OPT-Mimic: Imitation of Optimized Trajectories for Dynamic Quadruped Behaviors

Yuni Fuchioka[1], Zhaoming Xie[2], and Michiel van de Panne[1]

Abstract— **Reinforcement Learning (RL) has seen many recent successes for quadruped robot control. The imitation of reference motions provides a simple and powerful prior for guiding solutions towards desired solutions without the need for meticulous reward design. While much work uses motion capture data or hand-crafted trajectories as the reference motion, relatively little work has explored the use of reference motions coming from model-based trajectory optimization. In this work, we investigate several design considerations that arise with such a framework, as demonstrated through four dynamic behaviours: trot, front hop, 180 backflip, and biped stepping. These are trained in simulation and transferred to a physical Solo 8 quadruped robot without further adaptation. In particular, we explore the space of feed-forward designs afforded by the trajectory optimizer to understand its impact on RL learning efficiency and sim-to-real transfer. These findings contribute to the long standing goal of producing robot controllers that combine the interpretability and precision of model-based optimization with the robustness that model-free RL-based controllers offer.**

I. INTRODUCTION

Quadruped control has seen significant recent advances emerging from trajectory optimization and reinforcement learning approaches. As a model-based method, trajectory optimization offers fast iteration for designing motions. With appropriate simplifications, it can also be used in real-time for model-predictive control. On the other hand, reinforcement learning (RL) is well suited to providing particularly robust and fast-to-compute control policies. This comes at the cost of offline computation and often requires careful tuning of rewards and hyperparameters in order to arrive at meaningful solutions. A combined solution has the potential of providing the best of both worlds, wherein trajectory optimization provides fast and predictable motion design of a reference motion, after which RL can be used to imitate or mimic that motion.

While RL-based motion-imitation policies have recently seen much success, it remains unclear how it can best be used in conjunction with reference trajectories provided by trajectory optimization. Can the combined approach be used to design dynamic motions with minimal tuning? Which components of the optimized trajectory should be leveraged by the RL policy and the PD-controllers used to control the motions? For example, feed-forward joint velocities and joint torques are available, but should they be used? How do these different choices impact on sim-to-real transfer? We investigate these questions by designing four motions for the Solo8 robot, across three feed-forward configurations, and with consistent hyperparameter settings across these twelve scenarios, and test these using a Solo8 robot with predominantly proprioceptive sensing.

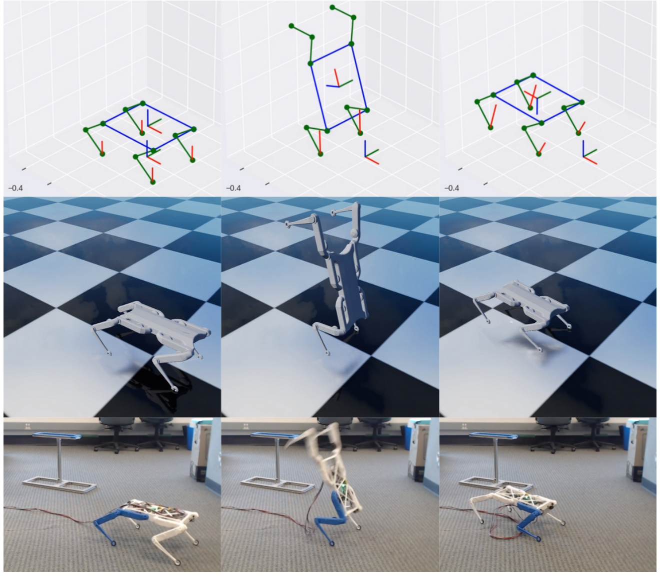

Fig. 1. Snapshots showing the 180-backflip motion produced from the motion generation system considered in this work, including the simple-model trajectory optimization (top), full-model reinforcement learning (middle), and transfer to the physical robot (bottom).

II. RELATED WORK

A. Trajectory Optimization for Legged Robots

Trajectory optimization is a process to generate physically feasible trajectories offline, online feedback controllers can then be designed to track such trajectories. Trajectory optimization is particularly challenging for legged robots due to the hybrid dynamics arising from various contact modes, in addition to the high dimensional and nonconvexity of the resulting problem. Various methods are proposed to solve trajectory optimization efficiently, including collocation method, e.g., [1], [2], [3], and shooting based method, e.g., [4], [5]. Simplified model such as the single rigid body dynamics model or inverted pendulum can also be used to get approximate solution [6], [7].

B. Reinforcement Learning for Quadrupedal Robots

RL has been used with good success to generate robust locomotion behaviors for quadrupedal robots, e.g., [8], [9], [10], [11]. Without providing the algorithm prior knowledge of how a quadruped should move, it often requires tedious

[1] Faculty of Computer Science, The University of British Columbia
[2] Department of Computer Science, Stanford University

reward tuning to obtain reasonable behaviors. Combining RL and model-based control can help mitigate the reward tuning issue and generate natural behaviors like trotting and jumping, e.g., [12], [13], [14]. This line of work often designs the reward based on the locomotion task, i.e., follow a desired velocity, and exhibits limited behaviors. In this work, we aim to apply RL to achieve agile behaviors that do not easily emerge from optimizing locomotion rewards.

C. Imitation-based Reinforcement Learning for Legged Robots

It is often hard to generate policies that behave as intended through task rewards alone, e.g., a policy will likely prefer a trotting gait over a pacing gait for a standard quadruped locomotion task. To provide the user more control over the behaviors, reference trajectories can be provided to encourage desired motions. One can design a reward function to explicitly track the reference trajectories, e.g., [15], [16], [17]. Inverse reinforcement learning techniques such as adversarial motion priors can also be used to learn a reward function to encourage the policy to produce motions that look similar to a prescribed motion dataset, e.g., [18], [19]. There are various ways to obtain a reference motion, e.g., trajectories optimization [20], [17], [7], [21], motion capture data from animals [15], [18] or even crude hand design motions [16], [22], [23]. In this paper, we use a tracking-based reward to generate highly dynamics behaviors. We demonstrate how a reference motion from trajectory optimization is crucial for learning performance as well as sim-to-real transfer. Furthermore, we explore how different feedforward components from the optimized motion can impact learning performance and sim-to-real transfer.

III. METHOD

A. Overview

The overview of our framework is given in Fig. 2. Trajectory Optimization is used to produce a library of open-loop reference motion trajectories that are feasible for the simplified dynamics model used by the optimization. These reference motions are then used by an imitation-based reinforcement learning framework to produce a neural network closed-loop feedback controller for the full-order robot model to mimic the open-loop reference. Finally, the reference motions and network controllers are loaded onto robot control software to test on the physical robot. The following sections describe each component in further detail.

B. Trajectory Optimization

Given a robot with configuration space $Q = \mathbb{R}^3 \times SO(3) \times \mathbb{R}^{n_j}$, where n_j is the number of joints on the robot. We wish to obtain a function $\mathbb{R} \rightarrow Q \times \mathcal{T}Q \times \mathbb{R}^{n_j}$: $\phi \rightarrow [p(\phi), R(\phi), q(\phi)] \times [\dot{p}(\phi), \omega(\phi), \dot{q}(\phi)] \times \tau(\phi)$, where p, R, q, ω are the linear position, orientation, joint configuration and angular velocity of the robot, $\tau(\phi)$ denotes the joint torque needed to accomplish the motion and $\phi \in [0, T]$ is the timing variable.

To avoid the expensive computation needed to optimize the full order model, Single Rigid Body (SRB) is first used to optimize for $p(\phi), R(\phi), \dot{p}(\phi)$ and $\omega(\phi)$, as well as a set of foot positions $p_{\text{foot}}(\phi) = [p_1, p_2, p_3, p_4]$ and ground reaction forces $f(\phi) = [f_1, f_2, f_3, f_4]$. We use the IPOPT interior-point solver [24] interfaced through the CasADi Python library [25] to solve the direct collocation problem that we wrote by combining elements of [26], [1], and [6], designed specifically to quickly and flexibly produce a variety of dynamic motions not limited to locomotion.

The SRB dynamics constraints are given by

$$p^+ = p + \dot{p}\Delta t \tag{1}$$

$$\dot{p}^+ = \dot{p} + \big(\frac{1}{m}\sum_i f_i + g\big)\Delta t \tag{2}$$

$$R^+ = Re^{([\omega\times]\Delta t)} \tag{3}$$

$$\begin{aligned} \omega^+ &= \omega \\ &+^B I^{-1}\big(R^T(\sum_i (p_i - p) \times f_i) - [\omega\times]^B I\omega\big)\Delta t, \end{aligned} \tag{4}$$

and for notational simplicity, we drop the time dependency on the variable $p, \dot{p}, R, \omega$, and use superscript $^+$ to denote the variable at the next timestep, Δt is the fixed length of the timestep set as 20ms, m is the total mass of the robot, g is the gravitational acceleration vector, ^BI is the body frame inertia vector of the robot body, $e^{(\cdot)}$ denotes the matrix exponential, and $[\omega\times] \in \mathbb{R}^{3\times3}$ is the skew-symmetric cross product matrix produced by $\omega \in \mathbb{R}^3$ [6]. The objective is a Linear Quadratic Regulator tracking cost summed over the fixed trajectory length for tracking the kinematic initial guess trajectory, as well as a regularization term smoothing the foot trajectories given by $\big(1/\Delta t\big)^2\big((p_i)^+ - p_i\big)^T R_{\dot{p}}\big((p_i)^+ - p_i\big)$ for diagonal regularization weight matrix $R_{\dot{p}}$. Following [26], we do not fix the foot contact locations and timings and allow the optimizer to choose foot swing phase trajectories and contact configurations. Therefore, we impose explicit contact complementary constraints

$$(p_i)_z \geq 0 \tag{5}$$

$$(f_i)_z(p_i)_z = 0 \tag{6}$$

$$(f_i)_z\big((p_i)^+_x - (p_i)_x\big) = 0 \tag{7}$$

$$(f_i)_z\big((p_i)^+_y - (p_i)_y\big) = 0, \tag{8}$$

where subscripts x, y, and z denote the corresponding component of the vector [26]. Similarly to [6], friction cone constraints are approximated with friction pyramid constraints along with a maximum force limit given by $\big((f)_z\big)_{max}$,

$$0 \leq (f)_z \leq \big((f)_z\big)_{max} \tag{9}$$

$$|(f_i)_x| \leq \mu(f_i)_z \tag{10}$$

$$|(f_i)_y| \leq \mu(f_i)_z. \tag{11}$$

Inspired by [1], we impose kinematic constraints as $L1$ norm

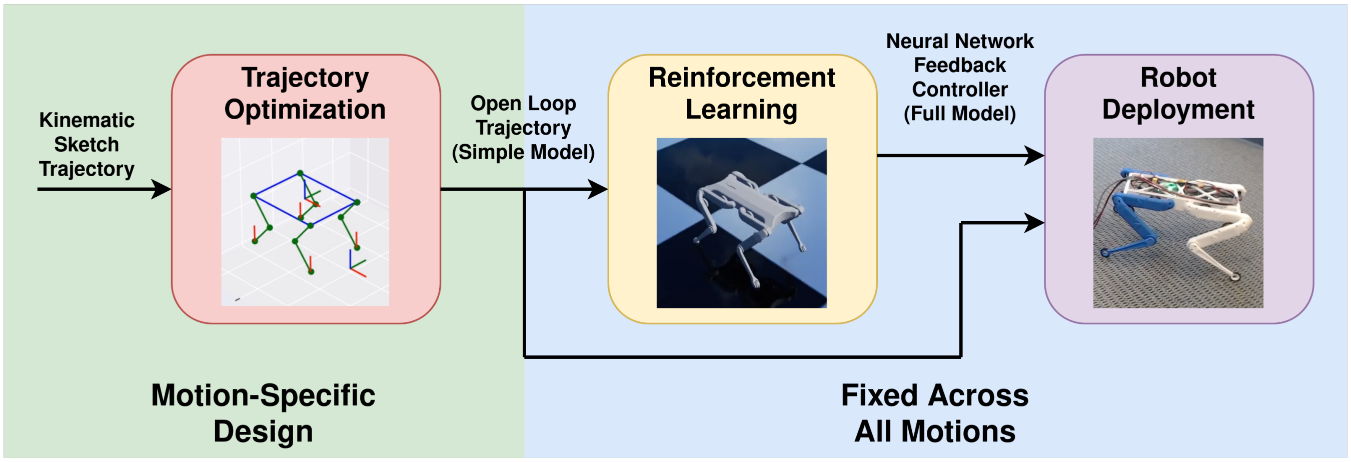
Kinematic Sketch Trajectory
Trajectory Optimization
Open Loop Trajectory (Simple Model)
Reinforcement Learning
Neural Network Feedback Controller (Full Model)
Robot Deployment
Motion-Specific Design
Fixed Across All Motions

Fig. 2. The motion generation system considered for this work. Trajectory Optimization is used to produce open-loop trajectories feasible for the simplified Single Rigid Body (SRB) model, given motion specification through kinematic sketch trajectories and optimization constraints. Reinforcement Learning (RL) is then used to produce closed-loop feedback controllers capable of executing these motions for a full-order model, which can then be deployed directly on a physical robot without additional online model adaptations. We note that motion-specific tuning happens only in the trajectory optimization phase, whereas the RL environment and robot control software are fixed over all four motions considered.

constraints in the shoulder plane

$$\left\| \begin{bmatrix} ({}^{B_i}p_i)_x \\ ({}^{B_i}p_i)_z \end{bmatrix} \right\|_1 \leq l_{leg} \tag{12}$$

$$({}^{B_i}p_i)_y = 0, \tag{13}$$

where $({}^{B_i}p_i)$ is the ith foot position in its corresponding shoulder frame, and l_{leg} denotes the maximum allowable extension length of the leg. Unlike [1], we use a $L1$ norm ball rather than a cube to allow the leg to fully extend downwards rather than in the diagonal directions.

Finally, we use a hand-crafted kinematic trajectory that roughly specifies the motion and serves as the tracking objective and initial guess solution. Noting that most of the objectives and constraints are convex with the exception of the SRB dynamics and contact complementarity, we found that fast and reliable convergence could only be achieved when the initial guess trajectory was set to have non-zero ground reaction forces whenever the corresponding foot was on the ground, since contact complementarity effectively encodes a discrete decision of whether the foot should be in contact or not, which the optimizer could not reliably choose without a guiding initial guess.

After the solver converges to a solution, we can obtain $q(\phi)$ and $\dot{q}(\phi)$ from $p_i(\phi)$ through inverse kinematics and finite differences. $\tau(\phi)$ is can be obtained through Jacobian transpose $\tau = J^T(-f)$, where J denotes the Jacobian of the all the legs.

C. Imitation-based Reinforcement Learning

Given the simple-model open-loop reference motion produced by the Trajectory Optimizer, the purpose of the RL training is to produce a full-order closed-loop feedback controller that mimics the reference motion as best as possible within the realistic physics simulation, for eventual deployment on the physical robot. The structure of the RL environment is based on [27] and [28]. We use the Proximal Policy Optimization (PPO) RL algorithm [29], chosen for its relative robustness towards hyperparameter choices, with fast CPU-parallelized simulation rollouts produced through the Raisim physics simulator [30]. We note that the RL environment and all of its hyperparameters are fixed for all motions that we evaluate, and the only variation comes from the different reference motions produced by the trajectory optimizer described in Section III-B, as well as the different control feedforward configurations as described in Section III-C.2.

1) State and Observation: The simulated state of the robot consists of the pose and velocity of the base position and orientation, in addition to the eight joints of the robot. However, only a subset of these variables can be obtained from the physical robot due to sensor limitations. Therefore, the observation for the RL agent consists of the joint angles and angular velocities, as well as the base orientation represented as a quaternion. Note that past work demonstrating dynamic behaviours on the Solo 8 and Solo 12 Robots [23], [31] relied on external motion capture systems to provide high fidelity base pose estimates, which we do not use here. As per [27], we additionally augment the observation with the phase represented as $o_{phase} = \big(\cos(2\pi\phi/T), \sin(2\pi\phi/T)\big)$.

2) Action: Similarly to [28], the action for the RL environment is a residual joint angle, added to the joint angle from the reference motion to provide the final joint angle target to feed into a position-derivative (PD) joint impedance controller. Unlike motion capture data and hand-crafted references used by [27] and [28] respectively, trajectory optimization produces physically feasible joint velocity and torque data. Therefore, we experiment with extending [28] to also include velocity targets and feedforward torques from the reference motion to the joint impedance controller, given as

$$u = k_p(\pi(o,\phi) + \hat{q}(\phi) - q) + k_d(\hat{\dot{q}}(\phi) - \dot{q}) + \hat{\tau}(\phi), \tag{14}$$

where u is the torque command sent to the motors, k_p and k_d denote gains for the joint impedance controller set to 3.0 and 0.3 respectively, $\hat{q}(\phi)$, $\hat{\dot{q}}(\phi)$, and $\hat{\tau}(\phi)$ are phase-indexed joint positions, velocities and torques respectively from the trajectory optimization, $q, \dot{q}$ are the physical joint positions and velocities on the robot, and $\pi(o, \phi)$ is the output of the neural network policy representing the residual joint angle conditioned on observation o and phase ϕ. The feedforward configurations that we experiment with are given in Fig. 3. In the RL simulation, the network output and reference target variables are updated at 50Hz, whereas the physics simulation and motor command torques given by (14) are updated at 1kHz, which differs from the physical robot PD control frequencies as discussed in Section III-D. Note that we do not experiment with the network outputting residual velocities or torques, given that the slow control rate of 50Hz and the noisy outputs that RL policies typically produce are not likely to result in stable behaviours on the physical robot.

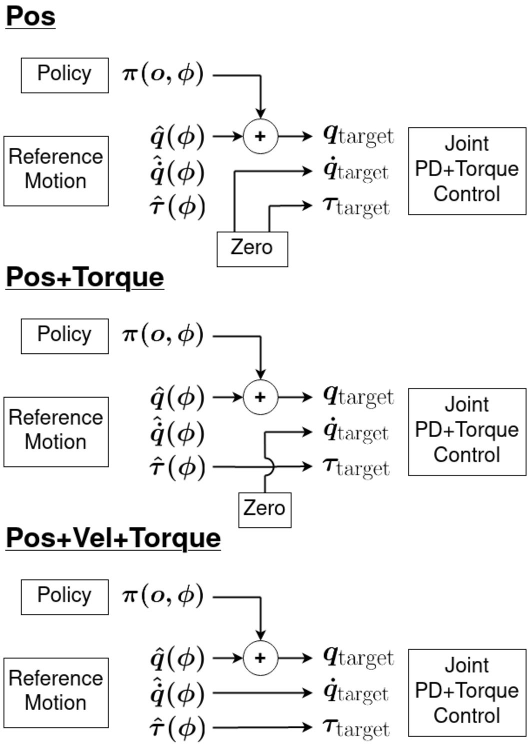

Fig. 3. Feedforward configurations that we experiment with in this work, extending the residual policy used in [28] to include feed-forward velocity and torque targets from the reference motion. Here, *pos* and *vel* are abbreviations for joint position and velocities, respectively.

3) Reward: Following [27], the reward function consists of a weighted sum of Gaussian functions to encourage the agent to mimic the reference motion, while satisfying regularization conditions to facilitate sim-to-real transfer. It has the form

$$r = w_p r_p + w_o r_o + w_j r_j + w_a r_a + w_t r_t, \tag{15}$$

where r_p, r_o, r_j, r_a, and r_t denote reward components for regularization of base position, base orientation, joint angles, action difference, and maximum torque respectively, and w_p, w_o, w_j, w_a, and w_t denote their respective weights chosen to sum to 1. Each reward term r_x has the form

$$r_x = \exp\Big(-\frac{||\hat{x} - x||^2}{2\sigma_x^2}\Big), \tag{16}$$

where $\hat{x}$ and x are the desired and actual values respectively for variable x, and the variance parameter σ_x controls the width of the Gaussian, serving as the tolerance parameter defining the acceptable error in the variable until its corresponding reward term starts to decrease. The values for these hyperparameters are given in Table I. In (15), the position, orientation, and joint rewards are the basic motion imitation rewards with desired values $\hat{x}$ corresponding to that of the reference motion, whereas the action difference and maximum torque rewards are regularizers designed to mitigate specific sim-to-real issues that we observed. The action difference reward penalizes large differences in actions during consecutive RL environment steps to limit vibration and encourage smooth motions [7]. The maximum torque reward penalizes the maximum joint torque observed across all eight joints over the 20ms integration interval between each step, as current spikes were observed to cause faults in the motor controller hardware during experiments, since the power supply units that we used were not dynamic enough to maintain the required voltage during motions involving large impacts with the ground. We believe that these considerations are specific to our particular robot setup, as the joints have no elasticity to absorb impacts, and we are limited by the peak currents of our bench-top power supplies.

4) Initialization and Termination: We use Reference State Initialization [27] during training, i.e., a random ϕ is sampled from $[0, T]$, and the state of the robot is initialized to the corresponding state given by the trajectory optimization. This is important for motions such as the 180-backflip that involve various challenging phases in the motion that must be learned in parallel. Additionally to the reward, we note that designing termination conditions are equally key to practical RL environment design. In this work, we terminate the episode whenever

1) $||\hat{x} - x|| > 2.5\sigma_x$, for all variables x included in the reward function (15),
2) a non-foot body of the robot contacts the ground, or
3) the contact state of a foot does not match that of the reference motion, with a tolerance window of 120ms around contact transitions where either contact state is allowed.

Termination condition 1 prevents the training from converging to a motion where an agent completely ignores a reward component and allows its value to be zero, condition 2 prevents unwanted contacts with the ground, and condition 3 prevents local minima where the agent never lifts its feet off the ground for motions such as the front-hop, where doing so incurs a large risk when the motion involves statically unstable poses with limited observability.

5) Dynamics Randomization: Dynamics randomization is a common technique to prevent the RL agent from overfitting to the physics model used to train the policy, which always has unmodelled effects and other inaccuracies compared to the actual dynamics exhibited on the physical robot, in addition to producing a policy that is generally more robust for real-world deployment [15], [16], [9], [8], [32]. Following the recommendations provided by [16], we only randomize physical parameters that we could not reliably measure, or were believed to be inaccurate in our simulation model. These include the ground coefficient of friction and restitution, clipped to be within $[0.1, 1.0]$ and $[0.0, 1.0]$ respectively, joint position offsets, and joint torque scaling. The joint position offset randomization adds a different random offset angle to each joint to simulate the inaccuracy of the calibration procedure used to specify the zero position for the relative encoders in each joint, and the joint torque scaling offset multiplies the joint torque command calculated by (14) by some constant factor. This is to account for the differences in PD+feedforward torque control frequencies between the RL simulation and the real robot as discussed in Section III-D, in addition to any additional imperfections that may exist in the motor control loop that we do not attempt to model in detail such as in [8]. All randomizations are performed by sampling from a normal distribution, with mean and standard deviation values given in Table I.

TABLE I

HYPERPARAMETERS USED FOR RL TRAINING

Reward		
Reward Component	w_x	σ_x
Position	0.3	0.05
Orientation	0.3	0.14
Joint	0.2	0.3
Action Difference	0.1	0.35
Maximum Torque	0.1	3.0

Dynamics Randomization		
Randomization Variable	μ	σ
Friction	0.8	0.25
Restitution	0.0	0.25
Joint Offset	0.0	0.02
Torque Scale	1.0	0.1

D. Sim-to-Real Transfer for the Solo 8 Robot

In this work, we built an Open Dynamics Robot Initiative Solo 8 Robot [33] to use as the platform to test our methods. In addition to being a lightweight and power-dense robot suitable for researching dynamic motions, the ability to easily repair and replace components ourselves was instrumental for testing motions with large impacts and high probability for damage, without downtime from sending robots back to manufacturers for maintenance. To test policies on the robot, we tethered it to an external workstation running real-time C++ control code at 1kHz, requiring only the reference motion file and a compact neural network converted to a PyTorch TorchScript model [34] to specify the motion to run.

Similarly to the RL simulation, target joint positions, velocities, and torques are sent to the robot from the workstation at 50Hz. Unlike the simulation however, the PD feedback control is calculated on a microcontroller onboard the robot at 10kHz, enabling highly stable PD control with the caveat of potentially producing different motor behavior from the simulation, which we mitigate through dynamics randomization.

IV. RESULTS

A. Trajectory Optimization

To evaluate our framework, the trajectory optimizer was used to produce four motions of varying difficulty and style– *trot*, *front-hop*, *180-backflip*, and *biped-step*. In addition to the objective and constraints outlined in Section III-B, we constrain leg motions to be symmetric for the *trot*, *front-hop*, and *180-backflip* motions. We additionally constrain the dynamics for the *biped-step* motion to lie in the sagittal plane, and rely on the subsequent RL-based tracking to realize the required underactuated 3D stepping behavior for the Solo 8. These motion-specific modifications can be iterated quickly through trajectory optimization (c.f. Table II), compared to RL training which can be slow and unintuitive to tune. Table II was obtained by taking the mean and standard deviation of solve times over running the corresponding trajectory optimization problem five times on a 12-core 3.8GHz workstation.

TABLE II

SOLVE TIMES FOR TRAJECTORY OPTIMIZATION

Motion Type	Trajectory Duration	Solve Time (s) $(N = 5)$
Trot	10s	286 ± 3.43s
Front-hop	5s	44.8 ± 0.162s
180-backflip	5s	107 ± 0.475s
Biped-Step	10s	50.7 ± 0.522s

B. RL Training

Using the imitation-based RL training environment outlined in Section III-C, we trained network controllers for all four motions. Although the velocity and feedforward torques could in principle improve training performance through feeding it directly to the PD+torque controller through the feedforward configurations illustrated in Fig. 3, we did not observe significant differences in RL training speeds across the feedforward configurations considered. In contrast, we observed a large difference in training performance through using reference motions from trajectory optimization as opposed to the kinematic motion sketch used as the initial guess for the optimization, as our RL environment completely failed to train the 180-backflip motion using the kinematic sketch trajectory.

C. Experiments on the Physical Robot

Once the four motion types were trained in RL, they were loaded onto the physical robot for testing. As shown in the accompanying video, the *trot*, *front-hop*, and *180-backflip* motions are able to transfer onto the physical robot, whereas the *biped-step* motion can only be executed on the robot for up to 5 seconds before the robot falls over. Since the

Solo 8 robot has only point feet and does not have any abduction DOF in the legs, the trained motion relies critically on the accuracy of the underactuated dynamics and upper body inertia. Additionally, we observe significant sideways flex in the lower legs of the robot when it is standing on one leg, whose compliance due to its plastic material serves as a major mismatch between the rigid simulation model and the physical robot. Despite these challenges, the outlined framework is able to reproduce these four behaviours of varying difficulty and style. Of the feedforward configurations we consider, we find that using velocity targets in our controller degraded the sim-to-real performance, making the robot more stiff and producing harder impacts. More problematically, we found that using velocity targets made the current spike issue described in Section III-C.3 far more common, likely due to the large target joint velocities produced by the post-processing phase of the trajectory optimizer. This is discussed in further detail in Section V-C. In examining motions from the robot, we observe that feedforward torques provide active gravity compensation during static standing, enabling the PD controller to closely track target values. In contrast, for the position-only configuration, there is a large constant offset between actual and target angle values which must be accurately learned by the RL training in order to effectively regulate pose and ground reaction forces.

V. Discussion

The results demonstrated in this work illustrates how trajectory optimization can be combined with RL to produce a variety of dynamic behaviours. In this section, we discuss the various advantages, limitations, and design considerations that arise with the methods considered.

A. RL Environment Generality

In comparison to prior work using trajectory optimization to produce reference motions for robotics RL training [17], [7], [20], [21], we demonstrate successful sim-to-real training of a variety of motions, all with identical RL environments and hyperparameters. This demonstrates a fundamental advantage of using trajectory optimization to produce reference motions, by allowing faster iteration than is afforded by tuning RL environments specifically to each task.

B. Model Choice for Trajectory Optimization

In this work, we opted to use the SRB as the dynamics model for the trajectory optimizer, as this gave us a good trade-off between simplicity and flexibility. While it is an effective tool for generating nearly feasible motions, it is typically not sufficient to run open-loop on a robot due to the many simplifications that it makes, such as the legs being massless. However, we demonstrate in this work how RL can be used to bridge this gap and to adapt the motion for a full-order robot model, while leveraging the SRB solutions.

C. Feedforward Configuration for Sim-to-Real

Using the full position+velocity+torque information produced by the trajectory optimizer improved training performance for some motions, but not for others, as compared to simply using the positions. We further observed that the full feed-forward information did not improve performance on the physical robot, with position and position+torque feed-forward information being approximately equally capable. We identified that using the target velocity was problematic because of current spikes (c.f. Section III-C.3) as well as making the robot more stiff and less reactive to perturbations.

D. Fixed timing

A major limitation of approach used in this work is the fixed timing of the motions that cannot by modified be the RL training to better fit the capabilities of the full-order robot model and to react to unexpected contacts, due to the phase variable ϕ, for which the reference motion and network are conditioned on, being incremented by a fixed external clock. This is in contrast to [17], which used a similar approach to ours but had an additional second RL training phase to remove this time dependency. Although there exists past work demonstrating sim-to-real RL training of behaviours with fixed timing [16], [7], [15], these works mainly focused on locomotion tasks without significant flight phases, where this issue may be less prevalent.

E. State Estimation and Sensor History

Another major limitation of this work is the lack of state estimation, and the generally small number of observation variables used to train and deploy the RL policy. A large body of previous literature on RL robotic locomotion relied on model-based or learned state estimators to produce base velocity and height estimates to input into the policy [9], [28], [7], [16], [32], or used a history of sensor readings to replace or augment state estimation [9], [8], [15]. Since the focus of this work is the control of motion variety and how this is affected by feedforward configuration, we limited observation variables to the bare minimum of joint positions and velocities, as well as the high quality orientation estimates produced by the Extended Kalman Filter onboard the Inertial Measurement Unit that we used. Our results may be improved if our policy was additionally conditioned on the output of a state estimator or sensor history.

VI. Conclusion

In this work, we investigated the use of trajectory optimizers to produce reference motions based on an SRB model, followed by imitation-based RL tracking to realize these motions on a full-order model for deployment on a physical Solo 8 robot. We characterized the various design decisions that arise when combining model-based trajectory optimization and model-free RL, with particular emphasis on analyzing feedforward configurations. In addition to achieving further motion variety beyond four motions and addressing the limitations discussed in the previous section, future work could explore training a single RL policy that can imitate a wide variety of reference motions via direct conditioning on those motions. This could be combined with a high level planner such that the RL policy acts as a learned whole body controller.

REFERENCES

[1] A. W. Winkler, C. D. Bellicoso, M. Hutter, and J. Buchli, "Gait and trajectory optimization for legged systems through phase-based end-effector parameterization," *IEEE Robotics and Automation Letters*, vol. 3, no. 3, pp. 1560–1567, 2018.

[2] Z. Manchester and S. Kuindersma, "Variational contact-implicit trajectory optimization," in *Robotics Research*. Springer, 2020, pp. 985–1000.

[3] I. Mordatch, E. Todorov, and Z. Popović, "Discovery of complex behaviors through contact-invariant optimization," *ACM Transactions on Graphics (TOG)*, vol. 31, no. 4, pp. 1–8, 2012.

[4] C. Mastalli, R. Budhiraja, W. Merkt, G. Saurel, B. Hammoud, M. Naveau, J. Carpentier, L. Righetti, S. Vijayakumar, and N. Mansard, "Crocoddyl: An efficient and versatile framework for multi-contact optimal control," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2020, pp. 2536–2542.

[5] M. Giftthaler, M. Neunert, M. Stäuble, J. Buchli, and M. Diehl, "A family of iterative gauss-newton shooting methods for nonlinear optimal control," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2018, pp. 1–9.

[6] Y. Ding, A. Pandala, and H.-W. Park, "Real-time model predictive control for versatile dynamic motions in quadrupedal robots," in *2019 International Conference on Robotics and Automation (ICRA)*, 2019, pp. 8484–8490.

[7] K. Green, Y. Godse, J. Dao, R. L. Hatton, A. Fern, and J. Hurst, "Learning spring mass locomotion: Guiding policies with a reduced-order model," *IEEE Robotics and Automation Letters*, vol. 6, no. 2, pp. 3926–3932, 2021.

[8] 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.

[9] J. Lee, J. Hwangbo, L. Wellhausen, V. Koltun, and M. Hutter, "Learning quadrupedal locomotion over challenging terrain," 2020. [Online]. Available: https://robotics.sciencemag.org/content/5/47/eabc5986

[10] 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.

[11] 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.

[12] Z. Xie, X. Da, B. Babich, A. Garg, and M. van de Panne, "Glide: Generalizable quadrupedal locomotion in diverse environments with a centroidal model," *arXiv preprint arXiv:2104.09771*, 2021.

[13] W. Yu, D. Jain, A. Escontrela, A. Iscen, P. Xu, E. Coumans, S. Ha, J. Tan, and T. Zhang, "Visual-locomotion: Learning to walk on complex terrains with vision," in *Conference on Robot Learning*. PMLR, 2022, pp. 1291–1302.

[14] G. B. Margolis, T. Chen, K. Paigwar, X. Fu, D. Kim, S. bae Kim, and P. Agrawal, "Learning to jump from pixels," in *Conference on Robot Learning*. PMLR, 2022, pp. 1025–1034.

[15] X. B. Peng, E. Coumans, T. Zhang, T.-W. Lee, J. Tan, and S. Levine, "Learning agile robotic locomotion skills by imitating animals," *arXiv preprint arXiv:2004.00784*, 2020.

[16] Z. Xie, X. Da, M. van de Panne, B. Babich, and A. Garg, "Dynamics randomization revisited: A case study for quadrupedal locomotion," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2021, pp. 4955–4961.

[17] M. Bogdanovic, M. Khadiv, and L. Righetti, "Model-free reinforcement learning for robust locomotion using trajectory optimization for exploration," *arXiv preprint arXiv:2107.06629*, 2021.

[18] A. Escontrela, X. B. Peng, W. Yu, T. Zhang, A. Iscen, K. Goldberg, and P. Abbeel, "Adversarial motion priors make good substitutes for complex reward functions," *arXiv preprint arXiv:2203.15103*, 2022.

[19] E. Vollenweider, M. Bjelonic, V. Klemm, N. Rudin, J. Lee, and M. Hutter, "Advanced skills through multiple adversarial motion priors in reinforcement learning," *arXiv preprint arXiv:2203.14912*, 2022.

[20] 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.

[21] P. Brakel, S. Bohez, L. Hasenclever, N. Heess, and K. Bousmalis, "Learning coordinated terrain-adaptive locomotion by imitating

a centroidal dynamics planner," 2021. [Online]. Available: https://arxiv.org/abs/2111.00262

[22] Z. Xie, P. Clary, J. Dao, P. Morais, J. Hurst, and M. Panne, "Learning locomotion skills for cassie: Iterative design and sim-to-real," in *Conference on Robot Learning*. PMLR, 2020, pp. 317–329.

[23] C. Li, M. Vlastelica, S. Blaes, J. Frey, F. Grimminger, and G. Martius, "Learning agile skills via adversarial imitation of rough partial demonstrations," *arXiv preprint arXiv:2206.11693*, 2022.

[24] A. Wächter and L. Biegler, "On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming," *Mathematical Programming*, vol. 106, no. 1, pp. 25–57, May 2006, copyright: Copyright 2008 Elsevier B.V., All rights reserved.

[25] J. A. E. Andersson, J. Gillis, G. Horn, J. B. Rawlings, and M. Diehl, "CasADi – A software framework for nonlinear optimization and optimal control," *Mathematical Programming Computation*, vol. 11, no. 1, pp. 1–36, 2019.

[26] H. Dai, A. Valenzuela, and R. Tedrake, "Whole-body motion planning with centroidal dynamics and full kinematics," in *2014 IEEE-RAS International Conference on Humanoid Robots*, 2014, pp. 295–302.

[27] X. B. Peng, P. Abbeel, S. Levine, and M. van de Panne, "Deepmimic: Example-guided deep reinforcement learning of physics-based character skills," *ACM Trans. Graph.*, vol. 37, no. 4, pp. 143:1–143:14, Jul. 2018. [Online]. Available: http://doi.acm.org/10.1145/3197517.3201311

[28] Z. Xie, G. Berseth, P. Clary, J. Hurst, and M. van de Panne, "Feedback control for cassie with deep reinforcement learning," 2018.

[29] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *CoRR*, 2017. [Online]. Available: http://arxiv.org/abs/1707.06347

[30] J. Hwangbo, J. Lee, and M. Hutter, "Per-contact iteration method for solving contact dynamics," *IEEE Robotics and Automation Letters*, vol. 3, no. 2, pp. 895–902, 2018. [Online]. Available: www.raisim.com

[31] M. Bogdanovic, M. Khadiv, and L. Righetti, "Model-free reinforcement learning for robust locomotion using demonstrations from trajectory optimization," *Frontiers in Robotics and AI*, vol. 9, 2022. [Online]. Available: https://www.frontiersin.org/articles/10.3389/frobt.2022.854212

[32] M. Aractingi, P.-A. Léziart, T. Flayols, J. Perez, T. Silander, and P. Souères, "Controlling the Solo12 Quadruped Robot with Deep Reinforcement Learning," Aug. 2022, working paper or preprint. [Online]. Available: https://hal.laas.fr/hal-03761331

[33] F. Grimminger, A. Meduri, M. Khadiv, J. Viereck, M. Wüthrich, M. Naveau, V. Berenz, S. Heim, F. Widmaier, T. Flayols, J. Fiene, A. Badri-Spröwitz, and L. Righetti, "An open torque-controlled modular robot architecture for legged locomotion research," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 3650–3657, 2020.

[34] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds. Curran Associates, Inc., 2019, pp. 8024–8035. [Online]. Available: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf