Quadruped Locomotion with Central Pattern Generators and Deep Reinforcement Learning

[*]Legged Robots Mini-Project 2

Xinran Wang
xinran.wang@epfl.ch

Hsu-li Huang
hsu-li.huang@epfl.ch

Dacheng Zhou
dacheng.zhou@epfl.ch

I. Introduction

Learning-based locomotion has become a central paradigm for controlling legged robots, offering the potential to generate agile and adaptive behaviors without explicitly modeling complex contact dynamics. However, the success of such approaches strongly depends on how the task is formulated and how physical objectives are encoded through observations, actions, and rewards. Besides classical RL for locomotion (PD Control, Cartesian PD Control), Central Pattern Generators (CPG) is also powerful.

II. Methodology

A. Part 1: Central Pattern Generators

1) Adjustment of PD gain: Both joint-space PD control and Cartesian-space PD control result in the same second-order closed-loop dynamics.

Joint PD Control:

$$\tau = K_p(q_d - q) + K_d(\dot{q}_d - \dot{q}), \tag{1}$$

with joint dynamics

$$I\ddot{q} = \tau. \tag{2}$$

Cartesian PD Control:

$$F = K_p(x_d - x) + K_d(\dot{x}_d - \dot{x}), \tag{3}$$

with Cartesian dynamics

$$M\ddot{x} = F. \tag{4}$$

Both controllers exhibit identical second-order mass–spring–damper behavior. The only difference is the inertia term: joint inertia I versus Cartesian inertia M.

Closed-loop form:

$$\frac{\text{Output}(s)}{\text{Reference}(s)} = \frac{K_d s + K_p}{\mathcal{I}s^2 + K_d s + K_p}, \tag{5}$$

where $\mathcal{I} = I$ for joint space and $\mathcal{I} = M$ for Cartesian space.

Natural frequency:

$$\omega_n = \sqrt{\frac{K_p}{\mathcal{I}}} \tag{6}$$

Damping ratio

$$\zeta = \frac{K_d}{2\sqrt{\mathcal{I}K_p}} \tag{7}$$

Settling time (2%)

$$T_s \approx \frac{4}{\zeta\omega_n} \approx \frac{8\mathcal{I}}{K_d} \tag{8}$$

stiffness:

$$K_p \tag{9}$$

TABLE I: PD Gain Tuning Effects

Increase	Effect
K_p	Faster, stiffer, more oscillatory
K_d	Less oscillation, slower rise
K_p and K_d	Fast and stable response
ζ	Reduced overshoot, quicker settling
I or M	Slower dynamics

Why PI or PID control is not used: An integral term is not introduced because the joint-level controller tracks dynamically generated CPG trajectories rather than fixed references. In this setting, integral action may accumulate tracking errors over time, leading to integrator windup and degraded stability without providing meaningful steady-state error reduction.

B. Part 2: Deep Reinforcement Learning

1) Observation space: In addition to joint angles, joint velocities, and base orientation, we include measurements available on a real quadruped robot such as in our case Unitree A1, including base linear and angular velocities obtained from the onboard IMU and foot–ground contact information. These measurements enable the policy to reason about balance, velocity tracking, lateral drift, and gait phase, which are critical for stable and efficient locomotion. When using a CPG-based controller, internal CPG states (amplitudes, phases, and their derivatives) are also included to enable phase-aware gait modulation, and the previous action is provided to supply short-term temporal context and promote smoother control commands. Observations returned by the environment consist of raw physical quantities and are normalized externally using the `VecNormalize` wrapper, which maintains running mean and variance statistics for each observation dimension. Each observation component is transformed to zero mean and unit

variance and clipped to a fixed range before being passed to the policy network. This normalization ensures that inputs with different physical units and numerical scales contribute comparably to learning, thereby improving training stability and sample efficiency.

To examine how much the locomotor performance depends on the richness of the observation space, and also to identify which information is sufficient to reach reasonable performance. We consider three different observation spaces with varying amounts of proprioceptive sensing. If using CPG-RL, the CPG states is always in the observation space.

Full observation space: The full observation space consists of body states (orientation, base linear velocity, and base angular velocity), joint states (joint positions and velocities), and binary foot–ground contact information. In addition, the internal CPG states $(r, \dot{r}, \theta, \dot{\theta})$ and the previous action selected by the policy network are concatenated with the proprioceptive measurements.

Medium observation space: The medium observation space differs from the full observation space by excluding joint states and the previous action. In this configuration, the policy relies primarily on the CPG states to infer the locomotion phase and generate control commands, rather than directly observing joint-level feedback. This design leverages the structure provided by CPG-based control and results in a lower-dimensional observation space. Despite the reduced state dimensionality, comparable performance is achieved in practice, and in some cases improved robustness to observation noise is observed compared to the full observation space.

Minimal observation space: The minimal observation space consists solely of binary foot–ground contact information and the internal CPG states. Under this configuration, and without substantial modification of the reward function, the learned policy fails to converge to a satisfactory locomotion behavior. Even after one million training time steps, both the achieved reward and the episode length remain far from optimal. We attribute this behavior to the insufficient state information provided by the minimal observation space, which limits the policy's ability to infer the robot's body dynamics and stability solely from CPG and contact signals. In the PyBullet–Gym simulation setting, this observation space is therefore too restrictive to support reliable learning of a high-quality control policy.

2) Action space: The policy network outputs actions in the normalized range $[-1, 1]$ to provide a consistent and well-conditioned interface between the neural network and the physical control commands. In general, different actions may correspond to control quantities with vastly different physical scales, such as joint angles, torques, or Cartesian displacements. Directly predicting these values would require the network to learn scale-dependent mappings, which can lead to poor numerical conditioning and unstable optimization. By constraining the policy output to a fixed, bounded range, the learning problem becomes scale-invariant and more stable. This choice also aligns with common activation functions such as $\tanh$, which naturally produce outputs in $[-1, 1]$ and help prevent output saturation and excessive gradients. The normalized actions are subsequently scaled to physically meaningful position limits through Scale helper(PD, Cartesian, CPG), ensuring safe exploration and improved training stability.

Cartesian PD control: In the Cartesian PD control mode, the policy network outputs normalized actions in the range $[-1, 1]$, which are first scaled to bounded Cartesian displacements of each foot relative to a nominal standing configuration. Apart from default value the scale up to $[0.2, 0.10, 0.16]$ is also tested for more natural gait. A Cartesian PD controller is then applied at the foot level, where position and velocity errors are converted into desired Cartesian forces using fixed gains. The corresponding joint torques are obtained by mapping these forces through the transpose of the leg Jacobian, resulting in an operational-space control law. This action formulation decouples high-level decision making from low-level actuation, improves learning stability, and ensures task and physically consistent joint torques during training.

CPG control: In the CPG control mode, the policy network outputs normalized actions also in the range $[-1, 1]$ that modulate the internal parameters of a Central Pattern Generator rather than directly commanding joint torques or positions. The action vector is split into frequency and amplitude modulation terms, which are scaled to physically meaningful ranges, using scale helper, and applied to the CPG to adjust gait timing and stride characteristics. The CPG is then integrated forward in time to generate smooth, rhythmic foot trajectories in Cartesian space. These desired foot positions are converted into joint-space targets via inverse kinematics, and Joint PD control is used to track the resulting trajectories and generate motor torques. This hierarchical control structure embeds a strong inductive bias toward periodic locomotion, reduces the dimensionality of the action space, and improves learning stability by decoupling high-level gait modulation from low-level actuator control.

3) Reward function: Two different reward function are used for velocity tracking and task-specific case.

Velocity tracking:

$$R_{velo} = r_{v_x} + r_{v_y} + r_{\omega_z} - 0.1\,|\phi| - 0.1\,|\theta| - 0.01\,E - 10\mathbb{1}_{\text{fallen}}, \tag{10}$$

The velocity-tracking reward function is designed to follow a desired velocity command (v_x, v_y, ω_z) while maintaining stability, energy efficiency, and robot base not touching the ground. The primary reward terms encourage tracking of the desired forward velocity, lateral velocity, and yaw rate through linearly scaled error terms, each clipped to a bounded range to prevent excessive reward magnitudes and ensure stable learning. Since it can both track x direction velocity and y direction velocity, our controller is omnidirectional. Body stability is promoted by linear penalties on roll and pitch angles, which discourage excessive body tilting while allowing unrestricted yaw motion. Energy efficiency is encouraged by penalizing the mechanical work of the actuators, computed as the time-integrated absolute inner product of joint torques and joint velocities and scaled by a constant factor to balance

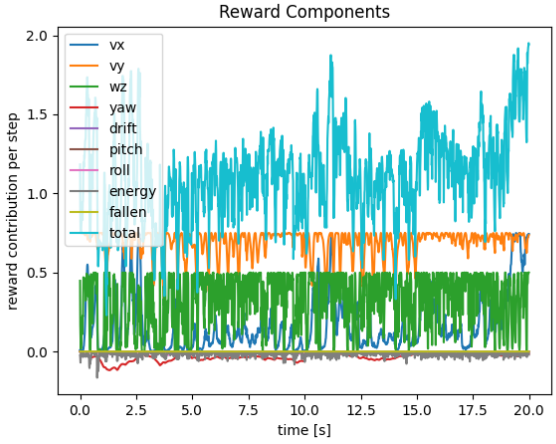


Fig. 1: Quantative Analysis of Reward Component (Slope)

its influence relative to the tracking terms. A large binary penalty is applied when the robot is detected as fallen, strongly discouraging unstable behaviors. All reward and penalty components could be either linear or exponential functions of their respective errors, as long as they are consistent. The overall reward is thus a weighted sum of normalized, bounded terms, providing interpretable contributions and stable gradients for reinforcement learning.

Although the reward function is constructed to maintain a clear physical interpretation and strong relevance to quadruped locomotion control, direct tuning of individual reward weights based solely on episode-level returns or raw proprioceptive sensor measurements remains challenging. In practice, such aggregate metrics provide limited insight into the relative influence of individual reward terms during training.

To better understand the learning dynamics, the total reward is decomposed into its constituent components, and their respective contributions are analyzed throughout training, as shown in Fig. 1. Large disparities among reward components, particularly during the early and intermediate stages of training, indicate that the learned policy may disproportionately optimize a subset of dominant terms. Excessive weighting of the linear velocity tracking term, for instance, biases the policy toward minimizing tracking error, whereas overly strong penalties on energy consumption favor low-work motion patterns that may limit locomotion performance.

Effective policy learning therefore requires a careful balancing of reward components to prevent any single term from dominating the optimization process. Such balancing enables the policy to achieve a more consistent trade-off among velocity tracking accuracy, stability, and energy efficiency, leading to improved overall locomotion behavior.

$$Reward = \max(Reward, 0) \tag{11}$$

Both the original and a non-negative reward formulation are evaluated, where the latter clips negative rewards to zero. This design aims to mitigate the influence of large penalty terms on episode termination and to promote stable policy learning during the early stages of training. In addition, the non-negative formulation effectively provides a warm-started optimization landscape, enabling the policy to explore meaningful gradients and progressively refine toward more effective locomotion behaviors. When the relative weights of reward components are properly balanced, the use of a non-negative reward formulation becomes optional.

Slope Locomotion Task:

$$\begin{aligned} R_{slope} = r_{v_x} + 0.01p[0] - 0.1\,|\phi| - 0.1\,|\theta| \\ - 0.001\,E - 10\mathbb{1}_{\text{fallen}}, \end{aligned} \tag{12}$$

The task-specific reward for slope locomotion follows a structure similar to the velocity-tracking formulation but with simplified objectives. Only the forward velocity component is retained to encourage increased uphill speed, which is effective for non-extreme slopes, together with a forward position term (p[0]) along the x direction to promote sustained progress. In addition, the energy-related penalty is assigned a lower weight, as slope climbing inherently requires higher mechanical work and excessive energy penalization tends to reduce episode length and the likelihood of successful ascent.

4) Deep reinforcement learning: We train the locomotion policy using *Proximal Policy Optimization (PPO)* as implemented in Stable-Baselines3, while also considering *Soft Actor–Critic (SAC)* for comparison. PPO is an on-policy policy-gradient method that employs a clipped surrogate objective to constrain policy updates, which is particularly important for maintaining stable learning in quadruped locomotion tasks where overly aggressive updates can quickly lead to unstable behaviors or falls. In contrast, SAC is an off-policy, entropy-regularized algorithm that is generally more sample efficient but was found to be more sensitive to hyperparameter tuning and reward shaping in our environment.

The policy and value networks are parameterized by a shared multilayer perceptron with two hidden layers of 256 units each, providing sufficient representational capacity to model the nonlinear mapping from high-dimensional proprioceptive observations to continuous actions. Key PPO hyperparameters include a discount factor $\gamma = 0.99$, a rollout length of $n_{\text{steps}} = 4096/N_{\text{env}}$, generalized advantage estimation with $\lambda = 0.95$, a learning rate of 1×10^{-4}, a clipping range of 0.2, a batch size of 128, and 10 optimization epochs per update. Gradient clipping and appropriate weighting of the value function loss were used to improve numerical stability. Observation normalization using `VecNormalize` was critical for handling heterogeneous observation scales and significantly improved convergence speed and training stability. Overall, PPO produced smooth and robust locomotion policies and was well suited to our control task despite lower sample efficiency compared to SAC.

5) Robustness: The CPG-RL framework demonstrates a high degree of robustness and strong transferability across both simulation environments and real hardware. This robustness primarily stems from the structured integration of CPG into the reinforcement learning pipeline, which significantly constrains the policy space toward smooth, rhythmic, and physically meaningful locomotion behaviors.

Sim-to-Sim Transfer Robustness: The approach exhibits strong sim-to-sim transfer performance. Unlike joint-space reinforcement learning controllers, which often overfit simulator-specific contact dynamics and numerical artifacts, CPG-RL relies on low-frequency periodic dynamics generated by oscillators. These dynamics are inherently more invariant to changes in physics engines, integration schemes, and solver parameters. As a result, policies trained with CPG-RL maintain stable gait patterns and command tracking accuracy even under substantial discrepancies between simulators.

Nevertheless, sim-to-sim transfer may still encounter difficulties related to differences in contact modeling, such as friction approximations and ground penetration handling. Since foot contact information plays a central role in our policy, inconsistencies in contact detection or timing can degrade performance.

Sim-to-Real Transfer Robustness: The CPG-RL framework shows excellent sim-to-real transfer performance, as demostate in [1]. By deployment on the Unitree A1 quadruped without the need for extensive domain randomization or observation noise injection. The policy remains robust to significant disturbances, including uneven terrain and dynamically added payloads up to 115% of the robot's nominal mass. This robustness can be attributed to two main factors: first, the action space operates at the level of CPG parameter modulation rather than direct torque or joint commands, reducing sensitivity to unmodeled actuator dynamics; and second the inclusion of CPG internal states in the observation space provides a noise-free phase reference that stabilizes coordination of limbs.

Despite these strengths, several challenges may arise when transferring to real hardware. These include actuator saturation and thermal constraints not fully captured in simulation, control latency and bandwidth limitations, and discrepancies in foot-ground interaction due to compliance, wear, or sensor inaccuracies.

Influence of Noiseness on Hardware: Within the full observation space, the internal CPG states and the previous action are the most reliable signals, as they are internally generated and free of sensor noise. Joint position measurements obtained from encoders also exhibit high fidelity and negligible noise. In contrast, IMU-based estimates of base orientation and angular velocity are affected by sensor noise and estimation uncertainty, which can be modeled in simulation using additive stochastic perturbations. Base linear velocity estimates and binary foot–ground contact signals are less reliable, due to a combination of estimation errors, contact uncertainty, and communication latency. Consequently, robust sim-to-real transfer necessitates extensive noise injection and domain randomization, particularly for velocity- and contact-related observations, to account for sensing and estimation discrepancies between simulation and hardware.

From a principled perspective, goal-directed locomotion requires explicit task-level command information in the observation space to define the control objective. In velocity-tracking locomotion, this objective is naturally represented by the desired linear and angular velocities. When these command signals are combined with proprioceptive feedback and CPG-related information, the policy can generate coordinated behaviors that are consistent with the commanded motion.

From an engineering perspective, reinforcement learning encodes these heterogeneous observations by concatenation, enabling joint representation of task objectives and system states. If the desired velocity remains fixed, its explicit inclusion in the observation is not strictly required, and the learned policy corresponds to a specialization to a constant goal. However, training a controller capable of tracking arbitrary or time-varying velocity commands necessitates explicitly including the desired velocity in the observation, together with an appropriately designed tracking reward and normalization scheme.

Training Environment Setup: To improve training stability and policy robustness, several modifications are applied to the simulation environment. These include variations in terrain slope, randomization of ground properties, and injection of observation noise to reflect sensing uncertainty. Such modifications expose the policy to a broader range of environmental conditions during training and reduce overfitting to a single nominal setup.

Empirically, these environment variations lead to more stable training dynamics and improved robustness at test time, particularly under unseen slopes and perturbations. By preventing the policy from exploiting simulator-specific regularities, the learned controller exhibits more consistent performance across different operating conditions.

III. Results

A. Part 1: Central Pattern Generators

1) CPG States: In Fig. 2 we can see the CPG states for the trot gait, and in Fig. 11 to Fig. 13 in supplementary material we can see the CPG states for walk, pace and bound gait. The amplitude r smoothly converges to the desired limit-cycle value $\sqrt{\mu}$, while its time derivative $\dot{r}$ converges to zero, indicating stable oscillator dynamics across all legs.

For visualization purposes, the phase θ is wrapped to the interval $[0, 2\pi)$. The phase θ increases approximately linearly over time; however, when the system switches between the swing and stance phases, the slope of θ changes accordingly due to the change in the natural frequency ω.

The phase derivative $\dot{\theta}$ is computed from the unwrapped phase to avoid artificial discontinuities introduced by phase wrapping.

2) Comparing the desired foot position vs actual foot position: Owing to the similarity between the joint-space and Cartesian-space PD controllers, both were tuned following the same principles. The Cartesian-space PD controller is used as an example to illustrate the tuning procedure. With the default Cartesian gains, the tracking performance degraded compared to the baseline case, as the controller followed the reference trajectory slowly in both joint and Cartesian spaces, resulting in large tracking errors. To improve responsiveness, the Cartesian proportional gain $K_{p,\text{cartesian}}$ was increased by a factor of ten and the derivative gain $K_{d,\text{cartesian}}$ by $\sqrt{10}$,

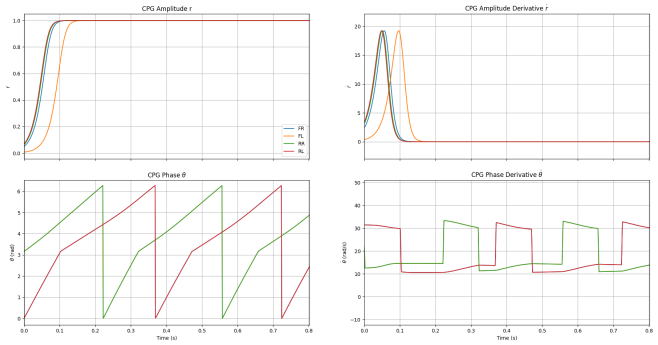


Fig. 2: CPG states for the trot gait

thereby increasing stiffness while preserving the damping ratio. This adjustment significantly reduced the response time. However, due to the non-penetration constraint of the ground in simulation, high gains in the x and z directions caused large overshoots during foot–ground contact, and $K_{p,\text{cartesian}}$ in these directions was therefore reduced. In contrast, the gain in the y direction remained insufficient and was further increased to achieve satisfactory tracking performance.

A combined joint PD and Cartesian PD controller is generally superior to a standalone implementation because it leverages the strengths of both control spaces. The Cartesian PD controller enforces accurate task-space behavior, such as end-effector or foot positioning and compliance, while the joint PD controller provides robust low-level stabilization and disturbance rejection at the actuator level. This hierarchical structure improves transient response, robustness to modeling errors, and contact stability, leading to more reliable and consistent task execution than using either joint-space or Cartesian-space control alone.

The final PD gains are given by

$$\begin{aligned} \mathbf{K}_{p,\text{cartesian}} &= \text{diag}(3000,\ 10000,\ 3000), \\ \mathbf{K}_{d,\text{cartesian}} &= 10\sqrt{10}\,\mathbf{I}_3, \\ \mathbf{K}_{p,\text{joint}} &= \begin{bmatrix} 100 & 170 & 170 \end{bmatrix}, \\ \mathbf{K}_{d,\text{joint}} &= \begin{bmatrix} 10 & 2 & 2 \end{bmatrix}. \end{aligned} \tag{13}$$

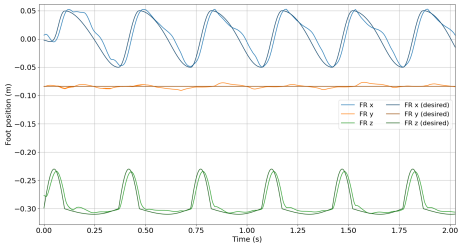
The desired foot position vs actual foot position using joint PD with/without Cartesian PD are shown in Fig. 3 and Fig. 4; the desired vs actual joint angles are shown in Fig. 5

The comparison of system responses before and after tuning the joint-level PD gains is shown in Fig. 14 in supplementary material.

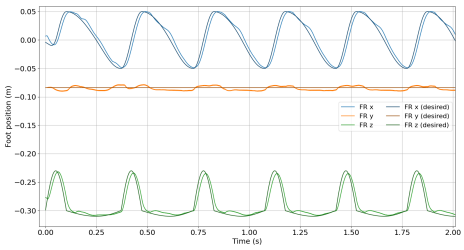
3) Hyperparameters that required tuning: In practice, stable locomotion with the open-loop CPG required tuning only a small subset of exposed hyperparameters. The parameters that consistently had the largest impact (and were therefore necessary to tune) were:

Swing/stance angular frequencies $(\omega_{\text{swing}}, \omega_{\text{stance}})$: these were the primary knobs to change cadence and (indirectly) the stance fraction. They also strongly affected impact severity at touchdown.

Step length d_{step}: this directly determined forward progress per stride. Large values increased peak speed but quickly degraded tracking and robustness.

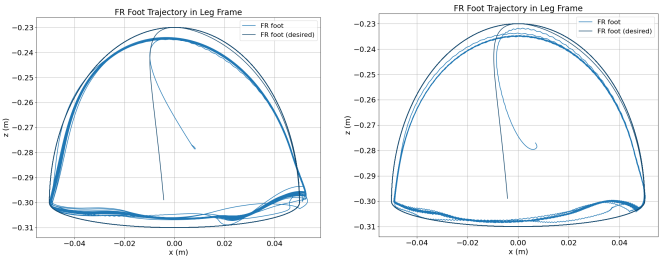


(a) Without Cartesian PD



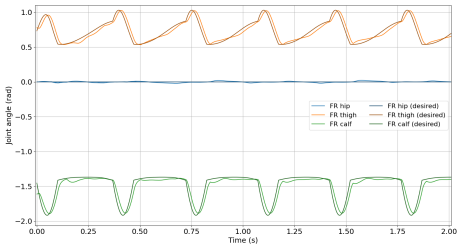
(b) With Cartesian PD

Fig. 3: Comparison in the world frame

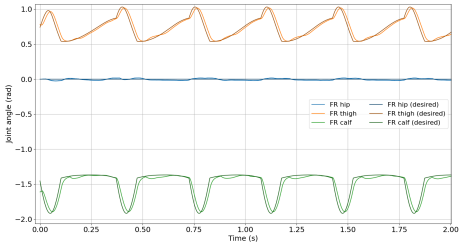


(a) Without Cartesian PD **(b)** With Cartesian PD

Fig. 4: Comparison in the leg frame



(a) Without Cartesian PD



(b) With Cartesian PD

Fig. 5: Comparison in joint space

Swing clearance g_c: The parameter g_c sets the peak *swing*

foot height in the trajectory mapping. Increasing g_c improves clearance and reduces toe scuffing, which is especially important at higher stride frequencies where the foot has less time to lift and tracking errors are larger. However, too large g_c typically increases swing effort (larger vertical motion and joint accelerations), which can raise energy consumption (higher CoT) and sometimes excite oscillations if the low-level PD gains are not retuned. In our tuning, g_c was increased only to the point where scuffing disappeared in steady-state locomotion.

Stance penetration g_p:The parameter g_p shapes the vertical foot motion during *stance* (i.e., when $\sin\theta < 0$ in the provided mapping). A larger magnitude of g_p tends to create a more pronounced downward "penetration" profile, which can improve ground contact consistency and make the gait less sensitive to small terrain/contact modeling errors. On the other hand, overly aggressive stance shaping can introduce unnecessary vertical ground reaction impulses, leading to harsher contacts, larger body bounce, and increased power spikes (thus higher CoT). In practice, we tuned g_p to obtain reliable contact without inducing excessive vertical excursions or impact-like behavior.

Nominal body height h: h sets the nominal foot height relative to the base. A larger h tends to make touchdown stiffer and more sensitive to tracking errors, while a smaller h improves compliance/robustness but can cause scuffing unless g_c is increased. Therefore, h was tuned together with (g_c, g_p) to balance clearance and contact smoothness.

4) Performance summary (speed, duty ratio, step timing, and CoT): Table II summarizes the lowest and highest steady-state forward body velocity we achieved for three gaits, together with the corresponding duty ratio D, stance/swing durations, and Cost of Transport (CoT). All values are computed over a steady-state window after convergence. Based on our observations, trot was consistently the most stable gait. It reached a clear steady state easily under both the low-speed and high-speed settings. Trot was also the most energy-efficient at high speed (CoT = 3.224).

A particularly interesting outcome is the behavior of pace. Unlike the other gaits, pace shows a low CoT at low speed (CoT = 0.866) but a much higher CoT at high speed (CoT = 8.839). A reasonable explanation is that, at low cadence and small step length, the quadruped can maintain balance more easily, requiring relatively small joint torques and producing limited impact transients; therefore, the energy remains low. When cadence and step length increase, however, the ipsilateral leg synchronization in pace tends to excite lateral body motions (roll/yaw), which requires additional stabilization effort. As a result, mechanical work increases significantly while forward speed does not increase proportionally (pace reaches only 0.502 m/s at its high-speed setting), leading to the observed sharp rise in CoT.

TABLE II: Lowest and highest steady-state forward body velocity for three gaits, with the corresponding metrics.

Metric		Trot	Pace	Walk
hyperparameter for lowest speed	$\omega_{\text{swing}}^{\min}$ $(2\pi\text{s}^{-1})$	3	3	2.5
	$\omega_{\text{stance}}^{\min}$ $(2\pi\text{s}^{-1})$	1.2	1.5	1
	$d_{step}^{\min}$ (m)	0.03	0.03	0.04
	$g_c^{\min}$ (m)	0.055	0.07	0.06
	$g_p^{\min}$ (m)	0.003	0.005	0.008
	$h_{\min}$ (m)	0.3	0.3	0.3
result when lowest speed	$v_{\min}$ (m/s)	**0.087**	**0.109**	**0.121**
	$D_{\min}$ (-)	0.62	0.55	0.34
	$T_{\text{stance}}^{\min}$ (s)	0.299	0.260	0.06
	$T_{\text{swing}}^{\min}$ (s)	0.188	0.216	0.098
	$\text{CoT}_{\min}$ (-)	12.619	0.866	27.505
hyperparameter for highest speed	$\omega_{\text{swing}}^{\max}$ $(2\pi\text{s}^{-1})$	9.2	8.5	10
	$\omega_{\text{stance}}^{\max}$ $(2\pi\text{s}^{-1})$	3.5	2.5	3
	$d_{step}^{\max}$ (m)	0.065	0.055	0.06
	$g_c^{\max}$ (m)	0.1	0.07	0.08
	$g_p^{\max}$ (m)	0.015	0.018	0.016
	$h_{\max}$ (m)	0.28	0.25	0.25
result when highest speed	$v_{\max}$ (m/s)	**0.983**	**0.502**	**0.639**
	$D_{\max}$ (-)	0.41	0.47	0.56
	$T_{\text{stance}}^{\max}$ (s)	0.056	0.075	0.05
	$T_{\text{swing}}^{\max}$ (s)	0.078	0.094	0.039
	$\text{CoT}_{\max}$ (-)	3.224	8.839	6.847

B. Extensions with feedback loops and descending control signals

The current implementation is largely *open-loop* at the gait-generator level: a CPG produces phase signals that are mapped to desired foot trajectories, and low-level PD controllers track those trajectories. This architecture can be substantially improved by adding feedback loops at multiple time scales, and by introducing descending commands that continuously modulate the gait generator and foot placement.

Outer-loop base velocity and heading feedback (task-level). A natural extension is to close a loop on the base planar motion using estimated body velocity and yaw rate. Let the descending command be a desired planar twist $\mathbf{v}_b^* = [v_x^*,\, v_y^*,\, \dot{\psi}^*]$. A PI/PID or LQR outer loop can regulate the error between measured and desired base motion and convert it into *gait modulation signals*, e.g., (i) adjust step length d_{step} and gait frequency via $(\omega_{\text{swing}}, \omega_{\text{stance}})$ to track v_x^*, (ii) introduce left–right asymmetry in step length or stance time to track $\dot{\psi}^*$ (turning), and (iii) bias lateral foot placement to regulate v_y and reduce lateral drift. This separates *what the robot should do* (track commanded speed/heading) from *how it does it* (CPG phase generation and tracking).

Descending gait modulation and gait switching. Beyond continuous speed commands, higher-level descending signals can select and smoothly transition between gaits. A practical method is to parameterize gait by a phase-offset matrix $\boldsymbol{\phi}_{\text{gait}}$ (trot/pace/bound) and interpolate between phase offsets over several strides, while simultaneously scheduling gait frequency and step length:

$$\boldsymbol{\phi}(t) = (1-\lambda)\,\boldsymbol{\phi}_A + \lambda\,\boldsymbol{\phi}_B, \qquad \lambda \in [0,1],$$

with λ ramped to avoid abrupt coordination changes. At the same time, a speed scheduler can map v_x^* to $(\omega_{\text{swing}}, \omega_{\text{stance}})$ and d_{step}, and adjust swing clearance g_c for terrain or speed.

Terrain adaptation and perception-driven modulation. If terrain height or slope is estimated (from proprioception or vision), descending signals can adapt swing height g_c, touchdown timing, and foothold positions to avoid obstacles and place feet on safe regions. Even without vision, online estimation of ground height per leg can update the swing apex and touchdown height to maintain clearance and reduce impacts.

C. Part 2: Deep Reinforcement Learning

TABLE III: Comparison of velocity controller performance

Metric	CPG	Cartesian
Cost of Transport (CoT)	0.594	0.648
Duty cycle / ratio	0.402	0.370
Time duration of one step (s)	0.291	0.093

1) Different 1-Direction Velocity Tracking: As shown in the Fig. 6, the RL-based velocity-tracking controller enables the quadruped to achieve accurate tracking of the desired velocity commands with low energy expenditure. Robust behavior is observed under both noise-free and noisy conditions. In all scenarios, the robot rapidly converges from rest to the commanded velocity and direction and maintains stable locomotion.

The average body velocity is computed over a $10\,\mathrm{s}$ interval for each trial. Since the robot starts from rest and gradually converges to the commanded velocity, the measured averages are slightly lower than the targets: $0.482\,\mathrm{m/s}$ for a command of $0.5\,\mathrm{m/s}$, $0.938\,\mathrm{m/s}$ for $1.0\,\mathrm{m/s}$, and $1.301\,\mathrm{m/s}$ for $1.5\,\mathrm{m/s}$. The discrepancies are mainly due to the initial acceleration phase. At convergence, the measured velocity lies within $\pm 0.05\,\mathrm{m/s}$ of the desired velocity.

As illustrated in Fig. 7, the training curves of the three velocity-tracking controllers show consistent growth in episode length and reward, reflecting stable reinforcement learning convergence. The resulting locomotion behaviors are consistent with the intended objectives and are further validated through video results.

Across experiments, convergence is typically observed within approximately 1,000,000 training timesteps, while excessive training may result in marginal performance degradation. In early stages with suboptimal reward weighting, sharp reductions in episode length were occasionally observed, leading to near-zero average rewards. This issue is no longer present under a well-balanced reward formulation and adequate training duration.

2) Omni-Direction Velocity Tracking Controller: Since the reward function explicitly incorporates velocity tracking terms for linear velocities in both the x and y directions as well as the angular velocity around the z axis, we further evaluate the controller under a desired velocity command of $[1,\ 1,\ 0]$. As observed in Fig. 8, although the convergence behavior under this omnidirectional command exhibits larger steady-state errors compared to single-direction tracking—partly attributable to the limited number of training timesteps—the learned policy is nevertheless able to simultaneously regulate motion along multiple directions. These results demonstrate

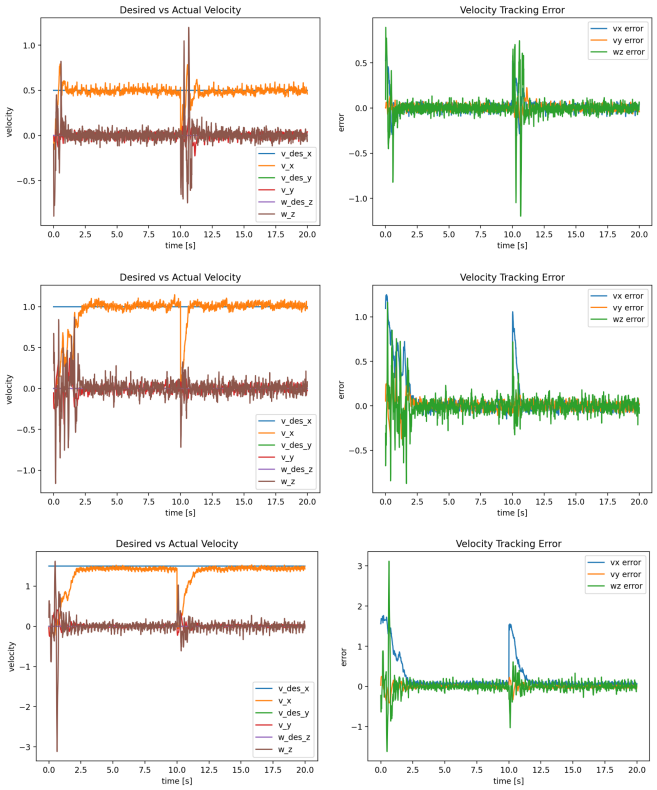


Fig. 6: Tracking of Different 1-Direction Velocity

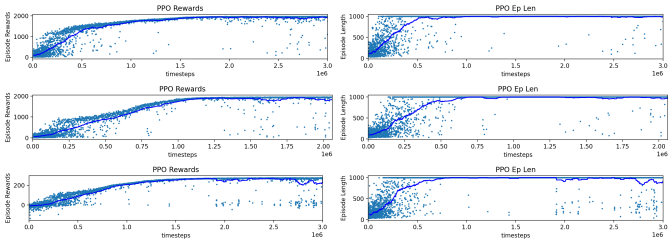


Fig. 7: Mean Episode Length and Reward of Velocity Tracking

that the proposed controller operates as an omnidirectional velocity-tracking controller rather than being restricted to unidirectional locomotion.

3) Action Space Design and Comparison: Under identical training conditions, we compare a Cartesian-space action space with Cartesian PD control and a CPG-based action space (CPG-RL) by examining both their training dynamics and the resulting locomotion behaviors.

From left to right, Fig. 9 presents the learned policies obtained using CPG-RL, Cartesian PD control with the original action scaling $[0.10, 0.05, 0.08]$, and Cartesian PD control with doubled scaling $[0.20, 0.10, 0.16]$, all of which successfully converge to stable locomotion policies. Notably, the CPG-based policy exhibits more structured and periodic gait patterns, attributable to the intrinsic oscillatory phase and amplitude generation of the CPG, whereas Cartesian PD control focuses on regulating foot positions relative to a nominal configuration with less explicit enforcement of

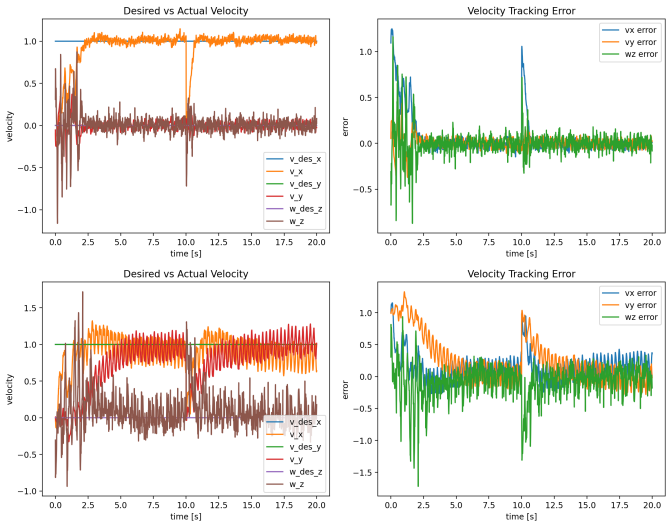


Fig. 8: Different Dimensions for Velocity Tracking

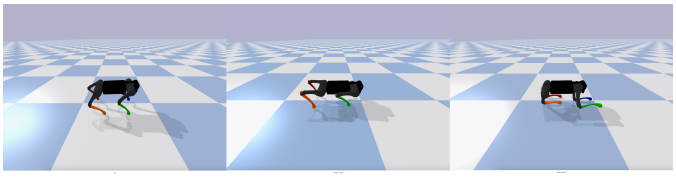


Fig. 9: Qualitative Comparison of Different Action Space

inter-leg coordination. Consequently, the gait is less natural, exhibiting a higher CoT, a reduced duty cycle, and a much shorter step duration (Table III).

4) Task Specific Performance (Slope): Training is conducted on a sloped terrain with a pitch angle of 0.2 for a total of 3,000,000 timesteps. The training curve is shown in Fig. 10

The learned policy is subsequently evaluated on steeper slopes. Only successful climbing trials, defined as episodes without failure, are included in the evaluation. Under these conditions, the robot is able to reliably traverse slopes with pitch angles up to 0.275, while successful ascents are occasionally observed at a pitch of 0.3.

The average forward velocities achieved during representative climbing trials are reported as follows: avg_speed $=$ $0.781\,\mathrm{m/s}$ at a pitch of 0.2, $0.658\,\mathrm{m/s}$ at a pitch of 0.25, and $0.698\,\mathrm{m/s}$ at a pitch of 0.275. These measurements are computed over trajectories that span the uphill phase from initial incline to crest and subsequent transition to flat terrain, while the downhill phase is not included in the evaluation.

IV. Discussion

A. Velocity Tracking Controller Design

From both training dynamics and task-level evaluation, the reward function is well aligned with the objectives of velocity-tracking control. The learning process exhibits consistent increases in episode length and cumulative reward, reflecting improvements in velocity tracking, posture stability, and energy efficiency, while remaining stable prior to convergence.

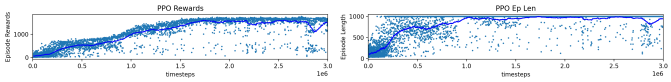


Fig. 10: Training Curve of Task-Specific Controller

Within the reward design, two formulations are considered for the velocity-tracking term. A linear tracking formulation promotes faster initial acceleration but tends to introduce oscillatory behavior at later stages, whereas an exponential formulation provides weaker early gradients but facilitates convergence near the desired velocity, analogous to steady-state error reduction in classical control.

To track arbitrary or time-varying velocity commands, the desired velocity must be explicitly included in the observation and synchronously provided at each control timestep, enabling the policy to learn a mapping from state and command to action. The reward function should evaluate tracking errors relative to the commanded velocity with appropriate normalization across different speed magnitudes, and apply consistent scaling to auxiliary terms such as energy penalties to ensure stable gradients. During training, velocity commands are randomized and periodically resampled, with a curriculum that progressively expands from low-speed to omnidirectional and wider velocity ranges.

B. Deployment of Slope Policy on Real Robot

In this project, training is performed only on a slope with a pitch of $0.2\,\mathrm{rad}$, while evaluation is conducted on steeper terrains. This setup differs from a standard sim-to-real training paradigm and does not fully capture real-world deployment requirements. For practical deployment or improved generalization, training should incorporate extensive domain randomization, including variations in slope angle, terrain geometry, and sensor noise.

Prior to hardware deployment, simulation-to-reality discrepancies must be explicitly characterized through noise modeling and system identification. In addition, hardware constraints such as actuator limits, control bandwidth, and safety margins must be enforced to ensure reliable and safe operation.

For maximum limit, the dominant performance limits are not imposed by the learning algorithm but by physical and system-level constraints, particularly in real-world deployment. These include actuator torque capacity, foot–ground friction, and controller execution frequency. As terrain difficulty increases, these constraints manifest concurrently, leading to actuator saturation, degraded velocity tracking, increased sensing uncertainty, and a higher likelihood of foot slip. While moderate slopes in the range of 0.3–$0.4\,\mathrm{rad}$ appear feasible in simulation, safe and sustained operation on hardware requires careful consideration of hardware limits, safety margins, and unmodeled dynamics.

V. Conclusion

This work demonstrates that integrating Central Pattern Generators with reinforcement learning produces structured, stable, and energy-efficient quadruped locomotion. Compared to Cartesian action spaces, the CPG-based approach yields

more natural gaits and improved robustness across velocity-tracking and slope-locomotion tasks.

VI. SUPPLEMENTARY MATERIAL

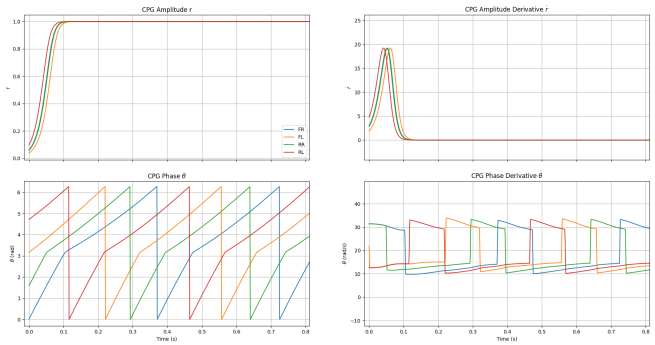


Fig. 11: CPG states for the walk gait

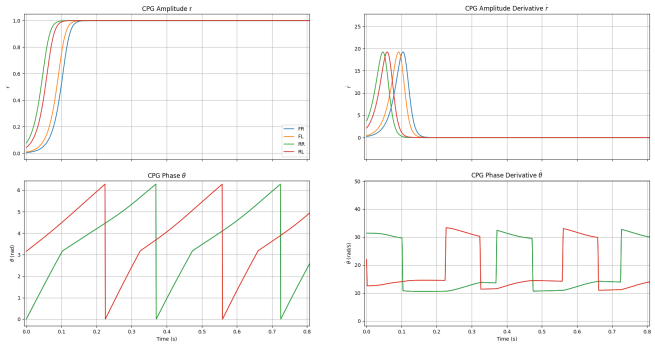


Fig. 12: CPG states for the pace gait

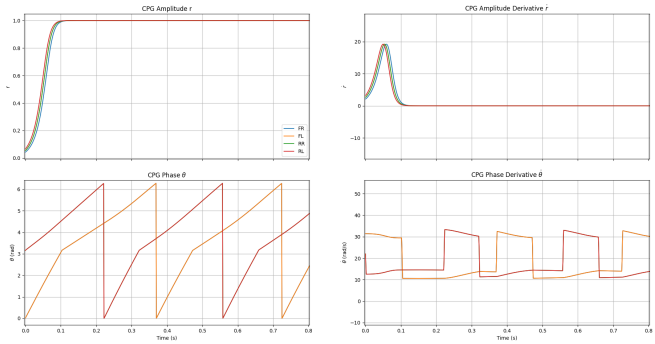
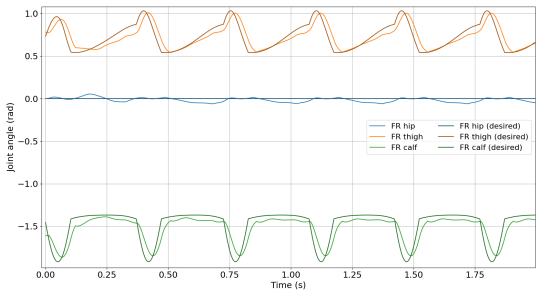


Fig. 13: CPG states for the bound gait

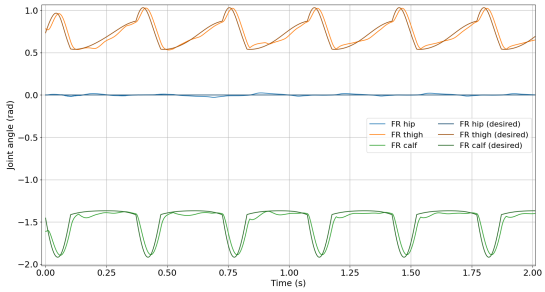
VII. GENERATIVE AI DECLARATION

A. *Which aspects of the project did you use generative AI for?*

We used generative AI tools primarily to clarify basic control concepts mentioned in the course materials (e.g. traditional control approaches referenced in the lectures), to assist in implementing non-core utility functions such as visualization scripts, and to refine the academic writing style of the report. In general, I use AI when learning new theoretical concepts,



(a) Default Gain



(b) Tuned Gain

Fig. 14: Comparison of joint PD control

implementing routine or tool-level code that does not involve deep reasoning, or proofreading written text. However, I deliberately avoid using AI for the core components related to robotics control and learning algorithms, which I completed independently.

B. *In which aspects was generative AI the most useful and least useful?*

AI was most useful for understanding completely new concepts at the beginning of the project. For example, "Can you explain what different quadruped gaits correspond to in intuitive examples?", the response provided clear explanations and examples that helped me build an intuitive understanding before diving into equations and implementation. AI also suggested related academic papers for further reading, which accelerated my conceptual learning.

C. *Has the use of AI made the project smoother to complete?*

Yes. By automating repetitive tasks and assisting with non-technical components such as visualization and documentation, AI significantly reduced the time spent on peripheral work, allowing me to focus more on the analytical and experimental aspects of the project.

D. *Has the use of AI impacted your ability to learn and understand the concepts behind this project?*

Yes, positively. AI helped me quickly grasp new concepts by offering clear explanations and guiding me toward key references. This allowed me to form an initial understanding

faster and to deepen it later through experimentation and reading original papers.

REFERENCES

[1] G. Bellegarda and A. J. Ijspeert, "CPG-RL: Learning Central Pattern Generators for Quadruped Locomotion," *IEEE Robotics and Automation Letters*, vol. 7, no. 4, pp. 11149–11156, Oct. 2022, doi: 10.1109/LRA.2022.3218167.