Guided Constrained Policy Optimization for Dynamic Quadrupedal Robot Locomotion

Siddhant Gangapurwala, Alexander Mitchell and Ioannis Havoutis

***Abstract*—Deep reinforcement learning (RL) uses model-free techniques to optimize task-specific control policies. Despite having emerged as a promising approach for complex problems, RL is still hard to use reliably for real-world applications. Apart from challenges such as precise reward function tuning, inaccurate sensing and actuation, and non-deterministic response, existing RL methods do not guarantee behavior within required safety constraints that are crucial for real robot scenarios. In this regard, we introduce guided constrained policy optimization (GCPO), an RL framework based upon our implementation of constrained proximal policy optimization (CPPO) for tracking base velocity commands while following the defined constraints. We introduce schemes which encourage state recovery into constrained regions in case of constraint violations. We present experimental results of our training method and test it on the real ANYmal quadruped robot. We compare our approach against the unconstrained RL method and show that guided constrained RL offers faster convergence close to the desired optimum resulting in an optimal, yet physically feasible, robotic control behavior without the need for precise reward function tuning.**

***Index Terms*—Deep Learning in Robotics and Automation, AI-Based Methods, Legged Robots, Robust/Adaptive Control of Robotic Systems, Underactuated Robots**

I. Introduction

LEGGED locomotion has been an active area of robotics research over the past few decades. Despite our best efforts, achieving extraordinarily dynamic robotic behavior still remains an open problem. Most of the existing work has focused on the use of traditional model-based control techniques, such as offline trajectory optimization (TO) [1] and online model predictive control (MPC) [2] which, due to their mathematical complexity, are often based on simplified models of the systems. Such simplifications result in control solutions that are often mechanically limiting and inefficient.

Considering robotic locomotion as a reinforcement learning (RL) problem [3] offers a model-free data-driven alternative to model-based control. Although RL has witnessed significant contributions from researchers to address issues such as sample inefficiency [4] and hyperparameter tuning [5], it still faces significant challenges to be used for real-world robotic locomotion applications mainly due to no hard guarantees on safety-critical constraints.

Manuscript received: September, 10, 2019; Revised December, 17, 2019; Accepted January, 19, 2020.

This paper was recommended for publication by Editor Nikos Tsagarakis upon evaluation of the Associate Editor and Reviewers' comments. This work was supported by the UKRI/EPSRC RAIN and ORCA Hubs [EP/R026084/1, EP/R026173/1], Robust Legged Locomotion [EP/S002383/1] and the EU H2020 Projects MEMMO and THING. It was conducted as part of ANYmal Research, a community to advance legged robotics.

The authors are with the Dynamic Robots Systems Group, Oxford Robotics Institute, University of Oxford, UK. Email: {siddhant,ioannis}@robots.ox.ac.uk.

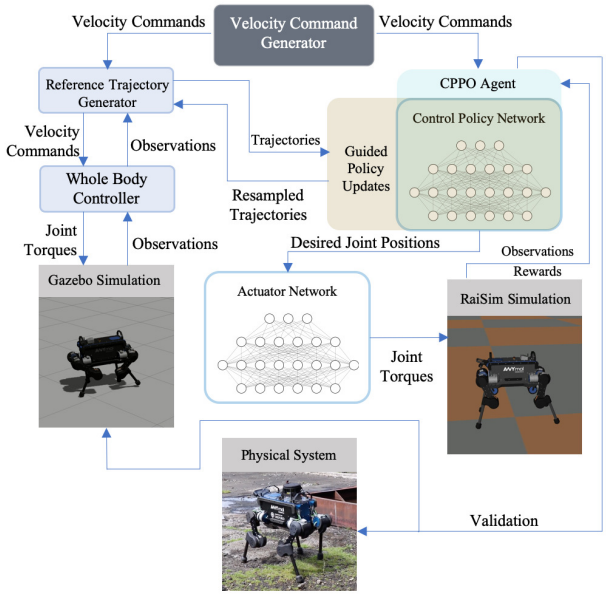

Fig. 1: Overview of our training and validation process. Accompanying video can be found at: https://youtu.be/iPDmG9knkLs

In this work, we develop an RL problem formulation that introduces constraints based on optimal control techniques. We train a control policy, using our constrained proximal policy optimization (CPPO) method based upon proximal policy optimization (PPO) [5], for tracking user-generated reference base velocities on the ANYmal [6] quadruped, a 33 kg legged robot. We experimentally validate its performance in comparison with unconstrained training procedures in a physically realistic simulation environment and on the real ANYmal robot.

A. Related Work

Control architectures for robotic quadrupedal locomotion have seen various forms. One of the common approaches leverages mathematical optimization techniques [7] to generate reference trajectories by solving an optimal control (OC) [8] problem with objectives such as minimization of energy consumption, and constraints that consider the dynamics of the robotic systems. Authors of [9] presented a TO formulation for legged locomotion that automatically generates reference motions without requiring any prior footstep planning.

Extending upon OC, some of the work has focused on formulating locomotion as multiple tasks [10], such as maintaining robot stability and tracking desired limb motions, solved by prioritizing each individual task using quadratic programming [11] solvers. This principle of sub-dividing tasks

into simpler problems has also been followed in some of the deep RL research [12].

Model-based control methods often use simplified mechanical models to ease on the mathematical complexity of the problem. For example, most formulations consider the robotic system as a point mass with massless limbs. These approximations result in control solutions that cannot exploit the full range of capabilities of the systems. Moreover, several model-based controllers require hand-tuned costs by human-experts which are specific to each system's task thereby limiting their generality.

Deep RL methods attempt to address some of these limitations of model-based control by employing model-free techniques which optimize over a control policy, a neural network which maps states into actions, so as to maximize a task-specific reward signal by means of trial and error. Such methods have been investigated for legged locomotion tasks [13], [14] but have mainly been demonstrated in simulations using unrealistic robot models, e.g. ideal torque sources, infinite velocity/torque ranges. Moreover, RL techniques usually require large amounts of data making training on a real robot infeasible. This necessitates the use of physics simulators. However, policies trained using simulations often do not transfer for real world tasks since the reality gap between simulations and the physical world are strongly pertinent. These issues of RL have been tackled using techniques such as actuator modeling [15], implementing a modular training approach [16], introducing domain randomization [17], increasing policy generalization [18] and adding noise to observations and actions in the training environments. Authors of [15] have demonstrated the use of a deep RL approach to complex legged locomotion tasks. Our work, extends upon their training methods to a constrained learning approach as discussed in section III.

Despite the success of deep RL approaches on real-world robotic applications, one of the main challenges in solving an RL problem is precise reward function tuning. As a solution, an inverse reinforcement learning (IRL) [19] problem can be characterized as: given reference trajectories of an agent in various circumstances, determine the reward function to be minimized. This reward function is then used to solve an RL problem. The authors of [20] and [21] have successfully implemented IRL methods for perception and control tasks, however, the need for the extra step of solving an RL problem adds to training delays. Instead, designing the problem as that of behavioral cloning (BC) [22] gets rid of the reward recovery step, and directly optimizes over a policy given reference demonstrations. Along similar lines, guided policy search (GPS) [23] techniques can be used to reduce training times by directing policy learning in turn avoiding poor local optima.

Model-based RL [24] techniques, which require a knowledge of system dynamics, have also been proposed as an approach to boost convergence along desired optima. In the pursuit of making RL methods desirable for use in safety critical systems, methods such as constrained policy optimization (CPO) [25] have also been investigated to ensure that an RL control policy obeys the necessary safety constraints during operation.

B. Contributions

Our work extends upon the above research to realize an RL problem formulation that considers the constraints required to guarantee the stability of a quadrupedal robot system. Furthermore, our problem formulation introduces constraints such as end-effector boundaries, joint velocity limits, and joint acceleration limits that direct policy optimization towards a desired quadrupedal locomotion behavior. This constrained formulation, coupled with techniques motivated by BC and GPS, which in our case is guided policy updates (GPUs), further results in the reduction of training time while also eliminating the need for precise reward function tuning.

We also present the importance of the setup of an RL environment, and show how differences in dynamic properties can result in a significantly different learnt behavior. We also compare different physics simulation frameworks and detail upon the motivations of preferring one over the others.

We introduce several schemes that make the quadrupedal system more robust and therefore better suited for use in real-world applications. Since we do not use approximations required for model-based control, our learnt policies better utilize system dynamics to generate efficient locomotion behavior requiring significantly lesser torque compared to a model-based trot controller. We successfully transfer the control policy trained in a simulator to the real system and further provide evidence of dynamic behavior of the control policy by testing its response after changing the physical properties of the real system, and also by continuously varying control step times.

II. Approach

In this section we describe the RL methods we use for the task of quadrupedal locomotion.

A. Algorithm

Based on the framework of Markov decision processes (MDPs) [26], a constrained Markov decision process (CMDP) [27] is defined as a tuple (S, A, R, C, P, d, μ), where S is the set of states, A is the set of actions, $R : S \times A \times A \to \mathbb{R}$ is the reward function, $P : S \times A \times S \to [0,1]$ is the state transition probablity, and μ is the starting state distribution as chracterized in the MDP tuple. CMDPs augment the MDP with a set C of cost functions, $C_1, ..., C_m$, with $C_i : S \times A \times S \to \mathbb{R}$, and limits $d_1, \ldots, d_m$ as described in [25]. Being consistent with the definitions and notation used by the authors of [25], a stationary policy $\pi : S \to \mathcal{P}(A)$ is defined as a function mapping states to probability distributions over actions. The set of stationary policies is defined as Π. $\pi(a|s)$ denotes the probability of selecting action a in state s.

Given a performance measure,

$$J(\pi) \doteq \mathop{\mathrm{E}}_{\tau\sim\pi}\left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t, s_{t+1})\right],$$

where $\gamma \in [0,1)$ is the discount factor, τ denotes a trajectory dependent on π, we aim to select a policy π which maximizes $J(\pi)$. For a CMDP, the expected discount cost return $J_{C_i}(\pi) \doteq \mathop{\mathrm{E}}_{\tau\sim\pi}[\sum_{t=0}^{\infty} \gamma^t C_i(s_t, a_t, s_{t+1})]$ for a policy π with

cost function C_i. The set of feasible stationary policies $\Pi_C \doteq \{\pi \in \Pi : \forall_i, J_{C_i}(\pi) \leq d_i\}$.

The RL problem is then expressed as

$$\pi^* = \arg\max_{\pi \in \Pi_C} J(\pi).$$

For a policy π_θ parameterized with θ, most policy optimization strategies iteratively update the base policy using local policy search methods [28] by maximizing $J(\pi)$ over a trust region [29]. For a CMDP, policy iteration using trust regions [25] can be expressed as

$$\begin{aligned} \pi_{k+1} &= \arg\max_{\pi \in \Pi_\theta} \mathop{\mathrm{E}}_{\substack{s \sim d^{\pi_k} \\ a \sim \pi}} [A^{\pi_k}(s,a)] \\ \text{subject to } & J_{C_i}(\pi_k) + \frac{1}{1-\gamma} \left(\mathop{\mathrm{E}}_{\substack{s \sim d^{\pi_k} \\ a \sim \pi}} \left[A^{\pi_k}_{C_i}(s,a) \right] \right) \leq d_i \quad \forall_i \\ & \bar{D}_{KL}(\pi || \pi_k) \leq \delta. \end{aligned} \tag{1}$$

where $\bar{D}_{KL} = \mathop{\mathrm{E}}_{s \sim \pi_k} [D_{KL}(\pi || \pi_k)[s]]$, and $\delta > 0$ is a step size. D_{KL} refers to the Kullback-Leibler divergence. Authors of [25] show that developing CPO as a trust region method implies CPO inherits the performance guarantee given by certain lower bound detailed in [25]. The authors of [25] also define the worst-case upper bound on the cumulative discounted return for a CMDP.

The PPO technique introduces a clipped objective function

$$L^{CLIP}_t(\theta) = \mathrm{E}_t \left[\min \left(r_t(\theta) A_t, \mathrm{clip}\left(r_t(\theta), 1-\varepsilon, 1+\varepsilon\right) A_t \right) \right]$$

where ε is a hyperparameter. In our implementation, we introduce an approximation of the constraint expressed in (1) to the above objective, and rewrite it as

$$L^{CCLIP}_t(\theta) = L^{CLIP}_t(\theta) - \sum_i \zeta_i J_{C_i,t}(\pi_\theta), \tag{2}$$

where ζ_i is an experimentally tuned hyperparameter. The objective L^{CLIP}_t is often augmented to include a value function loss term L^{VF}_t and an entropy term S [5]. The objective function, with coefficients c_1 and c_2 is then

$$\begin{aligned} & L^{CCLIP+VF+S}_t(\theta) \\ & = \mathrm{E}_t \left[L^{CCLIP}_t(\theta) - c_1 L^{VF}_t(\theta) + c_2 S[\pi_\theta](s_t) \right]. \end{aligned} \tag{3}$$

We introduced this approximation as, in our experiments, we observed that the constrained policy optimization objective was extremely sample inefficient. We observed no improvements in our training even after 5 billion sampling steps for the task of tracking base velocity commands. As a solution, we implement an approximated constrained proximal policy optimization (aCPPO) method along with generalized advantage estimate (GAE) [30], and optimize over the loss function represented in (3). After convergence, we perform hard constrained proximal policy optimization (hCPPO) using the objective $L^{CLIP+VF+S}_t$ and by introducing the cost return constraint expressed in (1). We collectively refer to both these optimization steps as constrained proximal policy optimization (CPPO).

In our work, we introduce three degrees of constraints and handle them accordingly:

1) *Soft* (ρ) constraints are included as part of the reward function. These need not be critical for safe operations. Instead these are introduced in order to direct policy search towards a desired behavior.
2) *Hard* (κ) constraints cannot be violated and are included in the set of constrained cost functions C. These are directly included during policy updates. In case of aCPPO, these are included in (2), and for hCPPO these are included as a constraint, as shown in (1) for the objective $L^{CLIP+VF+S}_t$.
3) *No-go* (η) constraints are introduced during training such that when κ-constraints are violated beyond a certain threshold the training episode is terminated to prevent exploration around regions which do not contribute towards policy optimization.

We use guided policy updates to warm start our control policy and then perform constrained proximal policy optimization for policy exploration. We then alternate between these during training as described in Alg. 1.

Algorithm 1 Guided Constrained Policy Optimization

Input: λ_d, s_{max}, c_{max}, n_{steps}, n_{batch}, in_{user}, c^h_{max}, H_a, H_h
Initialize θ, $\alpha = 1$
Generate set of trajectories $\mathcal{D} = \textsc{Controller}(in_{user})$ ▷ using model-based control strategy
for $t = 0, 1, 2 \ldots H_a$ **do**
  $\theta, \mathcal{D} = \textsc{GuidedPolicyUpdate}(\theta, \mathcal{D}, \alpha s_{max}, n_{batch})$
  $\theta = \textsc{PolicyOptimization}(\theta, n_{steps}, (1-\alpha)c_{max}, false)$
  $\alpha = \mathrm{e}^{\lambda_d t}$
end for
for $t = 0, 1, 2 \ldots H_h$ **do**
  $\theta = \textsc{PolicyOptimization}(\theta, n_{steps}, c^h_{max}, true)$
end for
function GUIDEDPOLICYUPDATE(θ, $\mathcal{T}$, it_{max}, n_{batch})
  $l_{batch} = it_{max}/n_{batch}$
  Sample l_{batch} state-action pairs (s, a^*) from $\mathcal{T}$
  for $i = 0$ **to** $n_{batch} - 1$ **do**
    Generate $\{a\}$ for $\{s\}$ using π_θ
    Update θ by minimizing $\sum_{j=0}^{l_{batch}} ||a_j - a^*_j||^2$
  end for
  $\mathcal{T} = \mathcal{T} \setminus \{(s, a^*)\}$
  return θ, $\mathcal{T}$
end function
function POLICYOPTIMIZATION(θ, $l_{episode}$, it_{max}, hard)
  $it = 0$, $\tau = \{\}$
  while $it < it_{max}$ **do**
    for $t = 0$ **to** $l_{episode} - 1$ **do**
      Sample $a_t \sim \pi_\theta(a|s_t)$
      $s_{t+1}, r_t, \{c_t\} = \textsc{RLEnvironmentStep}(a_t)$
      $it = it + 1$, $\tau = \tau \cup (s_{t+1}, r_t, \{c_t\})$
    end for
    if hard == $false$
      Update θ using aCPPO
    else
      Update θ using hCPPO
    end if then
  end while
  return θ
end function

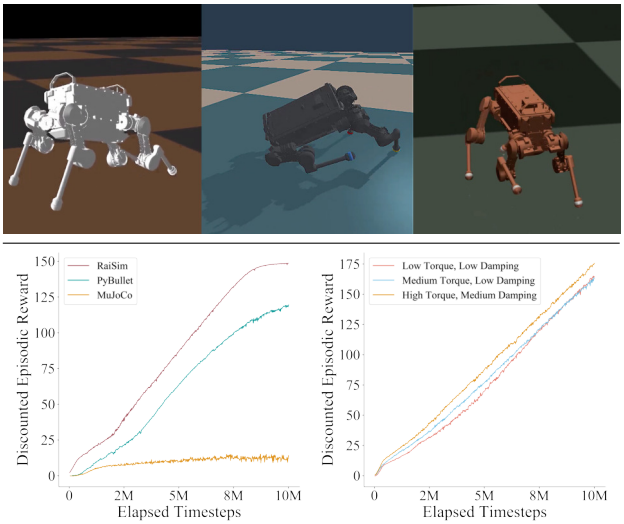

Fig. 2: Comparing differences in learnt behavior for different actuator models. **(top)** *left:* the RaiSim environment manages to sprint forward using a gait similar to galloping, *center:* the PyBullet environment converges at a local optimum where it uses hind legs to push its body forward while the front legs are not used, *right:* the MuJoCo environment struggles to find an optimal policy mainly because the desired joint positions at each step are not tracked reliably. **(bottom)** *left:* average discounted reward curves observed during training, *right:* average discounted reward curves observed during training in RaiSim for different ANYmal actuator models.

B. Simulation

Most RL algorithms are sample inefficient and require significant amount of trials to learn a desirable control policy. Instead of training on a physical platform, which is slow and unsafe, we train the locomotion policy on a significantly faster simulation environment. However, policies trained in simulations often do not perform well in real-world systems. This is mostly due to the reality gap associated with simulations which do not perfectly model the physical world. Moreover, while performing experiments we realized different actuator models for ANYmal resulted in considerably different behaviors for the same training parameters.

We tested ANYmal simulations in RaiSim [31], PyBullet [32] and MuJoCo [33] for a simple task of moving forward with maximum feasible base velocity in order to compare the generated behaviors and training simulation times. The input to the control policy (34-dimensional state vector) consisted of $\{base_{height}, base_{orientation}, base_{twist}, joint_{states}\}$, and the output (12-dimensional action vector) consisted of $\{joint_{positions}\}$ desired for the next state. We trained the policies using PPO with the same hyperparameters, and on the same device, using the reward function $0.3 \times base_{forwardVel} - 4e{-}5 \times \|joint_{torque}\|^2$.

We trained the policies for up to 10M time steps with each iteration comprising of 76.8k episodic step samples. Using a discount factor $\gamma = 0.998$, and maximum episode length of 6.4k simulation steps we achieved the results represented in Fig. 2.

We performed experiments using different simulators and actuator models to validate our point that learnt behavior significantly depends on the setup of the RL environment, further substantiated by running experiments in RaiSim using different feed-forward torque and damping parameters for the actuators, as represented in Fig. 2. Moreover, none of the policies trained with different actuator models were observed to be inherently stable, necessitating the use of a good actuator model. For this, we used the same technique as authors of [15] to approximate the actuator model using a neural network trained through supervised learning. We also used the same network architecture as used in [15] for training the actuator network.

Moreover, due to sample inefficiency in most RL algorithms, it is important to consider time required for each simulation step. In our experiments, we observed that for same number of parallel executions, RaiSim was faster than both PyBullet and MuJoCo as shown in Table I. With several more parallel executions possible across multiple threads, we managed to execute 1B simulation steps in less than 3 hours in RaiSim on a PC.

C. Environment Setup

Authors of [15] and [12] provide competitive baselines. We extend their approach to constrained policy optimization utilizing some of the hyperparameters used in their work. In this section we describe the setup of the ANYmal RL environment for the task of tracking user-generated reference base velocity commands.

1) Observation Space: In order to be extendable to the physical robot, the observation space chosen for the ANYmal environment needs to be accessible through on-board sensors and state-estimators. In this regard, the 109-dimensional state vector for the RL environment is defined as $\left\{b_h, \mathcal{O}, v_{base}, \omega_{base}, \mathcal{J}_t, \mathcal{J}^{des}_{t-1,t-2,t-3,t-4}, \dot{\mathcal{J}}_{t,t-1,t-2}, \mathcal{V}_{base}\right\}$ where b_h is the robot base height, $\mathcal{O}$ is the base orientation, v_{base} is the linear velocity in base frame, ω_{base} is the angular velocity in base frame, $\mathcal{J}_t$ is the joint position at time t, $\mathcal{J}^{des}_t$ is the policy output at time t, $\dot{\mathcal{J}}_t$ is the joint velocity at time t and $\mathcal{V}_{base}$ is the user-generated desired base velocity expressed in base frame.

2) Action Space: The control policy outputs a 12-dimensional action vector comprising of $\{\mathcal{J}^{des}_t\}$. The desired joint positions are forwarded as an input to the approximated actuator network which outputs the torques for each of the joints for the ANYmal quadruped. These torques, clipped between $[-35\ Nm, 35\ Nm]$, are then directly applied to the joints.

3) Network Architecture: Since our work focuses on the constrained RL formulation, we decided to use the same network architecture implemented in [15], which had been already demonstrated to perform well.

4) Reward Terms: The reward terms are shown in Table II. These terms are multiplied by coefficients which are scaled further to increase the difficulty for the RL agent as training progresses.

5) ρ-Constraint Costs: These costs are directly added to the reward function, and are as shown in Table III.

TABLE I: Training time required for executing 10M simulation steps using 12 parallel environment runs tested on a PC housing an Intel i7-8700K and an Nvidia RTX 2080Ti.

	RaiSim	**PyBullet**	**MuJoCo**
Training Time (seconds)	1031.6403	2043.9825	1820.8244

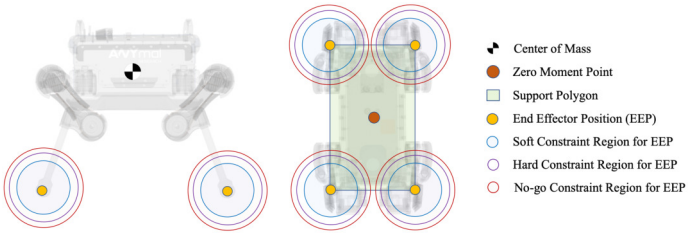

Fig. 3: Side-view and top-view of the ANYmal quadruped schematics representing some of the parameters used for constraint costs.

6) κ-Constraint Costs: These hard constraints are directly introduced in the aCPPO formulation as part of the expected discount cost return J_{C_i} for cost term i. We introduce cost terms that account for stability constraints such as ZMP, as shown in Fig. 3, that have been used extensively for optimal control problem formulations. The κ-constraint cost terms for the aCPPO are shown in table IV. We do not use the ZMP term in hCPPO since the violation of ZMP can be caused due to external perturbations and discarding policies based on ZMP violations implies that we cannot perform state recovery.

In order to encourage recovery into a stable state upon violation of κ-constraints, we introduce an additive reward term for each of the constraints in case the robot state shifts back to obeying these constraints upon violations.

7) η-Constraint Costs: For cases when the control policy executes actions that cause the robot to land in unstable and unrecoverable states, we introduce η-constraints. Upon violations of these constraints we terminate the training episode, and disregard the training steps that may have been explored between κ-constraint and η-constraint violations and add a negative terminal reward to the last training step in the reformatted episode samples. The intuition behind this was to limit updates through explorations in regions that, apart from violating constraints, do not contribute to learning a feasible and desired locomotion behavior. The constraint terms are shown in table V. When any of the expressions evaluate to true, the training episode is terminated.

III. Training

An overview of the training and validation process employed for our RL task is represented in Fig. 1.

TABLE II: Reward terms for the MDP formulation. Here K refers to the logistic kernel defined as $K(x) = (e^x + 2 + e^{-x})^{-1}$, $\mathcal{V}^{lin}_{base}$ is the desired linear velocity in base frame, τ is the joint torque, $\mathcal{V}^{ang}_{base}$ is the desired angular velocity in base frame, $v^{foot}_{world,t}$ is the foot velocity in world frame at time t, and $\mathcal{O}^{base}_{x,y,z}$ is the base orientation along the x, y, z axes.

Term	Expression
Linear Velocity	$K\left(v_{base} - \mathcal{V}^{lin}_{base}\right)$
Torque	$\|\tau\|^2$
Angular Velocity	$K\left(\omega_{base} - \mathcal{V}^{ang}_{base}\right)$
Foot Acceleration	$\|\|v^{foot}_{world,t} - v^{foot}_{world,t-1}\|\|^2$
Foot Slip	$\|\|v^{foot}_{world}\|\|^2$
Smoothness	$\|\|\mathcal{J}_t - \mathcal{J}_{t-1}\|\|^2$
Orientation	$\|\|\mathcal{O}^{base}_{x,y,z} - \{0,0,\mathcal{O}^{base}_{z}\}\|\|^2$

A. Generation of Reference Trajectories

We used a whole-body trotting controller to generate reference trajectories sampled at 400 Hz using the Gazebo simulator. The trajectories were represented as state-action (s, a^*) pairs as detailed in Section II-C.

B. Guided Policy Updates

Our training method alternates between supervised learning and reinforcement learning as presented in Alg. 1. The policy updates through supervised learning ensure that the policy search is directed towards a desired behavior. We trained the policy using the mean-squared-error loss between a^* and $\pi_\theta(s)$ minimized using the Adam [34] optimizer.

While performing experiments, we observed that the policy action entropy had to be reduced after each session of supervised learning, else the policy search still preferred exploration. In fact, during some training experiments we found that, without precise reward function tuning, not reducing entropy caused the RL agent to converge at a local minimum. We empirically determined the reduction in entropy after each successive update.

C. Constrained Policy Optimization

During policy exploration and optimization, we use the reward and cost functions described in the previous section to train the RL agent. We also introduce the following schemes to make our controller robust to unaccountable factors.

TABLE III: Cost terms for ρ-constraints. Here $\dot{\mathcal{J}}^{limit_\rho}$ refers to the joint speed limit for ρ-constraints, $\ddot{\mathcal{J}}^{limit_\rho}$ is the joint acceleration limit for ρ-constraints, $f_{h,i}$ is the height of end-effector i, $f_h^{des_\rho}$ is the desired end-effector height, $v_{f,i}$ is the velocity of the end-effector i, f_i is the position of end-effector i, $\mathcal{R}_i^\rho$ is the corresponding feasible end-effector region for ρ-constraints, and f_i^0 is the base position for end-effector i for a given joint configuration.

Term	Expression
Joint Speed	$\left\|\max(\|\dot{\mathcal{J}}_t\| - \dot{\mathcal{J}}^{limit_\rho}, 0)\right\|^2$
Joint Acceleration	$\left\|\max(\|\ddot{\mathcal{J}}_t\| - \ddot{\mathcal{J}}^{limit_\rho}, 0)\right\|^2$
Foot Clearance	$\Sigma_i \left(f_{h,i} - f_h^{des_\rho}\right)^2 \left\|v_{f,i}\right\|^2$
Foot Eligible Region	$\text{bool}\left(f_i \notin \mathcal{R}_i^\rho\right) \times \left\|f_i^0 - f_i\right\|^2$

TABLE IV: Cost terms for κ-constraints. Here $\dot{\mathcal{J}}^{limit_\kappa}$ refers to the joint speed limit for κ-constraints, $\ddot{\mathcal{J}}^{limit_\kappa}$ is the joint acceleration limit for κ-constraints, $\mathcal{R}_i^\kappa$ is the corresponding feasible end-effector region for κ-constraints, u is the ZMP, $\mathcal{S}$ is the region of support polygon with vertices given by the feet in contact with the ground, C is the center of mass of the quadruped, and F_{f_i} is the contact force at foot i. The foot contacts cost term ensures that if 2 feet are in contact with the ground, they are not on the same side.

Term	Expression
Joint Speed	$\left\|\max(\|\dot{\mathcal{J}}_t\| - \dot{\mathcal{J}}^{limit_\kappa}, 0)\right\|^2$
Joint Acceleration	$\left\|\max(\|\ddot{\mathcal{J}}_t\| - \ddot{\mathcal{J}}^{limit_\kappa}, 0)\right\|^2$
Foot Eligible Region	$\text{bool}\left(f_i \notin \mathcal{R}_i^\kappa\right) \times \left\|f_i^0 - f_i\right\|^2$
ZMP	$\text{bool}\left(u \notin \mathcal{S}\right) \times \|u - \text{C}\|^2$
Foot Contacts	$\text{bool}(\Sigma_i(F_{f_i} > 0) < 3$ & $((F_{f_{lf}} > 0$ & $F_{f_{lh}} > 0)$ or $(F_{f_{rf}} > 0$ & $F_{f_{rh}} > 0)))$

TABLE V: η-constraint terms. Here $\dot{\mathcal{J}}^{limit_\eta}$ refers to the joint speed limit for η-constraints, $\ddot{\mathcal{J}}^{limit_\eta}$ is the joint acceleration limit for η-constraints, $\mathcal{R}_i^\eta$ is the corresponding feasible end-effector region for η-constraints and u^η is the maximum allowed ZMP distance from the center of mass, C is the center of mass of the quadruped, and the minimum foot contacts have been set to 1 to ensure the control policy does not generate a behavior such as pronking.

Term	Expression
Joint Speed	$\text{bool}(\dot{\mathcal{J}} > \dot{\mathcal{J}}^{limit_\eta})$
Joint Acceleration	$\text{bool}(\ddot{\mathcal{J}} > \ddot{\mathcal{J}}^{limit_\eta})$
Foot Eligible Region	$\text{bool}\left(f_i \notin \mathcal{R}_i^\eta\right)$
ZMP	$\text{bool}\left(\|u - \text{C}\| > u^\eta\right)$
Foot Contacts	$\text{bool}\left(\sum_i \text{bool}(F_{f_i} > 0) < 2\right)$

1) Adding Noise to Observations and Actions: We add Gaussian noise to the state and action vector [29] to account for sensor noise and inaccurate actuation. The standard deviation vector for the observation space is given as $s_c\{0.02, 0.1, [0.05]_3, [0.07]_3, [0.02]_{12}, [0.0]_{48}, [0.05]_{36}, [0.0]_3\}$, and for the action space, it is given as $s_c\{[0.04]_{12}\}$, where $s_c \in [0,1]$ is a scaling term increased over the training period.

2) Changing Gravity: We randomly sample acceleration due to gravity between $[0.95g, 1.05g]$, where $g = 9.81$ m/s^2 to emulate inertial scaling.

3) Actuator Torque Scaling: We randomly scale the output torque of our actuator network with the scaling coefficient $s_t \in [0.5, 2.0]$ to account for differences between the real actuators and the approximated model.

4) Changing Link Mass and Size: To ensure the training does not converge to a local minimum, we scale the mass and size of each of the links by coefficients $s_m^{link} \in [0.93, 1.07]$, and $s_l^{link} \in [0.97, 1.05]$ respectively.

5) Adding Actuator Damping: We emulate actuator damping by changing the output of the control policy using a complementary filter given as $\mathcal{J}_t^{des'} = K_{damp}\mathcal{J}_t^{des} + (1 - K_{damp})\mathcal{J}_{t-1}^{des'}$ where the gain K_{damp} is randomized between $[1 - (s_c/4), 1]$.

6) Changing Simulation Step Time: For the possibility of execution on soft real time systems, we randomly set the step times of the control loop between $[2.25, 2.75]$ ms. During experiments we observed that the control policy even worked when the control frequency was changed from 400Hz to 200Hz.

IV. Results and Discussion

We performed all training and experiments on commodity hardware; an Intel i7-8700K and Nvidia RTX 2080Ti. For training a control policy using our method, we required 450M simulation samples for about 240 policy optimization iterations using aCPPO, requiring less than 2 hours for a RaiSim based simulation. We then performed 36 policy iterations over 20M simulation samples using hCPPO. The authors of [15] required more than 7B simulation steps for convergence.

Having obtained a visually stable behavior in RaiSim, we tested the control policy in Gazebo simulation of ANYmal which consisted of an analytical actuator model. We then tested the control policy on the physical system. In our RL training, we do not use Gazebo because the compute time required for each simulation step is significantly larger than for RaiSim.

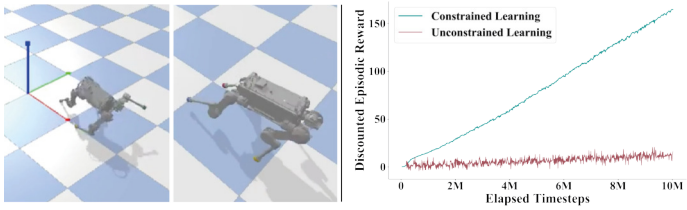

Fig. 4: Comparing differences in learnt behavior for constrained and unconstrained learning approaches. *left:* the unconstrained learning approach fails to develop an optimal strategy to track the desired base velocity commands even after 10M steps, *center:* the constrained learning approach limits the exploration of the policy within desired regions thereby directing policy optimization towards a preferred optimum, *right:* the average discounted episodic reward curves observed during training.

During our experiments with unconstrained learning methods, we observed that a constrained learning approach significantly directed policy convergence. We trained a control policy for a simple task of tracking forward base velocity of 0.7 m/s and observed that when we introduced even a basic constraint such as limits on end-effector positions with respect to the nominal stance, the policy trained using our constrained proximal policy optimization method performed much better than unconstrained proximal policy optimization, as is represented in Fig. 4. The reward for both the approaches was defined using the logistic kernel as $K(v_{base} - 0.7)$ scaled by a constant. We used the eligible foot regions defined in Table IV for constrained learning.

Moreover, for tracking base velocity commands, we required a minimum of about 2B simulation samples with precise reward function tuning to obtain a control policy similar to the trot controller. We changed the reward coefficients, increasing it for the torque required and decreasing for foot slip, foot clearance and smoothness empirically over at least 20 trials to get such a behavior. Without reward tuning, we obtained inefficient locomotion strategies such as pronking. This was, however, not the case with GCPO. Moreover, introducing GPUs in our approach helped us reduce the required training samples from approximately 1.6B, in the case of only CPPO, to 470M for GCPO.

The velocity tracking results obtained with our trained control policy on the physical robot system, outdoors on uneven terrain, are as shown in Fig. 5. It is important to note that the physical system comprised of additional sensor modules, amounting to approximately 10% of the robot mass, which had not been included in the simulations during training. We compared the results with the model-based trot controller we used for GPUs and observed that in most cases our controller tracks the velocity commands better than the trot controller as is evident from the tracking error plots. Here, we show that our policy closely tracks the velocity commands on the physical system despite having been trained on a simulator making this a successful sim-to-real policy transfer. Figure 6 represents the sum of the magnitude of torques measured at each joint for measured forward base velocities. We show that our controller, trained using GCPO, requires significantly lesser torque than the model-based trot controller. At most of the measured forward base velocities, the total joint torque measured for our controller is 20 Nm less than for the trot controller.

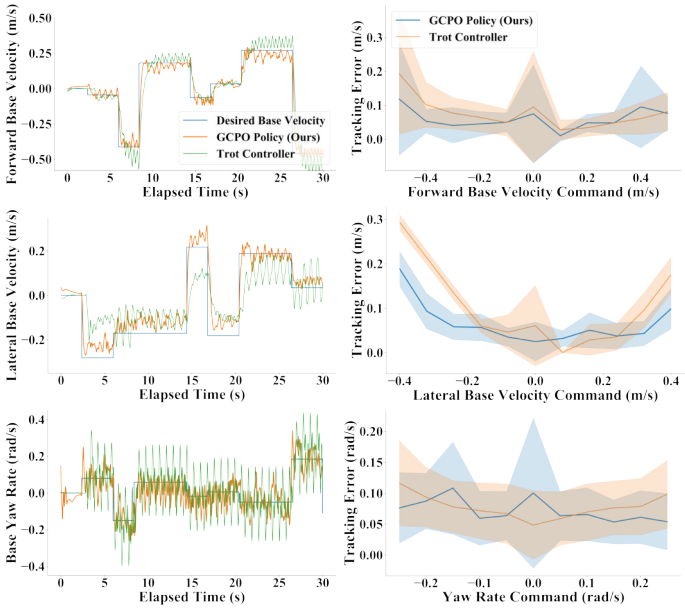

Fig. 5: Velocity tracking results obtained on the physical quadruped system. *Left:* base velocities measured for our trained control policy and model-based trot controller for the same sequence of velocity commands, *right:* the mean tracking error observed for given base velocity commands. The confidence bands represent the standard deviation of tracking error.

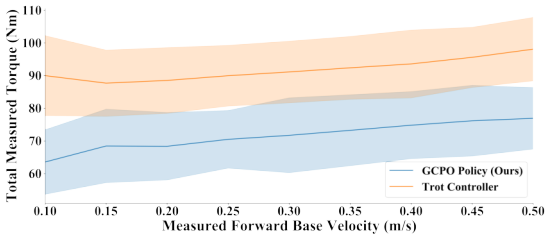

Fig. 6: The sum of the magnitude of joint torques measured on the physical system for each of the joints plotted against the measured forward base velocity. The confidence bands represent the standard deviation of torques measured. The controllers operate at 400 Hz.

As shown in Fig. 7, the density of joint velocity and joint accelerations for measured forward base velocity commands is very high below 5 rad/s and 50 rad/s^2 for the physical robot, while being even lower for RaiSim. In our experiments, for 5 minutes of data sampled at 400 Hz, we observed that the hard constraints on the physical system were only violated for 0.0196% and 0.0490% of times for joint velocities and joint accelerations respectively. For RaiSim, these values were 0.00119% and 0.00178%. As represented in Fig. 8 for the unconstrained approach, we measured these values in RaiSim and computed them to be 1.1710% and 1.2511%, 3 orders of magnitude larger than for our GCPO controller measured in RaiSim. Moreover, the density of the joint velocities and accelerations is significantly higher near the limits than for our constrained approach. It is important to note, however, that the behavior of the unconstrained control policies can be changed by significant re-tuning of the reward function. Furthermore, our control policy always maintains at least 2 stance legs during locomotion on flat terrain.

Exceeding our expectations, our GCPO controller was able to track the forward base velocity commands, on the physical

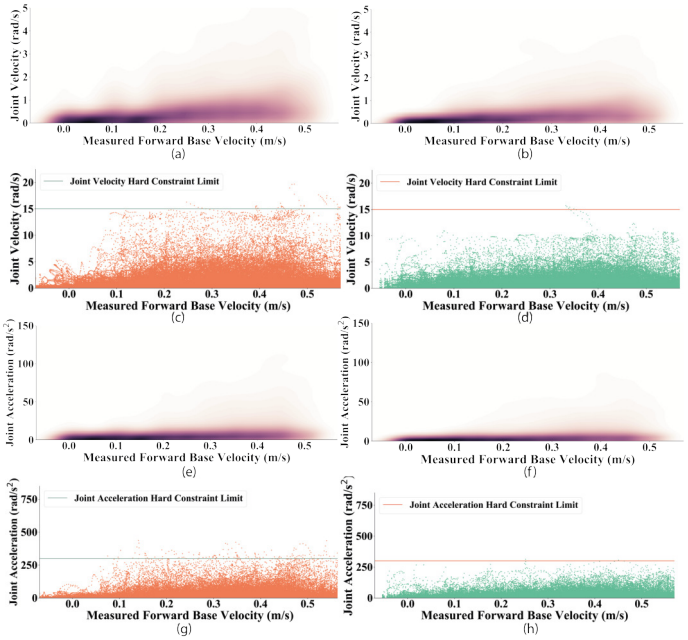

Fig. 7: Joint velocity and acceleration plots for each of the joint on the ANYmal quadruped plotted against measured forward base velocity. *(a)* the kernel density estimate (KDE) for the joint velocities measured on the physical system plotted using the parameters detailed in [35], *(b)* KDE for joint velocities measured in RaiSim, *(c)* scatter plot of the measured joint velocities on the physical system with hard constraint limit set to 15 rad/s, *(d)* scatter plot of the measured joint velocities in RaiSim, *(e)* KDE for joint accelerations measured on the physical system, *(f)* KDE for joint accelerations measured in RaiSim, *(g):* scatter plot of the measured joint accelerations on the physical system with hard constraint limit set to 300 rad/s^2, *(h)* scatter plot of the measured joint accelerations in RaiSim.

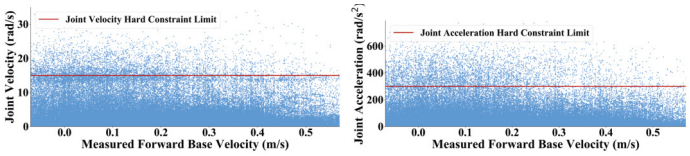

Fig. 8: *Left:* Joint velocity scatter plots obtained for every joint on the quadruped for an unconstrained locomotion behavior measured in RaiSim, *right:* joint acceleration scatter plots.

system, even when we introduced delays into the control execution with an RMS tracking error of 0.1736 m/s for following a base velocity command of 0.5 m/s for 10 s. These delays were randomly sampled from a uniform distribution between $[0, 17.5]$ ms. The controller became unstable when we sampled delays from distributions with upper limit greater than 20 ms. In most soft-realtime control systems, these delays are introduced by low-level hardware communication interfaces, and such a robust controller is certainly desirable.

We observed our controller's response to external perturbations by applying forces with magnitude ranging from 50 N to 120 N for duration between 1 s to 5 s to the robot base in Gazebo. We observed that our controller responded to these external perturbations by moving in the direction opposite to that of the applied force, ensuring it's stability. The controller was able to respond to external forces of up to 100 N for 1 s duration applied to the base horizontally. We also tested the controller's response on the physical system.

We emulated a weak actuator, to test the case where an

actuator becomes damaged, by reducing the position tracking gain of the weak actuator to 24. The position gain for all other actuators was set to 48. We observed that our controller was still able to track the base velocity commands with an RMS tracking error of 0.1975 m/s for following a base velocity command of 0.5 m/s for 10 s.

Furthermore, despite significant inertial scaling, our control policy, without any parameter re-tuning, still managed to track velocity commands even when we set the acceleration due to gravity to 1.62 m/s^2 in RaiSim. We observed an RMS tracking error of 0.2214 m/s for following a base velocity command of 0.5 m/s for 10 s.

V. Conclusion

We presented an RL training method for quadrupedal locomotion which considers safety-critical constraints in its problem formulation and further encourages system recovery into stable states upon constraint violations. We used reference trajectories, obtained using a trot controller, to perform GPUs in order to direct policy optimization. Our experiments demonstrate that our RL method offers a robust controller which requires significantly lesser torque for execution compared with a model-based controller. Furthermore, it is important to note that constrained policy optimization does not necessitate use of GPUs. In our work, CPPO can be used even without GPUs, but have been introduced to limit exploration and hence reduce the samples required for convergence.

As part of future research, we aim to demonstrate the applicability of our method to more complex environments, and to extend our method to use perception data as a means to generalizing to a wide variety of challenging terrains.

Acknowledgement

Siddhant would like to thank Dr. Jemin Hwangbo from RSL, ETH Zurich for his valuable inputs. We would also like to thank Luigi Campanaro and Benoit Casseau for helping us with our experiments.

References

[1] J. T. Betts, "Survey of numerical methods for trajectory optimization," *Journal of guidance, control, and dynamics*, vol. 21, no. 2, pp. 193–207, 1998.

[2] M. Neunert, M. Stäuble, M. Giftthaler, C. D. Bellicoso, J. Carius, C. Gehring, M. Hutter, and J. Buchli, "Whole-body nonlinear model predictive control through contacts for quadrupeds," *IEEE Robotics and Automation Letters*, vol. 3, no. 3, pp. 1458–1465, 2018.

[3] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[4] O. Pietquin, M. Geist, S. Chandramohan, and H. Frezza-Buet, "Sample-efficient batch reinforcement learning for dialogue management optimization," *ACM Transactions on Speech and Language Processing (TSLP)*, vol. 7, no. 3, p. 7, 2011.

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

[6] 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.

[7] C. Chow and D. Jacobson, "Studies of human locomotion via optimal programming," *Mathematical Biosciences*, vol. 10, no. 3-4, pp. 239–306, 1971.

[8] A. E. Bryson, *Applied optimal control: optimization, estimation and control*. Routledge, 2018.

[9] 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.

[10] C. D. Bellicoso, C. Gehring, J. Hwangbo, P. Fankhauser, and M. Hutter, "Perception-less terrain adaptation through whole body control and hierarchical optimization," in *2016 IEEE-RAS 16th International Conference on Humanoid Robots (Humanoids)*. IEEE, 2016, pp. 558–564.

[11] M. Frank and P. Wolfe, "An algorithm for quadratic programming," *Naval research logistics quarterly*, vol. 3, no. 1-2, pp. 95–110, 1956.

[12] J. Lee, J. Hwangbo, and M. Hutter, "Robust recovery controller for a quadrupedal robot using deep reinforcement learning," *arXiv preprint arXiv:1901.07517*, 2019.

[13] X. B. Peng, G. Berseth, K. Yin, and M. Van De Panne, "Deeploco: Dynamic locomotion skills using hierarchical deep reinforcement learning," *ACM Transactions on Graphics (TOG)*, vol. 36, no. 4, p. 41, 2017.

[14] A. Nagabandi, G. Kahn, R. S. Fearing, and S. Levine, "Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2018, pp. 7559–7566.

[15] 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.

[16] F. Zhang, J. Leitner, M. Milford, and P. Corke, "Modular deep q networks for sim-to-real transfer of visuo-motor policies," *arXiv preprint arXiv:1610.06781*, 2016.

[17] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel, "Domain randomization for transferring deep neural networks from simulation to the real world," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2017, pp. 23–30.

[18] K. Cobbe, O. Klimov, C. Hesse, T. Kim, and J. Schulman, "Quantifying generalization in reinforcement learning," *arXiv preprint arXiv:1812.02341*, 2018.

[19] A. Y. Ng, S. J. Russell, *et al.*, "Algorithms for inverse reinforcement learning." in *Icml*, vol. 1, 2000, p. 2.

[20] M. Wulfmeier, D. Rao, D. Z. Wang, P. Ondruska, and I. Posner, "Large-scale cost function learning for path planning using deep inverse reinforcement learning," *The International Journal of Robotics Research*, vol. 36, no. 10, pp. 1073–1087, 2017.

[21] C. Finn, S. Levine, and P. Abbeel, "Guided cost learning: Deep inverse optimal control via policy optimization," in *International Conference on Machine Learning*, 2016, pp. 49–58.

[22] F. Torabi, G. Warnell, and P. Stone, "Behavioral cloning from observation," *arXiv preprint arXiv:1805.01954*, 2018.

[23] S. Levine and V. Koltun, "Guided policy search," in *International Conference on Machine Learning*, 2013, pp. 1–9.

[24] K. Doya, K. Samejima, K.-i. Katagiri, and M. Kawato, "Multiple model-based reinforcement learning," *Neural computation*, vol. 14, no. 6, pp. 1347–1369, 2002.

[25] J. Achiam, D. Held, A. Tamar, and P. Abbeel, "Constrained policy optimization," in *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org, 2017, pp. 22–31.

[26] M. L. Puterman, *Markov Decision Processes.: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, 2014.

[27] E. Altman, *Constrained Markov decision processes*. CRC Press, 1999, vol. 7.

[28] J. Peters and S. Schaal, "Reinforcement learning of motor skills with policy gradients," *Neural networks*, vol. 21, no. 4, pp. 682–697, 2008.

[29] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, "Trust region policy optimization," in *International conference on machine learning*, 2015, pp. 1889–1897.

[30] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel, "High-dimensional continuous control using generalized advantage estimation," *arXiv preprint arXiv:1506.02438*, 2015.

[31] 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.

[32] E. Coumans and Y. Bai, "Pybullet, a python module for physics simulation for games, robotics and machine learning," http://pybullet.org, 2016–2019.

[33] E. Todorov, T. Erez, and Y. Tassa, "Mujoco: A physics engine for model-based control," *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 5026–5033, 2012.

[34] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[35] D. W. Scott, "On optimal and data-based histograms," *Biometrika*, vol. 66, no. 3, pp. 605–610, 1979.