Learning Locomotion for Quadruped Robots via Distributional Ensemble Actor-Critic

Sicen Li, Yiming Pang, Panju Bai, Jiawei Li, Zhaojin Liu, Shihao Hu, Liquan Wang, and Gang Wang, *Member, IEEE*

***Abstract*—Domain randomization introduces perturbations in the simulation to make controllers less susceptible to the reality gap, which enables remarkable sim-to-real transfer on real quadruped robots. However, aleatoric uncertainty originating from perturbations could often lead to suboptimal controllers. In this work, we present a novel algorithm called Distributional Ensemble Actor-Critic (DEAC) that blends three ideas: distributional representation of a critic, lower bounds of the value distribution, and ensembling of multiple critics and actors. Distributional representation and ensembling provide reasonable uncertainty estimates, while lower bounds of the value distribution offer finer-grained error control. The simulation results show that the controller trained by DEAC outperforms the other baselines in the domain randomization setting. The trained controller is deployed on an A1-like robot, demonstrating high-speed running and the ability to traverse diverse terrains such as slippery plates, grassland, and wet dirt.**

***Index Terms*—Quadruped robot, reinforcement learning, distributional method, ensemble method.**

I. Introduction

RECENTLY reinforcement learning (RL) [1], [2] has emerged as a promising approach to quadrupedal locomotion problems end-to-end without much manual effort. Domain randomization [3] is a key technique that enables successful zero-shot sim-to-real deployment for RL. Domain randomization randomizes the dynamics of the environment, thus exposing the agent to a diverse set of environments in the training phase to enhance robustness.

However, typical domain randomization may lead to suboptimal and high-variance controllers [4]. For example, external forces push the agent forward, which returns confused reward signals to the agent. Additionally, partially observational properties and sensor noise of quadrupeds can trigger severe perceptual aliasing: a large number of observations are produced that are identical in different states. Although modern simulators provide a cheap and essentially unlimited amount of data, aleatoric uncertainty originating from stochastic dynamics and noise will always plague an RL agent to learn high-performance policy.

In this work, we mitigate this limitation by proposing a new RL algorithm called Distributional Ensemble Actor-Critic (DEAC) to improve the controller performance under domain randomization. DEAC incorporates distributional [5], [6] and ensemble methods to effectively model the aleatoric uncertainty inherent in the agent's interaction with the environment. Learning the distribution of value function could help agents capture useful information about the environment, such as the probability of potential disturbances and high-return events.

To prevent catastrophic overestimation bias [7], [8], previous works [7], [8], [9] usually truncate nor minimize some approximations. However, these approaches are wasteful as they ignore some of the estimates and diminish the power of the ensemble of approximations. To take full advantage of all approximations, DEAC calculates the uncertainty estimate of each distributional atom across the critic ensemble to obtain the lower bound of value distribution, providing finer-grained error control. In addition, DEAC learns value distributions and policies with respect to various confidence bounds jointly in the same network, which can be considered a set of auxiliary tasks [10] that help build shared state representations and reduce the computational cost.

DEAC shows superior performance for learning legged locomotion in our setup, compared to SAC [9], REDQ [7], TQC [8], and PPO with automatic curriculum [2], [11]. We thoroughly analyze the value function model arising from our method and study the effects of network architectures, settings, and parameters in simulation. We further deploy the trained policy on a quadruped robot, demonstrating the ability to traverse challenging terrains flexibly with a maximum speed of 3.5 m/s.

Our contribution is threefold. First, we significantly improve the asymptotic performance and sample efficiency of the RL algorithm by learning value function distributions and ensembles. Second, our method improves the controller's performance under domain randomization. Finally, we investigate the learned locomotion behaviors in simulation and hardware.

II. Related Works

Control-based method: Control-based methods typically convert motion planning problems into nonlinear programming

Manuscript received 23 August 2023; accepted 19 December 2023. Date of publication 4 January 2024; date of current version 16 January 2024. This letter was recommended for publication by Associate Editor J. Hwangbo and Editor J. Kober upon evaluation of the reviewers' comments. This work was supported by the National Natural Science Foundation of Heilongjiang Province under Grant YQ2020E028. *(Corresponding author: Gang Wang.)*

Sicen Li, Yiming Pang, Panju Bai, Zhaojin Liu, Shihao Hu, and Liquan Wang are with the College of Mechanical and Electrical Engineering, Harbin Engineering University, Harbin 150001, China (e-mail: ihuhuhu@hrbeu.edu.cn; pangyiming@hrbeu.edu.cn; baipanju@hrbeu.edu.cn; liuzhaojin@hrbeu.edu.cn; jhjsb@hrbeu.edu.cn; wangliquan@hrbeu.edu.cn).

Jiawei Li and Gang Wang are with the College of Shipbuilding Engineering, Harbin Engineering University, Harbin 150001, China (e-mail: ljw1996@hrbeu.edu.cn; wanggang@hrbeu.edu.cn).

This letter has supplementary downloadable material available at https://doi.org/10.1109/LRA.2024.3349934, provided by the authors.

Digital Object Identifier 10.1109/LRA.2024.3349934

problems (NLP) with a finite number of decision variables and constraints. ANYmal robot [12], MIT Cheetah 3 [13] used regularized model predictive control (MPC) and simplified dynamics to achieve a maximum yaw rate of 180 degrees/sec and a 3.0 m/s maximum linear velocity. The ANYmal robot relies on foothold optimization, trajectory optimization, and tracking with a whole-body controller to move. HYQ [14] used a full-body controller to overcome nonlinear MPC issues to accomplish rough terrain adaptation problems. However, extensive task-specific feature engineering or heuristics are still required to enhance the cost function and reduce computation time for online execution.

Reinforcement learning for quadrupedal locomotion: Controllers trained with Reinforcement learning in simulations are difficult to deploy on real robots [15]. Additional efforts are usually required to transfer the policies to real robots, including constructing more accurate simulations [16], domain randomization [3], and privilege learning [11]. A remarkable example is the trained velocity tracking controllers for the ANYmal robot at speeds up to 1.5 m/s, which were extended using a privileged learning paradigm over various terrains [11]. Additionally, an end-to-end learned controller achieved record agility for the MIT Mini Cheetah, sustaining speeds up to 3.9 m/s [2]. Enhancing policy robustness (like domain randomization) can provide confused rewards or cause perceptual aliasing, significantly diminishing the performance of RL algorithms compared to randomization-free settings.

Distributional Reinforcement learning: The stochasticity of domain randomization means it changes in ways that are not fully predictable. Most previous legged control studies have given little consideration to this issue. Fortunately, distributional RL, which learns the value distribution instead of the value function, is an effective tool to solve the above problem [5], [8], [17]. QR-DQN [5] approximates the value distribution using a mixture of atoms. IQN [17] learns quantile values for a uniform grid of sampled [17] quantile fractions. Additionally, TQC [8] allows for arbitrary granular bias control by truncated predictions. Distributional PPO [18] learns to adapt robot locomotion behavior in risky situations. However, the powerful modeling capacity of distributional RL also brings the overfitting problem, which is accentuated in the domain randomization setting.

Ensemble Reinforcement learning: SUNRISE [19] bootstraps with random initialization, which improves the stability of the learning process by training a diverse ensemble of agents. However, learning such a huge ensemble can seriously reduce the efficiency of training and tuning. REDQ [7] uses in-target minimization across a random subset of Q-functions from the value ensemble to perform just as well as the state-of-the-art model-based algorithms for the MuJoCo benchmark [20]. However, REDQ can significantly underestimate the value function with domain randomization, leading to overly conservative policies.

III. DEAC

We propose the DEAC algorithm, which combines QR-DQN [5] and ensemble methods to improve SAC [9] in stochastic environments. An overview of the DEAC algorithm is present in Fig. 1. The pseudocode for DEAC is shown in Algorithm 1.

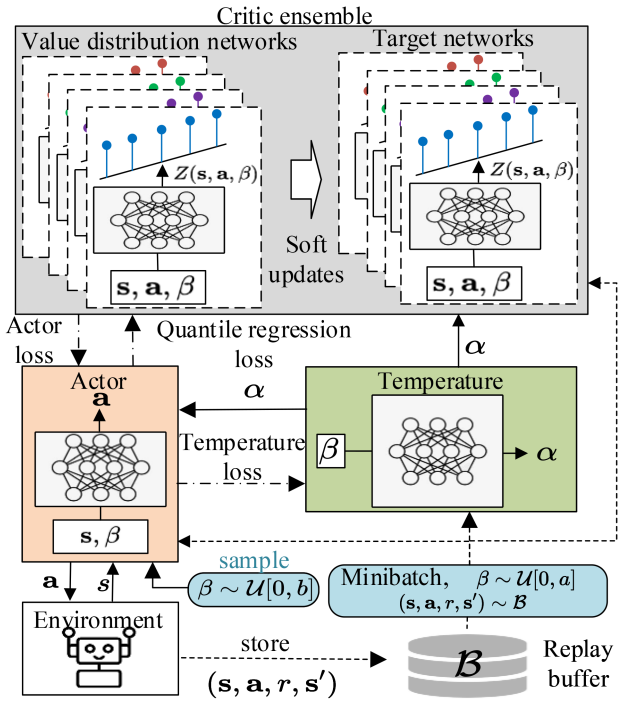

Fig. 1. Overview of the DEAC algorithm.

Algorithm 1: DEAC.

Initialize actor network ϕ, critic networks θ_n, target network $\bar{\theta}_n \longleftarrow \theta_n$, for $n = 1, \ldots, N$, temperature network ψ, empty replay buffer $\mathcal{B}$, uniform distribution U_1 and U_2, UTD ratio G
for each iteration **do**
  execute an action $a \sim \pi_\phi(\cdot \mid s, \beta), \beta \sim U_2$.
  Observe reward r_t, new state s'
  Store transition tuple $\mathcal{B} \leftarrow \mathcal{B} \cup \{(s, a, r_t, s')\}$
  for G updates **do** // UPDATE CRITICS
    Sample random minibatch:
    $\{\mathbf{T}_j\}_{j=1}^{B} \sim \mathcal{B}$, $\{\beta_i\}_{i=1}^{B} \sim U_1$
    Compute the target distribution $\hat{Z}$ (5)
    for $n = 1, \ldots, N$ **do**
      Update θ_n by minimize $\mathcal{L}^{\texttt{DEAC}}_{\texttt{critic}}$ (6)
      Update target networks $\bar{\theta}_n \leftarrow \rho\bar{\theta}_n + (1-\rho)\theta_n$
  // UPDATE ACTOR AND TEMPERATURE NETWORKS
  Update ϕ by minimize $\mathcal{L}^{\texttt{DEAC}}_{\texttt{actor}}$ (8)
  Update ψ by minimize $\mathcal{L}^{\texttt{DEAC}}_{\texttt{temp}}$ (4)

A. Problem Setting and Preliminaries

We formulate our control problem in discrete time dynamics [21], with states $\mathbf{s}$, actions $\mathbf{a}$, reward $r(\mathbf{s}, \mathbf{a})$, and dynamics $p(\mathbf{s}' \mid \mathbf{s}, \mathbf{a})$. The discounted return $R_t = \sum_{k=0}^{\infty} \gamma^k r_k$ is the total accumulated rewards from time step t, $\gamma \in [0, 1]$ is a discount factor determining the priority of short-term rewards. The objective is to find the optimal policy $\pi_\phi(\mathbf{s} \mid \mathbf{a})$ with parameters ϕ, which maximizes the expected return $J(\phi) = \mathbb{E}_{p_\pi}[R_t]$.

In the quadrupedal locomotion task, there is unobservable information, such as external forces or full terrain information, and the dynamics are modeled as a Partially Observable Markov Decision Process (POMDP) [21]. The POMDP can be reconstructed as an MDP by defining n-size observations as state [22]:

$$\mathbf{s_t} \stackrel{\text{def}}{=} (\mathbf{o}_{t-H}, \mathbf{a}_{t-H-1}, \mathbf{o}_{t-H+1}, \mathbf{a}_{t-H}, \ldots, \mathbf{o}_t, \mathbf{a}_{t-1}), \quad (1)$$

The maximum entropy RL [9] attempts to find a policy that maximizes the objective:

$$\pi^* = \arg\max_{\pi} \sum_t \mathbb{E}_{\mathbf{s}\sim p, \mathbf{a}\sim\pi} \left[r(\mathbf{s}, \mathbf{a}) + \alpha \mathcal{H}\left(\pi\left(\cdot \mid \mathbf{s}\right)\right)\right], \quad (2)$$

where α is the temperature parameter used to determine the relative importance of entropy and reward, π^* is the optimal policy, $\mathcal{H}(\pi(\cdot \mid \mathbf{s}))$ is the entropy of the policy π at state $\mathbf{s}$ and is calculated as $\mathcal{H}(\pi(\cdot \mid \mathbf{s})) = -\log \pi(\cdot \mid \mathbf{s})$.

B. Critic Loss and Temperature Loss

Distributional RL focuses on approximating the return random variable $Z^\pi(\mathbf{s}, \mathbf{a}) := \sum_{k=0}^{\infty} \gamma^k r_k$ as opposed to approximating the expectation of the return, also known as the Q-function $Q^\pi(\mathbf{s}, \mathbf{a}) := \mathbb{E}[Z^\pi(\mathbf{s}, \mathbf{a})]$ and the critic in actor-critic methods. QR-DQN [5] calculates $Z_\theta(\mathbf{s}, \mathbf{a}) := \frac{1}{M} \sum_{m=1}^{M} \delta_{\zeta_m(\mathbf{s},\mathbf{a})}$ to approximate $Z^\pi(\mathbf{s}, \mathbf{a})$, where δ_x denotes a Dirac at $x \in \mathbb{R}$, supported on atom $\zeta_1, \ldots, \zeta_M$. The expected value is decomposed into atoms of distributional representation to achieve precise value estimation.

DEAC proposes to train N approximations $Z_{\theta_1}(\mathbf{s}, \mathbf{a}, \beta), \ldots, Z_{\theta_N}(\mathbf{s}, \mathbf{a}, \beta)$ and the parameterized policy $\pi_\phi(\cdot \mid \mathbf{s}, \beta)$, known as the actor, all of them are extended with UVFA [23]. The approximations provides better uncertainty modeling ability. U_1 is a uniform training distribution $\mathcal{U}[0, a]$, $a > 0$, and $\beta \sim U_1$ generates various bounds of value approximations:

$$Z_{\theta_n}(\mathbf{s}, \mathbf{a}, \beta) := \frac{1}{M} \sum_{m=1}^{M} \delta_{\zeta^m_{\theta_n}(\mathbf{s},\mathbf{a},\beta)}, \quad (3)$$

supported on atom $\zeta^1_{\theta_n}, \ldots, \zeta^M_{\theta_n}$.

An independent temperature network α_ψ parameterized by ψ is used to accurately adjust the temperature with respect to β. DEAC uses an automated entropy adjusting mechanism [9] to update α with the following objective:

$$\mathcal{L}^{\texttt{DEAC}}_{\texttt{temp}}(\psi) = \mathbb{E}_{\mathbf{s}\sim\mathcal{B}, \mathbf{a}\sim\pi_\phi, \beta\sim U_1}[-\alpha_\psi(\beta) \log \pi_\phi\left(\mathbf{a} \mid \mathbf{s}, \beta\right) - \alpha_\psi(\beta)\overline{\mathcal{H}}]. \quad (4)$$

where $\overline{\mathcal{H}}$ is the target entropy. The target entropy usually is set heuristically $\overline{\mathcal{H}} = -\dim \mathcal{A}$, where $\mathcal{A}$ is the action space dim.

We train approximation $Z_{\theta_1}, \ldots Z_{\theta_N}$ on the temporal difference target distribution $\hat{Z}(\mathbf{s}, \mathbf{a})$:

$$\hat{Z}(\mathbf{s}, \mathbf{a}, \beta) = r + \gamma \mathbb{E}_{\mathbf{a}'\sim\pi_\phi}[\bar{\zeta}_{\bar{\theta}}(\mathbf{s}', \mathbf{a}', \beta) - \beta \hat{s}(\zeta_{\bar{\theta}}(\mathbf{s}', \mathbf{a}', \beta)) - \alpha_\psi(\beta) \log \pi_\phi(\mathbf{a}' \mid \mathbf{s}', \beta)], \quad (5)$$

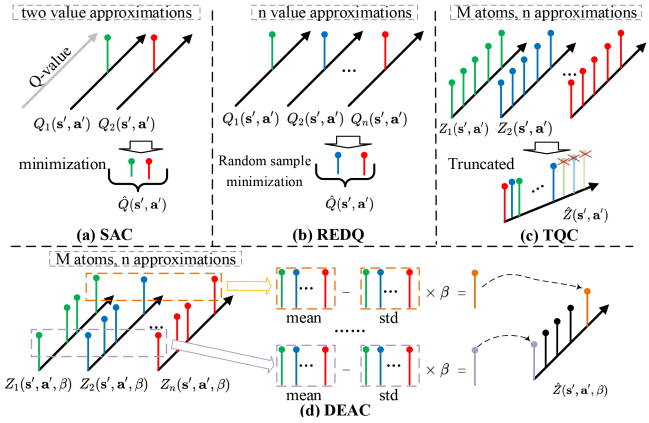

Fig. 2. Comparison of DEAC with SAC [9], REDQ [7], and TQC [8] on calculating the temporal difference target. (a) SAC: minimization of two value approximations. (b) REDQ: in-target minimization across a random subset of Q functions from the ensemble. (c) TQC: truncation of approximated distribution. (d) DEAC: calculating the lower bound of each approximated atom.

where $\bar{\zeta}_{\bar{\theta}}(\mathbf{s}, \mathbf{a}, \beta)$ is the sample mean of target atoms $\zeta_{\bar{\theta}_n}$ of the critic ensemble and $\hat{s}(\zeta_{\bar{\theta}}(\mathbf{s}, \mathbf{a}, \beta))$ is the corrected sample standard deviation, $\bar{\theta}$ are the delayed parameters which are updated by exponential moving average $\bar{\theta} \leftarrow \rho\theta + (1-\rho)\bar{\theta}$.

Learning in a stochastic environment is more likely to produce catastrophic overestimation errors than in a deterministic environment [24]. The target $\bar{\zeta}_{\bar{\theta}}(\mathbf{s}, \mathbf{a}, \beta)$ obtains a lower bound for the target distribution to prevent overestimation errors from propagating to other states through TD learning updates. Fig. 2 shows the differences between DEAC and SAC [9], REDQ [7] and TQC [8].

We minimize the 1-Wasserstein distance between each of $Z_{\theta_n}(\mathbf{s}, \mathbf{a}, \beta), n \in [1..N]$ and the temporal difference target distribution $\hat{Z}(\mathbf{s}, \mathbf{a}, \beta)$. Through learning the locations for quantile fraction $\tau_m = \frac{2m-1}{2M}, m \in [1..M]$, we can approximate the quantiles of the target distribution. The $\tau_m, m \in [1..M]$ quantiles of Z_θ are approximated by minimizing the quantile regression loss [5].

For quantile $\tau \in [0, 1]$, the quantile regression loss is an asymmetric convex loss function that penalizes overestimation errors with weight τ and underestimation errors with weight $1-\tau$:

$$\mathcal{L}^{\texttt{DEAC}}_{\texttt{critic}}(\theta_n) = \mathbb{E}_{\mathbf{T}\sim\mathcal{B}, \beta\sim U_1, z\sim\hat{Z}, \zeta\sim\zeta_{\theta_n}}\left[\rho^H_\tau(z - \zeta(\mathbf{s}, \mathbf{a}, \beta))\right],$$
$$\text{where} \rho^H_\tau(u) = |\tau - \mathbb{I}(u < 0)|\mathcal{L}^1_H(u), \quad (6)$$

where $\mathbf{T} = (\mathbf{s}, \mathbf{a}, r, \mathbf{s}')$ is a transition, $\mathcal{B}$ is a replay buffer, ρ is the target smoothing coefficient, y is the target value. $\mathcal{L}^1_H$ is the Huber loss that can improve gradients for small u:

$$\mathcal{L}^\kappa_H(u) = \begin{cases} \frac{1}{2}u^2, & \text{if } |u| \le \kappa \\ \kappa\left(|u| - \frac{1}{2}\kappa\right), & \text{otherwise .} \end{cases} \quad (7)$$

This loss gives unbiased sample gradients. As a result, we can find the minimizing $\zeta^1_{\theta_n}, \ldots, \zeta^M_{\theta_n}$ by stochastic gradient descent. A larger Update-To-Data(UTD) ratio G[7], which is the number of updates taken by the agent compared to the number of actual interactions with the environment, improves sample efficiency.

C. Actor Loss

The extended policy parameters ϕ can be optimized to maximize the entropy penalized estimate of the Q-value by minimizing the loss:

$$\mathcal{L}^{\texttt{DEAC}}_{\texttt{actor}}(\phi) = \mathbb{E}_{\mathbf{s}\sim\mathcal{B},\beta\sim U_1}[\mathbb{E}_{\mathbf{a}\sim\pi_\phi}[\alpha_\psi(\beta)\log\left(\pi_\phi\left(\mathbf{a}\mid\mathbf{s},\beta\right)\right) - Q_\theta\left(\mathbf{a},\mathbf{s},\beta\right)]], \tag{8}$$

where $Q_\theta(\mathbf{a},\mathbf{s},\beta) = \frac{1}{NM}\sum_{m,n=1}^{M,N}\zeta^m_{\theta_n}(\mathbf{a},\mathbf{s},\beta)$. We use the mean of the value distribution for policy optimization to avoid conservative policies: Z-functions approximate already underestimated future distribution.

When interacting with the environment, β is sample from a uniform distribution $U_2 = \mathcal{U}[0,b]$ instead of $U_1 = \mathcal{U}[0,a]$, where $0 < b < a$, to get optimistic exploratory behaviors to avoid pessimistic underexploration [25].

IV. Learning Quadrupedal Locomotion

To test our comprehensive learning algorithm DEAC, we utilized a quadruped robot with 12 actuated joints, each capable of delivering a maximum torque of 33.5 Nm. The robot stands at approximately 0.3 m tall and weighs 15 kg. Our robot shares the same actuators as the A1 robot (https://www.unitree.com/a1), but has a weight that's 3 kg heavier than the A1 robot. The stimulation is based on Pybullet [26] simulator.

A. Notations

Let's denote the linear velocity in the robot's base frame as $\mathbf{v}_t \in \mathbb{R}^3$, joint torques as $\boldsymbol{\tau}_t \in \mathbb{R}^{12}$, the position in the world frame as $\mathbf{p}_t \in \mathbb{R}^3$, the acceleration of the base in the robot's base frame $\ddot{a}_t \in \mathbb{R}^3$, the velocity of foot as $\mathbf{v}^f_t \in \mathbb{R}^4$, the binary foot contact indicator vector as $\mathbf{f}^t \in \mathbb{R}^4$, and the total power of the robot at the current time step as $\mathbb{W}_t$. $\mathbf{g}^{\text{ori}}_t \in \mathbb{R}^3$ and $\omega^{\text{ori}}_t \in \mathbb{R}^3$ denote the orientation and angular velocities are measured using the IMU. The body orientation command $\mathbf{c}_t$ is specified by a human operator via remote control. $\mathbf{q}_t \in \mathbb{R}^{12}$ and $\dot{\mathbf{q}}_t \in \mathbb{R}^{12}$ are joint angles and velocities.

B. Reward Design

The reward at time t is defined as the sum of the following quantities:

- Linear Velocity: $\exp\{-0.5(\mathbf{v}^{cmd}_t - \mathbf{v}^x_t)^2\}$
- Linear Velocity Penalties: $-0.4\mathbf{v}^y_t - 0.4\mathbf{v}^z_t$
- Orientation Tracking: $\exp\{-0.5(\mathbf{d}^{cmd}_t - \mathbf{g}^{\text{ori}}_t)^2\}$
- Height Constraint: $-|\mathbf{p}^z_t - \mathbf{p}^{\text{target}}_t|$
- Angle Velocity Penalties: $-\|\omega^{\text{ori}}_t\|^2$
- Self-collision Penalties: $-\mathbf{1}_{\text{selfcollision}}$
- Joint Torque Penalties: $-\|\boldsymbol{\tau}_t\|^2$
- Base Acceleration Penalties: $-\|\ddot{a}_t\|^2$
- Energy: $-\mathbb{W}_t$
- Foot Slip: $-\|\operatorname{diag}(\mathbf{f}^t)\cdot\mathbf{v}^f_t\|^2$

The scaling factor of each reward term is 2.0, 1.0, 2.0, 10, 0.21, 1.3, 0.018, 0.1, 0.012, and 0.3.

TABLE I
Ranges of the Randomized Environmental Parameter

Term	Min	Max	Unit
Contact friction	0.4	1.25	-
Payload mass	-1.0	5.0	kg
Robot mass bias	85%	115%	-
Motor friction	0	0.05	Nm
Control frequency	60	70	Hz
Latency	0	40	ms
IMU bias	-0.05	0.05	rad
IMU noise (std)	0	0.05	rad
Motor encoder bias	-0.08	0.08	rad
External disturbance force	0	15	N
External disturbance torque	0	5	Nm
Gravity direction shift	0	15	degree
Height fields range	-2.5	2.5	cm

TABLE II
Observations

Data	dimension	$\mathbf{o}_t$	$\mathbf{c}_t$	$\mathbf{x}_t$
Desire velocity $\mathbf{v}^{cmd}_t$	1		✓	
Desire direction $\mathbf{d}^{cmd}_t$	1		✓	
Last action $\mathbf{a}_{t-1}$	12	✓		
Joint angle $\mathbf{q}_t$	12	✓		
Joint velocity $\dot{\mathbf{q}}_t$	12	✓		
orientation $\mathbf{g}^{\text{ori}}_t$	3	✓		
Angular velocity ω^{ori}_t	3	✓		
Body velocity $\mathbf{v}_t$	3			✓
Binary foot contact indicator vector $\mathbf{f}^t$	4			✓
Relative position in the world frame $\mathbf{p}_t$	3			✓
Friction coefficient of feet $\mathbf{p}_t$	4			✓
Payload mass	1			✓

C. Training Techniques

Domain randomization: Domain randomization can help the learned controller overcome the reality gap [4], [11]. Before each training episode, we randomly select a set of physical parameters (Table I) to initiate the simulation.

Privileged learning: Learning control policies on rough terrain through RL alone has proven unsuccessful due to the sparse supervised signal and the network's inability to learn the motion within a reasonable time budget [2], [27]. To address this issue, we have adopted the teacher-student training paradigm, also known as privileged learning, an implicit system identification approach similar to previous works [11]. DEAC has also been applied to the teacher-learning process.

D. Control Architecture

Action space: The action $\mathbf{a}_t$ is a 12-dimensional desired joint positions vector. A PD controller is used to calculate torque $\boldsymbol{\tau} = Kp(\hat{\mathbf{q}} - \mathbf{q}) + Kd(\hat{\dot{\mathbf{q}}} - \dot{\mathbf{q}})$, Kp and Kd are manually-specified gains, that are set to 27.5 and 0.5 respectively. The target joint velocities $\hat{\dot{\mathbf{q}}}$ are set to 0.

Teacher Policy: The teacher observation is defined as $\mathbf{s}^{teacher}_t = (\mathbf{o}_{t-H:t}, \mathbf{x}_{t-H:t}, \mathbf{c}_t)$, where $H = 4$. $\mathbf{x}_t$ is the privileged state. The teacher policy $\pi^{teacher}_\theta$ consists of two multilayer perception (MLP) components: a state encoder g_{θ_e} and π_{θ_m}, such that $\mathbf{a}_t = \pi_{\theta_m}(\mathbf{z}_t)$ where $\mathbf{z}_t = g_{\theta_e}(\mathbf{s}_t)$ is a latent representation. We optimize the teacher parameters together using DEAC and the the hyperparameters are shown in Table IV

TABLE III
STUDENT NETWORK ARCHITECTURE

Layer	Student
input	44×100
1	Conv1d(44, 256, kernel size=5, stride=2, dilation=1)
2	Conv1d(256, 256, kernel size=5, stride=2, dilation=2)
3	Conv1d(256, 256, kernel size=5, stride=4, dilation=4)
4	Linear(256, 256)
5	Linear(256, 256)
6	Linear(256, 12)

TABLE IV
HYPERPARAMETERS OF DEAC

Hyperparameters	Value
Optimizer	Adam
Actor learning rate	3×10^{-4}
Temperature learning rate	3×10^{-4}
Critic learning rate (l_{target})	3×10^{-4}
number of hidden layers	2
number of hidden units per layer	256
Discount (γ)	0.99
Nonlinearity	ReLU
Minibatch size	256
Target smoothing coefficient (ρ)	0.005
Update-To-Data (UTD) ratio (G)	20 (Mujoco), 4 (quadrupedal task)
Ensemble size (N)	10
Initial random time steps	5000
Replay buffer capacity	1×10^6
U_1	$\mathcal{U}[0, 0.8]$
U_2	$\mathcal{U}[0, 0.3]$
Number of atoms M	25

Student Policy: The student observation is defined as $\mathbf{o}_{t-H:t}$ where $\mathbf{o}_t = [\mathbf{q}_t, \dot{\mathbf{q}}_t, \mathbf{g}_t^{\text{ori}}, \omega_t^{\text{ori}}, \mathbf{a}_{t-1}]$. The input to the student policy is $\mathbf{s}_t^{student} = (\mathbf{o}_{t-H:t}, \mathbf{c}_t)$, where $H = 100$. Command $\mathbf{c}_t = \{\mathbf{v}_t^{cmd}, \mathbf{d}_t^{cmd}\}$, where $\mathbf{v}_t^{cmd}$ is desire velocity and $\mathbf{d}_t^{cmd}$ is the desire running direction. The student network outputs action $\hat{\mathbf{a}}_t$ and latent representation $\hat{\mathbf{z}}_t$ and its architecture is shown in Table III. The student policy is trained via supervised learning with the dataset aggregation strategy (DAgger) [28]:

$$\mathcal{L} = (\hat{\mathbf{a}}_t - \mathbf{a}_t)^2 + (\hat{\mathbf{z}}_t - \mathbf{z}_t)^2. \tag{9}$$

V. RESULTS AND ANALYSIS

A. Mujoco Results

1) Setups: We compared baselines across four continuous control tasks, namely Walker2d, Hopper, Ant, and Humanoid, which are part of the MuJoCo benchmarks [20]. For mujoco environments, we utilized SAC [9], REDQ [7], TQC [8], and PPO [29] as baseline algorithms. For a fair comparison, we utilized TQC20 as a version of TQC with UTD $G = 20$.

2) Noisy Environment Performance: To explore the effects of sensor and dynamic noise on the performance of algorithms, we applied the Gaussian noise to the state and executed action across all environments. Fig. 4 shows learning curves. Overall, DEAC performs similarly to the baseline methods on easier tasks and surpasses them by a large margin on more challenging tasks. DEAC not only learns faster than SAC but also outperforms it due to SAC's inability to train critics with a high UTD ratio, leading to catastrophic overestimation errors. Although REDQ and TQC can learn all tasks using ensemble and distributional methods, they are slower than DEAC and have worse asymptotic performance. Our quantitative results demonstrate that DEAC's performance is comparable to other methods reported in prior work, indicating that DEAC maintains outstanding final t performance and sample efficiency in a noisy setting.

3) Value Approximation Analysis: Using the Monte Carlo method, we estimate the mean and std of normalized Q-function bias [7] as main analysis indicators to analyze the value approximation quality. The average bias lets us know whether Q_θ is overestimated or underestimated, while std measures whether Q_θ is overfitting. The results are shown in Table V.

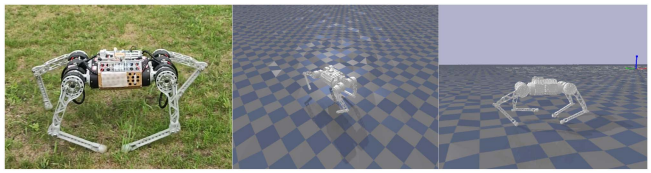

Fig. 3. Left: quadruped robot used in this work. Middle and right: walking on random height fields in simulation.

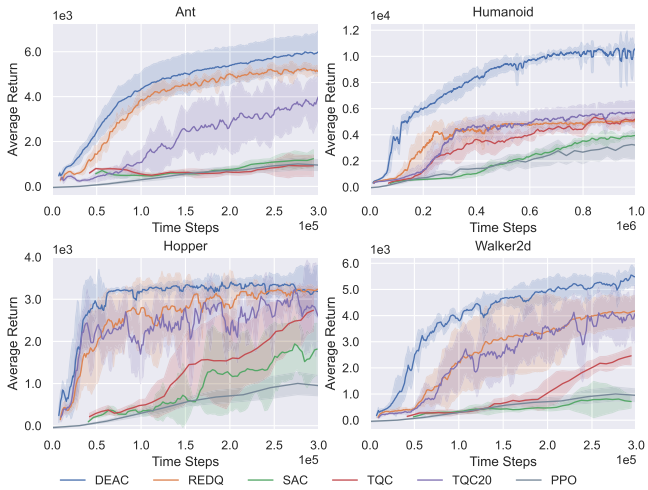

Fig. 4. Learning curves on four noisy Mujoco environments. The horizontal axis indicates the number of time steps. The vertical axis shows the average undiscounted return. The shaded areas denote one standard deviation over eight runs.

TABLE V
NORMALIZED MEAN AND STD OF ESTIMATED NORMALIZED Q-FUNCTION BIAS FOR ALGORITHMS

	DEAC	REDQ	TQC20	TQC
Hopper				
w/ noise	-0.03 ± 0.04	-0.07 ± 0.06	-0.02 ± 0.06	0.02 ± 0.04
w/o noise	-0.05 ± 0.02	-0.05 ± 0.04	-0.01 ± 0.03	-0.03 ± 0.02
Ant				
w/ noise	-0.08 ± 0.04	-0.22 ± 0.08	-0.18 ± 0.16	-0.16 ± 0.22
w/o noise	-0.1 ± 0.029	-0.2 ± 0.032	-0.18 ± 0.11	-0.16 ± 0.16
Walker2d				
w/ noise	-0.02 ± 0.06	-0.16 ± 0.08	0.05 ± 0.2	-0.12 ± 0.11
w/o noise	-0.02 ± 0.04	-0.16 ± 0.03	0.03 ± 0.15	-0.17 ± 0.06
Humanoid				
w/ noise	-0.03 ± 0.07	-0.14 ± 0.14	0.04 ± 0.16	-0.02 ± 0.11
w/o noise	-0.02 ± 0.05	-0.13 ± 0.07	-0.01 ± 0.09	-0.01 ± 0.05

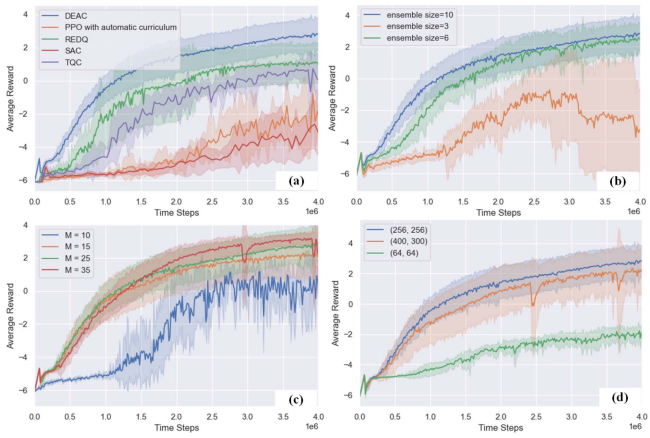

Fig. 5. Learning curves on quadrupedal locomotion tasks.

Even in noisy environments, DEAC efficiently reduces the overestimation error of the Q-function and maintains a lower normalized std of the Q-value bias. REDQ and DEAC have a much lower normalized std of bias, but REDQ's Q-function estimation is too conservative in Humanoid that it traps in a bad locally optimal policy, leading to pessimistic underexploration [25]. When examining TQC and TQC20, it was found that TQC20 has a higher normalized std of bias, indicating that the bias is highly non-uniform, which can be harmful. Due to the potential for overfitting with few samples in distributional RL, using a high UTD ratio for TQC may not be appropriate. In contrast, when looking at Humanoid, which has a high-dimensional state, overfitting is still an issue but has been somewhat alleviated.

B. Learning Quadrupedal Locomotion in Simulation

1) Baselines: The baselines are SAC [9], REDQ [7], TQC [8] and PPO [29] with automatic curriculum. In our experiments, PPO without a curriculum failed when increasing the range of commanded velocities to include high speeds [2]. Curriculum learning slowly increases the complexity of tasks, so that a multitask policy can be successfully learned. We implemented an automatic curriculum based on [11] and [2].

2) Metrics: We compare the performance of DEAC against baselines with a speed command of 2 m/s using the following metrics:

- average reward toward time steps;
- smoothness: the norm of the base acceleration $\ddot{a}_t$, smaller is better;
- mechanical cost of transport (COT);
- speed tracking error (m/s);
- direction tracking error (rad).

The dimensionless cost of transport (COT) [30] is computed to compare the efficiency of the controllers. The mechanical COT is define as $\sum_{12 \text{ actuators}} [\boldsymbol{\tau}\dot{\mathbf{q}}]^+/(\mathbf{m}g\mathbf{v})$, where $\mathbf{m}g$ is the total weight.

3) Simulation Results: Fig. 5(a) shows the learning curves and Table VI shows the final performance comparisons. Even with the help of an automatic curriculum strategy, the sample efficiency and asymptotic performance of PPO are lower than that of DEAC. When the randomness of the environment grows, the performance of baselines decreases. The performance degradation is most pronounced for SAC, which fails to learn when the complete domain randomization parameters are applied. While the performance of baselines in the environment without randomization is comparable, DEAC performs the best with only a slight degradation compared to performance in the randomization-free environment.

4) Wall-Clock Time: In our experiments, DEAC collected 4 million simulated time steps using 4 parallel agents for policy training. The process can be completed in under 10 hours of wall-clock time using an AMD Ryzen 9 5950X CPU and a single NVIDIA RTX 3090 GPU. As a comparison, PPO trained a policy in 12 hours with 40 parallel robots to collect 50 million simulated time steps. Combining with more advanced simulators to implement massively parallel sample collection can further reduce training wall-clock time.

C. Ablations

1) Ensemble Size: We vary the ensemble size of critics. Fig. 5(b) suggests the ensembling consistently improves results. With sufficient computility, we recommend increasing the ensemble size as much as possible.

2) Number of Atoms: We vary the total number of atoms $M \in \{10, 15, 25, 35\}$. The results (Fig. 5(c)) suggest M does not have much influence except for $M = 10$. Learning curves are indistinguishable for $M \geq 15$.

3) Network Architecture: We investigate three MLP architectures commonly seen in the literature: (64, 64), (256, 256), and (400, 300) to the actor and critics (Fig. 5(d)). We find that (256, 256) and (400, 300) have similar performance, while (64, 64) greatly reduces the performance of DEAC.

D. Real-World Experiments

1) Indoor Experiments: We evaluated the learned locomotion controller through qualitative testing by giving random commands using a joystick. The robot was subjected to multiple external pushes to the main body during the experiment. We conducted additional one-hour tests without encountering any failures, demonstrating the controller's robustness. It's worth noting that the controllers functioned smoothly in the real world without requiring further adjustments to the physical system.

We conducted tests to assess the stability of the controller in low-friction scenarios. The outcomes, depicted in Fig. 6, prove that the controller can effectively navigate over slippery surfaces without any destabilization. In contrast, classical controllers tend to fail when the footholds are unstable, but the presented controller maintains stable movement even under such circumstances.

Afterward, we conducted a thorough assessment of the acquired locomotion technique by testing the robot with randomly generated commands. The robot would receive a fresh command every two seconds, which remained constant until the next one was received. The test lasted 30 seconds, during which 15

TABLE VI
PERFORMANCE FOR QUADRUPEDAL LOCOMOTION WITH DIFFERENT ALGORITHMS

	Average Reward	Smoothness	mechanical COT	speed tracking error	direction tracking error
DEAC					
w/o domain randomization	2.72 ± 0.23	3.59 ± 1.54	0.37 ± 0.063	0.02 ± 0.046	0.022 ± 0.01
w/ small randomization	2.69 ± 0.39	3.62 ± 1.70	0.40 ± 0.075	0.04 ± 0.041	0.025 ± 0.013
w/ domain randomization	2.46 ± 0.41	4.07 ± 1.87	0.43 ± 0.081	0.05 ± 0.049	0.031 ± 0.012
SAC					
w/o domain randomization	2.55 ± 0.32	3.61 ± 1.55	0.41 ± 0.053	0.03 ± 0.051	0.031 ± 0.019
w/ small randomization	1.81 ± 0.51	5.32 ± 2.11	0.51 ± 0.075	0.13 ± 0.10	0.065 ± 0.027
w/ domain randomization	-	-	-	-	-
REDQ					
w/o domain randomization	2.69 ± 0.28	3.81 ± 1.33	0.40 ± 0.063	0.03 ± 0.036	0.031 ± 0.017
w/ small randomization	2.38 ± 0.29	4.13 ± 2.10	0.41 ± 0.075	0.04 ± 0.041	0.025 ± 0.013
w/ domain randomization	2.11 ± 0.72	4.47 ± 1.96	0.49 ± 0.081	0.08 ± 0.031	0.039 ± 0.030
TQC					
w/o domain randomization	2.73 ± 0.25	3.11 ± 1.71	0.40 ± 0.057	0.01 ± 0.032	0.019 ± 0.022
w/ small randomization	2.52 ± 0.42	4.22 ± 1.70	0.40 ± 0.079	0.06 ± 0.069	0.045 ± 0.023
w/ domain randomization	2.02 ± 0.37	4.77 ± 2.40	0.49 ± 0.072	0.11 ± 0.061	0.056 ± 0.019
PPO with automatic curriculum					
w/o domain randomization	2.70 ± 0.35	4.01 ± 1.32	0.41 ± 0.051	0.02 ± 0.039	0.024 ± 0.019
w/ small randomization	2.49 ± 0.49	4.94 ± 1.89	0.48 ± 0.088	0.05 ± 0.063	0.041 ± 0.025
w/ domain randomization	2.09 ± 0.82	5.12 ± 2.30	0.54 ± 0.074	0.07 ± 0.077	0.053 ± 0.028

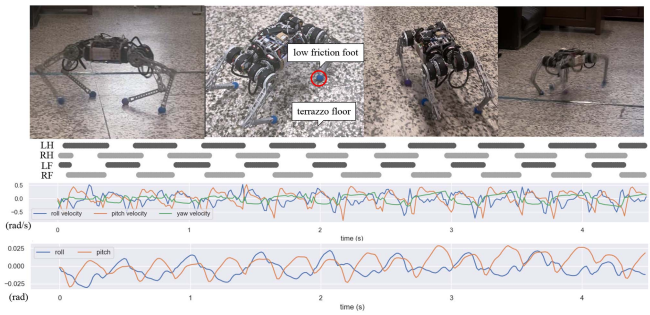

Fig. 6. Indoor experiments. We analyzed the robot walking at 0.8 m/s on a terrazzo floor, and the robot was equipped with low-friction feet. We plotted gait diagrams showing the contact of the four feet (F/H for forward/Hind and R/L for right/left). The lower graph shows the robot body's angular velocity(rad/s) and Euler angles(rad). The robot learns a stable natural gait and avoids slipping events as much as possible in low-friction situations.

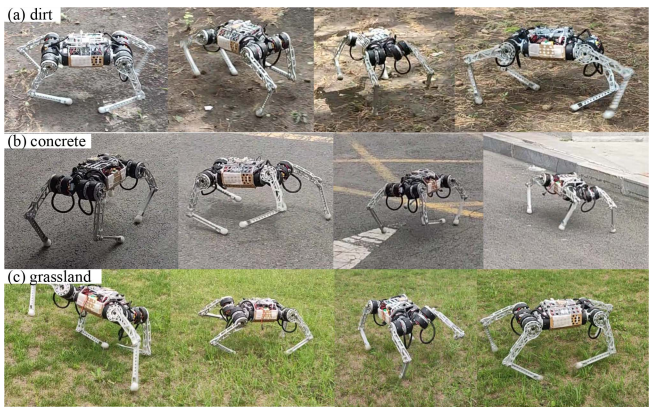

Fig. 7. Outdoor experiments.

random transitions were executed. Finally, our controller could handle the 5 kg payload in the payload analysis.

2) Outdoor Experiments: We demonstrate the performance of our control on several challenging outdoor environments

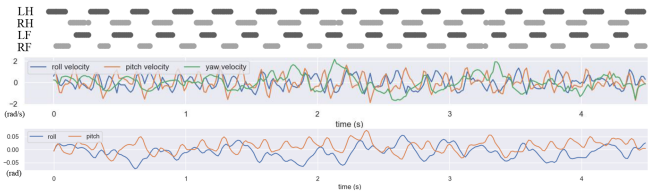

Fig. 8. Analysis of the robot running at 3.5 m/s on grassland. Compared to Fig. 6, the robot changed the walking gait to a running gait without human intervention. The robot encounters challenges from the uneven terrain, causing a higher angular velocity and more intense vibrations than on flat ground. Despite these challenges, the robot maintains its body roll and pitch close to zero, ensuring steady forward locomotion.

TABLE VII
COMPARISON BETWEEN VARIOUS PRIOR WORKS

robot	RL?	max speed (m/s)	weight (kg)	Motor torque (Nm)	Leg L (cm)
ours	Y	3.5	15	33.5	40
A1(Unitree)	N	3.3	12	33.5	40
A1 [27]	Y	1.8	12	33.5	40
GO1 [31]	Y	3.46	12	35.5/23.7	40
Mini Cheetah [2]	Y	3.9	9	16.5	30
Mini Cheetah [32]	N	3.7	9	16.5	30
ANYmal [11]	Y	0.8	30	40	50
ANYmal [33]	N	1.5	30	40	50

as shown in Fig. 7. The robot can successfully walk on dirt, grassland, and concrete without a single failure in all our trials. Traversing these terrains can be challenging for a robot as its feet may sink or stick, causing instability. To maintain stability, the robot has to adjust its footholds dynamically. This can lead to periodic instability while walking. In different terrains, the robot adapted the gait as needed. Even without specifying the gait pattern, the learned controller manifests run, a gait pattern commonly observed in quadrupedal animals. The gait pattern produced by our learned controller is shown in Fig. 8.

As shown in Table VII, although the robot used in this letter is 3 kg heavier than the A1 robot, the presented controller allows for higher running speeds. On flat terrain, the robot achieved a mechanical COT of 0.39 while running at 2 m/s with a mechanical power consumption of $117\,W$. While on rough terrain, the COT is 0.44 and running at 2 m/s with a power of $130\,W$.

VI. Conclusion

We introduced the DEAC algorithm, which enhances the controller's performance against environmental unpredictability, ensuring reliable learning in a complex domain randomization setting. DEAC allows a legged robot to walk on various terrains while learning its velocity-tracking controller. The learned controller can operate directly on a real robot, relying solely on proprioceptive data. However, a blind robot has limitations, as larger perturbations such as sudden falls while going downstairs or multiple leg obstructions from rocks may cause failures. For a more dependable walking robot, proprioception and exteroception with an onboard vision sensor are necessary. This demands higher robustness from the controller. Sensors such as vision and LIDAR play a crucial role in guiding long-range, fast locomotion, which is an area of focus for future work.

References

[1] Y. Yang, T. Zhang, E. Coumans, J. Tan, and B. Boots, "Fast and efficient locomotion via learned gait transitions," in *Proc. Conf. Robot Learn.*, 2022, pp. 773–783.

[2] G. Margolis, G. Yang, K. Paigwar, T. Chen, and P. Agrawal, "Rapid locomotion via reinforcement learning," *Robot.: Sci. Syst.*, 2022.

[3] X. Chen, J. Hu, C. Jin, L. Li, and L. Wang, "Understanding domain randomization for sim-to-real transfer," in *Proc. Int. Conf. Learn. Representations*, 2021.

[4] J. Tan et al., "Sim-to-real: Learning agile locomotion for quadruped robots," *Robot.: Sci. Syst.*, 2018.

[5] W. Dabney, M. Rowland, M. Bellemare, and R. Munos, "Distributional reinforcement learning with quantile regression," in *Proc. AAAI Conf. Artif. Intell.*, 2018, pp. 2892–2901.

[6] D. Yang, L. Zhao, Z. Lin, T. Qin, J. Bian, and T.-Y. Liu, "Fully parameterized quantile function for distributional reinforcement learning," in *Proc. Adv. Neural Inf. Process. Syst.*, 2019, vol. 32.

[7] X. Chen, C. Wang, Z. Zhou, and K. W. Ross, "Randomized ensembled double Q-learning: Learning fast without a model," in *Proc. Int. Conf. Learn. Representations*, 2020.

[8] A. Kuznetsov, P. Shvechikov, A. Grishin, and D. Vetrov, "Controlling overestimation bias with truncated mixture of continuous distributional quantile critics," in *Proc. Int. Conf. Mach. Learn.*, 2020, pp. 5556–5566.

[9] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," in *Proc. Int. Conf. Mach. Learn.*, 2018, pp. 1861–1870.

[10] C. Lyle, M. Rowland, G. Ostrovski, and W. Dabney, "On the effect of auxiliary tasks on representation dynamics," in *Proc. Int. Conf. Artif. Intell. Statist.*, 2021, pp. 1–9.

[11] T. Miki, J. Lee, J. Hwangbo, L. Wellhausen, V. Koltun, and M. Hutter, "Learning robust perceptive locomotion for quadrupedal robots in the wild," *Sci. Robot.*, vol. 7, no. 62, 2022, Art. no. eabk2822.

[12] C. Weibel, G. Valsecchi, H. Kolvenbach, and M. Hutter, "Towards legged locomotion on steep planetary terrain," in *Proc. IEEE/RSJ 36th Int. Conf. Intell. Robots Syst.*, 2023, pp. 786–792.

[13] G. Bledt, P. M. Wensing, S. Ingersoll, and S. Kim, "Contact model fusion for event-based locomotion in unstructured terrains," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2018, pp. 4399–4406.

[14] G. Lu et al., "Whole-body motion planning and control of a quadruped robot for challenging terrain," *J. Field Robot.*, vol. 40, no. 6, pp. 1657–1677, 2023.

[15] W. Zhao, J. P. Queralta, and T. Westerlund, "Sim-to-real transfer in deep reinforcement learning for robotics: A survey," in *Proc. IEEE Symp. Ser. Comput. Intell.*, 2020, pp. 737–744.

[16] N. Sontakke, H. Chae, S. Lee, T. Huang, D. W. Hong, and S. Hal, "Residual physics learning and system identification for sim-to-real transfer of policies on buoyancy assisted legged robots," in *IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2023, pp. 392–399.

[17] W. Dabney, G. Ostrovski, D. Silver, and R. Munos, "Implicit quantile networks for distributional reinforcement learning," in *Proc. Int. Conf. Mach. Learn.*, 2018, pp. 1096–1105.

[18] L. Schneider, J. Frey, T. Miki, and M. Hutter, "Learning risk-aware quadrupedal locomotion using distributional reinforcement learning," in *Proc. 41st IEEE Conf. Robot. Automat.*, 2024.

[19] K. Lee, M. Laskin, A. Srinivas, and P. Abbeel, "Sunrise: A simple unified framework for ensemble learning in deep reinforcement learning," in *Proc. Int. Conf. Mach. Learn.*, 2021, pp. 6131–6141.

[20] E. Todorov, T. Erez, and Y. Tassa, "MuJoCo: A physics engine for model-based control," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2012, pp. 5026–5033.

[21] V. François-Lavet et al., "An introduction to deep reinforcement learning," *Found. Trends Mach. Learn.*, vol. 11, no. 3/4, pp. 219–354, 2018.

[22] 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 Robot. Automat. Lett.*, vol. 7, no. 2, pp. 4630–4637, Apr. 2022.

[23] T. Schaul, D. Horgan, K. Gregor, and D. Silver, "Universal value function approximators," in *Proc. Int. Conf. Mach. Learn.*, 2015, pp. 1312–1320.

[24] H. Van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double Q-learning," in *Proc. AAAI Conf. Artif. Intell.*, 2016, pp. 2094–2100.

[25] K. Ciosek, Q. Vuong, R. Loftin, and K. Hofmann, "Better exploration with optimistic actor critic," in *Proc. Adv. Neural Inf. Process. Syst.*, 2019, vol. 32.

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

[27] A. Kumar, Z. Fu, D. Pathak, and J. Malik, "RMA: Rapid motor adaptation for legged robots," *Robot.: Sci. Syst.*, 2021.

[28] S. Ross, G. Gordon, and D. Bagnell, "A reduction of imitation learning and structured prediction to no-regret online learning," in *Proc. 14th Int. Conf. Artif. Intell. Statist.*, 2011, pp. 627–635.

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

[30] J. Lee, J. Hwangbo, L. Wellhausen, V. Koltun, and M. Hutter, "Learning quadrupedal locomotion over challenging terrain," *Sci. Robot.*, vol. 5, no. 47, 2020, Art. no. eabc5986.

[31] J. Wu, G. Xin, C. Qi, and Y. Xue, "Learning robust and agile legged locomotion using adversarial motion priors," *IEEE Robot. Automat. Lett.*, vol. 8, no. 8, pp. 4975–4982, Aug. 2023.

[32] D. Kim, J. Di Carlo, B. Katz, G. Bledt, and S. Kim, "Highly dynamic quadruped locomotion via whole-body impulse control and model predictive control," 2019, *arXiv:1909.06586*.

[33] W. Bosworth, J. Whitney, S. Kim, and N. Hogan, "Robot locomotion on hard and soft ground: Measuring stability and ground properties in-situ," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2016, pp. 3582–3589.