Legged robots - MICRO 507

EPFL 2020

Mini project : three-link 2D biped walker modeling and controlling with PD controller and reinforcement learning

---

Authors:
Joachim HONEGGER
Nicolas MINDER
Laura PUYET

Professors :
Prof. Auke Jan IJSPEERT

December 13, 2020

Contents

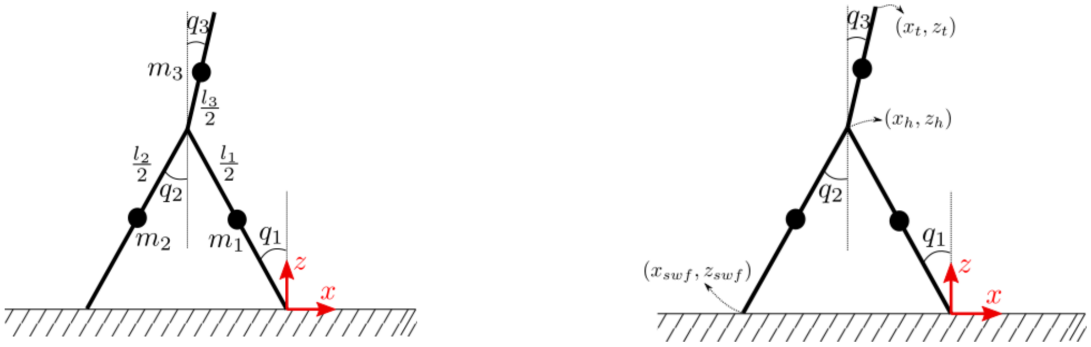
1 Introduction

As part of this course we implemented a model of a three-link 2D biped walker on Matlab. It will be controlled with two different methods. The first one was done using virtual constraints. The second one was done with deep reinforcement learning (RL).
In this report, we will first establish the kinematic and dynamic of our model. We will then demonstrate the implementation of a PD controller and a RL controller. We will optimize the RL controller to find the optimized parameters and finally we will compare the two controllers and discuss possible improvements to our project.

2 Theoretical background

2.1 Modeling

As for any complex model simulation, the establishment of a simplified model is required. It allows to understand how to adequately apply relevant models to simulate more complicated models afterwards. During the assignment 2, we modeled and visualized a three-link 2D biped illustrated in Figure 1 and implemented Kinematics and Dynamics models as well as an impact map by solving the equations of motion. These models are well-suited to describe double pendulum's problem and similar as the three-link 2D biped because these models use energy and constraints to obtain an equation of motion. In addition, they are easy to model and have therefore a small computation cost.



(a) Mass model (b) Foot placement model

Figure 1: General model of the tree-link 2D biped

2.1.1 Kinematics

To simulate the kinetics of the three-link 2D biped, we first identified the positions and velocities of the three masses placed on the biped (see on Figure 1a) as well as for the three points of interest the swing foot, hip joint and torso 1b. These parts are used to improve the control of the biped model that will be implemented later.

2.1.2 Dynamic

To represent the dynamic model, we used the inverse dynamics system relating to the actuator torque joints and the resulting motion as follows:

$$\tau = B(q)\ddot{q} + C(q,\dot{q}) + g(q) + F_v\dot{q} + F_s sign(\dot{q}) \tag{1}$$

For the moment, frictional forces are neglected (ie F_s =0 and $F_v = 0$). Then, we apply the following Lagrangian Mechanics method.

$$L(q, \dot{q}) = T(q, \dot{q}) - V(q) \tag{2}$$

where $T(q, \dot{q})$ is the kinetic energy and $V(q)$ the potential energy.
We derive the Lagrangian by the time:

$$\frac{d}{dt}(\frac{\partial L}{\partial \dot{q}_i}) - \frac{\partial L}{\partial q_i} = 0 \tag{3}$$

we can obtain the following formula:

$$M(q)\ddot{q} + C(q, \dot{q})\dot{q} + G(q) = Bu \tag{4}$$

where $M(q)$ represents the mass matrix, $C(q, \dot{q})$ the corriolis matrix, $G(q)$ the gravity matrix and u the controller's actions illustrated on Figure 2.

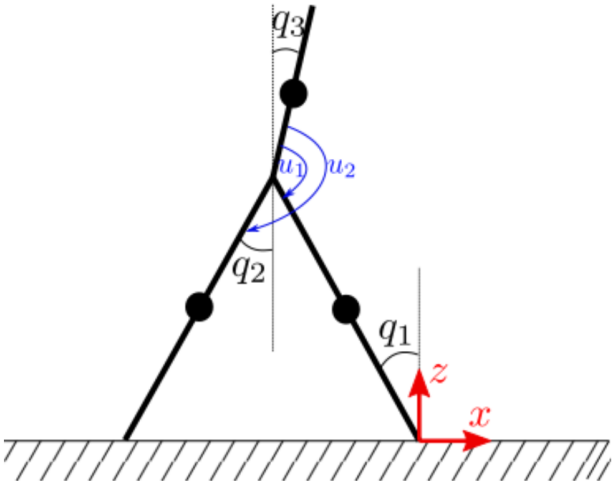


Figure 2: General model of the tree-link 2D biped illustration with the actions of the controller u_1 and u_2

In order to compute the torques τ which operate on the system, we need to compute the matrix B. To achieve this we need to compute the virtual work of the system:

$$\delta W = \delta W_1 + \delta W_2 = \frac{d\theta_1}{dt}u_1 + \frac{d\theta_2}{dt}u_2 = u^T\dot{\theta} \tag{5}$$

which can also be defined as:

$$\delta W = \tau^T\dot{q} \tag{6}$$

on the other hand $\dot{\theta}$ can be expressed as:

$$\dot{\theta} = B^T\dot{q} \tag{7}$$

here we can compute the B matrix and then it leads to:

$$u^T\dot{\theta} = \tau^T\dot{q} = u^T B^T\dot{q} \tag{8}$$

That gives the following relation:

$$\tau = Bu \tag{9}$$

2.1.3 Impact map

Now that our biped is well defined with the basic previous models, we can focus on the biped displacement simulation.
We use an impact map to describe the transition from one displacement step to another. Particular attention must be paid on the conventions established during a step change, it is illustrated on Figure 3.

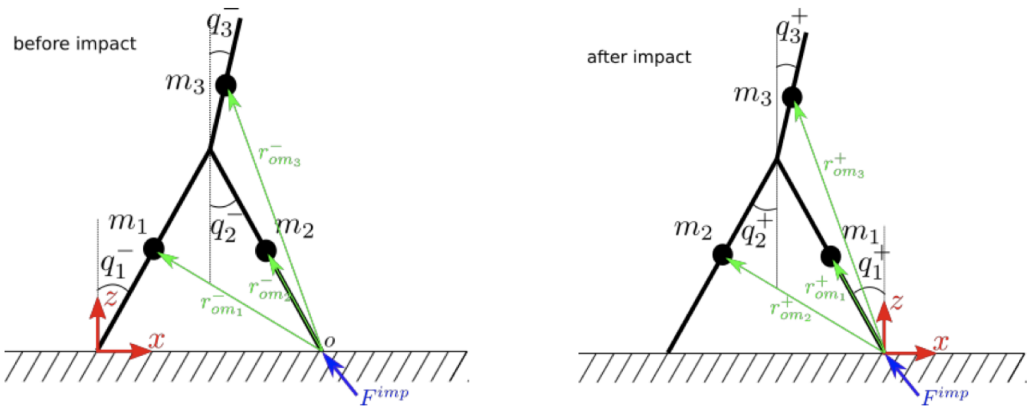


Figure 3: Tree-link 2D biped model defined with the convention before and after an impact

The impact map modifies the previous swing foot as the new stance leg. We thus model everything according to the stance leg by switching the conventions and the origin of the system. Our impact map is considered as a relation between angles and angular velocities between before and after an impact. The relation between before and after the impact can be defined as followed:

$$(q_1^+, q_2^+, q_3^+) = (q_2^-, q_1^-, q_3^-) \tag{10}$$

2.1.4 Numerical integration and simulation

Now that our model is well established, we are interested in implementing a display to watch its behaviour. We first need to solve the motion equation (see equation 4). We transform the second order equation into a first order one. The ODE dimension becomes 6 due to the fact that we have 3 DOFs on which we want to control both angle and velocity. Once the differential equation has been changed, it can be solved by using numerical integration.
To simulate the impact of the foot on the ground, we implement an event function. This function will detect when the z-position of the foot reach 0 when decreasing. We add to the z-position a small factor of 0.001 as the way to compute on Matlab is linked to the model of computer used. When the event function detects the foot touching the ground it stops the integration, to let the program updates the stance and swing foot.
After having interrupted the integrator, the initial condition are updated and a new gait cycle starts.
The model can then be illustrated with an animation to observe the behaviour of the biped model. The real time factor of the simulation is estimated with the time of animation (computed with a timer) and the simulation duration (from sln structure). If this real time factor is one it means the animation runs at the same speed than the real time of the biped model. If the real time factor is less than one, it means that the animation is slower than the biped model time. Therefore, the model takes more time to simulate than the events happen in real life.
The variable *skip* defines the number of step of the numerical integration shown in the animation. This means that if *skip* is set to 1 each frame of animation correspond to one increment of time. This will make the animation run slower and reduce the real time factor. To avoid long run time to animate the model, *skip* can be set to values greater than one, for example 5. This will mean that now each frame of

the animation will skip 5 increments of time of the numerical integration.
The position of the stance foot *r0* needs to be updated at each steps. This parameter allows us to plot the three-link biped in the global frame over time.

There are different techniques to achieve numerical integration. These methods performances can be evaluated with multiple criterion. The stability can be rated by looking at the state-space plot. It consists of a plot representing the position on the horizontal axis and the velocity on the vertical one. If the method is stable, the plot should make a perfect circle. Furthermore, the accuracy can be studied by analysing the error of the method and also by looking at the state-space plot.
The first model that can be used is the explicit Euler. The numerical solution of this method diverges. One explanation is that we only use the lower order term in the Taylor expansion for solving the ODE. To better approximate the actual solution we could take into account more terms from the Taylor expansion. However, the solution will always gain energy and still diverge from the solution.
One other possible algorithm is the semi-implicit Euler method. As this method is implicit, we need to compute $\dot{y}(t+h)$ to know the solution. It is defined as $y(t+h) = y(t) + h\dot{y}(t+h)$. The region of absolute stability of this method includes the whole left half of the complex plane, the only unstable area is a circle of radius 1 centered in one. Meaning that this solution has a good stability, it is even L-stable making it a good candidate to solve stiff equations. However, the fact that the method is implicit makes it hard to implement. In the state space this method does not give a perfect circle, because the computed solution is always leading the real solution. This leads to a low accuracy solution. The error of this method is of first order meaning that the error at each time step is $O(h^2)$ and the total accumulated error is $O(h)$.
The midpoint method, also called Runge-Kutta can be used. The error in terms of h^4 can be get by calculating the trajectory with different steps h (dt) and determines the differences in the results. We can compute the error per step, it is approximately equal to $O(h^5)$. However, the total accumulated error is of the order of h^4. As one-step and explicit methods, the stability can be discussed. It will not diverge that much because we have 4 orders for a second order equation. It will eventually diverge when it goes to infinity (or negative infinity).
One last model that can be use is the adaptive time stepping. This methods have a good stability as it adapts the time step according to the solution variations. We can have stability issue if the chosen step size h is too large. In this case, it is a step size control method, bigger the h is, bigger the corresponding gradient will be.
Some algorithms are supported by odeint. Odeint uses Lsoda from FORTAN to solve systems. It automatically selects between non-stiff (Adams) and stiff (backward differentiation formula - BDF) methods. It uses the non-stiff method initially, and dynamically monitors data in order to decide which method to use.
The odeint method will also automatically adapt the time step. It uses the comparison between two simple integrator. It uses these in order to have an estimated error below a user defined threshold.

2.1.5 Control and optimization

PD controller As a first controller and to validate our gaits metrics, we use virtual constrains to define our controller. This control approach for bounding gait of our biped robot is a relative motion relation between two related joints u_1 and u_2 imposed to the robots in terms of a specified gait, which can drive the robot to run with desired gait. To summary, the actual model has two actuators to control three degrees of freedom. It uses two constraints y_1 and y_2 that are a function of the model's coordinates (hip, stance and swing foot angle trajectories and the desired lean angle). A proportional derivative (PD) control is introduced to stabilize the robot running speed and enhance the stability. This PD controller has the following expression:

$$u_1 = {k_1}^P * y_1 + {k_1}^D * \dot{y}_1 \tag{11}$$

$$u_2 = {k_2}^P * y_2 + {k_2}^D * \dot{y}_2 \tag{12}$$

optimization criteria After evaluate the controller, we are looking for optimize its hyper-parameters $\mathrm{p} = [q_0, \dot{q}_0, \alpha, k_1^P, k_2^P, k_2^D]$. Therefore, we define a loss function to simulate the model using the input parameters, evaluate the quality of the gait and return a value to indicate the performance as follow:

$$f(p) = w_1 * |\dot{x}_{hip} - \dot{\bar{x}}_{hip}(p)| + w_2 * CoT(p) \tag{13}$$

where w_1 and w_2 are the weights to simulate a lower velocity by minimizing the effort through CoT computed as $CoT = \frac{effort}{x_{hip}(T) - \overline{x_{hip}}(1)}$.

2.2 Controller optimisation with reinforcement learning

2.2.1 Implementation of system

The goal of this part is to use reinforcement learning to make the biped model walk. To do so we use the same agent that the one presented in the Matlab tutorials [1][2] in which they used a Deep Deterministic Policy Gradient (DDPG) algorithm. As such we use the *Matlab Deep Reinforcement Learning Toolbox* to implement our new controller. The biped model evolves in a simulation environment. This environment sends the current states q, dq to the agent which will in return output the actions, in our case the torques u_1, u_2. Then, The environment computes the next states from the previous actions and a reward used to optimize the agent.

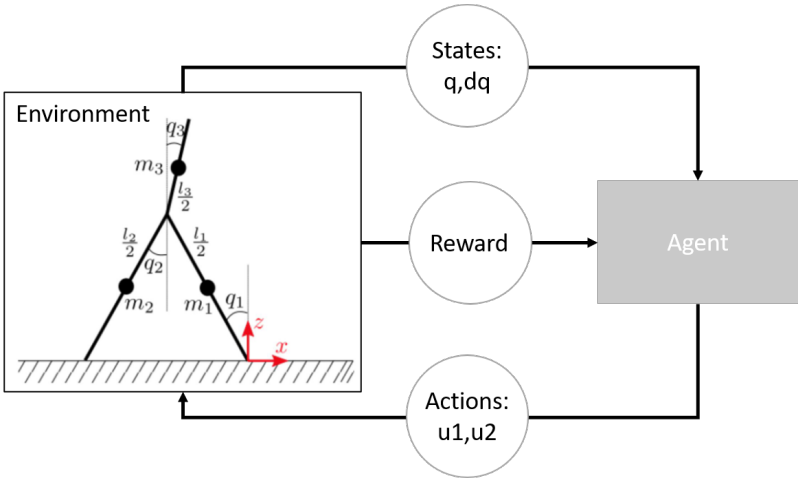


Figure 4: Scheme of the overall system

The agent consists of two different neural networks, an actor and a critic. On the Fig.5, we observe this system in more details.

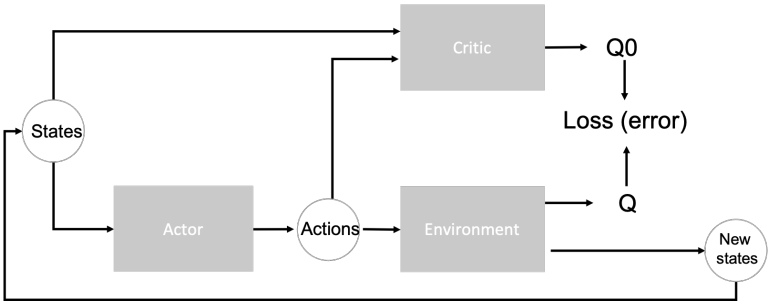


Figure 5: Scheme of the agent, it is composed of two neural networks an actor and a critique.

1. The actor takes the states q_n and dq_n and outputs the actions u_1, u_2.
2. The critic takes the states and the action and will estimate the value of the reward function Q_0.
3. The environment represents the physical world in which the robot tries to walk. It receives the actions, computes the new states q_{n+1}, dq_{n+1} and also the reward Q according to these new states.

The agent will then be optimized by trying to minimize the *Loss(error)*. Then, it will backpropagate a gradient to both the critic and the actor in order to optimize both of the networks. The detailed algorithm can be found in [4]. An important point is that the critic will only be used for the training, once the agent is trained only the actor will be used.
The agent is the same as the one presented in the Matlab tutorial [1] [2]. The actor is made of one input layer, two intermediate fully connected layers and a final output layer with a *tanh* activation function which are convenient to simulate the saturation of a motor. It is important to note that this output is between $[-1, 1]$ and are scaled outside of the agent. The critic takes the state and makes it go through two fully connected layers. The actions only pass through one fully connected layer then those two layers are added and used to produce the Q_0 output. The size of the layer could have been reduced since this problem is less complicated than the one in the tutorial. However we had no prior experience with deep reinforcement learning and could not estimate a smaller size that would be able to solve our problem so we decided to stick with the same sizes.

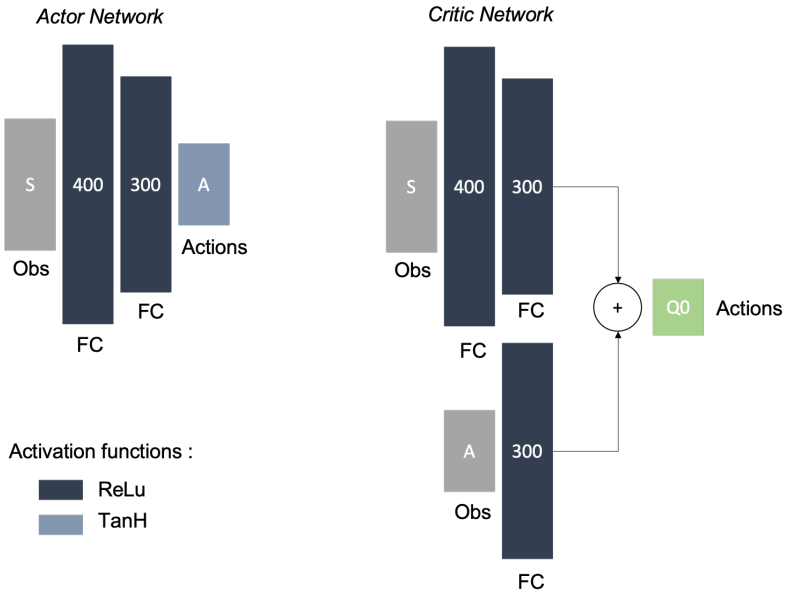


Figure 6: Description of the different layers of the agents: S = 6, A = 2 and Q0 = 1

2.2.2 Reward definition

The reward was defined through an iterative process and also with the help of the *Youtube* videos [2][1]. The final reward was given by the following equation where $r = reward$ and $p = penalty$:

$$Q = r_{sp} - p_t + r_{su} - p_{zh} - p_{zswf} - p_{q1q2} \tag{14}$$

where the terms are given by:

1. $r_{sp} = 0.5 * min(dx_h, max_{speed})$ which rewards the agent with a maximum reward set at max speed
2. $p_t = 0.125 * (q_3 - \frac{\Pi}{6})^2$ to force the agent to keep a torso angle close to $\frac{\Pi}{6}$
3. $r_{survival} = 0.5$ as explained in the video [2] to increment the reward at each step of the simulation encouraging the agent to not throw the robot on the floor
4. $p_zh = 1.5 * abs(z_h - z_{target})$ to keep the hip at a stable height
5. $p_{zswf} = 3 * z_{swf}$ to keep the swing foot close to the ground
6. $p_{q1q2} = 4 * abs(q_1 - (-q_2))$ the same constraint used in the PD controller so the agent tries to mimic this for the angles of the robot's legs

The weight of each term has been tuned to have values around 10^{-1} . With this reward the agent was able to produce a stable gait over a large amount of step. This is the standard reward which produced interesting agents, but some terms were added with underwhelming results.
The first supplementary term was a penality of the effort to avoid applying a lot of strong torques:

$$p_{effort} = 0.002 * (\sum actions^2) \tag{15}$$

However, we only found two possible behaviors. Either the agent did not optimize this parameter at all, or the term had too much impact and produced no interesting gaits. This was probably caused by a poor optimization of the weight of the factor. Other supplementary terms such as a reward of the step length:

$$r_{stl} = 25 * (x_{swf} - x_{swf0}) \tag{16}$$

This produced only agents throwing the swing foot as far as possible. As such we tried to penalize the impact event to see if the agent will try to avoid putting the swing foot down too fast.

$$p_{imp} = 0.5 * impact \tag{17}$$

Where *impact* is a boolean indicating if an impact occurred. The last term added to the reward was a penality to force the agent to produce torques similar to those of the previous PD controller:

$$p_{mimic} = 0.002\sqrt{\frac{\sum(actions^2 - u_{PD}^2)}{2}} \tag{18}$$

The goal was to minimize this term. This equation 18 was also planned to be used as a last resort if we would have produced no walking agent. We also tried to reward the step length, and to penalize the impact. However, these factors didn't succeed to improve our results. Finally, we tried to mimic a PD controller by adding a penality of the quadratic sum of the difference between the action and the PD control action, and then train the trained agent without the mimic reward. However, we realised that it was not necessary to add this step to obtain good results. Indeed, we did not get better result with this method.
Trying to fine tune parameters such as the speed, step length or step frequency is not as straightforward as we thought. As such even while trying to optimize the reward by giving a higher max_{speed} to r_{sp} this does not guarantee that an agent will walk faster. As such for the next sections in which the parameters are optimized the controller used is not always a result from a specific optimization.

2.2.3 Biped simulation protocol with RL

Once the environment and the reward have been correctly establish, we were able to train our agent. The training consists of a simulation of the biped model done multiple times and each simulation is called an episode. An episode has multiple stop conditions to prevent the agent to explore bad behaviors. The stop conditions are described as:

1. Success if the agent can walk 10 robot steps
2. Failure if $z_h < 0$ meaning that the robot fell below the ground
3. Failure if $abs(q_1)$ or $abs(q_2)$ or $abs(q_3) > \frac{\Pi}{2}$, i.e. if the robot make any angles above $\pm\frac{\Pi}{2}$ the episode stops. We added this condition as we thought that a real robot would not be able to do such angles.

During the training, i.e. the progression through episodes, the reward is maximized as it is evaluated for each episode and we adapted the code to save the agents when a certain reward threshold is reached. Once the training is over, we sort the agents saved for high reward and test them for various simulation steps.

2.2.4 Stability criterion with the number of steps

This method was able to produce stable gaits over a large number of step. We used a stable agent generated with a standard reward referring to the equation 14. It was not possible to let the robot walks up to infinity, we assumed that if the robot is able to take 1000 steps with a regular gait, it would be able to do this at infinity.

2.2.5 Minimum and maximum steady-state gait velocity

Maximization of the steady-state velocity
In order to maximize the velocity we first decided to change the reward term r_{sp} as it is directly related

to the hip velocity. We tested a maximum velocity target of 1.5 *[m/s]*. We could have tried a higher velocity target of 2 *[m/s]* but when the biped reached such a speed its gait is not stable anymore so we decided to not aim for higher target speeds.
We then tried to focus on the influence of the torso angle on the mean velocity. We have reset the r_{sp} with a target velocity of 1 *[m/s]* and changed the penalty torso p_t as $p_t = 0.125 * (q_3 - \frac{\pi}{4})^2$.

Minimization of the steady-state velocity
The minimization of the robot steady state speed is done with the same process as the maximization. The minimum velocity we succeed to set as target velocity value is 0.5 *[m/s]*. For the torso penality, we changed to $p_t = 0.125 * (q_3 - \frac{\pi}{8})^2$ while keeping the target velocity to 1 *[m/s]*.

2.2.6 Optimization for different step lengths

Maximization of the steady-state step length
For the step length maximization we tried in a preliminary stage to add an other reward parameter defined as equation 16. The purpose of this setting was to encourage the robot to throw its swing leg as far as possible to take a step. Unfortunately, we did not manage to train an agent properly with this reward. The robot threw his leg too far and could not produce an adequate gait and got stuck without being able to do a second step.
Following this, we tried to penalize the impact. We were expecting that by attempting to avoid an impact, the robot would throw its leg further away to move forward. The penality impact has been establish as equation 17, where *impact* here is a boolean which is 1 when an impact appends.

Minimization of the steady-state step length
Since maximizing step length was complicated and inconclusive, we decided to proceed differently for minimization. When optimizing the average speed, we observe that the robot reproduced a gait for which the steps length of the robot were smaller. We saved the agents from this train and we selected the ones that have a small step length while remaining stable.

2.2.7 Optimization for different step frequencies

The optimization of the step frequencies is difficult and not as intuitive as the other because the step frequency is linked to multiple factors. As discussed in the previous section, it was already complicated to optimize simple parameters. As such the step frequency was not optimized with a specific reward function, but we chose different agents that had low and high stable step frequencies from previous training.

2.3 Robustness examination against perturbations

Since multiple agents are generated from the same train and do not always reproduce the same gait, we found interesting to test the perturbation effects on two different robots from the same train and compare them.

2.3.1 External perturbations

In order to test the robustness of the robot, we added external positive and negative perturbations at the robot hip in the x direction. We add the perturbation at the 10th step during a time period of 0.2s.

2.3.2 Internal perturbations

For internal perturbations we apply a centered Gaussian noise N (0, σ) independently on each coordinate (angles and derivatives). This Gaussian noise is random and indeed produces different effects on the same

robot and σ. We decided to consider 20 steps or more as a good value to accept that the robot can tolerate internal noise.

3 Results

3.1 Energy loss during the impact

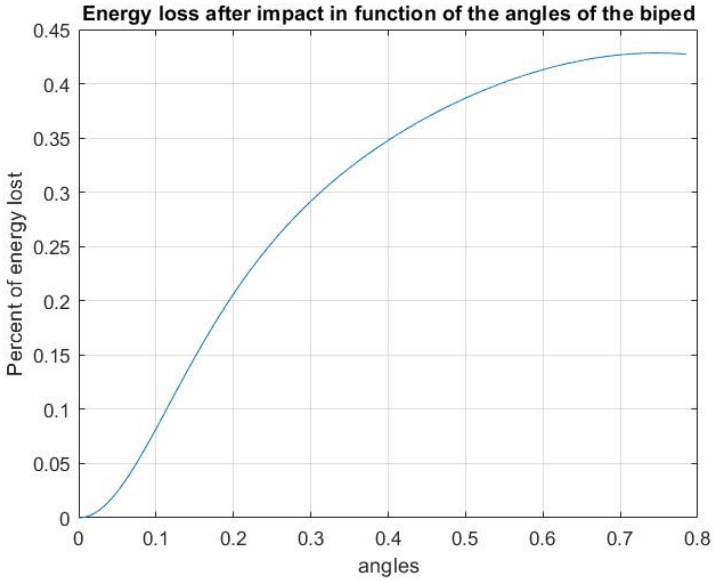
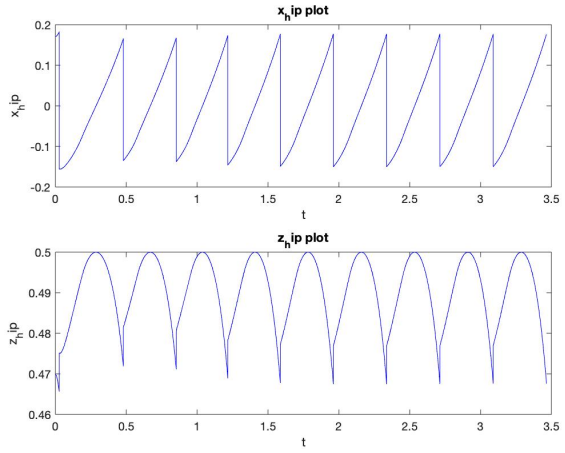


Figure 7: plot of the energy loss

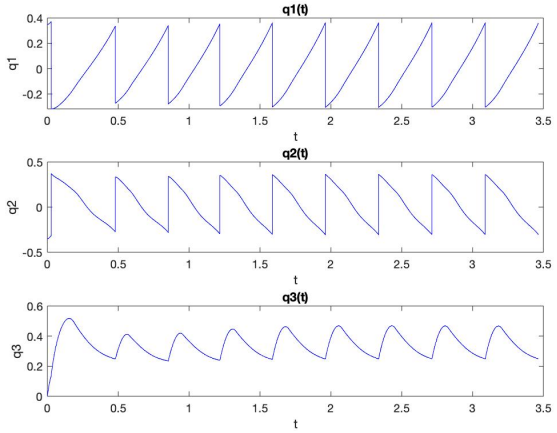
It can be observed that the potential energy does not change after the impact. Indeed, the masses m1, m2 and m3 stay at the same position during the impact.
As illustrated on Figure 7, 39.4% of the kinetic energy is lost after the impact done and this loss function could be approximated with $loss \approx \sqrt{angles}$.

3.2 Gait quality PD controller measurement

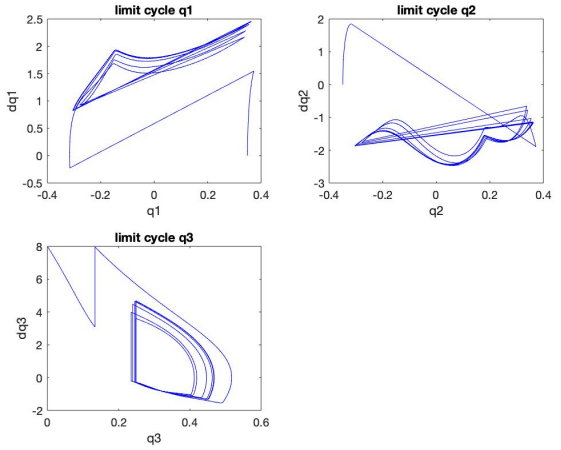
To evaluate the controller, we are using several gaits metrics and evaluation criteria. min-max velocities reached plots of the angles vs time velocity of the robot vs time displacement vs step number step frequency vs step number torques vs time normalized mean effort ($U_{max} = 30Nm$): cost of transport plot of $\dot{q}$ vs q for all three angles



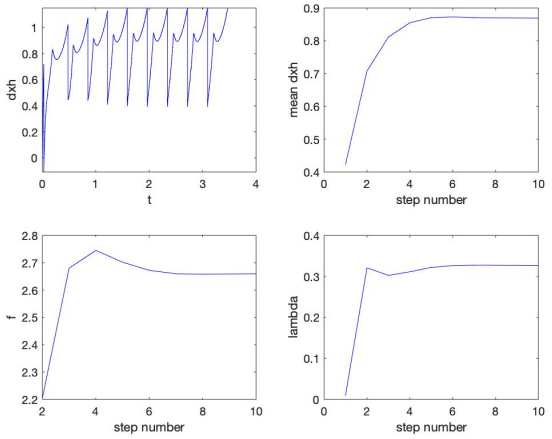
(a) plot of the hip x and z position against time



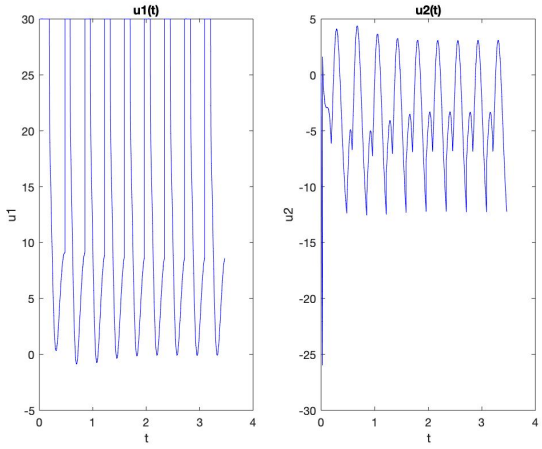
(b) plot of the angles q1,q2 and q3 against time



(c) Limit cycles : velocity against angles for q1, q2 and q3



(d) velocity against time, mean velocity, frequency and length against steps



(e) torques against time

Figure 8: PD controller gait quality measures

3.3 Training with RL

Figure 9 illustrates an evolution of agents training. Three terms are evolving during the training. The blue points are the values of the reward at each episode, the red curve is the average of the reward over 100 episodes. The green curve shows Q_0 the reward estimated by the critic network.

The evolution of the reward curve is always composed of ups and downs as expected with a DDPG.
It is not always the agents which have the highest reward that will produce the desired gait in the most stable way. We have noticed that it is when the average reward is high that the expected results occur.

We decided to stop the training evolution at approximately 3000 steps as we often tried to run the simulation over more episodes but the evolution did not provide much additional information. In Annex section 6.1 is presented a training evolution over 10'000 episodes. During the evolution the mean reward never drastically overpasses the mean reward values obtained during the 3000 first episodes which confirms the previous sentences. In addition, as a training for 10'000 episodes takes approximately 24h this leads us to reduce the number of training episodes.

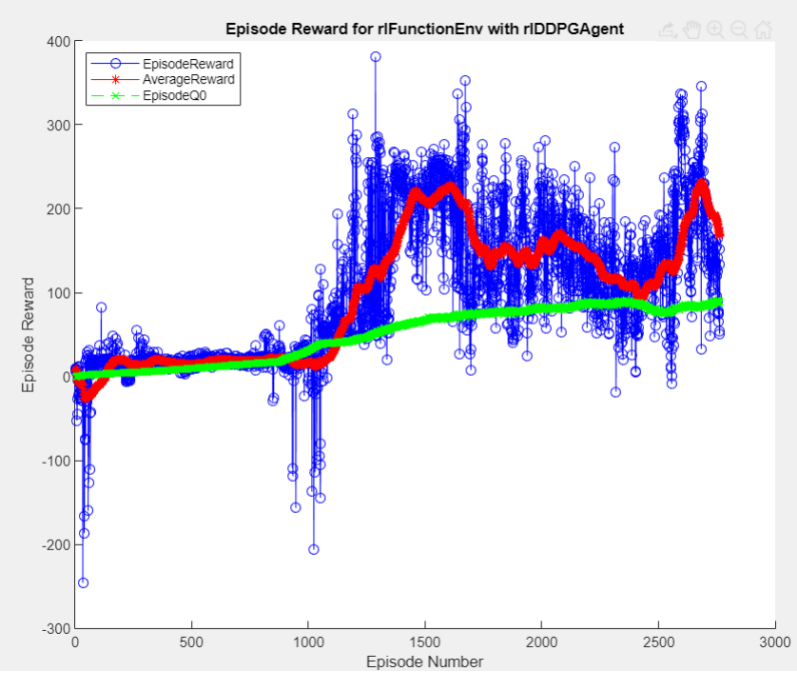
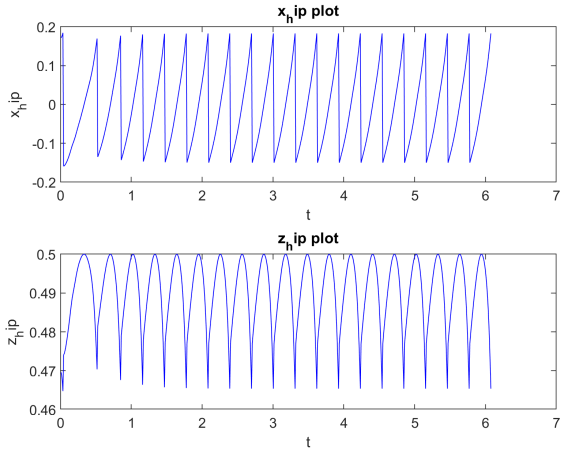


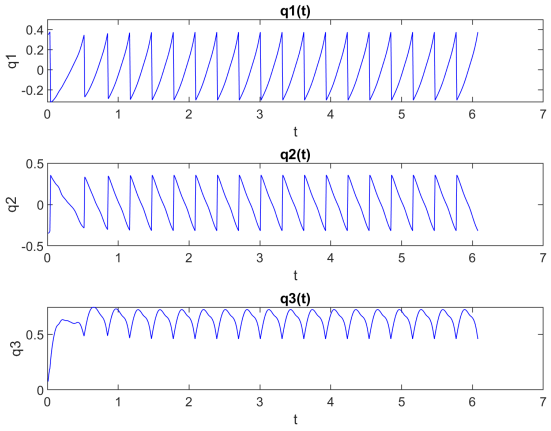
Figure 9: Training of standard agents (with penality effort) during approximately 3000 episodes

3.4 Standard results with the controller using RL

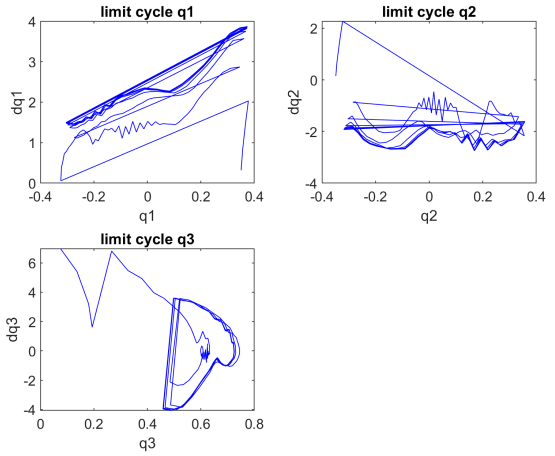
This controller was trained with the standard reward equation 14 with the addition of p_{effort} as shown in equation 15. According to Fig.10 this agent produces a stable gait at *1.2 [m/s]* the Figure 10d only shows 20 steps but it would continue walking in steady state for a large number of steps.



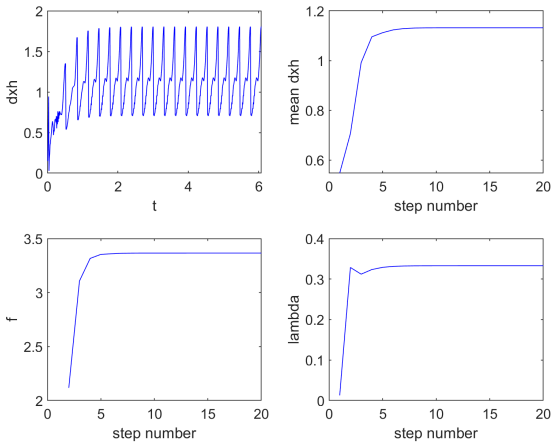
(a) plot of the hip x and z position against time



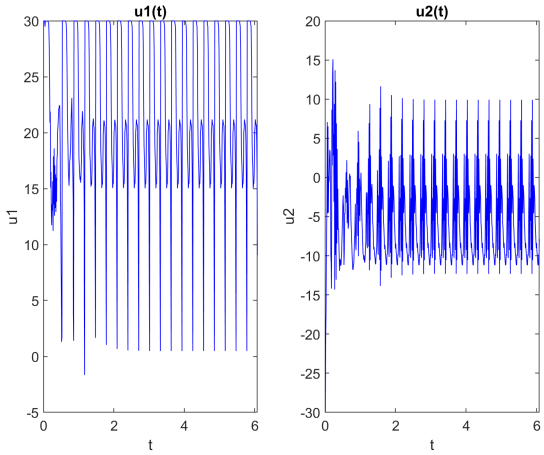
(b) plot of the angles q1,q2 and q3 against time



(c) Limit cycles : velocity against angles for q1, q2 and q3



(d) velocity against time, mean velocity, frequency and length against steps



(e) torques against time

Figure 10: Gait quality measures of the agent we will use for perturbations

3.5 Parameters optimisation

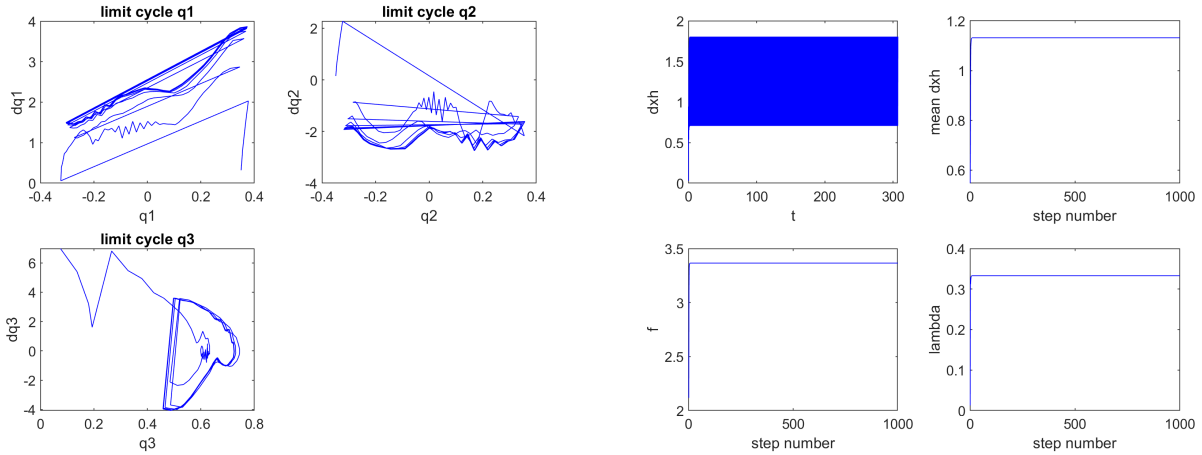
In the annex section 6.2 is a demonstrative video link to observe the different gait and results we found interesting during the parameters optimisation.

Table 1: Summary of extreme values obtained with different agents.

Min/Max velocity	Min/Max step length	Min/Max step frequency	Max external perturbation	Max internal perturbation for the joint angles and angular velocities	Max steps
0.7/1.7	0.26/0.45	1.7/4.2	[-250,102]	[0.11, 0.13, 0.25] [0.7, 1.6, 1.7]	inf

3.5.1 Stability criterion with the number of steps

To test the maximum number of steps our robot can take, we used a trained agent to walk 1000 steps. As we can see on Figure 11b, the robot walks the 1000 steps with a regular gait. The mean velocity and step length graphs stabilize very quickly at a plateau, which illustrates the stability of the robot's movement throughout the simulation.
If we focus on the Figure 11a we can see that the limit cycles are clearly defined which means that each gait cycle are stable over the 1000 steps.
With all this information gathered, we decide to assume that our robot would be able to move infinitely without falling down.



(a) Limit cycles : velocity against angles for q1, q2 and q3

(b) velocity against time, mean velocity, frequency and length against steps

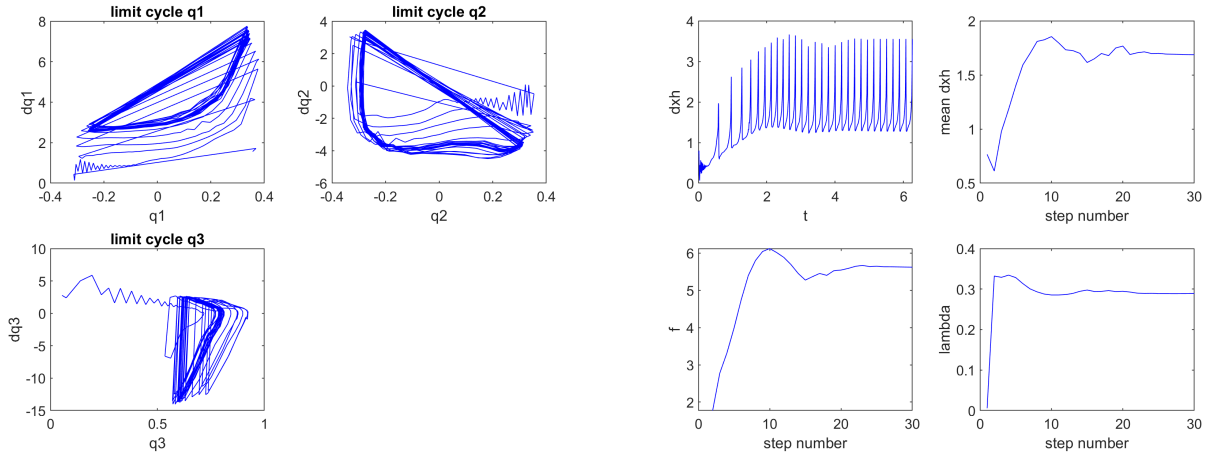
Figure 11: Maximum steps number measures

3.5.2 Minimum and maximum steady-state gait velocity

Maximization of the steady-state velocity

The maximization of the stead-state velocity has been done through an iterative process (see section 2.2.5). The Figure 12 illustrates the final simulation we arrived to produce as velocity optimisation. The reward we used was the standard reward from equation 14 except that the torso penality was modified to force the robot to bend further forward with the torso to $\frac{\Pi}{4}$ as explained in the theoretical section.

We tested this agent on 30 steps stimulation. The Figure 12b shows us that the steady state velocity is approximately 1.7 *[m/s]* and we also can see that the rest of the parameters on this figure are stable too. The Figure 12a also confirm that the robot gait is quite stable.

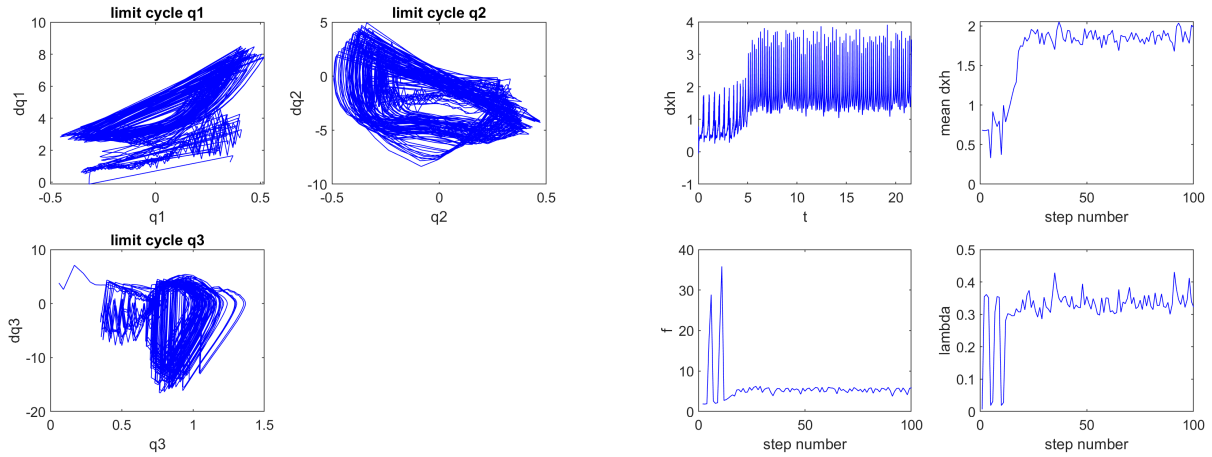


(a) Limit cycles : velocity against angles for q1, q2 and q3

(b) velocity against time, mean velocity, frequency and length against steps

Figure 12: Maximum steady-state velocity measures

Figure 13 illustrates a case when the robot is not stable. For this simulation we launched a train with a reward target velocity of 1.5 *[m/s]*. We can see that the robot try to reach a mean velocity value of 2 *[m/s]* (see Figure 13b). This velocity is too high for this agent which lead its gait to instability as we can clearly see on Figure 13a. Because of this simulation we modified the reward with the torso penality and got the final results presented above.



(a) Limit cycles : velocity against angles for q1, q2 and q3

(b) velocity against time, mean velocity, frequency and length against steps

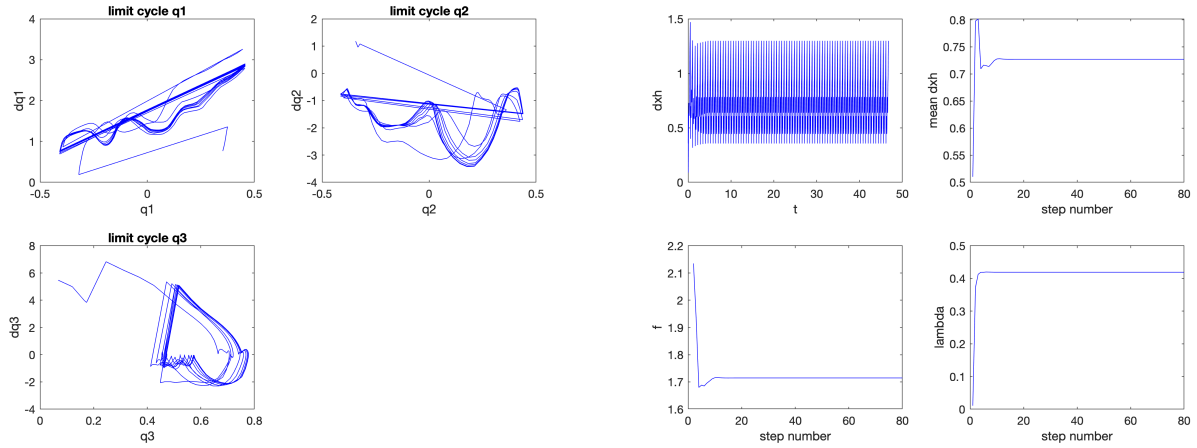
Figure 13: Example of an high mean velocity but unstable measures

Minimization of the steady-state velocity

We selected an agent which have been trained with a reward including the impact penality as explained in section 2.2.5. For the testing of this agent, we made the robot walk 80 steps.

On Figure 14a we can see that the gait of the robot can be considered as stable throughout its course. By looking at Figure 14b we see that the steady-state minimum velocity is 0.7 *[m/s]*. The gait of the robot

during the animation confirms that the robot is stable for this minimum speed.
These results underline the fact that it is not always easy to tune a gait with the expected parameters. It is neither with the reward of target velocity nor with an influence on the torso angle but with the impact reward that we were able to obtain the desired gait.



(a) Limit cycles : velocity against angles for q1, q2 and q3

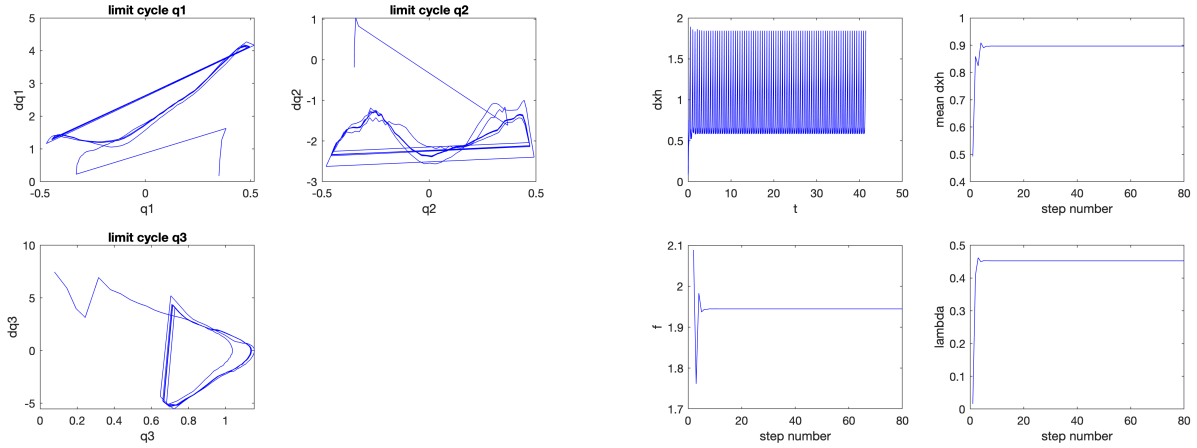
(b) velocity against time, mean velocity, frequency and length against steps

Figure 14: Minimum steady-state velocity measures

3.5.3 Optimization for different step lengths

Maximization of the steady-state step length

In order to maximize the steady-state step length we used the approach explained in section 2.2.6 for the training. Indeed, this agent was extracted from the training set where we tried to reduce the hip velocity target value, in this case the target was 0.5 *[m/s]*.
The robot has been simulated for 80 steps and the Figure 15a shows a good gait stability. On the figure 15b, we can see that the robot reached a maximal steady-state step length of 0.45 *[m]* during the whole simulation. The others parameter as the mean velocity and the step frequency are also stable.



(a) Limit cycles : velocity against angles for q1, q2 and q3

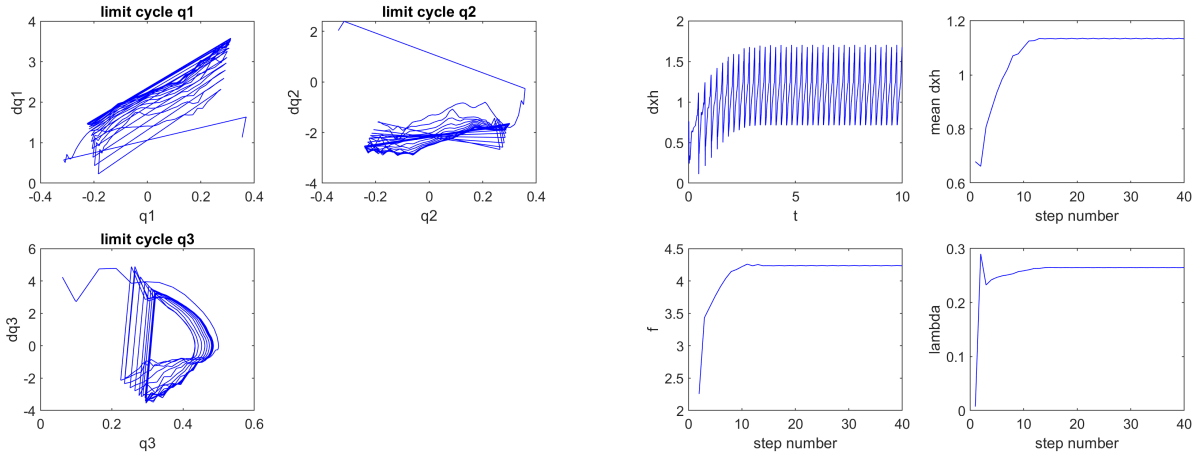
(b) velocity against time, mean velocity, frequency and length against steps

Figure 15: Maximum steady-state step length measures

Minimization of the steady-state step length

The minimal step length was produced with a train using the standard reward from equation 14 but with an added p_{effort} as in equation 15. Furthermore, the reward target speed was as explained in section 2.2.6 set to 1.5 *[m/s]*.

As illustrated on Figure 16b this agent can produce a stable gait with a step length of *0.26[m]* and has been simulated for 40 steps. The Figure 16a confirms with the limit cycles that the gait is quite stable. The animation of this simulation shows that the robot walks with a high velocity while doing small steps.



(a) Limit cycles : velocity against angles for q1, q2 and q3

(b) velocity against time, mean velocity, frequency and length against steps

Figure 16: Minimum steady-state step length measures

3.5.4 Optimization for different step frequencies

As said in the section 2.2.7 this value was not specifically optimized. The minimum step frequency was $1.7[\frac{step}{s}]$. It was achieved by the agent walking the slowest as can be seen in Fig.14. The maximum step frequency was $4.2[\frac{step}{s}]$. This value was reached by the same agent from Fig.16.

3.5.5 External perturbations

We chose two trained agents to compare the resilience to external perturbations: the standard agent from the section 3.4 and another stable agent from the same training.

For the pushing perturbation (positive value), we observe that our robot remains stable until a force of 21 N with the standard agent. The second agent remains stable until a force of 102 N (Figure 17). We can see that the robot needs more steps to recover from the pushing perturbation.

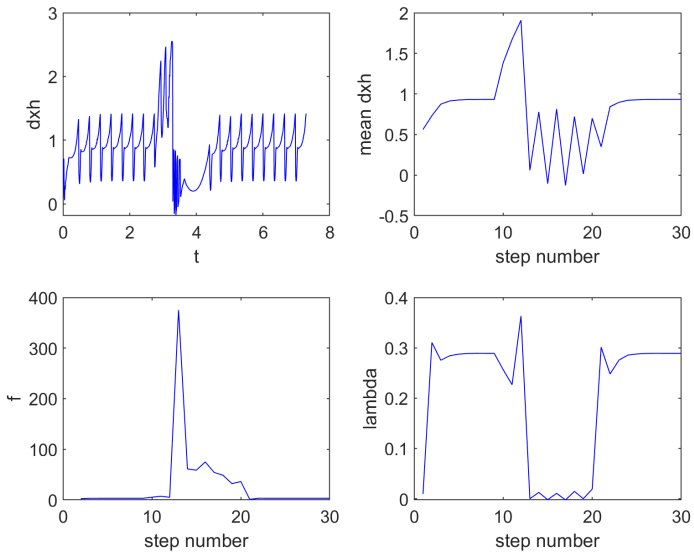


Figure 17: Gait quality measures for a pushing perturbation of 102N

For the pulling perturbation (negative value), our robot handles a maximum force of -150 N. The other agent remains stable until a force of -250 N (Figure 18). We can observe that the robot recovers quite fast from the perturbation without falling.

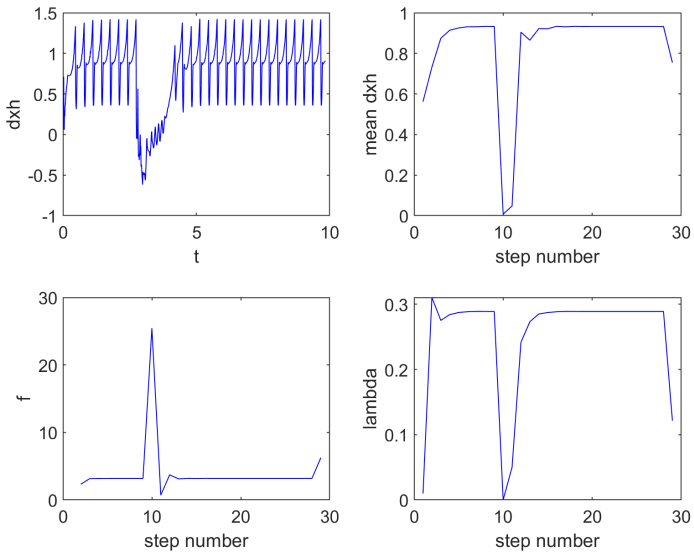


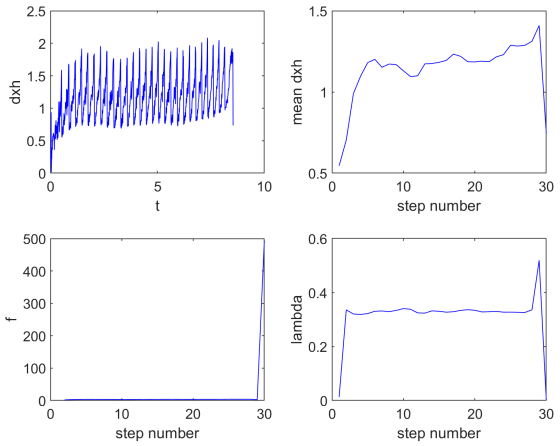
Figure 18: Gait quality measures for a pulling perturbation of 250N

This important difference between push and pull values shows that the center of gravity is at the front of the robots, due to the different rewards during training. Nevertheless, this results are good considering the robot has not been trained to remains from external perturbation.

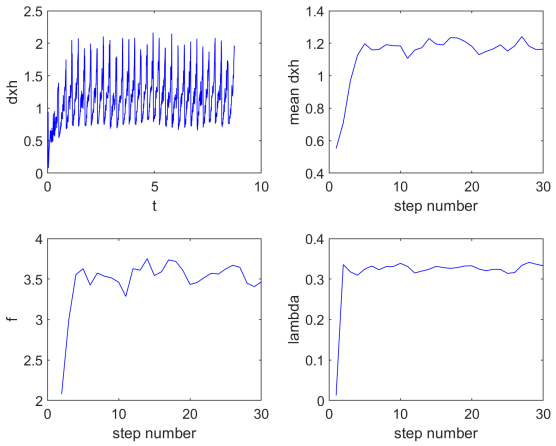
3.5.6 Internal perturbations

First robot test:
As we can see on Figures 19 and 20, the robot can not handle big internal perturbations, in fact the

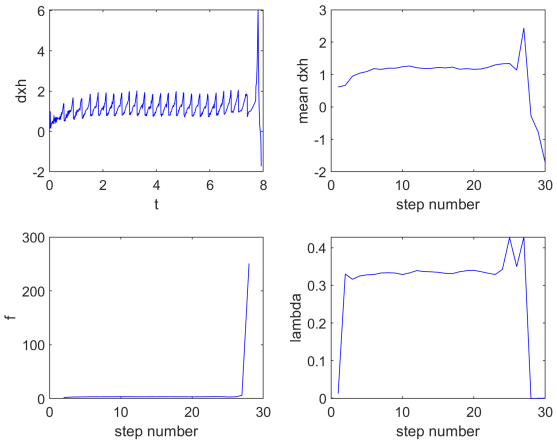
graphs are really similar to those in Fig.10. When the robot deviates a bit too much of its standard values it falls immediately, and the agent can't recover.



(a) velocity against time, mean velocity, frequency and length against steps for an internal perturbation of $\sigma = 0.022[rad^2]$ on q1

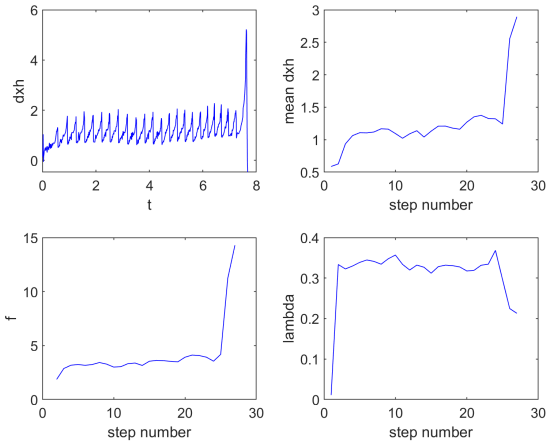


(b) velocity against time, mean velocity, frequency and length against steps for an internal perturbation of $\sigma = 0.021[rad^2]$ on q2

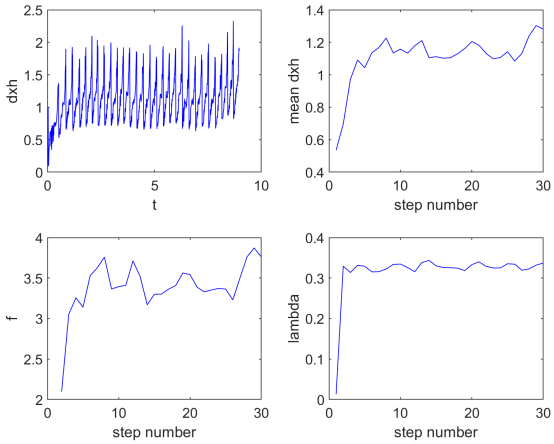


(c) velocity against time, mean velocity, frequency and length against steps for an internal perturbation of $\sigma = 0.07[rad^2]$ on q3

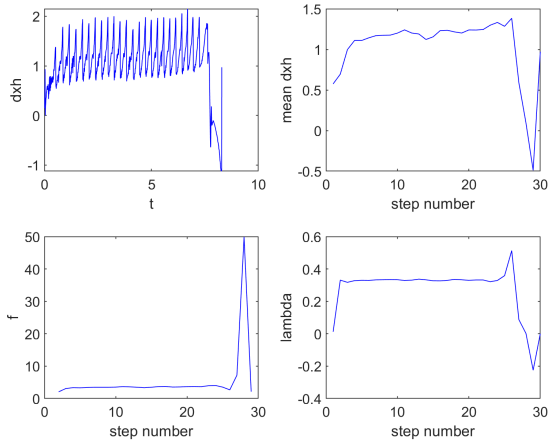
Figure 19: perturbation on the joint angles q1, q2 and q3 for the second agent measures



(a) velocity against time, mean velocity, frequency and length against steps for an internal perturbation of $\sigma = 0.35[rad^2/s^2]$ on dq1



(b) velocity against time, mean velocity, frequency and length against steps for an internal perturbation of $\sigma = 0.5[rad^2/s^2]$ on dq2

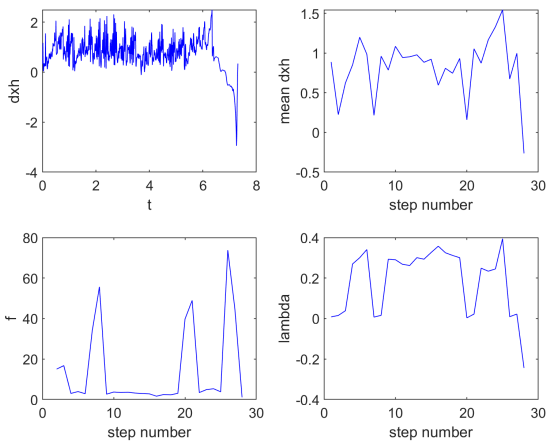


(c) velocity against time, mean velocity, frequency and length against steps for an internal perturbation of $\sigma = 0.35[rad^2/s^2]$ on dq3

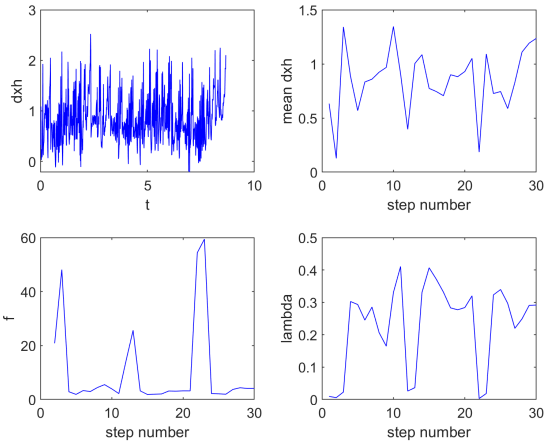
Figure 20: perturbation on the angular velocities dq1, dq2 and dq3 for the first agent measures

Second robot test :

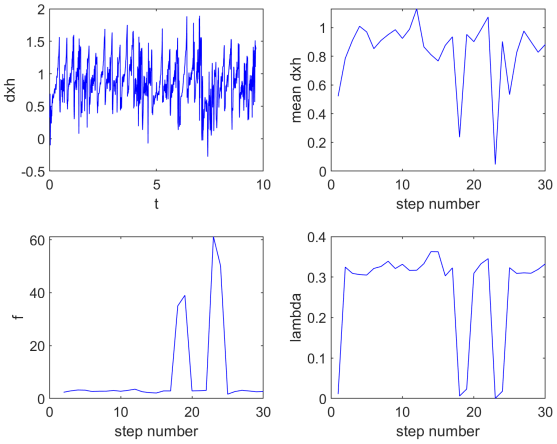
We took an other agent from the same train as the first agent but from another episode. This time the robot can handle higher noise variances. By looking at the graphs from Fig.19 and 22 the agent seems to do mistakes such as putting the swing foot down too fast. But the agent is now able to recover from such errors.



(a) velocity against time, mean velocity, frequency and length against steps for an internal perturbation of $\sigma = 0.11[rad^2]$ on q1

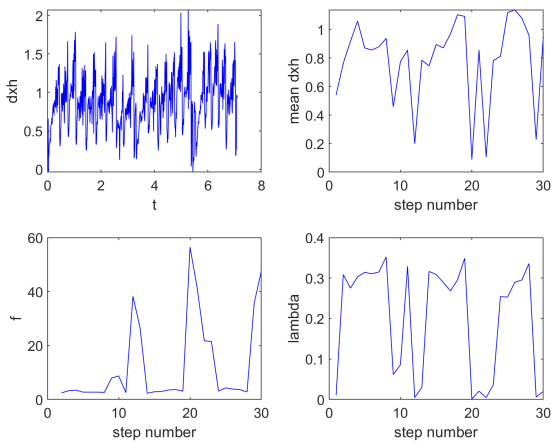


(b) velocity against time, mean velocity, frequency and length against steps for an internal perturbation of $\sigma = 0.13[rad^2]$ on q2

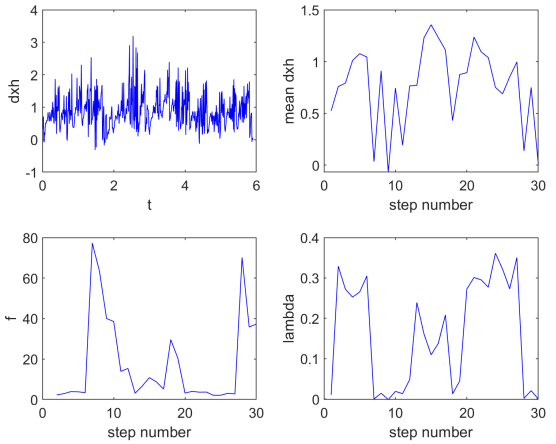


(c) velocity against time, mean velocity, frequency and length against steps for an internal perturbation of $\sigma = 0.25[rad^2]$ on q3

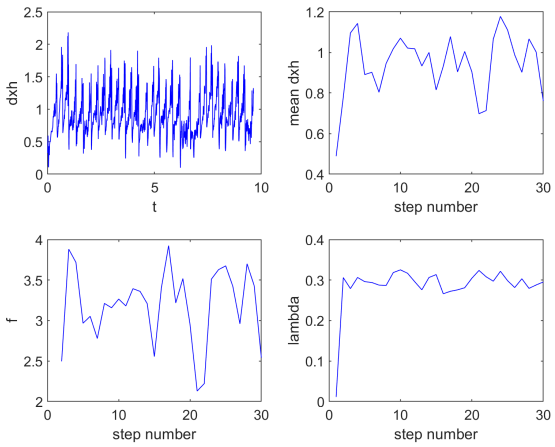
Figure 21: perturbation on the joint angles q1, q2 and q3 for the second agent measures



(a) velocity against time, mean velocity, frequency and length against steps for an internal perturbation of $\sigma = 0.7[rad^2/s^2]$ on dq1



(b) velocity against time, mean velocity, frequency and length against steps for an internal perturbation of $\sigma = 1.6[rad^2/s^2]$ on dq2



(c) velocity against time, mean velocity, frequency and length against steps for an internal perturbation of $\sigma = 1.7[rad^2/s^2]$ on dq3

Figure 22: perturbation on the angular velocities dq1, dq2 and dq3 for the second agent measures

In summary, the two controllers can tolerate the following maximum standard deviation for each coordinate independently:

coordinate	q1	q2	q3	dq1	dq2	dq3
maximum standard deviation σ for Robot 1	0.022	0.021	0.07	0.35	0.5	0.35
maximum standard deviation σ for Robot 2	0.11	0.13	0.25	0.7	1.6	1.7

Table 2: Maximum internal perturbation apply to the joints angles and velocities

3.6 Compete and reflect

3.6.1 Novelty

Starting from reliable kinematics and dynamics model, we chose to optimize our controller with deep reinforcement learning. Indeed, this method is getting more and more popular in the field of legged robot, because its purpose to learn the angles and velocities of the angles directly from a system of rewards. Well tuned, it could therefore produce very adaptable model. Hence, the problem is not to design a good controller, but to design a good reward. We tried different novelties to achieve that. You can find detailed explanations in the reward definition 2.2.2, notably by adding a reward to mimic the controller and then train this trained agent without the mimic controller reward. Another of our idea was to let the robot walks for 1 or 2 steps with the controller, during these steps the reward would be $1-p_{mimic}$. After those first 2 steps the control of the robot will be given back to the agent and the reward would be the standard reward as in equation 14 in order to help the robot to walk, but we didn't succeed to have successful training agent with this reward.

3.6.2 Performance comparison

The two controllers implemented, virtual constraints and RL showed very different results. As a first point the RL was able to produce a large amount of different stable gaits. These gaits would have been really complicated to hand design. Another advantage of RL is that it was able to push the limit of the robot further by making it walk at a maximum stable speed of 1.7 [m/s] as shown in Fig.12b.
Comparing the different gait quality measures with Fig.8 and Fig.10 shows how different the gaits are. Both reach a steady state in less than 10 steps. However, the RL uses bigger torques than the virtual constraints, this can be solved by training the agent with a higher weight on the penality of the effort.
Despite all these advantages, RL still comes with many drawbacks. The main one being the time to train the agents which can go up to more than 20 hours. Another issue is that the agent are not able to handle well situation they have never seen before. The best example is how much their behaviors can vary when exposed to perturbations. As discussed in the section about perturbations some agents are almost unable to handle noise in the joints measurements or an external force.
The last important difference is that with the virtual constraint controller parameters can be fine tuned by hand, which is really convenient when you need to have a slight difference in behavior. With RL there are no parameters to fine tune as such when you want to do a small modification of the gait you need to run another long training. Furthermore optimizing a parameter with a reward function is far from being intuitive, as it was shown when we wanted to produce different step lengths or mean velocities.

4 Discussion

The three-link 2D biped control with RL showed promising results as we succeeded to optimize several parameters. However, the training of agents with such methods remains much more complicated compared to conventional controllers such as the PD presented in this report. Indeed, we encountered computation costs issue (the training time is more than 20 hours with our computers) so we could have done more training and fitting with faster computation. In spite of that, there are always at least three to four agents who are very robust and exploitable produced by the training if we set it with an effective reward. In addition, part of the learning process of the agent is done by balancing the weights of the neural network in a way that is not intuitive for us, which makes the fine tuning process even more complex.

Multiple improvements could be made on this model. First of all, we could modify the networks of our agents (actor and critic). In this project we kept the sizes proposed by the sources [1] [2]. Due to our lack of knowledge in the field we did not try to explore this option in the framework of our project. We could also try to do transfer learning with existing models.
Another improvement that we could have made was to use trained agents when we wanted to fine tune parameters. This would have avoided to start a new training and explore again useless solutions. This process could have save a lot of time.
We thought of multiple training strategies such has letting the robot walk with the virtual constraint controller and force the agent to imitate the torques produced by defining the reward as $1 - p_{mimic}$ with p_{mimic} as in equation 18. Then we could have trained an agent, that would already be able to walk, for other parameters. Or letting the robot walk for 1 or 2 steps as explained in the part 3.6.1. It could have been interesting to explore in this direction.
As we have seen in the section about the perturbations, the agents could be more or less robust. This could have been improved by introducing the agents to perturbations during the training phase. However, the training will probably take even more time.

5 Conclusion

With the goal of developing a controller by using RL, we presented a method that proves it efficiency in order to make a three-link 2D biped walking. First of all we encountered some problems in the process of developing the RL environment and adapting the existing code to the Matlab toolbox for RL. This questioning helped us to understand the tools we were using and allowed us to acquire knowledge in a field that was groundbreaking for us. As part of the project, we managed to create walking robots with optimized parameters such as average speed and step length for example. We have also noticed that the resistance of our generated agents to internal and external perturbations was very different from one agent to another. This provides great avenues for investigation and ideas for extending the project, such as agent learning with PD model or with disruptions.

6 Annex

6.1 Training of standard agents during 10'000 episodes

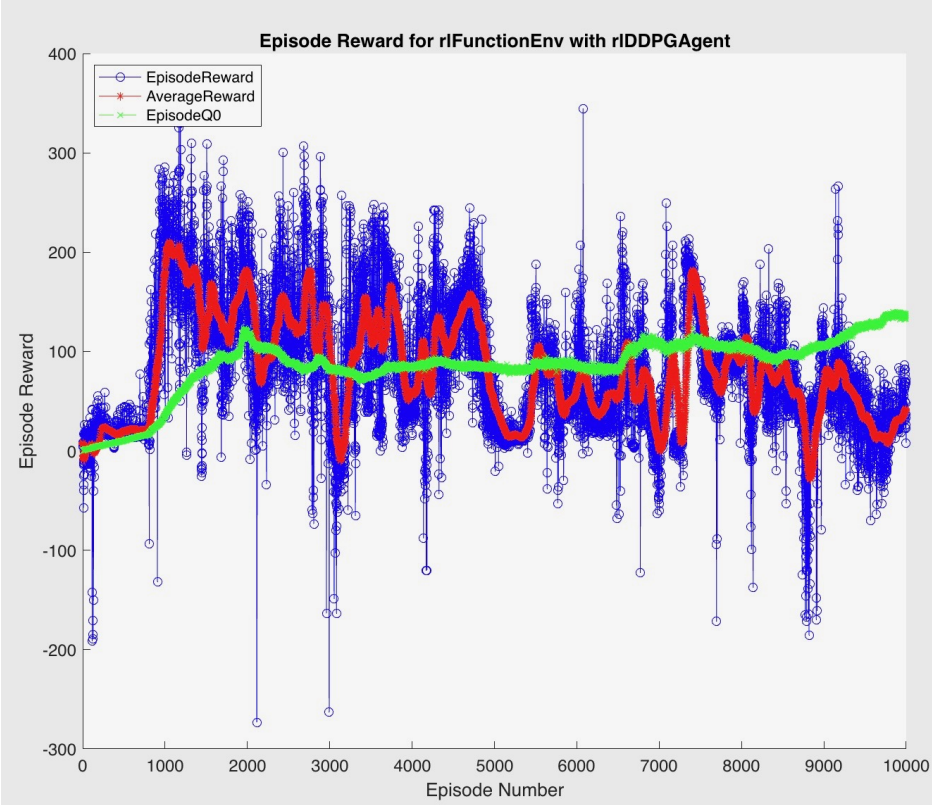


Figure 23: Training of standard agents during 10'000 episodes

6.2 Link to the demonstrative video

https://drive.google.com/drive/folders/12kLfMDkYwHm-nzbuvfvTz27RLLYvB20w?usp=sharing

References

[1] https://www.mathworks.com/videos/deep-reinforcement-learning-for-walking-robots–1551449152203.html?s_eid=PSM_15028 (05.12.2020)

[2] https://www.youtube.com/watch?v=Wypc1a-1ZYA (05.12.2020)

[3] https://www.mathworks.com/help/reinforcement-learning/ug/train-biped-robot-to-walk-using-reinforcement-learning-agents.html (05.12.2020)

[4] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess,Tom Erez, Yuval Tassa, David Silver Daan Wierstra, *Continuous control with deep reinforcement learning* (2016)