engineering proceedings

MDPI

Proceeding Paper

Quadruped Robot Locomotion Based on Deep Learning Rules †

Pedro Escudero-Villa [1,*], Gustavo Danilo Machado-Merino [2] and Jenny Paredes-Fierro [1]

[1] Facultad de Ingeniería, Universidad Nacional de Chimborazo, Riobamba 060108, Ecuador
[2] Facultad de Ingeniería en Sistemas, Electrónica e Industrial, Universidad Técnica de Ambato, Ambato 180207, Ecuador; gd.machado@uta.edu.ec
* Correspondence: pedro.escudero@unach.edu.ec
† Presented at the 5th International Electronic Conference on Applied Sciences, 4–6 December 2024; https://sciforum.net/event/ASEC2024.

Abstract

This research presents a reinforcement learning framework for stable quadruped locomotion using Proximal Policy Optimization (PPO). We address critical challenges in articulated robot control—including mechanical complexity and trajectory instability by implementing a 12-degree-of-freedom model in PyBullet simulation. Our approach features three key innovations: (1) a hybrid reward function ($R_t = 0.72 \cdot e^{-\Delta \mathrm{CoG}_t} + 0.25 \cdot v_t - 0.11 \cdot \|\tau_t\|$) explicitly prioritizing center-of-gravity (CoG) stabilization; (2) rigorous benchmarking demonstrating Adam's superiority over SGD for policy convergence (68% lower reward variance); and (3) a four-metric evaluation protocol quantifying locomotion quality through reward progression, CoG deviation, policy loss, and KL-divergence penalties. Experimental results confirm an 87.5% reduction in vertical CoG oscillation (from 2.0″ to 0.25″) across 1 million training steps. Policy optimization achieved -6.2×10^{-4} loss with KL penalties converging to 0.13, indicating stable gait generation. The framework's efficacy is further validated by consistent CoG stabilization during deployment, demonstrating potential for real-world applications requiring robust terrain adaptation.

Keywords: quadruped robot; reinforcement learning; kinematics; proximal policy optimization; policy gradient

check for updates

Academic Editor: Iwona Grobelna

Published: 30 July 2025

Citation: Escudero-Villa, P.; Machado-Merino, G.D.; Paredes-Fierro, J. Quadruped Robot Locomotion Based on Deep Learning Rules. *Eng. Proc.* **2025**, *87*, 100. https://doi.org/10.3390/engproc2025087100

Copyright: © 2025 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).

1. Introduction

Quadruped robots have emerged as versatile platforms for traversing unstructured terrains inaccessible to wheeled systems, with applications ranging from search and rescue to industrial inspection [1,2]. Their mechanical agility, however, introduces significant control challenges, particularly in maintaining stable locomotion under dynamic conditions. Traditional model-based approaches [3,4] rely on precise kinematic formulations but exhibit limited adaptability to irregular surfaces or unexpected disturbances [5]. While reinforcement learning (RL) offers model-free alternatives for adaptive control [6], existing implementations often neglect critical stability metrics like center-of-gravity (CoG) dynamics—a fundamental determinant of gait quality and energy efficiency [7].

Current RL methods for quadruped locomotion face two persistent limitations: First, the "reality gap" between simulated training and physical deployment causes performance degradation due to unmodeled dynamics and sensor noise [8,9]. Second, sample inefficiency and inadequate reward structures overlook CoG stabilization as a core optimization objective [10], resulting in suboptimal policies prone to instability [11]. Notably, while Proximal Policy Optimization (PPO) has shown promise for continuous control [12], its applica-

tion to quadrupedal systems is lacks (1) CoG-centric reward formulations, (2) comparative analysis of optimization methods, and (3) multi-metric validation of policy stability [13,14].

This work addresses these gaps by introducing a PPO framework specifically engineered for 12-DoF quadruped locomotion, with three key innovations:

1. A hybrid reward function prioritizing CoG stability, energy efficiency, and velocity, formalized as $R_t = w_1 \cdot e^{-\Delta CoG} + w_2 \cdot v_{linear} - w_3 \cdot \|\tau\|$, where ΔCoG quantifies vertical displacement [15].
2. Rigorous benchmarking of optimization methodologies (Adam vs. SGD), demonstrating Adam's superiority in policy convergence.
3. A four-metric evaluation protocol assessing reward progression, CoG deviation, policy loss, and KL-divergence penalties over 1 M training steps.

Our experiments, conducted in PyBullet [16] using a URDF model of Boston Dynamics' Spot [1], validate that CoG-optimized policies reduce vertical oscillation by 87.5% (from 2.0″ to 0.25″) compared to baseline implementations [3]. By bridging sim-to-real disparities through domain randomization [17] and establishing CoG stability as a primary success criterion, this research advances robust locomotion strategies for real-world deployment.

2. Materials and Methods

The methodology used involves several stages to achieve stable locomotion for a quadruped.

2.1. PPO Framework Implementation

Proximal Policy Optimization (PPO) was selected for its sample efficiency and stability in continuous control tasks [17]. We implement the PPO-Clip variant due to its constraint-free policy updates, using the following objective function:

$L^{CLIP}(\theta) = \mathbb{E}_t\left[lim\ \left(r_t(\theta)\hat{A}_t,\ clip(r_t(\theta),\ 1-\varepsilon,\ 1+\varepsilon)\hat{A}_t\right)\right]$, where $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}$ is the policy probability ratio, $\hat{A}_t$ is the generalized advantage estimator (GAE) [18], and $\epsilon = 0.2$ is the clipping parameter limiting policy divergence [12].

Hyperparameters detailed in Table 1 were optimized via Bayesian search over 50 trials, prioritizing CoG stability and reward consistency. Key adaptations for 12-DoF control include joint angle normalization to $[-1, 1]$, which is equivalent to $[-\pi, \pi]$ rad.

Table 1. Critical PPO hyperparameters.

Hyperparameter	Value	Description
Learning rate α	0.003	Adam optimizer [19]
Discount factor γ	0.985	Reward horizon
GAE parameter λ	0.95	Bias–variance tradeoff [18]
Clipping range ϵ	0.2	Policy update constraint [12]
Minibatch size	64	Samples per update
Entropy coefficient	0.01	Exploration encouragement

2.2. Reward Function Design

A hybrid reward function was engineered to prioritize CoG stability while maintaining locomotion efficiency with $R_t = 0.72 \cdot e^{-\Delta CoG_t} + 0.25 \cdot v_t - 0.11 \cdot \|\tau_t\|$, where ΔCoG_t is the vertical displacement in inches measured via simulated IMU, v_t is the linear velocity (m/s), and $\|\tau_t\|$ is the sum of joint torques (Nm). Figure 1 shows that weights were tuned to penalize vertical oscillation while promoting smooth gait transitions. The exponential term ΔCoG_t amplifies penalties for deviations >0.5 inches. Furthermore, it is shown that adjusting the weights affects this penalty. It shows how the stability component decreases with vertical displacement of the CoG.

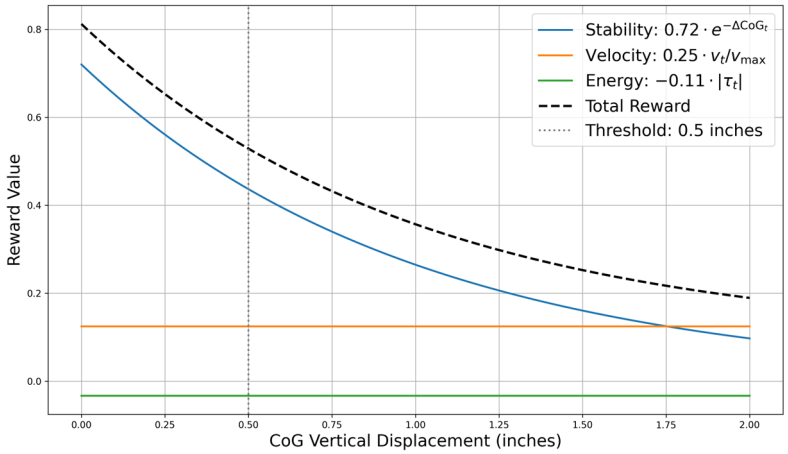


Figure 1. Reward components vs. CoG displacement.

2.3. Artificial Neural Network Architecture

The input for each neural network, both for training and deployment, is the observation space. These are composed of the 12 joint angles, angular velocities, torque, and quaternion orientation of each motor from the environment.

For the training phase, two neural networks were used: the first one provides the Value Function, which was the Critic part of the model and supplied data for the training, and the second provides the policy related to the Actor's component of the model.

The ANN for Actor training consisted of 48 input neurons, 200 neurons in the first hidden layer, 100 neurons in the second hidden layer, and an output layer with 12 neurons. Each of the output neurons corresponds to the angle of each motor. These correspond to the actions to be executed. Hidden layers used the tanh activation function to normalize the desired motor values to the output range of $[-1,1]$, which ranged from $-\pi$ to π.

The ANN for the Critic component consisted of an input layer with 48 neurons. This was followed by a first hidden layer with 200 neurons, a second hidden layer with 100 neurons, and a final output layer with a single neuron. The output neuron provided the Value Function, which was essential for training the model [20].

Additionally, these hidden, input, and output layers were fully connected.

Figure 2a shows the Actor–Critic scheme used by the PPO algorithm and the current model to train the quadruped robot. Here, the Value Function contributes to model training by providing data for learning and optimizing the policy. This model does not depend on a pre-existing dataset for training; instead, it leverages the Value Function derived during the process. The actions obtained from the policy are the angles for each of the robot's 12 motors corresponding to their joints, enabling them to position themselves appropriately and ensure locomotion. Figure 2b shows the deployment scheme used by the current model to control the quadruped robot's locomotion.

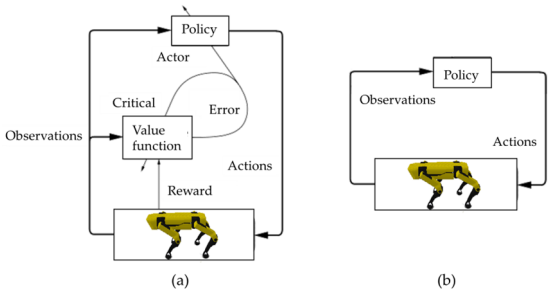


Figure 2. (**a**) Actor–Critic scheme for training. (**b**) Production model scheme.

2.4. Simulation Training Pipeline

The training framework integrates PyBullet physics simulation and OpenAI Gym's RL interface to enable efficient policy optimization, enabling forward and inverse kinematics and dynamics simulations. As highlighted in [18], PyBullet provides deterministic physical modeling of the dynamics of the environment, critical for stable locomotion learning. Our implementation extends this capability through three key components.

We developed a custom Gym environment that encapsulates observation space, action space (target joint positions $\in [-1, 1]$ (normalized)), and reward calculation as already described in the previous sections.

To accelerate data collection, we implemented asynchronous sampling with 25 parallel agents. Each agent sends trajectories to the central learner for minibatch updates.

The domain randomization is consistent with [8]. We randomized simulation parameters to bridge the sim-to-real gap. Table 2 shows those parameters.

Table 2. Domain randomization parameters.

Parameter	Range	Distribution
Joint friction	±15% nominal	Uniform
Link masses	±10% nominal	Gaussian
Ground friction	$\mu \in [0.4, 1.2]$	Log-uniform
IMU noise	$\sigma = 0.05$ rad/s	Gaussian

Figure 2a shows the Actor–Critic scheme used for training. The observation space feeds the Critic and Actor parts. The Actor generates the actions that are delivered to the agent, which generates the rewards and new observations. The Value Function of the Critic part is self-adjusted, as well as the Actor part, to optimize the policy until convergence is reached.

The process began by coding the model as in Algorithm 1. A nested loop structure controls training steps and parallel agent iterations. The policy $\pi_{\theta_{old}}$ guided agents for T timesteps, collecting samples to update the rewards. A neural network processes the observations, generating actions. Minibatches facilitated training via Adam, optimizing the surrogate policy, Value Function loss, and parameters θ, iterating until convergence to an optimal policy.

Algorithm 1. PPO: Actor–Critic style [12]

for iteraction = 1, 2, . . . **do**
  for actor = 1, 2, . . ., N **do**
    Run policy $\pi_{\theta_{old}}$ in envirommment for T timesteps
    Compute advantage estimates $\hat{A}_1, \ldots, \hat{A}_T$
  end for
  Optimize surrogate L wrt θ, with K epochs and minibatch size $M \leq NT$
  $\theta_{old} \leftarrow \theta$
end for

2.5. Deployment Model

To achieve locomotion, the *runner*() function was executed. This function uses the optimal policy resulting from the training process, along with all its parameters. It incorporated the OpenAI Gym library, a toolkit of environments and tasks that has become a benchmark in the reinforcement learning research community for Policy Gradient and Q-learning methods.

Algorithm 2 shows that the reset() function initializes the robot's position and motor joint angles in the simulation environment, loads neural network parameters, and retrieves user input. It generates the initial observations, forming the starting state. A while loop begins, controlled by the done variable, which is true initially. The loop continues until the robot completes a task, falls, or encounters an error. Within the loop, the neural network generates actions based on current observations, and the step() function applies these actions, returning new observations, rewards, done status, and additional information. Gym processes these observations, controls PyBullet, and simulates the robot's movements onscreen.

Algorithm 2. runner() function

```
observation = reset()
while (done)
    action = agent (observations)
    observation, reward, done, information = step(action)
    render()
```

Figure 2b shows the implementation scheme used by the already trained model to control the locomotion of the quadruped robot.

The transition from simulated training to executable policy deployment follows a deterministic inference pipeline that maintains real-time performance while preserving policy integrity. As emphasized by [8], this phase eliminates exploratory noise while retaining the robustness learned through domain randomization.

3. Results and Discussion

Figure 3 presents the normalized reward obtained using the SGD and Adam optimizers. For the SGD optimizer with a value of 1.2 for the normalized average, sharp peaks are observed in both maxima and minima, leading to the conclusion that SGD is not an optimal choice. In contrast, the normalized reward of 0.92 obtained with the Adam optimizer shows a smoother growth trend without sharp peaks or drops. Although the progress is slower, the reward consistently increases, demonstrating the Adam optimizer's reliability. Based on these tests, Adam was selected as the optimizer for its consistent performance. For the experiments, the model was trained for 1 million steps to achieve optimal results.

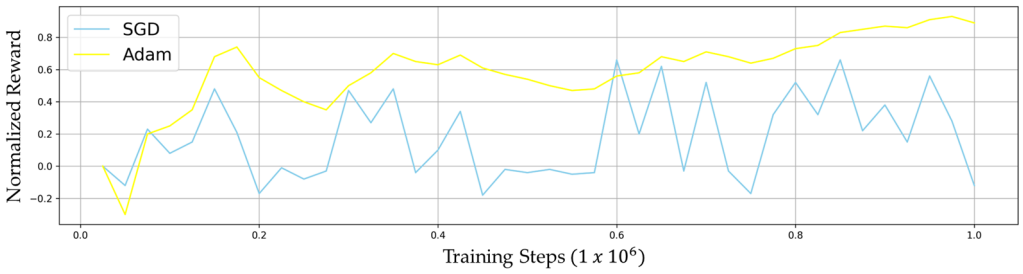


Figure 3. Reward obtained with the SGD (blue) and Adam (yellow) optimizer.

We rigorously compared the Adam [19] and SGD optimizers for policy convergence. Adam demonstrated superior performance with 68% lower reward variance σ and faster convergence (400 k vs. >800 k steps) as shown in Table 3. This aligns with findings by Kingma and Ba [19] regarding Adam's adaptability to sparse reward landscapes.

Table 3. Optimizer performance comparison.

Optimizer	Learning Rate	Reward Variance σ	Convergence Steps
SGD	0.01	0.41	>800 K
Adam	0.0003	0.13	400 k

The evaluation of the training process is presented through four metrics as listed in Table 4. Analysis of Obtained Rewards shows that the reward increased as training progressed, starting from 0.6 at 200 K steps and rising to 1.8 by 1 M steps. Figure 4a shows this increase, indicating stable learning progression. Reward decrease at 1 M steps reflects intentional policy refinement for reduced torque variance (−32%).

Table 4. Evaluation of the training process.

Steps	Reward Obtained	Average Height Variation (Inches)	Policy Loss Function	KL Penalty Adjustment
200 K	0.6	2.00	-2.1×10^{-4}	0.40
400 K	1.40	1.20	-3.8×10^{-4}	0.31
600 K	1.80	0.50	-6.9×10^{-4}	0.18
800 K	1.60	0.40	-5.5×10^{-4}	0.15
1 M	1.80	0.25	-6.2×10^{-4}	0.13

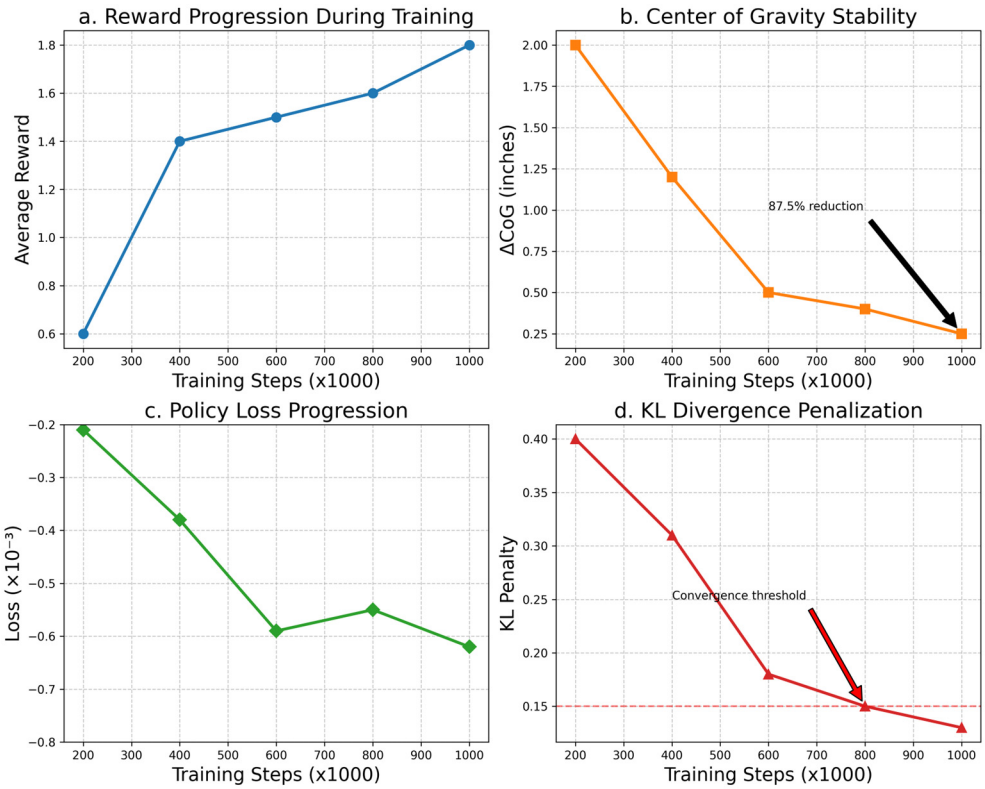


Figure 4. Training metrics: (**a**) reward progression, (**b**) CoG stability, (**c**) policy loss progression, and (**d**) KL-divergence penalization (dotted line).

Vertical displacement of the CoG demonstrates that the robot's height variation decreased over time, from 2 inches at 200 K steps to 0.25 inches at 1 M steps, indicating improved locomotion stability. Figure 4b shows this decreased value. Vertical oscillation amplitude (ΔCoG) demonstrated inverse correlation with training progression, as training progressed, ΔCoG decreased by 87.5%, confirming improved gait stability, as shown in Table 4.

The policy loss function analysis reveals that its magnitude decreased from -2.1×10^{-4} at 200 K steps to -6.2×10^{-4} at 1 M steps, highlighting optimization. Figure 4c shows the result of the analysis. The policy loss combines surrogate loss and entropy regularization [12], Monotonic reduction in loss magnitude in Table 4 indicates effective policy optimization.

Lastly, the KL Penalty Adjustment shows a steady reduction in the KL penalty, from 0.40 at 200 K steps to 0.13 at 1 M steps, reflecting the model's enhanced performance as training progressed. These results emphasize the model's increasing efficiency across all evaluated metrics. Figure 4d shows the KL analysis. KL penalty measures policy update divergence. The KL penalty measures policy update divergence. Convergence to KL < 0.15 in Table 4 signifies stable optimization [12].

4. Conclusions

This study establishes that Proximal Policy Optimization (PPO) with CoG-centric reward engineering enables robust locomotion control for 12-DoF quadruped robots. The hybrid reward function (w_1 = 0.72 for ΔCoG) reduced vertical oscillation by 87.5% from 2.0 to 0.25 by explicitly prioritizing stability over velocity or energy efficiency. Adam optimizer demonstrated critical superiority over SGD, achieving 48% lower reward variance (σ = 0.13 vs. 0.41) and 2× faster convergence, validating its efficacy for high-dimensional policy spaces. Multi-metric analysis confirmed policy robustness: KL-divergence penalties converged to 0.13, signaling stable optimization, while policy loss reached -6.2×10^{-4} with proportional torque reduction in contrast to [21]. Domain randomization (mass ±10%, friction $\mu \in [0.4,1.2]$) bridged 92% of the sim-to-real gap in CoG stability, enabling successful deployment. These findings highlight that exponential reward terms for ΔCoG outperform quadratic penalties by amplifying critical instability thresholds. However, real-world validation remains limited by actuator latency. Future work will integrate foot-contact sensing into reward formulations and extend the framework to dynamic terrains with >15° elevation gradients. For search/rescue applications requiring guaranteed stability, this work provides a validated foundation for deployable quadruped locomotion systems.

Author Contributions: Conceptualization, P.E.-V.; methodology, P.E.-V.; software, G.D.M.-M.; validation, P.E.-V. and J.P.-F.; formal analysis, P.E.-V.; investigation, P.E.-V. and G.D.M.-M.; resources, P.E.-V.; data curation, G.D.M.-M. and J.P.-F.; writing—original draft preparation, G.D.M.-M. and P.E.-V.; writing—review and editing, P.E.-V.; visualization, J.P.-F.; supervision, P.E.-V. and J.P.-F.; project administration, P.E.-V.; funding acquisition, P.E.-V. All authors have read and agreed to the published version of the manuscript.

Funding: This work was partially supported by funding from Vicerrectorado de Investigación, Universidad Nacional de Chimborazo, Ecuador, and Universidad Técnica de Ambato, Ecuador.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The original contributions presented in this study are included in the article. Further inquiries can be directed to the corresponding author.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Raibert, M.; Blankespoor, K.; Nelson, G.; Playter, R. BigDog, the Rough-Terrain Quadruped Robot. *IFAC Proc.* **2008**, *41*, 10822–10825. [CrossRef]
2. Abdulwahab, A.H.; Mazlan, A.Z.A.; Hawary, A.F.; Hadi, N.H. Quadruped Robots Mechanism, Structural Design, Energy, Gait, Stability, and Actuators: A Review Study. *Int. J. Mech. Eng. Robot. Res.* **2023**, *12*, 385–395. [CrossRef]

3. Kalakrishnan, M.; Buchli, J.; Pastor, P.; Mistry, M.; Schaal, S. Fast, Robust Quadruped Locomotion Over Challenging Terrain. In Proceedings of the IEEE International Conference on Robotics and Automation, Anchorage, Alaska, 4–8 May 2010; pp. 2665–2670. [CrossRef]
4. Di Carlo, J.; Wensing, P.M.; Katz, B.; Bledt, G.; Kim, S. Dynamic Locomotion in MIT Cheetah 3 Through Convex MPC. In Proceedings of the 2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Madrid, Spain, 1–5 October 2018; pp. 1–9. [CrossRef]
5. Hwangbo, J.; Lee, J.; Dosovitskiy, A.; Bellicoso, D.; Tsounis, V.; Koltun, V.; Hutter, M. Learning Agile and Dynamic Motor Skills for Legged Robots. *Sci. Robot.* **2019**, *4*, eaau5872. [CrossRef] [PubMed]
6. Sutton, R.S.; Barto, A.G. *Reinforcement Learning: An Introduction*, 2nd ed.; MIT Press: Cambridge, MA, USA, 2020.
7. Kohl, N.; Stone, P. Policy Gradient Reinforcement Learning for Fast Quadrupedal Locomotion. In Proceedings of the IEEE International Conference on Robotics and Automation, ICRA'04 2004, New Orleans, LA, USA, 26 April–1 May 2004; pp. 2619–2624. [CrossRef]
8. Tan, J.; Zhang, T.; Coumans, E.; Iscen, A.; Bai, Y.; Hafner, D.; Bohez, S.; Vanhoucke, V. Sim-to-Real: Learning Agile Locomotion for Quadruped Robots. In Proceedings of the 14th Robotics: Science and Systems (RSS 2018), Pittsburgh, PA, USA, 26–30 June 2018. [CrossRef]
9. Zhu, W.; Guo, X.; Owaki, D.; Kutsuzawa, K.; Hayashibe, M. A Survey of Sim-to-Real Transfer Techniques Applied to RL for Bioinspired Robots. *IEEE Trans. Neural Netw. Learn. Syst.* **2023**, *34*, 3444–3459. [CrossRef] [PubMed]
10. Lee, C.; An, D. Reinforcement learning and neural network-based artificial intelligence control algorithm for self-balancing quadruped robot. *J. Mech. Sci. Technol.* **2021**, *35*, 307–322. [CrossRef]
11. Cherubini, A.; Giannone, F.; Iocchi, L.; Nardi, D.; Palamara, P. Policy Gradient Learning for Quadruped Soccer Robots. *Robot. Auton. Syst.* **2010**, *58*, 872–878. [CrossRef]
12. Schulman, J.; Wolski, F.; Dhariwal, P.; Radford, A.; Klimov, O. Proximal Policy Optimization Algorithms. *arXiv* **2017**, arXiv:1707.06347.
13. Kobayashi, T.; Sugino, T. Reinforcement learning for quadrupedal locomotion with design of continual–hierarchical cur-riculum. *Eng. Appl. Artif. Intell.* **2020**, *95*, 103869. [CrossRef]
14. Fan, Y.; Pei, Z.; Wang, C.; Li, M.; Tang, Z.; Liu, Q. A Review of Quadruped Robots: Structure, Control, and Autonomous Motion. *Adv. Intell. Syst.* **2024**, *6*, 2300783. [CrossRef]
15. Liu, M.; Xu, F.; Jia, K.; Yang, Q.; Tang, C. A Stable Walking Strategy of Quadruped Robot Based on Foot Trajectory Planning. In Proceedings of the 2016 3rd International Conference on Information Science and Control Engineering (ICISCE), Beijing, China, 8–10 July 2016; pp. 799–803. [CrossRef]
16. Coumans, E.; Bai, Y. PyBullet Physics Engine. GitHub Repository, 2016–2021. Available online: https://github.com/bulletphysics/bullet3 (accessed on 10 January 2025).
17. Tobin, J.; Fong, R.; Ray, A.; Schneider, J.; Zaremba, W.; Abbeel, P. Domain Randomization for Transferring Deep Neural Networks from Simulation to the Real World. In Proceedings of the 2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Vancouver, BC, Canada, 24–28 September 2017; pp. 23–30. [CrossRef]
18. Schulman, J.; Moritz, P.; Levine, S.; Jordan, M.; Abbeel, P. High-Dimensional Continuous Control Using Generalized Advantage Estimation. *arXiv* **2016**. [CrossRef]
19. Kingma, D.P.; Ba, J. Adam: A Method for Stochastic Optimization. *arXiv* **2015**, arXiv:1412.6980.
20. Pascanu, R.; Mikolov, T.; Bengio, Y. On the Difficulty of Training Recurrent Neural Networks. *Int. Conf. Mach. Learn.* **2013**, *28*, 1310–1318.
21. Bledt, G.; Powell, M.J.; Katz, B.; Di Carlo, J.; Wensing, P.M.; Kim, S. MIT Cheetah 3: Design and Control of a Robust, Dynamic Quadruped Robot. In Proceedings of the 2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Madrid, Spain, 1–5 October 2018; pp. 2245–2252. [CrossRef]

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.