SimGAN: Hybrid Simulator Identification for Domain Adaptation via Adversarial Reinforcement Learning

Yifeng Jiang[1,2], Tingnan Zhang[1], Daniel Ho[3], Yunfei Bai[3], C. Karen Liu[2], Sergey Levine[1,4] and Jie Tan[1]

***Abstract*— As learning-based approaches progress towards automating robot controllers design, transferring learned policies to new domains with different dynamics (e.g. sim-to-real transfer) still demands manual effort. This paper introduces SimGAN, a framework to tackle domain adaptation by identifying a hybrid physics simulator to match the simulated trajectories to the ones from the target domain, using a learned discriminative loss to address the limitations associated with manual loss design. Our hybrid simulator combines neural networks and traditional physics simulation to balance expressiveness and generalizability, and alleviates the need for a carefully selected parameter set in System ID. Once the hybrid simulator is identified via adversarial reinforcement learning, it can be used to refine policies for the target domain, without the need to interleave data collection and policy refinement. We show that our approach outperforms multiple strong baselines on six robotic locomotion tasks for domain adaptation.**

I. Introduction

Using simulation data to train a reinforcement learning (RL) policy for robotic skills provides an effective way to acquire diverse behaviors [1], [2]. While learning-based approaches have made controller design in simulation more automatic and less demanding of domain-specific knowledge, transferring a trained policy from simulation to the real hardware is often a heavily manual process. For example, with domain randomization techniques, practitioners select aspects of the simulation to randomize and train policies that are robust across a wide range of dynamics. Such ranges should be large enough to cover the unmodeled discrepancies between the simulation and real domains, while not being overly large as to adversely impact task performance.

Alternatively, system identification methods (System ID) have the promise to improve performance by utilizing observations from the real hardware to fit model parameters accurately. However, they make the strong assumption that the system is parameterized by a set of predefined parameters, and the true model lies within the model class. Further, when designing a similarity loss between simulated and real trajectories, the commonly used l_p-norm assumes an additive error model which can be unrealistic, since errors from misidentified dynamics may not be stochastic but rather systematic. As such, heuristically crafting such a metric between paired trajectories could require additional engineering effort.

[1]Robotics at Google, Mountain View, CA, USA
[2]Computer Science Department, Stanford University, Stanford, CA, USA
[3]Everyday Robots, X The Moonshot Factory, Mountain View, CA, USA
[4]EECS, University of California, Berkeley, Berkeley, CA, USA

Code available at: https://github.com/jyf588/SimGAN Accompanying video: https://youtu.be/McKOGllO7nc

In this paper, we take a different approach and ask: can we identify the simulator to the extent that simulated trajectories are hard to distinguish from real ones, without manual design of randomization parameters or heuristic assumptions about model classes or model noise? We propose a new method for simulation identification, in which a Generative Adversarial Network (GAN) [3] discriminator distinguishes between training and target domains and provides a learned similarity loss. In addition to reducing manual effort for loss design, a learned discriminative loss also lifts the requirement of calculating loss on *paired* trajectories. Instead, the GAN loss incentivizes trajectory matching on the distribution (set of trajectory tuples) level [4]. This allows system identification with excitation trajectories of *variable* lengths which could be unstable or sensitive to initial conditions.

The adversarial learning framework provides us an objective for system identification, but we still need to select a proper parameterization. Simply learning a set of constant parameters in a standard simulator may not be sufficiently expressive to capture unmodeled phenomena. On the other hand, discarding all physics knowledge and learning a black-box model may cause model degradation when the state-action distribution shifts during policy refinement. We instead construct a hybrid simulator, where we replace the constant simulation parameters with a differentiable state-action-dependent function. Since we want to minimize the GAN loss over each of the trajectory tuples, similar to maximizing accumulated rewards in reinforce learning, we treat our simulation parameter function as an "RL agent" and optimize it using policy gradient methods. With such parameterization, we are able to only collect data once for the identification without the need to interleave it with policy training.

In summary, our contributions include:

- A novel formulation of simulation identification as an adversarial RL problem;
- A learned GAN loss that provides weak set-level supervision to alleviate issues associated with manual loss design and sensitive excitation trajectories;
- An expressive hybrid simulator parameterization to alleviate the need for a carefully selected parameter set.

Our method can be used for domain adaptation of RL policies, by first identifying the simulator to match target domain trajectories and then refining the suboptimal behavior (data-collection) policy under the learned simulator. We evaluate our method against multiple strong baseline methods on two simulated robots and three target environments with different

dynamics from the source environment[1]. We also show that the same simulator learned with our method can be used for training multiple different motor skills.

II. Related Work

To overcome the dynamics discrepancies which renders policies trained in simulations to fail in the real world, domain randomization techniques learn policies from simulation with randomized physical parameters. This results in robust policies that could successfully bridge the sim-to-real gap [5], [6], [7], [8]. As overly wide randomization may hurt task performance, algorithms have been proposed to automatically adapt the sampling distribution of parameters throughout the course of training. For example, [1] used task performance as a metric to design a curriculum for gradually expanding randomization ranges. [9] used GAN to distinguish difficult dynamics from easier ones, and biased the sampling towards difficult parameters. Although we also use GAN in this paper, it is for a different purpose of distinguishing dynamics between the source and the target domains.

Although domain randomization does not need any target domain data, collecting a small amount of data in the target domain can significantly improve the performance of adapted policies. These methods include fine-tuning [10], sample-efficient model-based learning [11], [12], [13], meta-learning [14], [15], and latent space methods [16], [17], [2]. These few-shots adaptation methods often interleave data collection with policy improvement to maximize data efficiency and avoid distribution shift. Similar to our method, [18] trained a domain classifier to tackle adaptation, focusing on scenarios when the target domain is more constrained than source. While we also assume limited target domain access in this work, we focus on whether our learned dynamics can generalize with only one batch of data collection, to further reduce manual burden with data collection.

System identification is an effective way to improve simulation fidelity, and thus narrows the sim-to-real gap [19], [20]. Researchers have examined the identifiability of each robot parameter and designed maximally informative excitation trajectories to reveal these parameters [21], [22], [23]. Recent works [24], [25], [26], [27], [28] focused on if the identified model can improve policy performance, rather than seeking the ground-truth physical parameters. [25], [26] identified distributions of simulator parameters, [29] learned a state-dependent similarity loss by formulating a meta-learning problem using multi-task training data, and [30] identified generalized forces using RL as the optimizer. Although our work also uses RL, we adopt a GAN loss to better handle possibly large mismatches between paired trajectories than the mean-squared-errors, and use a different simulator parameterization which is in between of the previous works.

While our work aims at bridging the domain gap on dynamics, the same challenge also appears when transferring policies with visual input [31], [32], [33]. GAN-based techniques have shown promises in solving the problem of domain adaptation for visual sim-to-real gaps [34], [35], [36].

[1]Due to COVID-19, physical robots are not accessible.

III. Simulator Identification via Adversarial RL

Our method (Fig. 1) optimizes a hybrid simulator via reinforcement learning to match its generated trajectories to those recorded in the target environment, with the similarity reward provided by a co-learned GAN discriminator. A hybrid simulator, compared with a parameterized analytical simulator or a purely statistical dynamics model, strikes a balance between model expressiveness and maximizing the state-action space where the model is physically valid. By using a GAN-style loss rather than the l_p norm, we eliminate the need to align trajectories by abstracting the loss between paired trajectories to one between distributions.

A. Background

Trajectory Representation: A trajectory is a sequence $\boldsymbol{\tau} = (\boldsymbol{o}_0, \boldsymbol{a}_0, \dots \boldsymbol{o}_T)$, where $\boldsymbol{o}_t \in \mathcal{O}, \boldsymbol{a}_t \in \mathcal{A}$ are observations and actions respectively. With the initial state $\boldsymbol{s}_0 \sim p_0(\boldsymbol{s}_0)$ sampled from an initial state distribution in a state space $\mathcal{S}$, an observation function $g(\cdot) : \mathcal{S} \rightarrow \mathcal{O}$, a dynamics model $\boldsymbol{s}_{t+1} \sim P(\boldsymbol{s}_{t+1}|\boldsymbol{s}_t, \boldsymbol{a}_t)$, and a policy given by $\boldsymbol{a}_t \sim \pi(\boldsymbol{a}_t|\boldsymbol{o}_t)$, the probability of generating one trajectory $p(\boldsymbol{\tau})$ is given by $p_\pi(\boldsymbol{\tau}) = p_0(\boldsymbol{s}_0)g(\boldsymbol{o}_0|\boldsymbol{s}_0)\prod_{t=0}^{T-1} g(\boldsymbol{o}_{t+1}|\boldsymbol{s}_{t+1})P(\boldsymbol{s}_{t+1}|\boldsymbol{s}_t, \boldsymbol{a}_t)\pi(\boldsymbol{a}_t|\boldsymbol{o}_t)$.

Hybrid Simulator: A hybrid simulator combines an analytical simulator and learnable components. Compared with a parameterized analytical simulator $\boldsymbol{s}_{t+1} = G(\boldsymbol{s}_t, \boldsymbol{a}_t; \boldsymbol{c})$ where $\boldsymbol{c}$ are the simulation parameters, in a hybrid simulator that is represented as $\boldsymbol{s}_{t+1} \sim \widetilde{G}(\boldsymbol{s}_{t+1}|\boldsymbol{s}_t, \boldsymbol{a}_t, \boldsymbol{c}_t)$, $\boldsymbol{c}_t$ can be learned – for instance $\boldsymbol{c}_t \sim f_{\boldsymbol{\theta}}(\boldsymbol{c}_t|\boldsymbol{s}_t, \boldsymbol{a}_t)$ where $f_{\boldsymbol{\theta}}$ is a learnable function, such as a neural network with weights $\boldsymbol{\theta}$. Note that $\boldsymbol{c}_t$ is general, and can represent or replace any component(s) in the analytical simulator. For example, $\boldsymbol{c}_t$ can represent actual motor forces or contact forces, modifying or replacing those solved by the simulator. Trajectory probabilities under a hybrid simulator is given by

$$p_{\pi,f}(\boldsymbol{\tau}) = p_0(\boldsymbol{s}_0)g(\boldsymbol{o}_0|\boldsymbol{s}_0)\left(\prod_{t=0}^{T-1}\pi(\boldsymbol{a}_t|\boldsymbol{o}_t)g(\boldsymbol{o}_{t+1}|\boldsymbol{s}_{t+1})\right.$$
$$\left.\widetilde{G}(\boldsymbol{s}_{t+1}|\boldsymbol{s}_t, \boldsymbol{a}_t, \boldsymbol{c}_t)f_{\boldsymbol{\theta}}(\boldsymbol{c}_t|\boldsymbol{s}_t, \boldsymbol{a}_t)\right). \quad (1)$$

B. Hybrid Simulator Identification

The goal of hybrid simulator identification is to learn the function f to maximize the similarity of distribution between the transition tuples that are generated from our hybrid simulator and those collected from target domains. In this work, we first use a fixed, suboptimal *behavior policy* π_B to collect N offline trajectories $\{\boldsymbol{\tau}_R\}_{1,\dots N}$ in the target environment, and then optimize for an $f_{\boldsymbol{\theta}}$ to generate similar simulated trajectories under the same π_B:

$$\max_{\boldsymbol{\theta}} J(f_{\boldsymbol{\theta}}) = \mathbb{E}_{\boldsymbol{\tau}}\sum_{t=0}^{T}\gamma^t r(\boldsymbol{o}_t, \boldsymbol{a}_t, \boldsymbol{o}_{t+1}) \quad (2)$$
$$\text{s.t.} \quad \boldsymbol{\tau} \sim p_{\pi_B,f}(\boldsymbol{\tau}) \text{ in Eq. (1)},$$

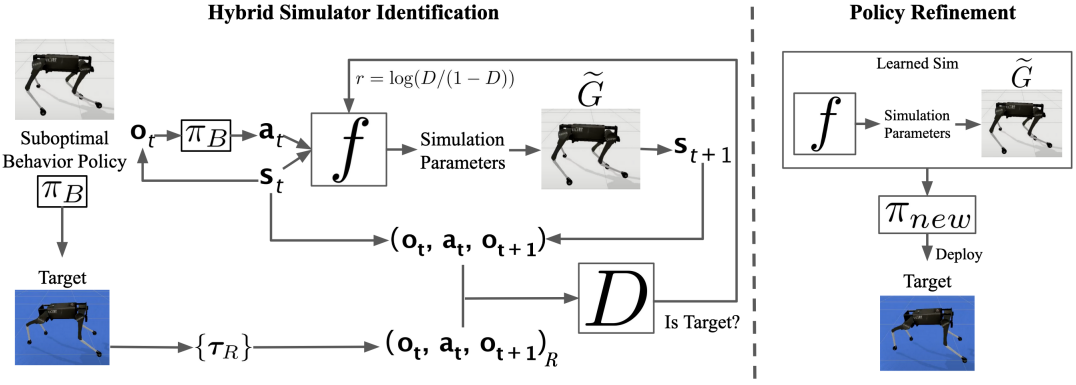

Fig. 1. Algorithm Overview: (Left) We propose to learn a hybrid simulator using Adversarial RL, with f being a learnable state-action-dependent simulation parameter function and $\widetilde{G}$ being an analytical simulator. Specifically, we collect a small amount of target domain data $\{\tau_R\}$ using a behavior policy π_B. A discriminator D is co-trained with f using standard GAN framework. (Right) The learned hybrid simulator is used as the training environment to train a new policy π_{new} that can achieve superior performance in the target domain. No target domain data are needed in this stage.

where γ is the reward discount factor and the reward function $r(\boldsymbol{o}_t, \boldsymbol{a}_t, \boldsymbol{o}_{t+1})$ measures the similarities between tuples generated by f and the target domain tuples from $\{\boldsymbol{\tau}_R\}$. Note that the optimization Eq. (2) resembles a typical formulation of reinforcement learning, with the only difference that the simulator function f is now to be optimized while the policy π_B is fixed. We purposefully use RL here because we would like to minimize the multi-step accumulated discrepancy between domains. To calculate the reward, we propose to adversarially train a discriminator $D_{\boldsymbol{w}}(d_t|\boldsymbol{o}_t, \boldsymbol{a}_t, \boldsymbol{o}_{t+1})$, parameterized by $\boldsymbol{w}$, to predict whether a tuple is generated from simulation (score $d = 0$) or sampled from the target domain $\boldsymbol{\tau}_R$ (score $d = 1$). Same as in previous works that combine GAN and RL [37], [38], our reward is constructed from the current discriminator score $r_t = \log(d_t/(1-d_t))$, favoring d values closer to 1. The training of the discriminator $D_{\boldsymbol{w}}$ alternates with the training of the generator f, as in standard GAN frameworks.

Depending on the sign of the reward, the learned dynamics can sometimes produce overly long or short trajectories in order to collect more rewards or avoid penalties, by sacrificing its accuracy. We propose a simple technique to offset r with a positive alive bonus discouraging over-early termination when the generated trajectories are unrealistically short, and vice versa. Inspired by [39], we design this *adaptive alive bonus* to be $b_i = \log(l_i/l_R)$, where l_i is the average length of the simulated trajectories under current f_i, and l_R is the average length of the target domain trajectories $\{\boldsymbol{\tau}_R\}$.

Finally, once the simulator component f is learned, a new control policy π_{new} is refined from π_B using standard deep reinforcement learning in the hybrid simulator $\widetilde{G}$ with the learned f. All the data is generated using the learned hybrid simulator. The fine-tuned policy π_{new} demonstrates strong performance in the target domain. Algorithm 1 summarizes the key steps of our hybrid simulator identification approach.

Algorithm 1 Simulator Identification and Policy Adaptation via Adversarial RL

Require: Suboptimal Behavior Policy: $\pi_B(\boldsymbol{a_t}|\boldsymbol{o_t})$,
Physics Simulator: $\widetilde{G}(\boldsymbol{s}_{t+1}|\boldsymbol{s}_t, \boldsymbol{a}_t, \boldsymbol{c_t})$,
Initial Simulator Parameter Function: $f_{\boldsymbol{\theta}_0}(\boldsymbol{c_t}|\boldsymbol{s_t}, \boldsymbol{a_t})$,
Initial Discriminator: $D_{\boldsymbol{w}_0}(p|\boldsymbol{o}_t, \boldsymbol{a}_t, \boldsymbol{o}_{t+1})$

// Data collection in target domain.
Collect N trajectories $\{\boldsymbol{\tau}_R\}_{1\cdots N}$ in Target by deploying π_B
$l_R :=$ average-length$(\{\boldsymbol{\tau}_R\})$

// Hybrid simulator identification.
for i = *0,1,... n* **do**
  Sample trajectories $\{\boldsymbol{\tau}_i\}$ from π_B under $\widetilde{G}$ and $f_{\boldsymbol{\theta}_i}$
  Update D from $\boldsymbol{w}_i$ to $\boldsymbol{w}_{i+1}$ with cross entropy loss:

$$-\hat{\mathbb{E}}_{\boldsymbol{\tau}_R}[\log D_{\boldsymbol{w}}] - \hat{\mathbb{E}}_{\boldsymbol{\tau}_i}[\log(1 - D_{\boldsymbol{w}})]$$

  $l_i :=$ average-length$(\{\boldsymbol{\tau}_i\})$
  Calculate adaptive alive bonus: $b_i := \log(l_i/l_R)$
  Update f from $\boldsymbol{\theta}_i$ to $\boldsymbol{\theta}_{i+1}$ using PPO, with reward r_t:

$$d_t = D_{\boldsymbol{w}_{i+1}}(\boldsymbol{o}_t, \boldsymbol{a}_t, \boldsymbol{o}_{t+1})$$
$$r_t = \log(d_t/(1-d_t)) + b_i$$

end

// Refining the policy in identified simulator.
$\pi_{new} := \pi_B$
Train π_{new} using PPO under $\widetilde{G}$ with $f_{\boldsymbol{\theta}_n}$
return π_{new}

C. Parameterization of Hybrid Simulator

The parameterization of $\boldsymbol{c}_t \sim f$ controls how much of the simulator is learned and how much is inherited from the analytical simulator. On one hand, during policy adaptation, the state-action distribution shifts away from where f is trained on, so f needs to avoid model degradation and remain physically valid for a reasonably large region of state-action space. On the other hand, $\boldsymbol{c}_t$ should be expressive enough to capture a large range of unmodeled domain discrepancies.

In this work, we model $\boldsymbol{c}_t \sim N(\boldsymbol{\mu}_t|(\boldsymbol{s}_t, \boldsymbol{a}_t), \boldsymbol{\sigma}_t|(\boldsymbol{s}_t, \boldsymbol{a}_t)))$ as a *state-action-dependent, stochastic* simulation parameter function, where $(\boldsymbol{\mu}_t, \boldsymbol{\sigma}_t) = f_{\boldsymbol{\theta}}(\boldsymbol{s}_t, \boldsymbol{a}_t)$, meaning f out-

puts the means and variances of the simulation parameters. Specifically, $\boldsymbol{c}$ include contact-related simulator parameters (friction, restitution, spinning friction, contact error reduction rate (ERP) of each body in contact) and scaling factors of each motor action (i.e., $\boldsymbol{\tau} = \boldsymbol{c}_a \boldsymbol{a}$, where $\boldsymbol{a}$ is torque output from the policy, $\boldsymbol{c}_a$ is the subvector of $\boldsymbol{c}$ representing the scaling factors, and $\boldsymbol{\tau}$ is the actual motor input passed to the analytical simulator). This choice reflects the fact that modeling contact [40] and joint actuation [41] are important for robotics tasks. But more importantly, state-action-dependent contact and actuator parameters respectively cover the spaces of external and internal forces in the equation of motions:

$$\ddot{\boldsymbol{q}} = \boldsymbol{M}(\boldsymbol{q})^{-1}(\boldsymbol{C}(\boldsymbol{q}, \dot{\boldsymbol{q}}) + \boldsymbol{\tau} + \boldsymbol{J}^T(\boldsymbol{q})\boldsymbol{p}),$$

where $\boldsymbol{\tau}$ and $\boldsymbol{p}$ are internal and external forces, which are directly influenced by our learned simulation parameter function $f_{\boldsymbol{\theta}}$, $\{\boldsymbol{q}, \dot{\boldsymbol{q}}\}$ is the state vector, $\boldsymbol{M}$ is the mass matrix, $\boldsymbol{J}$ is the Jacobian, and $\boldsymbol{C}$ is the Coriolis force. As we will show in the experiments, dynamics discrepancies not included in $\boldsymbol{c}$ such as inertial differences, or Young's modulus of a deformable surface, can also be "absorbed" by our state-dependent f (hence state-dependent $\boldsymbol{\tau}$ and $\boldsymbol{p}$). Learning an f with non-vanishing variance $\boldsymbol{\sigma}_t$ makes our hybrid simulator stochastic and improves the robustness of the refined policy.

IV. Experiment Setup

We set up our experiments to evaluate if our method can (1) improve domain adaptation for robots with different morphologies (2) handle dynamics discrepancies that are important in sim-to-real transfer (3) handle dynamics discrepancies not intuitively mapped to the list of parameters that our model identifies but can be absorbed by the state-dependent nature of our model. We also lay out the implementation details of our method.

A. Robots and Control Policies

We evaluate the performance of our framework both on standard Gym [42] benchmarks and on robots of real-world interest. We utilize two simulated robots in our experiments: a 4-link, 6-DoF 2D hopper and the 18-DoF Unitree Laikago quadruped [43]. The policy for each robot is a 3-layer feed-forward neural network, where the input is robot observation, and the output is the motor torque. The observation space of the hopper is the full robot state to match implementations in OpenAI Gym [42]. The observation space of the Laikago robot includes sensor measurements and state estimations we can collect from the physical robot, including: root global orientation, root height, root linear velocities and the 12-DoF joint angles. A 10% white noise is added to each observation, and a 5% white noise is added to each torque command. The control frequency for both robots is set to 50Hz.

In most of the experiments except in the multi-task setting (Sec.V-B), we choose a reward function that encourages the robot to move forward with smooth joint motion and low energy:

$$r(\boldsymbol{s}, \boldsymbol{a}) = w_c + w_v v_x - w_a \left\|\boldsymbol{a}\right\|^2 - w_j \left\|\boldsymbol{j}\right\|_0 - w_s \left\|\ddot{\boldsymbol{q}}\right\|,$$

where v_x is the root forward velocity, $\left\|\boldsymbol{j}\right\|_0$ is number of joints at joint limit, and w_c, w_v, w_a, w_j, w_s are the weights. These weights are tuned differently for Hopper and Laikago robots to learn a behavior policy π_B that is optimal in the source domain (suboptimal in target domains), and are kept unchanged for policy refinement and baseline methods.

B. Target Domain Setup

We use PyBullet [44] as the simulation platform in this work. The source environment contains the robot and a rigid ground plane in PyBullet. We set up the following target environments for domain transfer:

Deform: Contacts are difficult to model but play a key role in locomotion. For this reason, we first test our method for adaptation to different contact dynamics. We set up a large mattress-like deformable cube in PyBullet, replacing the hard floor for the robots to locomote on. This gap is also chosen to test the expressiveness of our method, as deformable bodies are simulated in PyBullet independently with FEM methods, hence their physical properties are controlled by a set of parameters that are not intuitively mapped to the list of contact parameters that our model identifies.

Power: Actuator dynamics is one of the major causes of the sim-to-real gap [8], [41]. Therefore, for the Laikago, we set up a target environment where the actual torques are the commanded ones subtracting $c\dot{\boldsymbol{q}}$, mimicking unmodeled motor friction and Back-EMF torques [8]. For the Hopper, as this discrepancy is not hard enough to fail the behavior policy, we halve its ankle torque, mimicking a broken motor.

Heavy: It is important that our model can handle domain shifts that are not caused by actuator or contact dynamics, as discussed in Sec. III-C. Here we evaluate if our identified state-action-dependent actuator and contact parameters can absorb changes in inertia. For the Laikago, we increase the masses of its front-left leg links by 3kg and move the Center of Mass (CoM) of the leg down by 10cm; for the Hopper, we increase the masses of its torso and thigh by 3kg in total and move the CoM of both links up by 10cm.

C. Implementation of Algorithm

We use a policy that is trained to saturation in the source environment as π_B. During data-collection using π_B, we add Gaussian noise with standard deviation of 0.25 to each policy action to make the recorded dataset more diverse. In each of the target environments, we collect 200 trajectories. The lengths of these trajectories (time before falling) ranges between one to two seconds. In total, the target environment dataset consists of 10,000 to 20,000 environment steps.

We model the simulator parameter function f as a 3-layer two-branch feed-forward neural network with $\tanh$ activations, with each branch outputting contact and actuator parameters (mean and log-variance) separately. We use large output range scaling for f without tuning. Differing from policies with partial observations, the input to f is the full state $\boldsymbol{s}_t$ concatenated with actions from $\pi_B(\boldsymbol{a}_t|\boldsymbol{o}_t)$. We model the discriminator as a 3-layer neural network as well. We train f with PPO [45] until the average discriminator

score converges, which takes roughly 200 iterations for Hopper and 1,000 iterations for Laikago. Between each RL iteration, the discriminator is trained for five epochs.

During the policy refinement step, no more data in the target domain is collected and f is kept unchanged. To re-train π_{new} under the dynamics of learned f, we warm-start π_{new} from π_B with halved learning rate. As the training environment has changed drastically, we only inherit the policy weights but re-initialize the value function. Consistent with the training curves of the behavior policies, Laikago takes more PPO iterations (500) than the Hopper (250) before learning converges.

V. Evaluations

We evaluate our method on the aforementioned robots and target domains with the following baseline methods:

1) Fine-tuning from π_B (FT): We directly fine-tune π_B in the target environments with the 200 trajectory budget in each environment, which is the same data budget as in our method. However, as the policy improves, FT generally produces longer rollouts and ends up using more samples than our method. We ran a grid search for the batch-size to optimize data-efficiency when using PPO to train FT.
2) Domain randomization (DR): The selected parameters for DR and their ranges are listed in Table I. For Laikago, we inherit the selection and ranges from previous works [2], [8] and add three contact-related parameters to match the output of f and to account for the contact gaps. The same selection is used for Hopper. For parameter ranges not listed in the previous papers, we manually search for the largest ranges that allow for successful policy training. We ran a grid search for the best-performing *learning* parameters.
3) DR + FT: We fine-tune the Domain Randomization policies in the target environments for 200 trajectories.
4) System ID with recorded open-loop actions (SysID-o): Using the same 200-trajectory dataset from π_B, we execute the recorded action sequences from the same initial states in the simulator. Following [25], we optimize the *state-independent* means and variances of parameters used in f. The same l_p-norm fitness function and trajectory Gaussian filtering are used and the parameters are optimized by CMA-ES [46]. We then re-train from π_B under the learned parameter distribution.
5) System ID with closed-loop actions (SysID-c): This baseline is same as above except that the rollouts in the simulator use closed-loop actions from π_B.

Additionally, we demonstrate the cross-task generalization of our hybrid simulator: we show that our hybrid simulator learned with data from one behavior policy can be used for learning different motor tasks. Finally, we conduct ablation studies to understand the impact of different design decisions, including data budget, alive bonus, and more task-relevant data. The qualitative results are presented in the accompanying video.

TABLE I
Dynamics parameters randomization ranges.

Parameter	**DR Range Hopper**	**DR Range Laikago**
Mass Ratio	[0.5, 1.5]	[0.8, 1.2]
Inertia Ratio	[0.4, 1.8]	[0.5, 1.5]
Motor Scaling	[0.5, 1.5]	[0.8, 1.2]
Motor Friction	[0.2, 3.0]	[0.2, 2.0]
Latency (ms)	[0, 40]	[0, 40]
Lateral Friction	[0.4, 1.5]	[0.4, 1.25]
Spinning Friction	[0, 0.2]	[0, 0.1]
Restitution	[0, 1.5]	[0, 0.5]
Contact ERP	[15, 1500]	[100, 1500]

A. Performance in the Target Domains

As shown in Table II, our method outperforms the baseline methods on five of the six domain adaptation experiments. Each number in Table II shows the average task reward in the target environment over 30 rollouts from random initial states. For each method, we report the standard deviation of five trials with different random seeds.

One advantage of our method over DR is that DR tends to produce policies that trade performance for robustness. For example, in the Laikago/Deform environment, our framework learns a walking gait that is 1.3x faster than the DR policies. As shown in the accompanying video, with the same reward function, DR learns to use conservative, small and frequent gaits, while the proposed method adopts agile galloping gaits with larger strides.

For the System ID baselines, we are unable to identify simulator parameters useful for improving policy performance in the target domain. For example, the task rewards of both variants of System ID are an order of magnitude lower (SysID-o: 189, SysID-c: 199) than our method in the Laikago/Power environment, regardless of hyper-parameters. Our investigation suggests that the System ID trajectories generated from the behavior policy are highly sensitive to initial conditions and physical parameters, possibly because even small errors in contact-rich locomotion environments can make trajectories diverge quickly. Therefore, executing the same open-loop control sequence in simulator and target domains will lead to mismatched observation sequences, and executing the same closed-loop policy will lead to mismatched observation and control sequences. In both cases, such large mismatch invalidates pairwise trajectory-based similarity metrics. Different from l_p-norms, our method does not require pairwise trajectory comparison, and instead adopts distributional supervision via GAN loss.

B. Cross-task Generalization

Our identified hybrid simulator can generalize and be used to learn different tasks, instead of overfitting to the task that the behavior policy solves. We use the same hybrid simulator that is learned for the Deform environment using the data collected by the behavior policy of the Laikago walking forward. But, during the policy refinement, we change the reward function to encourage sideway walk. After the policy

TABLE II

Average task reward in six domain adaptation experiments.

Environment	π_B	**Ours**	**FT**	**DR**	**DR+FT**
Hopper/Deform	205	**1389 ± 148**	622 ± 78	815 ± 422	996 ± 530
Hopper/Power	178	**1601 ± 19**	411 ± 75	590 ± 123	1237 ± 466
Hopper/Heavy	223	1658 ± 71	**1986 ± 38**	807 ± 254	1905 ± 193
Laikago/Deform	481	**2650 ± 220**	2023 ± 357	2206 ± 204	2231 ± 152
Laikago/Power	271	**2484 ± 106**	908 ± 73	2408 ± 153	2415 ± 144
Laikago/Heavy	308	**2211 ± 326**	550 ± 124	1523 ± 283	1865 ± 183

refinement, the Laikago is able to walk sideways in the target domain (see accompanying video). The result suggests that the learned hybrid simulator is generalizable, and can be used for learning multiple different motor skills.

C. Ablation Studies

We conduct three ablation studies in two domain adaptation tasks to gain more insight into our method. The results below are averaged across three random seeds.

Data budget in target domain: Table III shows the average reward of refined policies in the target environments, when varying the number of target environment trajectories used for learning the simulator. We are glad to find that our method could potentially use a even smaller amount of data in the target domain (e.g. with 50 trajectories) without noticeable performance deterioration.

TABLE III

Average task reward in target domains with varying numbers of target trajectories.

Num Trajs	**200**	**100**	**50**	**25**
Hopper/Heavy	1658	1554	1686	1551
Laika/Deform	2650	2809	2561	2272

Adaptive alive bonus: Table IV shows the average reward of refined policies in the target environments, with or without adaptive alive bonus b_i applied during the learning of the hybrid simulator. In the Hopper/Heavy environment, adaptive alive bonus helps to improve the simulation fidelity and thus increases the adapted policy performance.

TABLE IV

Average task reward in target domains with variation in alive bonus.

	With b_i	**Without b_i**
Hopper/Heavy	1658	1186
Laika/Deform	2650	2569

More iterations of Algorithm 1: One possible way to improve our framework is to run the whole Algorithm 1 for another iteration by using the refined policy as the new behavior policy in the second iteration. We find that this does not clearly improve the performance of the adapted policies. Since our hybrid simulator still uses the analytical simulation model as a strong prior and we have shown that it is already robust to the state-action distribution shift, collecting more task-specific data does not necessarily help in our case.

VI. Discussions

In this paper, we tackle the challenge of transferring policies to a new domain with different dynamics, by identifying a hybrid simulator that better matches target domain trajectories. We propose to formulate simulation identification as an adversarial reinforcement learning problem. We use a learned GAN loss in place of the standard mean-squared-errors to measure the discrepancies between distributions of transition tuples, and augment a classical physics simulator with a learned state-action-dependent stochastic simulator parameter function to improve its expressiveness to simulate unmodeled phenomena. Our method shows promising results on six difficult dynamical domain adaptation tasks, outperforming strong baseline methods.

We find that a good parameterization of the hybrid simulator plays a critical role in our system. Before we narrowed down to use contact parameters and motor scaling as the parameterization, we experimented with other alternatives. For example, we tried directly learning all contact forces, similar to [30]. But, it performed much worse because there is no constraint to prevent energy injection into the physical system. Consequently, the locomotion policy learned to exploit these fictitious forces in simulation which did not exist in the target domain. An interesting future direction would be to experiment with different parameterizations and add physical constraints to the learned components of the hybrid simulator, so that fundamental physical laws, such as energy conservation, are always respected.

Our current implementation has several limitations. First, the behavior policy cannot perform too poorly in the target domain. It needs to collect long enough trajectories (longer than one second at least) before the robot falls so that the trajectories are informative for learning the discriminator and the simulation parameter function. Second, while we keep the hyperparameters the same across all three target environments, they are currently tuned differently for the two robots, which might increase manual effort. Third, while the results for sim-to-sim transfer are promising, due to COVID-19, we are not able to test our method on real robots. In the future, we plan to address these limitations, and test our method for transferring locomotion policies from simulation to a real Laikago robot.

REFERENCES

[1] I. Akkaya, M. Andrychowicz, M. Chociej, M. Litwin, B. McGrew, A. Petron, A. Paino, M. Plappert, G. Powell, R. Ribas, *et al.*, "Solving rubik's cube with a robot hand," *arXiv preprint arXiv:1910.07113*, 2019.

[2] X. B. Peng, E. Coumans, T. Zhang, T.-W. E. Lee, J. Tan, and S. Levine, "Learning agile robotic locomotion skills by imitating animals," in *Robotics: Science and Systems*, 07 2020.

[3] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Advances in neural information processing systems*, 2014, pp. 2672–2680.

[4] J.-Y. Zhu, T. Park, P. Isola, and A. A. Efros, "Unpaired image-to-image translation using cycle-consistent adversarial networks," 2020.

[5] 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)*, May 2018, pp. 1–8.

[6] L. Pinto, J. Davidson, R. Sukthankar, and A. Gupta, "Robust adversarial reinforcement learning," in *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, 2017, pp. 2817–2826.

[7] O. M. Andrychowicz, B. Baker, M. Chociej, R. Jozefowicz, B. McGrew, J. Pachocki, A. Petron, M. Plappert, G. Powell, A. Ray, *et al.*, "Learning dexterous in-hand manipulation," *The International Journal of Robotics Research*, vol. 39, no. 1, pp. 3–20, 2020.

[8] 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," 2018.

[9] B. Mehta, M. Diaz, F. Golemo, C. J. Pal, and L. Paull, "Active domain randomization," 2019.

[10] A. A. Rusu, M. Večerík, T. Rothörl, N. Heess, R. Pascanu, and R. Hadsell, "Sim-to-real robot learning from pixels with progressive nets," ser. Proceedings of Machine Learning Research, S. Levine, V. Vanhoucke, and K. Goldberg, Eds., vol. 78. PMLR, 13–15 Nov 2017, pp. 262–270. [Online]. Available: http://proceedings.mlr.press/v78/rusu17a.html

[11] Y. Yang, K. Caluwaerts, A. Iscen, T. Zhang, J. Tan, and V. Sindhwani, "Data efficient reinforcement learning for legged robots," in *Conference on Robot Learning*. PMLR, 2020, pp. 1–10.

[12] S. Desai, H. Karnan, J. P. Hanna, G. Warnell, and P. Stone, "Stochastic grounded action transformation for robot learning in simulation," in *IEEE/RSJ International Conference on Intelligent Robots and Systems(IROS 2020)*, October 2020.

[13] Y. Song, A. Mavalankar, W. Sun, and S. Gao, "Provably efficient model-based policy adaptation," 2020.

[14] X. Song, Y. Yang, K. Choromanski, K. Caluwaerts, W. Gao, C. Finn, and J. Tan, "Rapidly adaptable legged robots via evolutionary meta-learning," *arXiv preprint arXiv:2003.01239*, 2020.

[15] A. Nagabandi, I. Clavera, S. Liu, R. S. Fearing, P. Abbeel, S. Levine, and C. Finn, "Learning to adapt in dynamic, real-world environments through meta-reinforcement learning," in *International Conference on Learning Representations*, 2018.

[16] W. Yu, V. C. Kumar, G. Turk, and C. K. Liu, "Sim-to-real transfer for biped locomotion," 2019.

[17] W. Yu, J. Tan, Y. Bai, E. Coumans, and S. Ha, "Learning fast adaptation with meta strategy optimization," 2020.

[18] B. Eysenbach, S. Asawa, S. Chaudhari, R. Salakhutdinov, and S. Levine, "Off-dynamics reinforcement learning: Training for transfer with domain classifiers," 2020.

[19] A. Farchy, S. Barrett, P. MacAlpine, and P. Stone, "Humanoid robots learning to walk faster: From the real world to simulation and back," in *Proc. of 12th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS)*, May 2013. [Online]. Available: http://www.cs.utexas.edu/users/ai-lab?AAMAS13-Farchy

[20] J. Tan, Z. Xie, B. Boots, and C. K. Liu, "Simulation-based design of dynamic controllers for humanoid balancing," in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2016, pp. 2729–2736.

[21] K. Ayusawa, G. Venture, and Y. Nakamura, "Identifiability and identification of inertial parameters using the underactuated base-link dynamics for legged multibody systems," *The International Journal of Robotics Research*, vol. 33, no. 3, pp. 446–468, 2014. [Online]. Available: https://doi.org/10.1177/0278364913495932

[22] M. Gautier and W. Khalil, "Exciting trajectories for the identification of base inertial parameters of robots," *The International Journal of Robotics Research*, vol. 11, no. 4, pp. 362–375, 1992. [Online]. Available: https://doi.org/10.1177/027836499201100408

[23] M. Jegorova, J. Smith, M. Mistry, and T. Hospedales, "Adversarial generation of informative trajectories for dynamics system identification," 2020.

[24] A. Rajeswaran, S. Ghotra, B. Ravindran, and S. Levine, "Epopt: Learning robust neural network policies using model ensembles," 2017.

[25] 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," 2019.

[26] F. Muratore, C. Eilers, M. Gienger, and J. Peters, "Bayesian domain randomization for sim-to-real transfer," 2020.

[27] S. Zhu, A. Kimmel, K. E. Bekris, and A. Boularias, "Fast model identification via physics engines for data-efficient policy search," 2018.

[28] S. Desai, I. Durugkar, H. Karnan, G. Warnell, J. Hanna, and P. Stone, "An imitation from observation approach to transfer learning with dynamics mismatch," in *NeurIPS*, 2020.

[29] K. Morse, N. Das, Y. Lin, A. S. Wang, A. Rai, and F. Meier, "Learning state-dependent losses for inverse dynamics learning," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2020, pp. 5261–5268.

[30] R. Jeong, J. Kay, F. Romano, T. Lampe, T. Rothorl, A. Abdolmaleki, T. Erez, Y. Tassa, and F. Nori, "Modelling generalized forces with reinforcement learning for sim-to-real transfer," 2019.

[31] F. Sadeghi and S. Levine, "Cad2rl: Real single-image flight without a single real image," *arXiv preprint arXiv:1611.04201*, 2016.

[32] 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 *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2017, pp. 23–30.

[33] N. Ruiz, S. Schulter, and M. Chandraker, "Learning to simulate," 2019.

[34] K. Bousmalis, A. Irpan, P. Wohlhart, Y. Bai, M. Kelcey, M. Kalakrishnan, L. Downs, J. Ibarz, P. Pastor, K. Konolige, S. Levine, and V. Vanhoucke, "Using simulation and domain adaptation to improve efficiency of deep robotic grasping," 2017.

[35] 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," 2019.

[36] K. Rao, C. Harris, A. Irpan, S. Levine, J. Ibarz, and M. Khansari, "Rl-cyclegan: Reinforcement learning aware simulation-to-real," 2020.

[37] J. Fu, K. Luo, and S. Levine, "Learning robust rewards with adversarial inverse reinforcement learning," *arXiv preprint arXiv:1710.11248*, 2017.

[38] J. Ho and S. Ermon, "Generative adversarial imitation learning," in *Advances in neural information processing systems*, 2016, pp. 4565–4573.

[39] I. Kostrikov, K. K. Agrawal, D. Dwibedi, S. Levine, and J. Tompson, "Discriminator-actor-critic: Addressing sample inefficiency and reward bias in adversarial imitation learning," in *International Conference on Learning Representations*, 2018.

[40] 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)*. IEEE, 2016, pp. 30–37.

[41] J. Hwangbo, J. Lee, A. Dosovitskiy, D. Bellicoso, V. Tsounis, V. Koltun, and M. Hutter, "Learning agile and dynamic motor skills for legged robots," *Science Robotics*, vol. 4, no. 26, 2019.

[42] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," 2016.

[43] Unitree, "Laikago: Let's challenge new possibilities," 2018. [Online]. Available: http://www.unitree.cc/

[44] E. Coumans and Y. Bai, "Pybullet, a python module for physics simulation in robotics, games and machine learning," 2017.

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

[46] N. Hansen, Y. Akimoto, and P. Baudis, "CMA-ES/pycma on Github," Zenodo, DOI:10.5281/zenodo.2559634, Feb. 2019. [Online]. Available: https://doi.org/10.5281/zenodo.2559634