Quadruped Locomotion with CPGs and Deep Reinforcement Learning

Tom Herrmann - 355973 Alexandros Dellios - 355873 Salah Slaoui Hasnaoui - 342907

I. Introduction

We present in this report a quadruped locomotion study using a Central Pattern Generator (CPG) and reinforcement learning. We implement and evaluate multiple gaits, and analyze key design choices (observations, actions, and rewards). We finally assess performance through both learning curves and locomotion metrics, with an additional focus on robustness in more challenging conditions.

II. Methodology

A. CPG Background

1) CPG Formulation: We model each leg $i \in \{1, \ldots, 4\}$ with a Hopf oscillator in polar form, whose state is the amplitude r_i and phase θ_i.The Hopf dynamics generate a stable limit cycle used as a rhythmic pattern generator.

a) Amplitude dynamics: The amplitude converges to a desired steady-state magnitude set by μ:

$$\dot{r}_i = \alpha(\mu - r_i^2)\, r_i, \tag{1}$$

where $\mu > 0$ defines the target limit-cycle amplitude ($r_i \to \sqrt{\mu}$) and $\alpha > 0$ is the convergence rate. Larger α yields faster convergence but can increase stiffness in the numerical integration.

b) Phase dynamics and coupling: The phase evolves according to a natural frequency and an inter-oscillator coupling term:

$$\dot{\theta}_i = \omega(\theta_i) + \sum_{j \neq i} r_j\, k\, \sin(\theta_j - \theta_i - \Phi_{ij}), \tag{2}$$

where $\omega(\theta_i)$ is selected as ω_{swing} or ω_{stance} depending on the phase interval, and k is the coupling strength. The matrix $\Phi \in \mathbb{R}^{4\times 4}$ encodes the desired phase-lags for a given gait; it imposes the relative timing between legs by driving the phase differences toward Φ_{ij}. The scaling by r_j makes coupling stronger once oscillators reach their limit-cycle amplitude.

Now that we have established CPG equation we can the dive into the quantities ω_{swing} and ω_{stance} which are phase angular velocities (rad/s) that set how fast θ_i evolves in time, hence controlling the gait timing, cycle period, and duty factor. From the equation (2), we can define

$$\omega(\theta_i) = \begin{cases} \omega_{\text{swing}}, & \theta_i \in [0, \pi], \\ \omega_{\text{stance}}, & \theta_i \in (\pi, 2\pi), \end{cases} \tag{3}$$

which is consistent with the swing/stance definition used in the foot height mapping (swing when $\sin(\theta_i) > 0$).

Since swing and stance each correspond to half a cycle (π radians), the corresponding durations for trot for example are

$$T_{\text{swing}} = \frac{\pi}{\omega_{\text{swing}}}, \qquad T_{\text{stance}} = \frac{\pi}{\omega_{\text{stance}}}, \qquad T = T_{\text{swing}} + T_{\text{stance}}, \tag{4}$$

and the duty factor is $D = T_{\text{stance}}/T$. Therefore, tuning ω_{swing} and ω_{stance} adjusts the stance and swing ratio and the step frequency, which impacts both stability and forward speed.

2) Gait Design: For this project, we implemented 4 essential gaits : Trot, walk, pace and bound.

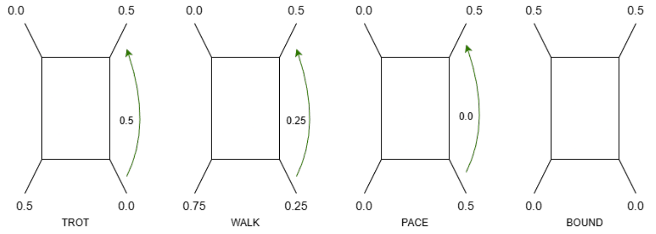


Fig. 1: Gaits Implemented

For each gait that you see in the figure above, we analysed the different phases between each leg. The labels 0.0, 0.25, 0.5, 0.75 are normalized phases. We convert them to oscillator phases via

$$\theta = 2\pi \cdot \text{phase},$$

so 0.5 means half a cycle $\Rightarrow \theta = \pi$.

We then assign a target phase θ_i to each leg (we chose FR leg as reference leg) with $\theta = 0$. The pairwise phase-lags are computed as :

$$\Phi_{ij} = \theta_j - \theta_i,$$

and wrapped modulo 2π (normalized to $[-\pi, \pi]$). Finally, we enforce the standard coupled-oscillator structure:

$$\Phi_{ii} = 0 \quad \text{(diagonal, same leg)}, \qquad \Phi_{ij} = -\Phi_{ji}$$

As we can see here the coupling matrix used as a reference for the different gaits :

$$\begin{bmatrix} 0 & \Phi_{\text{FL,FR}} & \Phi_{\text{FL,BL}} & \Phi_{\text{FL,BR}} \\ \Phi_{\text{FR,FL}} & 0 & \Phi_{\text{FR,BL}} & \Phi_{\text{FR,BR}} \\ \Phi_{\text{BL,FL}} & \Phi_{\text{BL,FR}} & 0 & \Phi_{\text{BL,BR}} \\ \Phi_{\text{BR,FL}} & \Phi_{\text{BR,FR}} & \Phi_{\text{BR,BL}} & 0 \end{bmatrix}$$

And then, for each gait, we can see the matrices found :

$$\Phi_{\text{trot}} = \begin{bmatrix} 0 & -\pi & -\pi & 0 \\ \pi & 0 & 0 & \pi \\ \pi & 0 & 0 & \pi \\ 0 & -\pi & -\pi & 0 \end{bmatrix} \qquad \Phi_{\text{walk}} = \begin{bmatrix} 0 & -\pi & -3\pi & -\pi \\ \pi & 0 & -\frac{\pi}{2} & \frac{\pi}{2} \\ \frac{3\pi}{2} & \frac{\pi}{2} & 0 & \pi \\ \frac{\pi}{2} & -\frac{\pi}{2} & -\pi & 0 \end{bmatrix}$$

$$\Phi_{\text{bound}} = \begin{bmatrix} 0 & 0 & -\pi & -\pi \\ 0 & 0 & -\pi & -\pi \\ \pi & \pi & 0 & 0 \\ \pi & \pi & 0 & 0 \end{bmatrix} \qquad \Phi_{\text{pace}} = \begin{bmatrix} 0 & -\pi & 0 & -\pi \\ \pi & 0 & \pi & 0 \\ 0 & -\pi & 0 & -\pi \\ \pi & 0 & \pi & 0 \end{bmatrix}$$

3) CPG to foot trajectory mapping: For each leg i, the CPG state (r_i, θ_i) is mapped to a desired sagittal foot trajectory $(x_{\text{foot},i}, z_{\text{foot},i})$ as

$$x_{\text{foot},i} = -d_{step}\, r_i \cos(\theta_i), \tag{5}$$

and

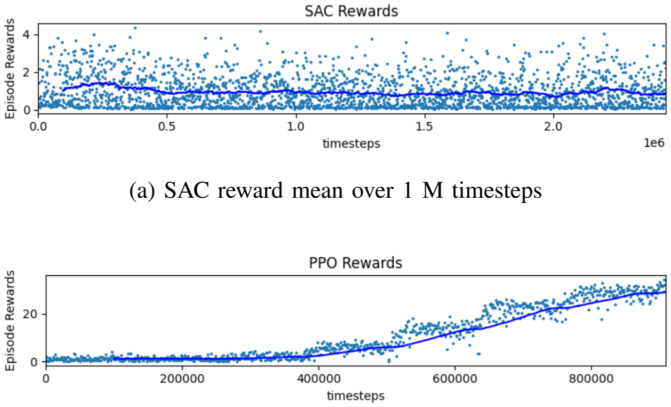
$$z_{\text{foot},i} = \begin{cases} -h + g_c \sin(\theta_i), & \sin(\theta_i) > 0 \quad \text{(swing)}, \\ -h + g_p \sin(\theta_i), & \sin(\theta_i) \leq 0 \quad \text{(stance)}, \end{cases} \tag{6}$$

where d_{step} is the step-length scaling, we need it because (r_i, θ_i) unitless rhythmic phase signal, with d_{step} we can convert that normalized amplitude into an actual step length (meters). Additionally it helps control the forward speed, and the stability (a step too large might cause slipping and too small might cause no progression). Then, h is the robot height, g_c is the ground clearance, and g_p is the ground penetration.

B. Training details

In order to set up our DRL pipeline we have some choices to make. We will discuss in section II-B1 the RL algorithm and hyperparameter we used. In section II-B2 we will discuss our observation space setup for our tasks. Then the action space in section II-B3 and our tuning process, and finally demonstrating the reward functions used in section II-B4.

1) DRL Framework: First, we need to explore the choice of learning algorithms to use. The more common pick is Proximal Policy Optimization, as it is more documented in the course material (see [1]) and is easier to use due to its robustness to hyperparameter tuning. As the provided PPO algorithm learns best on the CPU, we wanted to explore alternatives that use the GPU. Indeed the provided Soft Actor Critic algorithm can run on the CUDA cores of our RTX 3060 TI GPU. Moreover, in theory SAC provides a more efficient and better performing approach to reinforcement learning. The hardware change reduced the training time for 1 million timesteps to about 18 minutes using 128 environments. Despite this training efficiency, SAC demonstrated in practice limitations for finding locomotion policies. Even after extensive hyperparameter tuning, it could not get past 5% of PPO's average reward. This can be seen in the graph where the reward mean plateaued at a very low level. We hypothesized that early exploration plagued the replay buffer with catastrophic states from which the Q-function could not recover. This would result into local optima that do not provide satisfying solutions. SAC's ability to learn legged locomotion could have been explored further but this is not the aim of this project. Thus we opted to use the more reliable PPO algorithm with 128 parallel environments.



(a) SAC reward mean over 1 M timesteps

(b) PPO reward mean over 1M timesteps

Fig. 2: Comparison between SAC and PPO

Next, we study the effects of hyperparameter tuning on the performance of the PPO algorithm. To do this we conducted an ablation study to understand each hyper-parameter's effect on the reward curve, See Appendix A, Fig. 16

- **Clip range :** This value represents how much the policy can deviate in one update step. When increased from 0.2 to 0.4, we see a quicker initial learning, see 16b. This is due to the policy applying larger updates. Too high of an increase leads to a higher risk of mean reward collapse. Thus, a clip range of 0.2 is used as it converges fast and assures stability.
- **Learning Rate :** This parameter governs how large of a step to take in the direction of the gradient. Large values create a quicker initial learning but with a risk of forgetting good behaviors. In 16b a learning rate of 1×10^{-3} is tested. It results in quicker initial learning but a smaller reward plateau. We use the default value of 1×10^{-4}.
- **Entropy :** This coefficient dictates the stochastic behavior of the policy. A higher value can help escape local optima. A value too large will prevent the policy from committing to one approach. As seen in 16d, a value of 0.1 is too high as the policy plateaus at a lower value than the default. We will use of a value of 0.01.
- **Batch size :** This term corresponds to the amount of steps used in each gradient update. Larger values create more stable learning but uses more memory. A batch size of 512 was tested. And we observe that the policy starts to learn much later around 8 million timesteps, see 16e. We keep a batch size value of 128.
- **GAE λ :** This value determines how much the agent cares about future rewards compared to immediate ones within nsteps. In our tests with a value of 0.8, which makes the agent more myopic, we don't see much of a difference, see 16f. We keep the initial value of 0.4.

- **n_epochs :** The algorithm will loop over the collected batch of data n_epochs times to perform updates. The larger the value the larger the sample-efficiency. A value too large might result in overfitting. In our testing with a value of 25 we did not see much of a difference, see 16g. We kept the initial value of 10.
- **Gamma :** Similarly to GAE λ, this value dictates how much the agent cares about future rewards vs. immediate ones but over the entire task horizon. A lower value will most probably be detrimental as the robot might take an action that will make it fall one step later. This is confirmed looking at 16h. We keep the initial value of 0.99.
- **Max Grad Norm :** This parameter limits the maximum norm of the gradient. Lowering this value prevents a bad batch from causing a destructive policy update. We tested with a value of 1 but did not see much of a change, see 16i. We kept the default 0.5.
- **Vf coef :** This parameters balances learning the policy (actor) with learning the value estimate (critic). Lowering this value can make the critic learn poorly resulting in a reward plateau. This is confirmed when looking at 16j. We choose to keep the value at 0.5.
- **Architecture :** We tested two different neural network architecture. First the default two layer [256][256] then the 3 layer [512][256][128] we can often find in papers such as [1]. We first compare the performance of the 3 layer vs 2 layer on flat terrain, see 20. Performance is similar reaching max reward means of 76.4 and 73.6 respectively. Nevertheless we see a bigger disparity of rewards between episodes for the 3 layered network. This behavior is aggravated when training on sloped terrain indifferently of retraining the policy from a checkpoint. Indeed the 3 layered architecture reaches a maximum reward mean of 694 compared to the 724 of the 2 layered. Thus we choose to keep the more reliable 2 hidden layered neural network architecture.

2) Observation space: We use a single observation space that combines proprioceptive measurements with the CPG internal states. Our goal is to provide the policy with information that is realistically available on a quadruped (contacts, joint state, base state) while also leveraging the CPG as a structured rhythmic prior. The CPG variables acts as a stabilizing source of information for the policy and can ease learning by explicitly exposing the gait phase and amplitude.

Thus, the observation is formed by concatenating: (1) the four foot contact booleans, (2) joint proprioception (motor angles and motor velocities), (3) the base orientation, (4) the base linear velocity and base angular velocity, and (5) the CPG states $r,\ \dot{r},\ \theta,\ \dot{\theta}$. The observation space is then : $\{c,\ q,\ \dot{q},\ \mathbf{q}_{\text{base}},\ \mathbf{v}_{\text{base}},\ \boldsymbol{\omega}_{\text{base}},\ r,\ \dot{r},\ \theta,\ \dot{\theta}\}$.

We bound each component with realistic limits (e.g., joint limits, velocity limits, $r \in [\mu_{\text{low}}, \mu_{\text{upp}}]$, $\theta \in [0, 2\pi]$) and add a small ϵ margin to avoid numerical issues at the boundaries.

The normalization of the observation space is important because the different measurements in the observation vector can have very different scales. For example, joint angles in radians, joint velocities in rad/s, quaternions are in $[-1, 1]$, base velocities in m/s and CPG phases in $[0, 2\pi]$, contact bits in $\{0, 1\}$). Normalization makes inputs comparable, keeps hidden-layer activations in a good range to avoid saturation, and improves sample efficiency.

This observation space was mainly inspired from [1] in order to guide our reasoning.

3) Action space: We set up the action space based on the selected motor control mode. If the mode is {PD,Torque, Cartesian_PD }, the policy outputs one command per actuator. The robot has 12 motors, so we use $action\_dim = 12$. If the mode is CPG, the policy controls a lower-dimensional set of CPG-related parameters (amplitude and frequency for each leg), so we use $action\_dim = 8$. The action is defined as a continuous bounded vector : $a \in [-1, 1]^{action\_dim}$ which provides a normalized action range that is later mapped/rescaled by the controller.

Having the policy output actions in $[-1, 1]$ is standard because action dimensions can have very different physical scales. Normalizing and bounding the output keeps the network in a consistent numeric range, improving gradient conditioning, exploration stability, and preventing extreme commands.

4) Reward function: We define the reward function for a forward locomotion :

TABLE I: Reward function terms (velocity tracking).

Name	**Formula**	**Weight**
Velocity reward	$\begin{cases} 0.02 & \text{if } v_x < 0.2 \\ 0.1 \cdot v_x & \text{if } 0.2 \le v_x \le 1.0 \\ 0.1 & \text{if } v_x > 1.0 \end{cases}$	0.1
Tracking reward	$0.2 \exp\Big(-\dfrac{1}{0.25}\,(v_x - v_x^{\text{des}})^2\Big)$	0.2
Yaw penalty	$-\,0.2\,\lvert\psi\rvert$	-0.2
Lateral drift penalty	$-\,0.01\,\lvert y\rvert$	-0.01
Energy penalty	$-\,0.01 \sum_t \big\lvert \boldsymbol{\tau}(t)^\top \dot{\mathbf{q}}(t) \big\rvert\,\Delta t$	-0.01
Orientation penalty	$-\,0.1\,\lVert \mathbf{q} - \mathbf{q}_0 \rVert_2$	-0.1
Total: $r = \max(r_{\text{vel}} + r_{\text{track}} + r_{\text{yaw}} + r_{\text{drift}} + r_{\text{energy}} + r_{\text{ori}}, 0)$		

This reward function was the default provided in the code structure. When using desired velocity tracking for part III-D, the term "Velocity reward" is swapped with the term "Tracking reward". The weight for this tracking reward was tuned to 1/0.2 instead of 1/0.05 for better tracking performance.

We define the reward function for a slope environment :

TABLE II: Reward function terms.

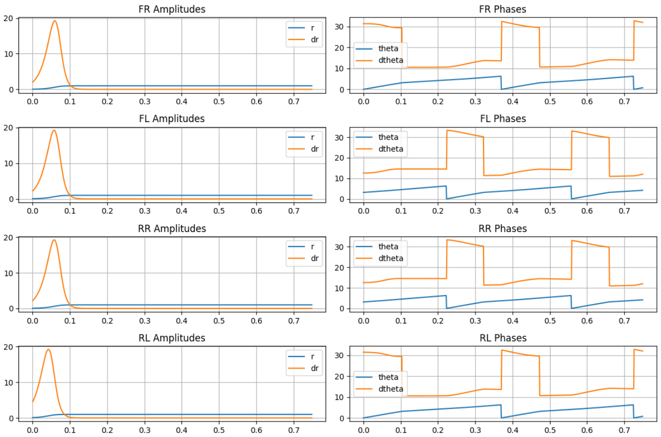
Name	Formula	Weight
Forward velocity reward	v_x	$w_{v_x} = 1$
Lateral velocity penalty	$-v_y^2$	$w_{v_y} = 0.002$
Torque penalty	$-\sum_{m=1}^{12} \tau_m$	$w_t = 0.0001$
Roll penalty	$-\phi_{\text{roll}}$	$w_{\text{roll}} = 0.003$
Yaw penalty	$-\psi_{\text{yaw}}$	$w_{\text{yaw}} = 0.002$
Total: $r = \max\big(w_{v_x} v_x - w_{v_y} v_y^2 - w_t \sum \tau - w_{\text{roll}}\phi - w_{\text{yaw}}\psi,\ 0\big)$		

For this last reward function, we took inspiration from [2]. The goal of it was to maximize smooth locomotion and encouraging the robot to follow surmount a slope by penalizing the torque usage, rotation and deviation from the axis.

III. Results

A. CPG Modeling

1) CPG States: Here is the **CPG States** graphs for Trot gait (other gaits in annex).



(a) TROT

Fig. 3: CPG state trajectories for the implemented gaits (trot, bound, pace, walk).

First, we see that in all the gaits, the amplitude of the oscillators quickly settle to $\sqrt{\mu}$, (in this case 1), then stays constant over time. The increase is explained by th big variation of $\dot{r}$ at initialization.

For the phase graphs, we see that a θ cycle lasts for 2π and then goes back to zero. We also detect a change in the slope of the graph. This corresponds to the two different values : ω_{swing} and ω_{stance}. The $\dot{\theta}$ value varies accordingly. Moreover, we see that the coupling of the different gaits can be directly visualized. For example, the Front Right leg is coupled with the Rear Left and the Front Left with the Rear Right.
Theses visualizations allows us to clearly identify if we implemented well our coupling matrices.

2) Desired foot position vs Actual Foot Position: Here is the plot of our desired foot position versus the real positions we obtained.

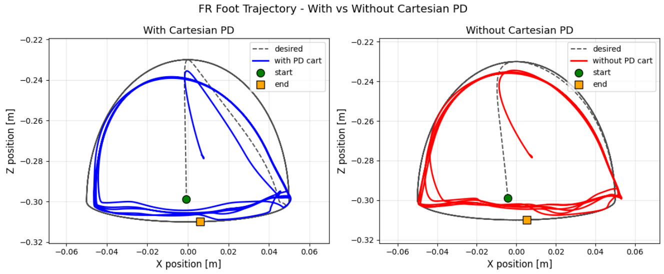


Fig. 4: TROT: $k_p = 150, k_d = 2, k_{p_{cart}} = 500, k_{d_{cart}} = 20$

Alongside these figures, adding Cartesian PD leads to a negligible improvement in joint-space tracking (RMS error reduced by 0.9%), but a clear improvement in end-effector tracking: the foot position RMS error drops from 2.96 cm to 2.69 cm (8.9%). This indicates that the Cartesian feedback mainly acts as a task-space correction t, reducing residual foot placement errors without substantially changing joint tracking performance. Visually, the Cartesian trajectory may appear less smooth due to small corrective actions, but the overall tracking accuracy in Cartesian space is improved.

See the appendix to view these graph for other gaits with the following gains :

WALK : $K_p = 300, K_d = 2, K_{p_{cart}} = 500, K_{d_{cart}} = 20$
PACE : $K_p = 150, K_d = 2, K_{p_{cart}} = 500, K_{d_{cart}} = 20$
BOUND : $K_p = 100, K_d = 3, K_{p_{cart}} = 500, K_{d_{cart}} = 20$

We tested several values for each gains to see the reaction on the tracking error. Below, you will see different results for the TROT gait of several values of K_p for $K_d = 2$ (in the appendix, you can take a look at this graph for other gaits). From 22, we see that a small k_p leads to smooth plots but more tracking error while a big k_p lead to less tracking error but more rough curves and small unwanted oscillations. For this gait, $K_p = 150$ was chosen as a good compromise.

3) Desired Joint Angles vs. Actual Joint Angles : Here is graphs comparing the desired and real joint angles for the FR leg during the TROT gait

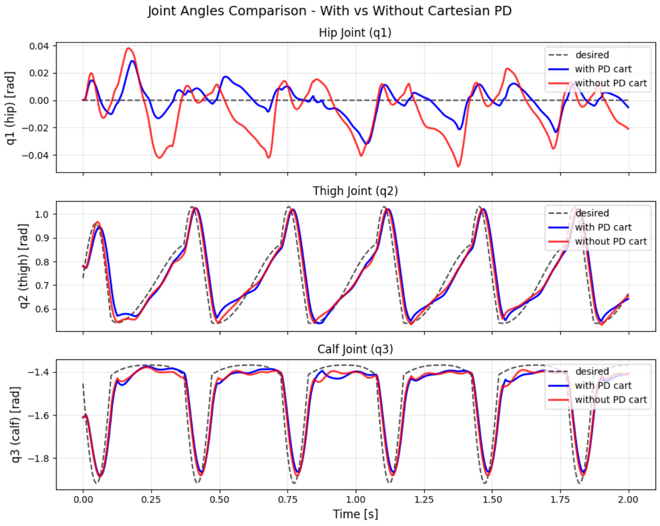


Fig. 5: TROT gait - $K_p = 150$

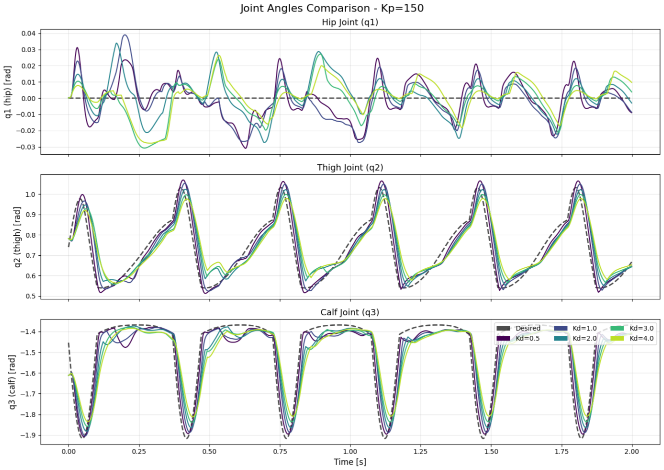


Fig. 6: TROT gait - Tests for different K_d values

Varying the K_d allowed us to see that a higher K_d increased the damping and reduced the overshoot but made the system more sensible to noise.

We chose $K_d = 2$ in all gaits, which was a good compromise, except for the BOUND gait where we took $K_d = 3$ for a more damped behavior as it used to fall over with other values.

4) Achieving high and low velocity: The main hyperparameters we tuned were the CPG timing and stride scaling parameters: ω_{swing}, ω_{stance}, and d_{step}. These parameters directly control cadence, stance and swing ratio, and stride length.

For each parameter set, we measure locomotion speed using the base linear velocity :

$$v_{\text{base}}(t) = \sqrt{v_x(t)^2 + v_y(t)^2} \tag{7}$$

The lowest and highest velocities achieved are

$$v_{\min} = 0.23\, m/s, \qquad v_{\max} = 1.53\, m/s. \tag{8}$$

In our implementation, we refer to the formulas provided by (4) and (5). The duty factor (stance time ratio) is

$$D = \frac{T_{\text{stance}}}{T_{\text{swing}} + T_{\text{stance}}} = \frac{\omega_{\text{swing}}}{\omega_{\text{swing}} + \omega_{\text{stance}}}.$$

The idea for us was to reduce the duty factor in order to increase the speed of our robot. We also wanted to increase the x position of each step.

For high speed, we tuned $w_{swing} = 10 * 2\pi$ and $w_{stance} = 4 * 2\pi$ and $d_{step} = 0.1\ m$. This was the maximum for our robot to remain stable. We obtained a duty factor of $D = 0.71$ and a cost of transport $CoT = 2.54$

For low speed, we tuned $w_{swing} = 2 * 2\pi =$ and $w_{stance} = 1 * 2\pi$ and $d_{step} = 0.03\ m$. We obtained a duty factor of $D = 0.67$ and $CoT = 0.95$

B. *Observation space Results*

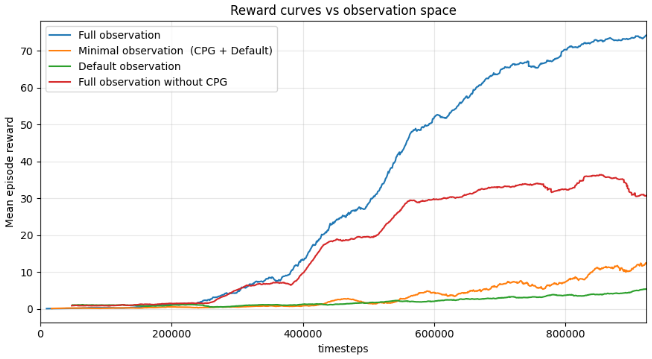


Fig. 7: Comparison between different observation spaces

- **Full observation:** This observation was the fastest learning and had the highest final performance 74.4. The curve is also the most stable after convergence, suggesting that the policy has sufficient state information to generate consistent locomotion. It contains $\{c,\ q,\ \dot{q},\ \mathbf{q}_{\text{base}},\ \mathbf{v}_{\text{base}},\ \boldsymbol{\omega}_{\text{base}},\ r,\ \dot{r},\ \theta,\ \dot{\theta}\}$ as inputs.
- **Full observation without CPG:** improves initially but stagnate to a much lower value 35.5. This indicates that removing CPG states removes an explicit phase/timing signal, making gait coordination harder even with rich proprioception. We trained with this observation space for up to 2 million timesteps to show the stagnation and importance of CPG states. This space contains $\{c,\ q,\ \dot{q},\ \mathbf{q}_{\text{base}},\ \mathbf{v}_{\text{base}},\ \boldsymbol{\omega}_{\text{base}}\}$ as inputs.
- **Minimal observation:** this observation space contains $\{q,\ \dot{q},\ \ r,\ \dot{r},\ \theta,\ \dot{\theta}\}$. It learns slowly and converges at a lower reward mean. This suggests that CPG states alone are not sufficient.
- **Default observation:** lowest performance and near-flat learning, implying that key measurements are missing for stabilization and effective credit assignment.

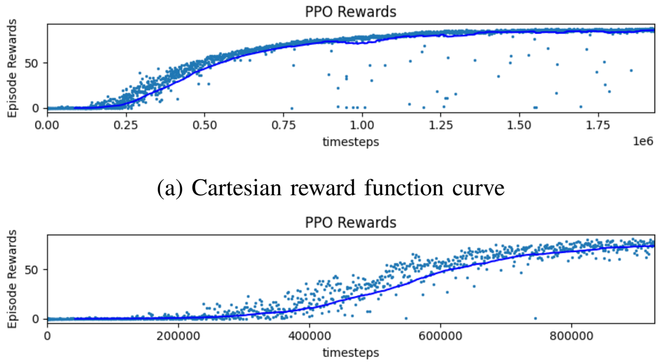
We can deduce that CPG states are not enough to provide a quantifiable result in a short period of time, the training needs a tremendous amount of learning. But coupled with other parameters (Base position, linear and angular velocity alongside foot contact booleans), we can achieve a high mean reward in a reasonable amount of time. That is why we proceed with a full observation space.

We add noise to the observation space in order to increase the robustness for sim to real transfers. On hardware, we have different noise characteristics:

- **CPG states** $\{r, \dot{r}, \theta, \dot{\theta}\}$**:** Noise-free (internal controller variables).
- **Foot contacts (bool):** Moderately noisy. Contact sensors can have false-negatives at touch-down, and differ by surface type. Often needs filtering.
- **Joint angles** q**:** Low noise. Encoders are very accurate. The only problems resides in small offset after calibration or backlash.
- **Joint velocities** $\dot{q}$**:** Medium noise. Often computed by differentiating encoder positions thus amplifying noise. Filtration adds a delay.
- **Base orientation (IMU):** roll/pitch are fairly reliable. Yaw is noisier and drifts over time.
- **Base angular velocity** ω **(gyroscope) :** Medium noise. Bias and temperature drift can cause orientation drift.
- **Base linear velocity** $\mathbf{v}$**:** typically the noisiest. Depends on state estimation (slip can degrade it).

C. Action Space Result:

We have tested 2 different action spaces : Cartesian and CPG action space (see the video titled RL_CARTESIAN.mp4):



(a) Cartesian reward function curve

(b) CPG learning reward function curve

Fig. 8: Comparison between Cartesian and CPG Action space.

Even though both policies are trained with the same reward, the high return of the Cartesian action space is misleading. In our runs, the Cartesian policy learns a reward-hacking behavior: it generate joint/foot motions that satisfy parts of the reward without producing a physically plausible gait. Qualitatively on the robot, this looks unnatural and not robust: motion is jerky, the robot lowers its center of mass which increases stability, to reach unrealistic step velocities. This inflates the reward mean in simulation, but it would fail on hardware due actuator constraints, and unmodeled compliance. Nevertheless, with better tuning and different observations space parameters, there is a possibility of achieving a fairly realistic behavior.

In contrast, the CPG action space is more constrained: it produces more rhythmic and repeatable stepping patterns, smoother and visually closer to real locomotion. Consequently, although the Cartesian policy achieves a high reward, it does not guarantee better locomotion. Qualitative behavior is equally important.

D. Velocity Tracking Controller

Using the reward function from I we have implemented a velocity tracking controller for 0.5 , 1.0 and 1.5 m/s. To do this we used the trotting gate and trained for 600 000 timesteps for the two first speeds. For 1.5 m/s we needed to retrain using the 1.0 m/s policy for 1 million timesteps.

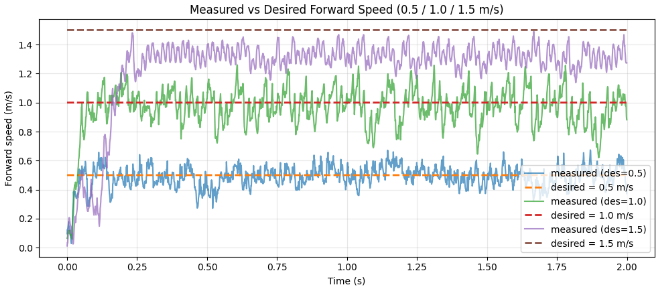


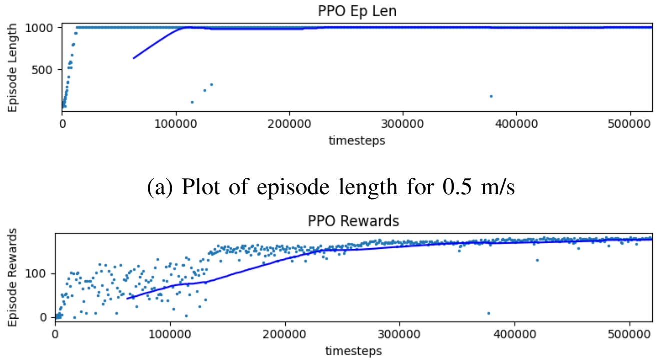
Fig. 9: Velocity tracking performance for 0.5, 1.0, 1.5 m/s

As we can see in 9, the tracking performance is good for the two first speeds. The tracking noise is within 0.1 m/s for the first speed and 0.2 m/s for the second. Tracking a velocity of 1.5 m/s was more difficult for our controller. Indeed the speed tracked was just bellow 1.4 m/s instead. We believe this is because we are starting to surpass the comfort velocities of the trot gait. Thus additional training would need to be conducted to improve performance. Additionally, the first 0.25 s of the graph represents the startup of the gait and is not included in the average velocity calculation.

Metric	0.5 m/s	1.0 m/s	1.5 m/s
Average velocity	0.5	1.0	1.3
CoT	1.37	0.983	0.825
Duty cycle	0.51	0.44	0.41
T_{swing}	0.168 s	0.20 s	0.23 s
T_{stance}	0.175 s	0.16 s	0.16 s

TABLE III: Performance metrics for velocity tracking at different speeds.

The CoT decreases as the velocity increases. It is due to the robot using energy to stabilizes statically. At higher speeds the robot is more dynamically stable, thus pays less of the static cost by traveling a farther distance. The cost might rise again as you combat elements such as air resistance or motor overheat. Swing and stance durations were measured by detecting the contact of each leg on the ground. The mean of all legs was computed to give us the above values.



(a) Plot of episode length for 0.5 m/s

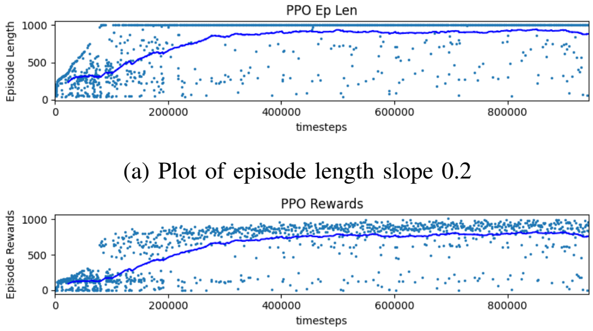
(b) Plot of reward for 0.5 m/s

Fig. 10: Training plots of velocity tracking controller of 0.5 m/s

Finally we can analyze the training plots, see 14,24,25. We can see that the robot learns to stay upright after 100 000 timesteps (apart from 1.5 m/s as it was retrained from 1.0 m/s). Then the reward mean converges as the robot is able to track the desired speed more closely.

E. Task specific Controller

1) First approach: We target the slope task using the reward in Table II, the full observation space, and a CPG PD action space. Our training follows a curriculum-based approach from [3], They found that direct training for a ”jumping onto an obstacle” task would consistently fail as the precise sequence of moves was too difficult to find. Thus they decomposed the movement by first training the policy to jump in place and progressively added complexion to the task. In our case, we first train a base policy for straight-line locomotion on flat terrain using Table I, then fine-tune from this policy on sloped terrain using Table II. This warm-start is expected to accelerate learning compared to training on the slope from scratch. For slope traversal, we use the **walk gait**, prioritizing stability over speed.



(a) Plot of episode length slope 0.2

(b) Plot of reward slope 0.2

Fig. 11: Results of first approach of slope training on fixed 0.2 incline

As seen in the video titled RL_SLOPE.mp4, the robot is able to surmount the slope and has a rather stable gait. The robot did find an ”exploit” where it found it could be more stable by deviating from the straight line. We think that this behavior is acceptable. The reward mean curve stabilizes at a value of 850 and the episode length stays around 1000 seconds. This shows that the agent was effective at learning slope traversal.

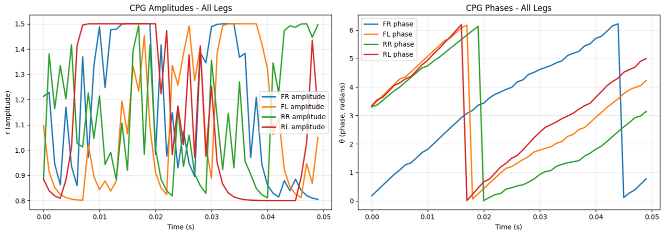


Fig. 12: Plot of CPG amplitudes and phases

We can see from the robot states, see 12, that the locomotion goes as planned. Indeed we can observe an approximate walk gait. Three legs are in phase and one isn’t.

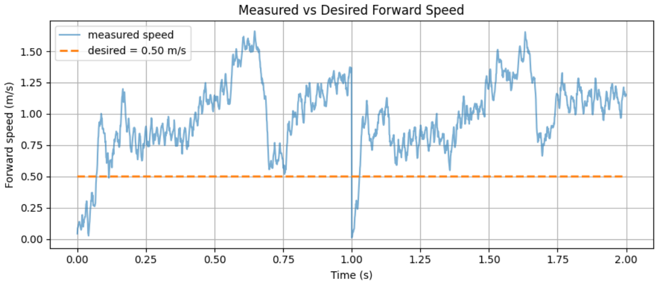
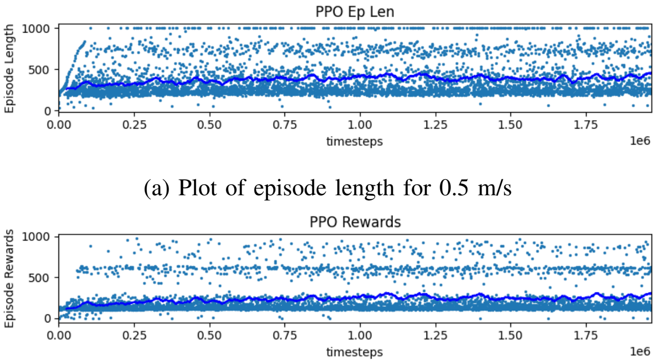


Fig. 13: Velocity curve when traversing slope

To test the velocity performance we plot it across the traversal. At ascension the robot had a velocity of 0.8 m/s. As it gains momentum from the downhill, it reaches a speed of 1.25 m/s which it keeps on the flat terrain after the slope. We can observe the different phases of the traversal graphically. At 1 second the simulation resets.

To test the robustness of the controller we launch the robot on a variation of slopes with which the agent did not train. As we can see from the video titled RL_SLOPE_NOTROBUST.mp4, the quadruped is only able to traverse up to an incline of 24%. This shows that this type of method doesn’t create much robustness as the range of incline values is small.

2) Adaptive curricular training: The above curricular approach is still sub-optimal and doesn’t result in much robustness. There are risks of catastrophic forgetting when changing from flat terrain to sloped so abruptly. As seen is [4] in some cases, pre-trained agents learned slower on the new terrain than fresh agents because they had to escape the deep local optimum of their previous gait. Moreover, this approach does not aid in robustness as the training is done on a fixed slope. More recent research shows the improvements of using adaptive curricular training instead. Papers such as [5] or [6] demonstrate how adding difficulty to the task progressively as the agent learns greatly improves sample efficiency. This inherently creates, (as seen in [7]) robustness as the agent is able to learn with a variety of slope inclines in our case. To implement this, we train first for 800 000 on a fixed incline of 10 % to give the agent a baseline without making it too specialized on fixed slopes. Next we train for 3 million timesteps on a incline randomly varying between 15 and 25 %. This step creates generalized robustness to varied slope inclines. Finally we increase the random variation between 25 and 35 % to train on steeper inclines for 2 million timesteps.



(a) Plot of episode length for 0.5 m/s

(b) Plot of reward for 0.5 m/s

Fig. 14: Training plots of velocity tracking controller of 0.5 m/s

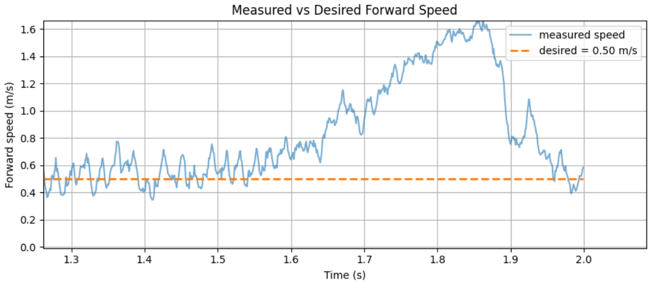


Fig. 15: Velocity curve of robust slope traversing

We test this new controller for a variety of slope inclines. The highest incline the robot can confidently traverse is 29%, see video titled RL_SLOPE_ROBUST.mp4. The robot traverses much larger pitches compared to the controller from III-E1, but does not have a high success rate, as seen in 14a. Indeed, some episodes reach full 1000 timestep length but most fail early. An increase in training time could have created better performance as the reward mean curve had not entirely plateaued. With the added randomness of training, finding a stable policy takes longer. We could also have tuned the reward function to allow for slower traversal for example. Exploring other gaits such as trot is also an option. We believed that the walk gait was more stable at the time. After examining the velocity curve, see 15, we can see that the robot reaches speeds of 1.6 m/s from the downhill, being too high for a stable walk gait. Unfortunately, these solutions weren't explored due to lack of time.

IV. Discussion and conclusion

First we want to discuss how the controllers could be extended :

A key limitation of an open-loop CPG is that swing/stance timing is only implicit (via phase), so contact events on uneven terrain can desynchronize the motion. This can be improved by injecting force/contact feedback into the CPG dynamics: touchdown (or measured normal force) can trigger phase resetting or accelerate/decelerate $\dot{\theta}$ to synchronize the oscillator with the real contact state, while slip or unexpected impacts can increase clearance, extend stance, or temporarily reduce stride length. In parallel, a Virtual Model Control (VMC) layer can be added above the CPG: the CPG provides desired foot trajectories, while VMC regulates trunk objectives (roll/pitch/yaw, height, lateral motion) by computing desired ground reaction forces from virtual springs/dampers acting on the body. These forces are then distributed to stance legs and mapped to joint torques through $\tau = J^{\top}F$. Together, force-feedback CPGs improve timing and terrain adaptation, and VMC improves whole-body stability and heading control without changing the underlying gait structure.

Then, in order to create a velocity tracking controller that tracks arbitrary desired velocities instead of preconditioning, we would need to change learning approaches. First the desired velocity needs to be added to the observation space for the agent to know what it is being asked to do. Next for the controller to be robust to a variety of velocities, we need to train with domain randomization, i.e with a random range of desired velocities. Moreover, to facilitate the learning process we could use a curricular approach as seen above. Starting with a smaller velocity range and ending with a larger one.

Finally, our way of adapting the approach of [5] for curriculum-based learning remains sub-optimal. Indeed, each agent learns at different speeds and requires more or less time at each step of the curricular training. The paper does not overlook this fact and creates an environment where completing a task levels up the robot to a harder one. This way the difficulty is increased more naturally. Moreover, to have better performance on the real hardware, in addition to the observation noise and friction coefficient randomization, we could have added external perturbation to the simulation or even base weight offsetting. This would condition the robot to unexpected variation in the real world. But also to sim-sim transfer problems. Indeed the physics engine could differ and thus require robustness to different dynamic models.

V. AI Declaration

We used a variety of LLMs such as Gemini pro, ChatGPT, Claude, Copilot and DeepSeek. Copilot was used mainly for the code of plots and general questions about the code structure. This tool was helpful as it had access to the whole code base and thus could pinpoint us to where in the files a certain action was being done. Deepseek and ChatGPT were mainly used for theoretical question about deep reinforcement learning. Using those LLM made the RL aspects of the project easier to understand as our group has not taken any RL class yet (we are first semester students). They were also used for some report rewriting. Finally, Gemini was used for deep research into training approaches and paper sourcing.

The most useful (or maybe most exciting) was the use of Gemini for deep research. Indeed after questioning ourselves on different training approaches, we wanted to explore the recent research on this topic. This was the prompt :

I am training a quadruped for legged locomotion using deep RL with PPO algorithm. The end goal is to walk on over a slope obstacle. I have been first training it for 1 million timesteps on flat terrain than retraining it for 1 more million timesteps on the slope obstacle. Is this good or should I do just 2 mil timesteps on the slope obstacle. Is there papers that support this curricular training approach ? if yes please cite them.

Gemini then provided long and complete research on the topic of curriculum-based learning and provided the sources (too long to paste here). What was interesting is how it explore different point of views. First on how a two step approach can be useful. Than how it is sub-opitmal and bettered by adaptive curriculum-based approaches. It then cited a variety of papers, some of which were used to support our tests.

On the other hand DeepSeek's overconfidence sometimes led us astray, especially in topics where we have less experience such as RL. When testing neural network architecture I had this question :

I tried a training with [512, 256, 128] on flat ground and it worked well although it was little slow to converge at the beginning. Then I retrained on a slope from the last training as checkpoint and it performed far less good than the original [256,256] why do you think that is ?

The answer we got was the following :

This is a classic case of catastrophic forgetting and overfitting! You've hit a fundamental issue in transfer learning for locomotion. Here's exactly what's happening and how to fix it:

The Problem: "Specialist" vs "Generalist" Networks

Your `[512, 256, 128]` network became a **flat ground specialist**:

- Learned extremely precise weight distributions for flat walking
- Developed **brittle, over-optimized** motor patterns
- "Memorized" flat terrain physics instead of learning general principles

Your `[256, 256]` network remained a **generalist**:

- Couldn't perfectly optimize flat terrain
- Learned **robust, adaptable** strategies
- Transferred better to new conditions

This answer ended being more confusing than helpful as there ended up just being a mistake in the code. Moreover, DeepSeek's over confident tone persuaded us in the answer more than it should have.

To conclude, AI made plotting graphs, understand parts of code and RL theory a smoother experience. Moreover, Gemini's deep research tool was very helpful in satisfying our curiosity of training approaches. These tools did not replace our own thoughts or research but aided us in our goal to learn.

REFERENCES

[1] G. Bellegarda and A. Ijspeert, *Cpg-rl: Learning central pattern generators for quadruped locomotion*, 2022. arXiv: 2211.00458 [cs.RO]. [Online]. Available: https://arxiv.org/abs/2211.00458.

[2] W. Jones, T. Blum, and K. Yoshida, "Adaptive slope locomotion with deep reinforcement learning," in *2020 IEEE/SICE International Symposium on System Integration (SII)*, 2020, pp. 546–550. DOI: 10.1109/SII46433.2020.9025928.

[3] V. Atanassov, J. Ding, J. Kober, I. Havoutis, and C. D. Santina, *Curriculum-based reinforcement learning for quadrupedal jumping: A reference-free design*, 2024. arXiv: 2401.16337 [cs.RO]. [Online]. Available: https://arxiv.org/abs/2401.16337.

[4] G. Minelli and V. Vassiliades, "Towards continual reinforcement learning for quadruped robots," *International Conference on Interactive Media, Smart Systems and Emerging Technologies (IMET)*, pp. 61–64, 2023. DOI: 10.2312/IMET.20231258. [Online]. Available: https://diglib.eg.org/handle/10.2312/imet20231258.

[5] N. Rudin, D. Hoeller, P. Reist, and M. Hutter, *Learning to walk in minutes using massively parallel deep reinforcement learning*, 2022. arXiv: 2109.11978 [cs.RO]. [Online]. Available: https://arxiv.org/abs/2109.11978.

[6] S. Li, G. Wang, Y. Pang, et al., "Learning agility and adaptive legged locomotion via curricular hindsight reinforcement learning," *Scientific Reports*, vol. 14, no. 1, p. 28 089, 2024. DOI: 10.1038/s41598-024-79292-4. [Online]. Available: https://doi.org/10.1038/s41598-024-79292-4.

[7] A. Kumar, Z. Fu, D. Pathak, and J. Malik, *Rma: Rapid motor adaptation for legged robots*, 2021. arXiv: 2107.04034 [cs.LG]. [Online]. Available: https://arxiv.org/abs/2107.04034.

Appendix A: Hyperparameter Ablation Study

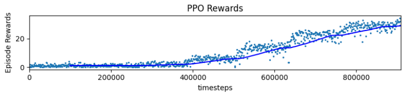
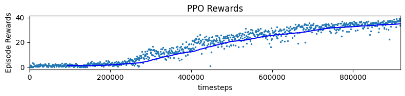
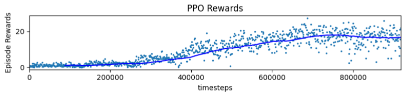
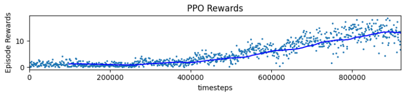
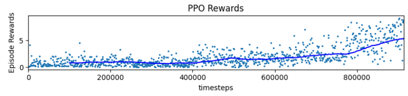
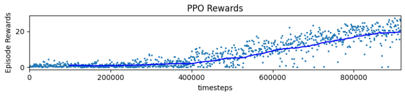
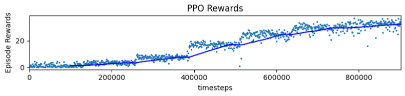
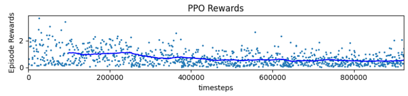
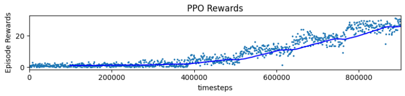
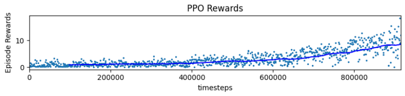
Parameter	Learning Curve
Default	 (a) Default hyperparameters
Clip Range	 (b) Clip range = 0.4
Learning Rate	 (c) Learning rate = 1e-3
Entropy	 (d) Entropy = 0.1
Batch size	 (e) batch size = 512
GAE λ	 (f) GAE Lambda = 0.8
n_epochs	 (g) n_epochs = 25
Gamma	 (h) Gamma = 0.8
Max grad	 (i) Max gradient norm = 1
Vf coef	 (j) Vf coef = 0.1

Fig. 16: PPO hyperparameter ablation study comparing hyperparameter effects on reward mean curve

Appendix B: CPG States

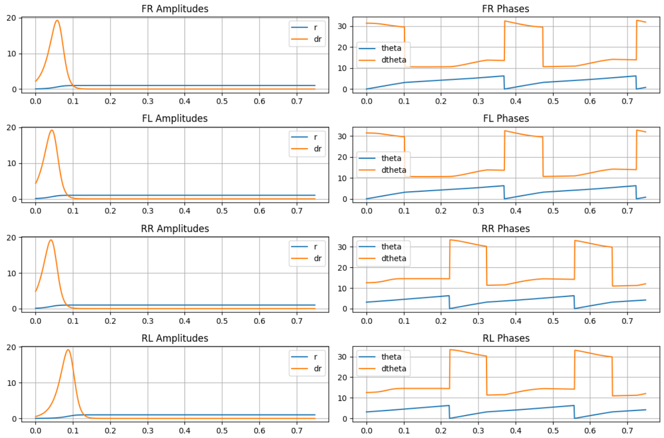


Fig. 17: CPG state trajectories for the BOUND gait.

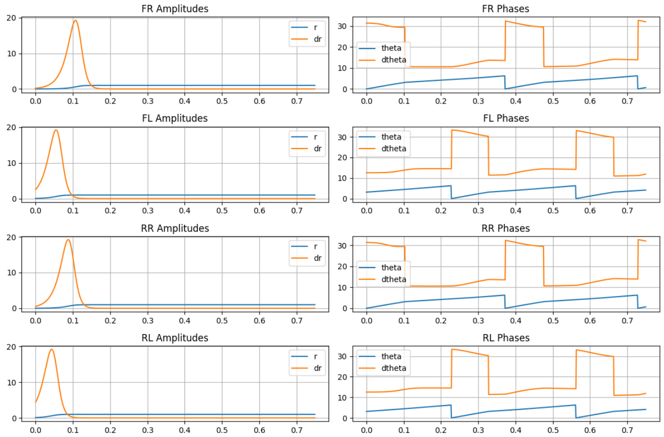


Fig. 18: CPG state trajectories for the PACE gait.

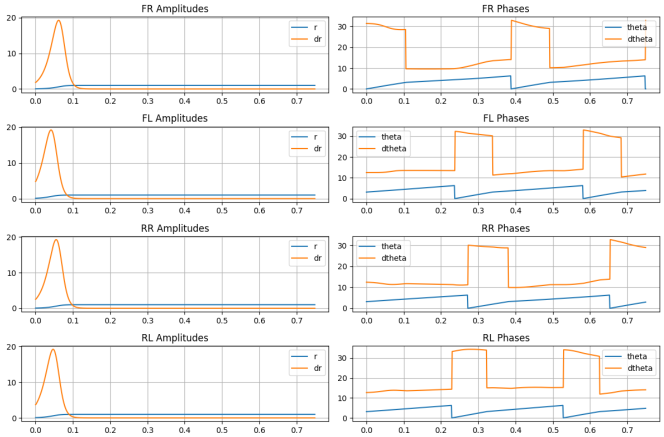


Fig. 19: CPG state trajectories for the WALK gait.

Appendix C: Architecture Ablation Study

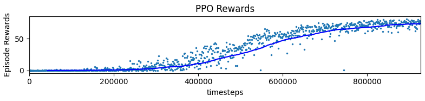
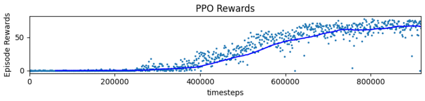
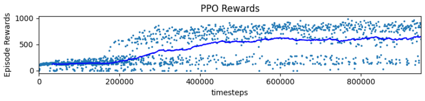
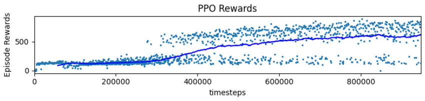
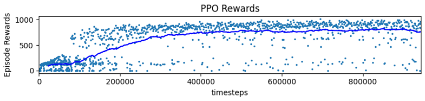
Parameter	Learning Curve
2 layer on flat	 (a) Final reward mean = 76.4
3 Layer on flat	 (b) Final reward mean = 73.6
3 Layer on slope retrained	 (c) Final reward = 694
3 Layer on slope fresh	 (d) Final reward = 686
2 Layer on slope retrained	 (e) Final reward = 729

Fig. 20: Effects of different neural network architecture on reward curve

Appendix D: K_p Comparison for other gaits

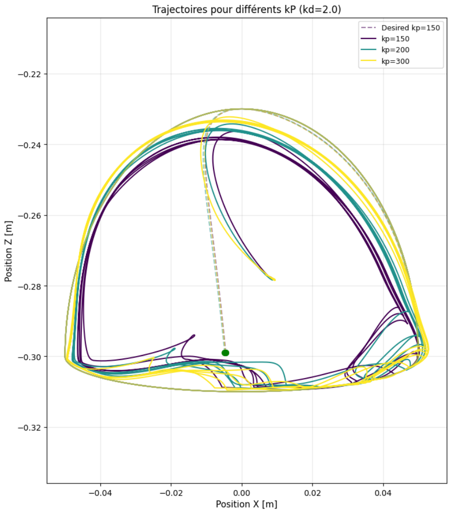


Fig. 21: WALK gait - Tests for different K_p values

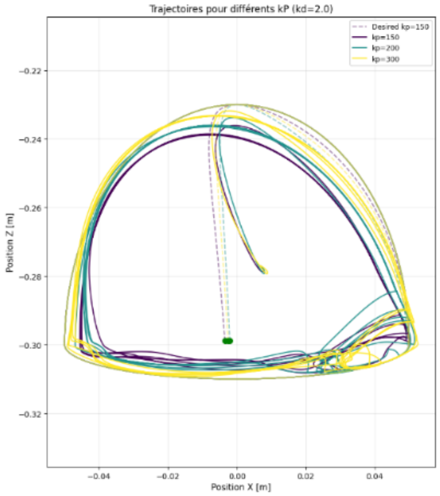


Fig. 22: TROT gait - Tests for different K_p values

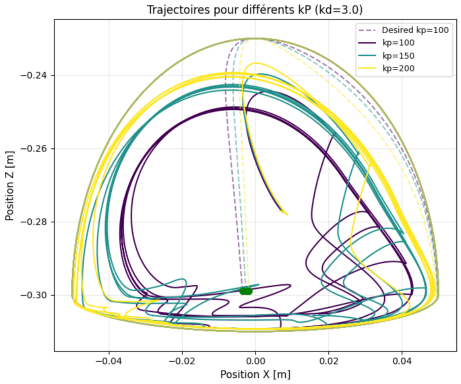
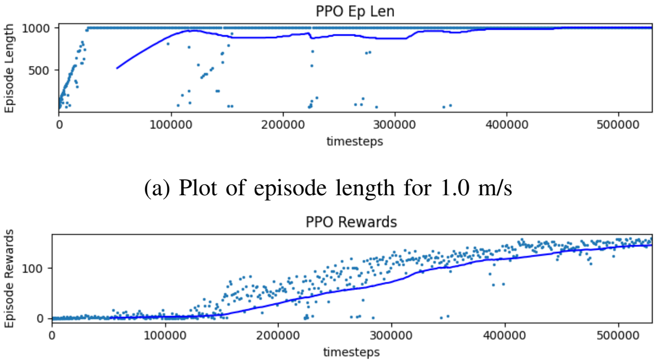


Fig. 23: BOUND gait - Tests for different K_p values

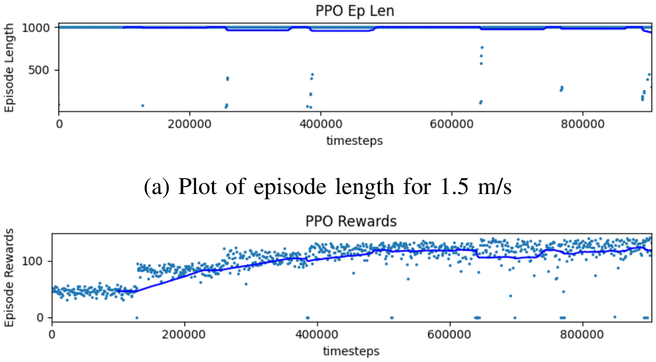
Appendix E: K_p Training plots of velocity tracking



(a) Plot of episode length for 1.0 m/s

(b) Plot of reward for 1.0 m/s

Fig. 24: Training plots of velocity tracking controller of 1.0 m/s



(a) Plot of episode length for 1.5 m/s

(b) Plot of reward for 1.5 m/s

Fig. 25: Training plots of velocity tracking controller of 1.5 m/s

APPENDIX F: DESIRED VS REAL OF OTHER GAITS

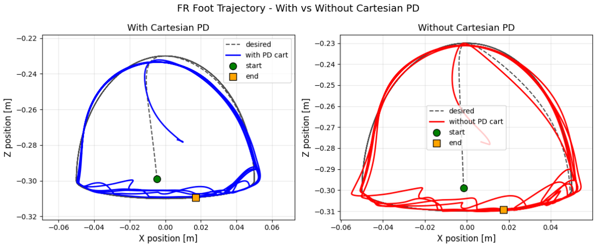


Fig. 26: WALK gait - $K_p = 300$

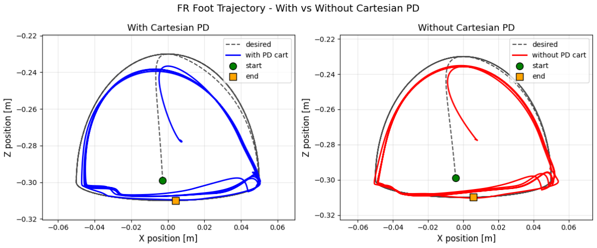


Fig. 27: PACE gait - $K_p = 150$

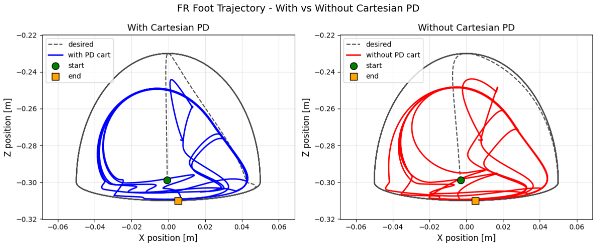


Fig. 28: BOUND gait - $K_p = 100$

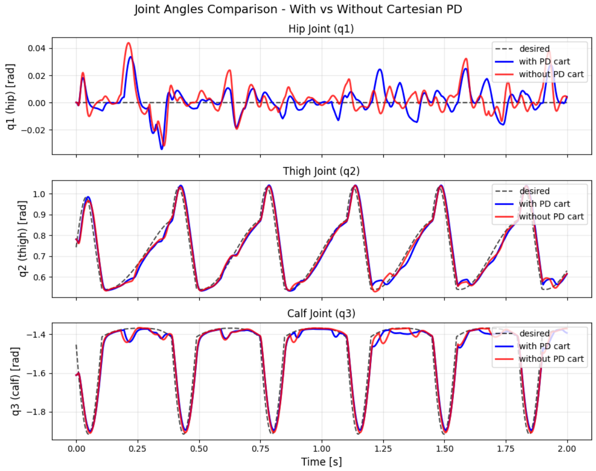


Fig. 29: WALK gait - $K_p = 300$

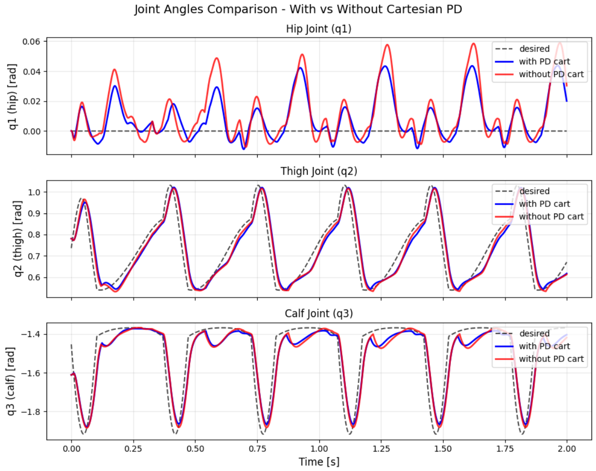


Fig. 30: PACE gait - $K_p = 150$

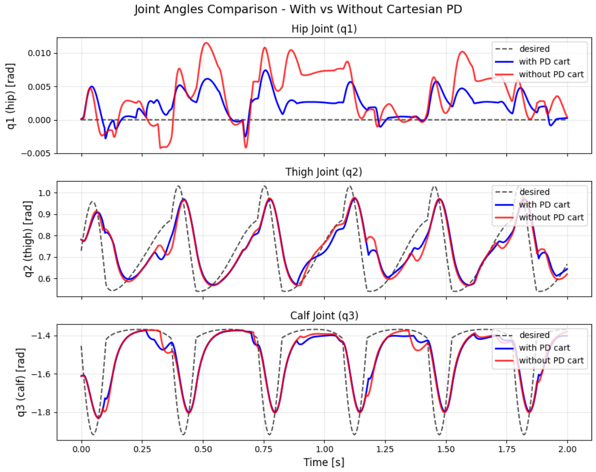


Fig. 31: BOUND gait - $K_p = 100$