CS230

Learning Distributions of Dynamics to Improve Sim-to-Real Transfer

Gabrael T. Levine*
Department of Computer Science
Stanford University
gabrael@stanford.edu

Abstract

This work evaluates a method of domain randomization and system identification using a neural network trained to mimic the distribution of dynamics seen in a specific type of robotic actuator. This model is subsequently used in a physics simulator to train a reinforcement learning policy. Finally, the policy is deployed on a real robot. Results indicate that the learned actuator model has poor fidelity. Nonetheless, it markedly improves sim-to-real transfer performance of the policy.

1 Introduction

Stanford Pupper is an inexpensive open-source quadruped robot designed for research and education [1]. An exciting potential use for such a robot is in reinforcement learning research, as a real-world platform to test locomotion policies trained in simulation.

However, deploying a policy trained in simulation on this robot presents some challenges. The low-cost construction of the robot results in significant mismatch between simulation and reality (flexibility of the 3D-printed plastic parts, backlash in the gear reducers) and uncertainty (changing physical parameters due to physical wear on the robot, sensor noise), leading to degraded performance of the policy.

In particular, the actuator model in the physics simulator is a poor representation of the real actuator dynamics. Therefore, the focus of this work will be on learning the actuator model.

Given a history of the position error (commanded position - actual position) and velocity over some number of discrete timesteps, the actuator model outputs the torque to be applied at the present timestep. This procedure is executed every timestep inside the physics simulator.

To evaluate the effect of the learned actuator model on sim-to-real performance, two stages of learning are required: first, training the actuator model using supervised learning; second, training a reinforcement learning policy inside the physics simulator using the learned actuator model.

2 Related work

Two common ways to improve sim-to-real performance are domain randomization and system identification. Domain randomization varies parameters in the physics simulator in an attempt to make the policy more robust to different dynamics. System identification attempts to make the simulator better match reality using data gathered on the real robot.

*Use footnote for providing further information about author (webpage, alternative address)—*not* for acknowledging funding agencies.

CS230: Deep Learning, Fall 2021, Stanford University, CA. (LateX template borrowed from NIPS 2017.)

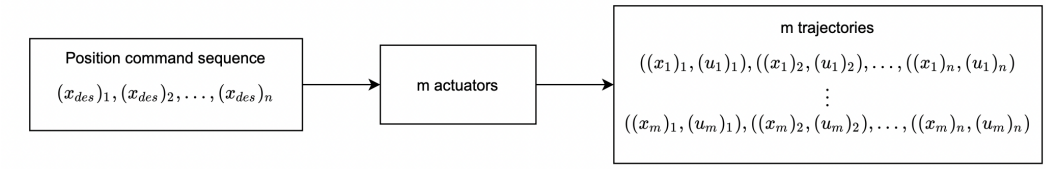

Figure 1: Data collection process

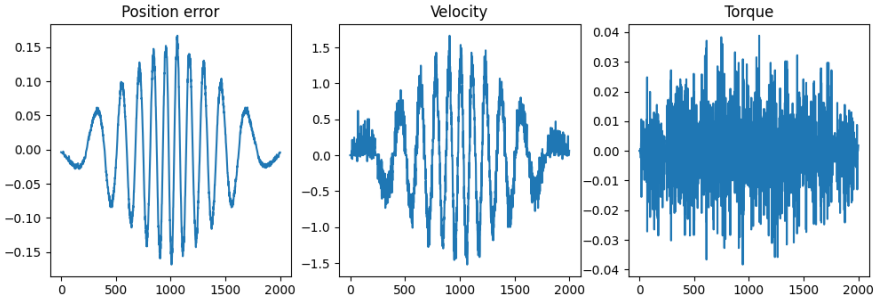

Figure 2: Real data

Both approaches have drawbacks; domain randomization doesn't necessarily capture important effects such as latency, while system identification requires laborious data collection on each robot on which an RL policy is to be deployed.

The method proposed in this work is similar to the ActuatorNet described in [2], but rather than learning the dynamics of a particular actuator, it learns a distribution of dynamics using data from many actuators. This distribution of actuator dynamics can then be randomly sampled during RL training to produce domain randomization. This method aims to achieve the goals of both system identification and domain randomization without suffering from the aforementioned drawbacks.

3 Dataset and Features

For the supervised learning step, the dataset consists of sequences of states (desired/actual position/velocity) and torque for several actuators. To produce this data, a sequence of position commands (excitation signal) is sent to the actuator. In this case, the excitation signal was a sinusoid with varying frequency and amplitude. The intent is to capture the nonlinear response of the actuators in a large region of state-space.

At first, it was planned for all of this data to be collected on several dozen real actuators using a dynamometer. However, due to time constraints, the dataset was instead synthetically generated. A small amount of real data was collected and used to hand-tune the synthetic data generation process to produce realistic results. Data was generated for 32 actuators over a length of 10000 timesteps, with a timestep length of 0.01s. The data generation process is illustrated in figure 1.

For the reinforcement learning step, the dataset consists of states, actions, and rewards collected in the PyBullet physics simulator during rollouts of the RL policy.

4 Methods

The goal of the supervised learning stage is for the actuator model to learn a distribution of dynamics. Therefore, there must be a way to sample from this distribution. This is accomplished by conditioning the neural network on a randomly sampled latent vector z. Training is performed in such a way that z encodes the information about an actuator that stays the same between timesteps, such as its inertia.

The loss function is defined identically to the variational autoencoder (VAE), as a weighted sum of the prediction and a KL-divergence term that encourages the values of z for the training set to be

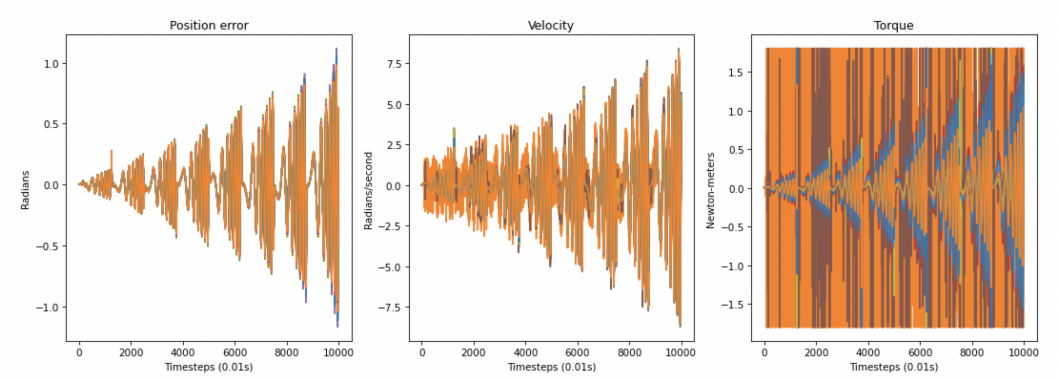

Figure 3: Synthetic data

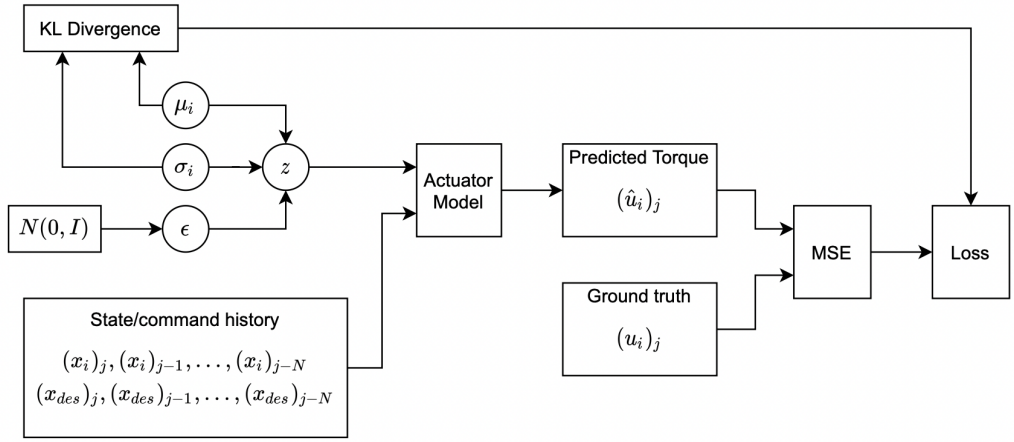

Figure 4: Supervised learning stage (actuator model training)

close to a unit normal distribution. During training, an embedding stores the mean μ_i and variance σ_i^2 corresponding to each individual motor i in the training set.

$$z_i = \mu_i + \sigma_i^2 \epsilon, \epsilon \sim N(0, I) \tag{1}$$

Similar to training a VAE, the reparameterization trick is used to backpropagate through the random sampling process. Then, during inference, z is directly sampled from a unit normal distribution to generate new dynamics behaviors from the learned distribution.

In summary, the learned actuator model takes a state history x and latent vector z, outputting the torque u. This is illustrated in figure 4.

For the reinforcement learning stage, a pre-existing software pipeline [3] was used to train a policy for a simple locomotion task (walking on flat ground) and deploy it to the real robot. More specifically, the training algorithm used is Augmented Random Search [4], and the policy is an implementation of Policies Modulating Trajectory Generators [5]. The simulator was modified to incorporate the learned actuator model. The simulator's default actuator model was used as the baseline for comparison. This default model is a simple proportional-derivative (PD) controller, which matches the behavior of ideal actuators with no latency, reflected inertia, nonlinear damping, or backlash.

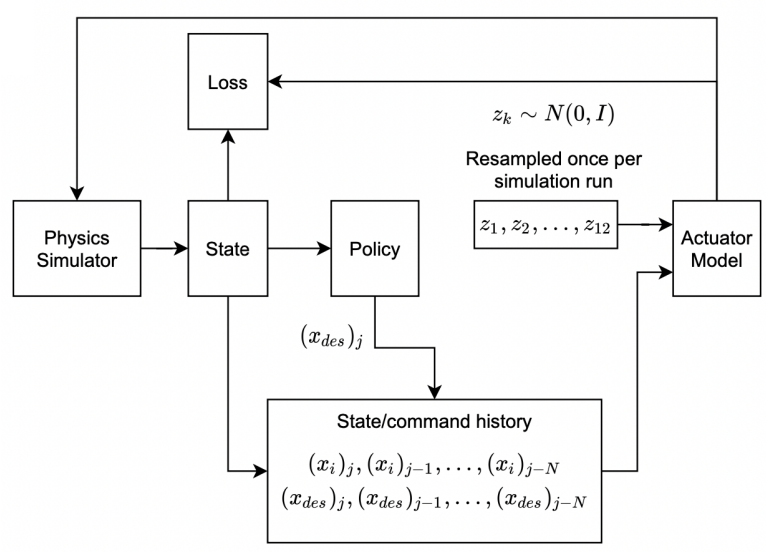

Figure 5: Reinforcement learning (locomotion policy training)

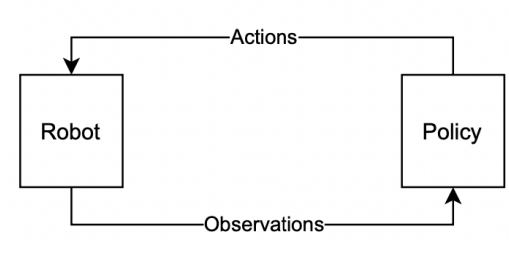

Figure 6: Locomotion policy deployment

5 Experiments/Results/Discussion

In simulation, the learned actuator model behaved significantly worse than the baseline with no learned actuator model. The learned actuator model caused oscillations of the robot's legs, making it struggle to walk, while the baseline resulted in a normal-looking trotting gait.

Interestingly, the opposite result was observed when the policies were run on the physical robot. When deployed on the real robot, the RL policy trained using the learned actuator model took lower-frequency steps and lifted the legs higher than the policy trained with the baseline physics simulator. This potentially indicates that the actuator model learned to imitate the limited bandwidth of the real actuators. The baseline policy was only able to wiggle its legs in place, which did not result in successful locomotion.

When taken together, these results indicate that the learned actuator model had a regularizing effect on the RL training, as it decreased performance on the training set (the simulator) while increasing performance on the test set (the real robot). This supports the hypothesis that sampling from a learned dynamics distribution is a useful method of domain randomization.

6 Conclusion/Future Work

The method presented in this work succeeded in improving sim-to-real transfer on the Pupper quadruped, enabling the robot to locomote. However, the deployed policy still adopts an inefficient gait that compares poorly to its performance in simulation. Thus, much work remains to truly make Pupper a useful platform for reinforcement learning research.

The main issue to be solved is the distributional shift between the training data (sinusoidal trajectories of single motors) and the load conditions on the real robot (for example, frequent impacts and sharp accelerations). This is the most likely cause of the poor fidelity of the learned actuator model, since when used in the physics simulator, it is given input data from a significantly different distribution from what it was trained on.

The most obvious place for future work is to replace synthetic data with real actuator data. However, this is unlikely to significantly improve the fidelity of the actuator model, since it doesn't solve the distributional shift.

Another idea is to try this method on a robot with torque sensing, such as the Anymal quadruped in [2]. If data can be collected directly on the robot, this would eliminate the distributional shift entirely.

Finally, another possibility is to use a neural network to generate an excitation signal similar to real robot data, which is then used during data collection for the actuator model. A generative adversarial network could be a good choice for this.

7 Contributions

Stanford Pupper version 2.1, the robot used in this work, was designed by Nathan Kau and myself in Stanford Student Robotics.

The pre-existing Pupper simulator and RL training/deployment pipeline was developed by researchers at Google Brain.

RL troubleshooting advice was received from Google Brain researcher Jie Tan.

References

[1] https://stanfordstudentrobotics.org/quadruped-benchmark N. Kau and S. Bowers, "Stanford Pupper: A Low-Cost Agile Quadruped Robot for Benchmarking and Education", 2021. arXiv:2110.00736v1

[2] J. Hwangbo, J. Lee, A. Dosovitskiy, D. Bellicoso, V. Tsounis, V. Koltun, and M. Hutter, "Learning Agile and Dynamic Motor Skills for Legged Robots", 2019. arxiv:1901.08652v1

[4] H. Mania, A. Guy, B. Recht, "Simple random search provides a competitive approach to reinforcement learning", 2018. arXiv:1803.07055v1

[5] A. Iscen, K. Caluwaerts, J. Tan, T. Zhang, E. Coumans, V. Sindhwani, and V. Vanhoucke, "Policies Modulating Trajectory Generators", 2019. arXiv:1910.02812

Libraries:

[3] Pupper simulator (https://github.com/jietan/puppersim)

[6] PyTorch (http://pytorch.org)