall adaptability depends on the generalization of these two parts. To analyze the overall adaptability, we perform a formal transformation and display the dependence concretely. Based on this, we theoretically prove that the overall adaptability can be preserved if the generalization of the environment identifier satisfies a certain condition. Note that in the setting of sim2real, the environment classifier can be trained in a supervised way rather than RL and its data samples can be adequately acquired from the simulator, so its generalization is generally feasible to control to satisfy the condition and to preserve the overall adaptability.

To verify the effectiveness of our method, we design a complex robot patrolling task that is vision-based with uncertain dynamics, as shown in Fig. 3. In this task, a robot car is tasked to move around according to first-person view and cover the whole area without collision. Visual styles such as lighting and wall textures are variable, and dynamics such as the robot's movement speed and friction between the robot and the floor are uncertain. These multiple kinds of variations necessitate a heavy DR. Note that our method of DD can handle randomizations on both visual styles and dynamics, but due to limited computation resources, we consider only the dynamics part in this paper and adopt R2C to randomizations on visual styles. Nevertheless, our comparison with pure R2C in simulation experiments shows that it is DD that significantly improves stability in training while preserving statisfying adaptability. In the real-world test, our method successfully transfers the policy trained in simulation to the real-world environment and completes the task. These results verify the overall effectiveness and practicality of our method.

We summarize our main contributions as follows:

- We offer a formal analysis of the instability problem in sim2real RL with DR;
- We propose DD, a novel method that relieves the conflict between adaptability and training stability in RL with DR, and demonstrate its effectiveness both theoretically and empirically;
- We apply our method to a real-world environment and complete a complex vision-based robot patrolling task.

II. RELATED WORK

In this section, we introduce current advances that are related to our problem, method, and design of the task.

Reality gap, i.e. the inevitable differences between simulation environments and reality, is the main barrier of sim2real transfer and has drawn the interest of many researches. Current works can be roughly divided into two categories: Domain Adaptation (DA) and DR. In DA, policies are tuned in the real-world environment after training in simulation, and current researches are mainly seeking to improve the efficiency of tuning [1]–[4]. We focus on DR, in which no data samples are required from real-world environments [7]–[9].

Generally, DR is categorized into visual randomization and dynamics randomization. Visual randomization [8], [10], [11] randomizes factors that cause significant visual changes, such as textures of objects and lighting conditions. It has been applied to help predict the success rate of robot arm grabbing and generate trajectories of manipulation [10], [17]–[19], realize an active perception of objects' poses according to real-world interaction [20], and predict 3D point cloud of objects according to 2D inputs to make grabbing policies robust [21].

Dynamics randomization [9], [13], [22] randomizes factors that determine physical dynamics of simulation environments, e.g. friction coefficient, damping coefficient and mass of objects. Through dynamics randomization, many tasks with uncertain dynamics have been completed, including "pushing" [9], particle, reacher, fetch pick [23], [24] with manipulators, control of a quadruped speed robot [21], and flying octave trajectories with a UAV [25].

Although DR has shown outstanding sim2real performance in the tasks mentioned above, it is still challenging to train in situations requiring heavy randomization, such as situations where both visual randomization and dynamics randomization are combined, due to its poor stability in RL training [12], [23]. To alleviate this problem, [14] learns a mapping from different styles of visual observations to one single style before training RL policy, and [26] constrain the policies for different visual domains to be consistent by adding a consistency constraint loss to RL's policy update. Nevertheless, these methods do not apply to dynamics randomization because different dynamics often correspond to different policies and cannot be mapped together. [20] constructs an extra RL process to learn a sampling policy instead of uniform sampling to improve sample efficiency in the reacher task; [23] combines this method with an asymmetric actor-critic framework to further stabilize training and to complete several manipulation tasks such as particle and fetch pick; [12] compresses visual inputs to low-dimensional sensor information to make RL training easier. These methods stabilize training by adjusting RL algorithms but suffer from the reliance on task-specific designs, e.g. selection of sensor information and design of sampling policy.

III. ANALYSIS OF RL WITH DR

In this paper, our goal is to relieve the conflict in DR between adaptability and training stability, i.e. to alleviate the

instability problem while preserving the overall adaptability. In this section, we first introduce some preliminaries. Then we analyze the causes of instability in RL with DR.

A. Preliminaries

For most vision-based robot tasks, the robot can only acquire a first-person view of the environment, i.e. the environment is partially observable. In RL, such environments are often formulated as Partially-Observable Markov Decision Process (POMDP), which is defined by the tuple $\langle \mathcal{S}, \Omega, \mathcal{A}, O, P, R, \gamma \rangle$. At each time-step t, the overall state of the environment is $s_t \in \mathcal{S}$, and the agent's observation is $\omega_t \in \Omega$, which is determined by the observation function $O : \mathcal{S} \to \Omega$. The agent chooses an action $a_t \in \mathcal{A}$ according to its policy $\pi : \Omega \times \mathcal{A} \to [0, 1]$ (in probability form written as $\pi\left(a_{t+1} | \omega_t\right)$). Then the environment transits to a new state s_{t+1} according to the transition function $P : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$ (in probability form written as $P\left(s_{t+1} | s_t, a_t\right)$), and returns the agent a reward signal r_t according to the reward function $R : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$. γ is the temporal discount factor.

In POMDP, the observation ω_t may only reflect a part of the state and does not possess Markov property, which is an important assumption in RL. Thus it is common to replace ω with the combination of all perceived observations and performed actions $h_t = \{\omega_0, a_0, \omega_1, a_1, \ldots, \omega_t\}$ to regain Markov property [27]. The space of histories is denoted as $\mathcal{H}$. For simplicity we continue using P, R and O to denote corresponding functions, with a slight abuse of notation.

Based on the above notations, the optimization target of RL is

$$\min_{\pi \in \Pi} J\left(\pi\right) = \min_{\pi \in \Pi} - \int_{h_0 \in \mathcal{H}} \rho\left(h_0\right) V_\pi\left(h_0\right) \mathrm{d}h_0 \ , \tag{1}$$

where ρ is the distribution of initial history h_0, $V_\pi\left(h_t\right)$ is the value function that satisfies the following Bellman equation:

$$\begin{aligned} V_\pi\left(h_t\right) = \int_{a_t \in \mathcal{A}} \pi\left(a_t | h_t\right) \int_{h_{t+1}} P\left(h_{t+1} | h_t, a_t\right) \\ \left[R\left(h_t, a_t\right) + \gamma V_\pi\left(h_{t+1}\right)\right] \mathrm{d}h_{t+1} \mathrm{d}a_t \end{aligned} \tag{2}$$

Then we consider the formulation of the reality gap. By modeling the environment with a POMDP, the characteristics of the environment are assumed to be captured by the tuple $\langle \mathcal{S}, \Omega, \mathcal{A}, O, P, R, \gamma \rangle$. Therefore, the gap in visual styles such as lighting and textures, which brings differences to the robot's observation, is reflected in the function O. The gap in dynamics such as friction coefficient, which brings differences to the environments' transition properties, is reflected in the function P. We use θ to denote the combination of parameters of O and P, and use θ^{real} and θ^{sim} to denote parameters in real-world and simulation environments respectively.

B. Instability

In DR, a randomization distribution of simulation environments is created and the agent is trained in some environments sampled from it. Concretely, DR consists of three steps: 1) design distributions Ξ_{rand} of θ^{sim}, e.g. a uniform distribution on $[\theta^{\mathrm{sim}}_{\mathrm{low}}, \theta^{\mathrm{sim}}_{\mathrm{high}}]$ when θ^{sim} is a real-valued scalar, to cover probable θ^{real}; 2) sample a set of parameters from it, e.g. $\Theta^{\mathrm{sim}} = \{\theta^{\mathrm{sim}}_i\}$, and generate a set of environments with them; 3) train agents in the created environments.

Because traditional RL regards the whole set of generated environments as a single task, the optimization objective is affected by the differences between environments. Concretely this reflects in the transformation of the Bellman equation from (2) to (3):

$$\begin{aligned} V_\pi\left(h_t\right) = \int_{a_t} \pi\left(a_t | h_t\right) \int_\theta \Big\{ \int_{h_{t+1}} P\left(h_{t+1} | h_t, a_t\right) \\ \cdot \big[R(h_t, a_t) + \gamma V_\pi\left(h_{t+1}\right)\big] \mathrm{d}h_{t+1} \Big\} p(\theta | h_t) \mathrm{d}\theta \mathrm{d}a_t \ , \end{aligned} \tag{3}$$

where $p\left(\theta | h_t\right)$ is the posterior of θ given h_t. Here the integral about θ shows that an unbiased evaluation of the RL policy requires consideration of all possible environments, which is because traditional methods merge all simulation environments and handle them simultaneously as one single RL task. This means that each step of optimization relies on data independently identically sampled from all environments.

For real-world tasks with a large extent of uncertainties, i.e. tasks in which the range of probable parameters θ^{real} is large, a complex distribution Ξ_{rand} and a large set of environment samples, i.e. a heavy DR, are needed to ensure coverage and the policy's adaptability. This reflects in the complexity of the integral about θ in (3). Therefore, ideal optimization should sample data independently identically from a complex set of environments. However, this is often hard to achieve in actual training, where only a small number of environments may be batched to approximate the whole distribution of environments. The error in this approximation leads to biases in policy evaluation, and further makes training unstable.

In summary, the instability in RL with DR is caused by the inability to train policies on a sufficiently large number of environments simultaneously. This leads us to consider decomposing the randomized domain to stabilize training.

IV. DOMAIN DECOMPOSITION

In the above section, we formulate reality gap and analyze the reason behind instability in training. In this section, we introduce DD and analyze why it can improve training stability.

A. Method

To decompose the original randomized domain into several simpler ones, we need a criterion for the decomposition. We notice that it is generally feasible to train RL policies in a single environment, so decomposing the domain according to environments, i.e. θ, should be sufficient. Intuitively this makes sense, as the robot needs to first identify what specific environment it is in and then act based on its judgments.

Therefore, we design our method of Domain Decomposition as: 1) train an environment identifier that classifies different environments, and 2) then train a separate RL policy for each class of environments. Concretely, the environment identifier approximates θ conditioned on a data sample, i.e. $\hat{p}\left(\theta | h_t\right)$, and a series of θ-conditioned policies π_θ is trained

for solving corresponding environments. The policy in our method takes the following form:

$$\pi\left(a_t\left|h_t\right.\right)=\int_\theta \pi_\theta\left(a_t\left|h_t\right.\right)\hat{p}\left(\theta\left|h_t\right.\right)\mathrm{d}\theta \tag{4}$$

This decomposition of π into π_θ and $\hat{p}$ leads to the following transformation of the Bellman equation: under the condition of accurate approximation of θ given h_t being possible, and each h_t comes from only one θ, the value function (3) strictly derives

$$\begin{aligned} V_\pi\left(h_t\right)&=\int_\theta\Big\{\int_{a_t}\pi\left(a_t\left|h_t\right.\right)\int_{h_{t+1}}P\left(h_{t+1}\left|h_t,a_t\right.\right)\\ &\cdot\left[R(h_t,a_t)+\gamma V_\pi\left(h_{t+1}\right)\right]\mathrm{d}h_{t+1}\mathrm{d}a_t\Big\}p(\theta|h_t)\mathrm{d}\theta \\ &=\int_\theta V_{\pi_\theta}\left(h_t\right)p\left(\theta\left|h_t\right.\right)\mathrm{d}\theta \end{aligned} \tag{5}$$

The main difference between (5) and (3) is the position of the integral: in traditional Bellman equation the integral about θ lies inside the value function V_π, while in Bellman equation for DD we have the integral outside the value functions V_{π_θ}. This means that in traditional methods evaluation of RL policy needs samples from all the environments, while in DD only samples from corresponding environments are needed, which is significantly easier to acquire.

A successful transformation from (3) to (5) calls for an accurate estimate of the environment identifier $\hat{p}$. Although training of $\hat{p}$ requires sampling from all the environments, in the setting of sim2real it can be done in a supervised manner and data can be adequately sampled: for RL policies a complete sequence of transitions is needed for one unbiased estimation, while for environment identifier every single transition is valid for training. From this perspective, our method of DD may be explained as transplanting adaptability from RL policy to an easier supervised model. Note that θ is generally acquirable in simulation environments, so using it as a part of data does not degrade our method's generality.

Therefore, we conclude that our method of DD should effectively reduce instability in training RL policies. As for the adaptability of DD, the analysis is in section 5.

B. Algorithm

DD consists of two parts, environment identifier, and policies for each environment.

As for the environment identifier, we denote it as $\hat{\theta}=\vartheta(h_t)$ and implement it using a neural network, as shown in Fig. 2. Note that we do not need to identify the exact original parameters of the environments; instead, normalized values or one-hot encoding are also acceptable as long as they reflect the differences between environments; below we still use θ to represent the encoding for simplicity in notation. Practically there are three ways for training the identifier: 1) training it before training RL policies, with data samples collected by a random policy; 2) training it and RL policies alternately; 3) first training RL policies for each class of simulation environments, then training the identifier with data samples collected by RL policies. Consider that collecting data samples with random policies is inefficient and that RL

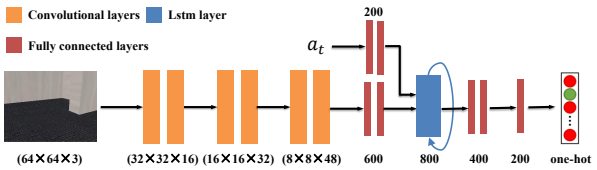


Fig. 2. Network architecture of the identifier. Each vertical bar indicates a layer; ReLU is used throughout and max-pooling is set between each group of the convolutional layers.

Algorithm 1 Domain Decomposition RL

Input: Distribution Ξ_{rand} and Simulator f^{sim};
Initialize: Policy network π, environment identifier ϑ, replay buffer B, a set of environments characterized by sampled parameters $\Theta\subset\Xi_{rand}$;
for episode= 1 **to** M **do**
  Sample an environment, get its parameter θ
  for $t=1,T$ **do**
    Observe ω_t and get history h_t
    Predict $\hat{\theta}$ using environment predictor $\vartheta(\theta|h_t)$;
    Execute action according to $a_t=\pi(h_t,\hat{\theta})$;
    Get reward r_t and new observation ω_{t+1};
    Store transition (ω_t, a_t, r_t, ω_{t+1}) in B;
  end for
  Sample a random minibatch of N transitions (ω_j, a_j, r_j, ω_{j+1}) from B and update ϑ and π using (6) and (7);
end for

policies have a dependence on the output of the identifier, we choose the second way in our method. We define the loss function using MSE and $l2$ norm, as follows:

$$\mathcal{L}_\vartheta=\int_\theta\int_{h_t}\left\|\vartheta(h_t)-\theta\right\|_2^2\mathrm{d}h_t\mathrm{d}\theta \tag{6}$$

As for the policies for each environment, we combine them together into an environment-conditioned policy $\pi(a_t|h_t,\hat{\theta})$, for simplicity in implementation. We approximate this policy with a neural network, which takes in $\hat{\theta}$ and h_t and outputs a_t in one-hot form. The optimization objective remains the form of (1), while the observation function and transition function is:

$$\begin{aligned} V_\pi\left(h_t\right)=&\int_a\pi(a|h_t,\hat{\theta})\int_{h_{t+1}}P\left(h_{t+1}\left|h_t,a_t\right.\right)\\ &\left[R\left(h_t,a_t\right)+\gamma V_\pi\left(h_{t+1}\right)\right]\mathrm{d}h_{t+1}\mathrm{d}a \end{aligned} \tag{7}$$

Beside the above designs, we also adopt (R2C) [14], a technique that maps the robot's observations with different styles but the same state together, thus alleviates the variation in visual styles. Although adopting this technique makes the tasks simpler to some extent, it reduces the consumption of computation resources and thus makes it tractable for us to demonstrate the effectiveness of DD. In our experiments, we compare our method with pure R2C and demonstrates that it is DD that brings the benefits.

The overall process is presented in Algorithm 1.

V. ANALYSIS OF ADAPTABILITY

Adaptability generally means the performance of a trained policy in testing. In the setting of DR, because the distribution of simulation environments are assumed to cover probable testing environments, adaptability reflects in the training performance across the whole distribution. We take this as the criterion in the following analysis.

Because DD contains two separate parts of learning, i.e. the environment identifier and the RL policies, the adaptability of the overall policy depends on the generalization of both parts, and there is not an existing framework that can be adopted for analysis, with the best of our knowledge. Besides, analysis of the generalization ability of RL algorithms with neural networks is often difficult to derive. Therefore, in the following part, we first perform a transformation of adaptability from RL to the error bound of function approximators. Then, we adopt the results in [28] to display the dependence concretely. Finally, we prove that the adaptability of DD can be preserved as the same in original DR so that DD indeed relieves the conflict.

A. Transformation to Error Bound

The expected return of RL training with DR is

$$G(\pi) = -J(\pi) = \int_{\theta\in\Xi} \Big[\int_{h\in\mathcal{H}} \rho_\pi(h) \int_{a\in\mathcal{A}} \pi(a|h)\, Q_\pi(a,h)\, \mathrm{d}a\mathrm{d}h \Big]\Big|_\theta p(\theta)\, \mathrm{d}\theta \tag{8}$$

where $\rho_\pi(h)$ is the distribution of history under policy π and $[\cdot]|_\theta$ is the θ-conditioned elements.

We denote the optimal expected return as G^* and the learned policy at k-th iteration of evaluation and improvement as π_k, and then the return gap between expected return achieved by π_k and the optimal one is $|G^* - G(\pi_k)|$. We first transform analysis to this return gap to the error of the function approximator e.g. a network that approximates the value function, through the following theorem.

Theorem 1 *Under the assumption of sufficient sampling and exploration, the return gap satisfies the following inequality:*

$$\lim_{k\to\infty} |G^* - G(\pi_k)| \le \frac{\delta_{\mathrm{com}} + 2\gamma\epsilon}{(1-\gamma)^2}\ , \tag{9}$$

where δ_{com} is the bound on error incurred in computation of policy update, and ϵ is the worst-case bound of error on function approximation:

$\max_{h_t\in\mathcal{H}} \left|\hat{V}_k(h_t) - V_{\pi_k}(h_t)\right| \le \epsilon, \quad k = 1, 2, \ldots$

Proof: [29] proves that
$\lim_{w\to\infty} \sup \max_{s\in\mathcal{S}} |V_{\pi_w}(s) - V^*(s)| \le \frac{\delta_{\mathrm{com}}+2\gamma\epsilon}{(1-\gamma)^2}$,
where V^* the optimal state value function. Because $\rho(s_0) \ge 0$ and $\sum_{s_0\in\mathcal{S}} \rho(s_0) = 1$ for any $s_0 \in \mathcal{S}$, the following equation gives the proof:

$$\begin{aligned} &|G(\pi_w) - G^*| \\ &\quad = |\Sigma_{s_0\in\mathcal{S}}\rho(s_0)\, V_{\pi_w}(s_0) - \Sigma_{s_0\in\mathcal{S}}\rho(s_0)\, V^*(s_0)| \\ &\quad \le \Sigma_{s_0\in\mathcal{S}}\rho(s_0) \lim_{w\to\infty} |V_{\pi_w}(s_0) - V^*(s_0)| \end{aligned}$$

B. Transformation to Expected Risk and Empirical Risk

Next, we concentrate on the function approximator. For simplicity we use $z = \langle h, V(h)\rangle$ to denote the data sample the RL algorithm generates and feeds to the function approximator and use $L(z,\theta)$ to denote the loss function about policy π_θ on data sample z; based on the assumption of sufficient samples, we may regard z as unbiased. The parameters of θ-conditioned policies are represented together as α_π; the parameters of θ approximator is represented as α_θ. The *expected risk* of function approximator then can be defined as:

$$\mathrm{Risk}(\alpha_\theta, \alpha_\pi) = \int_z \int_\theta L_{\alpha_\pi}(z,\theta)\, p_{\alpha_\theta}(\theta|z)\, \mathrm{d}\theta \mathrm{d}F(z) \tag{10}$$

Note that parameters of simulation environments, i.e. θ, can be regarded as independently identically sampled from Θ, it is reasonable to assume $F(z)$ correctly represents $G(\pi_k)$.

Because the numbers of both data samples and simulation environments are finite, the *empirical risk* takes the following form:

$$\mathrm{Risk}_{\mathrm{emp}}(\alpha_\theta, \alpha_\pi) = \frac{1}{m}\sum_{i=1}^{m} \frac{1}{n}\sum_{j=1}^{n} L_{\alpha_\pi}(z_j, \theta_i)\, p_{\alpha_\theta}(\theta|z) \tag{11}$$

where n and m are the numbers of data samples and possible θ respectively.

C. Results on Empirical Risk

Finally, because θ is taken from the simulation and uniquely characterizes the simulation environments, it is reasonable to assume that after grouping data samples z according to $p(\theta|z)$, the empirical risk (11) can be minimized. This allows us to adopt [28]'s work to analyze the gap between the expected risk and the empirical risk to get the theoretical bound of expected risk, i.e. ϵ in (9). The transformed theorem is as following:

Theorem 2 *With probability $1-\eta$ the parameters $\langle \alpha_\theta, \alpha_\pi\rangle$ minimizing the empirical risk satisfies*

$$\mathrm{Risk}(\alpha_\theta, \alpha_\pi) < \mathrm{Risk}_{\mathrm{emp}}(\alpha_\theta, \alpha_\pi) + \varepsilon \tag{12}$$

where ε satisfies

$$\begin{aligned} \eta =& 4\exp\big\{u_\theta(1+\ln\frac{2m}{u_\theta}) - \frac{(\varepsilon-1/m)^2}{(B_\theta - A_\theta)^2}m\big\} + \\ & 4\exp\big\{\ln(m) + u_\pi\big(1+\ln\frac{2n}{u_\pi}\big) - \frac{(\varepsilon-1/n)^2}{(B_\pi - A_\pi)^2}n\big\} \end{aligned} \tag{13}$$

A_π, B_π, A_θ and B_θ are bounds of following functions:

$$A_\theta \le \int_z L_{\alpha_\pi}(z,\theta)\, p_{\alpha_\theta}(\theta|z)\, \mathrm{d}F(z) \le B_\theta$$

$$A_\pi \le L_{\alpha_\pi}(z,\theta)\, p_{\alpha_\theta}(\theta|z) \le B_\pi$$

and u_θ and u_π are respectively the VC dimension of $p(\theta|z)$ and the combination of π_θ.

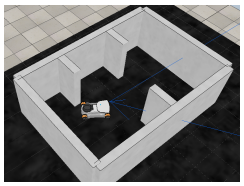
D. Discussion

So far we have gained the relationship between overall return gap, complexity (VC dimension) of function approximators, number of possible approximations of θ and data sample numbers. In our framework, traditional DR method can be represented as $m = 1$, while in our method $m > 1$. If our method and traditional method are to achieve the same return gap $|G^* - G(\pi_k)|$, we need to achieve the same ϵ, i.e. $\mathrm{Risk}(\alpha_\theta, \alpha_\pi)$; consider our function approximators all reach $\mathrm{Risk}_{\mathrm{emp}}(\alpha_\theta, \alpha_\pi) = 0$, together with the same ε with the same probability η, then we need the right side of (13) to be equal; then, because our method has a bigger m, we may adjust u_θ, i.e. the complexity of the function approximator used to fit $p(\theta|h_t)$, to keep the first exponential term equal; as for the second exponential term, because $\ln(m)$ is monotone increasing according to m and $u_\pi(1 + \ln\frac{2n}{u_\pi})$ is also monotone increasing according to u_π, we may decrease u_π, i.e. the complexity of the function approximator used to fit value functions.

Therefore, in theory, our method not only reduces the complexity of the training of RL policies but also can keep the adaptability through controlling u_θ; recall that u_θ refers to the supervised learning in our method, thus controlling it is practical.

(a) Real-world: overview

(b) Real-world: observation



(c) Simulation: overview

(d) Simulation: observation

Fig. 3. Setup of real-world and simulation environment.

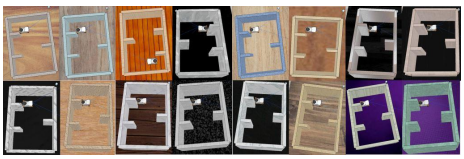


Fig. 4. Examples of our randomized domain of simulation environments

VI. EXPERIMENTS

In this section, we introduce our experiments in both simulation and real-world environments for validating the effectiveness of our method. Concretely, we are going to examine the following questions:

1) Does DD really improve the training stability?
2) Does DD preserve the adaptability of DR?
3) Does DD achieve better overall performance in the real-world task?
4) Does these results rely on the specific RL algorithm?

A. Experiment Setup

In our experiments, we consider a first-person patrolling task, in which the robot is only equipped with an on-board RGB camera and is required to move around the environment without collision and cover as much area as possible.

The real-world environment is a 2m×3m area, surrounded by 1m-high foams and containing several barriers (shown in Fig. 3a, b). The robot is KUKA youBot, whose action space is set to contain three discrete actions: turn left, turn right and move forward, all of which are open-loop-controlled, i.e. the effect of them may vary between different executions. For this environment, the agent needs to sequentially make at least 70 actions to successfully cover all the area; such a long time-horizon and the requirement for accuracy emphasize the significance of the performance of sim2real transfer, as a slight mistake in the decision may lead to a collision and thus failure of the task. Difficulties in simulating such real-world environment mainly come from: the complexity in textures of the foams and the floor; the variety in lighting caused by sunlight; the imperfectness in shape of foams; and uncertainty in the effects of each action (e.g. turn left) due to inconsistent friction coefficient across the floor.

For simulation environments, we constructed 150 of them on the V-REP platform (Fig. 3c, 3d, 4), each corresponding to a type of visual styles; these environments are similar to the real-world one in general shapes and colors, but have inevitable differences in textures, lighting, etc.; further, considering the uncertain effects of actions, we design 8 level of dynamic properties, with each level a certain size of movement, e.g. turning left 22.5° or 11.25° for one "turn-left" action.

As for RL, we split the ground area into an evenly distributed 3×4 grids to allocate reward signals: in each episode, once the agent reaches a new grid (i.e. not visited in current episode) it receives a $+1$ reward; once the agent collides with the walls or barriers it receives a -0.2 reward. To further avoid collisions, we give positive rewards only when the agent is within a specific region that is not near the walls or barriers. The grids and region are depicted in Fig. 8 as dashed and solid gray lines respectively. In our experiments, we mainly adopted PPO [15], which is a widely-used RL algorithm.

As for baseline, we implement R2C [14], one of the advanced methods recently developed for vision-based tasks with uncertain dynamics. Note that the core method of R2C only applies to visual randomization and pure DR is used to handle dynamics randomization. Because of the adoption of R2C in our method, this baseline also serves to examine the effectiveness of DD.

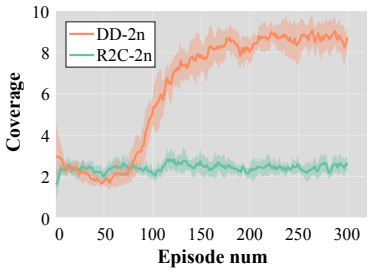


Fig. 5. Stability: coverage

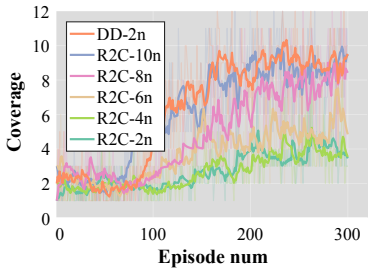


Fig. 6. Stability: adjusting the baseline

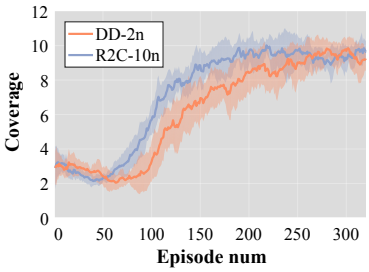


Fig. 7. Adaptability

B. Training Stability

In this subsection, we focus on the first question, i.e. whether DD indeed improves the training stability.

To reflect stability, we assign limitations to the scale of paralleled sampling of data. Concretely, the limitation is implemented as the number k of paralleled simulation environments. During training, each paralleled simulation environments provide $n = 300$ transitions after each episode, and the optimization steps are performed after each episode ends, resulting in a batch size of $n \times k$. Because n is fixed in our experiments, a larger k means the sampling is more unbiased and requires more computation resources; an algorithm is more stable if it can achieve the same final performance while requiring a smaller k.

We first limit $k = 2$ to compare the training performance of DD and R2C. As shown in Fig. 5, DD significantly outperforms R2C in coverage, which is the number of grids the agent successfully covers during an episode; the curves are smoothed using the exponential moving average method and averaged with 10 trials, while the shaded areas indicate one standard deviation. This shows that under a limited scale of paralleled sampling, DD can reach better training results than R2C. Further, our DD method can rapidly learn to cover averagely 9 grids; because there are variances in dynamics properties that may constrain the agent's ability to cover all 12 grids, a mean of 9 grid is generally satisfying for our task.

To validate the correctness of our implementation of R2C and also check our prediction on the relationship between k and training performance, we gradually increase k in R2C. As shown in Fig. 6, the final training result gradually increases with k, and when $k = 10$ R2C get close to DD-2n.

These results indicate that: 1) k truly is a key factor in RL with DR that represents training stability; 2) DD can improve training stability by requiring a lower k while achieving satisfying performance.

C. Adaptability

In this subsection, we focus on the second question, i.e. whether DD persists the adaptability of DR.

In the subsection above, we reach that R2C with 10 paralleled simulation environments can reach ideal training performance. Therefore, we take R2C-10n as the representative baseline for adaptability that the original DR should achieve. By training on a subset of designed dynamics properties and testing on remaining ones, we examine how our DD generalizes to unseen environments (with uncertain dynamics).

As shown in Fig. 7, DD adapts almost the same as original DR: both the final coverage and the growing speed of it do not exhibit inferiority. This experiment in simulation proves that DD can adapt as well as original DR while making training more stable. Results in the real-world environment are introduced in the next section.

(a) Simulation (b) Real-world

Fig. 8. Trajectories of the robot in simulation (orange) and real-world (purple) environments; gray grids show our design of reward function; starting points and directions of arrows respectively indicate position and direction of the robot.

D. Overall Performance in Real World

Then we focus on the third question, i.e. whether our method achieves better performance in the real-world task.

After training in simulation environments, we directly apply the identifier and policies to the real-word environment. We plot the trajectories of the robot as sequences of arrows. As shown in Fig. 8, DD achieves trajectories in the real-world almost the same as in simulation, which indicates a successful sim2real transfer. With our method, the robot successfully covers a large part of the ground before a collision happens, showing that our method practically completes this vision-based task with uncertain dynamics. A video of the experiment in real world is attached as supplementary material.

E. Independence to RL Algorithms

Finally, we focus on the fourth question, i.e. whether our method is compatible with different RL algorithms. Here we briefly list our experiments using A3C [16], another famous and commonly used RL algorithm. Fig. 9 shows the training curve on coverage; table I shows the relationship between training performance and k (superscripts and subscripts indicates the maximum and minimum value); Fig. 10 shows the adaptability of trained policies. These results are satisfying

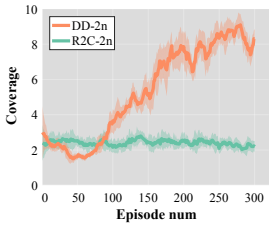


Fig. 9. Coverage (A3C)

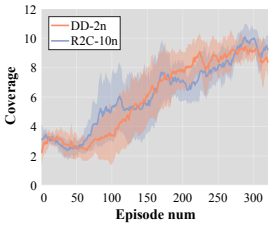


Fig. 10. Adaptability (A3C)

and similar to those got with PPO, showing that DD does not depend on specific RL algorithms. Compatibility with other commonly-used RL algorithms is left as our future study.

TABLE I
FINAL COVERAGE WITH DIFFERENT SCALE OF PARALLELED SAMPLING

Scale	DD-A3C	R2C-A3C	DD-PPO	R2C-PPO
2n	$8.20^{+3.80}_{-5.20}$	$3.75^{+3.75}_{-1.75}$	$9.40^{+2.6}_{-5.4}$	$3.85^{+2.15}_{-1.85}$
4n	$8.50^{+2.50}_{-2.50}$	$3.80^{+4.20}_{-1.80}$	$9.25^{+2.75}_{-1.25}$	$3.75^{+4.25}_{-1.75}$
6n	$8.95^{+3.05}_{-4.95}$	$5.04^{+5.96}_{-1.04}$	$9.40^{+2.60}_{-3.40}$	$6.1^{+4.90}_{-3.10}$
8n	$9.40^{+2.60}_{-6.40}$	$7.40^{+4.60}_{-4.40}$	$9.65^{+2.35}_{-2.65}$	$8.2^{+3.80}_{-4.20}$
10n	$9.60^{+2.40}_{-3.60}$	$8.55^{+3.45}_{-4.55}$	$9.80^{+2.20}_{-4.80}$	$9.3^{+2.70}_{-6.3}$

F. Summary

So far we have examined all the four questions. Our experiments show that our DD can improve training stability while preserving the adaptability that should be offered by DR, is practically feasible and can complete our vision-based patrol task with uncertain dynamics properties, and does not rely on the selection of a specific RL algorithm.

VII. CONCLUSION

In this paper, we propose DD, a method that relieves the conflict in DR between adaptability and training stability. One extra advantage of DD is that it does not require task-specific designs. With both theoretical and empirical proof of effectiveness and practicality, we believe DD can help complete a wide range of complex sim2real RL tasks.

REFERENCES

[1] J. Zhang, L. Tai, P. Yun, Y. Xiong, M. Liu, J. Boedecker, and W. Burgard, "VR-Goggles for robots: Real-to-sim domain adaptation for visual control," *IEEE Robotics and Automation Letters*, 2019.

[2] F. Zhang, J. Leitner, Z. Ge, M. Milford, and P. Corke, "Adversarial discriminative sim-to-real transfer of visuo-motor policies," *The International Journal of Robotics Research*, 2019.

[3] A. A. Rusu, M. Vecerík, T. Rothörl, N. Heess, R. Pascanu, and R. Hadsell, "Sim-to-real robot learning from pixels with progressive nets," in *Conference on Robot Learning*, 2017.

[4] P. F. Christiano, Z. Shah, I. Mordatch, J. Schneider, T. Blackwell, J. Tobin, P. Abbeel, and W. Zaremba, "Transfer from simulation to real world through learning deep inverse dynamics model," *arXiv:1610.03518 [cs]*, 2016.

[5] S. James and E. Johns, "3D simulation for robot arm control with deep q-learning," *arXiv:1609.03759 [cs]*, 2016.

[6] F. Zhang, J. Leitner, M. Milford, and P. Corke, "Modular deep q networks for sim-to-real transfer of visuo-motor policies," *arXiv:1610.06781 [cs]*, 2017.

[7] F. Sadeghi and S. Levine, "CAD2RL: real single-image flight without a single real image," in *Robotics: Science and Systems*, 2017.

[8] 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*, 2017.

[9] 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*, 2018.

[10] S. James, A. J. Davison, and E. Johns, "Transferring end-to-end visuomotor control from simulation to real world for a multi-stage task," in *Conference on Robot Learning*, 2017.

[11] F. Sadeghi, A. Toshev, E. Jang, and S. Levine, "Sim2Real viewpoint invariant visual servoing by recurrent control," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2018.

[12] O. M. Andrychowicz, B. Baker, M. Chociej, R. Jzefowicz, B. McGrew, J. Pachocki, A. Petron, M. Plappert, G. Powell, A. Ray, J. Schneider, S. Sidor, J. Tobin, P. Welinder, L. Weng, and W. Zaremba, "Learning dexterous in-hand manipulation," *The International Journal of Robotics Research*, vol. 39, no. 1, pp. 3–20, 2020.

[13] B. Mehta, M. Diaz, F. Golemo, C. J. Pal, and L. Paull, "Active domain randomization," in *Conference on Robot Learning*, 2019.

[14] S. James, P. Wohlhart, M. Kalakrishnan, D. Kalashnikov, A. Irpan, J. Ibarz, S. Levine, R. Hadsell, and K. Bousmalis, "Sim-to-real via sim-to-sim: Data-efficient robotic grasping via randomized-to-canonical adaptation networks," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2019.

[15] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv:1707.06347 [cs]*, 2017.

[16] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. P. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *International Conference on Machine Learning*, 2016.

[17] A. Hamalainen, K. Arndt, A. Ghadirzadeh, and V. Kyrki, "Affordance learning for end-to-end visuomotor robot control," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2019.

[18] J. Matas, S. James, and A. J. Davison, "Sim-to-real reinforcement learning for deformable object manipulation," in *Conference on Robot Learning*, 2018.

[19] J. Tobin, L. Biewald, R. Duan, M. Andrychowicz, A. Handa, V. Kumar, B. McGrew, A. Ray, J. Schneider, P. Welinder, W. Zaremba, and P. Abbeel, "Domain randomization and generative models for robotic grasping," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2018.

[20] X. Ren, J. Luo, E. Solowjow, J. A. Ojea, A. Gupta, A. Tamar, and P. Abbeel, "Domain randomization for active pose estimation," in *IEEE International Conference on Robotics and Automation*, 2019.

[21] X. Yan, M. Khansari, J. Hsu, Y. Gong, Y. Bai, S. Pirk, and H. Lee, "Data-efficient learning for sim-to-real robotic grasping using deep point cloud prediction networks," *arXiv:1906.08989 [cs]*, 2019.

[22] 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," in *Robotics: Science and Systems*, 2018.

[23] L. Pinto, M. Andrychowicz, P. Welinder, W. Zaremba, and P. Abbeel, "Asymmetric actor critic for image-based robot learning," in *Robotics: Science and Systems*, 2018.

[24] T. Schaul, D. Horgan, K. Gregor, and D. Silver, "Universal value function approximators," in *International Conference on Machine Learning*, 2015.

[25] A. Molchanov, T. Chen, W. Hönig, J. A. Preiss, N. Ayanian, and G. S. Sukhatme, "Sim-to-(multi)-real: Transfer of low-level robust control policies to multiple quadrotors," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2019.

[26] R. B. Slaoui, W. R. Clements, J. N. Foerster, and S. Toth, "Robust domain randomization for reinforcement learning," *arXiv:1910.10537 [cs]*, 2019.

[27] N. Heess, J. J. Hunt, T. P. Lillicrap, and D. Silver, "Memory-based control with recurrent neural networks," *arXiv:1512.04455 [cs]*, 2015.

[28] X. Su, S. Guo, and F. Chen, "Subjectivity learning theory towards artificial general intelligence," *arXiv:1909.03798 [cs]*, 2019.

[29] D. P. Bertsekas and J. N. Tsitsiklis, *Neuro-dynamic programming*. Athena Scientific, 1996.