Dynamics Randomization Revisited:
A Case Study for Quadrupedal Locomotion

Zhaoming Xie[*1,2], Xingye Da[1], Michiel van de Panne[2], Buck Babich[1], Animesh Garg[1,3]

Abstract— Understanding the gap between simulation and reality is critical for reinforcement learning with legged robots, which are largely trained in simulation. However, recent work has resulted in sometimes conflicting conclusions with regard to which factors are important for success, including the role of dynamics randomization. In this paper, we aim to provide clarity and understanding on the role of dynamics randomization in learning robust locomotion policies for the Laikago quadruped robot. Surprisingly, in contrast to prior work with the same robot model, we find that direct sim-to-real transfer is possible without dynamics randomization or on-robot adaptation schemes. We conduct extensive ablation studies in a sim-to-sim setting to understand the key issues underlying successful policy transfer, including other design decisions that can impact policy robustness. We further ground our conclusions via sim-to-real experiments with various gaits, speeds, and stepping frequencies. Additional Details: **pair.toronto.edu/understanding-dr/**

I. INTRODUCTION

Deep reinforcement learning (RL) is increasingly successful in adoption as a feasible approach for synthesizing locomotion policies for legged robots. However, direct training on hardware is often impractical due to the sample efficiency of RL algorithms and unsafe exploration behaviors during the training phase. Instead, a physics-based simulator is commonly employed during training. Moreover, the discrepancy between the simulator and the real world, also known as the "reality gap," can cause direct sim-to-real transfer to fail. One way to combat this problem is to employ *dynamics randomization*, where parameters of the simulation system are randomized during training, in order to obtain policies that are robust to modeling errors. This has been used extensively in recent work in sim-to-real transfer for learned legged robot policies [1]–[5]. However, conflicting observations have been made in other work where no dynamics randomization was needed for sim-to-real transfer [6]–[8].

In this paper, we revisit dynamics randomization in detail, with the aim of providing an improved understanding of when it should be used, grounded in sim-to-sim and sim-to-real experiments using the Unitree Laikago quadruped. More specifically, we make the following contributions:

1) We demonstrate that dynamics randomization is *not necessary* for successful sim-to-real transfer, across multiple gaits and speeds. Note that the same robot model has been demonstrated to fail the direct sim-to-real test for the same class of motions [2].
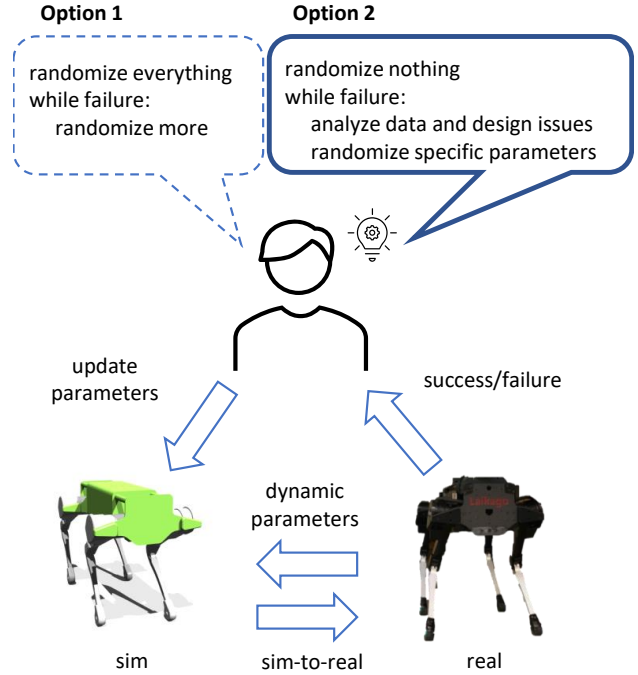
Fig. 1: Dynamics randomization is often used in an ad hoc fashion, while we advocate for identifying and randomizing parameters that matter and identifying other possibly problematic control policy design issues.

2) We identify and analyze particular design choices for which dynamics randomization is *not sufficient* to enable sim-to-real success.

3) We evaluate the consequences of dynamics randomization in our setting. Specifically, we find that unnecessary randomization of parameters can produce conservative policies with limited actual gains in robustness, while randomization in truly problematic parameters bridges the reality gap.

Empirical evidence in this paper illustrates that Domain Randomization is *neither necessary nor sufficient*. We instead advocate for conservative application of dynamics randomization, i.e., such computationally intensive techniques should be used when there is a clearly identified need in sim-to-sim tests. Further, randomization should only be on parameters that matter, which requires domain insight. Fig. 1 illustrates the naive approach and the advocated approach.

II. RELATED WORK

A. Quadrupedal Locomotion

Model-based approaches such as trajectory optimization and model predictive control are commonly used to achieve

agile locomotion skills on quadrupeds, typically making use of a low-dimensional model, e.g., [9]–[11]. Deep RL has also been used to train quadrupedal controllers in simulation and transfer to a physical robot, e.g., [1], [2], [4], [12].

B. Sim-to-real for Robot Control

The reality gap often prevents the success of direct sim-to-real transfer. Dynamics randomization describes the randomization of parameters such as masses and inertial moments of robot links, as well as parameters that govern control latency, actuator response, etc. Such parameters are randomized during training in order to obtain policies that are robust to modeling errors, as first proposed to solve robotic arm pushing tasks [13] and later used for various sim-to-real studies involving manipulation [14] and locomotion [1], [4], [15]. This technique complements the collection of on-robot data for online adaptation [2], [3] or system identification in the form of improved simulator accuracy [16], [17]. At the same time, multiple results also demonstrate successful sim-to-real transfer without dynamics randomization via appropriate design choices, e.g., [6]–[8], [18], [19].

C. Robust Control

Control policies obtained with a single model are often susceptible to modeling errors or noise, even for a simple linear quadratic regulator [20]. Robust policies can be obtained via model ensembling, where a distribution of models is used for control synthesis, e.g., [21], [22]. Perturbations can also be introduced during optimization to obtain robust behaviors, e.g., [23]–[25]. Further two stage models achieve robustness through reference tracking of a state trajectory obtained from an open-loop policy rollout with feedback control [26].

In this work, we employ the Laikago robot, which has been used for prior sim-to-real work [2], [8], and we explore in detail the role of dynamics randomization and the importance of appropriate design choices.

III. CONTROL POLICIES

We first describe the structure of our control policies together with the training methods. We denote the state of the robot, x, as the collection of tuples $x = [p \in \mathbb{R}^3, o \in S^3, j \in \mathbb{R}^{12}, \dot{p}, \dot{o}, \dot{j}]$, where p and o are the position and orientation of the base of the robot, and j is the set of twelve joint angles. An overview of our system setup in simulation and on the physical robot is shown in Fig. 2. In this section, we describe each component in detail.

A. Training Environment

We develop policies that can produce gaits that are typical for model-based control of quadrupedal robots. We follow an approach similar to prior work [2] and incorporate reference trajectories into our framework. Given a reference trajectory $\chi = \{\hat{x}_0, \hat{x}_1, \cdots\}$, where $\hat{x}_t$ is the desired state of the robot at time step t, we define the reward r_t as $r_t = 0.4r_t^j + 0.3r_t^p + 0.3r_t^o$, where

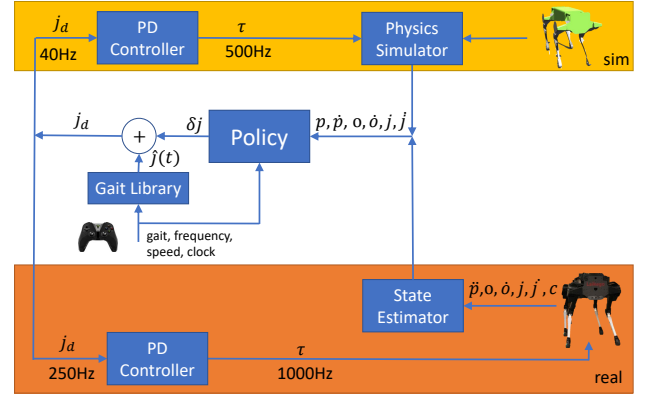$$r_t^j = \exp(-2 \left\| j_t - \hat{j}_t \right\|^2),$$

Fig. 2: Overview of our system. The input to the policy includes state of the robot and user commands. The output is a residual PD target, which is added to a reference target and applied to a joint PD controller. Various motions are achieved by using a library of gaits as reference trajectories.

$$r_t^p = \exp(- \left\| p_t - \hat{p}_t \right\|^2),$$

$$r_t^o = \exp(-2 \left\| o_t - \hat{o}_t \right\|^2 - 5 \left\| \dot{o}_t - \hat{\dot{o}}_t \right\|^2).$$

The goal of RL is to find a policy that will generate action a_t at each time step t, such that the cumulative reward $\sum_{t=0}^{\infty} \gamma^t r_t$ is maximized.

As is common practice for RL with legged robots, the input to the policy includes the current state of the robot but excludes the x, y coordinates, since their measurement tends to drift for the physical robot without exteroceptive sensors. To achieve a cyclic locomotion gait, we use a reference motion that is cyclic with a period T. To inform the policy about the time-varying nature of the reward, we also provide inputs $\cos(2\pi t/T)$ and $\sin(2\pi t/T)$ to the policy. Finally, we include the gait type (encoded as an integer index), the gait frequency, and the desired speed as inputs. These are controlled by a human operator to switch between different motions. Following [27], the output of the policy is a residual PD joint target δ_j, and the corrected joint position $j_d = \delta_j + \hat{j}$ is then used as the input to a PD controller that generates torques τ on the robot via $\tau = k_p(j_d - j) - k_d \dot{j}$. We use $k_p = 40$ and $k_d = 0.5$ as default gains. During training in simulation, the policy updates its output every 26 ms while the PD controller updates the commanded torque every 2 ms.

B. Gaits

We achieve multiple gaits via gait-specific reference trajectories, which provide a sketch of the desired gait phases. Different gaits are characterized by their contact sequences. The motion is divided into phases, with each phase having designated legs in either swing or stance roles. For the stance legs, the reference joint angles are fixed to $[0, 0.65, -1]$ for hip abduction, hip pitch, and knee angles, respectively (in radians). For the swing legs, the reference angles are $[0, 0.65 - 0.4\hat{v}_x \sin(\pi\rho/T_p), -1 + 0.7 \sin(\pi\rho/T_p)]$, where ρ is the time elapsed during the current phase, T_p is the duration of the phase, and $\hat{v}_x$ is the desired speed in the forward direction that the policy should achieve. The swing

trajectories are empirically designed so that the swing foot has high clearance while also going forward or backward to adapt to the desired velocity.

We adopt three common locomotion gaits for quadrupedal robots: (1) *Walking*, where each individual leg moves in turn. (2) *Trotting*, where diagonal legs share a common swing phase; (3) *Pacing*, where front-and-hind legs on a given side of the body share a common swing phase. We further increase the variety of trajectories by varying desired speed $\hat{v}_x$ and phase duration T_p to obtain a library of gaits that move at different speeds and stepping frequencies. While we demonstrate successful sim-to-real transfer with all three gaits, we focus our evaluation on the *Trotting* and *Pacing*.

C. Training Setup

We train our policies with an actor-critic algorithm using proximal policy optimization [28]. The policy is represented as a two-layer feedforward neural network with a hidden layer size of 128. During training, the actions follow a Gaussian distribution with mean given by the neural network output and a fixed standard deviation of $\exp(-1.5)$. During testing, we use the deterministic version of the policy, as given by the mean.

We use Isaac Gym during training, which is supported by a GPU-accelerated simulator [29]. This simulator has been validated to simulate rigid body dynamics with reasonable accuracy for locomotion [8] and manipulation [30]. We simulate 1600 robots in parallel and collect 2.4×10^5 environment tuples at each training iteration. We train each policy for a maximum of 10^3 iterations, with 2.4×10^8 samples in total, less than or comparable to the values used for related sim-to-real work for similar scale quadrupeds, e.g., [2], [4], [12]. The training time for one policy on a single GeForce RTX 2080 Ti GPU is 6 to 8 hours.

D. Physical Robot Setup

While the body position and velocity are available for the robot in simulation, we can only estimate these quantities for the physical robot via its onboard sensors, which consist of an IMU, joint encoders, and a one-dimensional force sensor on each foot for contact detection. We follow prior work [31] and build a Kalman filter for the purpose of state estimation.

The estimated body velocity can suffer from bias due to integrated accelerometer error. This results in the robot drifting in the plane while being commanded to step in place. This can be addressed by adding an artificial offset to the estimates online to counter the drift. We also add an offset to the yaw angle to compensate for initialization error and drift. These offsets can be also used as a command signal for moving sideways or turning, even though the policy has never explicitly been trained for these motions. For example, if we add a positive offset to the lateral velocity, the robot will move in the negative direction with speed matching the offset in order to make the overall lateral velocity observation zero. These aspects of control arise implicitly from encountering similar states during training due to the stochastic policy.

The target joint angles from the policy are updated at a slow rate, every 26 ms, during training, which improves learning efficiency. In the physical robot experiment, however, the slow update limits the control bandwidth of the PD controller and introduces a discontinuity that can harm the motors due to the large torque change. We mitigate this issue by updating the target joint angles every 4 ms on the physical robot. This is possible since the policy query time typically takes around 2 to 3 ms. To further smooth the target joint angles, we pass them through a discrete first-order low-pass filter before providing them to the PD controller. This produces smoother movements and therefore also helps realize improved state estimates from the robot. The low-pass filter $j_d = (1 - \lambda)j_{d,prev} + \lambda j_{d,policy}$ averages the previous target joint angles $j_{d,prev}$ and the current target joint angles $j_{d,policy}$ with a filter constant $\lambda \in [0, 1]$. We use a weak filter with $\lambda = 0.2$ and a cutoff frequency of 62.5 Hz. A similar smoothing procedure was also used in [2].

IV. DYNAMIC RANDOMIZATION IS NOT NECESSARY

The reality gap often causes direct policy transfer to fail. While dynamics randomization is frequently used to address this issue, the randomization is often applied in an *ad hoc* fashion, without a deeper understanding of other possible sources of failure, of the need (or possible lack thereof) for randomization, or of the impact of randomization choices, including unintended consequences. In this section, we demonstrate that dynamics randomization is not necessary for our learned controllers. We first show in a simulation that the default policy trained without dynamics randomization can already cope with modeling errors that are larger than the randomization variations used in prior sim-to-real work [2]. We also then run these policies on the physical robot and demonstrate successful sim-to-real transfer.

A. Types of Perturbation

We first describe the types of perturbation we use for testing the robustness of the policies, both in simulation and on the physical robot.

a) Mass Perturbation: One source of modeling error is given by the mass and moments of inertia of the body parts. These are also the most commonly used parameters for dynamics randomization. During robustness tests in simulation, we directly change the mass of the main body and record the maximum mass value we can add to the default value. During testing on the real robot, we put a box of bricks on top of the robot and record the maximum payload the robot can carry without falling.

b) Proportional Gain: The default proportional gain for the PD controller is $k_p = 40$ across all motors during training. During testing in simulation, we decrease the value of k_p and record the minimum value the policy can cope with. This is a proxy for possible friction in the motors as well as understanding the sensitivity of the learned policy to PD controller parameters. The same procedure is used for physical robot testing.

c) Latency: Another commonly randomized parameter is the latency in the system. We simulate the latency by implementing a first-in-first-out buffer, where the length of the buffer corresponds to the introduced latency. During testing, both in simulation and on the physical robot, we record the maximum latency the policy can cope with.

d) Lateral Push: In the simulation, we apply a constant lateral push for 5 seconds and record the maximum push the policy can recover from. We only perform qualitative tests on the physical robot since we do not currently have the hardware needed to apply a prescribed force or impulse.

e) Slope: We command the robot to walk up or down a slope and record the maximum slope the policy can walk on without falling, both in simulation and on the physical robot.

B. Robustness Tests in Simulation

We train policies that can perform trotting and pacing at various speeds without dynamics randomization. We deploy the policies in increasingly challenging scenarios for each of the five varieties of perturbation described in the previous subsection, recording the maximum perturbation (e.g., the largest slope angle) at which the robot can still complete the task. We repeat the experiment by training a total of three policies for each of the two gaits, computing the mean and sample standard deviation of the threshold values over the three trials. These results are shown in TABLE I. The policies can already cope with a larger range of perturbations as compared to ranges typically used in related sim-to-real literature for similar robots [2], [4]. For example, the robot can carry mass that is up twice the default mass value of the body while 20 percent randomization is typically used. These ranges also cover the possible modeling error. For example, the default value of the mass of the body is measured by the manufacturer and typically has less than a 5 percent error. The latency in the system is mainly caused by the policy query and is typically less than 4 ms, significantly smaller than what the policy can handle in simulation.

C. Tests on the Physical Robot

We directly apply these policies on the physical robot and observe sim-to-real success without adaptation. Fig. 3 shows snapshots of the physical robot tests. This is consistent with our observation in the simulation where the policies are capable of coping with large modeling errors.

We perform similar robustness tests on the physical robot for the trotting and pacing gaits. Due to concern over potential damage to the hardware, we only test one of the policies for each gait. The results are recorded in TABLE I. For the trotting gait, the reality gap is smaller as the robustness test results are similar between simulation and the physical robot, while for the pacing gait, we observe that the policy typically achieves worse performance on the physical robot compared to the simulation robustness test. However, the policies are nevertheless already robust against a large range of perturbations, and in both cases, the reality gap is easily overcome. As observed here, the reality gap is also dependent
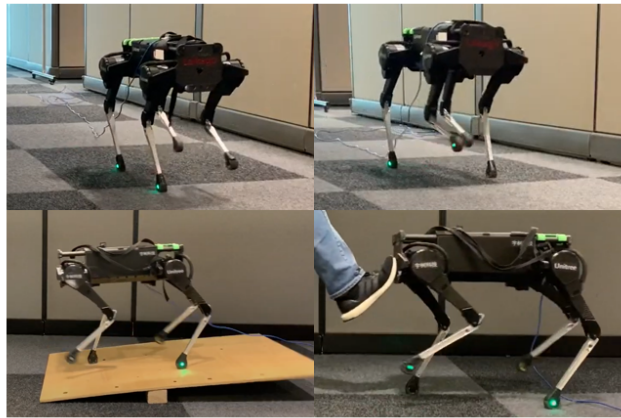

Fig. 3: We train policies for multiple gaits that transfer to the real robot without dynamics randomization. **TOP**: Trotting and pacing with learned policies. **BOTTOM**: The policies can walk up or down a slope and react to forceful pushes despite never encountering these scenarios during training.

on the type of motions considered and thus the results may vary for more dynamic motions such as gallops or jumps.

V. DYNAMICS RANDOMIZATION IS NOT SUFFICIENT

In the previous section we demonstrated that dynamics randomization is, in the right circumstances, not necessary for direct sim-to-real transfer. This contrasts with previous work [2] despite being similar in the use of reference trajectories, being evaluated on similar classes of motions (trotting and pacing), and on the same robot model. We observe several design decision discrepancies and conduct an ablation study on these design decisions. We thus provide an explanation for the apparent contradiction and show that with alternate design choices, control policies will fail to transfer and that dynamics randomization by itself, without further on-robot adaptation, is not sufficient to cross the reality gap in these scenarios. This is consistent with other sim-to-real results, where an additional adaptation step on the physical robot is needed [2], [3].

A. Design Choices

a) Choice of Observation: In [2], the state observations consists of raw sensory measurements such as the orientation of the body and joint angles, while we rely on a state estimator and include additional information such as body velocity and joint velocity. To understand the impact of this design choice, we train policies without the body velocity estimate provided by the state estimator, to create a more closely matched system. While some velocity information is implicit in the proprioceptive state, we hypothesize that a good estimate of the body velocity can help prevent drift and instability in the lateral direction.

b) Choice of Proportional Gain: In [2], the authors use stiff proportional gains ($k_p = 220$) for the PD controllers on the motors, while we use a soft gain ($k_p = 40$). We conduct experiments to evaluate the impact of this design choice.

Policy	Mass (kg)	P gain (Nm/rad)	Latency (ms)	Lateral Push (N)	Slope Up (degrees)	Slope Down (degrees)
Trot: Simulation	9 ± 3	27 ± 2	17 ± 1	50 ± 7	11 ± 1	6 ± 0
Trot: Real	10	27	16	——	4	6
Pace: Simulation	20 ± 3	23 ± 1	17 ± 1	43 ± 2	13 ± 1	11 ± 0
Pace: Real	8	30	16	——	4	6

TABLE I: Robustness tests of different controllers in simulation and in experiments with the real robot.

Parameter	Range
Mass	$[0.8, 1.2] \times$ default
Inertia	$[0.5, 1.5] \times$ default
P gain	$[-20 \, \text{Nm/rad}, 20 \, \text{Nm/rad}] +$ default
Latency	$[0 \, \text{ms}, 20 \, \text{ms}]$

TABLE II: Parameters and the corresponding ranges used for dynamics randomization.

B. Simulation Test

We train four different policies to explore the impact of the described design choices. First, we train pacing policies without velocity feedback, with and without dynamics randomization. Second, we train trotting policies with a high gain ($k_p = 160$), with and without dynamics randomization. The type and range of randomization can be seen in Table II.

For the high-PD-gain trot, we use $k_p = 160$ instead of $k_p = 220$ as used in [2] because the higher gain can cause instability on the physical robot. Additional smoothing procedures as used in [2] might be needed to use the higher gain. We also purposely choose not to apply randomization to lateral pushes or slopes during training. Instead, we wish to understand cross-correlations in robustness. Specifically, if we train for additional robustness along some dimensions or parameters, via dynamics randomization, will that positively or negatively influence the robustness along the other unrandomized dimensions?

We train three different policies under each setting and perform robustness tests similar to the previous section; the results are shown in TABLE III. We observe that dynamics randomization does not help the observed robustness in most of the robustness tests, including in the randomized dimensions. Thus, rather surprisingly, randomization does not generally help robustness in our experiments. The one exception is latency randomization, which was also significantly randomized in [2]. Overall, this points to much randomization being unnecessary or even harmful.

C. Robot Test

We also verify that the policies fail to transfer to the physical robot with the alternative design choices, even with dynamics randomization. This is consistent with prior results [2], where additional adaptation procedures were employed on the physical robot to cross the reality gap.

The pacing policies without velocity feedback cannot control the lateral velocity and often fall sideways. In an additional experiment, we train pacing policies with lateral pushes applied during training. We find, however, that this cannot compensate for the lack of lateral velocity feedback, and the policies fail in a similar way.
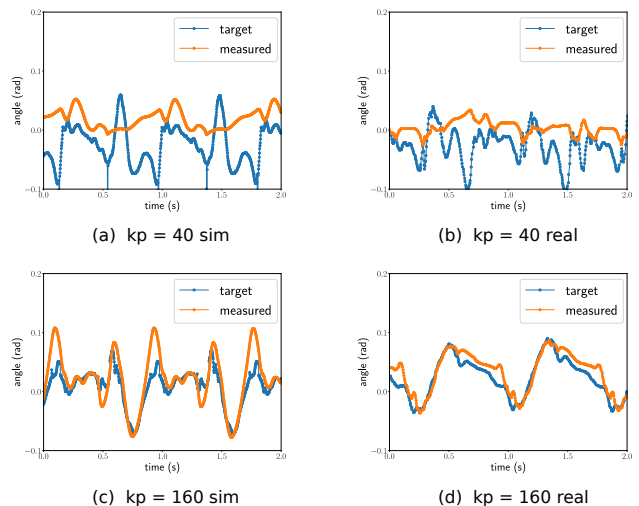

(a) $k_p = 40$ sim

(b) $k_p = 40$ real

(c) $k_p = 160$ sim

(d) $k_p = 160$ real

Fig. 4: Comparison of joint angle tracking with different proportional gains. **TOP**: With $k_p = 40$, the tracking error is large, and the policy behaves like a torque controller. The reality gap is relatively small. **BOTTOM**: With $k_p = 160$, the tracking error is much smaller, and the policy behaves like a position controller. The reality gap is visibly larger.

The trotting policies trained with $k_p = 160$ exhibit very stiff motion compared to those trained with $k_p = 40$, and the motion is sensitive to contact timing. The robot occasionally kicks its feet backward when the policy expects them to be in the stance phase while the foot has yet to make contact with the ground. This is typically a disadvantage of position control where unexpected contact modes can be problematic.

D. Comparison of Proportional Gain

Inspired by a comment in [4], we hypothesize that low proportional gains will force the policy to behave like a torque controller while high proportional gains will behave like a position controller. Related work shows that a torque controller is typically more robust and flexible than a position controller for unanticipated impacts [32]. In Fig. 4, we plot the joint trajectory and the target joint trajectory of one of the hip joints of the robot and observe that with $k_p = 40$, the policy generates motions with large tracking errors and thus behaves like a torque controller, while with $k_p = 160$, the policy generates motions with low tracking errors and behaves like a position controller.

VI. EFFECT OF DYNAMICS RANDOMIZATION

We have observed that dynamics randomization is neither necessary in our setting, nor sufficient in the face of other problematic design choices. However, different conclusions might be drawn for different robots or different motions. In

Policy	Mass (kg)	P gain (Nm/rad)	Latency (ms)	Lateral Push (N)	Slope Up (degrees)	Slope Down (degrees)	Sim-to-Real Outcome
Pace: Default	20 ± 3	23 ± 1	17 ± 1	43 ± 2	13 ± 1	11 ± 0	success
Pace: No vel	18 ± 0	24 ± 2	12 ± 2	22 ± 0	4 ± 0	7 ± 0	failure
Pace: No vel, with rand	9 ± 4	30 ± 2	38 ± 2	13 ± 5	4 ± 1	9 ± 1	failure
Trot: Default	9 ± 3	27 ± 2	17 ± 1	50 ± 7	11 ± 1	6 ± 0	success
Trot: $k_p = 160$	18 ± 12	$--$	17 ± 4	18 ± 6	10 ± 0	5 ± 3	failure
Trot: $k_p = 160$, with rand	8 ± 1	$--$	41 ± 1	12 ± 8	12 ± 1	1 ± 0	failure

TABLE III: Robustness tests for different policies in simulation, together with the result of attempted sim-to-real transfer. Blue indicates policies that perform similarly to the corresponding policy with default settings. Green and red indicate policies that perform better or worse than the default, respectively. Policies without velocity feedback or with $k_p = 160$ all fail the sim-to-real tests, with or without randomization. They also generally perform worse in the robustness tests compared to policies trained with the default setup.

this section, we explore the advantages and disadvantages of dynamics randomization in greater depth.

A. Dynamics Randomization Produces Conservative Policies

We observe in TABLE III that dynamics randomization can sometimes lead to policies that are overly-conservative in order to achieve unnecessary robustness in parameters that are being randomized. For example, the pacing policies trained with no velocity feedback and dynamics randomization perform worse than policies trained without dynamics randomization in general, except in terms of dealing with latency. However, the physical robot system has an estimated latency of less than 4 ms, and this unnecessary robustness against increased latency leads to compromised performance and robustness along other dimensions.

We further train trotting policies under the default setting with dynamics randomization. We observe a more conservative maximum speed (0.9 m/s with randomization and 1.1 m/s with no randomization), both in simulation and on the physical robot. This also corresponds to our intuition that dynamics randomization can produce conservative policies.

B. Randomize Parameters that Matter

We use the latency test to investigate the usefulness of dynamics randomization. We observe that policies trained without randomization fail when the latency exceeds 17 ms. We train another policy with randomized latency only; more specifically, the policy is trained with randomized latency of up to 20 ms. The resulting policy can handle latency up to 32 ms, both in simulation and on the physical robot.

This indicates that dynamics randomization can help in scenarios where significant modeling errors are present, such as latency in the system. In these scenarios, dynamics randomization provides a useful mechanism to cross the reality gap by only randomizing the parameters that are responsible for the sim-to-real issues.

C. Summary

We observe that blindly applying dynamics randomization when it is not necessary can generate suboptimal policies that are too conservative. However, if the system has fundamental modeling errors that hinder sim-to-real success, randomization is needed to cross the reality gap, as shown in our latency experiments. We note that actuator modeling errors can also pose a sim-to-real challenge, as noted in [4], where a learned actuator model is employed to cross the reality gap.

In summary, we suggest employing dynamic randomization or additional modeling only when significant modeling errors are present and to only randomize or model parameters that matter. Superfluous dynamics randomization harms performance in measurable ways while possibly giving no extra benefit in robustness, even for the randomized dimensions.

VII. DISCUSSION AND FUTURE WORK

In this paper, we have studied a number of factors that affect sim-to-real transfer for quadrupedal locomotion, and found that commonly-used dynamics randomization often offers negligible actual improvements in robustness. We have also evaluated design choices related to proportional gain stiffness and state observation parameters that may have been overlooked in prior work, despite their critical role in the success of sim-to-real transfer.

While many of these observations should generalize beyond our particular settings, additional modeling or randomization may be necessary for other tasks or robot morphologies. For example, due to different actuation, the ANYmal robot has been found to require additional actuator modeling [4], and significant latency in the control loop of the Ghost Minitaur robot has been found to require randomization of latency [1]. We advocate identifying important, i.e., high-sensitivity, sim-to-real bottlenecks using simulations and performing necessary additional modeling or randomization only for the relevant parameters instead of arbitrarily adding randomization to a larger set of parameters, as has sometimes been done in the past.

It is likely that the reality gap will be more pronounced for more dynamic motions such as running or jumping. We plan to learn more agile locomotion skills for the Laikago robot and to continue to improve our understanding of sim-to-real challenges that might be present in such settings. A related avenue of investigation has to do with the problem of identifying the key design choices necessary for sim-to-real success. In this work, we tuned certain parameters (such as proportional gains) empirically, while recognizing that such design considerations can vary from robot to robot. It should be possible to bring learning to bear on this problem.

REFERENCES

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

[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] W. Yu, V. C. Kumar, G. Turk, and C. K. Liu, "Sim-to-real transfer for biped locomotion," *arXiv preprint arXiv:1903.01390*, 2019.

[4] 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.

[5] J. Lee, J. Hwangbo, L. Wellhausen, V. Koltun, and M. Hutter, "Learning quadrupedal locomotion over challenging terrain," *Science Robotics*, vol. 5, no. 47, 2020. [Online]. Available: https://robotics.sciencemag.org/content/5/47/eabc5986

[6] Z. Xie, P. Clary, J. Dao, P. Morais, J. Hurst, and M. van de Panne, "Learning locomotion skills for cassie: Iterative design and sim-to-real," in *Proc. Conference on Robot Learning (CORL 2019)*, 2019.

[7] J. Dao, H. Duan, K. Green, J. Hurst, and A. Fern, "Learning to walk without dynamics randomization," *2nd Workshop on Closing the Reality Gap in Sim2Real Transfer for Robotics*, 2020.

[8] X. Da, Z. Xie, D. Hoeller, B. Boots, A. Anandkumar, Y. Zhu, B. Babich, and A. Garg, "Learning a contact-adaptive controller for robust, efficient legged locomotion," *arXiv preprint arXiv:2009.10019*, 2020.

[9] J. Di Carlo, P. M. Wensing, B. Katz, G. Bledt, and S. Kim, "Dynamic locomotion in the mit cheetah 3 through convex model-predictive control," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2018, pp. 1–9.

[10] C. Gehring, S. Coros, M. Hutter, M. Bloesch, M. A. Hoepflinger, and R. Y. Siegwart, "Control of dynamic gaits for a quadrupedal robot," in *IEEE International Conference on Robotics and Automation (ICRA), 2013: 6-10 May 2013, Karlsruhe, Germany*. IEEE, 2013, pp. 3287–3292.

[11] C. Gonzalez, V. Barasuol, M. Frigerio, R. Featherstone, D. G. Caldwell, and C. Semini, "Line walking and balancing for legged robots with point feet," *arXiv preprint arXiv:2007.01087*, 2020.

[12] S. Gangapurwala, A. Mitchell, and I. Havoutis, "Guided constrained policy optimization for dynamic quadrupedal robot locomotion," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 3642–3649, 2020.

[13] 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.

[14] 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.

[15] J. Siekmann, S. Valluri, J. Dao, L. Bermillo, H. Duan, A. Fern, and J. Hurst, "Learning memory-based control for human-scale bipedal locomotion," *arXiv preprint arXiv:2006.02402*, 2020.

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

[17] H. Karnan, S. Desai, J. P. Hanna, G. Warnell, and P. Stone, "Reinforced grounded action transformation for sim-to-real transfer," *arXiv preprint arXiv:2008.01279*, 2020.

[18] B. Thananjeyan, A. Garg, S. Krishnan, C. Chen, L. Miller, and K. Goldberg, "Multilateral surgical pattern cutting in 2d orthotropic gauze with deep reinforcement learning policies for tensioning," in *IEEE International Conference on Robotics and Automation (ICRA)*, jun 2017.

[19] M. Kaspar, J. D. M. Osorio, and J. Bock, "Sim2real transfer for reinforcement learning without dynamics randomization," *arXiv preprint arXiv:2002.11635*, 2020.

[20] C. G. Atkeson, "Efficient robust policy optimization," in *2012 American Control Conference (ACC)*. IEEE, 2012, pp. 5220–5227.

[21] M. McNaughton, "Castro: robust nonlinear trajectory optimization using multiple models," in *2007 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2007, pp. 177–182.

[22] I. Mordatch, K. Lowrey, and E. Todorov, "Ensemble-cio: Full-body dynamic motion planning that transfers to physical humanoids," in *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2015, pp. 5307–5314.

[23] H. Dai and R. Tedrake, "Optimizing robust limit cycles for legged locomotion on unknown terrain," in *2012 IEEE 51st IEEE Conference on Decision and Control (CDC)*. IEEE, 2012, pp. 1207–1213.

[24] T. Li, H. Geyer, C. G. Atkeson, and A. Rai, "Using deep reinforcement learning to learn high-level policies on the atrias biped," in *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, 2019, pp. 263–269.

[25] A. Mandlekar, Y. Zhu, A. Garg, L. Fei-Fei, and S. Savarese, "Adversarially robust policy learning: Active construction of physically-plausible perturbations," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2017, pp. 3932–3939.

[26] J. Harrison*, A. Garg*, B. Ivanovic, Y. Zhu, S. Savarese, L. Fei-Fei, and M. Pavone (* equal contribution), "AdaPT: Zero-Shot Adaptive Policy Transfer for Stochastic Dynamical Systems," in *International Symposium on Robotics Research (ISRR)*. Springer STAR, dec 2017.

[27] Z. Xie, G. Berseth, P. Clary, J. Hurst, and M. van de Panne, "Feedback control for cassie with deep reinforcement learning," in *Proc. IEEE/RSJ Intl Conf on Intelligent Robots and Systems (IROS 2018)*, 2018.

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

[29] NVIDIA, *Isaac Gym - Preview Release*, 2020. [Online]. Available: https://developer.nvidia.com/isaac-gym

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

[31] G. Bledt, M. J. Powell, B. Katz, J. Di Carlo, P. M. Wensing, and S. Kim, "Mit cheetah 3: Design and control of a robust, dynamic quadruped robot," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2018, pp. 2245–2252.

[32] M. Focchi, T. Boaventura, C. Semini, M. Frigerio, J. Buchli, and D. G. Caldwell, "Torque-control based compliant actuation of a quadruped robot," in *2012 12th IEEE international workshop on advanced motion control (AMC)*. IEEE, 2012, pp. 1–6.