MICRO 507 - Legged Robots

Project

Quadruped Locomotion with Central Pattern Generators and Deep Reinforcement Learning

Mehmet Furkan Doğan[1],
Ali Fuat Şahin[1], and
Mustafa Emre Gürsoy[1]

[1]École Polytechnique Fédérale de Lausanne

Prof. Auke Ijspeert

Contents

1 Introduction

The realm of quadruped locomotion is at the forefront of robotic advancements, pushing the boundaries of agility and adaptability in autonomous systems. In this project, we embark on a dual exploration, navigating the intricate landscapes of Central Pattern Generators (CPG) and state-of-the-art Deep Reinforcement Learning (DRL) algorithms, to unravel the secrets behind versatile and dynamic quadrupedal movement.

CPGs have emerged as a bio-inspired approach in legged robots, mimicking the rhythmic neural circuits observed in living organisms. Applying CPGs in legged robots aims to imbue robots with stable and coordinated movements reminiscent of natural locomotion. As usual, it comes with its own set of advantages and disadvantages. On the positive side, CPGs offer inherent biological mimicry, providing efficiency and stability to legged robots. Their ability to generate rhythmic and coordinated movements enhances resilience in dynamic environments, and reduced reliance on sensory feedback streamlines integration efforts. Nevertheless, challenges arise in terms of limited adaptability, intricate tuning requirements, and a dependency on predefined gaits.

Recently, the popularity of machine-learning applications in legged robots has increased due to advancements in machine-learning techniques, increased computation powers, and the availability of faster simulators. They offer generic methods requiring less expertise than model-based methods and apply to complex structures. On the other hand, learning algorithms usually require long training sequences and lack proof of stability.

In this work, we are going to apply and combine both methods and investigate their performances depending on different parameters.

1.1 Quadruped Specifications

The robot used in this study is Unitree A1. Technical specifications of the robot are as follows.

- Dimensions: 500x300x400 mm
- Weight: 12kg
- Max Speed: 3.3m/s
- Torque/Motor: 33.5N.m
- Rotation Speed/Motor: 21rad/s

1.2 Quadruped Model

The quadruped has 4 legs each with 3 joints (hip, thigh, and calf) corresponding to 12 motors. The legs of the robot are numbered 0-3 in order Front Right (FR, 0), Front Left (FL, 1), Rear Right (RR, 2), and Rear Left (RL, 3). The motor order is always hip, thigh, and calf. In addition, the coordinate system in the simulations is fixed on the robot x-axis is directed at the direction of movement while the y-axis is directed to the left and the z-axis is perpendicular to the robot body and directed upwards. The following figures demonstrate the visualization of the ordering and the coordinate system.

From the coordinate system definition, the relationship between the foot position and the joint angles can be defined as follows.

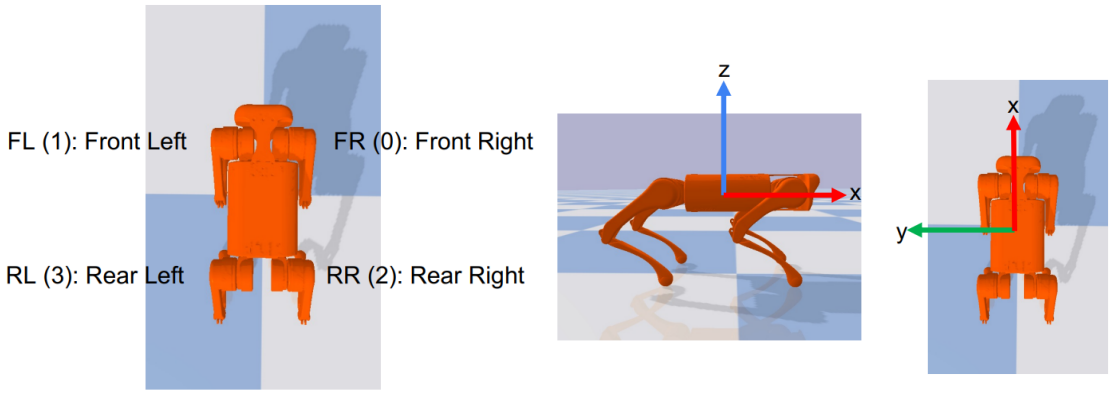
$$\mathbf{p} = f(\mathbf{q}) \tag{1}$$

Note that the function $f(.)$ denotes the forward kinematics. Jacobian matrix can be obtained by differentiating this function to obtain the foot velocities and torque and foot contact force relationships.

$$\mathbf{v} = \mathbf{J}(\mathbf{q}) \cdot \dot{\mathbf{q}} \tag{2}$$

$$\tau = \mathbf{J}^T(\mathbf{q})\mathbf{F} \tag{3}$$

These three relationships construct the fundamentals of the model throughout the simulations and will be repeatedly used throughout the study.



(a) Leg orders.

(b) Coordinate system definition.

1.3 Control Policies

In this project, we are going to implement both Cartesian and Joint PD controllers. For position control of a leg, the following formula can be used.

$$\tau_{joint} = \mathbf{K}_{p,joint}(\mathbf{q}_d - \mathbf{q}) + \mathbf{K}_{d,joint}(\dot{\mathbf{q}}_d - \dot{\mathbf{q}}) \tag{4}$$

Note that $\mathbf{q}_d$ implies the desired joint angles whereas $\mathbf{q}$ denotes the angle measurements from the joint encoders. The gains $\mathbf{K}_{p,joint}$ and $\mathbf{K}_{d,joint}$ are proportional and derivative gains, respectively, and they are to be optimized in the study.

In addition, to improve the tracking performance, Cartesian PD controllers are also implemented. From the desired joint angle and velocity trajectories, desired foot positions and velocities are extracted using the Jacobian matrix to calculate the motor torques required to track the trajectory.

$$\tau_{Cartesian} = \mathbf{J}^T(\mathbf{q})[\mathbf{K}_{p,Cartesian}(\mathbf{p}_d - \mathbf{p}) + \mathbf{K}_{d,Cartesian}(\mathbf{v}_d - \mathbf{v})] \tag{5}$$

Similarly, the gains $\mathbf{K}_{p,Cartesian}$ and $\mathbf{K}_{d,Cartesian}$ are proportional and derivative gains, respectively, and they are to be optimized in the study.

1.4 Report Structure

The details of the design choices of our CPG model and RL model are contained in sections 2 and 3. Lastly, the discussion of the results can be found in section 4.

2 Central Pattern Generators

2.1 Introduction

In this section, we are going to investigate open-loop CPG controller performance with various parameters and different gaits. CPGs generate periodic oscillations for various duty factors, stride durations, swing, and stance frequencies. Given commands are to be executed with PD controllers in joint and cartesian spaces.

2.2 Method

2.2.1 Hopf Oscillators

Our locomotion controllers are based on CPGs consisting of coupled cartesian space Hopf oscillators. Oscillator equations for any leg i can be written as follows.

$$\dot{r}_i = \alpha(\mu - r_i^2)r_i \tag{6}$$

$$\dot{\theta}_i = \omega_i + \sum_{j=0}^{3} r_j w_{ij} \sin(\theta_j - \theta_i - \phi_{ij}) \tag{7}$$

Where r_i is the amplitude of the oscillator, θ_i is the phase of the oscillator, $\sqrt{\mu}$ is the desired amplitude of the oscillator, α is the positive constant that determines the speed of convergence to the limit cycle, ω_i is the natural frequency of the oscillations, w_{ij} is the coupling strength between the oscillators i and j, and ϕ_{ij} is the desired phase offset between oscillators i and j.

2.2.2 Gaits and Coupling Matrices

In this study, we are going to investigate 7 of the most common gaits which are characterized by their coupling matrices, and swing and stance frequencies. Some of the most common gaits can be seen in the following figure. Studied gaits are trot, pace, walk, bound, and gallop, with their coupling matrices being hard-codded using the known footfall sequences.

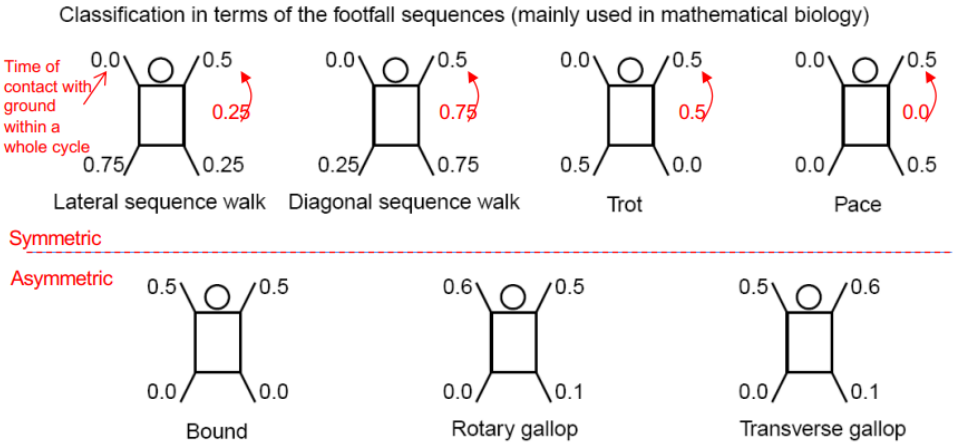


Figure 2: Most common quadruped gaits.

2.2.3 Swing and Stance Frequencies

As we have discussed, here we will split the natural frequency ω_i of any leg i between ω_{swing} and ω_{stance}, where the phase variable of the oscillator i will dictate the swing or stance phase of the leg i. The swing phase is defined as $0 \leq \theta_i \leq \pi$ and the stance phase is defined as $\pi \leq \theta_i \leq 2\pi$. The natural frequencies of the swing and stance phases are optimized to increase the locomotion performance.

To evaluate the gait performance we defined duty ratio which is the duration of one cycle in which the foot is in the stance phase for one cycle in which the foot is in the swing phase. The natural frequencies are optimized based on the gaits since in each gait the duty ratio should be dependent on the number of legs touching the ground at the same time and the number of alternating legs. The duty ratio can be calculated depending on the swing and stance periods from the following formula.

$$D = \frac{T_{stance}}{T_{swing}} = \frac{\omega_{swing}}{\omega_{stance}} \tag{8}$$

The calculation of the duty ratio for different speeds of various gaits provides a reference for tuning the swing and stance phase frequencies.

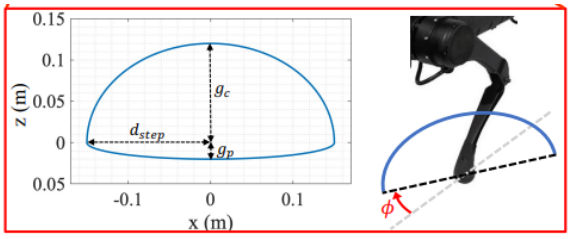
2.2.4 Mapping from CPG States to Cartesian Foot Positions

There are multiple methods to convert CPG states to joint commands. As suggested in the instructions the following formulation is implemented for mapping states to cartesian foot positions in the leg xz plane.

$$x_{foot} = -d_{step} r_i \cos(\theta_i) \tag{9}$$

$$z_{foot} = \begin{cases} -h + g_c \sin(\theta_i) & \text{if } \sin(\theta_i) > 0 \\ -h + g_p \sin(\theta_i) & \text{otherwise} \end{cases} \tag{10}$$

Where d_{step} is step length, h is the robot height, g_c is the max ground clearance during swing and g_p is the max ground penetration during stance. All of these parameters are to be optimized to increase the locomotion performance. The default parameters are given in the following table.



(a) The parameters of the mapping visualized for one leg.

Parameters	Values [m]
Step Length (d_{step})	0.04
Ground Penetration (g_p)	0.01
Ground Clearance (g_c)	0.05
Robot Height (h)	0.25

(b) Default parameters for mapping.

2.2.5 Controllers and Tuning

After tuning the discussed parameters, desired foot positions will be reached using joint PD and cartesian PD controllers. To have a better performance we have implemented a combination of joint and cartesian PD controllers.

The tracking performance of the controller depends on the controller gains. Default gains are given as follows.

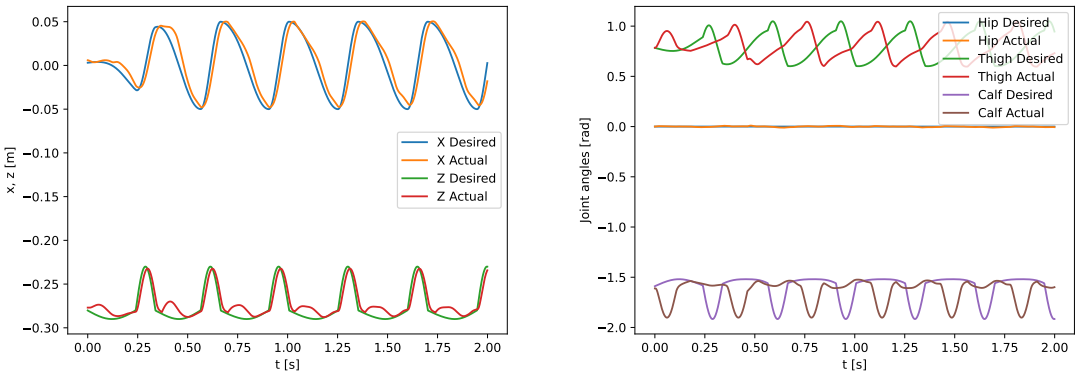
$$\mathbf{K}_{p,joint} = \begin{bmatrix} 100 & 100 & 100 \end{bmatrix}$$
$$\mathbf{K}_{d,joint} = \begin{bmatrix} 2 & 2 & 2 \end{bmatrix}$$
$$\mathbf{K}_{p,cartesian} = \text{diag}(\begin{bmatrix} 500 & 500 & 500 \end{bmatrix})$$
$$\mathbf{K}_{d,cartesian} = \text{diag}(\begin{bmatrix} 10 & 10 & 10 \end{bmatrix})$$

These gains are optimized based on the desired and actual foot position plots before testing with other parameters of CPG states and mapping functions.

2.3 Discussion

2.3.1 Controller Parameters and CPG States

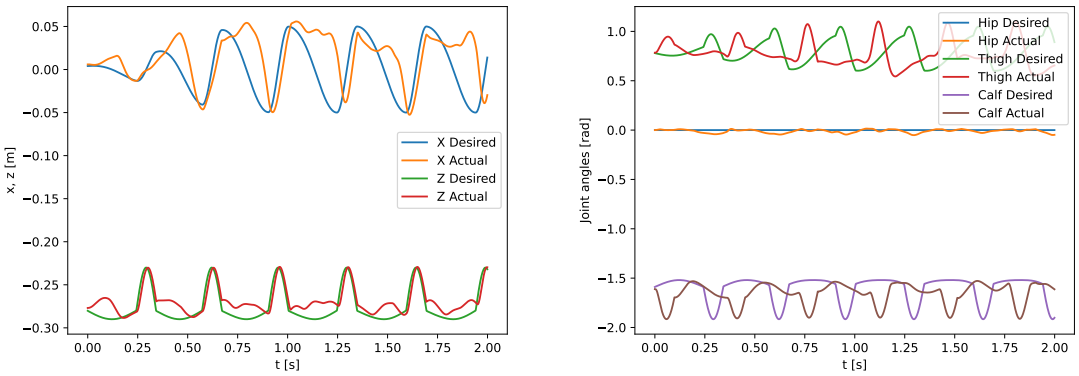
It can be observed from Figures 4, 5 and 6 that the best performance is obtained whenever the combination of joint and cartesian PD controllers are used. Also, the worst performance is obtained when only a cartesian PD controller is used. Comparing the performance of sole cartesian and sole joint PD controllers, we can observe that only using joint PD results in smoother tracking performance. We expect that since cartesian PD uses only foot positions and joint PD calculates the joint angles directly.



(a) x and z positions

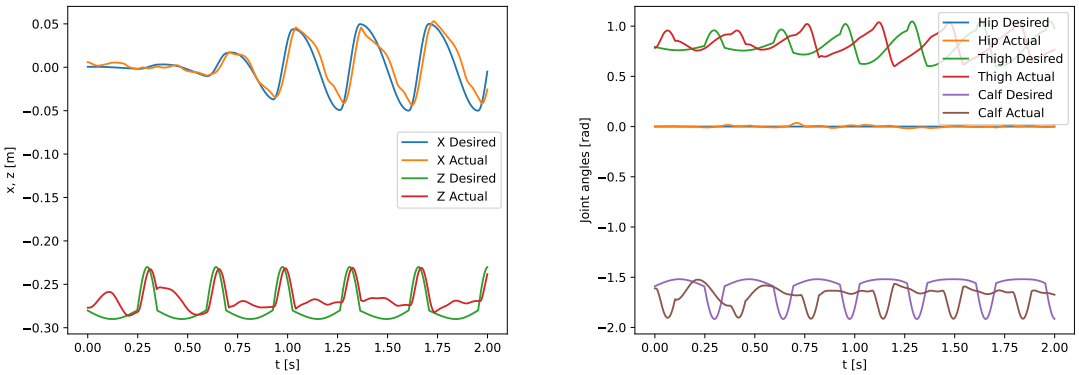
(b) hip thigh and calf angles

Figure 4: Actual and desired parameters



(a) x and z positions (b) hip thigh and calf angles

Figure 5: Actual and desired parameters using only Joint PD



(a) x and z positions (b) hip thigh and calf angles

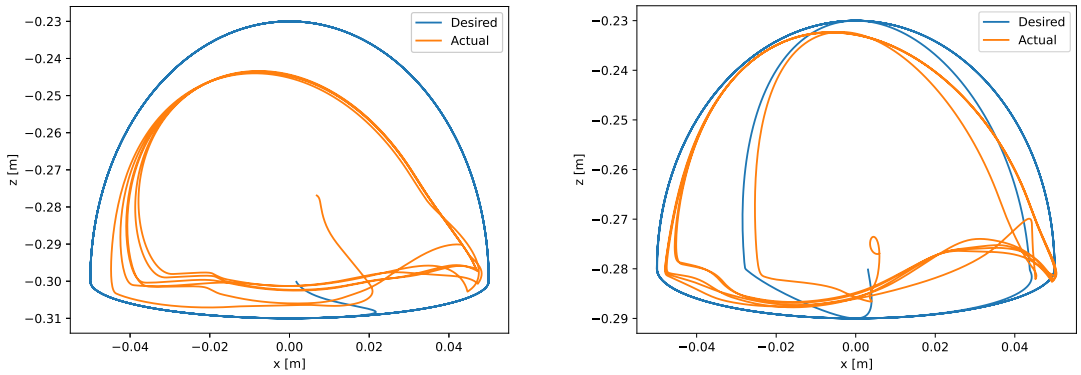
Figure 6: Actual and desired parameters using only Cartesian PD

In Figure 7a, the resulting limit cycle of using default gains and the default CPG parameters can be observed. To improve the tracking performance of the controller, we have changed the default gains of the controllers. The following gains for the controllers are used.

$$\mathbf{K}_{p,joint} = \begin{bmatrix} 200 & 50 & 80 \end{bmatrix}$$
$$\mathbf{K}_{d,joint} = \begin{bmatrix} 2 & 0.7 & 0.7 \end{bmatrix}$$
$$\mathbf{K}_{p,cartesian} = \text{diag}\left(\begin{bmatrix} 2400 & 2400 & 2400 \end{bmatrix}\right)$$
$$\mathbf{K}_{d,cartesian} = \text{diag}\left(\begin{bmatrix} 35 & 35 & 35 \end{bmatrix}\right)$$

It can be observed from the gains that in the combination cartesian PD is dominant. This

is consistent with our mapping since from the mapping we are calculating foot positions in the cartesian workspace. In figure 7b resulting limit cycle of modified gains can be observed.



(a) Before optimization (b) After optimization

Figure 7: Desired foot position tracking with default controller gains and CPG parameters shown in leg xz plane.

Also Figure 8 demonstrates the open loop performance of the controller on the CPG states. The Hopf oscillator model encapsulates the rhythmic and cyclic patterns underlying the leg movements during the trot gait. The given figure demonstrates the states of the Hopf oscillator.

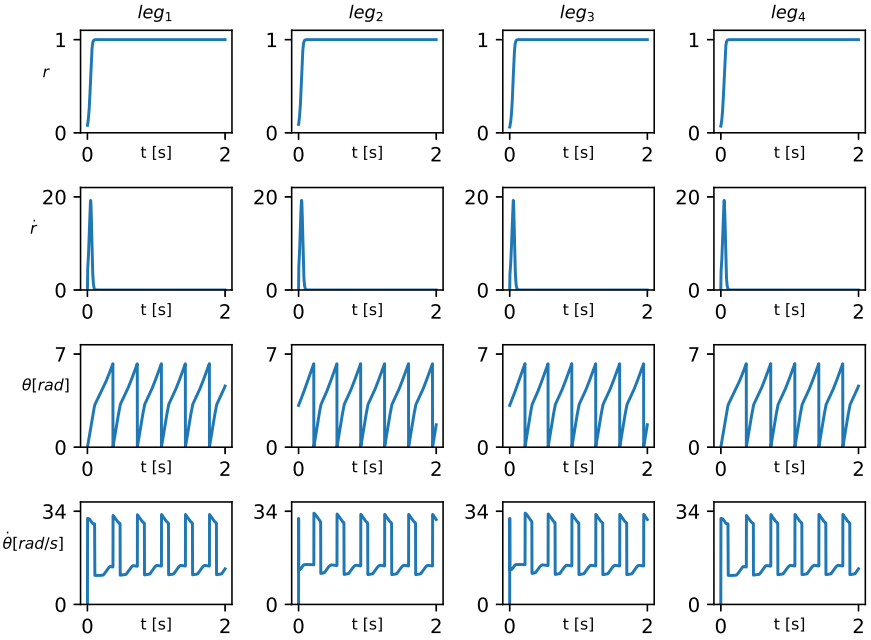


Figure 8: CPG States for a trot gait over time.

The amplitude (r) captures the oscillatory strength, while the phase (θ) characterizes the cyclic progression. Radial and angular velocities ($\dot{r}$ and $\dot{\theta}$) provide insights into the dynamics of these oscillatory patterns. By examining the states of each leg's Hopf oscillator, we gain a comprehensive understanding of the coordinated and synchronized behaviors inherent in the trotting motion.

2.3.2 CPG and Mapping Parameters

Given default parameters for the gaits are pre-tuned for all the gaits that we are working on in this project. Indeed, when ideal swing and stance frequencies are selected, gait performances are fine. However, for better gait-specific performance we have modified these parameters.

Table 1: Gait-specific parameters to increase performance.

Gait	ω_{swing}	ω_{stance}	d_{step}	g_p	g_c	h
Trot	10π	4π	0.05	0.01	0.08	0.25
Pace	13π	32π	0.09	0.007	0.085	0.23
Bound	13π	32π	0.09	0.007	0.085	0.23
Lateral Sequence Walk	16π	8π	0.03	0.01	0.045	0.26
Diagonal Sequence Walk	16π	8π	0.07	0.02	0.045	0.26
Rotary Gallop	20π	40π	0.06	0.012	0.05	0.21
Transverse Gallop	16π	36π	0.065	0.0075	0.085	0.23

The meanings of these parameters and the considerations while tuning are summarized as follows.

- **Stance and Swing Frequencies:** These constants dictate the swing/stance status of the legs. Increasing these parameters causes shorter periods for swing/stance phases. Parameters are tuned considering the duty ratio. In slower gaits, the duty ratio is increased whereas in faster gaits it is decreased. Also, the stance frequency is mainly determined by ground penetration and the robot height since these are the two parameters that define the foot trajectory in the stance phase. Similarly, swing frequency is determined by ground clearance and step length since these parameters define the foot trajectory in the swing phase.
- **Step Length:** Step length is constrained by the kinematic workspace of the robot. The magnitude of step length depends on the speed of the gait. While increasing it results in better performance in faster gaits, it is better to keep it smaller for stability with slower gaits. In addition, too much increase causes instability and a decrease in locomotion performance for faster gaits. Together with the ground clearance parameter, they define the length of the step trajectory.
- **Ground Penetration:** Larger ground penetration causes larger ground reaction forces. Therefore, the robot bounds higher with larger ground penetration and will be less slippery. In the case of slower gaits, keeping this variable smaller leads to a better gait performance. On

the other hand, faster gaits with a smaller stance period require larger ground penetration. Together with robot height, this parameter defines the length of the foot trajectory in the stance phase.

- **Ground Clearance:** This parameter depends on the terrain and the kinematic workspace of the robot. Large ground clearance might be unreachable due to the joint limits of the robot. Very low ground clearance decreases locomotion performance in uneven terrain and might even cause backward locomotion due to the foot rubbing to the ground. Together with the step length, they define the length of the step trajectory.
- **Robot Height:** Robot height should be adjusted considering the ground clearance parameter. The lower the robot's height, the more stable the locomotion; however, it constrains the kinematic workspace of the robot. The lower robot height requires smaller ground clearance. Together with ground penetration, this parameter defines the length of the foot trajectory in the stance phase.

The performance parameters of each gait can be found in Table 2 with tuned parameters. Please note that in some of the gaits CoT is much larger than it should be. This CoT can be decreased by decreasing swing stance frequencies and manipulating the duty ratio for each gait. In our case, we have aimed for higher velocities for some of the gaits and large CoTs are ignored.

Table 2: Gait-specific parameters to increase performance.

Gait	Body Velocity [m/s]	D	T_{stance} [ms]	T_{swing} [ms]	CoT
Trot	0.39	1.46	174	130	0.74
Pace	3.12	0.20	11	58	1.60
Bound	2.19	0.21	55	255	2.41
Lateral Sequence Walk	0.42	1.29	102	79	0.53
Diagonal Sequence Walk	0.47	0.53	66	125	1.06
Rotary Gallop	1.58	0.12	15	127	2.10
Transverse Gallop	1.72	0.11	27	235	1.72

2.3.3 Possible Improvements in Controllers

One of the most important problems of pure feedforward control is that it is very susceptible to environmental noise and process noise. Therefore, to increase the locomotion performance, controllers can be extended with feedback loops with descending modulation.

The foot contact booleans and the ground reaction forces are some of the most useful sensory readings that we have observed through the deep reinforcement learning process. Therefore, as a reflex model, force feedback that modified CPG parameters can be integrated into the CPG model. Here we propose a feedback model that modifies the oscillator states based on the contact information and ground reaction forces [1].

Considering the Hopf oscillator equations, the control input can be added to the $\dot{\theta}$ equation.

$$\dot{r}_i = \alpha(\mu - r_i^2)r_i \tag{11}$$

$$\dot{\theta}_i = \omega_i + \sum_{j=0}^{3} r_j w_{ij} \sin(\theta_j - \theta_i - \phi_{ij}) + u_i \tag{12}$$

The control input is added to the $\dot{\theta}_i$ equation rather than $\dot{r}_i$ equation since θ determines if we are in the stance or swing phase. By controlling $\dot{\theta}_i$ one can dictate the beginning and ending of the phases. Also, r_i mainly dictates the trajectory of the gait. Therefore, adding control input to θ_i ensures r_i variables produce a smooth trajectory.

Control input is determined depending on the pressure sensor readings. We want a leg to stop in a swing-to-stance transition if it is not in contact with the ground yet. Or, in a stance-to-swing transition when the ground reaction forces are large enough to indicate that the leg still supports significant body weight. We stop the leg by equating $\dot{\theta}_i$ to zero. This control eliminates the coupling effect and stops the leg. The control input for this case is defined in Eq. (13).

$$u_i = -\omega_i - \sum_{j=0}^{3} r_j w_{ij} \sin(\theta_j - \theta_i - \phi_{ij}) \tag{13}$$

In addition, we want to fast-forward the phase transitions when the weight under a leg is very low in the stance phase or when the foot touches the ground during the swing phase. To fast forward a phase, we need to reset variable θ_i. Therefore, the $\dot{\theta}_i$ variable should be increased to fast forward a phase. In this case, we define a constant τ which dictates the transition. The control input for this case is implemented as in Eq. (14).

$$u_i = \tau \tag{14}$$

Variable τ is defined such that when the control input is applied, it is larger than the other terms in $\dot{\theta}_i$ equation so that $\dot{\theta}_i \approx \tau$. Therefore, the speed of the transition can be increased by increasing τ and the delay in the transition can be controlled.

In summary, the control input is defined as in Eq. (15).

$$u_i = \begin{cases} \tau & \text{if fast-forward transition} \\ -\omega_i - \sum_{j=0}^{3} r_j w_{ij} \sin(\theta_j - \theta_i - \phi_{ij}) & \text{if stop transition} \\ 0 & \text{otherwise} \end{cases} \tag{15}$$

3 Deep Reinforcement Learning

3.1 Introduction

This section is dedicated to explaining the implementation of reinforcement learning on quadruped locomotion. First, we explained the method we are using for reinforcement learning and then the resulting performances are discussed.

3.2 Method

3.2.1 Action Space

Available action spaces are CPG, cartesian PD, and joint PD. For joint PD, the action space consists of twelve actions, each scaled linearly to the lower and upper joint angle limits we have specified. We are always making sure that the actions stay within the limits by utilizing the **numpy** command **clip**.

In the case of Cartesian PD, the action space again includes twelve actions, where each of them is mapped to the desired foot positions. The mapping is done by using a scale matrix multiplied by the actions, and the scaled actions are added to the nominal foot positions. The choice of scaling matrix is the most vital decision for this part since it gives the robot relative freedom in different axes. The scaling matrix looks like this:

$$\mathbf{scale} = \begin{bmatrix} 0.12 & 0.075 & 0.12 \end{bmatrix} \tag{16}$$

We deliberately gave the scale on the Y axis a lower value, since we don't want the robot to make radical movements on that axis, and we wanted to keep the desired Y value close to the nominal one. However, we have found that for more complex tasks like 3D exploration and locomotion, keeping this value too low results in the robot not being able to turn sufficiently to the goal direction. For the forward locomotion task, we did not change the scaling matrix and used the default values. However, for the flag run task, we have increased the movement range in the y direction to let the robot turn more efficiently.

Similarly, with the joint PD, we are clipping the actions to the range $\begin{bmatrix} -1 & 1 \end{bmatrix}$ as it is a standard RL technique.

Lastly, we have tried the CPG-RL, mapping the actions coming from the RL to the frequencies and the desired amplitudes of oscillations. For this approach, the actions space includes eight actions, hence computationally this method is more effective.

We have tried two main approaches with the CPG-RL. The first one was to keep the coupling matrices and enforce a specific gait, and the second one was to get rid of the coupling matrices and leave the robot the freedom to come up with the most appropriate gait. We have observed that some of the natural gaits come up over time. For the lower speeds in forward locomotion, trot gait is more dominant, however with higher speeds gaits similar to a bound gait has emerged.

3.2.2 Reward Function

We have defined several reward components for different aspects of tasks. They can be seen in Table 3. [2]

Table 3: Reward function components.

Reward	**Expression**
Linear Velocity	$r_v = k_v \cdot e^{-\|\|cmd_{vxy} - V_{xy}\|\|}$
Angular Velocity	$r_\omega = k_\omega = e^{-1.5 \cdot (cmd_{\omega z} - \omega_z)^2}$
Airtime	$r_{air,i} = \begin{cases} k_a \cdot \min(T_{s,i}, 50) & \text{if } T_s < 80 \\ 0 & \text{otherwise} \end{cases}$
Foot Slip	$r_{slip,i} = k_{slip} \cdot C_{f,i} \|\|V_{f,xy,i}\|\|$
Foot Clearance	$r_{cl,i} = k_{cl} \cdot (p_{f,z,i} - p_{f,z}^{des})^2 \cdot$
Orientation Z	$r_{ori,z} = k_{ori,z} \cdot (\text{angle}(\phi_{body,z}, \phi_{world,z}))^2$
Base Motion	$r_{base,m} = k_{base} \cdot (0.8V_z^2 + 0.2\|\omega_x\| + 0.2\|\omega_y\|)$
Base Position	$r_{base,pos} = k_{base,pos} \cdot (p_{b,z} - 0.26)^2$
Drift	$r_{drift} = k_{drift} \cdot \|p_{b,y} - p_{b,y,des}\|$
Energy	$r_{\text{energy}} = k_e \cdot \sum_i \|\boldsymbol{\tau}_i \cdot \mathbf{v}_i\| \cdot \Delta t$
Distance to goal	$r_{dist} = k_{dist} \cdot \Delta\text{distance}$
Angle to goal	$r_{yaw} = k_{yaw} \cdot e^{-\|\text{angle to goal}\|}$
Velocity maximization	$r_{vel,max} = k_{vel,max} \cdot (e^{(\frac{\mathbf{goal}}{\|\|\mathbf{goal}\|\|}) \cdot \mathbf{P_b}} - 1)$
Orientation XY	$r_{ori,xy} = k_{ori,xy} \cdot (\text{angle}(\phi_{body,x}, \phi_{world,x}))^2 + (\text{angle}(\phi_{body,y}, \phi_{world,y}))^2$
Stability	$r_{stability} = \begin{cases} -10 & \text{if } \textit{number of contact} < 2 \\ 0 & \text{otherwise} \end{cases}$

Table 4: Coefficients for the Reward Functions

Reward	**Coefficient**		
Linear Velocity	$k_v = 6$	Base Position	$k_{base,pos} = -1000$
Angular Velocity	$k_\omega = 3$	Drift	$k_{drift} = -0.1$
Airtime	$k_a = 0.01$	Distance to goal	$k_{dist} = 100$
Foot Slip	$k_{slip} = -0.5$	Angle to goal	$k_{yaw} = 2$
Foot Clearance	$k_{cl,fwd} = -300$	Velocity maximization	$k_{vel,max} = 1$
	$k_{cl,flagrun} = -80$	Orientation XY	$k_{ori,xy} = -3$
Orientation Z	$k_{ori,z} = -3$	Energy	$k_{e,fwd} = -0.2$
Base Motion	$k_{base} = -3$		$k_{e,flagrun} = -0.01$

For three different tasks, we shaped three different reward functions. Also, we implemented curriculum learning [3] on some of the penalty functions. We introduced curriculum learning

because when all the reward functions are introduced together the robot tends to ignore the velocity-based rewards and this results in a standing-still robot. We did not prefer to increase the weights of the velocity-based reward functions since we thought introducing complexity step by step would be a better solution in terms of two aspects. Firstly, when all the rewards are introduced together, the reward function might be too complex and the training might be over-constrained. This results in degraded performance. Instead, introducing the penalties gradually leads to a simpler reward function at the beginning of the training and after the penalties are introduced the locomotion is fine-tuned according to our parameters. Secondly, dynamics randomization is also introduced after a certain training step; therefore, we thought that the fine-tuning of the locomotion performance should be done with dynamics randomization to obtain a better performance.

The curriculum constant is linearly increasing until a certain point in the simulation (17).

$$c = \begin{cases} ts/threshold & ts < threshold \\ 1 & otherwise \end{cases} \tag{17}$$

Here, ts is the absolute time step in the simulation and the threshold is determined depending on the task. For CPG, learning was faster; therefore, the threshold is kept at $2 \cdot 10^5$ time steps. On the other hand, for the flag run task and forward locomotion task with cartesian PD and joint PD, the threshold is increased up to $7 \cdot 10^5$ time steps.

Forward Locomotion:

$$\begin{aligned} \mathbf{R} = r_v + r_\omega + \sum_{i=1}^{4} r_{air,i} + c \cdot r_{ori,z} + \sum_{i=1}^{4} r_{cl,i} + c \sum_{i=1}^{4} r_{slip,i} + c \cdot r_{base,m} \\ + c \cdot r_{drift} c \cdot (r_{energy}) + r_{base,p} + r_{stability} \end{aligned} \tag{18}$$

Flagrun:

$$\begin{aligned} \mathbf{R} = r_{dist} + r_{yaw} + r_{vel,max} + \sum_{i=1}^{4} r_{air,i} + c \cdot (r_{ori}) + \sum_{i=1}^{4} r_{cl,i} + c \cdot \sum_{i=1}^{4} r_{slip,i} \\ + c \cdot (r_{base,m}) - c \cdot (r_{energy}) + r_{base,p} \end{aligned} \tag{19}$$

CPG:

$$\begin{aligned} \mathbf{R} = r_v + r_\omega + \sum_{i=1}^{4} r_{air,i} + c \cdot r_{ori,z} + \sum_{i=1}^{4} r_{cl,i} + c \sum_{i=1}^{4} r_{slip,i} + c \cdot r_{base,m} \\ + c \cdot r_{drift} - c \cdot (r_{energy}) + r_{base,p} \end{aligned} \tag{20}$$

The weights of the rewards and penalties are selected according to their respective importance and magnitudes. We have tried to shape the reward function such that every component of the reward function is on the same scale; however, their relative importance creates a difference in total reward. We gave the most significance to the main task, which is velocity tracking for forward locomotion and CPG, and reaching goals for flag run.

The most essential thing we have discovered is that the energy penalty has a huge impact on how natural is the resulting gait. When the energy penalty is too low, the robot makes abrupt joint movements since energy is not a critical factor in learning. On the other hand, keeping it too high results in standing still behaviour [4]. The curriculum learning also solves this issue by slowly incorporating the high energy penalty, hence resulting in a much more natural gait.

3.2.3 Observation Space

The main thing to consider while shaping the observation space is that it should fully describe the state of the agent. We want to map the actions to the observations, hence what we include in the observation space is quite important in this aspect.

In our case, we have explored three different observation spaces for different tasks.

Forward Locomotion Task:

- **Quadruped's Motor Angles:** $\mathbb{R}^{12}[rad]$ with joint limits $[-0.2, 0.78, -2.09]$ to $[0.2, 1.25, -0.94]$.
- **Motor Velocities:** $\mathbb{R}^{12}[\text{deg}/s]$ with limits -21 to 21.
- **Base Orientation:** $\mathbb{R}^4$ with limits $[-1, -1, -1, -1]$ to $[1, 1, 1, 1]$.
- **Base Angular Velocity:** $\mathbb{R}^3[rad/s]$ with limits $[-0.052, -0.052, -0.052]$ to $[0.052, 0.052, 0.052]$.
- **Base Linear Velocity:** $\mathbb{R}^3[m/s]$ with limits $[-0.5, -0.5, -0.5]$ to $[2, 2, 2]$.
- **Contact Information:** $\mathbb{R}^4[\sim, \sim, N]$ including contact booleans, number of legs in contact, and contact forces, with limits $[0, 0, 0, 0, 0, 0, 0, 0, 0]$ to $[1, 1, 1, 1, 4, 117.72, 117.72, 117.72, 117.72]$.

Flagrun Task (in addition to Forward Locomotion OS):

- **Distance to goal:** $\mathbb{R}[m]$ with limits 0 to 1000
- **Angle to goal:** $\mathbb{R}[rad]$ with limits $-\pi$ to π

CPG (in addition to Forward Locomotion OS)

- **r:** $\mathbb{R}^4$ with limits $[0, 0, 0, 0]$ to $[3, 3, 3, 3]$
- $\boldsymbol{\theta}$ **:** $\mathbb{R}^4[deg]$ with limits $[0, 0, 0, 0]$ to $[10, 10, 10, 10]$
- $\dot{\mathbf{r}} : \mathbb{R}^4$ with limits $[-3, -3, -3, -3]$ to $[3, 3, 3, 3]$
- $\dot{\boldsymbol{\theta}}$ **:** $\mathbb{R}^4[deg/s]$ with limits $[0, 0, 0, 0]$ to $[45, 45, 45, 45]$

While giving the limits for each of these variables, our main concern was to keep the training as fast as possible by shrinking the problem space while keeping all the realistic states that the robot can enter in the observation space. Expanding the observation space is a double-edged sword, increasing it too much will give a policy mapped to a wider range of states; however, it will also increase the training time. Since most of those states are not realistic anyway, we filtered them out with limits.

While designing the observation space, in the beginning, we only considered the crucial information to learn locomotion. Therefore, we have investigated real-life applications and sensors used in legged robots to include the sensor information in the observation space [5], [6], [7]. In addition, we considered the information needed to increase the reward. Therefore, as we modify the reward function we have also modified the observation space throughout the process.

For the CPG-RL, we have observed that the robot cannot learn to move without including the CPG variables in the observation space. Also including the contact information and the ground reaction forces helped the robot to come up with a more natural walking motion. Before including the contact information, we observed that the rear left leg was always in the air during the runtime. With this term, the robot fixed this problem. Additionally, on uneven terrain, contact information and normal force information were critical to go over the obstacles.

The observation space variables for the forward locomotion, flagrun, and central pattern generator (CPG) tasks can typically be measured using various sensors such as encoders, inertial measurement units (IMUs), and vision systems. Motor angles and velocities can be obtained from joint encoders, while base orientation and linear/angular velocities can be measured using IMUs. Contact information relies on force/torque sensors on each leg. Distance and angle to goal in the flagrun task are commonly derived from vision or distance sensors. For larger distances, a global positioning system can be used. CPG variables involve proprioceptive sensors. Noise in these measurements can impact the accuracy of the robot's perception, control, and decision-making processes. Techniques such as sensor calibration, filtering, and sensor fusion are employed to mitigate the effects of noise and ensure robust and reliable operation in real-world hardware scenarios. Experimental validation and tuning on the specific hardware are essential for optimizing performance and minimizing noise-induced errors.

3.2.4 Training Environment

For our robot to recover from unexpected situations such as obstacles or low/high friction, we have included dynamics randomization after a certain time step in the training. The randomization creates blocks in front of the robot to pass in random heights in the range $(0.005, 0.04)m$. This is done to teach the robot how to recover its feet when stuck on an obstacle.

We implemented dynamics randomization by assigning random weights, ranging from 0 to 8 kg, to the robot's back at random positions. Additionally, we varied the friction within the range of 0.5 to 1 to enhance the diversity of simulated scenarios.

The randomized environment can be seen in Figure 9.

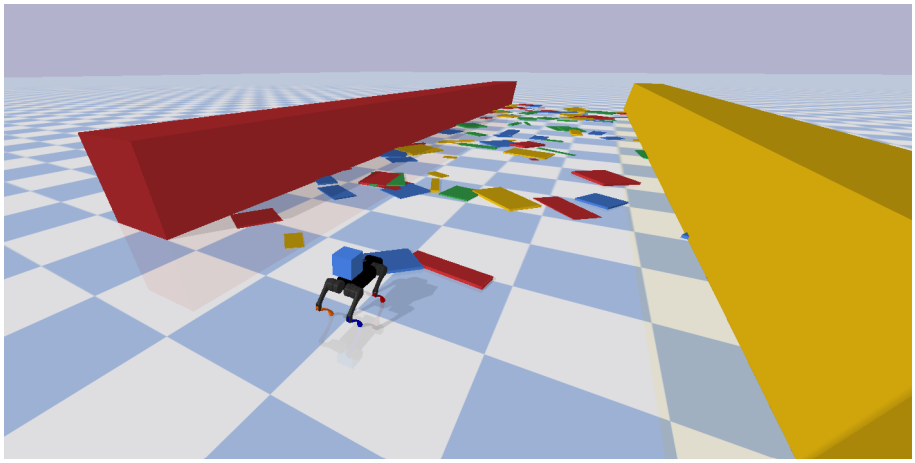


Figure 9: Environment and Dynamics Randomization

The main reason for the delay in introducing the randomized environment is that we wanted to first teach the robot how to move reliably without any disturbance. Then, the randomization is introduced to improve the robustness of the locomotion.

Introducing dynamics randomization in the training resulted in a more robust performance in the competition environment as expected.

3.2.5 Reinforcement Learning Algorithms

In this study, we have implemented Stable Baselines3 (SB3) [8] which provides a set of reliable implementations of reinforcement learning algorithms. For training algorithms, we have used both Proximal Policy Optimization (PPO) and Soft Actor-Critic (SAC).

- **Proximal Policy Optimization:** The idea behind PPO is that it improves the training stability by constraining the changes in the policy at each training epoch [9].
- **Soft Actor-Critic:** The idea behind SAC is that it is trained to maximize a trade-off between expected return and entropy, a measure of randomness in the policy [10].

In our experiments, we have implemented a fully connected artificial neural network. The number of layers and layer sizes are determined according to the following intuition [11].

Table 5: Determining the number of hidden layers in a fully connected neural network.

Number of Hidden Layers	Result
0	Only capable of representing linearly separable functions or decisions.
1	Can approximate any function that contains a continuous mapping from one finite space to another.
2	Can represent an arbitrary decision boundary to arbitrary accuracy with rational activation functions and can approximate any smooth mapping to any accuracy.

While increasing the number of hidden layers might allow you to extract more features from the data and to have a better generalization, it also increases the hyperparameters of a network. Therefore, longer training sequences might be required for only a slight increase in performance. In addition, increasing the number of hyperparameters might cause overfitting which leads to a significant decrease in performance in the test data. Even though Table 5 provides a useful insight into determining the number of hidden layers, it only describes the need while having more layers increases the ability of a network to learn and to generalize since each layer might specialize in finding another feature. In our case, we will only consider a maximum of 3 layers.

Deciding the number of neurons in a hidden layer is the other problem. While too few neurons in the hidden layers might cause underfitting, too many neurons cause overfitting of the data and a larger training sequence. We have used the following rule of thumb methods in determining the correct number of neurons in the hidden layers [11].

- The number of hidden neurons should be between the size of the input layer and the size of the output layer.
- The number of hidden neurons should be 2/3 the size of the input layer, plus the size of the output layer.
- The number of hidden neurons should be less than twice the size of the input layer.

Considering the simplicity of CPG, we have implemented a two-layer configuration. For the flag run and forward locomotion task with joint and cartesian PD, we have implemented a three-layer configuration. In the two-layer configuration, we had input and output layers with sizes of 256 neurons. In the three-layer configuration, we have also added a hidden layer of 384 neurons. The same neural network architectures are implemented for both PPO and SAC.

Lastly, we have investigated the effect of learning rate and batch size on the training performance.

- **Learning Rate:** While a smaller learning rate is guaranteed to reach a solution, it takes a longer time to reach the optima. On the other hand, increasing the learning rate might cause the gradient descent to diverge.
- **Batch Size:** Batch size defines the number of samples we use in one epoch to train a neural network. It affects the quality of the model and the training time.

We kept the learning rate at 10^{-4} while the batch size was changed between 128 for PPO and 256 for SAC.

3.3 Experiments

In the experiments, we have tested all the models that we have trained so far; however, we can only provide the final results since the observation space and the reward function changed significantly throughout the process.

For the policies throughout the experiments, even though PPO achieved a faster speed in training sequences, we have obtained better results with SAC since it does not constrain the changes in the parameters after each training step as much as PPO does. Initially, we tested our reward functions and observation spaces by training with PPO to obtain faster results. Then, after we had obtained sufficient results, we trained with SAC. This is because training with SAC converged earlier than PPO and we had more training steps to train in a dynamically randomized environment to obtain more robust results.

We have tested our locomotion performance in a competition environment and a simple environment without any disturbances and dynamics randomization. Relevant performance plots for the locomotion performance in a competition environment are provided for the best performances. Performance metrics are measured in the plain environment. Measured performance metrics are as follows.

- Average body velocity (v)
- Duty ratio (D)
- Cost of Transport (CoT)
- Time duration of one step (T_{stance}, T_{swing})
- Final base position (x)

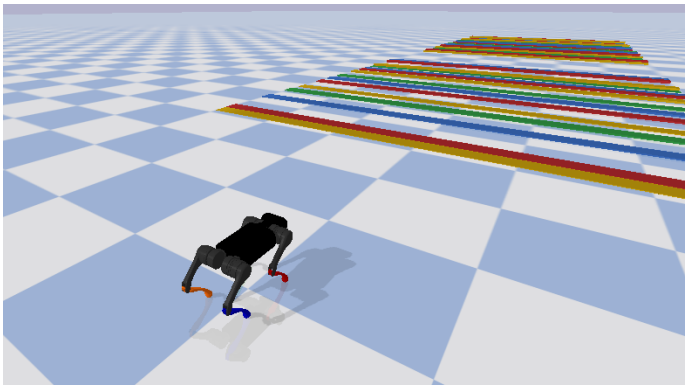


Figure 10: Competition Environment

3.4 Results

3.4.1 CPG Results

The training for CPG has emerged to a trot-like gait for a speed of around 1 m/s. The learning curves can be seen in Figure 11. Both reward and episode length increased rapidly up to 300,000 steps and converged to a reward.

The main deduction from the reward and episode length plots is that, even after dynamics randomization is introduced, mean reward and episode length are not significantly affected. This phenomenon is observed after the addition of foot contact information into the observation space. We think that it improved the robustness of the model by adding a reflex model based on contact information.

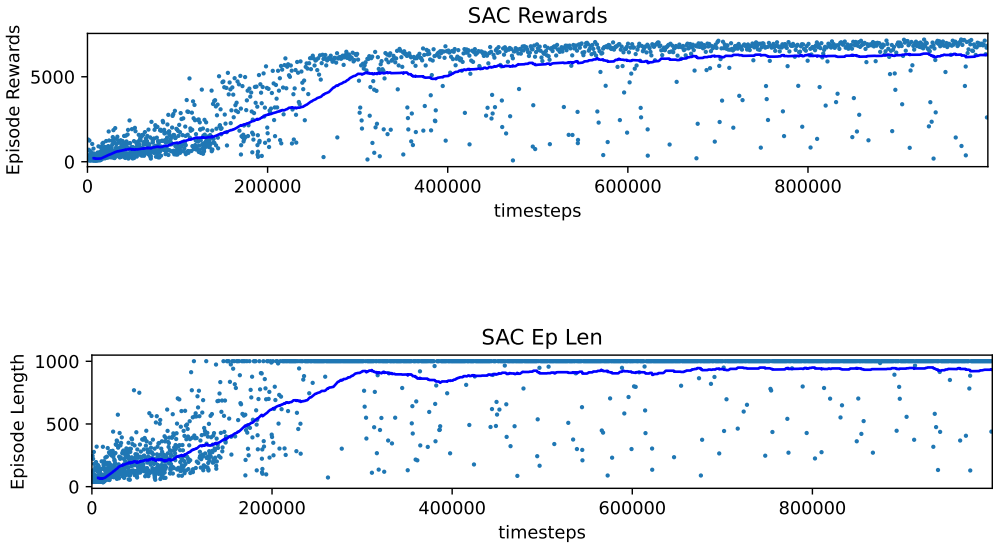
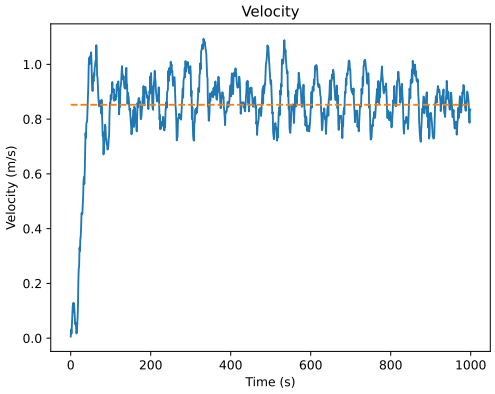


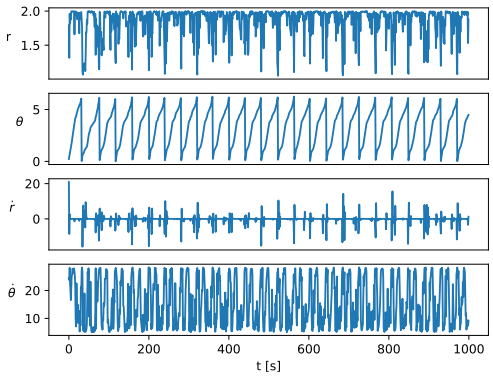
Figure 11: CPG-RL reward and episode length plots.

Figures 12a and 12b represent the body velocity and the CPG states of the front leg during

the gait. It can be seen that the robot tracks the reference velocity well enough.



(a) CPG-RL body velocity plot.



(b) CPG states of the front right leg.

3.4.2 Forward Locomotion Results

In the forward locomotion task, we have implemented both Cartesian PD and joint PD. For our Cartesian PD controller, we set the target velocity as $1.5m/s$, which is as fast as a bound gait in open loop CPG. Therefore, with suitable constraints in the reward function, this resulted in the bound as the emerging gait. The average reward and episode lengths of the Cartesian PD can be seen in Figure 13. There are drops in the reward function and episode length since after some time dynamics randomization is introduced in the training process step by step. This drop is more obvious in these plots than in the CPG plots because this is a faster gait which makes the gait more unstable.

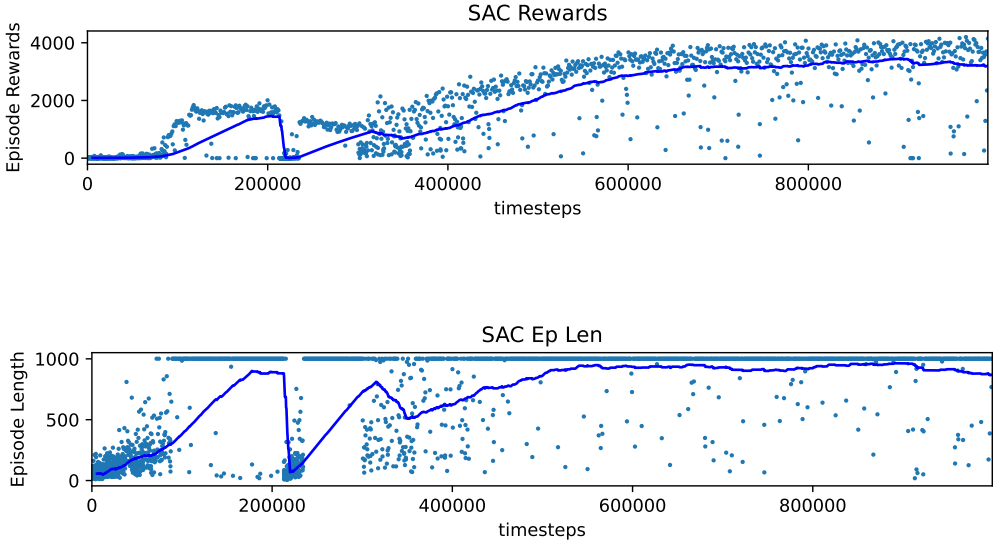


Figure 13: Average reward and episode length for cartesian PD action space.

The velocity during the locomotion can be seen in Figure 14. The tracking performance is close to excellent. Even though the plot is taken in a competition environment and there are obstacles and dynamics randomization, the robot can track the desired velocity.

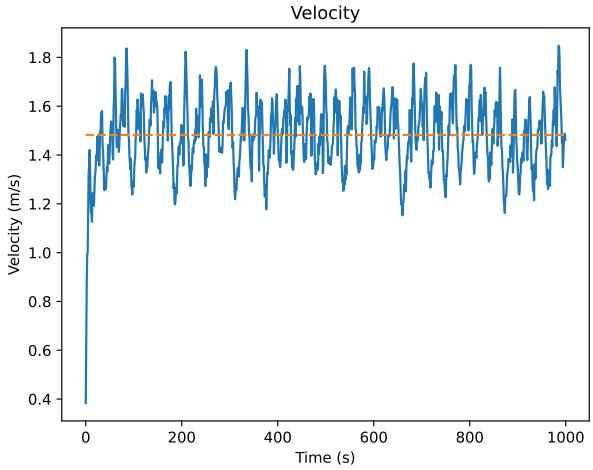


Figure 14: X velocity for cartesian PD action space.

The overall performance metrics can be seen in Table 6.

Table 6: Performance Metrics for Forward Locomotion

Action Space	Average Velocity [m/s]	D_{front}	D_{rear}	$T_{stance,front}$ [ms]	$T_{swing,front}$ [ms]	$T_{stance,rear}$ [ms]	$T_{swing,rear}$ [ms]	CoT
Cartesian	1.3	0.0007	10	0.026	37	3	0.3	0.09

The reason for such low stance times for the front legs is that during the gait, the front legs are mostly in the air, and they touch down for a very short time. We think that the motivation behind this policy is to minimize the energy by applying minimum torque to the joints by sliding on the rear legs. The training gave a more natural-looking bound gait right around the $4.8 \cdot 10^5$ steps, which can be seen in the videos. As the training progressed, the robot learned to minimize the energy even more but strayed further from the natural-looking gait.

The velocity of the gait is different when the front feet are in contact with the ground and the rear feet are in contact with the ground. These oscillations in the body velocity are also caused by both the dynamics randomization.

The foot positions of the robot during the simulation can be seen in Figure 15. It visualizes the end-effector positions in their reference frame.

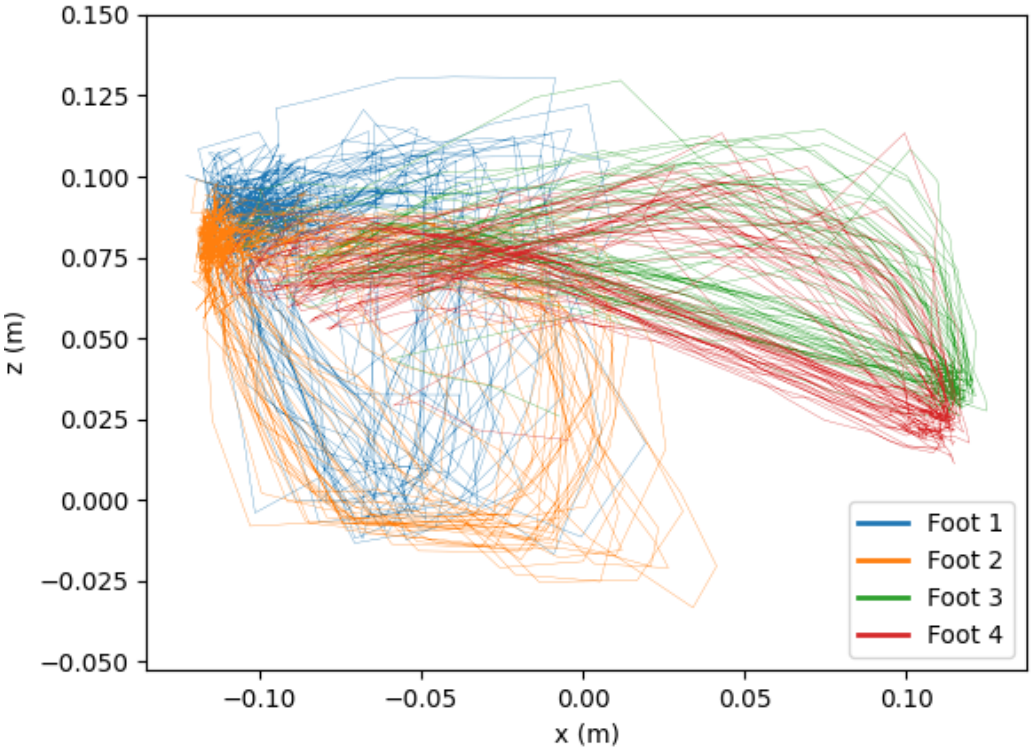


Figure 15: XZ plot of end effector positions for Cartesian PD action space.

3.4.3 Flag Run Results

For the flag run task, both joint PD and Cartesian PD results are obtained. However, the best results are obtained with a joint PD controller. One of the main difficulties that the robot encounters in the flag run task is to turn to the goal. The task is performed better with joint PD because we are directly controlling the hip, knee, and ankle joints. On the other hand, in Cartesian PD the final foot position is specified but different joint angles correspond to the same end effector position which negatively affects the learning sequence.

In addition to our flag run presentation in the competition week, we have changed the observation space a bit as discussed in the previous sections to obtain better performance in the flag run task. The average reward and episode lengths of the training can be seen in Figure 16. Due to the random placement of the goal, reward and episode length variance are significantly larger than other tasks. In addition, even though it may not be seen in the training curves after $6 \cdot 10^5$ training steps dynamics randomization is introduced, which causes mean reward and mean episode length to drop.

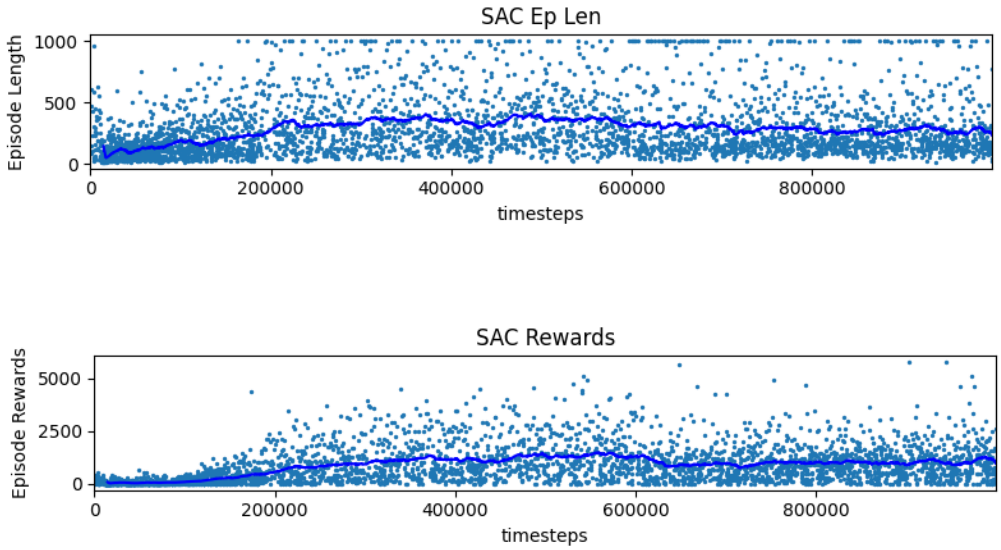
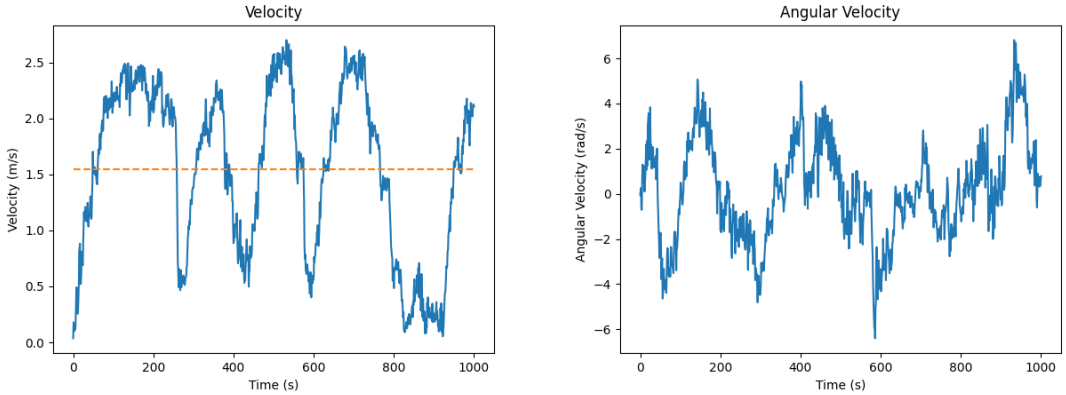


Figure 16: Mean reward and episode length for forward locomotion task.

The body velocity during the locomotion can be seen in Figure 17a.



(a) Body velocity for flag run task.

(b) Body rotational velocity for flag run task.

Since the goal is randomly placed in each episode, episode length and reward change significantly. When the goal is placed behind the robot, it is most likely to fall off while trying to turn towards the goal. Therefore, better performances are obtained in the competition environment since the goals are not placed right behind the robot in the competition environment.

3.5 Discussion

3.5.1 Enhancements for Multiple Known Consecutive Goals

In an extension scenario where multiple consecutive goals are known in advance, incorporating path planning algorithms such as Dijkstra's or A* could enhance the training and execution process

within the Markov Decision Process (MDP). This way, according to the optimal path, the desired goal will be included in the reward function. After one goal is reached, the reward function calculation is going to be reset according to the calculated path and the aim will be set for the next goal.

The MDP could be further adjusted to integrate a path optimization algorithm that pre-adjusts the robot's orientation before reaching a goal, considering the orientation of the next goal. This approach facilitates smoother transitions between consecutive goals, reduces turnaround time, enhances adaptability, and minimizes abrupt movements.

Instead of relying solely on learned policies, leveraging these algorithms could help in planning optimal paths through the known sequence of goals, presenting a structured and efficient approach. The MDP could be adjusted to incorporate the planned paths as high-level actions, facilitating a seamless integration of learned policies with path-planning outcomes. This integration ensures adherence to the known goal sequence while optimizing navigation by combining the strengths of both learned policies and classical planning algorithms. This integration ensures adherence to the known goal sequence while optimizing navigation by combining the strengths of both learned policies and classical planning algorithms.

3.5.2 Command Velocity Tracking

To traverse in a commanded velocity, the command velocity should be included in the observation space as well. Random velocity commands in a certain range should be created at the beginning of each episode, and the reward should be calculated according to the randomized command velocity. This way the robot learns to track a specified velocity. The reward term stays the same, except for the reference velocity part.

3.5.3 Effect of Action Space Preference in Learning

The choice between joint space and Cartesian space drastically depends on the complexity of the task. We think that when the trajectory of the foot positions is tightly defined in the reward function, the Cartesian space gives better results due to its natural precedence against the joint space following a hard-coded trajectory. We have observed this phenomenon with the training of the forward locomotion task. However, for more complex tasks, such as flag run, the Cartesian approach degrades performance in turning tasks, and joint space is much more preferable since the robot learns the consequences of its joint angles better when the angles are controlled directly. In the case of CPG, learning is simplified significantly since the parameters are reduced. However, it is harder to learn complex tasks with CPG since the adaptation also decreases with the reduced number of parameters.

3.5.4 Sim-to-Real Transfer

The policy we have trained for the forward locomotion task seems to be quite robust in the uneven terrain. Even with dynamics randomization and in a competition environment, in which the robot was not trained, locomotion performance is good enough. However, the generalization of the neural network is still questionable. When deploying the policy learned from the simulation on the real robot, the robot often produces unsatisfactory performance since the simulation does not convey the real-world information accurately. This may be due to some reasons, including measurement noise, actuator performance, inaccuracies in modeling the robot, and real-world dynamics.

In this aspect, we believe that our approach is not sufficient to be a bridge to this reality gap. Possible improvements for this part can be including more dynamics randomization to the robot such as [2]:

- Observation Noise
- Motor Frictions
- Foot Positions and Collision Geometry

Another good solution would be modeling the actuator as a neural network [12]. Also, our final policies allow abrupt joint movements, which might not be possible on the real robot, due to basic dynamics. We could have included an action smoothness penalty in the reward function such that the current commands are not so much different than the previous ones.

References

[1] L. Righetti and A. J. Ijspeert, "Pattern generators with sensory feedback for the control of quadruped locomotion," in *2008 IEEE International Conference on Robotics and Automation*, IEEE, 2008, pp. 819–824.

[2] G. Ji, J. Mun, H. Kim, and J. Hwangbo, "Concurrent Training of a Control Policy and a State Estimator for Dynamic and Robust Legged Locomotion," *IEEE Robotics and Automation Letters*, vol. 7, no. 2, 2022, ISSN: 23773766. DOI: 10.1109/LRA.2022.3151396.

[3] M. Aractingi, P.-A. Léziart, T. Flayols, J. Perez, T. Silander, and P. Souères, "Controlling the solo12 quadruped robot with deep reinforcement learning," *Scientific Reports*, vol. 13, no. 1, p. 11 945, 2023.

[4] J. Hwangbo, J. Lee, A. Dosovitskiy, *et al.*, "Learning agile and dynamic motor skills for legged robots," *Science Robotics*, vol. 4, no. 26, eaau5872, 2019.

[5] A. Agrawal, S. Chen, A. Rai, and K. Sreenath, "Vision-aided dynamic quadrupedal locomotion on discrete terrain using motion libraries," in *2022 International Conference on Robotics and Automation (ICRA)*, IEEE, 2022, pp. 4708–4714.

[6] G. Bledt, M. J. Powell, B. Katz, J. Di Carlo, P. M. Wensing, and S. Kim, "Mit cheetah 3: Design and control of a robust, dynamic quadruped robot," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2018, pp. 2245–2252.

[7] S. Seok, A. Wang, M. Y. Chuah, D. Otten, J. Lang, and S. Kim, "Design principles for highly efficient quadrupeds and implementation on the mit cheetah robot," in *2013 IEEE International Conference on Robotics and Automation*, IEEE, 2013, pp. 3307–3312.

[8] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dormann, "Stable-baselines3: Reliable reinforcement learning implementations," *Journal of Machine Learning Research*, vol. 22, no. 268, pp. 1–8, 2021. [Online]. Available: http://jmlr.org/papers/v22/20-1364.html.

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

[10] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," in *International conference on machine learning*, PMLR, 2018, pp. 1861–1870.

[11] J. Heaton, *Introduction to neural networks with Java*. Heaton Research, Inc., 2008.

[12] J. Hwangbo, J. Lee, A. Dosovitskiy, *et al.*, "Learning agile and dynamic motor skills for legged robots," *Science Robotics*, vol. 4, no. 26, eaau5872, 2019.