MICRO-507 - Legged Robots

Chevalley Arthur, Mermoud Paco, Pereira Portela Tifanny
Swiss Federal Institute of Technology in Lausanne, EPFL

Quadruped robots applications are very promising but their inherent control is a complex field. Their robustness and adaptability to multiple environments make them very attractive to navigate on rough terrain. Determining an accurate model of such complex robots is challenging and hence model-based controllers are not good enough to control them precisely. To overcome this issue multiple novel methods have been developed and this work focuses on two of them: *Central Pattern Generator* and *Reinforcement Learning*.

I. Introduction

In this project, two controllers are implemented for the *Unitree* A1 robot to wander over unperceived rough terrain.
The first one is inspired from the concept of Central Pattern Generators, CPGs, found in animals. This locomotion controller is implemented as a network of coupled cartesian space Hopf oscillators. By designing the oscillators signals, the trotting, walking, bound, pace, pronk and gallop gaits have been generated. Furthermore, the outputs of the CPG model are augmented with Virtual Model Control, VMC [1] , whose output torques provide attitude and orientation control of the base.

The second one is based on reinforcement learning. The idea behind reinforcement learning is to give the wanted dynamics to the algorithm and let the robot learn the according control strategy by multiple simulations.

A. Quadruped modeling

In this project, *PyBullet* is used to simulate the robot dynamics. This simulation environment is the Python bindings for the physics engine which simulates collision detection as well as soft and rigid body dynamics called *Bullet*.

Figure 1 shows the leg notation and the base reference frame used in *PyBullet* for the A1 robot. On this figure, the abbreviation Li (i = 0,1,2,3) denotes the i^{th} leg. Furthermore, the letters L, R, F, H stand for *L*eft, *R*ight, *F*ront and *H*ind and are used to name the legs.

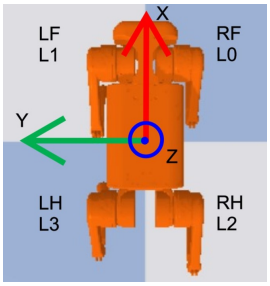
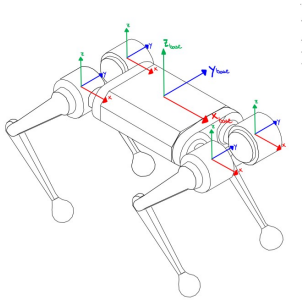
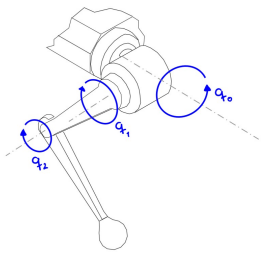


Figure 1: Robot leg numeration and base reference frame

Figure 2a describes the 4 leg frames. This figure highlights the difference between them. Indeed, for example, the foot position in the leg frame of the RF leg is (x,y,z) = (0, -hip_length, -0.25) and the foot position in the leg frame of the LF leg is (x,y,z) = (0, hip_length, -0.25).
As shown in figure 2b, the robot has four legs and each leg has three actuated joints. On this figure q_0 represents the hip joint, q_1 the thigh joint and q_2 the calf joint. By stacking up these joint angles, one can define the i^{th} leg joint angle vector $\mathbf{q}_i = [q_{0,i},\ \ q_{1,i},\ \ q_{2,i}]^T$.



(a) Leg frame on the top of each leg and base axis



(b) Joint angle denomination

Figure 2: Axis placement and joint denomination

The modeling of the quadruped used in this project being done, the locomotion control method using central pattern generators and reinforcement learning will be explained in the following sections.

II. Central Pattern Generator

A. Central pattern generators

The control of locomotion of legged robots can be solved by taking inspiration from the way nature solves the problem through the concept of Central Pattern

Generators, CPGs. CPGs are neural networks located in the spine of vertebrates and controlled by descending paths from the brain.[2] Even though the communication between the brain and the CPGs is low frequency, they are able to generate complex locomotion patterns. Indeed, thanks to the high bandwidth communication between the spinal cord and the musculoskeletal system, the CPGs provide motor primitives for a large range of movements that can be modulated by higher centers.

For robotics applications, CPGs are frequently modeled as coupled oscillators. This type of models are very useful for locomotion control as they can generate smooth rhythmic pattern which are stable against state perturbations, thanks to their inherent limit cycle behaviour.[3] Furthermore, by modulating only simple parameters such as the frequency, the amplitude and the coupling between the oscillators, one can generate complex and synchronised rhythmic patterns. Hence, CPG based models are able to reduce the controller dimensionality. [2]
In the following, a locomotion controller based on a network of coupled oscillators will be proposed.

B. CPG Model and Equations

In this section, a network of coupled cartesian space Hopf oscillators able to generate different gaits is presented. As shown on figure 3, the network proposed has one oscillator per leg and is fully coupled.

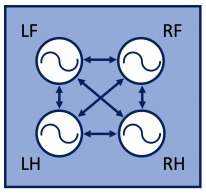


Figure 3: Network couplings

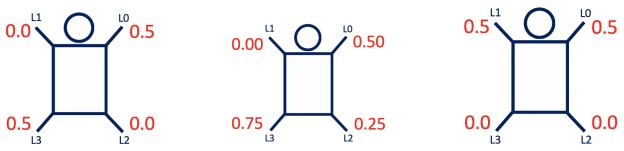
Each oscillator can be characterised by the following equation, where r_i and θ_i are the oscillator amplitude and phase and w_{ij} is the coupling strength between oscillators i and j. Note that in order to simplify the problem, the same weight has been used for every pair of oscillators, i.e $w_{ij} \equiv w$.

$$\begin{aligned} \dot{r}_i &= \alpha(\mu - r_i^2)r_i \\ \dot{\theta}_i &= \omega_i + \sum_{j=0}^{3} r_j w_{ij} sin(\theta_j - \theta_i - \phi_{ij}) \end{aligned} \tag{1}$$

In Eq. 1, $\sqrt{\mu}$ represents the desired amplitude, α is a positive constant controlling the speed of convergence to the limit cycle and ϕ_{ij} is the desired phase offset between oscillators i and j as they are phase-dependent. Finally, ω_i denotes the natural frequency of oscillations.
Every gait pattern of a single leg is composed of a stance phase, i.e when the leg is in contact with the ground, and swing phase, i.e when the leg is in the air. In order to explicitly and independently control the frequencies of these two phases, the natural frequency ω_i is set to take two values depending on the oscillator phase θ_i. Indeed, if $0 \leq \theta_i \leq \pi$, the i^{th} legged will be in swing phase, otherwise it will be in stance phase, i.e when $\pi \leq \theta_i \leq 2\pi$.

1. *Phase offset*

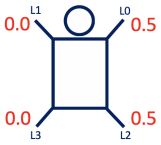
Since CPG models encode the control policies in phase space [2], in order to generate different gaits, one has to define the phase offsets ϕ_{ij} between leg i and j according to the desired gait footfall pattern (i,j = 0,1,2,3). In this project, six different gaits are implemented: the trot, the walk, the bound, the pace, the pronk and the gallop. In the following sections, the gait matrix $\phi_{gait} \in \mathbb{R}^{4x4}$ will be given for each one of these gaits based on their respective footfall pattern highlighted on figure 4.



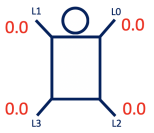
(a) Trot

(b) Walk

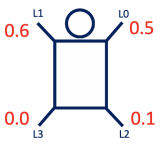
(c) Bound



(d) Pace



(e) Pronk



(f) Gallop

Figure 4: Footfall sequence of six gaits

a. Trot

The first gait implemented is a symmetric one. As shown on figure 4a, the trot is a two-beat gait, where the diagonal legs move together, i.e. (L0-L3) and (L1-L2) are in stance and swing phase always at the same time. Since the phase offsets are defined over a cycle of 2π, the matrix ϕ_{Trot}, defining the desired phase offsets for the trotting gait, can be expressed as in Eq. 2. Indeed, by using a simple linear mapping between the time of contact with the ground within the whole cycle (i.e numbers $\in [0.0,\ 1.0]$ in red on figure 4) and the limits of the phase offset domain (i.e $[0,\ 2\pi]$), one can easily determine ϕ_{Trot}.

$$\phi_{Trot} = \begin{pmatrix} 0 & \pi & \pi & 0 \\ -\pi & 0 & 0 & -\pi \\ -\pi & 0 & 0 & -\pi \\ 0 & \pi & \pi & 0 \end{pmatrix} \tag{2}$$

Note that all the phase offset matrices are defined up to a modulo of 2π. Hence, for the specific case of the trotting gait, $-\phi_{Trot}$ and $|\phi_{Trot}|$ (i.e a phase offset matrix whose elements are defined as the absolute value of the ones of ϕ_{Trot}) lead to the same gait.

b. Walk

The second gait implemented is also a symmetric one. As shown on figure 4b, the lateral sequence walk is a four-beat gait, where the legs follow this sequence: L1, L2, L0, L3 in a regular 1-2-0-3 beat. This gait alternates between having three or two feet touching the ground at the same time.

Similarly to the method applied to find ϕ_{Trot}, the matrix ϕ_{Walk}, defining the desired phase offsets for the walking gait, can be expressed as in Eq. 3.

$$\phi_{Walk} = \begin{pmatrix} 0 & \pi & \frac{3\pi}{2} & \frac{\pi}{2} \\ -\pi & 0 & \frac{\pi}{2} & -\frac{\pi}{2} \\ -\frac{3\pi}{2} & -\frac{\pi}{2} & 0 & -\pi \\ -\frac{\pi}{2} & \frac{\pi}{2} & \pi & 0 \end{pmatrix} \tag{3}$$

c. Bound

As for the previous gait, the third gait is also symmetric. As shown on figure 4c, the bound is a two-beat gait, where the front and hind legs move together (i.e. (L0-L1) and (L2-L3) are in stance and swing phase always at the same time).

Similarly to the method applied to find ϕ_{Trot}, the matrix ϕ_{Bound}, defining the desired phase offsets for the bound gait, can be expressed as in Eq. 4.

$$\phi_{Bound} = \begin{pmatrix} 0 & 0 & -\pi & -\pi \\ 0 & 0 & -\pi & -\pi \\ \pi & \pi & 0 & 0 \\ \pi & \pi & 0 & 0 \end{pmatrix} \tag{4}$$

d. Pace

The fourth gait implemented is the last symmetric one. As shown on figure 4d, the pace is a two-beat gait, where the left and right legs move together (i.e. (L0-L2) and (L1-L3) are in stance and swing phase always at the same time).

Similarly to the method applied to find ϕ_{Trot}, the matrix ϕ_{Pace}, defining the desired phase offsets for the pace gait, can be expressed as in Eq. 5.

$$\phi_{Pace} = \begin{pmatrix} 0 & \pi & 0 & \pi \\ -\pi & 0 & -\pi & 0 \\ 0 & \pi & 0 & \pi \\ -\pi & 0 & -\pi & 0 \end{pmatrix} \tag{5}$$

e. Pronk

The fifth gait implemented is an asymmetric one. As shown on figure 4e, the pronk is a one-beat gait, where all the legs move together (i.e. (L0-L1-L2-L3) are in stance and swing phase always at the same time). Similarly to the method applied to find ϕ_{Trot}, the matrix ϕ_{Pronk}, defining the desired phase offsets for the pronk gait, can be expressed as in Eq. 6.

$$\phi_{Pronk} = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \tag{6}$$

f. Gallop

The last gait implemented is an asymmetric one. As shown on figure 4f, the gallop is a four-beat gait, where the legs follow this sequence: L3, L2, L0, L1 in an irregular 3-2-0-1 beat.

Similarly to the method applied to find ϕ_{Trot}, the matrix ϕ_{Gallop}, defining the desired phase offsets for the gallop gait, can be expressed as in Eq. 7.

$$\phi_{Gallop} = \begin{pmatrix} 0 & -\frac{\pi}{5} & -\frac{6\pi}{5} & -\pi \\ \frac{\pi}{5} & 0 & -\pi & -\frac{4\pi}{5} \\ \frac{6\pi}{5} & \pi & 0 & \frac{\pi}{5} \\ \pi & \frac{4\pi}{5} & -\frac{\pi}{5} & 0 \end{pmatrix} \tag{7}$$

The CPG model used being explained, the mapping between the CPG states and the legs cartesian space will be described in the following section.

2. CPG states mapping

In order to control the robot, one need to map the CPG states to joint commands. In this work, they are mapped into the legs cartesian space first and then using one of the methods described in section II B 3, one can find the torque to apply to the robot.

In the following sections, the mapping from CPG states to the legs cartesian space (x,y,z) will be given for forward, backward, lateral and diagonal locomotion.

a. Forward locomotion

For forward locomotion, the mapping is the following:

$$\begin{aligned} x_{foot} &= -d_{step} r_i cos(\theta_i) \\ y_{foot} &= \begin{cases} y_{hip} & \text{for RF(L0) or RH(L2) leg} \\ -y_{hip} & \text{for LF(L1) or LH(L3) leg} \end{cases} \\ z_{foot} &= \begin{cases} -h + g_c sin(\theta_i) & \text{if } sin(\theta_i) > 0 \\ -h + g_p sin(\theta_i) & \text{otherwise} \end{cases} \end{aligned} \tag{8}$$

Where d_{step} is the step length, h the robot height, g_c the maximum ground clearance during swing and g_p the maximum ground penetration during stance. The ground clearance describes the desired space between the ground and the foot. A higher ground clearance leads to a gait with feet much further from the ground. Note that the amplitude of r_i is between 0 and 1, because the initial amplitudes are randomly sampled in [0,1] and the amplitude target, μ, is set to 1. Hence, since alpha is bigger than 0, the time evolution of the r_i's are described by stable dynamics and they will converge to $\sqrt{\mu} = 1$. Hence, by multiplying r_i by d_{step} to find the x_{foot} position, we can directly control the step length. Otherwise at steady state, when r_i reached μ, we would

only have the possibility to do steps of 1 meter.

By simply looking at the z_{foot} component of equation 8, one can find back the differentiation between stance and swing phases. Indeed, the first line stands for the swing phase, i.e. $0 \leq \theta_i \leq \pi$, where the z position of the foot in the leg frame is simply the difference between the desired robot's height and the ground clearance. This difference is modulated by the sinus of the oscillator's phase. One can see that during swing phase, this sinus leads to a positive value which decreases the distance between the base and the foot.
Similarly, for the stance phase, the foot distance to the base is increased by the ground penetration of the foot. Indeed, this time the sinus of the oscillator's phase is negative and hence during stance, the distance to the base is increased.

b. Backward locomotion
For backward locomotion, the mapping is very similar to the one presented for forward locomotion. The only difference resides in the mapping of the foot's x position in the leg frame:

$$\begin{aligned} x_{foot,backward} &= -x_{foot,forward} \\ y_{foot,backward} &= y_{foot,forward} \\ z_{foot,backward} &= z_{foot,forward} \end{aligned} \tag{9}$$

This mapping has been found by looking closely at the x position equation enabling forward locomotion (see equations 8). Indeed, during swing phase (i.e. $0 \leq \theta_i \leq \pi$), the cosine of the oscillator's phase goes from a positive to a negative value. Hence, the desired x position of the foot goes from a negative to a positive value inducing forward locomotion. Similarly, during stance phase (i.e. $\pi \leq \theta_i \leq 2\pi$), the cosine of the oscillator's phase goes from a negative to a positive value. Hence, the desired x position of the foot goes from a positive to a negative value inducing forward locomotion.
The video "TROT BACKWARD.mp4" shows backward locomotion of the robot with a trotting gait.

c. Lateral locomotion
For lateral locomotion, the mapping is quite similar to the one presented for forward locomotion. The difference can be seen with the following equations:

$$\begin{aligned} x_{foot,lateral} &= 0 \\ y_{foot,lateral} &= y_{foot,forward} \pm x_{foot,forward} \\ z_{foot,lateral} &= z_{foot,forward} \end{aligned} \tag{10}$$

This mapping has been found by looking closely at the equations enabling forward locomotion (see equations 8). Indeed, when moving laterally, the x position of the foot in the leg frame should be set to 0 and the z position should not be altered. Then, to obtain a lateral motion along the positive y direction of the base frame, one needs to add an oscillatory movement given by the $x_{foot,forward}$ term to the hip offset (i.e $y_{foot,forward}$). Similarly, to obtain a lateral motion along the negative y direction of the base frame, one needs to subtract an oscillatory movement given by the $x_{foot,forward}$ term to the hip offset, i.e $y_{foot,forward}$.
The video "TROT LATERAL LEFT.mp4" shows lateral locomotion of the robot with a trotting gait, when one uses a positive sign in Eq. 10. The video "TROT LATERAL RIGHT.mp4" shows lateral locomotion of the robot with a trotting gait, when one uses a negative sign in Eq. 10.

d. Diagonal locomotion
The lateral and forward motion equations (see equations 8 and 10) can be merged to induce diagonal locomotion as follows:

$$\begin{aligned} x_{foot,diagonal} &= \alpha \cdot x_{foot,forward} \\ y_{foot,diagonal} &= y_{foot,forward} \pm \beta \cdot x_{foot,forward} \\ z_{foot,diagonal} &= z_{foot,forward} \end{aligned} \tag{11}$$

Where α, $\beta \in [-1, 1]$ are parameters modulating the direction of locomotion. One can see that forward locomotion is a special case of diagonal locomotion, since it can be obtained with the pair $(\alpha, \beta) = (1, 0)$. Similarly, backward locomotion is a special case of diagonal locomotion, since it can be obtained with the pair $(\alpha, \beta) = (-1, 0)$. Finally, lateral locomotion is also a special case of diagonal locomotion, since it can be obtained with the pair $(\alpha, \beta) = (0, \pm 1)$.
Hence, equations 11 provide an elegant way to describe locomotion with a great variety of direction achieved with this specific CPG mapping.
The video "TROT DIAGONAL.mp4" shows a purely forward diagonal locomotion (i.e $\alpha = \beta = 1$) of the robot with a trotting gait.

Note that equations 11 only change the direction of locomotion of the robot, but not the orientation of its base. This problem will be tackled in section II C 2 b with the proposition of a virtual model controller.

3. Quadruped control

In order to convert the feet positions expressed in the leg cartesian space to torques to be applied to the robot, a joint PD controller, a cartesian PD controller or a combination of both can be used.

Before diving more deeply into the details of these controllers, one must define various notations and formulas which will be useful throughout the rest this report.
First of all, the position of each foot in its respective leg frame must be expressed as a function of this leg joint angles. This is achieved by computing the forward kinematics of a single robot's leg which is denoted by the function $f(\cdot)$. Hence, the position of each foot in its respective leg, $\boldsymbol{p}$, is defined as $\mathbf{p} = f(\mathbf{q})$ with $\boldsymbol{q}$ the joint angles of the studied leg.
Since the CPG states are mapped into the legs Cartesian space, one need to use inverse kinematics denoted by f^{-1} to determine the target joint angles: $q_d = f^{-1}(\mathbf{p})$.

Then, based on the forward kinematics equation, the velocity $\boldsymbol{v}$ of each foot in its respective leg frame can be determined with the following formula:

$$\boldsymbol{v} = \frac{dp}{dt} = \dot{\boldsymbol{p}} = \frac{df}{dq}(q) \cdot \frac{dq}{dt} = \mathbf{J}(\mathbf{q}) \cdot \dot{\boldsymbol{q}} \tag{12}$$

where the matrix $\mathbf{J}(\mathbf{q}) \in \mathbb{R}^{3x3}$ represents the jacobian.

Finally, using the jacobian one can compute the joint torques needed, to apply a desired force at the foot. This is done by computing $\mathbf{J}^T(\mathbf{q}) \cdot \mathbf{F}_{desired}$.

With the help of these formulas, the details of the 3 aforementioned controllers will be described in the following sections.

a. Joint PD controller
The first option is to use a simple joint PD controller by monitoring the joint position and velocity errors. The joint torques to apply to the robot are the ones expressed in equation 13.

$$\tau_{joint} = \mathbf{K}_{p,Joint} * (\mathbf{q}_d - \mathbf{q}) + \mathbf{K}_{d,Joint} * (\dot{\mathbf{q}}_d - \dot{\mathbf{q}}) \in \mathbb{R}^3 \tag{13}$$

Where $\mathbf{q}_d$ are the target joint angles, $\dot{\mathbf{q}}_d$ are the target joint velocities, and $\mathbf{q}$ and $\dot{\mathbf{q}}$ are the current joint angles and joint velocities in the leg frame, respectively. the $\mathbf{K}_{p,Joint} \in \mathbb{R}^3$ and $\mathbf{K}_{d,Joint} \in \mathbb{R}^3$ are vectors of proportional and derivative gains and the $*$ sign represents element-wise multiplication.
b. Cartesian PD controller
The second option is to use a cartesian PD controller by monitoring the foot position and velocity errors in the leg frame. This method is conducted in two steps. The first step defines the desired force to be applied to the foot in the leg frame based on the foot position and velocity errors. The second step uses the transpose of the jacobian to map this desired force to torques.
These steps can be seen in equation 14

$$\begin{aligned} \tau_{Cartesian} = \mathbf{J}^T(\mathbf{q}) \cdot [\mathbf{K}_{p,Cartesian} \cdot (\boldsymbol{p}_d - \boldsymbol{p}) + \\ \mathbf{K}_{d,Cartesian} \cdot (\boldsymbol{v}_d - \boldsymbol{v})] \in \mathbb{R}^3 \end{aligned} \tag{14}$$

where $\mathbf{J}(\mathbf{q})$ is the jacobian evaluated at the current joint positions $\mathbf{q}$, and $\mathbf{K}_{p,Cartesian} \in \mathbb{R}^{3x3}$ and $\mathbf{K}_{d,Cartesian} \in \mathbb{R}^{3x3}$ are diagonal matrices of proportional and derivative gains.
c. Cartesian and Joint PD controllers
Finally, as mentioned previously, the third option is to combine the effects of the joint PD controller and the cartesian PD controller. This controller simple add both contributions as shown in equation 15.

$$\tau_{Combined} = \tau_{Joint} + \tau_{Cartesian} \in \mathbb{R}^3 \tag{15}$$

C. Possible extensions

In this section, in order to improve the performances of the simple controllers described previously, several possible extensions are reported.

First of all, the description of the grid search algorithm used on several hyper parameters is given. Then, the Virtual Model Control (VMC) method, used to augment the CPG model allowing attitude and orientation control of the base, is explained. Finally, a few ideas regarding the possible sensory feedback methods to be added to the open-loop CPG model are expressed.

1. Grid search

First of all, a grid search on several parameters is conducted to improve the overall performance of the 6 different gaits. These parameters and their respective search space are the following:

1. The foot swing height g_c has been varied between 0.04 and 0.06 by steps of 0.01
2. The foot stance penetration g_p into the ground has been varied between 0.01 and 0.02 by steps of 0.01
3. The nominal robot height h has been varied between 0.23 and 0.28 by steps of 0.01
4. The step length d_{step} has been varied between 0.02 and 0.04 by steps of 0.01
5. The coupling weight w has been varied between 0.8 and 1.2 by steps of 0.2
6. The convergence rate to the limit cycle α has been varied between 40 and 60 by steps of 10
7. The joint PD gain $\mathbf{G}_{p,Joint}, \mathbf{G}_{d,Joint}$ have been varied respectively between 140 and 160 by steps of 10 and between 1.5 and 2.5 by steps of 0.5
8. The cartesian PD gains $\mathbf{G}_{p,Cartesian}, \mathbf{G}_{d,Cartesian}$ have been varied respectively between 2400 and 2500 by steps of 50 and between 36 and 44 by steps of 4

Note that the $\mathbf{K}_{p,Joint}, \mathbf{K}_{d,Joint}$ and the $\mathbf{K}_{p,Cartesian}, \mathbf{K}_{d,Cartesian}$ are simplified as

- $\mathbf{K}_{p,Joint} = \begin{bmatrix} 2 \cdot \mathbf{G}_{p,Joint} & \mathbf{G}_{p,Joint} & \mathbf{G}_{p,Joint} \end{bmatrix}^T$
- $\mathbf{K}_{d,Joint} = \begin{bmatrix} \mathbf{G}_{d,Joint} & \frac{\mathbf{G}_{d,Joint}}{4} & \frac{\mathbf{G}_{d,Joint}}{4} \end{bmatrix}^T$
- $\mathbf{K}_{p,Cartesian} = 2 \cdot \mathbf{G}_{p,Cartesian} \cdot Diag([1,1,1])$
- $\mathbf{K}_{d,Cartesian} = \mathbf{G}_{d,Cartesian} \cdot Diag([1,1,1])$

One simulation is run for each combination of theses parameters using the Cartesian and Joint PD controller presented previously. For each simulation, one compute the objective defined as in Eq. 16.

$$objective = (w_{bv} \cdot o_{bv} + w_{fw} \cdot o_{fw} + w_{fb} \cdot o_{fb}) \cdot w_{fall} \tag{16}$$

The first terms $w_{bv} = 50$ and o_{bv} are respectively the weight and reward of the base velocity. The term o_{bv} takes into account the mean velocity of the base during the simulation. The product $w_{bv} \cdot o_{bv}$ has the smallest contribution in the *objective* term due to the fact that the base velocity was generally limited to 1 [m/s].

The second terms $w_{fw} = 50$ and o_{fw} are respectively the weight and reward of the total forward displacement. The term o_{fw} takes into account the total forward

displacement of the base during the simulation. The product $w_{fw} \cdot o_{fw}$ has generally a bigger contribution compared the previous element since the displacement can reach several tens of meters.

The third terms w_{fb} and o_{fb} are respectively the weight and reward of the foot booleans. This term tries to emphasise the good execution of the gait. At each time step, it compares the ground contact booleans of the feets that should be synchronised. It adds one to a counter every time they are equal and zero otherwise. The weight w_{fb} is defined as follows:

$$w_{fb} = \begin{cases} 0, \ if \ gait \ is \ walk \ or \ gallop \\ \frac{1}{1000}, \ otherwise \end{cases} \tag{17}$$

The weight is set to zero for walk and gallop since they do not have any pair of synchronised legs. For the rest of the gaits, the relatively small weight has a big contribution since the accumulation of the reward can reach hundreds of thousands.

And finally the last term w_{fall} penalises simulations where the robot falls or jumps too high. This term is defined as follows:

$$w_{fall} = \begin{cases} 0, \ if \ z_{base} < 0.13 \ or \ z_{base} > 0.65 \\ 1, \ otherwise \end{cases} \tag{18}$$

Finally, the best set of parameters is defined as the one leading to the highest objective function.

2. Virtual Model Control

Until this point of the report, the framework proposed uses only open-loop commands sent to the joints. Indeed, there is no feedback using the current state of the robot to change the desired behaviour. In this section, the outputs of the open-loop CPG controller are augmented using Virtual Model Control. VMC is a motion control framework that uses simulations of virtual components to generate desired joint torques by creating the illusion that the simulated components are connected to the real robot. This method has proved to be useful for posture control on unperceived rough terrain locomotion with dynamic gaits[4].

Locomotion over rough terrain can lead to unwanted body rotations (mainly roll and pitch rotations) which can lead to a loss of balance. To tackle this problem, a VMC controller to adjust the attitude can be designed. The detailed equations of such a controller can be found in section II C 2 a. Furthermore, a VMC controller can be designed to make the robot follow a prefered direction given by a human operator in real-time for example. The detailed equations of such a controller can be found in section II C 2 b.

The methods for attitude and direction control using VMC have been inspired by the work of Mostafa Ajallooeian and al. [1].

a. VMC - attitude control

For attitude control, four virtual and identical springs are attached between pair of points ((G_i,H_i) i = 0,1,2,3) on the robot's base (H-plane) and a horizontal plane (G-plane) w.r.t to the world coordinates as shown in figure 5. The plane G and H are connected to the center of the robot's trunk (i.e they are centered at the point (x,y,z)= (0,0,0) in figure 5). The role of these virtual springs is to reject pitch and roll variations from the target horizontal plane (G-plane) by applying a vertical force only on the points on the H-plane (i.e H_1, H_2, H_3 and H_4).

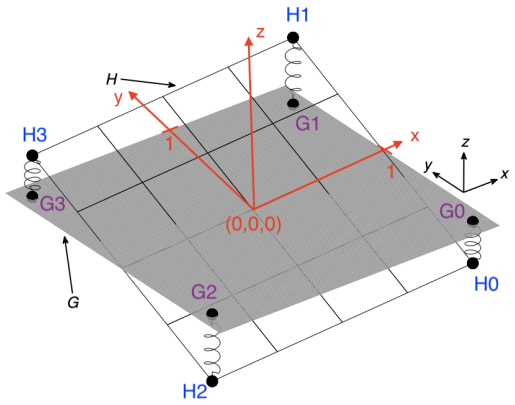


Figure 5: Virtual springs for attitude control and the two virtual planes (H,G) used to compute the VMC control commands [1]

According to the base reference frame described in section I, the coordinates of the points on the H-plane (i.e H_1, H_2, H_3 and H_4) are the following: $H_0 = (1,-1,0)^T$ $H_1 = (1,1,0)^T$ $H_2 = (-1,-1,0)^T$ $H_3 = (-1,1,0)^T$. These coordinates can be concatenated into of matrix H as follows:

$$\mathbf{H} = [H_0, H_1, H_2, H_3] = \begin{bmatrix} 1 & 1 & -1 & -1 \\ -1 & 1 & -1 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{19}$$

The coordinates of the points on the G-plane (i.e $\mathbf{G} = [G_0, G_1, G_2, G_3] \in \mathbb{R}^{3x4}$) in the base reference frame can be found using the following formula, where $\mathbf{R(q)} \in \mathbb{R}^{3x3}$ is the base orientation matrix w.r.t world coordinates.

$$\mathbf{G} = \mathbf{R(q)} \cdot \mathbf{H} \in \mathbb{R}^{3x4} \tag{20}$$

The vector of vertical virtual forces ($F_z = [F_{z,0}, F_{z,1}, F_{z,2}, F_{z,3}] \in \mathbb{R}^{1x4}$) to be applied at the points on the H-plane (i.e H_1, H_2, H_3 and H_4) is expressed in equation 21:

$$\mathbf{F}_z = k_{att,z} \cdot \begin{bmatrix} 0 & 0 & 1 \end{bmatrix} \mathbf{G} \in \mathbb{R}^{1x4} \tag{21}$$

where $k_{att,z} \in \mathbb{R}$ represents the stiffness of the springs. From these vertical forces, one can compute the force in 3 dimensions ($\mathbf{F}_i \in \mathbb{R}^3$) to be applied at the points on the H-plane (i = 0,1,2,3):

$$\mathbf{F}_i = \begin{bmatrix} F_{x,i} \\ F_{y,i} \\ F_{z,i} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ F_{z,i} \end{bmatrix} \tag{22}$$

As mentioned previously, for each leg, this virtual force $\mathbf{F}_i$ has to be applied at each point H_i, which are located close to the robot's hips. The trick is to generate $-\mathbf{F}_i$ at each foot instead.
Hence, for each leg, to convert the virtual force $-\mathbf{F}_i$ to be generated at the foot to torques, the formula 23 is used.

$$\tau_{att,i} = -\mathbf{J}_i^T(\mathbf{q}_i)\mathbf{F}_i \tag{23}$$

Each leg is responsible to generate its own virtual force separately. The torques ($\tau_{att,i}$ i= 0,1,2,3) are only applied during stance phase and this VMC attitude controller is only active when more than one leg is in stance phase for stability reasons.

b. VMC - direction control

In this section, a locomotion direction controller is implemented using virtual forces to compensate for wrong heading direction. Contrarily to attitude control, the virtual forces are applied only in the horizontal plane for direction control. The target yaw angle of the base in the world frame is given to the robot by descending modulation. The user can hence easily change the commanded direction in real-time. Then the controller generates a momentum to correct the robot orientation as a function of the difference between the target and the current yaw angle in world frame, i.e $\Delta\psi = \psi_{target} - \psi_{current}$.

The virtual forces to be applied on the robots hips can be computed as follows.

$$\mathbf{F}_0 = \mathbf{F}_1 = \begin{bmatrix} cos(\Delta\psi) \\ sin(\Delta\psi) \\ 0 \end{bmatrix}, \mathbf{F}_2 = \mathbf{F}_3 = \begin{bmatrix} -cos(\Delta\psi) \\ -sin(\Delta\psi) \\ 0 \end{bmatrix} \tag{24}$$

For clarity purposes, an example on figure 6 is provided. As one can see, the angle difference is positive in this case and hence the controller produces a counterclockwise momentum M, resulting from the $\mathbf{F}_i$'s, to compensate for this angle difference.
As mentioned previously, instead of applying the virtual forces on the robot hips, one can generate $-\mathbf{F}_i$ at each foot instead.

Hence, for each leg, to convert the virtual force $-\mathbf{F}_i$ to be generated at the foot to torques, the formula 25 is used.

$$\tau_{orient,i} = -\mathbf{J}_i^T(\mathbf{q}_i)\mathbf{F}_i \tag{25}$$

Again, each leg is responsible to generate its own virtual force separately. The torques ($\tau_{orient,i}$ i= 0,1,2,3) are applied during stance phase and this VMC orientation controller is only active when more than one leg are in stance phase for stability reasons.
Finally, in order to reproduce a specific gait while controlling the orientation and attitude of the quadruped, the total torque be applied to each joint of the robot is:

$$\tau_{total,i} = \tau_{gait,i} + \tau_{att,i} + \tau_{orient,i} \tag{26}$$

Where $i = 0, 1, 2, 3$ and $\tau_{gait,i}$ represents the torque resulting from one of the three methods presented in section II B 3.

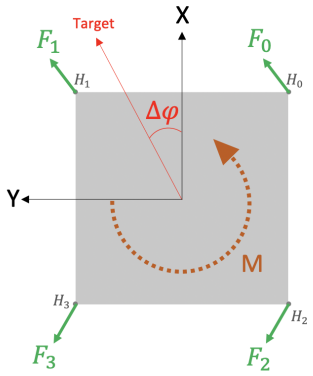


Figure 6: Forces diagram for orientation control using VMC

3. Sensory feedback

Another way to improve the robustness of the open-loop CPG controllers is to change directly the CPG equations by adding a feedback term. In the rest of this section, a description of a possible local feedback controller is given. Using the Tegotae-based appoach, D. Owaki and al. proposed a local sensory feedback mechanism that improved the interlimb coordination of a hexapod robot.[5] They define the Tegotae concept as the extent to which a perceived reaction matches an expectation of a controller. To quantify the Tegatae, they designed a function of the control variables, that becomes more positive when enhanced Tegotae is detected. Based on this function, they define the local sensory feedback so that the control law aims to increase the amount of Tegotae received.

Thanks to their novel Tegotae-based approach, they provided a systematic manner to design a local sensory feedback for a decentralized interlimb coordination mechanism. Hence, their method could be easily implemented

on the A1 quadruped robot used in this project to improve its interlimb coordination and to allow gait transition according to locomotion speed for example. Unfortunately, due to time constraints, their Tegotae-based approach has not been implemented in this work.

D. Metrics

In order to evaluate and compare the different gaits, the following metrics are used.
The **time duration in stance** of one step T_{stance} and the **time duration in swing** of one step T_{swing}, both expressed in seconds.
The **duty cycle** T defined as the time needed to go trough a complete cycle expressed in seconds: $T = T_{stance} + T_{swing}$.
The dimension-less **duty ratio** D defined as the ratio between the time duration in stance phase over the time duration in swing phase: $D = \frac{T_{stance}}{T_{swing}}$
Finally, in order to compare the energy efficiency of the different modes of locomotion, the **Cost Of Transport** (CoT), whose formula is given in Eq. 27, is used.

$$CoT = \frac{1}{T_{Sim}} \cdot \frac{1}{m \cdot g \cdot v_{base}} \sum_{k=0}^{T_{sim}} \left(\sum_{i=1}^{N_{Legs}} \sum_{j=1}^{N_{joints}} |\dot{q}_{i,j} \cdot \tau_{i,j}| \right)_k \tag{27}$$

In Eq. 27, the index i iterates over the legs, up to N_{Legs}, and the index j iterates over the joint angles up to N_{joints}. In addition, $\dot{q}_{i,j}$ and $\tau_{i,j}$ represents respectively the joints angular velocity and mechanical torque of the i^{th} leg and j^{th} joint at discrete time step k. Furthermore, T_{sim} denotes the number of discrete time steps of the entire simulation, $m = 12\ [kg]$ is the robot mass and $g = 9.81\ [m/s^2]$ is the gravitational acceleration. Finally, v_{base} is the average velocity base over the entire simulation and is calculated as follows: $v_{base} = \frac{||\vec{x}_f - \vec{x}_0||_2}{T_{sim} dt}$, where $dt = 0.001[s]$ is the time step, and $\vec{x}_0 \in \mathbb{R}^3$ and $\vec{x}_f \in \mathbb{R}^3$ are respectively the initial position and final positions of the quadruped in meters.

E. Results

1. CPG results

In this chapter, the results regarding the CPG part of the report are given. First of all, for each gait, a plot of the CPG states as well as performance comparisons between the joint and Cartesian PD controllers are provided. Then a short discussion on central pattern generators and virtual model control is presented to conclude the first part of the report.

For all the following figures, the desired values are plotted in blue and the measured ones in orange.

a. Trot

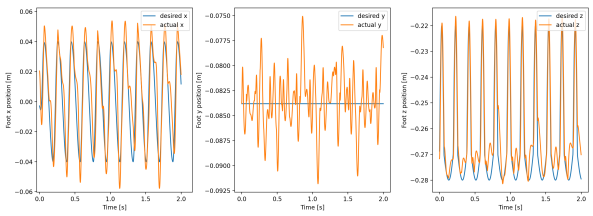
Regarding the Trot gait, the parameters leading to the highest score of the grid search algorithm are the following:

- $\omega_{swing} = 16\pi[rad/s]$
- $\omega_{stance} = 8\pi[rad/s]$
- $g_c = 0.05[m]$
- $g_p = 0.01[m]$
- $h = 0.27[m]$
- $d_{step} = 0.03[m]$
- $\omega = 1.2$
- $\alpha = 60$
- $\mathbf{K}_{p,Cartesian} = 5000 \cdot Diag([1,1,1])$
- $\mathbf{K}_{d,Cartesian} = 40 \cdot Diag([1,1,1])$
- $\mathbf{K}_{p,Joint} = \begin{bmatrix} 300 & 150 & 150 \end{bmatrix}^T$
- $\mathbf{K}_{d,Joint} = \begin{bmatrix} 2 & 0.5 & 0.5 \end{bmatrix}^T$

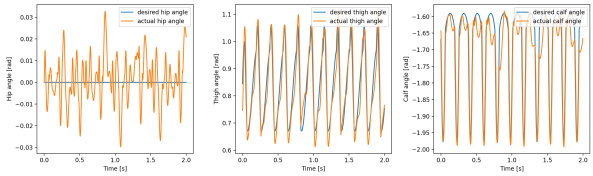
Some of the parameters were then changed by hand to have a qualitatively better looking gait. The modified parameters are the following:

- $d_{step} = 0.04[m]$
- $\alpha = 70$
- $\mathbf{K}_{d,Cartesian} = 44 \cdot Diag([1,1,1])$
- $\mathbf{K}_{p,Joint} = \begin{bmatrix} 320 & 160 & 160 \end{bmatrix}^T$
- $\mathbf{K}_{d,Joint} = \begin{bmatrix} 2.5 & 0.625 & 0.625 \end{bmatrix}^T$

For this set of parameters, the effect of using a joint PD controller, a Cartesian PD controller or both controllers were tested on the overall performance of the quadruped. Figure 7 shows the tracking performance of the controller using joint PD only. As one can see on figure 7a, the tracking is not perfect. Indeed, on average, one can observe an offset reaching 0.01 [m] for the X position, 0.006 [m] for the Y position and some fluctuations on the Z axis, specially in stance phase, reaching an average of 0.015 [m]. In addition, as shown on figure 7b, on average, one can observe an offset reaching 0.015 [rad] for the hip angle, 0.05 [rad] for the thigh angle and some fluctuations for the calf angle specially in stance phase, reaching an average of 0.08 [rad].



(a) Desired foot positions (x,y,z) vs. actual foot positions with joint PD for L0

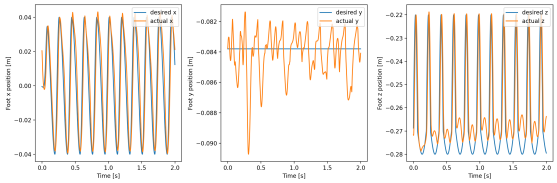


(b) Desired joint angles (q_0, q_1, q_2) vs. actual joint angles with joint PD for L0

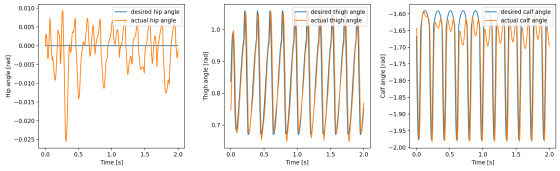
Figure 7: Tracking performance with joint PD only for L0

Figure 8 shows the tracking performance of the controller using Cartesian PD only. As one can see on figure 8a, the tracking is not perfect, but improved compared to the performance achieved by the joint PD controller. Indeed, on average, we can observe an offset reaching 0.002 [m] for the y position, almost no offset for the x position and some fluctuations for the z position specially in stance phase, reaching an average of 0.01 [m]. Note that these fluctuations are more stable and slightly smaller on average compared to the ones obtained with the joint PD controller. In addition, as shown on figure 8b, on average, we can observe an offset reaching 0.0075 [rad] for the hip angle, almost no offset for the thigh angle and some fluctuations for the calf angle specially in stance phase, reaching an average of 0.06 [rad]. Note that, again, these fluctuations are more stable and slightly smaller on average compared to the ones obtained with the joint PD controller.

Figure 9 shows the tracking performance of the controller using joint and Cartesian PD. As one can see on figure 9a, the tracking is not perfect, but improved compared to the performance achieved by the joint PD controller. Indeed, on average, we can observe an offset reaching 0.003 [m] for the y position, almost no offset for the x position and some fluctuations for the z position specially in stance phase, reaching an average of 0.008 [m]. Note that these fluctuations are very similar to the ones obtained with the Cartesian PD controller. In addition, as shown on figure 9b, on average, we can observe an offset reaching 0.01 [rad] for the hip angle, almost no offset for the thigh angle and some fluctuations for the calf angle specially in stance phase, reaching an average of 0.04 [rad].



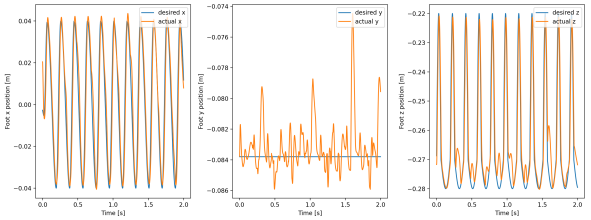
(a) Desired foot positions (x,y,z) vs. actual join angles with Cartesian PD for L0



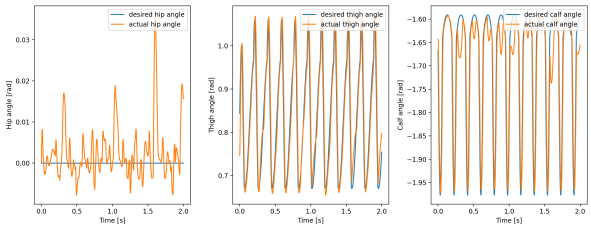
(b) Desired joint angles (q_0, q_1, q_2) vs. actual joint angles with Cartesian PD for L0

Figure 8: Tracking performance with Cartesian PD only for L0

For each one of these controllers, the CoT and the base velocity v_{base} have been calculated. Table I summarises the results for the trot gait. In this table, (Δx, Δy, Δz) denote the average foot position tracking errors and (Δq_0, Δq_1, Δq_2) represent the average joint angles tracking errors (hip, thigh and calf).



(a) Desired foot positions (x,y,z) vs. actual joint angles with joint and Cartesian PD for L0



(b) Desired joint angles (q_0, q_1, q_2) vs. actual joint angles with joint and Cartesian PD for L0

Figure 9: Tracking performance with joint PD and Cartesian PD for L0

Trot gait			
	Joint PD	Cartesian PD	Both
Δx [m]	0.01	0	0
Δy [m]	0.006	0.002	0.003
Δz [m]	0.015	0.01	0.008
Δq_0[rad]	0.015	0.0075	0.01
Δq_1[rad]	0.05	0	0
Δq_2[rad]	0.08	0.06	0.04
v_{base}[m/s]	0.4087	0.6561	0.6283
CoT[-]	3.7604	1.7699	2.0536

Table I: Average tracking errors over time, velocity of the base and CoT for the Trot gait

From table I, as expected, one can see that using solely a joint PD controller leads to bad results. Furthermore, the results show that, for the trot gait, using both controllers or the Cartesian PD alone leads to similar results regarding tracking errors. However, the Cartesian PD controller achieves a smaller CoT, while providing a bigger velocity of the base and is hence the best controller for the trot gait with the set of parameters proposed.

Since, the best controller seems to be the one using Cartesian PD only, a plot of the CPG states $r, \theta, \dot{r}, \dot{\theta}$ for this controller and for each leg is computed. Figure 10 shows the result. As one can see, for each leg, the CPG amplitude r (blue curve) reaches quickly the desired amplitude $\sqrt{\mu} = 1$. For each leg, the CPG

amplitude rate $\dot{r}$ (green curve) highlights this behaviour by exhibiting a quick and big peak at the beginning of the simulation, that is then quickly stabilised with time. The periodic behaviour of the phase and the phase rate (orange and red curves) show the rapid convergence of the CPG states to the limit cycle. Furthermore, these plots highlight the ratio between swing and stance phase ($R = \frac{\omega_{swing}}{\omega_{stance}} = \frac{16\pi}{8\pi} = 2$). Indeed, the swing phase is twice as fast as the stance phase. On figure "LF(L1)" for example, the phase rate curve (red curve) has two plateaus. The first one, which is the highest one (values around 50) and has a negative slope, represents the swing phase. The second one, which is the smallest one (values around 25) and has a positive slope, represents the the stance phase and lasts two times more time than the swing plateaus.

Figure 10 shows the synchronicity of the diagonal legs pairs. Indeed, the CPG states evolution over time of the L1 leg are very similar to the ones of the L2 leg (see "LF(L1)" and "RH(L2)" plots). The same remark is valid for the L0 and L3 leg (see "RF(L0)" and "LH(L3)" plots).

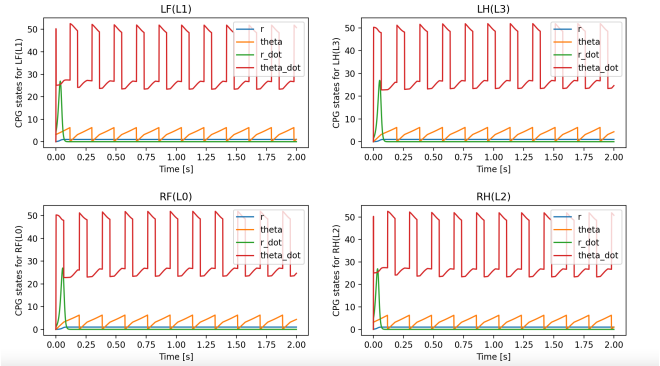


Figure 10: CPG states obtained with a Cartesian PD controller for each leg and for the Trot gait

Regarding the metrics, for the Trot gait using a Cartesian PD controller, the average time duration in swing phase (across the simulation and the 4 legs) reaches 0.096 [s], while the average time duration in stance phase reaches 0.09 [s], leading to a duty cycle mean of 0.186 [s]. The duty ratio mean reaches 0.928. Since the target duty ratio for the trotting gait is 1, we have an error of 7.2%. Figure 11 highlights the fact that the parameters we found are not the ideal ones. Indeed, according to figure 4a, the feet contact booleans of the legs pairs (L1,L2) and (L0,L3) should be perfectly superimposed over time, which is not the case on figure 11.
Finally, table II shows the results of the fastest and slowest Trot gaits achieved with the parameters proposed in this section and using a Cartesian PD controller. In order to increase (respectively decrease) the speed of the base, the ratio between swing and stance phase was kept constant (i.e $R = 2$), but the swing and stance natural frequencies of oscillations were changed.
From table II, one can see that the higher the pair

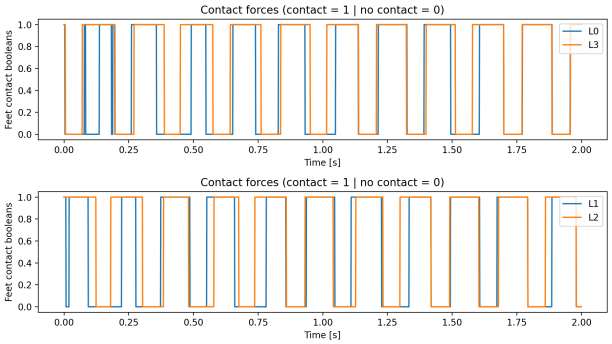


Figure 11: Feet contact booleans for the Trot gait

Trot gait		
	Fast	Slow
ω_{swing}[rad/s]	32	8
ω_{stance}[rad/s]	16	4
T [s]	0.09	0.366
D [-]	1.24	1.18
v_{base}[m/s]	1.308	0.26
CoT[-]	2.74	2.15

Table II: Average duty cycle T, duty ratio D, velocity of the base and CoT for the slowest and fastest Trot gait

(ω_{swing}, ω_{stance}), the faster the gait and the smaller the duty cycle T. However, these results come at the cost of increasing the CoT and degrading the quality of the gait. Indeed, the faster the gait, the more the duty ratio D diverges from the ideal case (i.e D = 1). Furthermore, one can see that the gait closer to the ideal case is the one found previously (see detailed results in table I) with a base velocity of 0.6561 [m/s] and which led to the smallest CoT for the Trot gait.
The video "TROT_FAST_1.308.mp4" shows the results of the fastest Trot gait achieved. Similarly, the video "TROT_SLOW_0.26.mp4" shows the results of the slowest Trot gait achieved.

b. Pace

Regarding the pace gait, the parameters leading to the highest score of the grid search algorithm are the following:

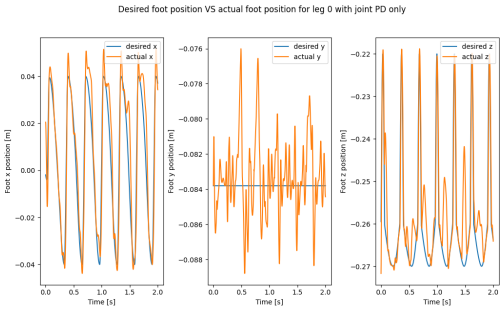
- $\omega_{swing} = 16\pi[rad/s]$
- $\omega_{stance} = 4\pi[rad/s]$
- $g_c = 0.05[m]$
- $g_p = 0.01[m]$
- $h = 0.27[m]$
- $d_{step} = 0.04[m]$
- $\omega = 1.2$
- $\alpha = 60$
- $\mathbf{K}_{p,Joint} = \begin{bmatrix} 300 & 150 & 150 \end{bmatrix}^T$
- $\mathbf{K}_{d,Joint} = \begin{bmatrix} 3 & 0.75 & 0.75 \end{bmatrix}^T$
- $\mathbf{K}_{p,Cartesian} = 5000 \cdot Diag([1, 1, 1])$

- $\mathbf{K}_{d,Cartesian} = 40 \cdot Diag([1, 1, 1])$

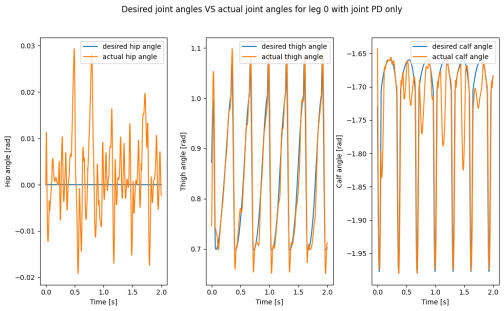
Some of the parameters were then changed by hand to have a qualitatively better looking gait. The modified parameters are the following:

- $g_c = 0.04[m]$
- $h = 0.26[m]$
- $\mathbf{K}_{p,Joint} = \begin{bmatrix} 320 & 160 & 160 \end{bmatrix}^T$
- $\mathbf{K}_{d,Joint} = \begin{bmatrix} 2.5 & 0.625 & 0.625 \end{bmatrix}^T$
- $\mathbf{K}_{d,Cartesian} = 44 \cdot Diag([1, 1, 1])$

For this set of parameters, the effect of using a joint PD controller, a Cartesian PD controller or both controllers were tested on the overall performance of the quadruped. Figure 12 shows the tracking performance of the controller using joint PD only. As one can see on figure 12a, the tracking is not perfect. Indeed, on average, one can observe an offset reaching 0.016 [m] for the X position, 0.005 [m] for the Y position and some fluctuations on the Z axis, specially in stance phase, reaching an average of 0.015 [m]. In addition, as shown on figure 12b, on average, one can observe an offset reaching 0.03 [rad] for the hip angle, 0.07 [rad] for the thigh angle and some fluctuations for the calf angle specially in stance phase, reaching an average of 0.09 [rad].
Figure 13 shows the tracking performance of the controller using Cartesian PD only. As one can see on figure 13a, the tracking is not perfect, but improved compared to the performance achieved by the joint PD controller. Indeed, on average, we can observe an offset reaching 0.004 [m] for the y position, almost no offset for the x position and some fluctuations for the z position specially in stance phase, reaching an average of 0.01 [m]. Note that these fluctuations are more stable and slightly smaller on average compared to the ones obtained with the joint PD controller. Indeed, regarding the y position, the average is roughly the same, but except for the foot position at each stance phase, there are much less oscillations. In addition, as shown on figure 13b, on average, we can observe an offset reaching 0.025 [rad] for the hip angle, almost no offset for the thigh angle and some fluctuations for the calf angle specially in stance phase, reaching an average of 0.09 [rad]. Note that, again, these fluctuations are more stable and slightly smaller on average compared to the ones obtained with the joint PD controller.
Figure 14 shows the tracking performance of the controller using joint and Cartesian PD. As one can see on figure 14a, the tracking is not perfect, but improved compared to the performance achieved by the previous controller. Indeed, on average, we can observe almost no offset for the x position, quite some oscillations for the y position, in the order of 0.003 [m], and some fluctuations for the z position specially in stance phase, reaching an average of 0.01 [m]. Note that these fluctuations are very similar to the ones obtained with the Cartesian PD controller for the x, y and z foot positions. It is also interesting to note that the z error has the same mean but less fluctuations. In addition, as shown on figure 14b, on average, we can observe an offset around 0.02 [rad] for the hip angle, for the thigh angle almost no offset and some important fluctuations for the calf angle specially in stance phase, reaching an average of 0.08 [rad].



(a) Desired foot positions (x,y,z) vs. actual foot positions with joint PD for L0



(b) Desired joint angles (q_0, q_1, q_2) vs. actual joint angles with joint PD for L0

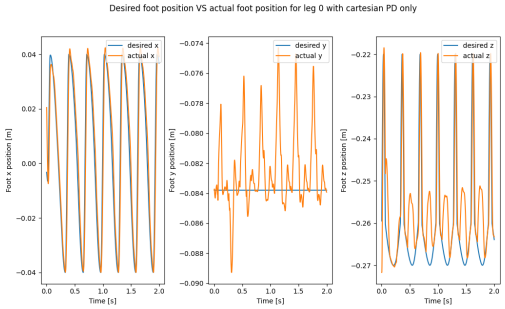
Figure 12: Tracking performance with joint PD only for L0

For each one of these controllers, the CoT and the base velocity v_{base} have been calculated. Table III summarises the results for the pace gait.

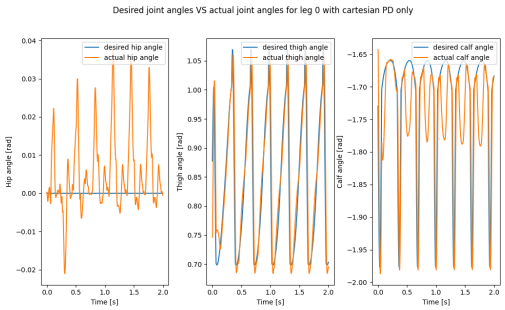
Pace gait			
	Joint PD	Cartesian PD	Both
Δx [m]	0.016	0	0
Δy [m]	0.005	0.004	0.003
Δz [m]	0.015	0.01	0.01
Δq_0[rad]	0.03	0.025	0.02
Δq_1[rad]	0.07	0	0
Δq_2[rad]	0.09	0.09	0.08
v_{base}[m/s]	0.2484	0.3471	0.3306
CoT[-]	3.2630	1.6609	1.9649

Table III: Average tracking errors over time, velocity of the base and CoT for the pace gait

From table III, as expected, one can see that using solely a joint PD controller leads to bad results. Furthermore, the results show that, for the pace gait, using both controllers or the Cartesian PD alone leads to similar results. However, the Cartesian PD controller achieves a smaller

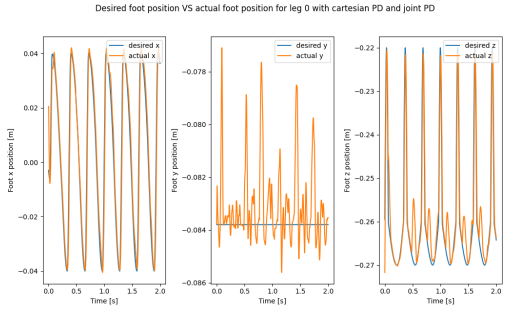


(a) Desired foot positions (x,y,z) vs. actual join angles with Cartesian PD for L0

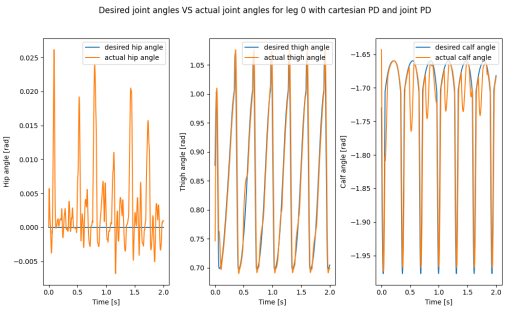


(b) Desired joint angles (q_0, q_1, q_2) vs. actual joint angles with Cartesian PD for L0

Figure 13: Tracking performance with Cartesian PD only for L0



(a) Desired foot positions (x,y,z) vs. actual joint angles with joint and Cartesian PD for L0



(b) Desired joint angles (q_0, q_1, q_2) vs. actual joint angles with joint and Cartesian PD for L0

Figure 14: Tracking performance with joint PD and Cartesian PD for L0

CoT, while providing a bigger velocity of the base and is hence the best controller for the pace gait with the set of parameters proposed.

Since, the best controller seems to be the one using Cartesian PD only, a plot of the CPG states $r, \theta, \dot{r}, \dot{\theta}$ for this controller and for each leg is computed. Figure 15 shows the result. As one can see, for each leg, the CPG amplitude r (blue curve) reaches quickly the desired amplitude $\sqrt{\mu} = 1$. For each leg, the CPG amplitude rate $\dot{r}$ (green curve) highlights this behaviour by exhibiting a quick and big peak at the beginning of the simulation, that is then quickly stabilised with time. The periodic behaviour of the phase and the phase rate (orange and red curves) show the rapid convergence of the CPG states to the limit cycle. Furthermore, these plots highlight the ratio between swing and stance phase ($R = \frac{\omega_{swing}}{\omega_{stance}} = \frac{16\pi}{4\pi} = 4$). Indeed, the swing phase is four time faster than the stance phase. On figure "LF(L1)" for example, the phase rate curve (red curve) has two plateaus. The first one, which is the highest one (values around 50) and has a negative slope, represents the swing phase. The second one, which is the smallest one (values around 15) and has a positive slope, represents the the stance phase and lasts four times longer than the swing plateaus.

Figure 15 shows the synchronicity of the lateral legs pairs. Indeed, the CPG states evolution over time of the L1 leg are very similar to the ones of the L3 leg (see "LF(L1)" and "RH(L3)" plots). The same remark is valid for the L0 and L2 leg (see "RF(L0)" and "LH(L2)" plots).

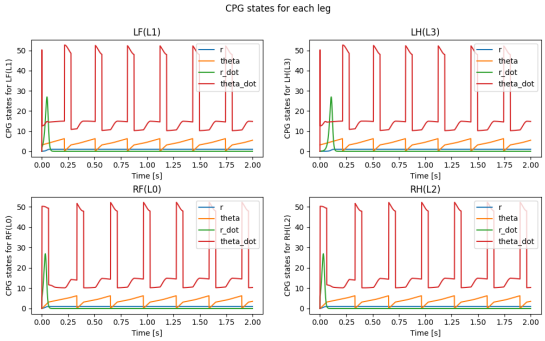


Figure 15: CPG states obtained with a Cartesian PD controller for each leg and for the pace gait

Regarding the metrics, for the pace gait using a Cartesian PD controller, the average time duration in swing phase (across the simulation and the 4 legs) reaches 0.116 [s], while the average time duration in stance phase reaches 0.127 [s], leading to a duty cycle mean of 0.243 [s]. The duty ratio mean reaches 1.095.

Since the target duty ratio for the pace gait is 1, we have an error of 9.5%. Figure 16 highlights the fact that the parameters we found are not the ideal ones. Indeed, according to figure 4d, the feet contact booleans of the legs pairs (L0,L2) and (L1,L3) should be perfectly superim-

pace gait		
	Fast	Slow
ω_{swing}[rad/s]	48	16
ω_{stance}[rad/s]	12	4
T [s]	0.105	0.243
D [-]	0.692	1.102
v_{base}[m/s]	0.8725	0.3471
CoT[-]	3.10	1.66

Table IV: Average duty cycle T, duty ratio D, velocity of the base and CoT for the slowest and fastest pace gait

posed over time, which is not the case on figure 16. Indeed, one can see some unwanted contacts phases leading to a longer stance phase, which is also reflected by the duty ratio value that is bigger than 1. However, the legs contact booleans over time show an overall good overlapping, (by neglecting these unwanted contact phases), meaning that the parameters are right but not exactly optimal.

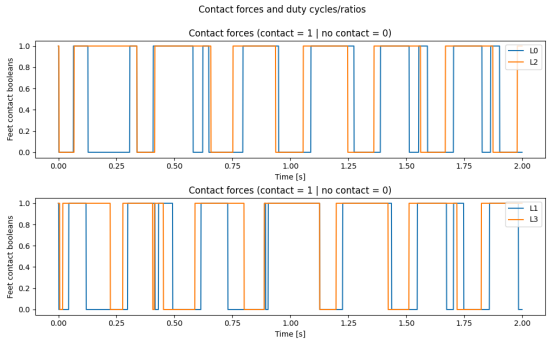


Figure 16: Feet contact booleans for the Pace gait

Finally, table IV shows the results of the fastest and slowest pace gaits achieved with the parameters proposed in this section and using a Cartesian PD controller. In order to increase, respectively decrease, the speed of the base, the ratio between swing and stance phase was kept constant (i.e $R = 4$), but the swing and stance natural frequencies of oscillations were changed.
From table IV, one can see that the higher the pair (ω_{swing}, ω_{stance}), the faster the gait and the smaller the duty cycle T. However, these results come at the cost of increasing the CoT as the joint needs bigger torques. It is interesting to note that using a Cartesian PD controller for the pace gait with high frequencies gives better visual gaits but worst tracking performances than with lower frequencies.

The video "PACE_FAST_0.87.mp4" shows the results of the fastest pace gait achieved. Similarly, the video "PACE_SLOW_0.35.mp4" shows the results of the slowest pace gait achieved.

c. Bound

Regarding the bound gait, the parameters leading to the highest score of the grid search algorithm are the following:

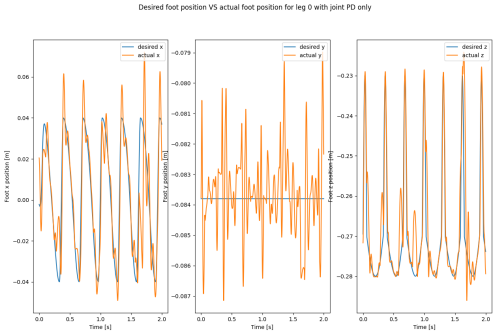
- $\omega_{swing} = 16\pi[rad/s]$
- $\omega_{stance} = 4\pi[rad/s]$
- $g_c = 0.04[m]$
- $g_p = 0.01[m]$
- $h = 0.24[m]$
- $d_{step} = 0.03[m]$
- $\omega = 1$
- $\alpha = 40$
- $\mathbf{K}_{p,Joint} = \begin{bmatrix}300 & 150 & 150\end{bmatrix}^T$
- $\mathbf{K}_{d,Joint} = \begin{bmatrix}2 & 0.5 & 0.5\end{bmatrix}^T$
- $\mathbf{K}_{p,Cartesian} = 5000 \cdot Diag([1,1,1])$
- $\mathbf{K}_{d,Cartesian} = 44 \cdot Diag([1,1,1])$

Some of the parameters were then changed by hand to have a qualitatively better looking gait. The modified parameters are the following:

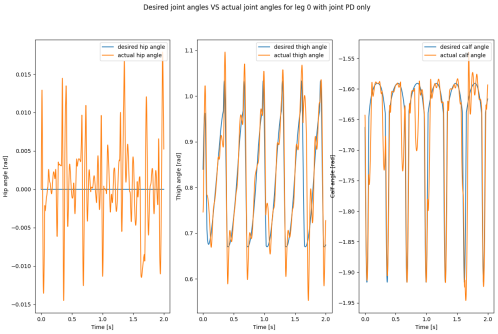
- $h = 0.27[m]$
- $d_{step} = 0.04[m]$
- $\alpha = 50$

For this set of parameters, the effect of using a joint PD controller, a Cartesian PD controller or both controllers were tested on the overall performance of the quadruped. Figure 17 shows the tracking performance of the controller using joint PD only. As one can see on figure 17a, the tracking is not perfect. Indeed, on average, one can observe an offset reaching 0.015 [m] for the X position, 0.002 [m] for the Y position and some fluctuations on the Z axis, specially in stance phase, reaching an average of 0.01 [m]. In addition, as shown on figure 17b, on average, one can observe an offset reaching 0.008 [rad] for the hip angle, 0.07 [rad] for the thigh angle and some fluctuations for the calf angle specially in stance phase, reaching an average of 0.02 [rad].
Figure 18 shows the tracking performance of the controller using Cartesian PD only. As one can see on figure 18a, the tracking is not perfect, but improved compared to the performance achieved by the joint PD controller. Indeed, on average, we can observe an offset reaching 0.007 [m] for the X position, almost no offset for the y position and some fluctuations for the z position specially in stance phase, reaching an average of 0.01 [m]. Note that these fluctuations are more stable and slightly smaller on average compared to the ones obtained with the joint PD controller. In addition, as shown on figure 18b, on average, we can observe an offset reaching 0.0035 [rad] for the hip angle, 0.015 [rad] for the thigh angle and some fluctuations for the calf angle specially in stance phase, reaching an average of 0.05 [rad]. Note that, again, these fluctuations are more stable and slightly smaller on average compared to the ones obtained with the joint PD controller.
Figure 19 shows the tracking performance of the controller using joint and Cartesian PD. As one can see

on figure 19a, the tracking is not perfect, but improved compared to the performance achieved by the joint PD controller. Indeed, on average, we can observe an offset reaching 0.0007 [m] for the y position, almost no offset for the x position and some fluctuations for the z position specially in stance phase, reaching an average of 0.006 [m]. Note that these fluctuations are very similar to the ones obtained with the Cartesian PD controller. In addition, as shown on figure 19b, on average, we can observe an offset reaching 0.0035 [rad] for the hip angle, almost no offset for the thigh angle and some fluctuations for the calf angle specially in stance phase, reaching an average of 0.03 [rad].

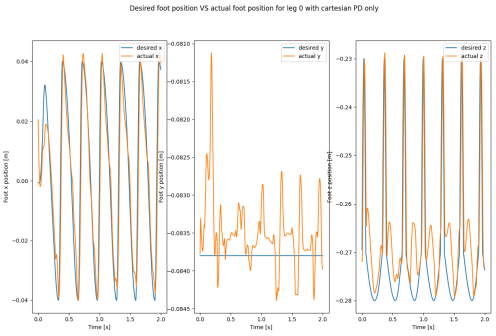


(a) Desired foot positions (x,y,z) vs. actual foot positions with joint PD for L0

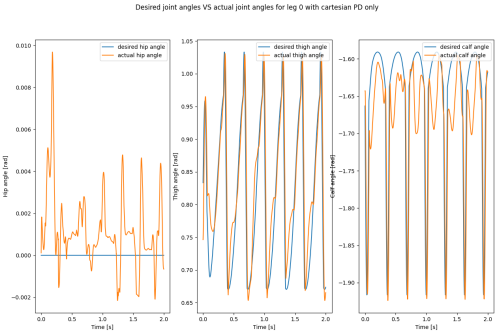


(b) Desired joint angles (q_0, q_1, q_2) vs. actual joint angles with joint PD for L0

Figure 17: Tracking performance with joint PD only for L0



(a) Desired foot positions (x,y,z) vs. actual joint angles with Cartesian PD for L0



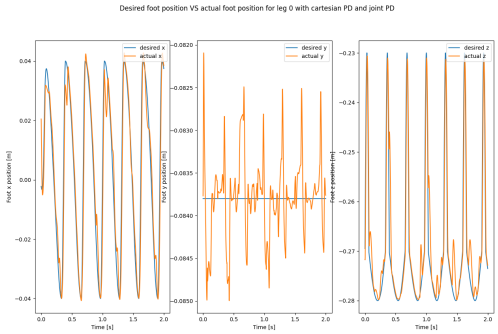
(b) Desired joint angles (q_0, q_1, q_2) vs. actual joint angles with Cartesian PD for L0

Figure 18: Tracking performance with Cartesian PD only for L0

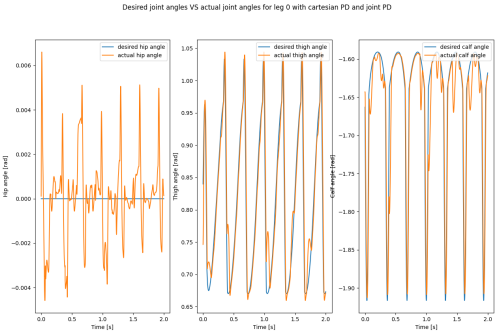
Bound gait			
	Joint PD	Cartesian PD	Both
Δx [m]	0.015	0.007	0
Δy [m]	0.002	0	0.0007
Δz [m]	0.01	0.01	0.006
Δq_0[rad]	0.008	0.0035	0.0035
Δq_1[rad]	0.07	0.015	0
Δq_2[rad]	0.02	0.05	0.03
v_{base}[m/s]	0.276	0.300	0.250
CoT[-]	4.095	2.123	2.946

Table V: Average tracking errors over time, velocity of the base and CoT for the bound gait

For each one of these controllers, the CoT and the base velocity v_{base} have been calculated. Table V summarises the results for the bound gait.
From table V, as expected, one can see that using solely a joint PD controller leads to bad tracking results. Furthermore, the results show that, for the bound gait, using both controllers or the Cartesian PD alone leads to similar results regarding tracking errors. However, the Cartesian PD controller achieves a smaller CoT, while providing a bigger velocity of the base and is hence the best controller for the bound gait with the set of parameters proposed.
Since, the best controller seems to be the one using Cartesian PD only, a plot of the CPG states $r, \theta, \dot{r}, \dot{\theta}$ for this controller and for each leg is computed. Figure 20 shows the result. As one can see, for each leg, the CPG amplitude r (blue curve) reaches quickly the desired amplitude $\sqrt{\mu} = 1$. For each leg, the CPG amplitude rate $\dot{r}$ (green curve) highlights this behaviour by exhibiting a quick and big peak at the beginning of the simulation, that is then quickly stabilised with time. The periodic behaviour of the phase and the phase rate (orange and red curves) show the rapid convergence of the CPG states to the limit cycle. Furthermore, these plots highlight the ratio between swing and stance phase ($R = \frac{\omega_{swing}}{\omega_{stance}} = \frac{16\pi}{4\pi} = 4$).



(a) Desired foot positions (x,y,z) vs. actual joint angles with joint and Cartesian PD for L0



(b) Desired joint angles (q_0, q_1, q_2) vs. actual joint angles with joint and Cartesian PD for L0

Figure 19: Tracking performance with joint PD and Cartesian PD for L0

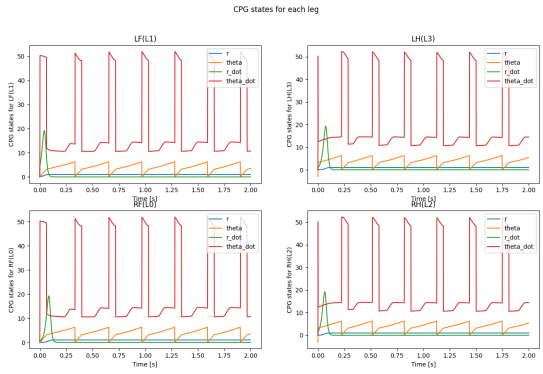


Figure 20: CPG states obtained with a Cartesian PD controller for each leg and for the Bound gait

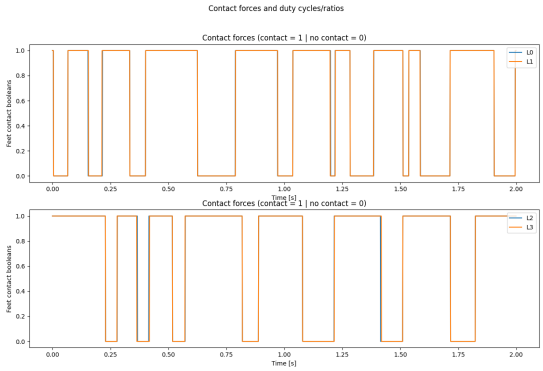


Figure 21: Feet contact booleans for the Bound gait

Indeed, the swing phase is four time as fast as the stance phase. On figure "LF(L1)" for example, the phase rate curve (red curve) has two plateaus. The first one, which is the highest one (values around 50) and has a negative slope, represents the swing phase. The second one, which is the smallest one (values around 13) and has a positive slope, represents the the stance phase and lasts four times more time than the swing plateaus.

Figure 20 shows the synchronicity of the front and back legs pairs. Indeed, the CPG states evolution over time of the L0 leg are very similar to the ones of the L1 leg (see "RF(L0)" and "LF(L1)" plots). The same remark is valid for the L2 and L3 leg (see "RH(L2)" and "LH(L3)" plots).

Regarding the metrics, for the Bound gait using a Cartesian PD controller, the average time duration in swing phase (across the simulation and the 4 legs) reaches 0.136 [s], while the average time duration in stance phase reaches 0.091 [s], leading to a duty cycle mean of 0.227 [s]. The duty ratio mean reaches 0.668. According to figure 4c, the feet contact booleans of the legs pairs (L0,L1) and (L2,L3) should be perfectly superimposed over time, which is the case on figure 21.

Finally, table VI shows the results of the fastest and slowest bound gaits achieved with the parameters proposed in this section and using a Cartesian PD controller. In order to increase (respectively decrease) the speed of the base, the ratio between swing and stance phase was kept constant (i.e $R = 4$), but the swing and stance natural frequencies of oscillations were changed.

Bound gait		
	Fast	Slow
ω_{swing}[rad/s]	48	16
ω_{stance}[rad/s]	12	4
T [s]	0.11	0.23
D [-]	4.46	0.67
v_{base}[m/s]	0.863	0.300
CoT[-]	2.74	2.15

Table VI: Average duty cycle T, duty ratio D, velocity of the base and CoT for the slowest and fastest bound gait

From table VI, one can see that the higher the pair (ω_{swing}, ω_{stance}), the faster the gait and the lower the duty cycle T. However, these results come at the cost of increasing the CoT.

The video "BOUND_FAST_0.86.mp4" shows the results of the fastest bound gait achieved. Similarly, the video "BOUND_SLOW_0.30.mp4" shows the results of the slowest bound gait achieved.

d. Walk

Regarding the Walk gait, the parameters leading to the highest score of the grid search algorithm are the following:

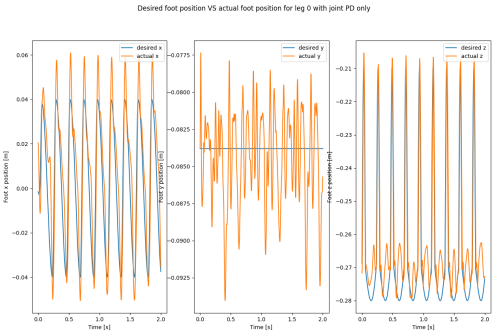
- $\omega_{swing} = 18\pi[rad/s]$
- $\omega_{stance} = 6\pi[rad/s]$
- $d_{step} = 0.04[m]$
- $g_c = 0.06[m]$
- $g_p = 0.01[m]$
- $h = 0.25[m]$
- $\omega = 1$
- $\alpha = 60$
- $\mathbf{K}_{p,Joint} = \begin{bmatrix} 320 & 160 & 160 \end{bmatrix}^T$
- $\mathbf{K}_{d,Joint} = \begin{bmatrix} 2 & 0.5 & 0.5 \end{bmatrix}^T$
- $\mathbf{K}_{p,Cartesian} = 5000 \cdot Diag([1,1,1])$
- $\mathbf{K}_{d,Cartesian} = 40 \cdot Diag([1,1,1])$

Some of the parameters were then changed by hand to have a qualitatively better looking gait. The modified parameters are the following:

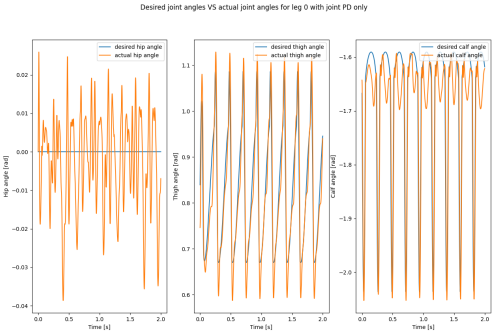
- $\alpha = 70$
- $h = 0.27[m]$
- $\mathbf{K}_{d,Joint} = \begin{bmatrix} 2.5 & 0.625 & 0.625 \end{bmatrix}^T$
- $\mathbf{K}_{d,Cartesian} = 44 \cdot Diag([1,1,1])$

For this set of parameters, the effect of using a joint PD controller, a Cartesian PD controller or both controllers were tested on the overall performance of the quadruped. Figure 22 shows the tracking performance of the controller using joint PD only. As one can see on figure 22a, the tracking is not perfect. Indeed, on average, one can observe an offset reaching 0.016 [m] for the X position, 0.007 [m] for the Y position and some fluctuations on the Z axis, specially in stance phase, reaching an average of 0.015 [m]. In addition, as shown on figure 22b, on average, one can observe an offset reaching 0.01 [rad] for the hip angle, 0.075 [rad] for the thigh angle and some fluctuations for the calf angle specially in stance phase, reaching an average of 0.03 [rad].

Figure 23 shows the tracking performance of the controller using Cartesian PD only. As one can see on figure 23a, the tracking is not perfect, but improved compared to the performance achieved by the joint PD controller. Indeed, on average, we can observe an offset reaching 0.003 [m] for the y position, almost no offset for the x position and some fluctuations for the z position specially in stance phase, reaching an average of 0.01 [m]. Note that these fluctuations are more stable and slightly smaller on average compared to the ones obtained with the joint PD controller. In addition, as shown on figure 23b, on average, we can observe an offset reaching 0.009 [rad] for the hip angle, almost no offset for the thigh angle and some fluctuations for the calf angle specially in stance phase, reaching an average of 0.05 [rad]. Note that, again, these fluctuations are more stable and slightly smaller on average compared to the ones obtained with the joint PD controller.



(a) Desired foot positions (x,y,z) vs. actual foot positions with joint PD for L0



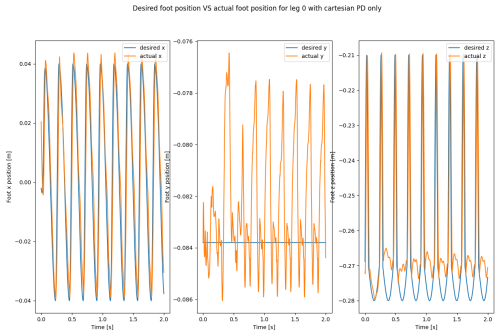
(b) Desired joint angles (q_0, q_1, q_2) vs. actual joint angles with joint PD for L0

Figure 22: Tracking performance with joint PD only for L0

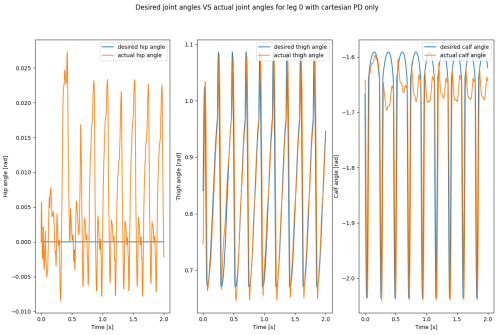
Figure 24 shows the tracking performance of the controller using joint and Cartesian PD. As one can see on figure 24a, the tracking is not perfect, but improved compared to the performance achieved by the joint PD controller. Indeed, on average, we can observe an offset reaching 0.003 [m] for the y position, almost no offset for the x position and some fluctuations for the z position specially in stance phase, reaching an average of 0.008 [m]. Note that these fluctuations are very similar to the ones obtained with the Cartesian PD controller. In addition, as shown on figure 24b, on average, we can observe an offset reaching 0.01 [rad] for the hip angle, almost no offset for the thigh angle and some fluctuations for the calf angle specially in stance phase, reaching an average of 0.04 [rad].

For each one of these controllers, the CoT and the base velocity v_{base} have been calculated. Table VII summarises the results for the walk gait.

From table VII, as expected, one can see that using solely a joint PD controller worsens the performance. Furthermore, the results show that, for the walk gait, using both controllers or the Cartesian PD alone leads to similar results regarding tracking errors. However, the Cartesian PD controller achieves a smaller CoT, while providing a

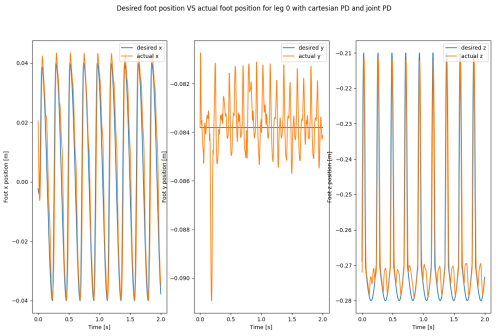


(a) Desired foot positions (x,y,z) vs. actual join angles with Cartesian PD for L0

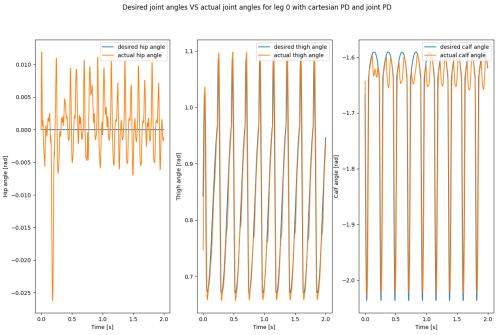


(b) Desired joint angles (q_0, q_1, q_2) vs. actual joint angles with Cartesian PD for L0

Figure 23: Tracking performance with Cartesian PD only for L0



(a) Desired foot positions (x,y,z) vs. actual joint angles with joint and Cartesian PD for L0



(b) Desired joint angles (q_0, q_1, q_2) vs. actual joint angles with joint and Cartesian PD for L0

Figure 24: Tracking performance with joint PD and Cartesian PD for L0

Walk gait			
	Joint PD	Cartesian PD	Both
Δx [m]	0.016	0	0
Δy [m]	0.007	0.003	0.003
Δz [m]	0.015	0.01	0.008
Δq_0[rad]	0.01	0.009	0.01
Δq_1[rad]	0.075	0	0
Δq_2[rad]	0.03	0.05	0.04
v_{base}[m/s]	0.1555	0.4905	0.2877
CoT[-]	11.4158	2.3282	4.5875

Table VII: Average tracking errors over time, velocity of the base and CoT for the walk gait

bigger velocity of the base and is hence the best controller for the walk gait with the set of parameters proposed.

Since, the best controller seems to be the one using Cartesian PD only, a plot of the CPG states $r, \theta, \dot{r}, \dot{\theta}$ for this controller and for each leg is computed. Figure 25 shows the result. As one can see, for each leg, the CPG amplitude r (blue curve) reaches quickly the desired amplitude $\sqrt{\mu} = 1$. For each leg, the CPG amplitude rate $\dot{r}$ (green curve) highlights this behaviour by exhibiting a quick and big peak at the beginning of the simulation, that is then quickly stabilised with time. The periodic behaviour of the phase and the phase rate (orange and red curves) show the rapid convergence of the CPG states to the limit cycle. Furthermore, these plots highlight the ratio between swing and stance phase ($R = \frac{\omega_{swing}}{\omega_{stance}} = \frac{18\pi}{6\pi} = 3$). Indeed, the swing phase is three time as fast as the stance phase. On figure "LF(L1)" for example, the phase rate curve (red curve) has two plateaus. The first one, which is the highest one (values around 50) and has a negative slope, represents the swing phase. The second one, which is the smallest one (values around 25) and has a positive slope, represents the stance phase and lasts three times more time than the swing plateaus.

Figure 25 shows that the legs are not synchronised for the Walk gait.

Regarding the metrics, for the Walk gait using a Cartesian PD controller, the average time duration in swing phase (across the simulation and the 4 legs) reaches 0.137 [s], while the average time duration in stance phase reaches 0.078 [s], leading to a duty cycle mean of 0.215 [s]. The duty ratio mean reaches 0.564.

Finally, table VIII shows the results of the fastest and slowest walk gaits achieved with the parameters proposed in this section and using a Cartesian PD controller. In order to increase (respectively decrease) the speed of the base, the ratio between swing and stance phase was kept constant (i.e $R = 3$), but the swing and stance natural frequencies of oscillations were changed.

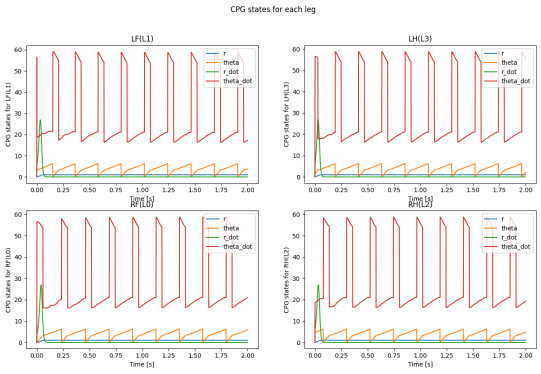


Figure 25: CPG states obtained with a Cartesian PD controller for each leg and for the Walk gait

Walk gait		
	Fast	Slow
ω_{swing}[rad/s]	54	16
ω_{stance}[rad/s]	18	4
T [s]	0.066	0.215
D [-]	1.74	0.564
v_{base}[m/s]	0.79	0.49
CoT[-]	9.15	2.31

Table VIII: Average duty cycle T, duty ratio D, velocity of the base and CoT for the slowest and fastest Walk gait

From table VIII, one can see that the higher the pair (ω_{swing}, ω_{stance}), the faster the gait and the higher the duty cycle T. However, these results come at the cost of increasing the CoT and degrading the quality of the gait.

The video "WALK_FAST_0.79.mp4" shows the results of the fastest Walk gait achieved. Similarly, the video "WALK_SLOW_0.49.mp4" shows the results of the slowest Walk gait achieved.

e. Pronk

In this section, the results of the first bonus gait, called pronk, are discussed. For this gait, the parameters leading to the highest score of the grid search algorithm are the following:

- $\omega_{swing} = 16\pi[rad/s]$
- $h = 0.27[m]$
- $\omega_{stance} = 4\pi[rad/s]$
- $\omega = 1$
- $d_{step} = 0.04[m]$
- $\alpha = 50$
- $g_c = 0.05[m]$
- $g_p = 0.01[m]$
- $\mathbf{K}_{p,Cartesian} = 5000 \cdot Diag([1,1,1])$
- $\mathbf{K}_{d,Cartesian} = 40 \cdot Diag([1,1,1])$
- $\mathbf{K}_{p,Joint} = \begin{bmatrix}300 & 150 & 150\end{bmatrix}^T$
- $\mathbf{K}_{d,Joint} = \begin{bmatrix}2 & 0.5 & 0.5\end{bmatrix}^T$

For this specific gait, the grid search led to bad looking gait. Hence, almost all the parameters were then changed by hand to have a qualitatively better looking gait. The modified parameters are the following:

- $\omega_{swing} = 8\pi[rad/s]$
- $h = 0.2[m]$
- $\omega_{stance} = 17\pi[rad/s]$
- $\omega = 1.1$
- $d_{step} = 0.035[m]$
- $g_c = 0.04[m]$

With this set of parameters and a controller using Cartesian and joint PD, the resulting gait can be seen on video "PRONK.mp4". To obtain a nice looking gait, the parameters tuning phase was very difficult and less intuitive compared to the other gaits (Trot, Pace, Walk, Bound). Indeed, by just reducing the ground clearance value for example, we were able to accomplish a double backflip, as shown on video "PRONK double backflip.mp4". This example shows the meta stability of the set of parameters found.

f. Gallop

In this section, the results of the second bonus gait, called gallop, are discussed. For this gait, the parameters leading to the highest score of the grid search algorithm are the following:

- $\omega_{swing} = 16\pi[rad/s]$
- $h = 0.27[m]$
- $\omega_{stance} = 4\pi[rad/s]$
- $\omega = 1.2$
- $d_{step} = 0.06[m]$
- $\alpha = 50$
- $g_c = 0.05[m]$
- $g_p = 0.01[m]$
- $\mathbf{K}_{p,Cartesian} = 5000 \cdot Diag([1,1,1])$
- $\mathbf{K}_{d,Cartesian} = 40 \cdot Diag([1,1,1])$
- $\mathbf{K}_{p,Joint} = \begin{bmatrix}320 & 160 & 160\end{bmatrix}^T$
- $\mathbf{K}_{d,Joint} = \begin{bmatrix}2.5 & 0.625 & 0.625\end{bmatrix}^T$

For this specific gait, the grid search led to bad looking gait. Hence, almost all the parameters were then changed by hand to have a qualitatively better looking gait. The modified parameters are the following:

- $\omega_{swing} = 24\pi[rad/s]$
- $d_{step} = 0.03[m]$
- $\omega_{stance} = 8\pi[rad/s]$
- $\omega = 1.4[m]$
- $h = 0.24[m]$

With this set of parameters and a controller using Cartesian and joint PD, the resulting gait can be seen on video "GALLOP.mp4". To obtain a nice looking gait, the parameters tuning phase was again very difficult and less intuitive compared to the other gaits (Trot, Pace, Walk, Bound).

2. VMC results

In this chapter, the results regarding the VMC attitude and orientation controllers will be given. For both parts

the analysis is conducted using the best Trot gait parameters found previously (i.e the ones leading to the results in table I for the Cartesian PD controller).

a. VMC - orientation control

In this section, the analysis is driven for a positive target yaw angle of $\psi_{target} = \frac{\pi}{8} \approx 0.4$.

As mentioned previously, the VMC orientation controller is applied only when at least two feet touch the ground at the same time. The first plot on figure 26 shows when the controlled is applied. The controller is applied only when the blue curve reaches the value 1. The second plot on figure 26 shows the time evolution of the actual yaw angle of the base in orange and the target yaw angle of the base in blue. As one can see, the tracking is slow because the VMC orientation controller is not applied continuously. Despite this discrete application, the robot is able to track the reference yaw angle quite nicely after a transient time of approximately 2 seconds.

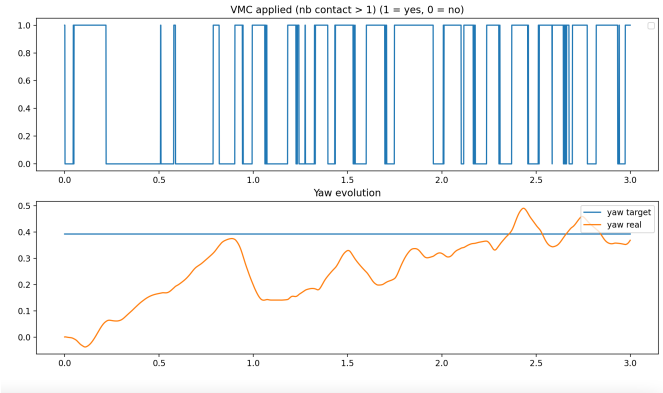


Figure 26: The first plot shows the booleans indicating when the VMC controller is applied, while the second plot shows the actual and target yaw angle over time

Table IX shows the average duty cycle T, the duty ratio D, the velocity of the base v_{base} and the CoT with and without the VMC orientation controller.

Trot gait		
	With VMC	Without VMC
T [s]	0.192	0.186
D [-]	1.672	0.928
v_{base}[m/s]	0.561	0.6561
CoT[-]	1.682	1.7699

Table IX: Average duty cycle T, duty ratio D, velocity of the base and CoT with and without VMC orientation controller for the Trot gait

As one can see, when applying the orientation controller, the duty ratio increases and hence the robots spends more time in stance phase than in swing phase. Furthermore, the velocity of the base is reduced. Finally, the average duty cycle and CoT remain similar with and without applying the orientation controller.

The video "TROT_VMC_orientation.mp4" shows the simulation results.

b. VMC - attitude control

In this part, the results using the VMC attitude controller presented previously are discussed and obtained using a spring stiffness $k_{att,z}$ equal to 80. To show the stabilising property of such a controller, in this section, the simulation is done on a rough terrain composed of a flat terrain will small boxes.

Similarly to what has been previously discussed for the VMC orientation controller, the VMC attitude controller is applied only when at least two feet touch the ground at the same time. Hence, a similar plot as the first graph on figure 26 can be produced to show when the controlled is applied. Furthermore, a similar comparison between the average duty cycle, duty ratio, the base velocity and the CoT can be conducted, as shown on table IX. However, for readability reasons, these results are omitted in this part, because they are very similar to the ones presented above.

Figure 27 shows the comparison between the roll-pitch variations of open-loop gaits in blue and their closed-loop counterparts in orange when the robot walks on the exact same rough terrain. The first plot reveals the roll angle value evolution over time with and without applying the VMC attitude controller. The second plot shows the pitch angle value evolution over time with and without applying the VMC attitude controller. The third plot of figure 27 reveals the pitch angle value as a function of the roll angle value with and without applying the VMC attitude controller. For the three plots, one can see that, when the attitude controller is active, the roll and pitch fluctuations over time a greatly reduced.

Note that, the orange curves in these plots are not periodic, because the simulation was done on an uneven terrain. We know from [1] that, when the controller is active on a flat terrain, these curves reach a periodic behaviour. This characteristic can be seen with the graphs on figure 28, where the roll-pitch variations of the closed-loop gaits (in blue) show periodicity.

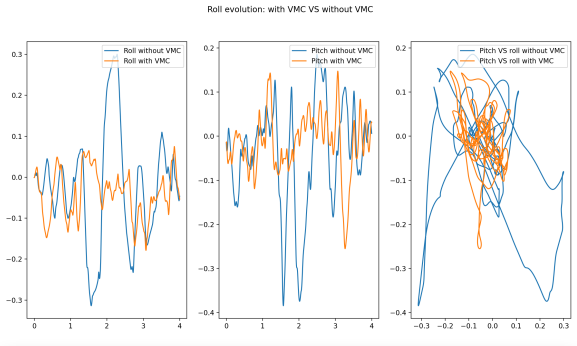


Figure 27: Comparison between the roll-pitch variations of open-loop gaits in blue and their closed-loop counterparts in orange on rough terrain

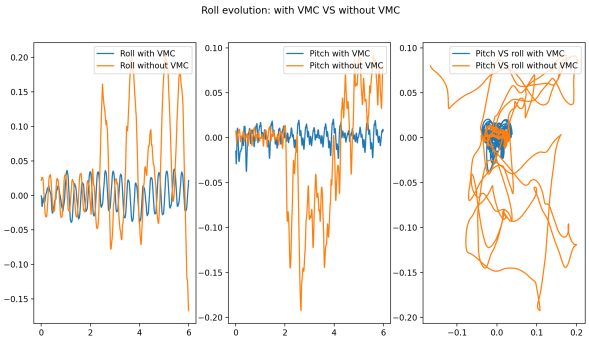


Figure 28: Comparison between the roll-pitch variations of open-loop gaits in orange and their closed-loop counterparts in blue on flat terrain

The video "TROT_VMC_attitude.mp4" shows the simulation results. The left hand-side of this video shows the robot behaviour with the VMC attitude controller active, while the right hand-side of the video shows the robot behaviour without the VMC attitude controller. Thanks to this video, one can appreciate the effectiveness of such a stabilising controller.

F. CPG & VMC Discussion

Throughout the previous sections, the results achieved with CPG & VMC controllers have been presented. In this section, we will first discuss the pros and cons of the Virtual Model Control and the open-loop CPG methods. Then, a short discussion of the results will conclude the CPG part of the report.

There are two mains advantages of using VMC: ease of implementation and robustness. The first one is translated by the absence of a dynamical model. Indeed, to compute the VMC control actions, only the robot kinematics are needed, since the virtual forces to be applied are translated to torques through the Jacobian. Hence, this method allows to avoid the heavy computations required by methods using dynamical models. In addition, the second benefit of such a method compared to using CPG in open-loop is its robustness. Indeed, as shown with the results, when the VMC attitude controller is active and a leg hits an obstacle or go down a step, the virtual forces change accordingly to keep the base balanced. However, the main downside of VMC is that it only works for slow gaits. Indeed, since the controller is active only when two or more feet are on the ground, it requires to have a longer stance phase, so that the probability of having two feet on the ground is higher.
Another drawback is linked to the motor strength. Indeed, VMC methods compute the virtual forces to be applied without any considerations regarding input constraints. Hence, if the motors used are not powerful enough, the torques needed to be applied may not be feasible. Hence, the controller may not always be able to achieve its mission.

Regarding CPG open-loop controllers, multiple conclusions can be derived. As for VMC methods, implementing a CPG controller does not require the full body dynamics of the system, which ease the controller modelling. Another positive aspect of using a CPG controller is its robustness against hardware faults. Indeed, as it is a distributed controller, hardware faults, such as leg malfunctions for example, do not necessarily impact the whole body control, since the other legs are controlled on their own. The oscillators also provide smooth trajectories and ease gait changes. Indeed, the gait can be changed by varying only a few control signals. However, there are some major drawbacks using CPG controllers. The first one is the absence of strong mathematical stability proofs and detailed analysis. Indeed, as this approach is not very used in robotics, at least for now, few mathematical tools exists. Another drawback of open-loop CPG controllers is their parameter tuning. Indeed, in order to find the right parameters leading to a good looking gait, one needs to run multiple simulations.
Regarding our results, one can first conclude that it is hard to characterise a good looking gait with a grid search. Indeed, in order to have a qualitatively better looking gait, the parameters found by the grid search algorithm had to be adapted afterwards for each gait.
Throughout the parameter tuning phase, we realised that the parameters having a strong impact on the gait performance were mainly the step length d_{step}, the robot height h and the swing and stance natural frequencies. Furthermore, the experiments showed that, in order to increase the base speed, one had to increase the swing and stance natural frequencies by keeping their ratio constant. Finally, the results highlight that a fast gait leads generally to a qualitatively bad looking gait and to a high CoT. Hence, the optimal parameters for a gait are not the ones leading to the fastest gait, but the one reaching the target duty ratio.

III. Reinforcement Learning

A. Introduction RL

In the rest of this work, another control method called Reinforcement Learning, RL, is explained and implemented.
Reinforcement learning is about an autonomous agent taking suitable actions to maximise rewards in a particular environment. The basic idea behind this method is to conduct an important number of simulations (in the order of millions for complex environment) to learn a control policy.
RL is a promising control strategy for legged locomotion on abrupt and challenging terrains. Indeed, it can provide a robust controller by training the policy with varying physical parameters and environment perturbations, called domain randomisation. During this project, the ground friction coefficient is changed and an unknown and variable mass is added at different locations on the robot base at training time. Finally, to learn obstacle crossing, several obstacles with different

widths and sizes are randomly placed in the world at training and testing time.

B. Algorithms comparison & hyper-parameters

In order to train the policy and the value function we chose two different policy-learning algorithms close to the state of the art in terms of reliability and sample efficiently :

- PPO : Proximal Policy Optimization
- SAC : Soft Actor-Critic

They both update the value function and the policy at the same time and achieve great results in different environment with only little parameters tuning. However, they are both model-free algorithm and therefore need a lot of data to reach good results. Furthermore, they both need to find a trade-off between exploration and exploitation which should prevent local minimas and lead to optimal results. Their differences are shown in table X.

SAC	PPO
Works in a continuous action space	Works in both discrete and continuous action spaces
Off-policy	On-policy
Adds entropy to the maximisation objective	Uses entropy regularisation

Table X: Comparison between PPO & SAC algorithm

In the following sections both algorithms will be further described and their hyper-parameters presented.

1. Similar hyper-parameters presentation

SAC and PPO algorithms have multiple hyper-parameters and some of them are the same. In this section, this set of similar hyper-parameters is described.

◇ *Policy*
As its name implies, it is the policy model used. In this work, we used a Multi Layer Perceptron (MLP) model. Note that a Convolutional Neural Network (CNN) model could also be used, however for simplicity reasons, this option was discarded.

◇ *Env*
It represents the environment to learn from. Every simulator is different and uses different models influencing the robustness and the performance of the final result.

◇ *Gamma*
Gamma is the discount factor influencing the weight of the instant reward compared to future rewards. This value is always between zero and one, with zero in favour of immediate rewards and 1 for future rewards.

◇ *batch_size*
The number of steps in a batch before a learning step. It influences the confidence of a learning step.

2. PPO

PPO is a policy gradient method that succeed other algorithms like TRPO for example. It is relatively simple to implement and tune, while empirically performing at least as well as its predecessors. Its success lies in its scalability for large scale problems with the formalisation of constraint in its objective function instead of imposing hard constraints. The method uses the stochastic gradient descent method as optimiser. PPO uses the concept of *Trust region* which limits the maximum step size of exploration.

At every iteration, a local optimum point is computed in the trust region. Then, this point is set as the center of the new trust region on which, at the next iteration, a new local optimum will be found and this process goes on during the entire training time. This method tries to limit the performance losses that can happen when using too big steps. In the rest of this section, the hyper-parameters specific to the PPO algorithm are presented.

a. Hyperparameters

◇ *n_steps*
The number of steps to run for each environment per update.

◇ *ent_coef*
The entropy coefficient for the loss calculation. This helps prevent premature convergence of one action probability dominating the policy and preventing exploration.

◇ *learning_rate*
The learning rate influences the amplitude of each learning step. In general, it is a good practice to have a big learning rate at the beginning of the training to accelerate the learning and to reduce it at the end to optimise the final result. For PPO, the learning rate is set using a Stochastic Gradient Descent (SGD) optimiser.

◇ *vf_coef*
This is the value function coefficient used as the learning rate for the value function optimiser.

◇ *max_grad_norm*

The threshold above which the gradient is clipped. This term influences the maximum size of a learning step.

◇ *gea_lambda*
Bias-variance trade off factor of the trajectories. It is a form of reward shaping. GEA stands for

Generalised Advantage Estimator. It is always between zero and one.

◇ *n_epochs*
Number of epochs of interaction to perform while optimising the loss.

◇ *clip_range*
This parameter helps to clip the policy objective. It determines how far a new policy can deviate from the old one and still be profitable. This term is usually small, in the order of 0.1 to 0.3, and always between zero and one.

◇ *clip_range_vf*
This parameter help to clip the parameters of the value function. It is specific to the OpenAI implementation. This term is usually big, around 0.7 to 1, and always between zero and one.

3. SAC

SAC is an off-policy algorithm using the concept of maximum entropy learning and incorporates the double Q-learning trick. It is the successor of the Soft Q-learning and reaches state-of-the-art performance for RL algorithms.

Entropy regularisation is a key concept of this algorithm. The policy is trained to maximise a trade-off between expected return and entropy, a measure of randomness in the policy. It encouraged exploration while giving up unpromising paths and can discover multiple modes of near optimal behaviour.

a. Hyperparameters

◇ *ent_coef*
The entropy coefficient for the loss calculation. It controls the exploration/exploitation trade-off. It is between zero and one where one is in favour of exploration and zero in favour of the exploitation.

◇ *buffer_size*
The size of the replay buffer. This optimises the use of previous experiences by learning with them several times.

◇ *learning_rate*
The learning rate for both policy and value learning. It is used with an Adam optimiser.

◇ *tau*
The soft update coefficient smooths the target value and is used to stabilise training. Fast moving target (i.e large τ) can result in instabilities, whereas slow moving target (i.e small τ) slows down training. Its value is between zero and one, but generally quite small around 0.001.

◇ *train_freq*
The update frequency of the algorithm with gradient descent.

◇ *gradient_steps*
The number of gradient steps to do after each rollout.

◇ *learning_starts*
The number of steps before learning starts. It is useful to gather enough experience for a meaningful update.

4. Algorithms comparison

Both algorithms share similarities and even similar hyper-parameters as explained before, however they differ in 3 key aspects [6]:

First, PPO is an on-policy algorithm. This means that it only uses the current policy observation to learn the value function of its environment. On the other hand, SAC is an off-policy algorithm. This means that it uses previous policies observations to learn its value function. In general, on-policy algorithms privilege stability on data efficiency where off-policy apply the inverse strategy.

Secondly, the trade-off between exploration and exploitation is dealt differently for these two algorithms. PPO uses entropy regularisation in order to avoid local minima, while SAC finds an equilibrium between exploration and exploitation by adding entropy to its maximisation objective.

Finally, they both use the concept of entropy which relates to the confidence a policy has at choosing a certain action given a certain state. A policy with high entropy is very confident on its action and vice versa. But SAC has the advantage to use entropy maximisation which tends to give up unpromising paths during exploration.

In the context of this project the choice between both algorithm was determined based on a training with a simple reward function. The reward function used is further described in section III C 1 but it is essentially a controller that make the robot move as much as possible along the X axis while minimising the energy consumed on a flat terrain. With this setup PPO obtained better results compared to SAC. Indeed, by looking at figure 29, showing the reward at each training iterations, one can easily see a difference. The simulation setup used is simply composed of an energy and fall penalty as well as a distance maximisation over a flat terrain. For this simple setup, one can see that the reward reaches a high plateau in roughly 700'000 iterations using PPO in comparison to SAC that does not reach a plateau and oscillates a lot. This is caused by the fact that SAC explores multiple modes of near optimal behaviours illustrated with the two parallel lines of dots. Contrarily

to SAC, in this specific example, PPO gradually improves its best solution, as expected by its conservative behaviour. This is illustrated by the single line of dots on figure 29. Furthermore, figure 30 shows the mean episode length throughout the training process. As it can be seen, PPO is able to reach the maximum episode length and stay close to it where SAC oscillates much more.

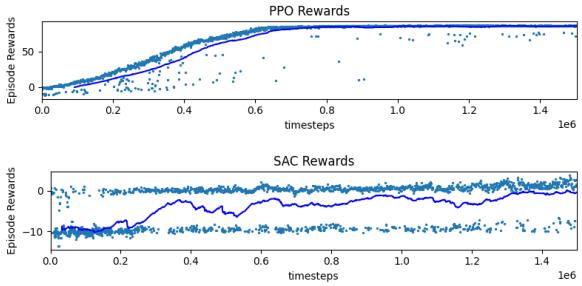


Figure 29: PPO and SAC reward for a standard model training without obstacles

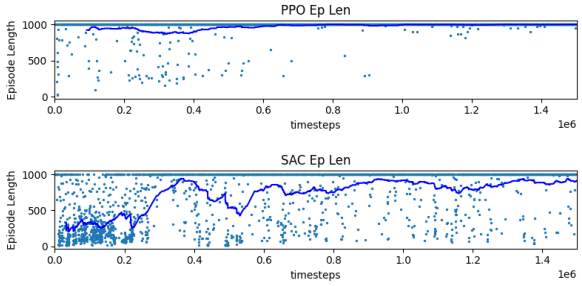


Figure 30: PPO and SAC episode length for a standard model training without obstacles

In the context of this project PPO converged faster than the SAC algorithm and quickly reached the maximum of episode length as it can be seen on figures 29 and 30. Indeed, when the reward plateau is reached, the episode length is constant showing good behaviour in comparison to SAC. Therefore PPO was selected for the rest of the training with the following hyper-parameters.

- ◇ gamma: 0.99
- ◇ max_grad_norm: 0.5
- ◇ n_steps: 4096
- ◇ gae_lambda: 0.95
- ◇ ent_coef: 0.0
- ◇ batch_size: 128
- ◇ learning_rate: 1e-4
- ◇ n_epochs: 10
- ◇ vf_coef: 0.5
- ◇ clip_range: 0.2
- ◇ clip_range_vf: 1

As the impact of these hyper-parameters on the system is complicated to asses and time consuming, the same hyper-parameters were kept for all the training phases even though they may not be optimal.

C. Simulations

The learned policy depends on three main components: the action space, the observation space and the reward function. To create diverse controllers, the observation space and the reward function will change. Indeed, the action space is kept the same as it dictates how the robots can move. This is described in section III C 0 a. To achieve various goals, two controllers will be designed. Hence they will be one observation space and reward function for each controller which are further detailed in section III C 1 and III C 2

a. Action space

For every training, the action space is the same. It is a 12-dimensional vector as there are four legs with three motors each. Hence with a joint controller, the action space will be composed of all joint angles, three by leg, leading to the vector:

$$a_t = [q_{0,0},\ q_{0,1},\ q_{0,2},\ q_{1,0}\cdots,\ \ldots,\ q_{3,2}]$$

where $q_{i,j}$ is the j^{th} joint angle of the i^{th} leg.
However, with a Cartesian controller, the action space will have the same dimension but not with the same components as it will contain the X, Y and Z coordinate of each foot:

$$a_t = [X_1, Y_1, Z_1,\ \ldots,\ X_i, Y_i, Z_i,\ \ldots,\ X_4, Y_4, Z_4]$$

where X_i, Y_i, Z_i are the XYZ-coordinate of the i^{th} leg.

For all simulations and training schemes, the Cartesian controller action space is used. Doing so allow the robot to be trained in task space instead of the joint space. This approach is better, because, in joint space, the policy needs to learn the inverse kinematics. As Marco Hutter said : *we should not learn what is easy to model* and hence, since the inverse kinematics are known, there is no use in learning them. Furthermore, learning them will necessarily take more simulation steps. This is indeed visible on figure 31 showing the reward curve during training with a simple reward function, i.e only forward velocity.

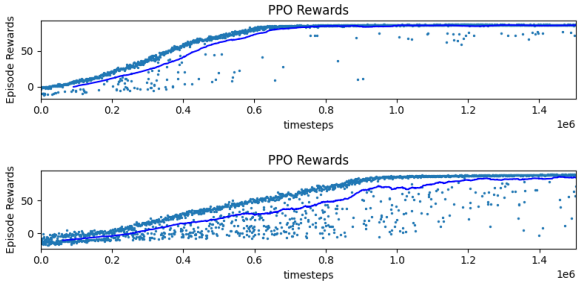


Figure 31: Reward during training using a Cartesian PD controller, on top, or joint PD controller, on bottom

However, it must be said that training a controller in joint space allows the robot to reach higher base velocities, as shown on figure 32. However, as it can also be seen, the robot reaches speeds up to 10 $[\frac{m}{s}]$, which is over the 5

$[\frac{m}{s}]$ limit. This indicates an exploitation of the simulator dynamics. The speed reached using a Cartesian PD is also slightly over 5 $[\frac{m}{s}]$, but it is more stabilised around 6 $[\frac{m}{s}]$, showing realistic behaviour.

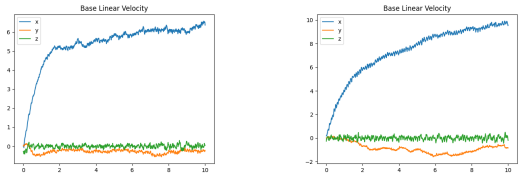


Figure 32: Speed at test time using a Cartesian PD controller, on the left, or joint PD controller, on the right. V_x is in blue, V_y in orange and V_z in green.

In the following section, several scenarios and their respective observation space and reward function are given.

1. Going straight with lowest energy consumption

The first simulation is the easiest one as the observation space is small and the reward function simple. The goal is to have a robot, running along the X axis, while minimising the consumed energy. The environment used for learning is a simple flat environment. However, to make the controller more robust, noise was added during training. Hence a noise with a standard deviation of 0.01 was added.

a. Observation space

For this scenario, the observation space is small to reduce the computational complexity. Having a compact observation space makes the sim-to-real transfer more accurate. Indeed, if the observation space is high dimensional, the learned policy can easily over-fit the simulated environment, which makes it difficult to transfer to the real robots. In this scenario, the chosen observation are the joint angles and velocities, as well as the base orientation. These three measurements can be obtained by simple on-board sensors readings.
The observation space is of dimension [28x1]. The joint measurements have a dimension of [12x1] for the angles as well as their velocities. Finally, the four last dimensions are coming from the base orientation expressed in a quaternion form.

Finally, the observations must be bounded. The joint velocities limits are coming from the real world limitations of the robot. According to [7] the maximum joint velocity is 21 $[\frac{rad}{s}]$. The joint angles are also bounded by the reachable angle. The hip angle will be allowed to move by $\sim \pm$ 11.5 [°]. The calf and thigh angles will be allowed to move by $\sim \pm$ 23 [°]. However, by default the thigh angle has an angle of 45 [°], hence the thigh observation boundary are 45 $\pm$ 23 [°]. Similarly, the calf angle has a default position of 90 [°], hence the calf angle range of motion is 90 $\pm$ 23 [°].

b. Reward function

As we want to robot to move in the X direction, the reward function must maximise the displacement along this axis. However, one must be sure that the policy does not learn from simulator defects. Hence, the maximum velocity must be bounded to 5 $[\frac{m}{s}]$ to prevent unrealistic policy learning. As we want to maximise a distance, the maximum speed is converted into the maximum distance achieved in a time step, denoted d_{max}. Hence the reward term for the displacement is:

$$r_b = min((x_{Base,t+1} - x_{Base,t}), d_{max}) \tag{28}$$

In addition, we want to reduce the energy used to achieve this displacement. This is done by adding a minimisation of the energy in the reward function. Hence, by computing the product of each joint torques, τ_i, & velocities, $\dot{q}_i$, and summing this over the time-steps gives the total energy. As we maximise the reward function, the minus sign is necessary to minimise the energy. Finally, the energy reward function is given by:

$$r_e = -\sum_{i=1}^{K_{Joints}} |\tau_i \cdot \dot{q}_i| \tag{29}$$

Finally, we do not want the robot to fall. Hence, if it happens the robot will have a penalty. To add this in the reward function, we have a base penalty of zero if the robot does not fall and one otherwise. As for the energy, there must be a minus sign as we maximise the reward. The reward function term is then:

$$r_f = \begin{cases} 0 & \text{if the robot does not fall} \\ -1 & \text{otherwise} \end{cases} \tag{30}$$

Finally, the final reward function is a weighted sum of each term mentioned above. The weights are chosen accordingly to the importance of each term and to balance the order of magnitudes between the terms. As the most important term is the fall penalty one, its weight is w_f = 10. Then, the distance reward is the second most important one, as we want to robot to move as much as possible. The weight is set to $w_b = 2$. Finally, the least important term is the energy one as it is quite a secondary objective. Hence the weight is $w_e = 0.008$. Those weights were chosen accordingly to [7]. Hence the complete reward function is:

$$R(s_t, a_t, s_{t+1}) = w_b \cdot r_b + w_e \cdot r_e + w_f \cdot r_f \tag{31}$$

This setup leads to the training curves shown on figure 33. As one can see, the reward quickly reaches a plateau, showing good learning as the episode length is almost constantly to its maximum, around 1000, for every iterations. This figure also highlights the number of iterations during training. This number reaches 1.5 millions, but could be reduced to 900'000 as the plateau is reached at this point.

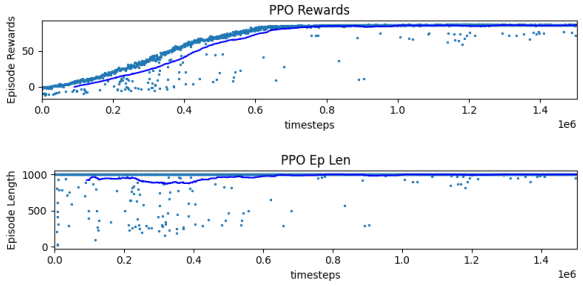


Figure 33: Training reward term comparison between reward with energy.

c. Results

To compare the benefits of adding the energy reward term, r_e, two training were conducted, one with the energy term and one without it. Then, the robot energy efficiency is studied using the cost of transport defined as :

$$CoT = \frac{1}{T_{Sim}} \cdot \frac{1}{m \cdot g \cdot v_{base}} \sum_{k=0}^{T_{sim}} \left(\sum_{i=1}^{N_{Legs}} \sum_{j=1}^{N_{joints}} |\dot{q}_{i,j} \cdot \tau_{i,j}| \right)_k \tag{32}$$

where N_{Legs} is the number of legs, i.e. four, and N_{joints} is the number of joints per leg, i.e. three, T_{sim} is the total number of timesteps, m and v_{base} are the robot mass and linear velocities and $\dot{q}_{i,j}$ and $\tau_{i,j}$ are the j^{th} joint speed and torque of the ith leg at iteration k.

The first control is computed without the energy term as a reference controller. The results of the training are shown on table XI. As one can see the CoT per discrete timestep is 1.383. This value is to be compared with the reward function which includes the energy term, i.e. the second controler, and gives a CoT by discrete timestep of 1.400 which is slightly higher. (This second controller is shown on video basic_energy_controller.mp4. The first one is not shown as it is very similar)
From these results, one can think that the weight of the energy reward term may be too small. Hence, a third training is done with a weight four times higher for the energy term. This time, by looking at table XI, the benefits of this increase are explicit. Indeed the CoT per timestep is 15 % lower. It is however important to note that this new controller leads to slower behaviour. From these experiments, one can conclude, that setting a too low weight for the energy term in the reward function has the same effect as not including this term at all.

	No energy term	Low importance energy term	Higher importance energy term
w_e	0	0.008	0.032
CoT per timestep	1.383	1.400	1.166
Average speed	5.485	5.382	4.610

Table XI: Comparison of energy consumption

Finally, by looking at the training curves between both reward functions, one can compare their efficiency. As the best behaviour regarding energy consumption is reached with the bigger weight, only its training will be looked at. Figure 34 shows the training reward evolution for both controllers. On this figure, one can easily see that, the more complex the reward, the more time the learning phase takes. Indeed, the training with the energy term reaches its plateau around 200'000 training iterations after the training without the energy term. This also shows how quickly the training time is increased when adding terms to the reward function.

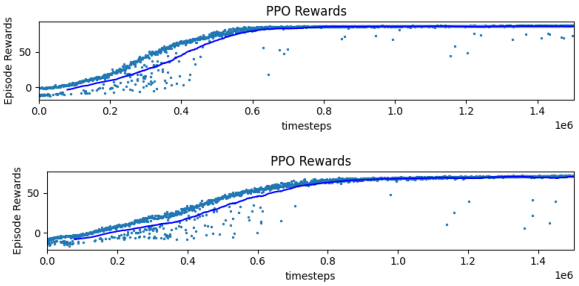


Figure 34: Comparison of training curves without energy term, on top, and with energy term on bottom.

2. Rough Terrain

The second controller is a more complete and complex one. It aims to have a robot crossing obstacles while matching a reference speed.

a. Observation space

To achieve robust locomotion, the observation space should as big as possible. However, the idea is to have a control law that can be transferred to the real world. Hence, the observations are limited to realistic data.

In addition to the observation space presented in the previous section, multiple base observations have been added. First of all, the base position is added in order to add the base height in the reward function. This choice allows to impose a higher base height, which should increase the robot obstacles crossing capability. Then, the base linear velocities are added. Observing them should allow to track a speed reference by training the network to match various speeds. Then, by setting a speed at testing, the robot should be able to match it. Observing the base angular velocities should stabilise the system and ease the obstacles crossing. Finally, some information on the feet are added. Their position, velocities and contact information are observed. These parameters are added to increase the obstacle crossing behaviour by monitoring the foot clearance and their position during training. For this scenario, the total dimension of the observation space is 73.

Even though this project is only about simulation tests, the controller transferability to the real world is an interesting topic to consider. All the joint information

can be easily measured on a real robot using proprioceptive sensors. Hence, their measurement should be quite precise as the noise is limited. The base height can be computed thanks to the joint angles and the leg length hence this observation is dependant on the joint angle measurement. The same reasoning can be applied to the foot positions. Measuring the joint velocities allows computing the foot velocity using the system Jacobian, hence the observation depends on the quality of joint speed measurements. Regarding the foot contact information, multiple solutions can be adopted. The best one is to add pressure sensors at the tip of the leg to give the forces applied to each foot. This measure is very precise as these sensors suffer from little noise. The remaining observations are the base linear and angular velocities as well as the base pose. Those are the most complicated to compute and the noisiest. An accelerometer with a gyroscope allows to estimate the robot pose. Then, the velocities can be found using the position derivative which is even noisier. As we can see, the observation space is quite easily observed in real-life. However, for good sim-to-real transfer, noise should be added to the observations during training using domain randomisation to have more robust behaviour against noise. Hence, a white noise with a standard deviation of 0.01 was added to the measurement. The ground friction coefficient was also changed for the simulation with the obstacles during training to have a more robust controller.

Once again, the observation space must be bounded. As the base position is monitored to impose a base height, the X and Y position will not have bounds to allow the robot to move freely in the world. The linear and angular base velocities are bounded to avoid using simulator defaults. Hence, the linear velocity bounds are $\pm$ 5 $[\frac{m}{s}]$ and the angular velocity bounds are $\pm$ 2 $[\frac{rad}{s}]$ for the pitch and roll but only $\pm$ 1 $[\frac{rad}{s}]$ for yaw. The yaw bounds are smaller than the roll and pitch ones, because the robot can move quicker in roll and pitch directions as it requires less complex movement compared to movement in the yaw direction. Indeed, roll and pitch can be changed by simple leg extensions where yaw requires more complex leg movement.

b. Reward function

The reward function is drastically changed compared to the other simulations. Indeed, the previous reward function was composed of linear terms only, which is not the case for this one. Using exponential terms allows to penalise more bad behaviour than when using linear terms. To simplify the weight determination for each reward term, some parameters have been chosen as in [8]. For stable behaviour the reward terms chosen are:

- Fallen penalty
- Command direction
- Advancement ratio
- Base stabilisation
- Base height

Note that the energy term has been removed to have more power available to correct the robot pose if it stumbles on an obstacle.

As before the fallen penalty is only added if the robot falls. Hence the reward is:

$$r_f = \begin{cases} 0 & \text{if not fallen} \\ -1 & \text{Otherwise} \end{cases} \tag{33}$$

The second reward term is the commanded direction. Its aim is to steer the robot towards a given direction by setting a speed vector to track in the X-Y plane. To do so, an intermediate projection of the speed on the commanded direction is done and expressed as $\mathbf{v}_{pr}$. It is computed as :

$$v_{pr} = \vec{v}_{Real} \cdot \vec{v}_{Cmd}$$

Where $\vec{v}_{Real}$ is the robot speed vector, $\vec{v}_{Cmd}$ is the commanded speed vector and $\cdot$ is the dot product. Note that both velocity vectors are normalised. From this projection the reward term can be computed. The reward is defined as:

$$r_{lv} = \begin{cases} 0 & \text{if } v_{pr} \text{ is zero} \\ e^{-2(v_{pr}-1.0)^2} & \text{Otherwise} \end{cases} \tag{34}$$

Note that the first line is here as $v_{pr} = 0$ is the worst scenario that can happens, i.e. robot going normal to the desired direction, so it must be heavily penalised.

In addition, an *Advancement* term is added to the reward. This has been done to evaluate the real advancement D_{Real} with respect to the maximum displacement D_{Cmd} if the robot was at the commanded speed. This term, r_{adv}, is defined as the ratio between the distance covered during a simulation step at the real speed over the command speed. Two details are worth mentioning here. If the ratio is bigger than one, i.e the robot speed is higher than the command, the reward must be lowered and is equal to $2 - r_{adv}$. Finally, if the commanded speed is zero, there will be a mathematical issue. Hence, the reward term is changed to an exponential term, only if the command is set to [0.0, 0.0], which penalise wrong velocities. This is done to bound its value and penalise more bigger velocities.(see Eq. 35) Finally:

$$r_{adv} = \begin{cases} \frac{D_{Real}}{D_{Cmd}} & \text{if } \vec{v}_{Cmd} \neq \vec{0} \text{ and } \frac{D_{Real}}{D_{Cmd}} \leqslant 1 \\ 2 - \frac{D_{Real}}{D_{Cmd}} & \text{if } \vec{v}_{Cmd} \neq \vec{0} \text{ and } \frac{D_{Real}}{D_{Cmd}} \geqslant 1 \\ e^{-1.5(D_{Real})^2} & \text{Otherwise} \end{cases} \tag{35}$$

This term, for non-zero speed, is not exponential but bounded by one. The lower bound is negative if the robot goes twice as fast as it should, which must be heavily penalised.

The fourth reward function term is the base stabilisation. This term is here to maintain the robot body as flat as

possible. This is done to have a more stable pose if the robot hits an obstacle. This term is composed of two elements: A roll and pitch minimisation and a penalisation of orthogonal velocities. The term which minimise the roll and pitch is quite simple and defined as:

$$r_{pr} = e^{-1.5||\omega||^2} \tag{36}$$

where ω is the roll and pitch vector. This term makes sense, because its value is maximum when the roll and pitch have a zero angle and this happens when the base is flat. The second term is slightly more complex as another speed must be defined: v_0. This speed translate the orientation difference between the command one and the actual robot speed which is defined as :

$$v_0 = ||\vec{v}_{Real} - v_{pr} \cdot \vec{v}_{Cmd}||$$

Putting it all together gives a reward term which has the following structure:

$$r_{bs} = e^{-1.5v_0^2} + r_{pr} \tag{37}$$

where r_{pr} is the roll-pitch reward defined in eq 36.
Finally the last term in the reward puts the robot as high as possible to have as much clearance under it to cross obstacles more easily. The base must be as high as possible hence the reward term is defined as:

$$r_{hb} = e^{1.5 \cdot z_{base}} \tag{38}$$

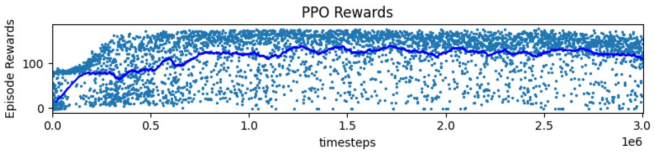
Where z_{base} is the current base height. Note that this term is not bounded by 1. However, as the maximum base height is 0.4, because it is limited by the leg length, it will be bound by default to $e^{0.6} \approx 2$ which is easily scalable with the other ones.
Putting it all together gives the final reward function as:

$$R(s_t, a_t, s_{t+1}) = w_{lv} \cdot r_{lv} + w_{adv} \cdot r_{adv} + w_{bs} \cdot r_{bs} + w_{hb} \cdot r_{hb} + r_f \tag{39}$$

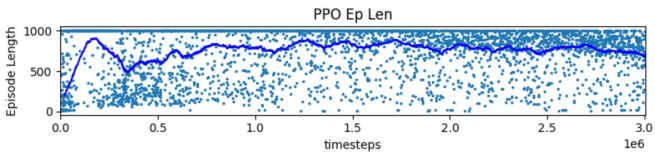
Each weight is chosen depending on the control task. As this reward function is much more complex, the weights are more difficult to estimate. Their choice in our case is explained in the following section.

c. Results

In order to reduce the training time, two controllers were trained to achieve diverse goals, i.e. one for the obstacle crossing and one for speed matching. By tuning a little more the weights and much longer training phase, they could be fused in a single one. Indeed, the obstacle crossing controller was trained using the same reward function but the reference speed was constantly set to [1.0, 0.0] which trains the robot to go at this desired speed. The training weights for this task were set depending on the importance of the related term. Hence the most important term is the fallen one as we want to avoid at all cost the robot falling, i.e. $w_f = 2$. Then the speed matching behaviour is the second most important as we want a moderate speed which should reduce the number of falls if the robot hits an obstacle as a limited speed limits the robot inertia, i.e. $w_{lv} = 0.08$. Then the third most important term is the stabilising one. Indeed, if the robot hits an obstacle we want it to be the most reactive possible in order to stabilise itself, i.e. $w_b = 0.04$. Finally, the advancement ratio and base heigh are close as they are less important for a robust behaviour,i.e. $w_{hb} = 0.02$ and $w_{adv} = 0.03$. By doing so, the robot is



(a) Reward during training for obstacle crossing



(b) Episode length during training for obstacle crossing

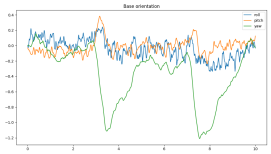
Figure 35: Training reward and episode length

able to properly walk through unseen obstacle. The video *obstacle_crossing.mp4* shows the results of the controller which has been trained three millions times with random weights on its back and random boxes on its path. By looking at figure 35 one can see that the reward throughout the whole simulation is spread between 0 and 150. This is caused by the random aspect present during the simulation. However, the mean episode length as well as its reward are increasing showing good learning from the controller.

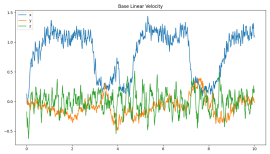
Figure 36: Robot stumble on an obstacle at testing time

There are multiple important points to note. First of all, thanks to the robust element of the reward, the robot is able to recover from severe impacts. As it can be seen on figure 36 the robot hits badly an obstacle with its front leg leading to a bad pose, however it is able to stabilise and continue its walking. This is even more visible on some measurement done during the simulation. Indeed, by looking at figure 37a it is clear that two impacts happened around 3.5 [s] and 7.5[s] as the yaw angle varies

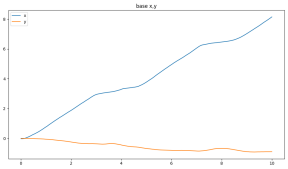
rapidly (green curve on figure 37a). These impacts are also visible on figures 37b and 37c where a plateau appears at the same instant. Indeed, one can see that the speed along the X axis is close to zero on figure 37b and there is no advancement on the X direction on figure 37c. It is also interesting to note how well the reference speed is tracked. Indeed, by looking at the base velocities, fig. 37b, one can see that the forward instantaneous velocity (blue curve in figure 37b) is oscillating around one and the side one (orange curve in figure 37b) around zero which corresponds to the given reference $\vec{v}_{ref} = [1.0, 0.0]$. Note that no reference speed is set along Z axis as it is unnecessary for omnidirectionnal motion. Finally, one can assess the good performances of the speed matching by looking at the robot orientation after the obstacle hit. As it can be seen with the orientation as well as the robot speed, the robot quickly turn to match the reference speed.



(a) Base orientation. Roll angle is blue, pitch is orange and yaw green.



(b) Base velocities. V_x is in blue, V_y in orange and V_z in green.



(c) Base position along X, in blue, and Y axis, in orange

Figure 37: Robot measurement during obstacle crossing simulation

Finally, one can see that these results are reflected on the reward function shown in figure 38. Indeed, the reward value is high when the robot moves nicely and drops when the robot stumble on an obstacle.

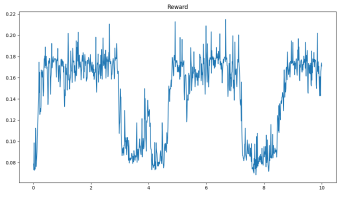
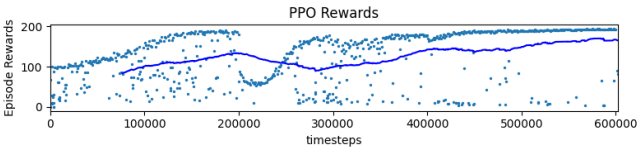


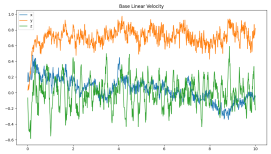
Figure 38: Reward at testing time

The video called "longer_robust.mp4" shows the performance of the same controller but with a heavier mass placed on top of the robot base.
Regarding the speed-matching controller, the training seems not able to learn to match a varying speed. The idea to learn how to match speed was to change the reference speed multiple times during training. Doing so should allow to match various speeds at testing time. However, with the chosen network architecture it does not seem possible to learn varying speeds. Indeed, the network is able to match the last speed used at training time, but not the previous ones. This is maybe due to the fact that the neural network used (MLP) is too simple and has no memorising capabilities. In further work, one should maybe change the network architecture by adding memory terms such as LSTM components to improve the controller performance.

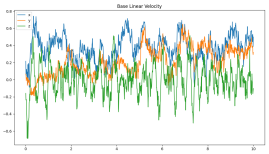
This remark is even clearer when looking at figure 39 which shows the testing behaviour of the robot when trained with the following three speeds: [1.0, 0.0], [0.0, 0.5] and [0.0, 0.75] with 200'000 iterations each. As it can be seen on figure 39a, a plateau is reached before changing speed which means that the controller learnt how to track the desired speed. At testing, if the reference velocity is set to [0.0, 0.75], the robot is able to properly achieve this motion as it can be seen on figure 39b, where the mean robot speed is roughly [0.0, 0.75]. The performance of this controller can be seen with the video called "speed_match_y.mp4". It is interesting to note that the controller learns to turn to robot to increase its stability as forward robot motion is more stable than side motion. However, if the robot as a reference speed of [1.0, 0.0] at testing, the robot moves along the X axis but also along the Y axis as it is the last learned behaviour. This is visible by looking at figure 39c where the mean speed is about [0.4, 0.4].



(a) Reward during training



(b) Base velocities when testing with a speed input along Y axis



(c) Base velocities when testing with a speed input along X axis

Figure 39: Advancement measurement for speed-matching. V_x is in blue, V_y in orange and V_z in green.

To conclude, over 100 different simulations with different weights and reward functions were tested without any better results regarding speed matching. This includes changing the reference speeds during training according

to various patterns. First, an uniformly increasing speed reference along each axis has been tested, which led to the same issues as the ones described before. Then, randomly mixing these speed samples has been tested, once again showing poor results. Taking random reference speed in an interval has also been tested but seems to lead to even worst results. Finally, the number of iterations for each reference speed has also been varied, as well as the training time without more success. However, one key point to keep in mind is that, as the time allowed for this project was not very long, every training as to be kept relatively small, i.e. not over three millions iterations, in order to be able to analyse the result. Hence, longer training time as well as the addition of the aforementioned LSTM components could be a potential solution to tackle this issue.

IV. Conclusion

During the first part of this report, the *Central Pattern Generator* concept has been explored to generate an open-loop controller for legged locomotion. Even though the ϕ matrices were easily determined for each gait, finding the hyper-parameters leading to qualitatively good looking gaits was challenging. Since there is little literature on this subject, a grid search algorithm has been proposed in this work, but was less effective than expected. This method gave a good starting point to then tune the hyper-parameters by hand to find the optimal ones in terms of base velocity and gait quality. Furthermore, we realised how hard it was to quantify what defines a good gait. Despite these challenges, a set of near optimal hyper-parameters has been found for 6 different gaits on a flat ground.

Then, the outputs of the CPG open-loop controller have been augmented using VMC to allow attitude stabilisation and orientation tracking. The VMC orientation controller allowed the robot to follow a preferred direction given by higher centers effectively. The VMC attitude control approach led to very promising results as it improved the robot robustness. Indeed, when the VMC attitude controller was active, the robot was able to move on rough terrain with increased stability, but at the cost of decreasing its base speed.

Finally, the last part of this report focused on *Reinforcement Learning*. According to literature, *RL* is used for most of the more impressive legged robot behaviours. Throughout this work, designing reward functions to achieve complex motion has been shown to be quite challenging. Moreover, it showed that adding terms to match the speed and go over obstacles increased a lot the training time. In addition, we realised that weight design was a whole new problem on its own.

Finally, as the time allowed for this project was limited, a complete controller for a complex environment has not been entirely developed, but its main building blocks have been successfully implemented.

[1] M. Ajallooeian, S. Pouya, A. Sproewitz, and A. J. Ijspeert, "Central pattern generators augmented with virtual model control for quadruped rough terrain locomotion," pp. 3321–3328, 2013.

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

[3] L. Righetti and A. Ijspeert, "Design methodologies for central pattern generators: an application to crawling humanoids," 01 2006.

[4] J. Pratt, C.-M. Chew, A. Torres, P. Dilworth, and G. Pratt, "Virtual model control: An intuitive approach for bipedal locomotion." *I. J. Robotic Res.*, vol. 20, pp. 129–143, 01 2001.

[5] S. M. Dai Owaki, Masashi Goda and A. Ishiguro, "A minimal model describing hexapedal interlimb coordination: The tegotae-based approach," 06 2017.

[6] AmazonWebServices, "Aws deepracer training algorithms proximal policy optimization (ppo) versus soft actor critic (sac)," *AWS DeepRacer - Developer Guide*, pp. 11 – 13. [Online]. Available: https://docs.aws.amazon.com/deepracer/latest/developerguide/awsracerdg.pdf#deepracer-how-it-works-reinforcement-learning-algorithm

[7] G. Bellegarda and Q. Nguyen, "Robust high-speed running for quadruped robots via deep reinforcement learning," *CoRR*, vol. abs/2103.06484, 2021. [Online]. Available: https://arxiv.org/abs/2103.06484

[8] J. Lee, J. Hwangbo, L. Wellhausen, V. Koltun, and M. Hutter, "Learning quadrupedal locomotion over challenging terrain," *CoRR*, vol. abs/2010.11251, 2020. [Online]. Available: https://arxiv.org/abs/2010.11251

[9] J. Tan, T. Zhang, E. Coumans, A. Iscen, Y. Bai, D. Hafner, S. Bohez, and V. Vanhoucke, "Sim-to-real: Learning agile locomotion for quadruped robots," *CoRR*, vol. abs/1804.10332, 2018. [Online]. Available: http://arxiv.org/abs/1804.10332

[10] T. Haarnoja, A. Zhou, S. Ha, J. Tan, G. Tucker, and S. Levine, "Learning to walk via deep reinforcement learning," *CoRR*, vol. abs/1812.11103, 2018. [Online]. Available: http://arxiv.org/abs/1812.11103

[11] J. Hwangbo, J. Lee, A. Dosovitskiy, D. Bellicoso, V. Tsounis, V. Koltun, and M. Hutter, "Learning agile and dynamic motor skills for legged robots," *CoRR*, vol. abs/1901.08652, 2019. [Online]. Available: http://arxiv.org/abs/1901.08652

[12] A. Iscen, K. Caluwaerts, J. Tan, T. Zhang, E. Coumans, V. Sindhwani, and V. Vanhoucke, "Policies modulating trajectory generators," *CoRR*, vol. abs/1910.02812, 2019. [Online]. Available: http://arxiv.org/abs/1910.02812

[13] X. B. Peng, E. Coumans, T. Zhang, T. E. Lee, J. Tan, and S. Levine, "Learning agile robotic locomotion skills by imitating animals," *CoRR*, vol. abs/2004.00784, 2020. [Online]. Available: https://arxiv.org/abs/2004.00784

[14] R. Boney, J. Sainio, M. Kaivola, A. Solin, and J. Kannala, "Realant: An open-source low-cost quadruped for research in real-world reinforcement learning," *CoRR*, vol. abs/2011.03085, 2020. [Online]. Available: https://arxiv.org/abs/2011.03085

[15] V. Tsounis, M. Alge, J. Lee, F. Farshidian, and

M. Hutter, "Deepgait: Planning and control of quadrupedal gaits using deep reinforcement learning," *CoRR*, vol. abs/1909.08399, 2019. [Online]. Available: http://arxiv.org/abs/1909.08399

[16] G. Bellegarda and K. Byl, "Training in task space to speed up and guide reinforcement learning," *CoRR*, vol. abs/1903.02219, 2019. [Online]. Available: http://arxiv.org/abs/1903.02219

[17] A. J. Ijspeert, "Central pattern generators for locomotion control in animals and robots: A review," *Neural Networks*, vol. 21, no. 4, pp. 642–653, 2008, robotics and Neuroscience. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0893608008000804

[18] C. García-Saura, "Central pattern generators for the control of robotic systems," *CoRR*, vol. abs/1509.02417, 2015. [Online]. Available: http://arxiv.org/abs/1509.02417

[19] C. Liu, Y. Chen, J. Zhang, and Q. Chen, "Cpg driven locomotion control of quadruped robot," 10 2009, pp. 2368–2373.

[20] A. Winkler, I. Havoutis, S. Bazeille, J. Ortiz, M. Focchi, R. Dillmann, D. Caldwell, and C. Semini, "Path planning with force-based foothold adaptation and virtual model control for torque controlled quadruped robots," 05 2014, pp. 6476–6482.