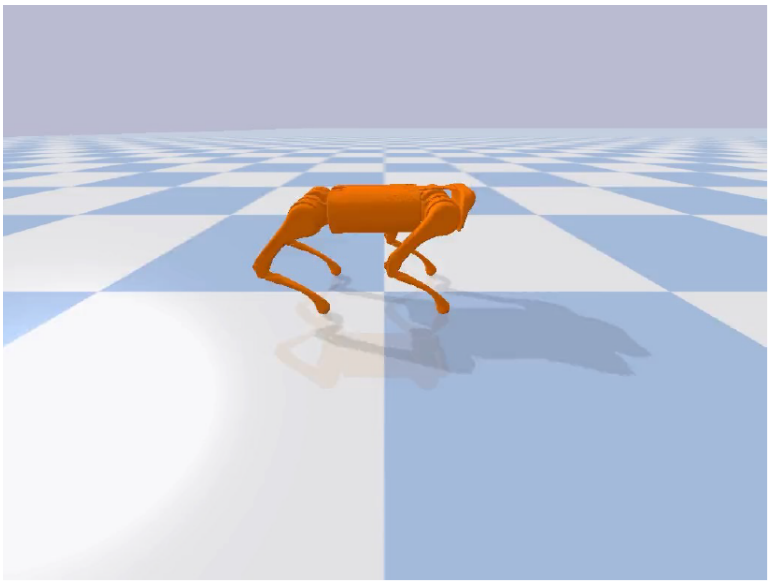
EPFL

# Legged Robot

# Quadruped locomotion with Central Pattern Generators and Deep Reinforcement Learning



*Authors*
Victor Dramé
Lucas Represa
Carole Ruppli

*Professor*
Auke Ijspeert

January 7th, 2022

# Table of contents

# Introduction

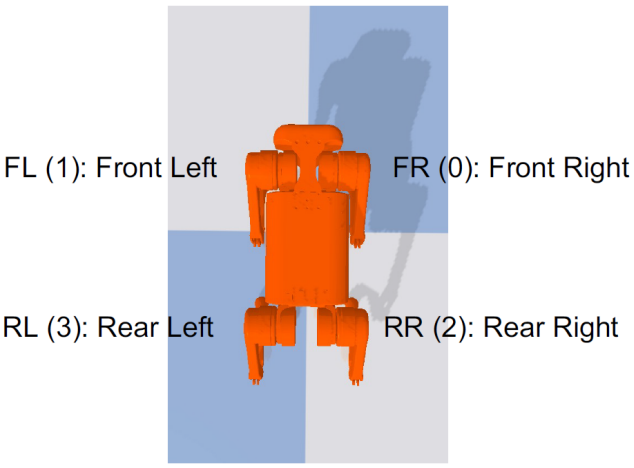
Quadruped locomotion modelling is a complex task which can be implemented using many different methods. The goal of this project is to study two of those: Central Pattern Generators (CPG) and a Markov Decision Process (MDP).

CPG is a bio-inspired method reproducing the mechanism of locomotion identified in vertebrates and invertebrates animals. It consists of generating oscillators to replicate their distributed control mechanisms. MDP is a learning based method where outputs are a mix of random and under the control of a decision maker. By using machine learning tools and algorithms, it allows to find model for locomotion and generate movements, from various observations and specific rewards.

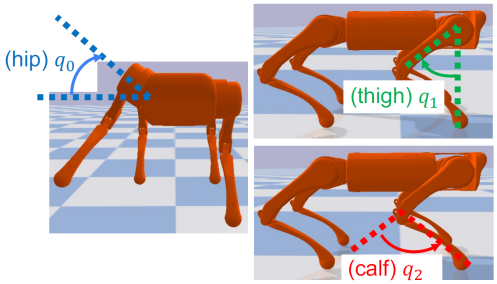
For each of those methods, different gaits are produced and the process to obtain them is detailed in the methods section. Then, the results obtained are shown and discussed.

# 1 Quadruped modelling

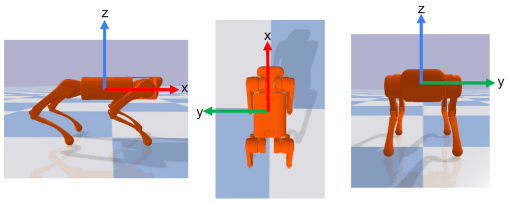
The quadruped studied here has 4 legs numbered from 0 to 3, starting with the front and the right side first as shown in Figure 1a. Each leg has 3 joints (hip, thigh, calf) for a total of 12 motors, as shown in Figure 1b. The reference frame used is the one in Figure 1c.




(a) Quadruped model leg references




(b) Quadruped model joints references




(c) Quadruped model reference frame

Figure 1: References for the model

For both methods, the torque is computed using a joint PD and cartesian PD controller. The formulas used are :

$$\tau_{joint} = K_{p,joint}(q_d - q) + K_{d,joint}(\dot{q}_d - \dot{q}) \tag{1}$$

with $q_d$ the desired joint angles, $\dot{q}_d$ the desired joint velocities, $q$ and $\dot{q}$ the current joint angles and joint velocities, and $K_{p,joint}$ and $K_{d,joint}$ vectors of proportional and derivative gains.

$$\tau_{cartesian} = J^T(q)\big[K_{p,cartesian}(p_d - p) + K_{d,cartesian}(v_d - v)\big] \tag{2}$$

with $J(q)$ the foot Jacobian at joint configuration $q$, $K_{p,cartesian}$ and $K_{d,cartesian}$ diagonal matrices of proportional and derivative gains.

# 2 Central patterns generators

## 2.1 Methods

The first part of this project implements a locomotion controller based on CPGs, consisting of coupled Cartesian space Hopf oscillators.

Four oscillators are generated with the following equations for the $i^{th}$ oscillator:

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

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

With $r_i$ the current amplitude of the oscillator, $\theta_i$ the phase of the oscillator, $\sqrt{\mu}$ the desired amplitude of the oscillator, $\alpha$ a positive constant that controls the speed of convergence to the limit cycle, $\omega_i$ the natural frequency of oscillations, $\omega_{ij}$ the coupling strengh between oscillators $i$ and $j$, and $\phi_{ij}$ the desired phase offset between oscillators $i$ and $j$.

Those are implemented in the `hopf_network.py` file, using the following way to map the CPG to joint commands:

$$x_{foot} = -d_{step} r_i cos(\theta_i) \tag{3}$$

$$z_{foot} = \begin{cases} -h + g_c sin(\theta_i) & \text{if } sin(\theta_i) > 0, \\ -h + g_p sin(\theta_i) & \text{otherwise} \end{cases} \tag{4}$$

With $d_{step}$ the step length, $h$ the robot height, $g_c$ the max ground clearance during swing, and $g_p$ the max ground penetration during stance.

From this, inverse kinematics is done to get the corresponding desired joint references to compute the torque, as shown in section 1.

### Choice of the parameters

The following parameters are implemented and/or tuned to find stable gaits.

- **Coupling matrices** $\phi_{gait}$ : for each type of gait, the coupling matrix defined is:

$$\phi_{walk} = \begin{bmatrix} 0 & -\pi & -\pi/2 & \pi/2 \\ \pi & 0 & \pi/2 & 3\pi/2 \\ \pi/2 & -\pi/2 & 0 & \pi \\ -\pi/2 & 3\pi/2 & -\pi/2 & 0 \end{bmatrix}$$

$$\phi_{trot} = \begin{bmatrix} 0 & \pi & \pi & 0 \\ -\pi & 0 & 0 & -\pi \\ -\pi & 0 & 0 & -\pi \\ 0 & \pi & \pi & 0 \end{bmatrix}$$

$$\phi_{bound} = \begin{bmatrix} 0 & 0 & -\pi & -\pi \\ 0 & 0 & -\pi & -\pi \\ \pi & \pi & 0 & 0 \\ \pi & \pi & 0 & 0 \end{bmatrix}$$

$$\phi_{pace} = \begin{bmatrix} 0 & -\pi & 0 & -\pi \\ \pi & 0 & \pi & 0 \\ 0 & -\pi & 0 & -\pi \\ \pi & 0 & \pi & 0 \end{bmatrix}$$

- **Swing and stance frequencies**: for each type of gait and controller (i.e with or without cartesian PD), we found experimentally the values for swing and stance frequencies in order to achieve manually the fastest and most stable gait as possible with respect to the other parameters that we will cover.

| | | **Walk** | **Trot** | **Bound** | **Pace** |
|---|---|---|---|---|---|
| **With cartesian PD** | $\omega_{swing}$ | $6 * 2\pi$ | $12 * 2\pi$ | $12 * 2\pi$ | $16 * 2\pi$ |
| | $\omega_{stance}$ | $3 * 2\pi$ | $6 * 2\pi$ | $6 * 2\pi$ | $4 * 2\pi$ |
| **Without cartesian PD** | $\omega_{swing}$ | $7 * 2\pi$ | $9 * 2\pi$ | $14 * 2\pi$ | $9 * 2\pi$ |
| | $\omega_{stance}$ | $4 * 2\pi$ | $3 * 2\pi$ | $3 * 2\pi$ | $4 * 2\pi$ |

Table 1: Swing and stance frequencies for the different gaits

- **The joint and Cartesian PD controllers**: the relation used is the one seen in equations 1 and 2. We optimized all of our gaits with and without adding the cartesian PD.
- **The mapping from the CPG states to desired quadruped foot positions**: the relation used is the one seen in equations 3 and 4.
- **Ground clearance and ground penetration**: for each type of gait, these values have also been tuned experimentally to get the best gait possible. Therefore, the values found are:

| **Gait** | **Walk** | **Trot** | **Bound** | **Pace** |
|---|---|---|---|---|
| $g_c$ | 0.035 | 0.05 | 0.06 | 0.05 |
| $g_p$ | 0.005 | 0.007 | 0.0045 | 0.007 |

Table 2: Ground clearance and ground penetration for the different gaits

- **The robot height**: the height is kept to its default value which is 0.25.
- **The desired step length**: the default value is 0.04. Only for the bound gait, it is set to 0.042.
- **The joint PD and cartesian PD gains**: for all gaits, they are set as following:

$$K_{d,joint} = \begin{bmatrix} 150 & 70 & 70 \end{bmatrix}$$

$$K_{p,joint} = \begin{bmatrix} 2 & 0.5 & 0.5 \end{bmatrix}$$

$$K_{p,cartesian} = \begin{bmatrix} 2500 & 0 & 0 \\ 0 & 2500 & 0 \\ 0 & 0 & 2500 \end{bmatrix}$$

$$K_{d,cartesian} = \begin{bmatrix} 40 & 0 & 0 \\ 0 & 40 & 0 \\ 0 & 0 & 40 \end{bmatrix}$$

## 2.2 Results

### 2.2.1 CPG states

The plots of the CPG states $(r, \theta, \dot{r}, \dot{\theta})$ for each leg and for each gait are shown below, with 1000 *test_step* as X axis (with *test_step* corresponding to the number of steps divided by the timestep, with 10 number of steps and 1ms as timestep).

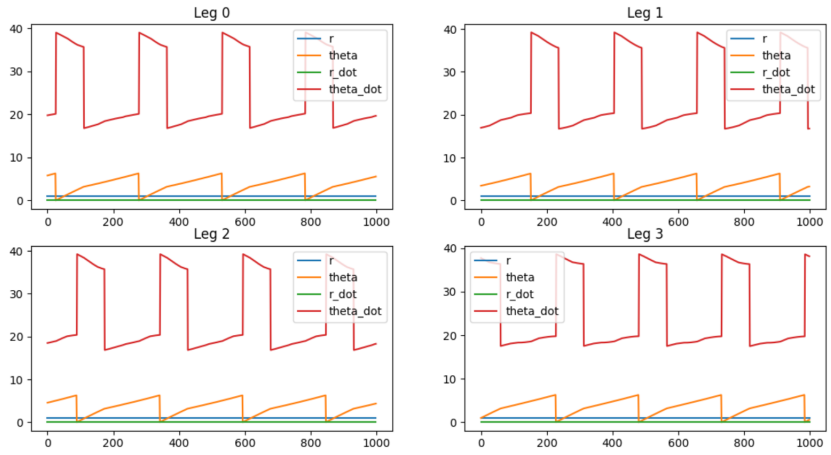



Figure 2: Plot of the CPG states $(r, \theta, \dot{r}, \dot{\theta})$ for each leg for walking gait for 1000 *test_step* (1000ms)

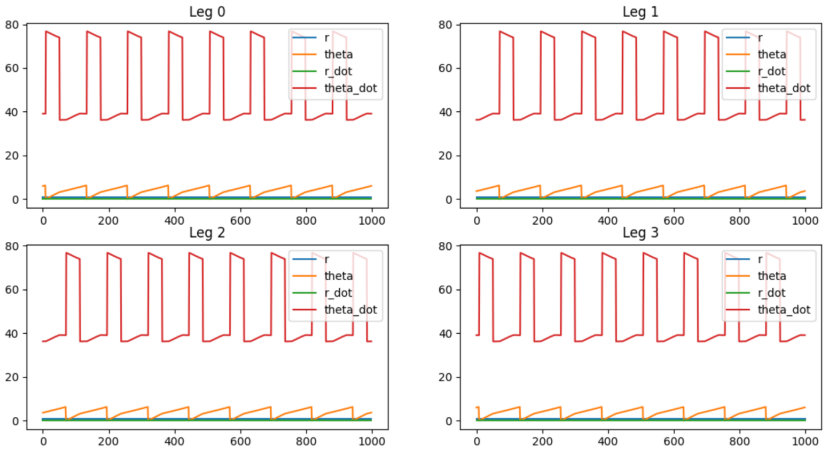



Figure 3: Plot of the CPG states $(r, \theta, \dot{r}, \dot{\theta})$ for each leg for trotting gait for 1000 *test_step* (1000ms)

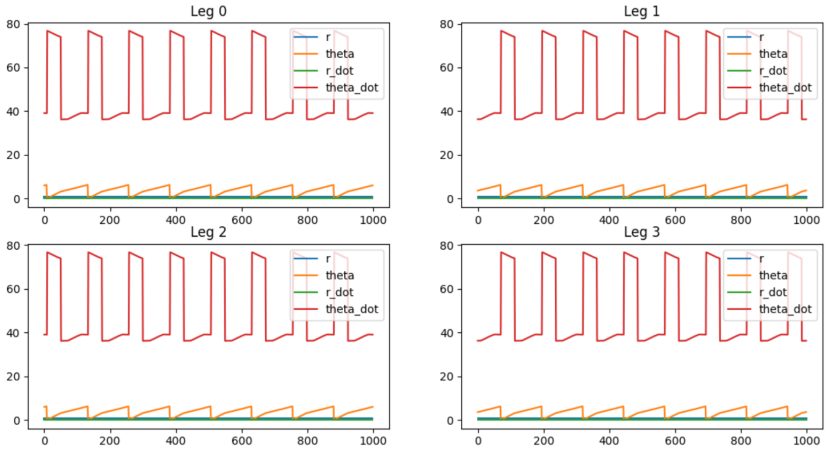



Figure 4: Plot of the CPG states $(r, \theta, \dot{r}, \dot{\theta})$ for each leg for pacing gait for 1000 *test_step* (1000ms)

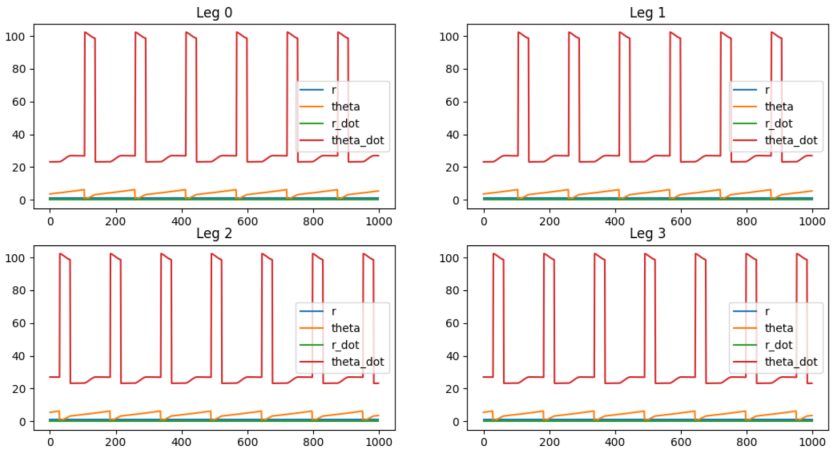



Figure 5: Plot of the CPG states $(r, \theta, \dot{r}, \dot{\theta})$ for each leg for bounding gait for 1000 *test_step* (1000ms)

The variables here are not plotted for the first 1000 ms but rather for from 1000 to 2000 ms so that the `r` is stabilized. We can mostly observe the impact of the swing and stance frequencies in the oscillators generated. Having values twice higher for the trot (Figure 3) and the pace (Figure 4) than for the walk (Figure 2) allows to make double the number of steps as it is increasing the frequency. Their difference also determines the proportion of stance and swing for one gait.

Finally, we can also observe that the amplitude of the oscillations is also dependant of the frequencies set. We see that the amplitude of the $\dot{\theta}$ depends on the difference between swing and stance frequencies (the higher the difference, the higher the amplitude).

### 2.2.2 Desired foot position

The plots of the desired foot position for the leg number 1 and for each gait are shown below, for 1000 teststeps (1000ms), and for each type of controller, with the best parameters found for each.

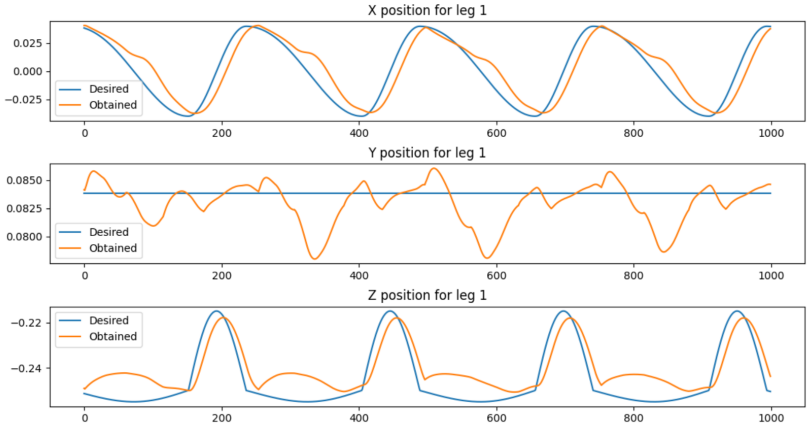



Figure 6: Desired foot position vs. actual foot position (in meters) with joint PD and Cartesian PD for walking gait for 1000 *test_step* (1000ms)

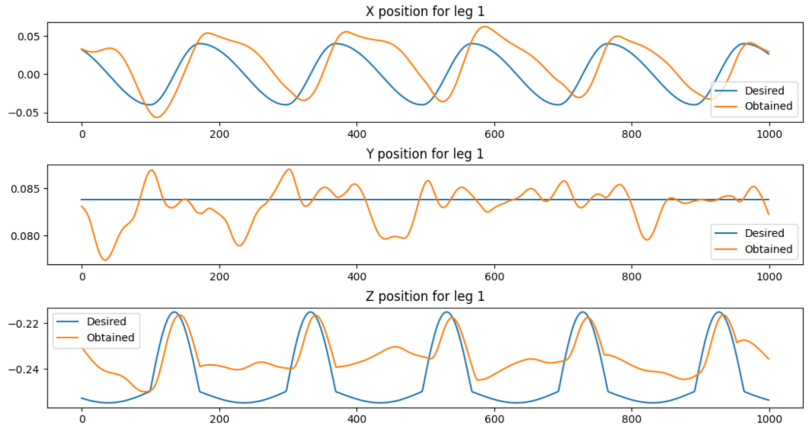



Figure 7: Desired foot position vs. actual foot position (in meters) with joint PD and without Cartesian PD for walking gait for 1000 *test_step* (1000ms)

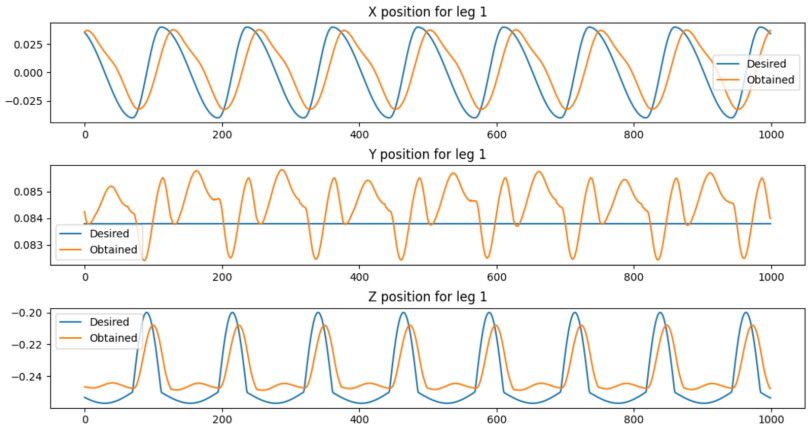



Figure 8: Desired foot position vs. actual foot (in meters) position with joint PD and Cartesian PD for trotting gait for 1000 *test_step* (1000ms)

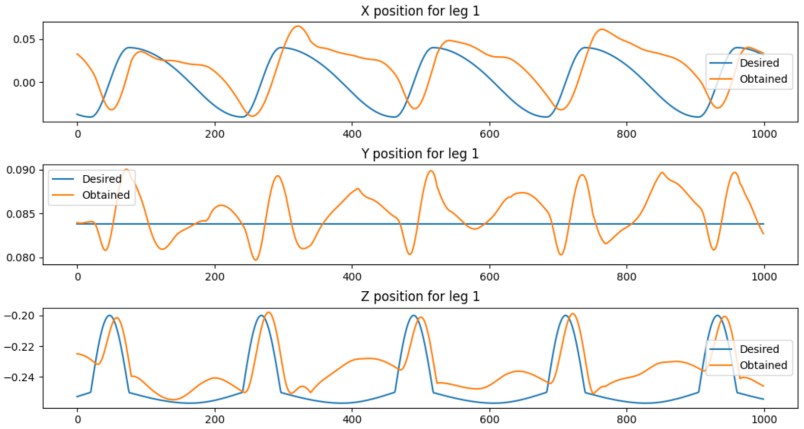



Figure 9: Desired foot position vs. actual foot position (in meters) with joint PD and without Cartesian PD for trotting gait for 1000 *test_step* (1000ms)

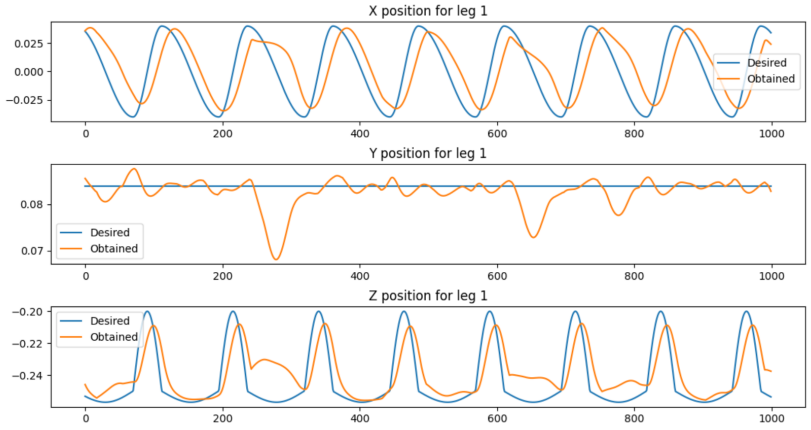



Figure 10: Desired foot position vs. actual foot position (in meters) with joint PD and Cartesian PD for pacing gait for 1000 *test_step* (1000ms)

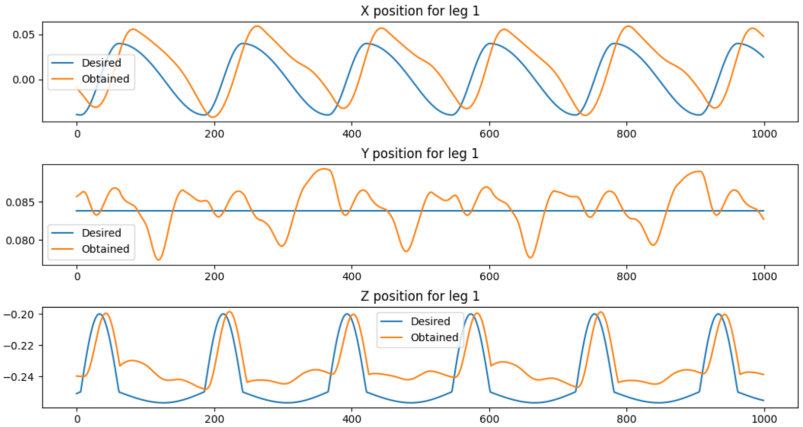



Figure 11: Desired foot position vs. actual foot position (in meters) with joint PD and without Cartesian PD for pacing gait for 1000 *test_step* (1000ms)

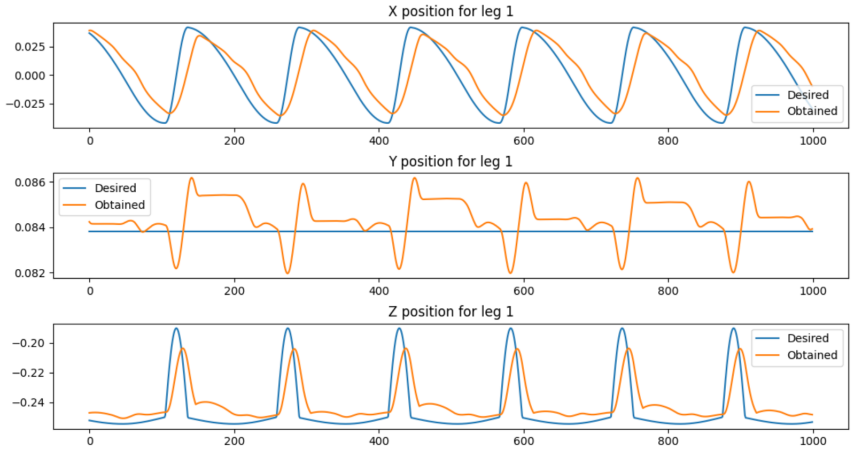



Figure 12: Desired foot position vs. actual foot position (in meters) with joint PD and Cartesian PD for bounding gait for 1000 *test_step* (1000ms)

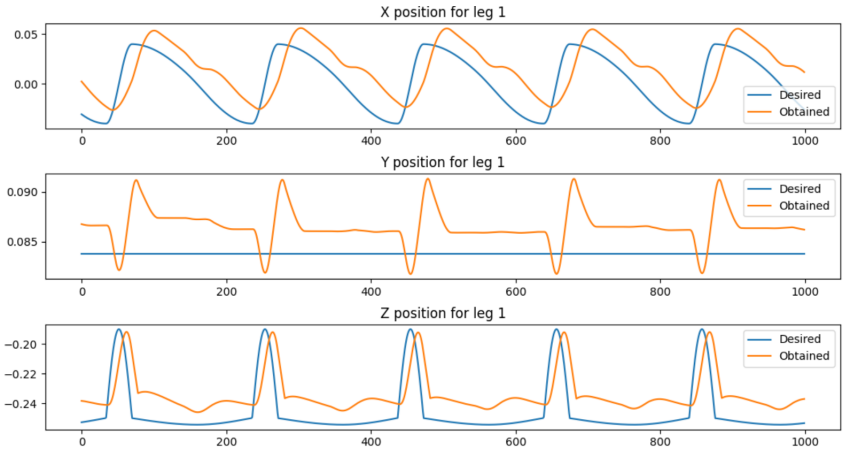



Figure 13: Desired foot position vs. actual foot position (in meters) with joint PD and without Cartesian PD for bounding gait for 1000 *test_step* (1000ms)

Here, we can see that the robot is pretty good at following in the X direction for all types of gaits, when both PD controllers are used. This could still be tuned to better results by changing the gains of the PD controllers, but using both PD controllers already allows for better tracking performances than only one. Indeed, the same parameters without cartesian PD controller produce a gait that is very unstable.

For the Z position, the tracking is pretty accurate as well in the swing phase. The actual position can not exactly reach the desired one during stance as it is touching the ground and needs to pull the body instead.

As for the Y direction, the desired position is to stay always in the X axis with zero Y component, but we observe a small variation that could be explained by the motors having a small rotation in that direction if they are not perfectly aligned.

### 2.2.3 Desired joint angles

The plots of the desired foot angle for the leg number 1 and for each gait are shown below, for 1000 Test Steps. Again, both PD controllers have been used here to optimize the parameters.

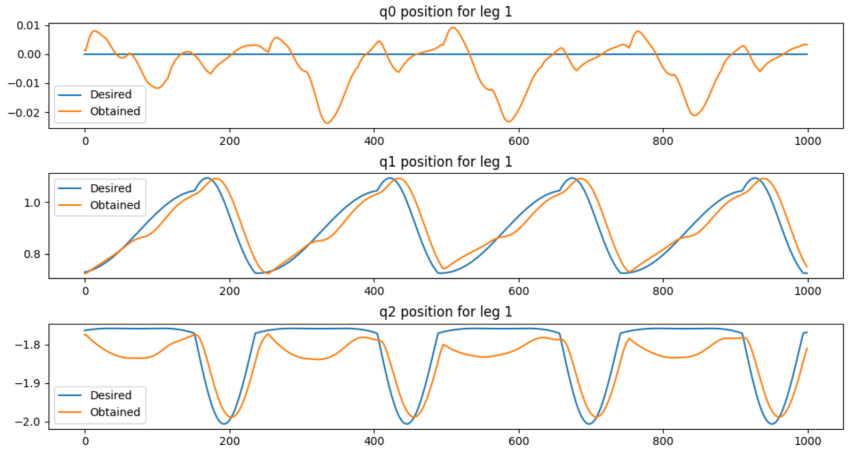



Figure 14: Desired joint angles vs. actual joint angles with joint PD and Cartesian PD for walking gait for 1000 *test_step* (1000ms)

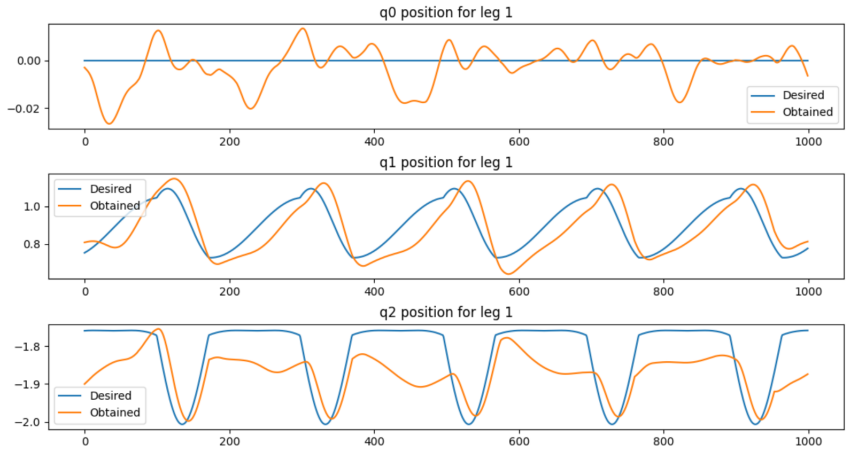



Figure 15: Desired joint angles vs. actual joint angles with joint PD and without Cartesian PD for walking gait for 1000 *test_step* (1000ms)

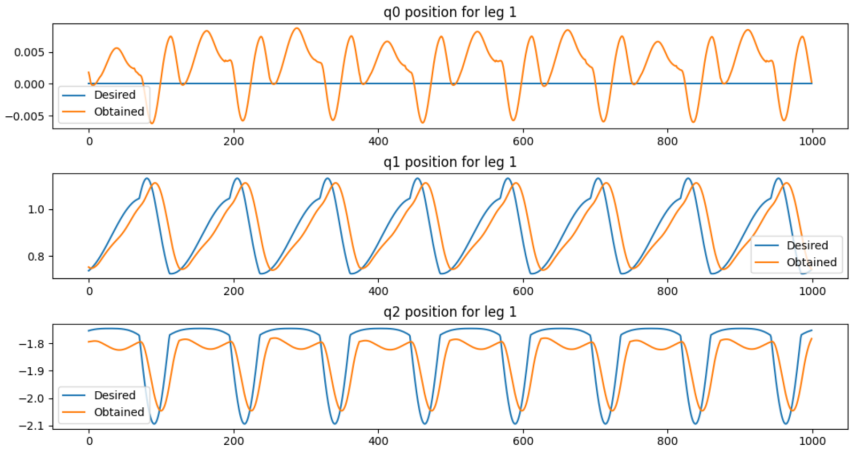



Figure 16: Desired joint angles vs. actual joint angles with joint PD and Cartesian PD for trotting gait for 1000 *test_step* (1000ms)

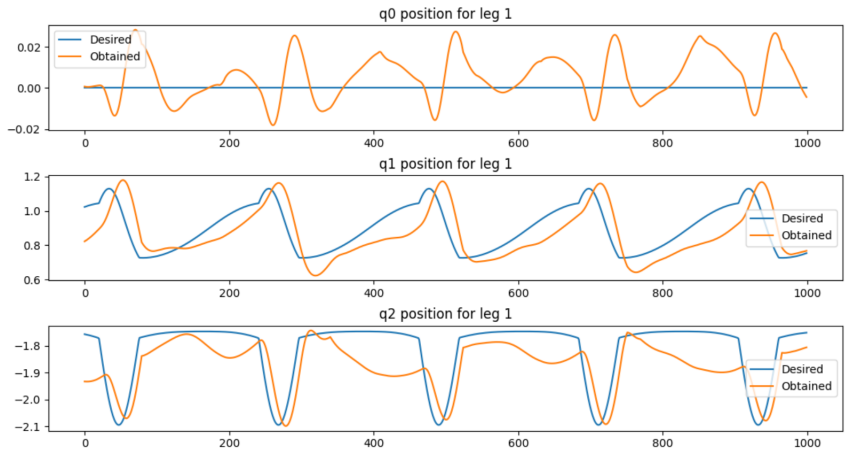



Figure 17: Desired joint angles vs. actual joint angles with joint PD and without Cartesian PD for trotting gait for 1000 *test_step* (1000ms)

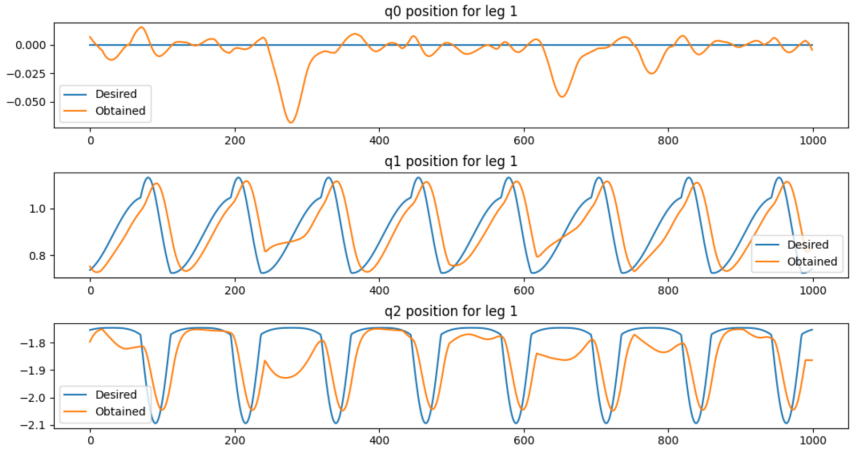



Figure 18: Desired joint angles vs. actual joint angles with joint PD and Cartesian PD for pacing gait for 1000 *test_step* (1000ms)

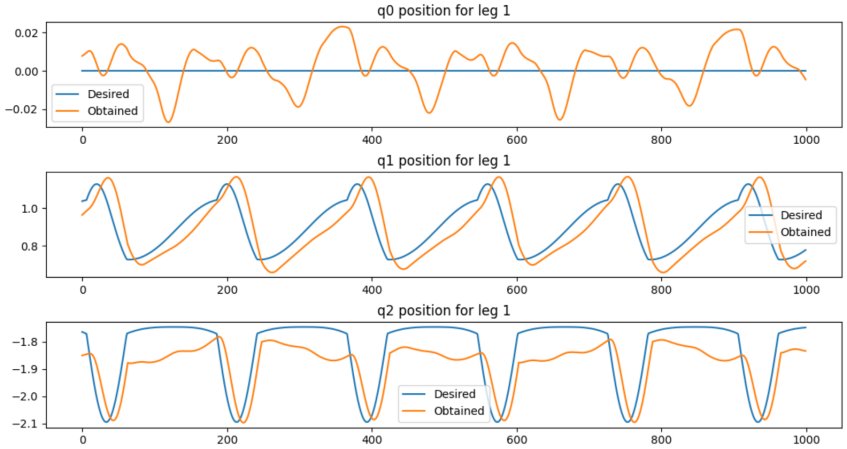



Figure 19: Desired joint angles vs. actual joint angles with joint PD and without Cartesian PD for pacing gait for 1000 *test_step* (1000ms)

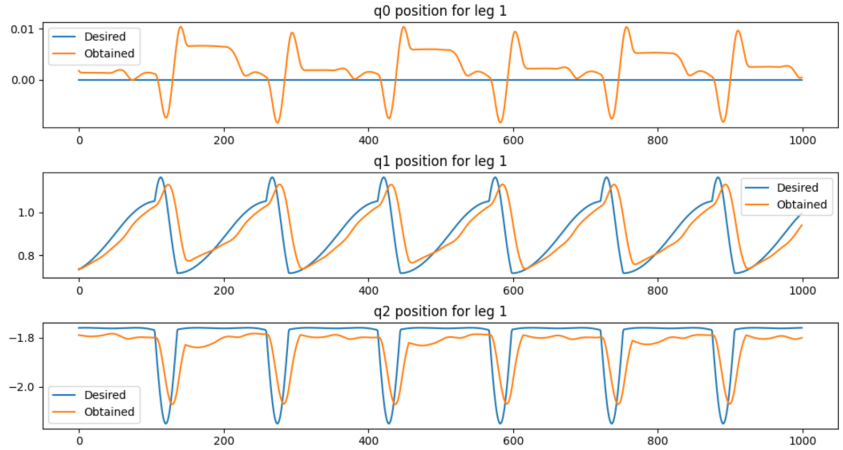



Figure 20: Desired joint angles vs. actual joint angles with joint PD and Cartesian PD for bounding gait

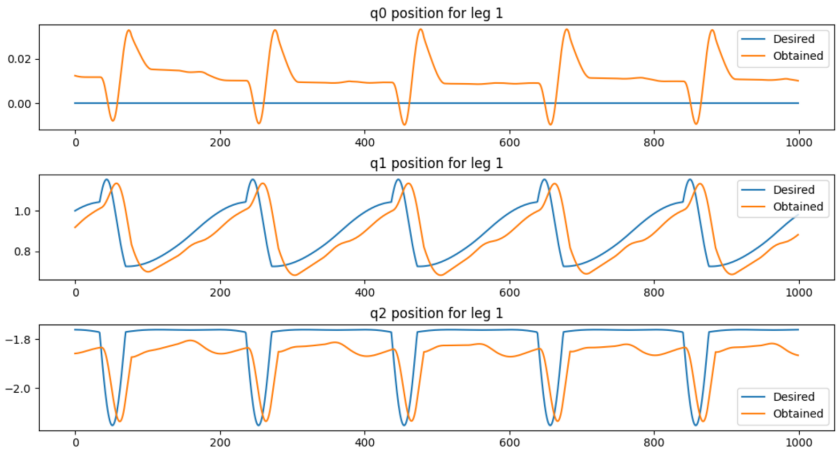



Figure 21: Desired joint angles vs. actual joint angles with joint PD and without Cartesian PD for bounding gait for 1000 *test_step* (1000ms)

Finally, we can see here similar behaviours as for the desired position. q0 is similar to Y as it is the angle allowing to turn around the X axis. q2 is similar to Z as it is the angle that allows to adapt the height of the robot.

## 2.3 Discussion

### 2.3.1 Choice of parameters

To get the most stable gaits, the parameters necessary to tune were:

- The swing and stance frequencies: with increasing them, it produces oscillations with higher frequencies, which makes a faster movements. The rate of changes from swing to stance is determined by the ratio between them. To keep balance, the ratio between stance and swing is 2 to 4 times bigger for the swing frequency.
- The ground clearance: this parameter changes the desired foot height when the foot is in swing mode. By setting it high, it settles a higher desired foot position. This was necessary to tune to obtain better gait specific performances. It is interesting to have the biggest value as possible to go faster. But by setting it too high, the goal is higher and the movement needs to be faster, so it also implies instability due to the delay.
- The ground penetration: this parameter changes the desired foot height when in stance mode, which corresponds to the force exerted on the ground to push the robot. A too high value also creates a bouncy movement.
- The step length: The step length is almost always the same, but helps to go a little bit more forward as it is used in the desired X position of the foot computation.

With these parameters tuned, it gave us the following results for different metrics.

| **Gait** | **Walk** | **Trot** | **Bound** | **Pace** |
|---|---|---|---|---|
| **velocity [m/s]** | 0.31 | 0.93 | 0.34 | 0.82 |
| **Duty cycle** | 0.61 | 0.56 | 0.68 | 0.51 |
| **Duration of one step [s]** | 0.22 | 0.124 | 0.135 | 0.125 |
| **Cost of transport** | 0.59 | 1.06 | 3.92 | 1.17 |

Table 3: Results of different metrics depending on the gaits

From Table 3, it is clear that our BOUND gait did not achieve good performances since its cost of transport is very high compared to the others. However, for the other gaits, we can assume that we have much better behaviours. For the walking gait, the cost of transport and the speed are low which is expected. For the Pace and Trot, we achieved faster locomotion associated with a higher cost of transport.

### 2.3.2 Controllers

As for the impact of the cartesian controller, we notice an upgrade of stability when combining it with the joint PD controller thanks to better tracking. Even if the two controllers can seem redundant, when turning off the cartesian PD, some issues can appear while keeping the same parameters. The robot can even go backwards or stagnate at some point. Tuning the duration of swing and stance phase allows to reach a stable gait again, but with a loss of velocity. Nevertheless, we focused on both controllers in order to achieve an optimized version with the more robust controller. That's why we didn't tried to optimize the parameters with only one controller which could have been possible as well.

### 2.3.3 Extensions

One extension that could significantly improve the robustness of the CPG's model would be a reflex based feedback loop. This method has been implemented in many quadrupedal robots and has been reviewed in articles notably the Ajallooeian et al. [1] from which we take inspiration.

This kind of feedback relies on sensors from the limbs in contact with the ground. If the swing leg hits an obstacle, and the stance leg is not about to retract (otherwise it would be useless), a simple feedback function gives the command to the swing leg to retract. This extra flexion thus allows to blindly pass many obstacles with a simple feedback loop to the CPGs.

# 3 Deep reinforcement learning

## 3.1 Methods

In the 2nd part of the project, a Markov Decision Process (MDP) is designed. Then, control policies are trained to solve it with state-of-the-art Deep Learning (DL) algorithms. The first step is to determine what to put in the observation space, in the action space and which reward function to use.

### 3.1.1 Observation space

The observation space is composed of the following variables :

- **Motor angle and torque**: they allow the agent to know the current torque it applies and to be coherent with it. They are one of our basic observations. In real life, angle observation should have noise but not too high since the encoders have a fairly high resolution. For the torque, the measure would probably have more noise unless we have high quality force sensors.
- **Base linear velocity, angular velocity and orientation (quaternion)**: Out of these parameters, the most important is the orientation. We chose quaternion over roll pitch as it provides a more stable orientation representation. On a real robot, the accuracy of these representation depends highly on the grade and type of sensors available for the robot. For a classical IMU (gyroscope and accelerometer), we could consider that the angular velocity should be fairly accurate and that the linear velocity and orientation would be more noisy since it needs integration to derive it.
- **Foot contact information (optional)**: it is used sometimes because we want to be able to force certain gait to the agent with a specific reward mentioned in the following section.

### 3.1.2 Reward functions

Our reward function is based on the forward locomotion multiplied by the cosinus of the orientation minus a phase shift $\phi$, that allows us to control the orientation of the body. $\phi$ is the angle between the robot and its moving direction, so it is 0 when moving forward. The locomotion reward is then :

$$w1 * (currentBasePosition - lastBasePosition) * cos(yaw - \phi)$$

To avoid unrealistic movements, we penalized high energy movements with a cost of transport. It is calculated by the following formula and will be substracted to the reward function in order to compensate the energy loss (the two timesteps were chosen experimentally):

$$w2 * torque * velocity * 2 * timesteps$$

The final reward that we implemented is a gait reward. It rewards certain type of foot contact between the agent and the ground and is computed as :

$$w3 * gaitMatrix * footContact + PastGaitReward$$

To ensure constancy, we keep not changing the reward when there is no contact between the agent and the ground. The different gait matrices are provided below:

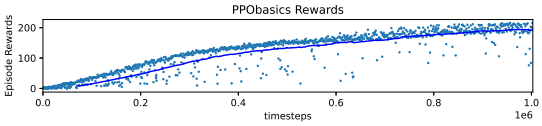
| **Gait** | **GaitMatrix** |
|---|---|
| *Pace* | [1, -1, 1, -1] |
| *Bound* | [1, 1, -1, -1] |
| *Trot* | [-1, 1, 1, -1] |

Table 4: GaitMatrix values used in the equation for the reward

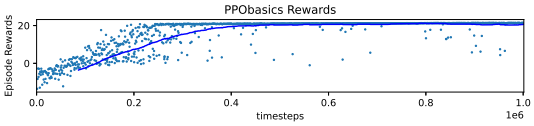
All the weights $(w1, w2, w3)$ were chosen experimentally and settled when satisfying results where reached.

### 3.1.3 Action space

For our action space, a design choice needed to be done between a joint PD control and cartesian PD control. We grounded our choice based on the following results:

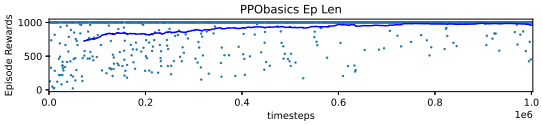



Cartesian PD controller

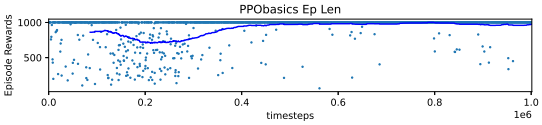



Joint PD controller

Figure 22: Reward over time steps for the two control methods

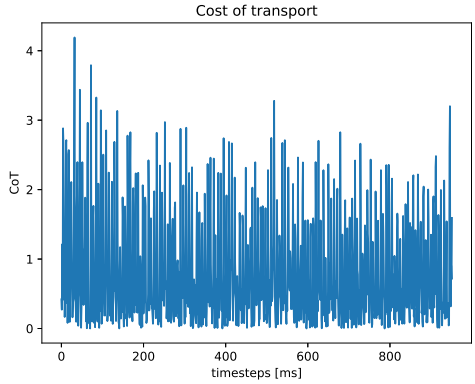



Cartesian PD controller

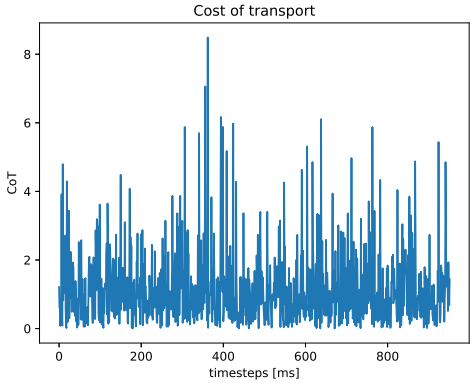



Joint PD controller

Figure 23: Episode length over time steps for the two control methods

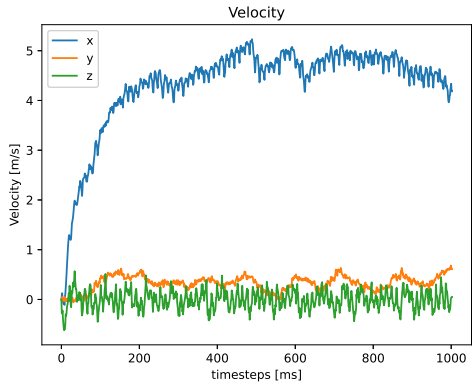



Cartesian PD controller, with an average CoT of 0.809

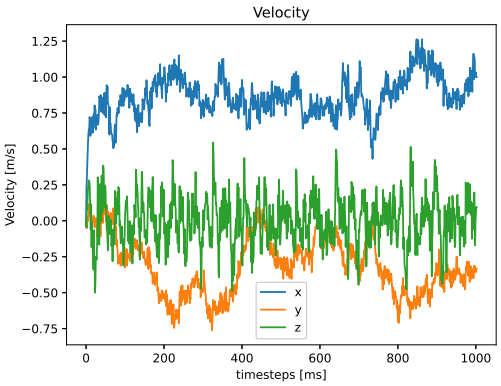



Joint PD controller, with an average CoT of 1.149

Figure 24: Cost of transport (CoT) over time steps for the two control methods




Cartesian PD controller




Joint PD controller

Figure 25: Velocity over time steps for the two control methods

As we can see in figure 22 and 23, both methods tend to converge to a solution. However, the one with the joint space converges to a solution that is sub-optimal compared to the one with the Cartesian PD. This could probably be resolved if we added more noise on the observations/actions, played with the scaling of the action or even modified the reward. However, we wanted here to compare the performances of our implementation over two different action methods.

As we can see in figure 24 and 25, our cartesian PD achieves good performances, the speed approaches the physical limitation of the robot (≈5m/s) and has a small cost of transport (0.809). The reward curve is also interesting as we can distinguish two slopes where the robot learns to advance in the first part of the training, then learns to maximize its speed and takes into account the energy cost introduced.

To compare both result, links are available in the videos section with the name 'Fast boi' & 'joint PD controller test' for the cartesian PD and joint PD respectfully.

## 3.2 Reinforcement learning policy (RLP)

For the RLP, we explored both *Proximal Policy Optimization* (PPO) and *Soft Actor Critic* (SAC). Each of these architectures has a different logic and will provide different behaviours.

### 3.2.1 Proximal Policy Optimization (PPO)

PPO is a on-policy gradient method, meaning that PPO learns online without replaying the buffer. Once a batch of experiment is used for updating the gradient, it is no longer useful. To avoid modifying the policy too much between batches and breaking the networks, an idea similar to the Trust Region Policy Optimization (TRPO) is implemented, which basically clipps the loss function to limit the gradient descent over the networks. Another main characteristics of PPO is that it enables the use of multiples agents, each collecting data which will then be used to update the policy. We didn't use these features much in our training since we achieved satisfying results without them.

For our implementation, we didn't changed the default parameters much since the results from them were satisfying. However, if we were to change some hyperparameters, it would probably be the following ones:

- **NUM_ENVS**: number of simulation running in parallel.
- **ent_coef**: It's the entropy coefficient for the loss calculation. Modifying would allow to explore new behaviors in random ways.
- **learning_rate**: we could modify the learning rate, for example, giving a bigger learning rate at the beginning of the training.
- **max_grad_norm**: It's the maximum value for the gradient clip.

### 3.2.2 Soft Actor Critic (SAC)

SAC is an off-policy actor-critic, meaning that it can be trained on all previously recorded data. The agent's aim here is to maximize the expected reward while maximizing entropy/randomness of the policy. It is reported by Haarnoja and al. [2] that SAC should be more sample efficient and converges easily. However, we had more difficulty tuning the parameters here and since our environment was cheap to sample and PPO was already working, we didn't dig too deep into this RLP.

For further exploration, we would play with the different parameters :

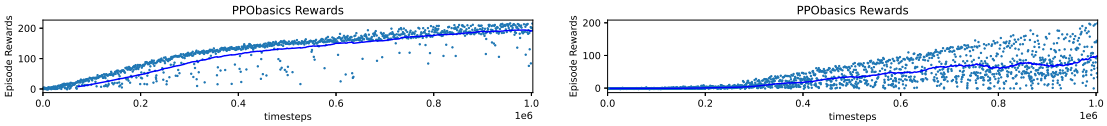
- **ent_coef** is the entropy regularization coefficient, we would keep it to the autoregulation setting.
- **buffer_size** is the size of replay buffer, we should keep it big to have memory of the past policy.
- **batch_size** is the size of the batch for each gradient update.
- **learning_rate**
- **train_freq** represented the frequency at which we update the state, we would probably set it to higher than one since one seems to make our policy change a lot between iterations.
- **gradient_steps** is how many gradient steps to do after each rollout, we would probably keeps it to one but other value could be explored.

## 3.3 Environmental details

For the environment, we decided to do the training only on a flat terrain, with a friction coefficient changing over simulation. Then we assess the performances on both flat terrain and terrain with obstacles. Some results are available in the Videos section.

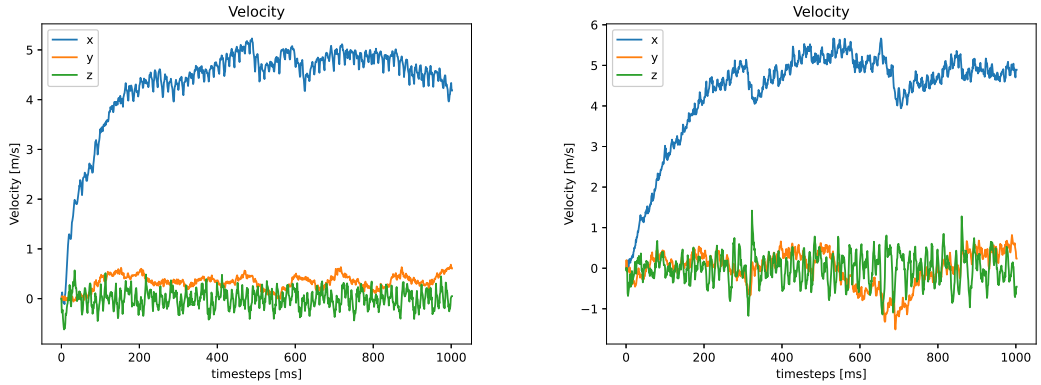
## 3.4 Results

For the results, we decided to test our policy for multiple $\phi$, leading to multiple modes of locomotion. The more interesting ones where for $\phi = 0$ and $\phi = \frac{\pi}{2}$, where the agent was moving sideway or running in a bound pace. Qualitative videos are available as 'Fast boi' and 'Side boi' respectively.



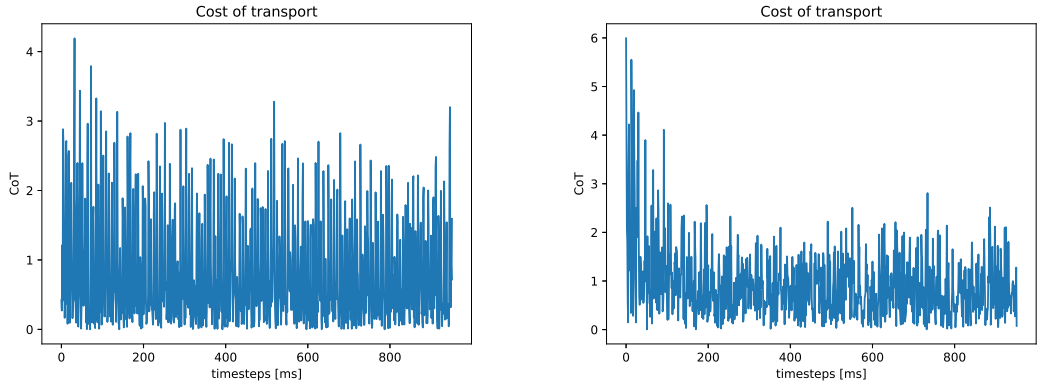

Fast boi - $\phi = 0$ Side boi - $\phi = \frac{\pi}{2}$

Figure 26: Reward over time steps for various $\phi$




Fast boi - $\phi = 0$ Side boi - $\phi = \frac{\pi}{2}$

Figure 27: Velocity over time steps for various $\phi$



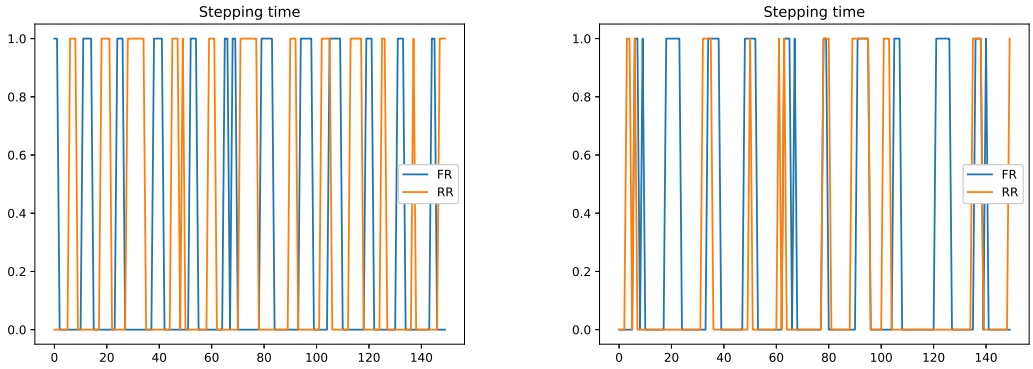

Fast boi - $\phi = 0$, with an average CoT of 0.809 Side boi - $\phi = \frac{\pi}{2}$, with an average CoT of 1.122

Figure 28: Cost of transport (CoT) over time steps for various $\phi$

Both models tend to attend to the limits of the robot in term of velocity as shown in the figure

27, however the figure 26 show that more training for the side boi would be welcome as it did not converge yet.

With more time, we could also try to implement a controllable orientation. To achieve this, we would give a command in the orientation space containing the desired $\phi$, with different $\phi$ in training, and we could probably achieve to control the agent orientation at test time. A similar approach could be tested for the speed but we would need to adjust our reward function, since we are rewarding for maximum speed and not for a desired speed for now.




Fast boi - $\phi = 0$ Side boi - $\phi = \frac{\pi}{2}$

Figure 29: Steps over time steps for various $\phi$

For the characterization of the steps, it is less clear than for the CPGs. As this is a learned approach and even though the steps tend to something similar to a BOUND gait when $\phi = 0$, the time in stance/swing and duration of a steps is not fixed. However, on average, what we obtain for one step is:

- Time duration of 0.122s
- Stance time of 0.027s
- Swing time of 0.095s
- Duty cycle of 0.22

For our 'Side boi', comparison with CPG is irrelevant since we did not implement any gait comparable to this one. For our 'Fast boi', even though the time duration of the steps is similar to the CPG, the duty cycle is about a third as much (0.22 compared to 0.69 for the CPG). This could explain why our bound CPG performed poorly compared to the RL approach. Another point is that the RL can modulate its swing and stance time to achieve stability even at high speed, and this could have possibly been enable for CPG if we added some feedback.

## 3.5 Discussion

For now, we don't think that our approach would be robust to a sim-to-real transfer. Our training was done on a flat terrain and we were trying to push the robot for maximum speed, which is not great in terms of stability. Others potential difficulties are that the motor control will probably behave differently in reality. We have to take into account the delay between the parts, the motor current limitation in terms of power and intensity variations and more generally variations between the agent model and the real robot.

To overcome some of these issues, three aspects could be considered :

- Do a very accurate robot simulation with system identification for the motor characteristics as proposed by Hwangbo and al. [3].
- Train a more robust policy by adding terrain variations and body part modifications.
- Tune the reward to prioritize more stability over speed.

# 4 Conclusion

To conclude we have noticed a great difference between the Central Pattern Generators (CPG) that were designed with the best hyperparameters we could find and the deep reinforcement learning. The RL method allowed to generate not only fast and robust gaits that would have required much more time of hand tuning for the dynamic model, but also to generate new surprisingly highly robust gaits.

# 5 Videos

- Fast boi & Fast boi in competion environnement
- Side boi & Side boi in competion environnement
- Side movement and pace gait & Side movement - PACE gait and competition environnement
- Joint PD controller test

# References

[1] Mostafa Ajallooeian et al. "Central Pattern Generators augmented with virtual model control for quadruped rough terrain locomotion". In: 2013 IEEE International Conference on Robotics and Automation. 2013, pp. 3321–3328. DOI: 10.1109/ICRA.2013.6631040.

[2] Tuomas Haarnoja et al. "Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor". In: CoRR abs/1801.01290 (2018). arXiv: 1801.01290. URL: http://arxiv.org/abs/1801.01290.

[3] Jemin Hwangbo et al. "Learning agile and dynamic motor skills for legged robots". In: Science Robotics 4.26 (Jan. 2019). ISSN: 2470-9476. DOI: 10.1126/scirobotics.aau5872. URL: http://dx.doi.org/10.1126/scirobotics.aau5872.

# Annexes

# List of Figures



# List of Tables