High-Performance Reinforcement Learning on SpotMini: Optimizing Simulation Parameters with Distributional Measures

A.J. Miller[1,2], Fangzhou Yu[1], Michael Brauckmann[1], and Farbod Farshidian[1]

Abstract— This work presents an overview of the technical details behind a high-performance reinforcement learning policy deployment with the *Spot RL Researcher Development Kit* for low-level motor access on Boston Dynamic's SpotMini. This represents the first public demonstration of an end-to-end reinforcement learning policy deployed on Spot hardware with training code publicly available through Nvidia IsaacLab and deployment code available through Boston Dynamics. We utilize Wasserstein Distance and Maximum Mean Discrepancy to quantify the distributional dissimilarity of data collected on hardware and in simulation to measure our sim-to-real gap. We use these measures as a scoring function for the Covariance Matrix Adaptation Evolution Strategy to optimize simulated parameters that are unknown or difficult to measure from Spot. Our procedure for modeling and training produces high-quality reinforcement learning policies capable of multiple gaits, including a flight phase. We deploy policies capable of over 5.2m/s locomotion, more than triple Spot's default controller maximum speed, robustness to slippery surfaces, disturbance rejection, and overall agility previously unseen on Spot. We detail our method and release our code to support future work on Spot with the low-level API.

I. INTRODUCTION

Boston Dynamics' SpotMini (Spot) [1] is known the world over for opening doors [2], working in factories [3], and its many dances [4]. It has captured the curiosity of the public and the imagination of roboticists about what legged robots can begin to look like in everyday life. With the release of its SDK [5] and the commercial launch of the hardware platform, Spot took some of the first steps for a highly articulated robot toward successful real-world commercial applications. In another step earlier this year, Boston Dynamics unveiled their first inclusion of reinforcement learning (RL) techniques in their control stack [6].

In continuation of these advancements, we present in this work the training and deployment of the first fully learned control policy on Spot hardware, describe a sim-to-real gap quantification procedure using only on-board sensing, and provide a simulation parameter optimization process based on this quantification. We demonstrate the capability of our procedure by training a policy to push the limits of Spot. We show new speed, agility, and robustness with our end-to-end policy. We describe our modeling, training, and deployment as an example for researchers and engineers to train and deploy their own RL policies on Spot.

Our contributions are the following:

[1] Boston Dynamics AI Institute, Cambridge, MA 02139, USA: amiller@theaiinstitute.com, ffarshidian@theaiinstitute.com
[1] Department of Electrical Engineering and Computer Science, MIT, Cambridge, MA 02139, USA

Fig. 1. Spot galloping in flight on a flat track with our policy at 5.2m/s.

1) An evaluation procedure for measuring sim-to-real gap inspired by generative learning techniques and an optimization procedure for selecting simulator parameters
2) The first end-to-end RL control policy on Spot hardware and open-source training and deployment code
3) Demonstrations of the extended capabilities of our control policy's robustness and agility, including a more than triple maximum forward velocity over the default Spot controller

II. RELATED WORKS

The last decade has developed many highly articulated fully electric quadruped robots like Spot. These have included MIT's Cheetah 3 [7], IIT's HyQ [8], and ANYbotic's ANYmal [9]. Traditionally these systems have been controlled by model-based optimization schemes such as Model Predictive Control (MPC) [10][11] and Whole Body Control (WBC) [12] but in recent years, RL has demonstrated itself as a reliable, performant and robust tool in developing policies for the real world [13][14][15] while leveraging the advantages of simulation parallelization [16]. RL enables concurrent training with specialized networks [17], adaptation during deployment [18], and new dynamism on existing platforms including higher speeds [14][19] and entirely new behaviors like parkour [15].

Policy training in the simulation enables fast data collection, but simulation rollouts may not accurately match hardware. Sim-to-real differences can dramatically impede performance [20][21] and many methods have been proposed to minimize this gap. Domain randomization enables the policy to capture the hardware within the learned distribution

[22][23] but may cause degradation in the performance and precision of the policy. Actuator networks [13] replicate actuator input/output behaviors and show great performance but require reliable joint torque measurements. Others have forgone improving the simulation altogether and focused on tuning the policy during deployment [18].

System identification approaches instead search for better values for model parameters [24][19] from the evaluation of hardware data. A naive approach compares trajectory rollouts directly; this is difficult, however, because errors compound and result in substantial drift. Single-step updates are often used instead and evaluated with state-based methods such as least squares [25][26]. This requires accurate simulation resets to match the hardware state, but this is not always feasible. For example, states like contact need body height information that isn't directly measurable from onboard sensing. With modern simulators, we can instead easily generate large amounts of data and evaluate them in aggregate.

Generative learning provides examples of evaluating performance like this and many solutions focus on the data distribution [27]. Specifically, Wasserstein distance [28] and Maximum Mean Discrepancy (MMD) [29] have been used with great success to evaluate the performance of Generative Adversarial Networks (GANs) by comparing features of the distribution and are indifferent to compounding error like in time-series data. Applications of these measures in robotics include multi-modal behaviors with GAN structures [30], imitation learning using GAN-style losses [31], and transfer learning of exteroceptive information [32]. These methods of scoring performance based on distributional characteristics instead of state-based evolution of dynamics leverage the advantages of modern simulators without the onerous requirement that the initial states match.

III. METHODS

Here we detail our methods to model Spot and train our robust high-speed RL policies. We explain our modeling of both parameters known directly from measurements or the specifications of Spot as well as parameter estimates for joint friction and the torque-speed limit profiles using our scoring methodology. We collect state rollouts on hardware and in simulation and use Wasserstein Distance and MMD to quantify distributional similarity. We optimize our similarity score using the Covariance Matrix Adaptation Evolution Strategy (CMA-ES) on the parameter estimates to improve our model. We train a high-speed locomotive policy using our optimized Spot model and describe our training procedure.

A. Modeling

Modeling the actuators as best as possible is key so that the policy remains in distribution when deployed on the platform. In the case of Spot, the motor controllers use torque-sensing feedback to achieve the desired torque in the motor output. This alleviates some common modeling and randomization including accounting for gear ratios or unreliable actuators. However, we still need to account for communication delay within the low-level API, quantify state
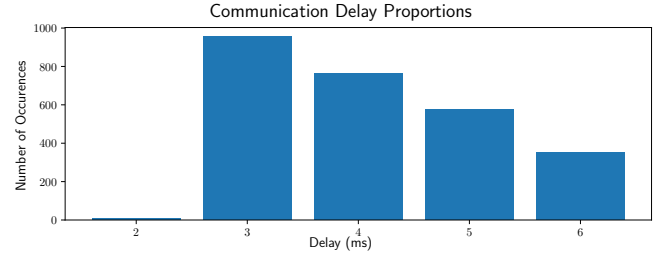

Fig. 2. Histogram plot of the measured communication delay of Google Remote Procedure Call messages sent to the Spot API. This is used to estimate the delay in torque application we use in the simulated actuators.
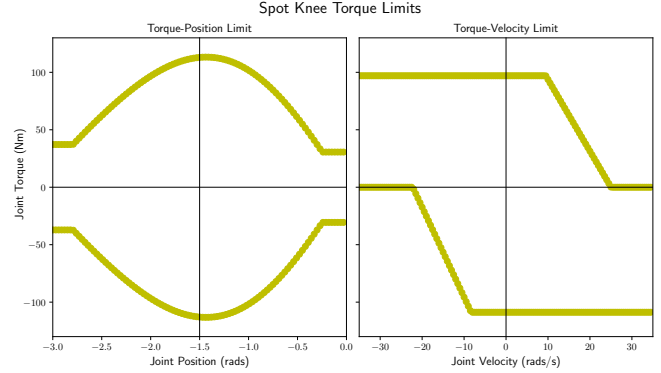

Fig. 3. Torque-position (left) and torque-speed (right) limits of the remotized knee actuator scatter plotted into two dimensions. These values are used to cap the applied torque in the simulated joints to better match hardware limitations.

estimation noise for the policy observations, include torque limits, and approximate frictional effects within each joint.

Communication Delay: We communicate with the motors using Google Remote Procedure Calls (gRPC) [33]. We quantified communication delay by measuring the time to send a command and receive application confirmation. We collected sample times in milliseconds and show the results in Fig. 2. To model the delay within our simulation, we buffer the action outputs of the network for 5ms. Delaying action application the total delay is identical to delaying both the observations and actions.

Actuators: Each leg of Spot has three actuators; from the base towards the foot: hip$_x$, hip$_y$, and knee. The motor packages are identical for each, but the knees also have a remotized lead-screw joint. This joint has a higher friction coefficient, and as expected we found the optimized knee friction values to be higher than for the hips Table I. We model torque limits for the joints to cap the torque available for the policy to request. The torque limits of the hip joints are set to flat maximum values. The torque limits of the remotized knees depend on the joint's position and velocity. We model these limits as shown in Fig. 3. We set the torque-position limits using the Spot API documentation [5] and torque-speed limits from our method described in Section III-C. The limit is computed as the minimum of the torque-position and torque-speed limits.

Observation Noise: We reduce the observation sim-to-real

gap using data-driven methods to construct a Gaussian noise model. Although empirical results indicate training with corrupted observations is not necessary for good sim-to-real behavior on Spot, we found noise modeling results in more robust policies and accelerates training by injecting more exploration into the policy rollout. For every observation feature o_t, we apply a white noise model by sampling a Gaussian distribution: $\mathbf{n}_t \sim \mathcal{N}(0, \sigma_t)$, for the corrupted observation $\hat{\mathbf{o}}_t = \mathbf{o}_t + \mathbf{n}_t$. The values for σ_t are derived from data collected from the Spot state estimator throughout a conservative walking gait and logged at the control frequency f_s. The raw data is transformed into the frequency domain using a Fast Fourier Transform (FFT). We then normalize the FFT by multiplying by the frequency f such that the energy content of the noise signal is constant across all frequencies [34]. The value for σ_t is estimated by computing the average signal power of the normalized FFT over the right half of the frequency domain $[\frac{f_s}{4}, \frac{f_s}{2}]$ where $\frac{f_s}{2}$ is the Nyquist frequency.

B. Similarity Scoring of Simulator Performance

In simulation training, there are inevitable mismatches between the synthetic data generated from the simulator and the real hardware. Minimizing the sim-to-real gap is a necessity for high-performance hardware results, and many methods have been proposed in the literature to mitigate this challenge [35][36][18][37][17][38]. In one extreme, a first principles approach may attempt to estimate uncertain physical parameters such as the actuator model [39]; however, the complexity can be quite high and quickly become arduous to replicate on new systems. On the other extreme, a black-box learned approach such as the actuator net [13], may obfuscate the full details. Additionally, the learned approach relies on output torque measurements, which many systems do not have, such as robots with pseudo-direct-drive actuators [7][40], or may require invasive procedures to disassemble and take measurements, which may not be simple or possible.

We balance these considerations with a gray-box parameterization. Potential characteristics may include joint friction, actuator armature, limitations from the torque-speed curve, or power supply dependencies on temperature and charge state. The limitations we observed in our original deployments were lower measured torque than expected, slower swing cycle re-circulation, and joint position overshooting. We chose to focus on joint friction and torque-speed curve limits as our hyperparameters.

Regardless of the hyperparameter space, defining evaluation metrics for quantifying the sim-to-real gap is another daunting task. We were inspired by the techniques used in generative learning, where the alignment of qualitatively "good" desired behavior and quantitative formulations used to train is very difficult. A common approach in generative learning is to use statistical measures to evaluate the distributional differences between the target and sample data [27]. We see an analogy where the execution of our policy in simulation acts like a generator and the real data is the execution of our policy on hardware. Similarly, we
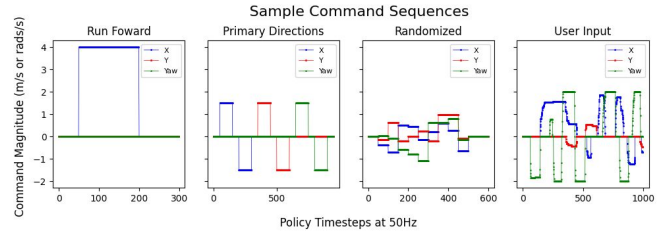

Fig. 4. Command sequence used to generate test data for evaluation of hardware and simulation similarity. This includes running at 4m/s, moving 1.5m/s in all six command directions, randomized, and user commands.

aim to minimize distributional differences between simulated rollouts and pre-collected hardware for a given policy.

We utilized two distribution measurement methods widely adopted in generative applications: Wasserstein Distance [28] and MMD [29]. These measures are known for their high discriminability, boundedness, ease of implementation, and lack of dependency on pre-trained models [27], which make them prime candidates for our application. Wasserstein distance, also known as Earth Mover's distance, measures the minimum distance needed to convert one probability distribution into another so that the shapes of the distributions match. MMD measures the difference in the means of the distributions projected into an embedded higher dimensional space. We used both measures to quantify the dissimilarity between hardware and simulation data.

A final consideration in using the evaluation metrics is the feature space used to represent the samples. We chose measured joint position, velocity, and policy action[1]. For data collection, we used a series of scripted commands to gather data. We used four different command sequences, shown in Fig. 4. The sequences included running forward at high velocity, sampling the six commandable directions at moderate velocity, a random sampling of quickly changing commands, and a series of user commands. For each sequence, we sampled on hardware 5 times and the same command sequences were used in the simulation. We divided the simulation samples by the number of variations of each command sequence and calculated the measure distance only between hardware/simulation pairs with the same command sequence. We aggregated the variations together by taking the average of the distance measure.

C. Hyperparameter Optimization

With measures to quantify the similarity of policy performance, we can devise an optimization on the multi-dimensional hyperparameter space to reduce the sim-to-real gap. We first train a policy in simulation with the default hyperparameter values. Next, we deploy this policy on hardware and log the real data. Based on the real data, we perform a hyperparameter optimization. We then retrain a new policy using the updated hyperparameter values. Some fine-tuning of the reward functions may be required at

---

[1]To normalize the feature space, we scaled the joint position by a proportional joint gain k_p, joint velocity by damping joint gain k_d, and the policy action by action scaling factor σ_a and k_p.

TABLE I
JOINT FRICTION, TORQUE, AND SPEED LIMIT PARAMETERS OPTIMIZED
BY OUR METHOD DESCRIBED IN SECTION III-C AND USED IN TRAINING.

Parameter	Values	
Friction	Hips: 0.008Nm	Knees: 0.180Nm
Torque Limits	Max: 97.00Nm	Min: -108.79Nm
Speed Limits	Max: 25.03rad/s	Min: -22.22rad/s
Torque-Speed Intersect	Max: 9.48rad/s	Min: -8.32rad/s

this step. Finally, we deploy this newly trained policy on hardware. In practice, we found that a single iteration of this process was sufficient to find performant simulation hyperparameters but the process could be repeated if desired.

We chose joint friction and torque-speed limits with a simplified model of 8 hyperparameters to optimize. The model includes two friction parameters with hip_x and hip_y set jointly and the remotized knee friction set separately. We created a 6-parameter simplification of the torque-speed curve with maximum and minimum torques, maximum and minimum speeds, and the torque-speed trade-off in quadrants 1 and 3 through the intersection between the torque and speed limits. A torque-speed curve limit can be seen in Fig. 3.

For our optimization, we used the CMA-ES [41] as a gradient-free sampling method to set the hyperparameters and minimize distributional measure error. We selected CMA-ES because of its successful applications in hardware design [42][43] and motion generation [44][45][21]. The algorithm works well in a non-smooth optimization landscape and scales well for tens to one hundred search parameters.

We ran CMA-ES for 100 iterations with a sample population size of 10. For each iteration, we run the simulator with a new hyperparameter sample, collect simulated data based on the command generation sequence, and evaluate using the measures to score the similarity to real data. Afterward, we selected the parameters shown in Table I and iterated the learning process by retraining a new policy.

D. Training

We trained our policies using the Nvidia IsaacLab [46] framework built on Nvidia IsaacSim [47] and with learning algorithms from RSL_RL [48]. IsaacLab provides a general RL training environment to model and simulate the Spot, interfaces with IsaacSim for observations, actions, and rewards, and parallelizes data collection and neural network training. We train a policy using the Actor-Critic [49] formulation and updated with Proximal Policy Optimization (PPO) [50]. An overview of the training pipeline can be seen in Fig. 5.

Actions: The policy is sampled at 50Hz and the actions are converted into motor torques τ by the PD control law as defined in (1). The control law calculates the desired torque from the set points and gains and is then applied at the simulated joints at 200Hz.

$$\tau = k_p(\sigma_a a + q_d - q) - k_d \dot{q}, \qquad (1)$$

k_p and k_d are the stiffness and damping gains for the actuators. k_p is 60.0 N·m·rad^{-1} and k_d is 1.5 N·m·s·rad^{-1} for training and deployment on all 12 Spot actuators. The policy output action a is multiplied by a scaling factor $\sigma_a = 0.2$. The action is represented as a joint position set-point offset and is summed with the default joint position q_d minus the measured joint position q. The negative of measured joint velocity $\dot{q}$ damps towards zero.

Observations: The observations for the neural network are described in (2). They are collected as the simulated states of the robot for training and from the state estimator when on hardware.

$$o = (v_{xyz}, \omega_{xyz}, g, cmd, q, \dot{q}, a_{t-1}), \qquad (2)$$

v_{xyz} is the body relative base linear velocity where x is longitudinal, y is lateral, and z is vertical axes. ω_{xyz} is the base angular velocity. g is the normalized gravity vector projection in the body frame. This acts as a proxy for the base orientation. cmd is the velocity tracking command of the robot consisting of base longitudinal V_x, lateral V_y, and yaw ω_z. Commands were sampled in the range $V_{cmd,x} \sim U(\text{-}2.0, 5.5)$m/s, $V_{cmd,y} \sim U(\text{-}1.5, 1.5)$m/s, and $\omega_{cmd,z} \sim U(\text{-}2.0, 2.0)$rad/s. q are the joint angles, $\dot{q}$ are the joint velocities, and a_{t-1} are the previous network actions.

Rewards: The task rewards include tracking the base linear ($r_{v_{xy}}$) and base angular (r_{ω_z}) velocity tracking in the three commanded axes, and a penalty for base velocity in the three non-commanded axes ($r_{v_z \omega_{xy}}$). The style rewards include the time each foot spends in the air versus in contact (r_{at}), penalizing the variance of the total air and contact time of the previous step between the feet (r_{atv}), the vertical foot clearance from the ground during swing and stance (r_{fc}), penalizing the impact velocity of the foot when transitioning from swing to stance (r_{fi}), penalizing foot slippage during stance (r_{fs}), penalizing low distance between the feet (r_{fd}), penalizing deviation from flat orientation (r_ϕ) relative to the ground plane, and encouraging swing/stance synchronization for the desired gait (r_g). The regularization rewards include penalizing large changes in the current action from the previous action (r_{as}), penalizing deviation from the default joint position (r_q), penalizing joint velocity ($r_{\dot{q}}$), penalizing joint acceleration ($r_{\ddot{q}}$), and penalizing joint torque (r_τ).

Environment: We randomized the terrain, ground friction, base mass, initial body position X_0 and velocity V_0, and initial joint position q_0 and velocity $\dot{q}_0$. We randomized a low poly noisy terrain to deviate from flat. The foot-ground friction was randomized between $U(0.3, 1.0)$ for static and $U(0.3, 0.8)$ for dynamic friction. Additional mass of the base $\delta_{m,\text{base}} \sim U(\text{-}2.5, 2.5)$, initial body position $X_{xy,0} \sim U(\text{-}0.5, 0.5)$, $\phi_{z,0} \sim U(\text{-}\pi, \pi)$ and body velocity $V_{x,0} \sim U(\text{-}1.5, 1.5)$, $V_{y,0} \sim U(\text{-}1.0, 1.0)$, $V_{z,0} \sim U(\text{-}0.5, 0.5)$, $\omega_{x,0} \sim U(\text{-}0.7, 0.7)$, $\omega_{y,0} \sim U(\text{-}0.7, 0.7)$, $\omega_{z,0} \sim U(\text{-}1.0, 1.0)$, deviation from initial joint position $\delta_{q,0} \sim U(\text{-}0.2, 0.2)$ and velocity $\dot{q}_0 \sim U(\text{-}2.5, 2.5)$ were uniformly sampled in these
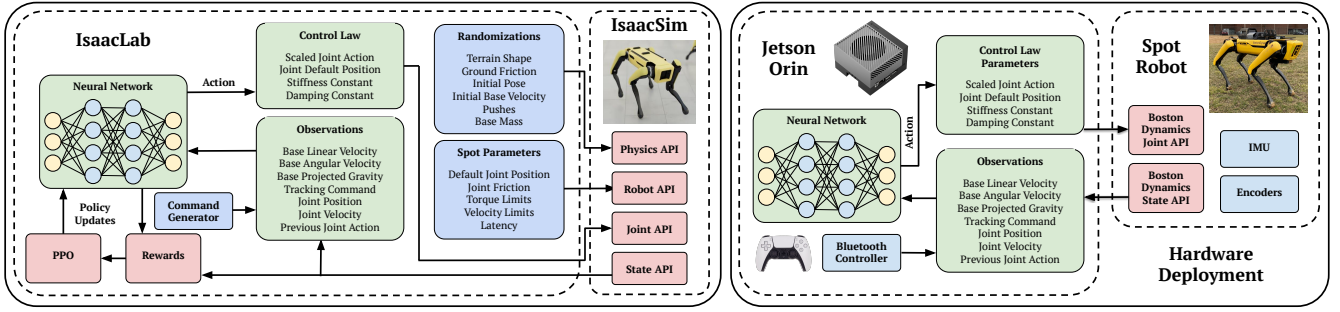
Fig. 5. Visual overview of the training and deployment pipeline. On the left, is an approximation of how the policy is trained in simulation, and on the right, is how it's deployed on hardware. IsaacLab is the simulation environment, provides the physics engine, and employs our modeling for training. We deploy our policies on Spot using an Nvidia Jetson Orin that communicates with the Boston Dynamics API for state estimates and sends motor control law values with the *Spot RL Researcher Development Kit* to command the motors.

ranges and fixed at the beginning of the episode. We randomly pushed the robot by modifying its base velocity in the range $V_{xy} \sim U(-0.5, 0.5)$. We did not randomize joint friction, stiffness, damping, torque limits, ground restitution, action scale, limb mass, or apply forces on the body during training.

Termination: We terminate an episode and reset the agent under three conditions: if the agent reaches the episode timeout maximum of 20 seconds; if the agent touches the ground with its base or upper or lower leg; or if the agent goes beyond the terrain boundary. We considered this out-of-bound condition similar to a timeout as it happens when the robot successfully runs to the terrain boundary. We found this to be a reasonable trade-off instead of increasing the terrain size due to its impact on simulation performance.

E. Deployment

In the Spot API, we communicate with the state estimator and motors via gRPC messages. We query the API for the state estimates and input the concatenated states into the trained neural network, represented in runtime ONNX. The network output is used as a joint position set-point offset. This offset is sent with the default joint positions, joint velocity set-points (set to 0.0rad/s), stiffness ($k_p = 60.0$ N·m·rad^{-1}), and damping ($k_d = 1.5$ N·m·s·rad^{-1}) in a gRPC message to the low-level motor control API. The desired joint torques are calculated from these values and updated using measured torque feedback.

IV. RESULTS

In this work, we pushed Spot in ways previously unseen to better understand the limitations of the platform. RL is a powerful tool for this and accurate modeling is important for best results. Our method produces the necessary modeling without disassembling the hardware to achieve our desired performance. We demonstrate a more than tripling of the maximum forward velocity, new gaits including a flight phase, and robustness to disturbances on slippery ground surfaces. We showcase these abilities and results in the supplementary video.

A. CMA-ES Optimization

We ran our CMA-ES optimization using a weighted average of the measure scores for each of the four command sequences. We initialized the values for actuator frictions to 0.0Nm, torque limits to +/-70Nm, torque speed limits to +/-20rad/s, and torque-speed intersects at +/-0.0rad/s. The optimization resulted in the values of the actuator friction and torque-speed curves shown in Table I. We minimize the distributional differences between the joint positions, joint velocities, and actions of the policy. The values are normalized by their respective control law scaling constants. Importantly, simulated data does not include sensor noise.

We used these values to retrain our policy where only minor weight value adjustments were made to maintain behavior quality. We found a single iteration of the algorithm sufficient to markedly improve performance and increase maximum linear velocity from 3.8m/s to 5.2m/s on hardware. These improvements also enabled flight phase gaits as seen in Fig. 1, which previously failed on hardware.

B. Comparison to Baseline on Hardware

Empirical results of our method's improvements in minimizing sim-to-real differences are shown in Fig. 6. For these results, we deployed a baseline policy trained without our modeling and a policy trained with the CMA-ES modeling. Both policies were commanded from a standstill to run forward at an increasing velocity until 4.0m/s. Rollouts were collected on hardware and in simulation using the same settings as training. We plotted the actions and desired torques for the front and hind right knees as these joints are out of phase during the trot gait, and the knees required the most modeling with our method.

As shown in the figures, the joint actions and torque more closely match in simulation and on hardware for our policy than the baseline. This can be seen in the magnitude and frequency of the swing and the consistency of the motion. Notably, both policies struggle to produce negative torque during swing leg recirculation resulting in a slower swing and larger error that increases the desired torque value. This is due to power distribution limits not being included in our modeling, however, our policy is more resistant to these
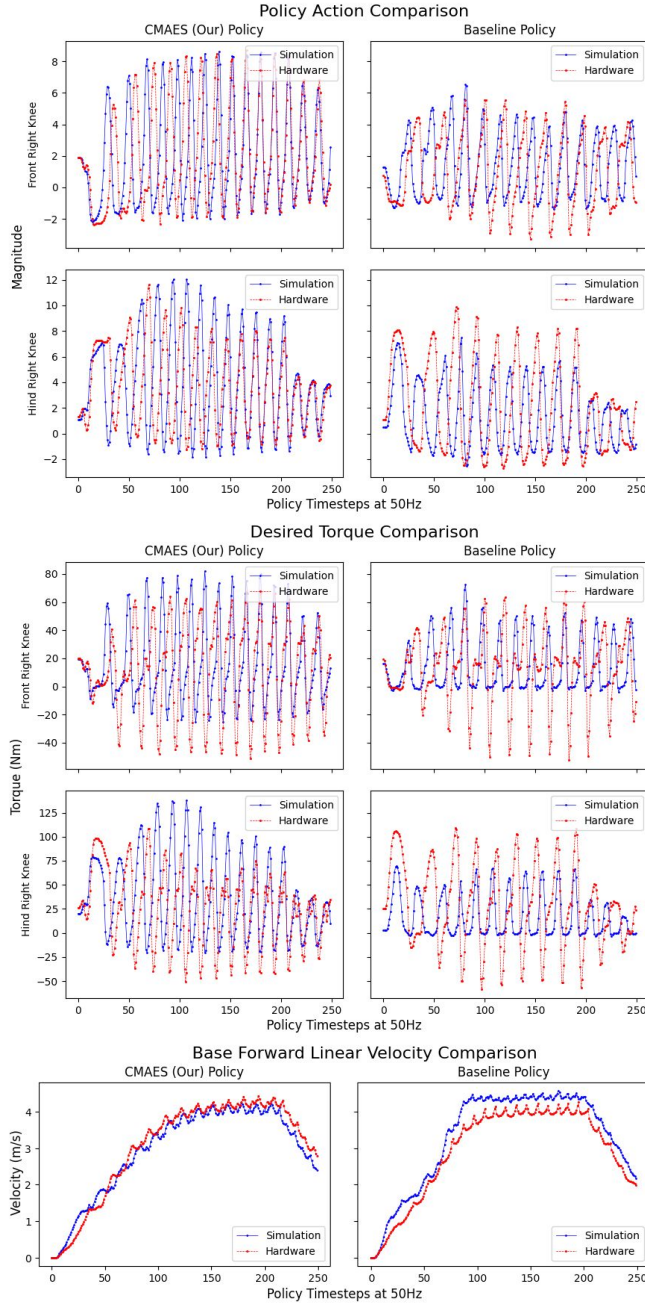
Fig. 6. Comparison of knee joint action and desired torque during a commanded 4.0m/s acceleration on hardware and in simulation comparing the performance of our CMA-ES modeling trained policy versus a baseline policy. The improvements are apparent in the better base velocity matching and amplitude and frequency matching of joints during the gait cycle.

effects as seen in the greater stability in the joint actions. This causes a larger difference in the swing leg time between the baseline simulation and the hardware reducing overall performance.

Importantly our better joint behavior matching results in better forward base linear velocity matching. The divergences seen in the baseline policy deployment are critically detrimental to the overall performance of the controller. The baseline policy is too out of distribution on the hardware

and results in nontrivial performance degradation which is substantially mitigated by our method.

C. Robustness and High-Performance

A valuable randomization for policy robustness is the ground friction coefficient. Our policy testing predominantly used moderate to high friction coefficient surfaces. To test our policy, we ran Spot on a low-friction surface created by applying soap and water on plexiglass plates. We ran tests on this surface to show the policy adapting to sudden changes in friction and recovering from foot slips, including staying upright after a knee collided with the ground.

The primary demonstration of our method's advantages is in the high-speed performance of our policy. We deployed on a rubberized outdoor running track to find the maximum velocity of our approach. In our testing, we reached sustained running speeds of 5.2m/s in a flying trot gait; far exceeding the default Spot controller's fastest forward velocity of 1.6m/s [51]. Our modeling of the noise in the state estimation, joint friction effects, and torque limits are necessary to reach this level of performance and validate the efficacy of our method in finding good parameter values for our simulated robot model. These demonstrations can be found in the supplementary video.

V. CONCLUSION

In this work, we present the first end-to-end deployment of an RL policy on Spot, detail our modeling, training, and deployment procedure, and release our code to support future research with the *Spot RL Researcher Developer Kit*. We provide this as a demonstration of our workflow using Spot low-level motor access. We quantify and evaluate the sim-to-real gap of our simulator model to the real hardware using a Wasserstein Distance and MMD-based scoring mechanism and create an optimization procedure with CMA-ES to tune our simulator parameters for better performance. Our process produces a high-quality Spot model of Spot and its actuators to enable greater speed, agility, and robustness never before seen on the platform. While there is more to do in leveraging Spot's vision capabilities, improving the modeling in simulation further, and pushing the hardware with even more ambitious behaviors, we believe this is a foundational first step in enabling broader future work.

REFERENCES

[1] Boston dynamics: Spot. Boston Dynamics. [Online]. Available: https://bostondynamics.com/products/spot/

[2] Hey buddy, can you give me a hand? Boston Dynamics. [Online]. Available: https://www.youtube.com/watch?v=fUyU3lKzoio

[3] What does it take to put spot to work? Boston Dynamics. [Online]. Available: https://www.youtube.com/watch?v=_Ux-N-NK2GM

[4] Uptown spot. Boston Dynamics. [Online]. Available: https://www.youtube.com/watch?v=kHBcVlqpvZ8

[5] Spot sdk. Boston Dynamics. [Online]. Available: https://dev.bostondynamics.com/

[6] P. Domanico. Reinforcement learning with spot. Boston Dynamics. [Online]. Available: https://bostondynamics.com/video/reinforcement-learning-with-spot/

[7] 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.

[8] C. Semini, N. G. Tsagarakis, E. Guglielmino, M. Focchi, F. Cannella, and D. G. Caldwell, "Design of hyq–a hydraulically and electrically actuated quadruped robot," *Proceedings of the Institution of Mechanical Engineers, Part I: Journal of Systems and Control Engineering*, vol. 225, no. 6, pp. 831–849, 2011.

[9] M. Hutter, C. Gehring, D. Jud, A. Lauber, C. D. Bellicoso, V. Tsounis, J. Hwangbo, K. Bodie, P. Fankhauser, M. Bloesch *et al.*, "Anymal–a highly mobile and dynamic quadrupedal robot," in *2016 IEEE/RSJ international conference on intelligent robots and systems (IROS)*. IEEE, 2016, pp. 38–44.

[10] 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.

[11] F. Farshidian, E. Jelavic, A. Satapathy, M. Giftthaler, and J. Buchli, "Real-time motion planning of legged robots: A model predictive control approach," in *2017 IEEE-RAS 17th International Conference on Humanoid Robotics (Humanoids)*. IEEE, 2017, pp. 577–584.

[12] C. D. Bellicoso, F. Jenelten, P. Fankhauser, C. Gehring, J. Hwangbo, and M. Hutter, "Dynamic locomotion and whole-body control for quadrupedal robots," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2017, pp. 3359–3365.

[13] 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, p. eaau5872, 2019.

[14] G. B. Margolis, G. Yang, K. Paigwar, T. Chen, and P. Agrawal, "Rapid locomotion via reinforcement learning," *The International Journal of Robotics Research*, vol. 43, no. 4, pp. 572–587, 2024.

[15] D. Hoeller, N. Rudin, D. Sako, and M. Hutter, "Anymal parkour: Learning agile navigation for quadrupedal robots," *Science Robotics*, vol. 9, no. 88, p. eadi7566, 2024.

[16] N. Rudin, D. Hoeller, P. Reist, and M. Hutter, "Learning to walk in minutes using massively parallel deep reinforcement learning," in *Conference on Robot Learning*. PMLR, 2022, pp. 91–100.

[17] G. Ji, J. Mun, H. Kim, and J. Hwangbo, "Concurrent training of a control policy and a state estimator for dynamic and robust legged locomotion," *IEEE Robotics and Automation Letters*, vol. 7, no. 2, pp. 4630–4637, 2022.

[18] A. Kumar, Z. Fu, D. Pathak, and J. Malik, "Rma: Rapid motor adaptation for legged robots," *arXiv preprint arXiv:2107.04034*, 2021.

[19] Y.-H. Shin, T.-G. Song, G. Ji, and H.-W. Park, "Actuator-constrained reinforcement learning for high-speed quadrupedal locomotion," *arXiv preprint arXiv:2312.17507*, 2023.

[20] F. Muratore, F. Ramos, G. Turk, W. Yu, M. Gienger, and J. Peters, "Robot learning from randomized simulations: A review," *Frontiers in Robotics and AI*, vol. 9, p. 799893, 2022.

[21] Z. Xie, G. Berseth, P. Clary, J. Hurst, and M. Van de Panne, "Feedback control for cassie with deep reinforcement learning," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2018, pp. 1241–1246.

[22] I. Exarchos, Y. Jiang, W. Yu, and C. K. Liu, "Policy transfer via kinematic domain randomization and adaptation," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2021, pp. 45–51.

[23] Z. Li, X. Cheng, X. B. Peng, P. Abbeel, S. Levine, G. Berseth, and K. Sreenath, "Reinforcement learning for robust parameterized locomotion control of bipedal robots," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2021, pp. 2811–2817.

[24] M. Kaspar, J. D. M. Osorio, and J. Bock, "Sim2real transfer for reinforcement learning without dynamics randomization," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2020, pp. 4383–4388.

[25] C. H. An, C. G. Atkeson, and J. M. Hollerbach, "Estimation of inertial parameters of rigid body links of manipulators," in *1985 24th IEEE Conference on Decision and Control*. IEEE, 1985, pp. 990–995.

[26] T. Lee, P. M. Wensing, and F. C. Park, "Geometric robot dynamic identification: A convex programming approach," *IEEE Transactions on Robotics*, vol. 36, no. 2, pp. 348–365, 2019.

[27] A. Borji, "Pros and cons of gan evaluation measures," *Computer vision and image understanding*, vol. 179, pp. 41–65, 2019.

[28] L. Rüschendorf, "The wasserstein distance and approximation theorems," *Probability Theory and Related Fields*, vol. 70, no. 1, pp. 117–129, 1985.

[29] G. K. Dziugaite, D. M. Roy, and Z. Ghahramani, "Training generative neural networks via maximum mean discrepancy optimization," *arXiv preprint arXiv:1505.03906*, 2015.

[30] X. B. Peng, Z. Ma, P. Abbeel, S. Levine, and A. Kanazawa, "Amp: Adversarial motion priors for stylized physics-based character control," *ACM Transactions on Graphics (ToG)*, vol. 40, no. 4, pp. 1–20, 2021.

[31] C. Li, M. Vlastelica, S. Blaes, J. Frey, F. Grimminger, and G. Martius, "Learning agile skills via adversarial imitation of rough partial demonstrations," in *Conference on Robot Learning*. PMLR, 2023, pp. 342–352.

[32] M. M. Rahman, T. Rahman, D. Kim, and M. A. U. Alam, "Knowledge transfer across imaging modalities via simultaneous learning of adaptive autoencoders for high-fidelity mobile robot vision," in *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2021, pp. 1267–1273.

[33] grpc: A high performance, open source universal rpc framework. Google. [Online]. Available: https://grpc.io/

[34] F. Mignard, "About the nyquist frequency," *Observatoire de la Côte d'Azur, Dpt. Cassiopée. GAIA FM*, vol. 22, 2005.

[35] 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.

[36] 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.

[37] S. James, P. Wohlhart, M. Kalakrishnan, D. Kalashnikov, A. Irpan, J. Ibarz, S. Levine, R. Hadsell, and K. Bousmalis, "Sim-to-real via sim-to-sim: Data-efficient robotic grasping via randomized-to-canonical adaptation networks," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2019, pp. 12 627–12 637.

[38] R. Buchanan, M. Camurri, F. Dellaert, and M. Fallon, "Learning inertial odometry for dynamic legged robot state estimation," in *Conference on robot learning*. PMLR, 2022, pp. 1575–1584.

[39] M. Hutter, "Starleth & co.: Design and control of legged robots with compliant actuation," Ph.D. dissertation, ETH Zurich, 2013.

[40] B. Katz, J. Di Carlo, and S. Kim, "Mini cheetah: A platform for pushing the limits of dynamic quadruped control," in *2019 international conference on robotics and automation (ICRA)*. IEEE, 2019, pp. 6295–6301.

[41] A. Auger and N. Hansen, "Tutorial cma-es: evolution strategies and covariance matrix adaptation," in *Proceedings of the 14th annual conference companion on Genetic and evolutionary computation*, 2012, pp. 827–848.

[42] T. Chen, Z. He, and M. Ciocarlie, "Hardware as policy: Mechanical and computational co-optimization using deep reinforcement learning," *arXiv preprint arXiv:2008.04460*, 2020.

[43] G. Fadini, T. Flayols, A. Del Prete, and P. Souères, "Simulation aided co-design for robust robot optimization," *IEEE Robotics and Automation Letters*, vol. 7, no. 4, pp. 11 306–11 313, 2022.

[44] N. Dehio, R. F. Reinhart, and J. J. Steil, "Multiple task optimization with a mixture of controllers for motion generation," in *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2015, pp. 6416–6421.

[45] X. Li, Z. Liang, and H. Feng, "Kicking motion planning of nao robots based on cma-es," in *The 27th Chinese Control and Decision Conference (2015 CCDC)*. IEEE, 2015, pp. 6158–6161.

[46] M. Mittal, C. Yu, Q. Yu, J. Liu, N. Rudin, D. Hoeller, J. L. Yuan, R. Singh, Y. Guo, H. Mazhar *et al.*, "Orbit: A unified simulation framework for interactive robot learning environments," *IEEE Robotics and Automation Letters*, 2023.

[47] Nvidia isaacsim. Nvidia. [Online]. Available: https://developer.nvidia.com/isaac/sim

[48] Rsl_rl. Robotic Systems Lab. [Online]. Available: https://github.com/leggedrobotics/rsl_rl

[49] I. Grondman, L. Busoniu, G. A. Lopes, and R. Babuska, "A survey of actor-critic reinforcement learning: Standard and natural policy gradients," *IEEE Transactions on Systems, Man, and Cybernetics, part C (applications and reviews)*, vol. 42, no. 6, pp. 1291–1307, 2012.

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

[51] Spot specifications. Boston Dynamics. [Online]. Available: https://support.bostondynamics.com/s/article/Robot-specifications