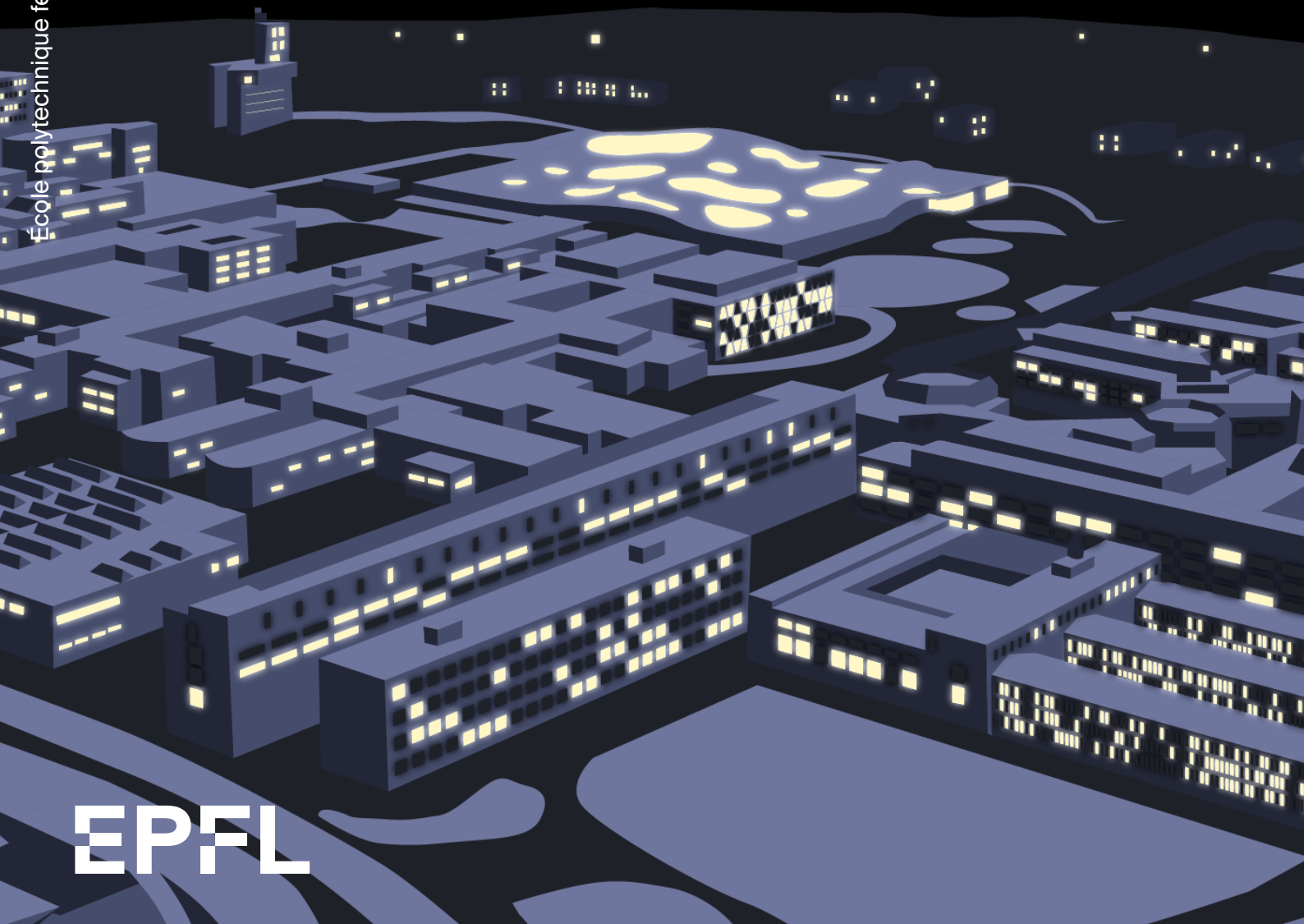
Legged Robots Project 2

Quadrupedal Locomotion with Central Pattern Generator and Reinforcement Learning

MICRO-507

Jiangfan Li, Shuhan He, Chengkun Li

École polytechnique fédérale de Lausanne



EPFL

Legged Robots Project 2

Quadrupedal Locomotion with Central Pattern Generator and Reinforcement Learning

by

Jiangfan Li, Shuhan He, Chengkun Li

Student Name	SCIPER Number
Jiangfan Li	337387
Shuhan He	322317
Chengkun Li	340485

Instructor:	Auke Jan Ijspeert
Teaching Assistant:	Guillaume Bellegarda
Institution:	École polytechnique fédérale de Lausanne

Cover Image: THE GALACTIC CHLOE SHOW
https://renezumstein.com/project/the-galactic-chloe-show

EPFL

Contents

1 Introduction

1.1. Legged Locomotion

Maintaining a stable and agile locomotion for legged robots is still a challenging problem now. The first computer-controlled quadruped is Phony Pony built by McGhee [5]. In this work, static stability is ensured (i.e. the center of mass is constrained within the supporting polygon). Another influential work is the legged robots made by Raibert [6]. Their robots achieved a fast locomotion while maintaining the dynamic stability. The BigDog robot built by Boston Dynamics was also inspired by this work [7].

Researchers with biology background also utilized the central pattern generator (CPG) combined with reflex to control the locomotion (e.g. Righetti and Ijspeert [8]). This control method is robust to perturbations in environment and reduces the dimension of the control problem.

Controlling legged locomotion with deep reinforcement learning (DRL) is one of the most promising approaches now (e.g. Bellegarda et al. [1] and Hwangbo et al. [2]). With domain randomization, control policy trained by DRL is robust to perturbations and could overcome the reality gap successfully. However, the learning method lacks theoretical proof of stability.

1.2. Quadruped Gait

In Chapter 2, we implement a CPG to generate different gaits for the quadruped robot. The gaits that our CPG could generate are trot, lateral sequence walk, diagonal sequence walk, bound, gallop, pace and pronk. The detailed definition of each gait is discussed in Chapter 2.

1.3. Quadruped Specification

The robot used in simulation is Unitree A1. The robot is roughly 300 mm in width, 620 mm in length and 250 mm in height. The robot is equipped with 12 motors, 4 for each leg. The joint torque is 33.5 N·m and the maximum joint velocity is 21 rad/s. There is a pressure sensor on each foot so the controller could get contact information of the robot.

1.4. Quadruped Modeling

The four legs of the robot are referred to as: Front Right (FR, 0), Front Left (FL, 1), Rear Right (RR, 2) and Rear Left (RL, 3) as shown in Fig. 1.1. The 12 motors are labelled as: FR hip (0), FR thigh (1), FR calf (2), FL hip (3), FL thigh (4), FL calf (5), RR hip (6), RR thigh (7), RR calf (8), RL hip (9), RL thigh (10), RL calf (11). The coordinate system of the robot is defined as below: the x-axis is aligned with the robot's heading; the y-axis points to the left and the z-axis points upward.

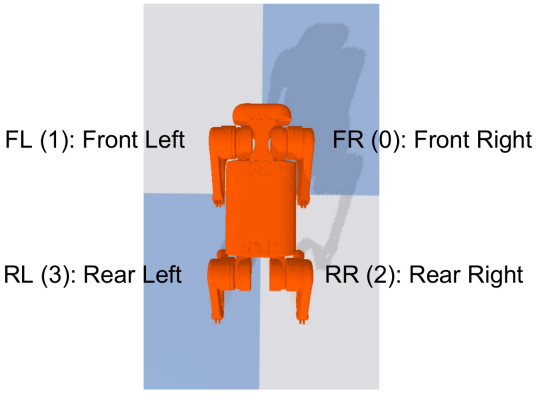


Figure 1.1: Order of legs

With respect to kinematics and dynamics of the robot, the relationship is listed (in robot coordinate system):

$$\boldsymbol{p} = f(\boldsymbol{q}) \tag{1.1}$$

$$\boldsymbol{v} = \boldsymbol{J}(\boldsymbol{q}) \cdot \dot{\boldsymbol{q}} \tag{1.2}$$

$$\boldsymbol{\tau} = \boldsymbol{J}^T(\boldsymbol{q})\,\boldsymbol{F} \tag{1.3}$$

where $\boldsymbol{v} \in \mathbb{R}^3$ and $\boldsymbol{p} \in \mathbb{R}^3$ denotes the foot velocity and position respectively; $\boldsymbol{q} \in \mathbb{R}^3$ denotes the joint angle. $\boldsymbol{J} \in \mathbb{R}^{3\times3}$ is the Jacobian of the foot position. $\boldsymbol{\tau} \in \mathbb{R}^3$ and $\boldsymbol{F} \in \mathbb{R}^3$ are torques and forces at the foot.

For control problem, both joint PD control and Cartesian PD control are used to compare their performance. The torques are computed by the following equations:

$$\boldsymbol{\tau}_{joint} = \boldsymbol{K}_{p,joint}(\boldsymbol{q}_d - \boldsymbol{q}) + \boldsymbol{K}_{d,joint}(\dot{\boldsymbol{q}}_d - \dot{\boldsymbol{q}}) \tag{1.4}$$

$$\boldsymbol{\tau}_{Cartesian} = \boldsymbol{J}^T(\boldsymbol{q})\left[\boldsymbol{K}_{p,Cartesian}(\boldsymbol{p}_d - \boldsymbol{p}) + \boldsymbol{K}_{d,Cartesian}(\boldsymbol{v}_d - \boldsymbol{v})\right] \tag{1.5}$$

1.5. Structure of This Report

The details of our CPG design and DRL configuration are contained in Chapter 2 and Chapter 3 respectively. We will discuss the results of our experiments and summarize the project in Chapter 4.

2

Central Pattern Generator

2.1. Introduction

In this chapter, we will control the robot's locomotion by Central Pattern Generator (CPG).

CPG generates periodic locomotion references according to different gaits, and the references will be mapped to physical parameters. The command will be executed by PD controllers in joint space and Cartesian space.

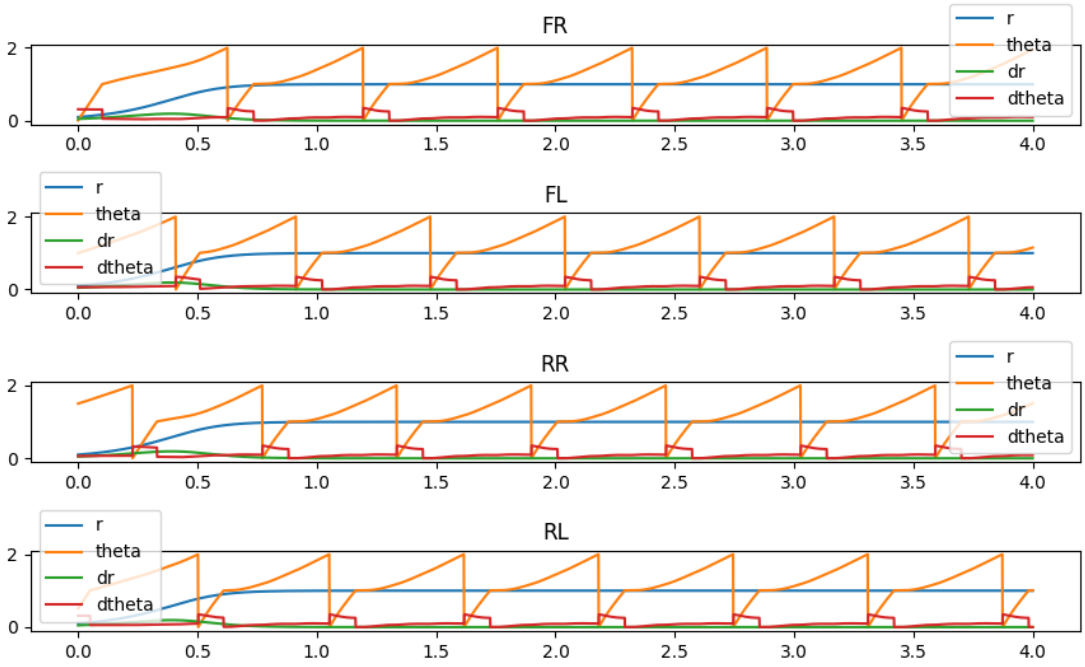


Figure 2.1: A plot of the CPG states, slow walk gait

2.2. Method

2.2.1. Hopf oscillators

We use the oscillator from the assignment instruction, that is

$$\dot{r}_i = \alpha(\mu - r_i^2)r_i \tag{2.1}$$

$$\dot{\theta}_i = \omega_i + \sum_{j=0}^{3} r_j w_{ij} sin(\theta_j - \theta_i - \phi_{ij}) \tag{2.2}$$

Equation 2.1 ensures r_i will converge to μ. Since we don't have feedback for r_i, r_i will keep the same once it converge to μ. Equation 2.2 makes θ_i rotate with desired speed(ω_i) and adjust to fit the desired latency between different θ_j, whose strength is controlled by ω_{ij}.

All the simulations below share the same $\alpha = 5.0$ and $\mu = 1.0$. α is reduced comparing to the default(50) to make the θ_is' intervals converge from the initial values. We use the same ω_{ij} for all the i, j, and assign it with a third of the minimum of ω_{stance} and ω_{swing}. It's divided by 3 in order to avoid negative $\dot{\theta}_i$.

$$\omega_{ij} = min\{\omega_{stance}, \omega_{swing}\}/3.0$$

2.2.2. Gaits and Coupling

In this chapter, gaits are characterized by their order of footfall(Φ), swing frequency(ω_{swing}) and stance frequency(ω_{stance}). Spacial differences of footfall are neglected.

We've learned some knowledge about equine gait in the class, which can be a reference for canine gaits, but the the musculoskeletal anatomy of horses is very different from that of dogs, causing canine gaits more "flexible" [9]. Assuming our robot is a robot dog, we implement the major canine gaits along with some other quadrupedal gaits on it. Their characters are listed in Table 2.1. S in the Footfall, indicates that there's a suspension(all the legs in the swing phase).

Table 2.1: Gaits definition

Gait	Footfall	Speed	Remark
Walk	RL, FL, RR, FR	Slowest	three-leg support
Diagonal Walk	FL, RL, FR, RR		
Amble	RL, FL, RR, FR		accelerated walk
Trot	(FR RL), (FL RR)		efficient, ground−covering
Pace	(FR RR), (FL RL)		center of gravity shifts
Transverse Canter	RR (RL FR) FL		Horses always use the transverse canter
Rotatory Canter	RR (RL FL) FR		Dogs preferentially use the rotary canter
Transverse Gallop	(RL,RR), (FL,FR), S		Primary gallop of the horse
Rotatory Gallop	(RR,RL), S, (FL,FR), S	Fastest	More flexible
Bound	(RR RL), S, (FL FR), S		Bunnies
Pronk	(RL FR RR FL)		Alpacas, gazelles

2.2.3. Swing-Stance Phases and frequencies

Swing phase is defined as $0 \leq \theta_i \leq \pi$, and stance phase as $\pi < \theta_i \leq 2\pi$. Swing phase and stance phase have different rotational velocity, ω_{stance} and ω_{swing}. Duty factory can be calculated by 2.3. Note that this is an approximate value as coupling will change the ω during phases to synchronize all the legs.

$$D = \frac{T_{stance}}{T_{stance} + T_{swing}} = \frac{1}{1 + T_{swing}/T_{stance}} = \frac{1}{1 + \omega_{stance}/\omega_{swing}} \tag{2.3}$$

We can use the features, the number of supporting legs and the speed to help choose a gait's ω_{stance} and ω_{swing}. For example, in the normal walk gait, two legs alternate with three legs in supporting body,

which means the duty factor for each leg should be in range $(1/2, 3/4)$. Therefore, $\omega_{stance}/\omega_{swing}$ should be in range $(1/3, 1)$. In the slow walk gait, where occurs the 4-leg supporting, $\omega_{stance}/\omega_{swing}$ should be less than $1/3$. This calculation provide reference for later parameters tuning.

2.2.4. Mapping from CPG to Physical Parameters

In our project, we map the state to Cartesian foot positions in the leg xz plane as suggested in instruction.

$$\begin{aligned} x_{foot} &= -d_{step} \cdot r_i \cdot cos(\theta_i) \\ z_{foot} &= \begin{cases} -h + g_c \cdot sin(\theta_i) & if\ sin(\theta_i) > 0 \\ -h + g_p \cdot sin(\theta_i) & otherwise \end{cases} \end{aligned} \tag{2.4}$$

To achieve omnidirectional locomotion, we remap x_{foot} to x_{foot} and y_{foot} as shown in 2.5, where β stands for the desired direction(counter-clockwise, starting from x-axis) and $ydisplacement \cdot sideSign_i$ stands for desired width of legs stretched out.

$$\begin{aligned} x'_{foot} &= x_{foot} \cdot cos(\beta) \\ y'_{foot} &= x_{foot} \cdot sin(\beta) + ydisplacement \cdot sideSign_i \end{aligned} \tag{2.5}$$

2.2.5. PD Controllers

We use both joint PD and Cartesian to have a better performance. The parameters of PD controllers keep same as the default. That is,

$$\begin{aligned} k_{p,joint} &= [150, 70, 70] \\ k_{d,joint} &= [2, 0.5, 0.5] \\ k_{p,Cartesian} &= diag(2500, 2500, 2500) \\ k_{p,Cartesian} &= diag(40, 40, 40) \end{aligned}$$

With this setting, Cartesian PD controller dominates, consistent with our control parameters – positions of feet.

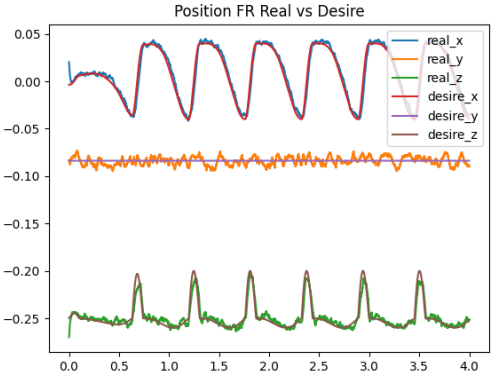
Foot position

Fig 2.2 shows the position desired and archived on front right leg with gait slow walk. Apparently, using both PD controllers produce the best track curve. Using only Cartesian PD is the worst. As far as we concerned, two factors may account for this poor performance. 1) Jacobian matrix is local, so that the calculation results may not accurate enough for the whole time step. 2) Cartesian PD uses desired position and linear velocity to calculate the supposed torque, without calculating joint positions directly.

Compare to joint PD only, result curve with two PD controllers is smoother, and especially, desire_z doesn't reach the peak as curve with joint PD does, as if a low-band filter has been applied.

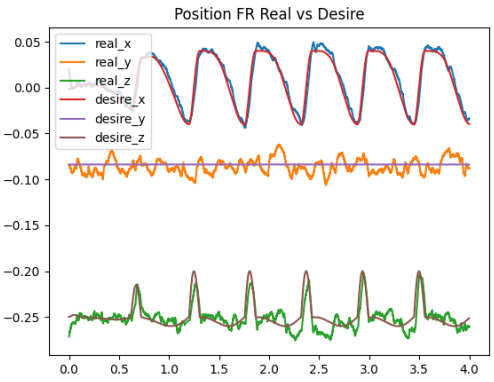
Joint angles

Same as the position tracking, shown in Fig 2.3, using both PD controllers performs better than using only one controller. And Cartesian PD controller, once again, is the worst.



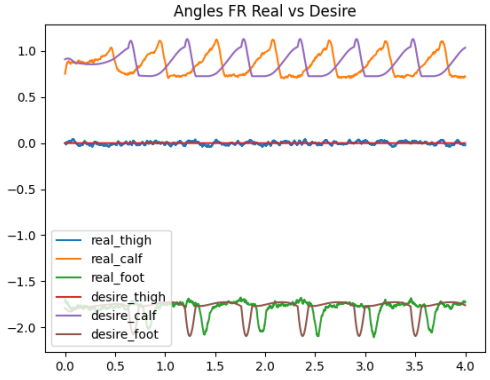
(a) with joint PD and Cartesian PD

(b) with joint PD



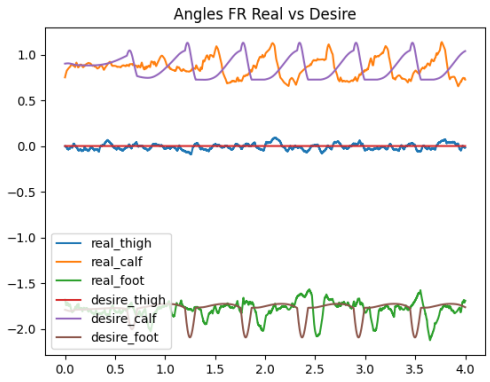
(c) with Cartesian PD

Figure 2.2: desired foot position vs. actual foot position, FR



(a) with joint PD and Cartesian PD

(b) with joint PD



(c) with Cartesian PD

Figure 2.3: desired joint angles vs. actual joint angles, FR

2.3. Discussion

2.3.1. Gaits

As shown in Table 2.1, we implement several major canine gaits, namely: walk, amble, trot, pace, canter(transverse and rotatory) and gallop(transverse and rotatory), and some other quadrupedal gaits including diagonal walk, bound and pronk. Though there are some spacial differences of gaits(e.g. one

leg will lead a bit in gallop gait), we neglect them in our CPG control structure.

Horses always use transverse gaits while most dogs prefer to use rotatory ones, possibly because that dogs' body are more flexible than horses'. Rotatory gaits require more subtle tuning to keep balance.

Parameters are pre-tuned for the original 4 gaits, but for newly added gaits, parameters still need to be tuned. The parameters used in our project are shown in Table 2.3, other than mentioned, other parameters are kept same as the default shown in Table 2.2, where y displacement stands for the distance of foot away from the shoulder on y-axis, i.e. `foot_y`. And we add a new parameter, x displacement. Similar to y displacement, it represent the base bias of foot position on x axis. The choosing of these parameters will be discussed in the following section 2.3.2.

As mentioned in Section 2.2.4, omnidirectional locomotion has been archived by remapping x to x-y plane.

Table 2.2: Default gait parameters

Parameter	value(m)
step length	0.04
ground penetration	0.01
ground clearance	0.05
robot height	0.25
x displacement	0.0
y displacement	0.0838

Table 2.3: Gaits parameters

Gait	Footfall Sequence	ω_{swing} $(\cdot 2\pi)$	ω_{stance} $(\cdot 2\pi)$	others
Slow Walk	0, 0.5, 0.75, 0.25	5	1	
Normal Walk	0, 0.5, 0.75, 0.25	5	3	
Amble	0, 0.5, 0.75, 0.25	10	15	
Diagonal Walk	0, 0.5, 0.25, 0.75	5	3	
Trot running	0.5, 0, 0, 0.5	9	10	
Trot walking	0.5, 0, 0, 0.5	2.2	2	
Pace	0.5, 0, 0.5, 0	8	6	step length = 0.05, ground clearance = 0.06, y displacement = 0.1, robot height = 0.2
Pace fly	0.5, 0, 0.5, 0	7	20	step length = 0.05, y displacement = 0.126, robot height = 0.2
Transverse Canter	0.5, 0.75, 0.25, 0.5	6	13	step length = 0.07, ground penetration = 0.005, ground clearance = 0.03, y displacement = 0.1, robot height = 0.2
Rotatory Canter	0.75, 0.5, 0.25, 0.5	6	13	step length = 0.07, ground penetration = 0.005, ground clearance = 0.06, y displacement = 0.1, robot height = 0.2
Transverse Gallop	0.45, 0.4, 0.05, 0	12	20	step length = 0.05, ground penetration = 0.02, y displacement = 0.1, robot height = 0.225
Rotatory Gallop	0.55, 0.5, 0, 0.05	9.5	20	step length = 0.05, ground penetration = 0.02, ground clearance = 0.06, robot height = 0.23, y displacement = 0.1
Bound	0.5, 0.5, 0, 0	5.5	22	step length = 0.065, ground penetration = 0.02, ground clearance = 0.07, robot height = 0.2, x displacement = 0.04, y displacement = 0.1
Pronk	0, 0, 0, 0	11	25	step length = 0.065, ground penetration = 0.022, ground clearance = 0.07, robot height = 0.2, x displacement = 0.02, y displacement = 0.1

2.3.2. Parameters

Hyper-parameters

The original gaits seems has relatively fixed definition, so their Φ matrices are not tuned. As for the newly added gaits, almost no effort is made to tune comparing to that of ω_{stance} and ω_{swing}.

According to observation, with higher rotatory frequencies, robot will move more stable. But with too high frequencies, robot may slip. A stance speed around $20 \cdot 2\pi$ with reasonable swing speed often produce relative high frequency locomotion of good performance, like trot and amble.

Slow walk gait is used to achieve the slowest speed. Using the parameters listed in Table 2.3, the performance of a trial of 10s is shown in Table 2.4.

Table 2.4: Slowest Gait Performance

average (last half) x-y speed	0.131m/s
duty factor	80.7%
swing duration	0.455s
stance duration	0.109s
Energy Cost	3012J
Distance Traveled (in x-y plane)	0.60m
Cost of Transport	41.1
$\lvert\bar{pitch}\rvert$	0.02

The fastest gait, according to observation of real dogs, is rotatory gallop. However, in general, gaits with less symmetry is harder to tune. As a rotatory gait with suspension, it's hard to tune the parameters for rotatory gallop to make sure that the robot will not fall down, impeding to find a parameter set that it runs quite fast.

The fastest speed we find is 3.5m/s by pronk gait using the parameters listed in Table 2.3. And the performance are listed in Table 2.5. The locomotion looks a bit of unstable (You can find the attached video). It is because that the parameters are tuned for the fastest speed, tune down the step length will gain a more stable locomotion.

Table 2.5: Fastest Gait Performance

average (last half) x-y speed	3.50m/s
duty factor	30.3%
swing duration	0.046s
stance duration	0.02s
Energy Cost	4936J
Distance Traveled (in x-y plane)	30.7m
Cost of Transport	1.32
$\lvert\bar{pitch}\rvert$	0.14

CPG parameters

Default parameters works fine for 4 original gaits once you find a good omega pair. To make other gaits work, to have a better balance and to run faster, more tuning work needs to be done.

For gaits with suspension periods, it's not easy to keep balance in air. Mismatched omega pair and inappropriate pose to land off pose great threat to balance.

Suspension time varies due to tiny disturbance. Omega pair and other parameters need subtle tuning to have a robust accordance, guaranteeing legs are in stance phase when they are about to land off.

As shown in Fig 2.4, if the pitch angle is negative (i.e. gravity center is in the rear), fore legs may not manage to offer any support when they are in stance phase. If the pitch angle is large, it's somehow equivalent to have a large ground penetration which we don't want. To alleviate this problem, taking bound gait as an example, we add a x displacement. When robot is bounding, the propulsion is provided

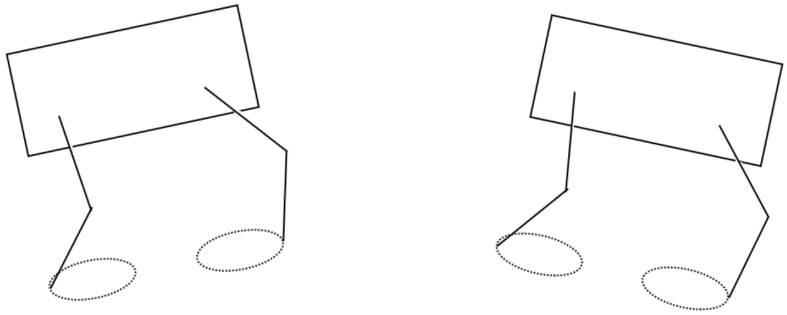


Figure 2.4: Influence of different pitch when landing off

mostly by rear legs. Once the robot jumps up (towards right) by rear legs, the body will rotate clockwise, ultimately resulting in big pitch angle. With a x displacement, the gravity center is relative behind the support polygon, and less rotation will be made after rear legs' exertion.

Here are some summaries about the parameters we have tuned.

- **coupling strength**
  If the CPG network is open-loop, coupling strength doesn't matter a lot once θ_i has converged. Without coupling, the phase offset will keep as the initial value. Mentioned in section 2.2.1, we choose coupling strength according to the rotatory speed of θ_i.
- $\boldsymbol{\alpha}$
  We set a fix α of 5.0, to give a bit more time for the controller to "warm up", avoiding apply too much force when θ_is haven't converged.
- **ground clearance**
  The bigger ground clearance, the higher feet will raise. High ground clearance will consume more energy. Also, it may cause some position with same x value unreachable (no inverse kinematic resolution). Low ground clearance may result in a reverse locomotion because feet rub against ground when it's supposed to swing. It should be selected depend on gaits and speed.
- **ground penetration**
  The bigger ground penetration, the more feet will hit against the ground. High ground penetration leads to stronger upward component, and the robot will bound higher. Low ground penetration may result in slippery because that, not much support forces are applied to the stance legs, therefore, the friction coefficient is too small and forces to push the robot forward are weak.
- **step length**
  Apparently, the bigger step length, the faster the robot will go, under same conditions. However, it's not possible to enlarge the step length without any limit. Generally, big step length will threat the balance and itself may not reachable. Small step length can't provide enough propulsion. The robot will walk like moonwalk.
- **robot height**
  Intuitively, the lower robot height, more stable the locomotion will be for letting down the gravity center. But there will be less operational space for feet.
- **y displacement**
  Similar to robot height, the more legs stretched out, more stable the locomotion will be for enlarging the support polygon. And it will also limit the operational space if the feet are stretched out too much.

- **x displacement**
  As discussed before, x displacement can adjust the gravity center while moving.

2.3.3. Feedback and descending control

Many problems about keeping balance has shown before. If there are some reflexes to report the center controller about the robot's status, it may be able to adjust parameters automatically.

Fig 2.5 shows a descending control diagram we proposed. Central controller will generate base CPG parameters like Φ, ω_{stance}, ω_{swing} and so on, according to the commanded gait, speed, heading and other requirements. CPG will generate locomotion pattern, and adjust according to the feedback and couple among the CPGs.

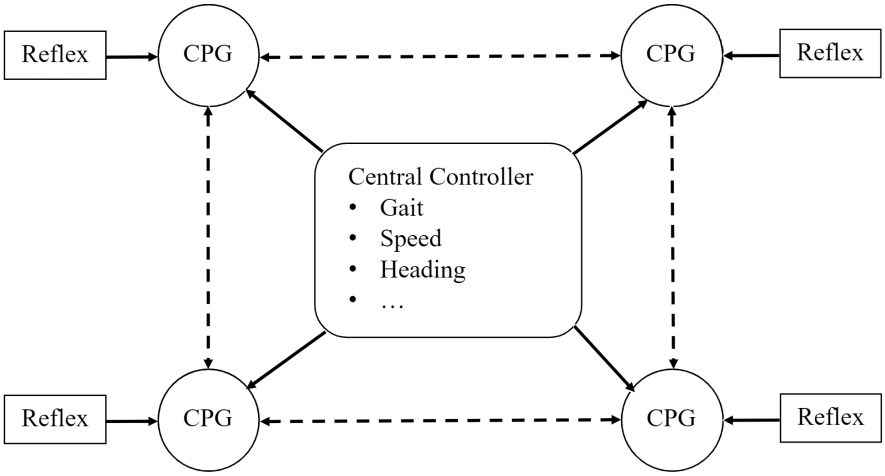


Figure 2.5: Descending control diagram

Force Feedback

We add a force feedback described in [8]. Specifically, according to force information and CPG status, we may speed up transition between phases or stop it.

While the i^{th} leg is swinging, if it contact the ground, then we increase the ω to speed up the transition. Likewise, while the i^{th} leg is in stance phase, if its support force is quite small, then it's time to swing.

When the i^{th} leg is about to stop swinging, if it hasn't contact the ground, then it should freeze and wait until the foot contact the ground. Likewise, when the i^{th} leg is about to begin swinging, if it still share great proportion of supporting the body, then it should wait until its absence will not affect the balance much.

The implementation codes are shown as following. We use a scalar factor instead of a augmentation of a fix value while speeding up, in order to reduce feedback's leading role.

```
suppose_theta = theta + theta_dot * self._dt
if theta < np.pi: # SWING
  if suppose_theta > np.pi and not contactBool[i]:
    theta_dot = 0
  elif contactBool[i]:
    theta_dot *= 2.0
else:           # STANCE
  if suppose_theta > 2*np.pi and (forceNormal[i] > force_threshold):
    theta_dot = 0
  elif forceNormal[i] < force_threshold:
    theta_dot *= 2.0
```

The simulation result shows that, force feedback is dominating. Almost whatever gait you choose, all the legs will act like contact indicators – they stretched forward until they hit the ground, then swing back and start another cycle.

2.4. Conclusion

CPG provides a convenient and explainable approach to convert the gait to specific physical parameters, and has shown its expansibility through this chapter. On the other hand, I have to say, it's quite hard to tune the parameters for some gaits. Maybe some optimization algorithms can be applied to find a good parameter set.

3

Reinforcement Learning

3.1. Overview

In this chapter we discuss the locomotion of the quadruped via reinforcement learning. We first present our reward functions design and discuss their compositions, then we present the training setup of

3.2. Reward Function Design

In this section we introduce two of our designed reward functions where we denote them as type I and type II.

3.2.1. Type I Reward Function

Type I reward function is fairly simple and was modified base on the one in [1] and is shown in Eq. (3.1), the only modification we made is that we added an additional reward term to constrain the quadruped's displacement on the direction perpendicular to the moving direction.

$$\begin{aligned} R\left(s_t, a_t, s_{t+1}\right) = & w_1 \min\left(x_{b,t+1} - x_{b,t}, d_{\max}\right) \\ & - w_2 \int_t^{t+1} |\boldsymbol{\tau} \cdot \dot{\boldsymbol{q}}| dt \\ & - w_3 \left(y_{b,t+1} - y_{b,t}\right) \\ & + 0.01 \end{aligned} \tag{3.1}$$

Type I reward function consists of the distance reward, energy penalty, displacement penalty, and episode length reward. The coefficients for each term indicate their relative importance in the locomotion performance, here we set ω_1 as 2, ω_3 as 0.02 and we tweak ω_2 to test the locomotion performance under different combinations which will be shown in the Experiment section.

3.2.2. Type II Reward Function

Type II reward function is based on the work of Lee et al. [3]. We make two major changes in the reward function: first we incorporate the command velocity into the linear velocity reward, hoping to get an omni-directional locomotion policy; second we add a foot height term in the reward to penalize overly small ground clearance, in order to improve the ability to traverse rugged terrains. The reward function is shown in Eq. (3.2).

$$
\begin{aligned}
R\left(s_t, a_t\right) = & w_1 \exp\left(-2\left(\overline{\boldsymbol{v}}_{b,t} \cdot \overline{\boldsymbol{v}}_{cmd} - 1\right)^2\right) + \\
& w_2 \exp\left(-1.5\left\|\overline{\boldsymbol{v}}_{b,t} - \left(\overline{\boldsymbol{v}}_{b,t} \cdot \overline{\boldsymbol{v}}_{cmd}\right)\overline{\boldsymbol{v}}_{cmd}\right\|^2\right) + \\
& w_3 \exp\left(-1.5\left(\omega_{b,t,roll}^2 + \omega_{b,t,pitch}^2\right)\right) + \\
& w_4 \exp\left(-1.0\left\|\boldsymbol{quat}_{cmd} - \boldsymbol{quat}_{b,t}\right\|^2\right) + \\
& - w_5 \left|\boldsymbol{\tau_t} \cdot \dot{\boldsymbol{q}}_t\right| dt + \\
& w_6 \left(1 - \exp\left(-0.1\left\|\boldsymbol{v}_{b,t}\right\|^2\right)\right) + \\
& w_7 \left(1 - \exp\left(-2.0\left(\boldsymbol{z}_{foot,t} \cdot \boldsymbol{I}_{swing,t}\right) / \left(\mathbf{1} \cdot \boldsymbol{I}_{swing,t}\right)\right)\right)
\end{aligned}
\tag{3.2}
$$

The $\boldsymbol{v}_{b,t}$ denotes base linear velocity and $\overline{\boldsymbol{v}}_{b,t}$ is the normalized velocity. $\overline{\boldsymbol{v}}_{cmd}$ is the command velocity. $\omega_{b,t,roll}$ and $\omega_{b,t,pitch}$ represents base angular velocity. The command orientation, which is aligned with command velocity, is denoted by $\boldsymbol{quat}_{cmd}$ in quaternion form. $\boldsymbol{z}_{foot,t}$ is a vector of feet height and $\boldsymbol{I}_{swing,t}$ is an indicator vector, whose element is 1 if the leg is in swing phase. $\mathbf{1}$ is just an all-ones vector. In summary, the first two terms make sure the actual base velocity is close to the command velocity. The third term penalize aggressive angular velocity of the base. The fourth term pushes the orientation of the base to the orientation of command velocity. The fifth term minimizes the energy. The sixth term reward high speed and the last term reward average swing foot height.

With respect to balancing weights in reward function, we first keep the weight of energy term far less than other terms. We believe if the weight of energy term is too large, the robot would keep feet height low to minimize energy, which would decrease the ability to go over obstacles. Other terms that should be kept small could be the third and fourth term in reward function Eq. (3.2). Actually the performance of the robot with this reward function is even worse than the first one in some cases. We think the reason could be that the penalty on base angular velocity and orientation is too large, which constrains the locomotion of the robot.

3.3. Observation Space Design

When designing the observation inputs, we referred to several papers: [1, 4]. In our controller, the observation space (in $\mathbb{R}^{78}$) consists of:

- The linear velocity of the quadruped's base in $\mathbb{R}^3$.
- The angular velocity of the quadruped's base in $\mathbb{R}^3$.
- The motor angles of the quadruped in $\mathbb{R}^{12}$.
- The motor velocities of the quadruped in $\mathbb{R}^{12}$.
- The motor torques of the quadruped in $\mathbb{R}^{12}$
- The orientation of quadurped's base in $\mathbb{R}^4$.
- The foot positions and velocities each in $\mathbb{R}^{12}$.
- The contact booleans of the quadruped foots in $\mathbb{R}^4$.
- The normal contact foot forces on the normal direction in $\mathbb{R}^4$.

The observation space is normalized before feeding into the policy network, the reason is that different observations have different scales, so it's better to scale all the observations to the same scaling before giving them to the network to learn the mapping. Otherwise, the network might only learn the policy based on part of the observations. For instance, the maximum normal contact force could be over 1000 newtons while the maximum linear base velocity should be below 5 meters per second. Normalization is necessary to ensure different terms have roughly the same influence on the policy network. Additionally, normalizing the data can also increase the convergence speed of the network.

It is noteworthy that the performance improved significantly after the contact booleans and normal contact forces added. (Actually we could only have normal contact forces.) In order to maintain a stable gait and/or traverse rugged terrains, the phase (i.e. swing phase or stance phase) of each leg is an essential part in the policy network input. For instance, the robot's ability to traverse rugged terrains increased after the contact information is added. This is a quite intuitive result since the robot should know which legs are in stance in order to plan the locomotion to go over obstacles.

Additionally, in order to addressing the problem of omni-directional locomotion, we add an optional command velocity in observation space, which is a vector of $\mathbb{R}^3$. Thus the final observation space is in $\mathbb{R}^{81}$.

3.4. Action Space Design

In our locomotion controller training, we have trained policies based on both controlling schemes, Cartesian PD and Joint PD. We skip their introduction in this part since we have mentioned their properties in Section 1.4.

From [1] we learned that the range for action is $[-0.2, 0.2]$ in x direction, $[-0.05, 0.05]$ in y direction, and $[-0.33, -0.15]$ in z direction. As we chose the upper and lower boundary of action output from the policy network as 1 and -1, so we set the scale factor as $[0.1, 0.05, 0.08]$ to satisfy the range limit of the quadruped's action.

In the experiment, we test both Cartesian PD controller as well as Joint PD controller. And a more detailed discussion will be presented in the Experiment section.

3.5. Training Environment

During the training phase, in order to make the quadruped adapts better in various environments, we alter the training environments in terms of several physical properties as shown in Table 3.1.

Table 3.1: Environment Randomization

Property	Description	Randomization range
μ_{ground}	Ground friction coefficient, to simulate different terrain textures.	$[0.5, 1]$
g_x	Gravitational acceleration on x direction	$[-0.01, 0.01]$
g_y	Gravitational acceleration on y direction	$[-0.01, 0.01]$
g_z	Gravitational acceleration on z direction	$[-9.8, 11.76]$
Obstacles	Add random boxes with variable height as obstacles	height $\in [1, 1.2]$
σ	Standard deviation of zero mean observation noise	0.02

A glimpse of our training environment is shown in Fig. 3.1. The purpose of introducing variance on gravitational acceleration in different directions is to simulate different terrains and slopes while at the same time simulating different carrying loads. Also, we notice that by introducing random boxes with variable height to the training environment, it prompts the agent to learn a variety of gaits that are more adaptive to the environment.

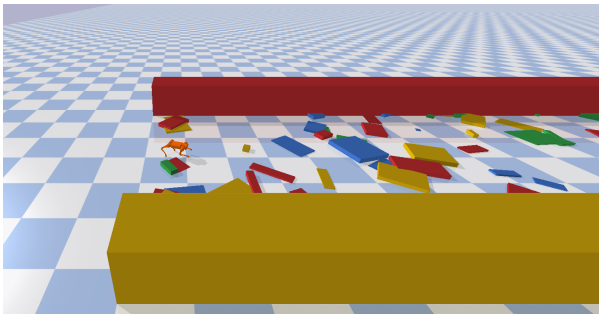


Figure 3.1: Training environment

3.6. Policy Network

As for training Algorithm, we use both Proximal Policy Optimization (PPO) and Soft Actor Critic (SAC) in our experiments. And we use a two hidden layer Multi Layer Perceptron (MLP) with each layer of size 256 as our policy network for both PPO and SAC.

3.6.1. Proximal Policy Optimization

PPO is a policy gradient based training algorithm, which combines the idea A2C and TRPO. The objective function of PPO is shown in Eq. (3.3).

$$L^{CLIP}(\theta) = \hat{E}_t \left[\min \left(r_t(\theta)\hat{A}_t, \text{clip}\left(r_t(\theta), 1-\varepsilon, 1+\varepsilon\right)\hat{A}_t \right) \right] \tag{3.3}$$

where

θ = parameterized policy.
$\hat{E}_t$ = empirical expectation over timesteps.
r_t = ratio of probability of new policy to probability of old policy.
$\hat{A}_t$ = Estimated advantage at time t.
ε = Clip parameter, usually 0.1 or 0.2

We use the implementation of PPO from `stable-baselines 3` with the selected hyperparameters shown in Table 3.3.

3.6.2. Soft Actor-Critic

Soft Actor-Critic (SAC) is a off-policy with the central feature entropy regularization. The policy is trained to maximize the trade-off between expected return and entropy, measuring its randomness. In SAC, the learning takes place to learn the policy π_θ as well as two value functions.

3.7. Experiments

In this section, we present the experiment design to test our quadrupedal locomotion.

3.7.1. Test Environment

In terms of testing environment, we use 3 environments to test our policies, namely, the competition environment with two blocks of evenly spaced obstacles (as shown in Fig. 3.2c), the testing environment with randomly spaced obstacles and boxes locating on the quadruped (as shown in Fig. 3.2a), and the plain environment without any perturbations (as shown in Fig. 3.2b).

Table 3.2: Hyperparameters of PPO in our implementaion

Hyperparameter	Description	Value
n_steps	The number of steps to run for each environment per update. (We set num_env to 4 in most of our trainings)	4096/num_env
batch_size	Mini batch size.	128
ent_conf	Entropy coefficient for the loss calculation.	0
gamma (γ)	Discount factor.	0.99
learning_rate	The learning rate of Policy gradient based PPO.	Linear scheduler decay in the rate of $1/(\text{total\_timesteps})$ starting from 1×10^{-4}
vf_coef	Value function coefficient for the loss calculation.	0.5
max_grad_norm	The maximum value for gradient clipping.	0.5
gae_lambda	Factor for trade-off bias vs variance for Generalized Advantage Estimator	0.95
n_epochs	Number of epochs when optimizing the surrogate loss	10
clip_range	Clipping parameter	0.2
clip_range_vf	Clipping parameter for value function.	1

Table 3.3: Hyperparameters of SAC in our implementation

Hyperparameter	Description	Value
learning_rate	The learning rate for Q values, Actors and Value function.	Linear scheduler decay in the rate of $1/(\text{total\_timesteps})$ starting from 1×10^{-4}
buffer_size	Size of replay buffer.	300000
batch_size	Mini batch size.	128
ent_conf	Entropy coefficient for the loss calculation.	learned automatically
gamma (γ)	Discount factor.	0.99
tau (τ)	soft update coefficient.	0.05
train_freq	Number of experience to collect by doing rollouts of current policy.	1
gradient_step	Number of gradient step to do after each rollout.	0.5
learning_starts	How many steps of the model to collect transitions before learning starts	0.95

3.7.2. Metrics

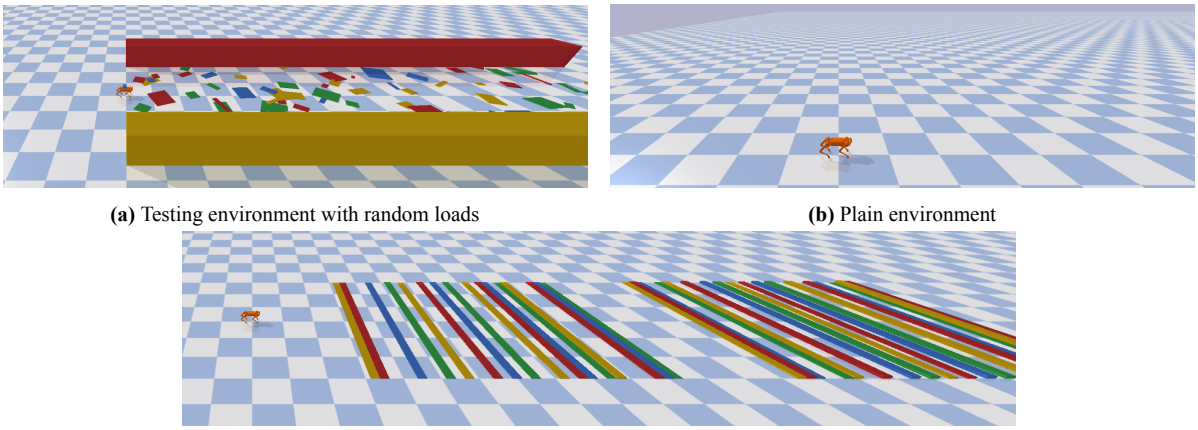
In the experiments, we test most of our Quadruped locomotion controllers in the competiton environment because we notice that if the quadruped could get through the competition environment, it mostly likely will also get through the other two environments. As for duty cycle calculation, we place the quadruped in the plain environment to get undisturbed results. We chose several metrics for evaluation of our learned locomotions:

1. Average speed ($\bar{v}$) of all testing trails[1] in competition environment.
2. Average Cost of Transport ($\bar{COT}$). This is a measure for the energy efficiency of our locomotion in challenging environment, where each COT is calculated as

$$\text{COT} = \frac{\text{Total energy consumed by the joints}}{\text{Total distance traveled} \times m \times g}$$

3. Average foot height $\bar{z_{foot}}$ of all testing trials.

[1] In testing we set total timesteps to 5000, so there should have at least 5 trials.



(a) Testing environment with random loads

(b) Plain environment

(c) Competition environment

Figure 3.2: Three testing environments used in the experiments

4. Average final base position of all testing trials $\bar{x_{base}}$ in competition environment (shown in Fig. 3.2c). Which evaluates the ability of quadruped to traverse various terrains.
5. Duty factor of each foot in plain environment.

3.8. Results

The results of experiments with Type I reward function (as mentioned in Section 3.2.1) are shown in Table 3.4.

Table 3.4: Experiment results for Type I reward function; **C** denotes **C**artesian and **J** denotes **J**oint

Algorithm	Dynamics randomization	Energy weight	PD type	$\bar{v}$ (m/s)	$\bar{z_{foot}}$ (FR, FL, RR, RL)	$\bar{x_{base}}$	$\bar{COT}$	Duty factor (FR, FL, RR, RL)
PPO	✓	0.001	C	1.2	0.02, 0.02, 0.04, 0.04	6.3	0.19	0.357, 0.394, 0.487, 0.452
PPO	✗	0.0	C	2.9	0.05, 0.04, 0.03, 0.04	6.8	0.11	0.309, 0.325, 0.418, 0.371
PPO	✓	0.0	J	1.6	0.05, 0.04, 0.06, 0.03	8.3	0.10	0.314, 0.307, 0.172, 0.282
PPO	✓	0.004	C	2.3	0.04, 0.03, 0.07, 0.07	10.1	0.058	0.335, 0.263, 0.455, 0.440
SAC	✓	0.0	J	2.5	0.07, 0.08, 0.05, 0.04	**24.5**	-0.004	0.314, 0.266, 0.152, 0.169
SAC	✓	0.008	C	1.2	0.03, 0.05, 0.04, 0.06	8.6	0.092	0.26,0.14,0.47,0.565
SAC	✓	0.008	J	2.0	0.05, 0.07, 0.06, 0.05	14.2	-0.07	0.03, 0.01, 0.07, 0.05
SAC	✗	0.008	C	2.4	0.03, 0.04, 0.04, 0.05	6.2	0.010	0.299, 0.328, 0.393, 0.395
SAC	✗	0.008	J	3.1	0.04, 0.06, 0.03, 0.02	7.4	0.014	0.256, 0.209, 0.041, 0.261
SAC	✓	0.0	C	2.2	0.03, 0.02, 0.07, 0.07	10.7	0.110	0.176, 0.253, 0.432, 0.310
SAC	✗	0.0	C	2.8	0.05, 0.07, 0.06, 0.05	7.6	0.033	0.220, 0.300, 0.386, 0.398
SAC	✗	0.0	J	3.2	0.06, 0.04, 0.05, 0.05	8.6	0.015	0.123, 0.274, 0.268, 0.181

We noticed that by calculating the energy term in COT in the way as shown in Eq. (3.4) without an absolute operator on the outside (similar to the form of Type I reward function in Section 3.2.1), we sometimes got negative COT results and interestingly we found that it's irrelevant to both PD type and Algorithm we chose based on the results we got. One explanation to this phenomenon is that the energy increases when the motor is braking since $\dot{q}$ is negative and τ is positive, and the energy flow in the pybullet environment might not resemble the real world environment in absent of various forms of energy loss.

$$E = \int \tau \cdot \dot{q} dt \tag{3.4}$$

The results of experiments with Type II (Eq. (3.2)) are demonstrated in Table 3.5

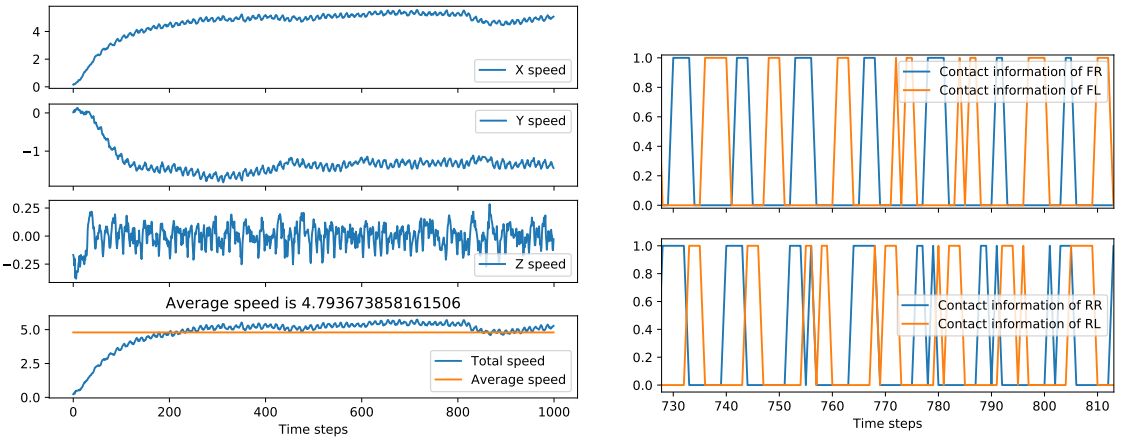
Table 3.5: Experiment results for Type II reward function; **C** denotes **C**artesian and **J** denotes **J**oint

Algorithm	Dynamics randomization	Energy weight	PD type	$\bar{v}$ (m/s)	$\bar{z_{foot}}$ (FR, FL, RR, RL)	$\bar{x_{base}}$	$\bar{COT}$	Duty factor (FR, FL, RR, RL)
PPO	✓	0.0001	C	1.7	0.02, 0.03, 0.07, 0.06	11.1	0.324	0.293, 0.240, 0.438, 0.469
PPO	✗	0.0001	C	1.1	0.03, 0.02, 0.04, 0.04	5.6	0.653	0.400, 0.432, 0.468, 0.554

From the results, we found the performance of reward function Type II was not significantly better than Type I, as we mentioned above. Sometimes the performance was even worse. We suppose the reason could be that some terms (e.g. third term on base angular velocity and/or fourth term on base orientation) in the reward function overly constrained the locomotion of the robot. However, the performance with dynamics randomization was obviously better than that without using this method.

3.8.1. Fastest Policy in Plain Environment

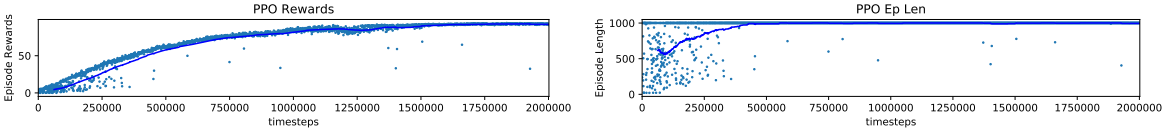
The fastest speed achieved by our training is $4.8\ m/s$ (tested in plain environment) as shown in Fig. 3.3a, and the contact information of four legs is shown in Fig. 3.3b the training was done with PPO without dynamics randomization, the training curve is shown in Fig. 3.4.



(a) Fastest x-y speed policy

(b) Contact information of fastest policy

Figure 3.3: Information on fastest policy



(a) Training reward curve

(b) Training episode length curve

Figure 3.4: Training of fastest locomotion

3.8.2. Different Gaits

In the results we also observed different gaits with different training setup, some might have only one different factor in the training setup.

Most of the gaits we got are in the form walking gait. But during the training we also get other types of gaits such as gallop from the fourth result in Table 3.4 we observed the emerging of gait Galloping as shown in Fig. 3.5. We compare this gait with the gait we got without dynamics randomization, we

believe the emerging of gallop is a mixed factors result of dynamics randomization and energy constraint. And its contact information is shown in Fig. 3.6a.

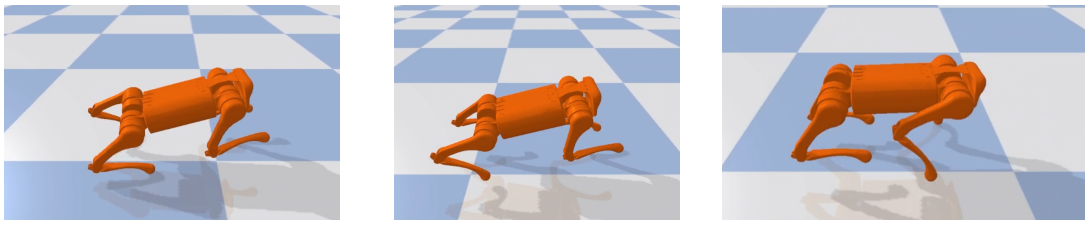
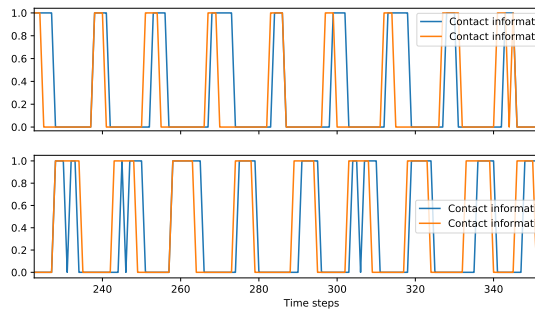
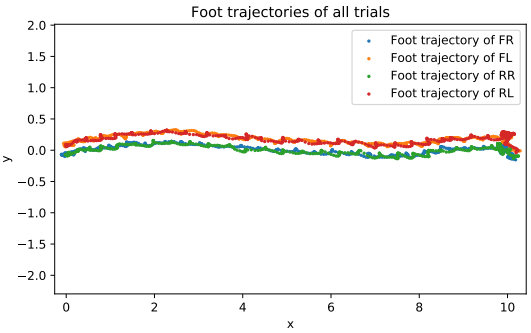


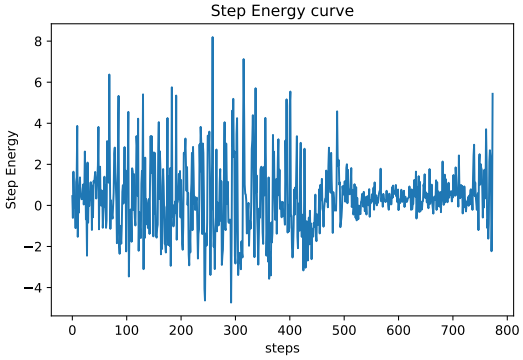
Figure 3.5: Galloping gait



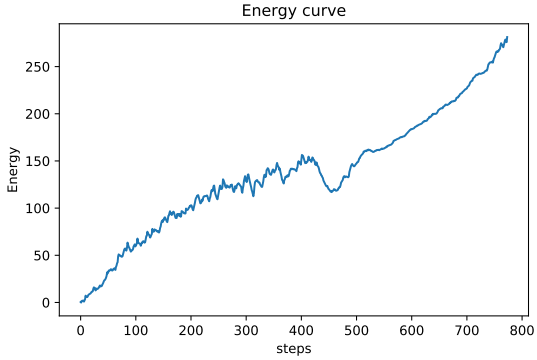
(a) Contact information of trial 4



(b) Foot Trajectory of trial 4



(c) Step energy of trial 4



(d) Total motor energy of trial 4

From Fig. 3.6b we can also see that the quadruped's trajectory only diverged slightly on the direction perpendicular to the target direction. And the energy trend is shown in Fig. 3.6c and Fig. 3.6d.

3.8.3. The Most Robust Policy

The most robust policy we get is trial 5 in Table 3.4, where the quadruped passes the obstacles in the competition environment in most of its trials ending with the highest $x_{\bar{base}}$, the gait in this policy however, might seem "wierd" compared to other gaits we got in other trainings. The gait displayed in this trial is the quadruped first turn around until its back facing the finish line and then it trots backward as shown in Fig. 3.7.

Our explanation to this seemingly weird gait is that when there's obstables, because of the structural design of the quadruped, it is less likely to fall over when its legs are pointing backward than pointing forward.

Figure 3.7: The most robust gait and its turning stage

3.8.4. PD Controller Comparison

When comparing the gaits learned with Cartesian PD to the gaits learned with joint PD, we found that the gaits learned by joint PD have a better performance when in competition environment. The robot's ability to go over obstacles is improved by joint PD. We think one reason for this is that Cartesian PD only considers the feet position of the robot, which is insufficient when encountered with obstacles. The feet of the robot could collide with obstacles and then the robot loses its balance. Another reason could be our mapping from action to feet position should be improved.

3.9. Discussion

Even though the quadruped worked as expected in the simulations, transferring to quadruped in reality however requires more in depth consideration and experimenting.

In terms of policy training, one aspect that we learned from [2] is that the selection of nonlinearity functions in the MLP policy has a great impact on real world quadruped controlling, the paper mentioned that unbounded activation functions, such as ReLU, can suffer from degrade performance on the real robot resulting in aggressive trajectories when there are disturbances imposed, the reason is that the action of real robot can perform high magnitude actions when the robot reaches states that were not visited in the training. Moreover, the joint properties vary among different robots.

Lastly, in order to achieve omni-directional locomotion, we have added the command velocity term in observation space and modified the reward function to incorporate the command velocity and orientation. Then we implement a new callback function – "CmdVelCallback". During the training, after certain "n_calls" of the "step" function, our callback function would change the command velocity to a pre-determined one (we could also generate the new command velocity randomly as long as it is proximity to the current velocity of the robot).

In our experiments, however, the performance of the robot locomotion is not good. The learned policy prefers the command velocities which are learned later. For example, we trained the robot with a list of command velocities of

$$[\cos 0, \sin 0, 0]\ , [\cos 30, \sin 30, 0]\ , [\cos 60, \sin 60, 0]\ , [\cos 90, \sin 90, 0]$$

but the policy seemed only followed the last two velocities. This could be checked in the video (in first half of the test the command velocity was $[\cos 0, \sin 0, 0]$, in the second half it was $[\cos 90, \sin 90, 0]$). We tried to increase the frequency of changing command velocity during training but that led to a easy-to-fall policy. We suppose finding an appropriate frequency to change command velocity and generating command velocities randomly (of course it should be close to the current base velocity of the robot) could be a solution to this problem. But unfortunately we did not have enough time to test this.

4

Conclusion

In this project, we implement two methods to generate the quadruped locomotion based on CPG and DRL respectively.

With respect to CPG part, we develop 7 different gaits for the robot. In order to improve the performance, we tune the parameters carefully and try to add some force feedback. From the results, we conclude that 1)For gaits with good symmetry, CPG shows great power; but for some gaits, much effort needs to be paid for good performance. 2)Force feedback really exert a significant impact on locomotion, which implies the importance of reflex in CPG method.

As for DRL part, we design 2 reward functions and test their performance. But the experimental results reveal that dynamics randomization plays a much more important role than reward function in training. Besides, a complicated reward function is likely to degrade the performance since it could overly constrain the robot locomotion. We also noticed the joint PD control seems to have a better performance than Cartesian PD control. We suppose the reason could be that in Cartesian PD control, the policy network only controls the feet position, which is not sufficient for the robot to go over obstacles. Also, another reason could be our mapping from action to feet position needs to be improved. Lastly, we attempt to handle the problem of omni-direction locomotion. We incorporate the command velocity in observation space and reward function and then design a callback function to change it during training. Unfortunately we do not have a good result.

References

[1] Guillaume Bellegarda and Quan Nguyen. "Robust High-speed Running for Quadruped Robots via Deep Reinforcement Learning". In: *arXiv preprint arXiv:2103.06484* (2021).

[2] Jemin Hwangbo et al. "Learning agile and dynamic motor skills for legged robots". In: *Science Robotics* 4.26 (Jan. 2019). arXiv: 1901.08652, eaau5872. ISSN: 2470-9476. DOI: 10.1126/scirobotics.aau5872. URL: http://arxiv.org/abs/1901.08652.

[3] Joonho Lee et al. "Learning quadrupedal locomotion over challenging terrain". EN. In: *Science Robotics* (Oct. 2020). Publisher: American Association for the Advancement of Science. DOI: 10.1126/scirobotics.abc5986. URL: https://www.science.org/doi/abs/10.1126/scirobotics.abc5986.

[4] Joonho Lee et al. "Learning quadrupedal locomotion over challenging terrain". In: *Science robotics* 5.47 (2020).

[5] Robert B. McGhee. "Finite state control of quadruped locomotion". en. In: *SIMULATION* 9.3 (1967). Publisher: SAGE Publications Ltd STM, pp. 135–140. ISSN: 0037-5497. DOI: 10.1177/003754976700900308. URL: https://doi.org/10.1177/003754976700900308.

[6] M. H. Raibert. "Legged robots that balance". English. In: (Jan. 1985). Publisher: The MIT Press,Cambridge, MA. URL: https://www.osti.gov/biblio/5606728.

[7] Marc Raibert et al. "BigDog, the Rough-Terrain Quadruped Robot". en. In: *IFAC Proceedings Volumes*. 17th IFAC World Congress 41.2 (2008), pp. 10822–10825. ISSN: 1474-6670. DOI: 10.3182/20080706-5-KR-1001.01833. URL: https://www.sciencedirect.com/science/article/pii/S1474667016407020.

[8] Ludovic Righetti and Auke Jan Ijspeert. "Pattern generators with sensory feedback for the control of quadruped locomotion". In: *2008 IEEE International Conference on Robotics and Automation*. 2008, pp. 819–824. DOI: 10.1109/ROBOT.2008.4543306.

[9] Chris Zink and Brittany Jean Carr. "Locomotion and Athletic Performance". In: *Canine Sports Medicine and Rehabilitation*. John Wiley & Sons, Ltd, 2018. Chap. 2, pp. 23–42. ISBN: 9781119380627. DOI: https://doi.org/10.1002/9781119380627.ch2. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/9781119380627.ch2. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/9781119380627.ch2.