Developing a Walking Controller for a Three-link 2D Biped

Océane Ouillon, Nicolas Lefebure

Autumn 2019

1 Introduction

As future engineers in the world of mechanics and robotics, understanding how to control a robot and design such control is crucial. But before trying our ideas on a real robot, there is a necessary procedure: modelling the robot in its environment, then design a control method and finally simulate the walking robot. The last two steps are meant to be repeated over and over until a suitable control mechanism is found. The following Control Design Pipeline represents the thinking procedure that we have adopted for this project:

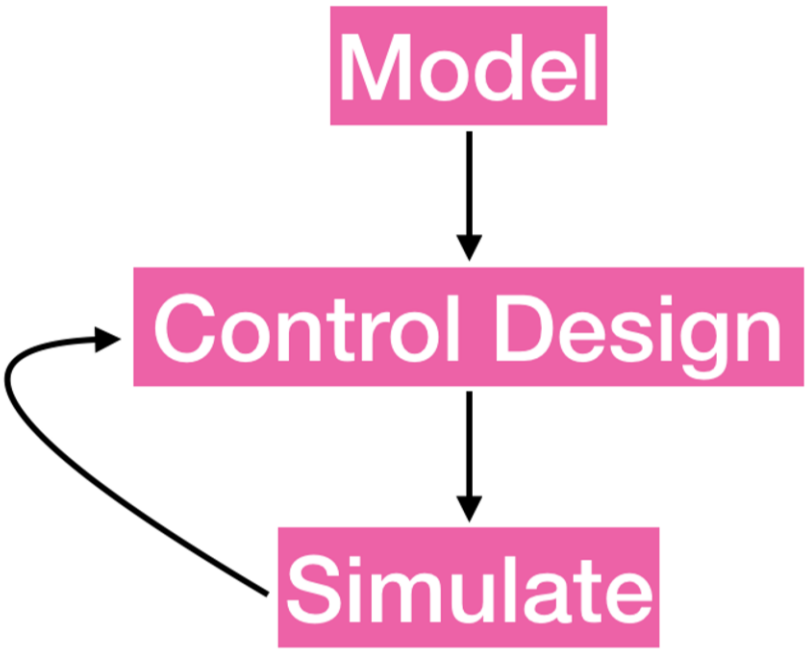


Figure 1: Control Design Pipeline

2 Methods

This section explains the methodology used for the development of the controller. First, in section 2.1 we settle the equations of motion that describes the behavior of the Biped walker. Next, in section 2.2, we simulate the walker by solving its equations of motion. Finally, in the section 2.3, using previous results, we implement two walking controllers: PD controller and trajectory planing. It is important to state that everything was done using MATLAB environment.

2.1 Equations of motion

2.1.1 Kinematics

Without considering forces acting on the robot, we can first develop the geometry of motion. The robot is modeled as an assembly of three link (two legs and a torso). Each link, identified by the index i, is represented as a rod of length l_i with centered mass m_i and located with respect to the hip with angles q_i, where i = 1,2,3. At first, we consider the origin of coordinates at the location of the front foot. The figure 2 shows the inertial frame in which we describe the position and velocities of the masses.

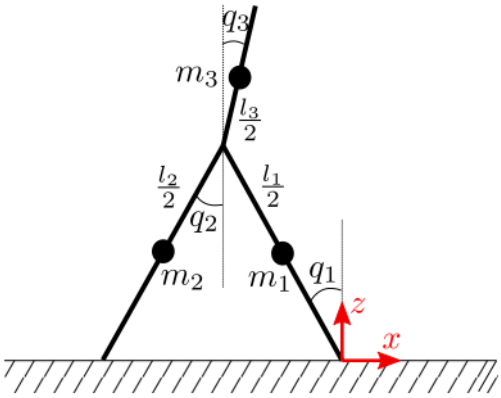


Figure 2: Model of the robot

The coordinates of each mass can be computed:

$$\begin{aligned} x_1 &= \tfrac{1}{2} \cdot l_1 \sin(q_1) \quad x_2 = l_1 \sin(q_1) - \tfrac{1}{2} \cdot l_2 \sin(q_2) \quad x_3 = l_1 \sin(q_1) + \tfrac{1}{2} \cdot l_3 \sin(q_3) \\ z_1 &= \tfrac{1}{2} \cdot l_1 \cos(q_1) \quad z_2 = l_1 \cos(q_1) - \tfrac{1}{2} \cdot l_2 \cos(q_2) \quad z_3 = l_1 \cos(q_1) + \tfrac{1}{2} \cdot l_3 \cos(q_3) \end{aligned} \tag{1}$$

The same can be done for the velocities by taking the derivative of the position:

$$\begin{aligned} \dot{x_1} &= \tfrac{1}{2} \cdot l_1 \cos(q_1) \cdot \dot{q_1} \quad \dot{x_2} = l_1 \cos(q_1) \cdot \dot{q_1} - \tfrac{1}{2} \cdot l_2 \cos(q_2) \cdot \dot{q_2} \quad \dot{x_3} = l_1 \cos(q_1) \cdot \dot{q_1} + \tfrac{1}{2} \cdot l_3 \cos(q_3) \cdot \dot{q_3} \\ \dot{z_1} &= -\tfrac{1}{2} \cdot l_1 \sin(q_1) \cdot \dot{q_1} \quad \dot{z_2} = -l_1 \sin(q_1) \cdot \dot{q_1} + \tfrac{1}{2} \cdot l_2 \sin(q_2) \cdot \dot{q_2} \quad \dot{z_3} = -l_1 \sin(q_1) \cdot \dot{q_1} - \tfrac{1}{2} \cdot l_3 \sin(q_3) \cdot \dot{q_3} \end{aligned} \tag{2}$$

Also, the position of particular points of the walker can be computed, as they will be useful later for the simulation and control. In particular, the position of the swing foot, the hip and the torso:

$$\begin{aligned} x_h &= l_1 \sin(q_1) \quad x_{swf} = l_1 \sin(q_1) - l_2 \sin(q_2) \quad x_t = l_1 \sin(q_1) + l_3 \sin(q_3) \\ z_h &= l_1 \cos(q_1) \quad z_{swf} = l_1 \cos(q_1) - l_2 \cos(q_3) \quad z_t = l_1 \cos(q_1) + l_3 \cos(q_3) \end{aligned} \tag{3}$$

The development of the calculation is presented in the live MATLAB script *generate_kynematics.mlx*

2.1.2 Dynamics

Now, we allow ourselves to consider the forces in order to calculate the dynamics of our system. The goal is to find the mass matrix, M, the Coriolis Matrix C, the gravity Matrix G as well as the control matrix B that will describe the system with the following equation:

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

We can find the equation by using Lagrange equations. Thus, the Lagrangian is calculated using kinetic and potential energy:

$$\begin{aligned}
T_1 &= \tfrac{1}{2}\cdot m_1 \cdot \left(\dot{x_1}^2 + \dot{z_1}^2\right) = \tfrac{1}{8}\cdot m_1 \cdot \dot{q_1}^2 l_1^2 \\
T_2 &= \tfrac{1}{2}\cdot m_2 \cdot \left(\dot{x_2}^2 + \dot{z_2}^2\right) = \tfrac{1}{2}\cdot m_2 \cdot \left(\left(\dot{q_1} l_1 \cos(q_1) - \tfrac{1}{2}\dot{q_2} l_2 \cos(q_2)\right)^2 + \left(\dot{q_1} l_1 \sin(q_1) - \tfrac{1}{2}\dot{q_2} l_2 \sin(q_2)\right)^2\right) \\
T_3 &= \tfrac{1}{2}\cdot m_3 \cdot \left(\dot{x_3}^2 + \dot{z_3}^2\right) = \tfrac{1}{2}\cdot m_3 \cdot \left(\left(\dot{q_1} l_1 \cos(q_1) + \tfrac{1}{2}\dot{q_3} l_3 \cos(q_3)\right)^2 + \left(\dot{q_1} l_1 \sin(q_1) + \tfrac{1}{2}\dot{q_3} l_3 \sin(q_3)\right)^2\right) \\
V_1 &= m_1 \cdot g \cdot z_1 = \tfrac{1}{2}\cdot m_1 \cdot g \cdot l_1 \cos(q_1) \\
V_2 &= m_2 \cdot g \cdot z_2 = m_2 \cdot g \cdot \left(l_1 \cos(q_1) - \tfrac{1}{2}\cdot l_2 \cos(q_2)\right) \\
V_3 &= m_3 \cdot g \cdot z_3 = m_3 \cdot g \cdot \left(l_1 \cos(q_1) + \tfrac{1}{2}\cdot l_3 \cos(q_3)\right)
\end{aligned} \tag{5}$$

Hence, the Lagrangian is:

$$\begin{aligned}
&T = T_1 + T_2 + T_3, \quad V = V_1 + V_2 + V_3 \\
&L = T - V
\end{aligned} \tag{6}$$

From the equation of Lagrange given by,

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{q_i}}\right) - \frac{\partial L}{\partial q_i} = 0, i = 1,2,3 \tag{7}$$

We can deduce the matrices:

$$G = \begin{pmatrix} -\frac{1}{2} g l_1 \sin(q_1)\left(m_1 + 2m_2 + 2m_3\right) \\ \frac{1}{2} g l_2 m_2 \sin(q_2) \\ -\frac{1}{2} g l_3 m_3 \sin(q_3) \end{pmatrix}$$

$$M = \begin{pmatrix} l_1^2\left(\frac{1}{4}m_1 + m_2 + m_3\right) & -\frac{1}{2} l_1 l_2 m_2 \cos(q_1 - q_2) & \frac{1}{2} l_1 l_3 m_3 \cos(q_1 - q_3) \\ -\frac{1}{2} l_1 l_2 m_2 \cos(q_1 - q_2) & \frac{1}{4} l_2^2 m_2 & 0 \\ \frac{1}{2} l_1 l_3 m_3 \cos(q_1 - q_3) & 0 & \frac{1}{4} l_3^2 m_3 \end{pmatrix} \tag{8}$$

$$C = \begin{pmatrix} 0 & -\frac{1}{2}\dot{q}_2 l_1 l_2 m_2 \sin(q_1 - q_2) & \frac{1}{2}\dot{q}_3 l_1 l_3 m_3 \sin(q_1 - q_3) \\ \frac{1}{2}\dot{q}_1 l_1 l_2 m_2 \sin(q_1 - q_2) & 0 & 0 \\ -\frac{1}{2}\dot{q}_1 l_1 l_3 m_3 \sin(q_1 - q_3) & 0 & 0 \end{pmatrix}$$

Now that we have matrices G, M and C, the remaining matrix B is computed by looking at the actuators. As shown in the figure 3 below we have two controllers. They drive the angles between the stance leg and torso and between the swing leg and torso.

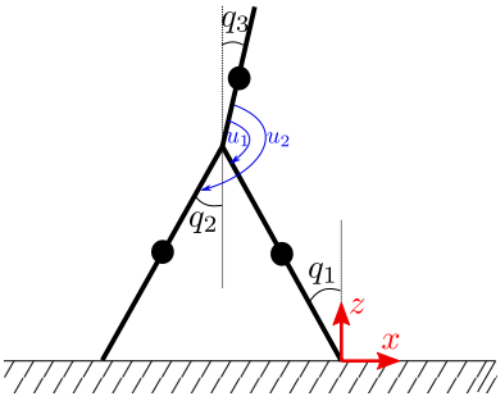


Figure 3: Actuator u_1 and u_2 acting on the robot

We model the external torques done by the actuators as virtual work. Thus, as the angle between link 1 and link 3 is $\theta_1 = \pi + q_1 - q_3$ and the angle between link 2 and link 3 is $\theta_2 = \pi + q_2 - q_3$, the virtual work done by the actuators is:

$$\delta W = \delta W_1 + W_2 = u_1 \cdot \delta\theta_1 + u_2 \cdot \delta\theta_2 = u_1 \cdot (\delta q_1 - \delta q_3) + u_2 \cdot (\delta q_2 - \delta q_3) \tag{9}$$

From the identification of the coefficients of the expression above, we can deduce the B matrix:

$$B = \begin{pmatrix} 1 & 0 \\ 0 & 1 \\ -1 & -1 \end{pmatrix} \tag{10}$$

The development of the calculation is presented in the live MATLAB script *generate_dynamics.mlx*

2.1.3 Hybrid dynamics: impact model

The bipedal walking dynamics model of the walker is hybrid: the equations are adapted as the stance leg and swing leg interchange after each cycle. When the biped completes a step, the swing foot hits the ground and afterwards the swing leg becomes the new stance leg and the stance leg becomes the new swing leg. The impact map Δ defines this transition. It maps the state of the robot right before the impact to its state right after the impact. Mathematically, we have:

$$\Delta\left(q^-, \dot{q}^-\right) = \left(q^+, \dot{q}^+\right) \tag{11}$$

where $(q^-, \dot{q}^-)$ is the state of the robot right before the impact and $(q^+, \dot{q}^+)$ denotes its state right after the impact. We can write the impact map as:

$$\Delta\left(q^-, \dot{q}^-\right) = \left(\Delta_q\left(q^-\right), \Delta_{\dot{q}}\left(q^-, \dot{q}^-\right)\right) \tag{12}$$

Indeed, before and after impact the configuration of the robot (not the velocities) remains the same. So, for Δ_q the idea is to swap the indices of the stance and swing legs appropriately:

$$\begin{pmatrix} q_1^+ \\ q_2^+ \\ q_3^+ \end{pmatrix} = \begin{pmatrix} q_2^- \\ q_1^- \\ q_3^- \end{pmatrix} \tag{13}$$

As for the angular velocities, unlike angles, to calculate their modification, we need some physics. Since the angular momentum is conserved, we have:

$$\begin{pmatrix} H_a^+ \\ H_b^+ \\ H_c^+ \end{pmatrix} = \begin{pmatrix} H_a^- \\ H_b^- \\ H_c^- \end{pmatrix} \tag{14}$$

With H_a the total angular momentum around the swing foot before impact (H_a^-) or around the stance foot after the impact (H_a^+). H_b is the angular momentum of the stance leg around the hip joint before impact (H_b^-) or of the now swing leg around the hip joint after the impact (H_b^+). As for H_c, it is the angular momentum of the torso around the hip joint before impact (H_c^-) or after the impact (H_c^+). We can rewrite the two matrices H^- and H^+ as respectively $A^-\dot{q}^-$ and $A^+\dot{q}^+$. Thus this gives the following equation:

$$\dot{q}^+ = (A^+)^{-1}A^-\dot{q}^- \tag{15}$$

The matrices are presented in the live MATLAB script *generate_impact_map.mlx*.

2.1.4 Kinetic energy loss analysis

As a matter of energy optimisation, it can be interesting to study the kinetic loss of the biped. As we have computed the equation of motion of the impact map, we can use arbitrary values for q^- and $\dot{q}^-$ in order to compare the kinetic and potential energies before and after impact.

First, we can state that, as the potential energy depends only on the position of the angles and that they are the same before and after the impact (swap of indices), the potential energy does not change. It is not the case for the kinetic energy. Assuming initial conditions given by a uniform step $q^- = [\alpha, -\alpha, 0]$ and $\dot{q}^- = [1, 0.2, 0]$, we can plot the percentage of the kinetic energy loss due to impact as a function of the angle α. The result is visible in the figure 4.

For a more intuitive representation, we can plot the percentage of the kinetic energy loss due to impact not as a function of the angle α, but rather the step length. As it is given by, $l = l1 \cdot \sin(\alpha) + l2 \cdot \sin(\alpha)$, the resulting plot is visible in the figure 5.

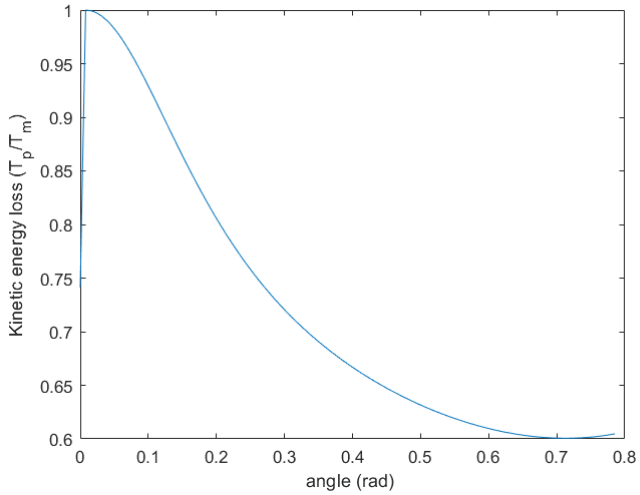


Figure 4: Kinetic loss before and after the impact depending on α

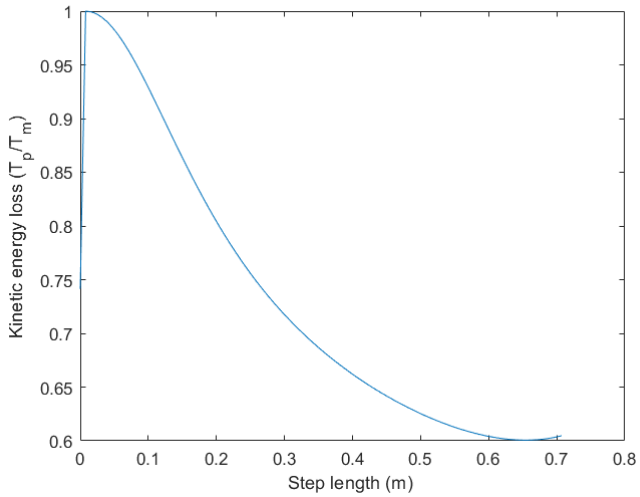


Figure 5: Kinetic loss before and after the impact depending on the step length l

2.2 Solving the equations of motion

2.2.1 Simulation

We would like to solve the equation of motion:

$$M(q)\ddot{q} + C(q,\dot{q})\dot{q} + G(q) = Bu \tag{16}$$

It is possible to solve it with ode45 solver of MATLAB. The signature of the ode45 solver is as follows:

$$[T, Y, TE, YE] = ode45(@eqns, tspan, y0, options) \tag{17}$$

where eqns.m is a MATLAB function with the signature $dy = eqns(t, y)$ representing the state-space form of the equations of motion above. We write the equations of motion $M(q)\ddot{q} + C(q,\dot{q})\dot{q} + G(q) = Bu$ in the form $\dot{y} = eqns(t, y)$, where we have $y = [q, \dot{q}]$. Indeed, we have:

$$\dot{y} = \hat{C}y + \hat{G} + \hat{B}u \tag{18}$$

where,

$$\hat{C} = \begin{pmatrix} 0 & I \\ 0 & -M^{-1}C \end{pmatrix} \quad \hat{G} = \begin{pmatrix} 0 \\ -M^{-1}G \end{pmatrix} \quad \hat{B} = \begin{pmatrix} 0 \\ M^{-1}B \end{pmatrix} \quad \hat{y} = \begin{pmatrix} 0 \\ M^{-1}B \end{pmatrix} \tag{19}$$

2.2.2 Animation

Once we have solved the equation of motion for a several number of steps, we have implemented a MATLAB code that animates the the solution. Basically, as Y is the solution to the ode45 in equation (13), at each time step i we extract the angles q by q = Y(i, 1:3). We then use this q as the input to animate the solution Y of the equations of motion. After each step, the input r0 (memory of x0 and z0), which is the position of the stance foot in the global frame, is updated. There are some parameters that allow us to modify the animation of the simulation:

- real-time factor: it is the actual duration of the simulations to the time it takes for MATLAB to animate the simulations. A real-time factor of 1 mean that the simulation time is equal to the time taken to determine the simulation values. If it's less than 1, it means that the duration of the simulation is smaller than the duration of the calculation of the simulation values.
- skip: for each step, we take all the q values but only examine one every 'skip'. So the higher the 'skip', the less we visualize the intermediaries values that where previously determined. This mean that the duration of the simulation goes down, so the real time factor will be less than one. The speed of the simulation will be greater.

2.3 Walking Controllers

We set ourselves the goal of implementing two different controllers. Each controller should able to generate a family of walking gaits, corresponding to various velocities. The Maximum available torque of each actuator is 30 Nm. Also, the torque signals have to be continuous (i.e., no spikes).

It is important to note that we have only two actuators to control the three degrees of freedom. Hence, the robot has one degree of under-actuation. Some variables or combination of variables will be to fully control while some of them will be regarded as a "free variable".

2.3.1 Proportional-derivative control

In order to get a better understanding of the robot's behavior, we started by designing a simple PD controller. The controller would have to be able to generate an input signals that would allow the control of the three generalized coordinates q_1, q_2 ans q_3. As we had only two actuators, we had the idea to try to keep the q3 on a certain reference, while giving a target to q_2 and letting q_1 behaving freely. This makes sense as the movement of coordinate 3 is nearly independent of the movement coordinates 1 and 2, so it must have its own controller. On the other hand, the coordinates 1 and 2 are symmetrical in their movements and thus one controller for one drives the other by inertia of movement.

The control rule was the following:

$$\begin{aligned} u_1 &= Kd_1 \cdot \dot{q_3} + Kp_1 \cdot (q_3 - q_{30}) \\ u_2 &= Kd_2 \cdot \dot{q_2} + Kp_2 \cdot (q_2 - q_{20}) \end{aligned} \tag{20}$$

where Kd_1, Kd_2, Kp_1 and Kp_2 are the derivative and proportional terms of the controller and q_{30} and q_{20} are the targets of the coordinates q_3 and q_2.

This controller leaves us with 6 hyper-parameters to determine. Intuitively we can find an order of magnitude for each one of them. For instance, q_{30} has to make sure that the torso of the robot stays vertical. It should be more or less around 0. it can be a bit positive or negative depending on whether we want the torso to lean forward or backward. The parameter q_{20} directly relates to the length of the step. At the end of a length q_2 is negative, so q_{20} should be a non-zero negative value. It should neither be too low, otherwise the robot losses too much kinetic energy (as seen in figure 4). For the PD parameters, some tuning is required to find the right ones. However, as the total torques are limited, they have to be lowered as much as possible. We set saturation for the input signal of [-30,30] Nm.

Based on this assumptions, we created different gaits by varying the hyper-parameters:

- Gait 1 (0.6 [m/s]): For the first gait, the main goal was to obtain a simple stabilised controller. Therefore, we took $q_{30} = 0$ and $q_{20} = -\frac{\pi}{12}$. Then we tuned the PD parameters until the system was stable.
- Gait 2 (0.4 [m/s]): To get a slower speed for the second gait, we had the idea to increase a little bit the derivative controller of u_2 to slow down the legs and set $q_{30} = -\frac{\pi}{30}$, so that the torso would lean backward and decelerate the walker.
- Gait 3 (0.9 [m/s]): Finally to increase the speed, we decreased the PD parameters of the torso (q_3) to maximise the PD parameters of the legs (q_2).

2.3.2 Trajectory planing

By looking at the gaits of the PD controller, we noticed that the walking of the robot was very rough. we thought that the balancing of the leg could be smoother by using a trajectory-based approach. Thus, it would be interesting to plan the trajectory for the angle q_2. We had the idea to predict the next position of q_2 by using the actual position of q_1.

We can notice that the difference between the position of the hip and the swing foot is given by the following relation : $\mathrm{l} = l_2 \cdot sin(q_2)$. During the swing of the leg, it is clear that the position of q_1 is related to that difference. So we can plan a trajectory of that difference depending on q_1 and it will then dictate the path to follow for q_2.

This method also enabled more intuitive possibilities for tuning the velocity of the robot. It would be easier to change the length of the step to achieve higher speeds.

The figure 6 represent an example of trajectory planning. We created all our trajectories on the basis of four points. The first and the last points represent the beginning and the end of the step (first and last blues circles on the figure 6. The intermediate points are targets that were set to try to simulate the natural swinging of a step.

Based on those four points, instead of interpolating linearly between each point to generate the trajectory, we created a Bezier curve to have a smother path. For the points $(-target\_q1, -dist)$, $(-target\_q1/2, -1.25 \cdot dist)$, $(target\_q1/2, 1.25 \cdot dist)$ and $(target\_q1, dist)$, the equation $\mathrm{S}(q_1)$ was generated using MATLAB function spline.
The new control rule was then was pretty similar than the previous one except for the target of q_2:

$$\begin{aligned} u_1 &= Kd_1 \cdot \dot{q_3} + Kp_1 \cdot (q_3 - q_{30}) \\ u_2 &= Kd_2 \cdot \dot{q_2} + Kp_2 \cdot (q_2 - S(q_1)) \end{aligned} \tag{21}$$

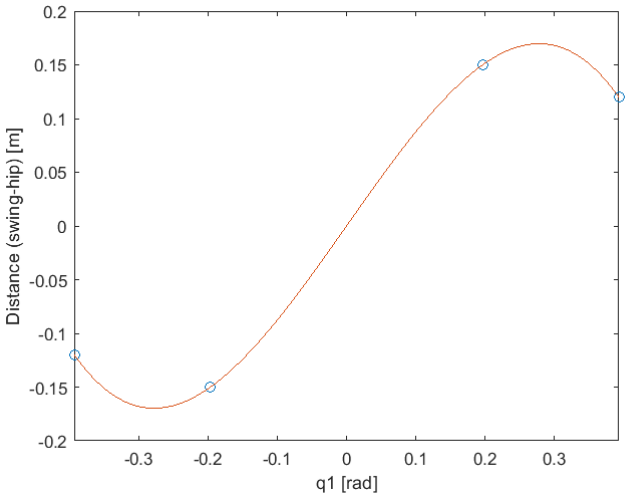


Figure 6: Example of trajectory planing

2.4 Tables resuming the characteristics of our systems

For the perturbations, we chose to add some noise to our final torques and see which positive and negative torques were the limit for the leg controller for each control method and each gait. We didn't look at the torso controller since it would double the value we'd have to determine and also, when there's a perturbation on the torso, the legs help re-stabilize whereas when there's a perturbation applied on the legs, they have to stabilize themselves, which we found was more interesting. We decided to add the noise at step 50 for 150 points (for too few points, it's like the noise doesn't affect the system) so we're sure that the stickman is in a steady-state and visualize a total of 100 steps. We choose this moment so that it will be consistant and not depend on the time, nor the speed of the stickman.
We were surprised by the results since it seems the PD controller is more robust than the trajectory planning controller which shouldn't be the case. By visualizing the stickman walking with some perturbations, we found out that some of them really affect visually the system (we see the stickman bending or moving forward/backwards) whereas some perturbations are not that visible event though they're the limit. The video shows the stickman beginning to walk and entering steady-state and also a few reactions noticeable to the perturbations.

Gait	1	2	3
Target q3 [-]	0	$-\pi/30$	0
Target q2 [-]	$-\pi/12$	$-\pi/12$	$-\pi/18$
Kp1 [-]	100	100	80
Kp2 [-]	-50	-50	-145
Kd1 [-]	20	20	10
Kd2 [-]	-5	-7	-10
Speed [m/s]	0.7	0.4	0.8
Max perturbation leg controller [Nm]	65	70	34
Min perturbation leg controller [Nm]	-10	-35	-18
CoT [-]	0.09	0.04	0.15

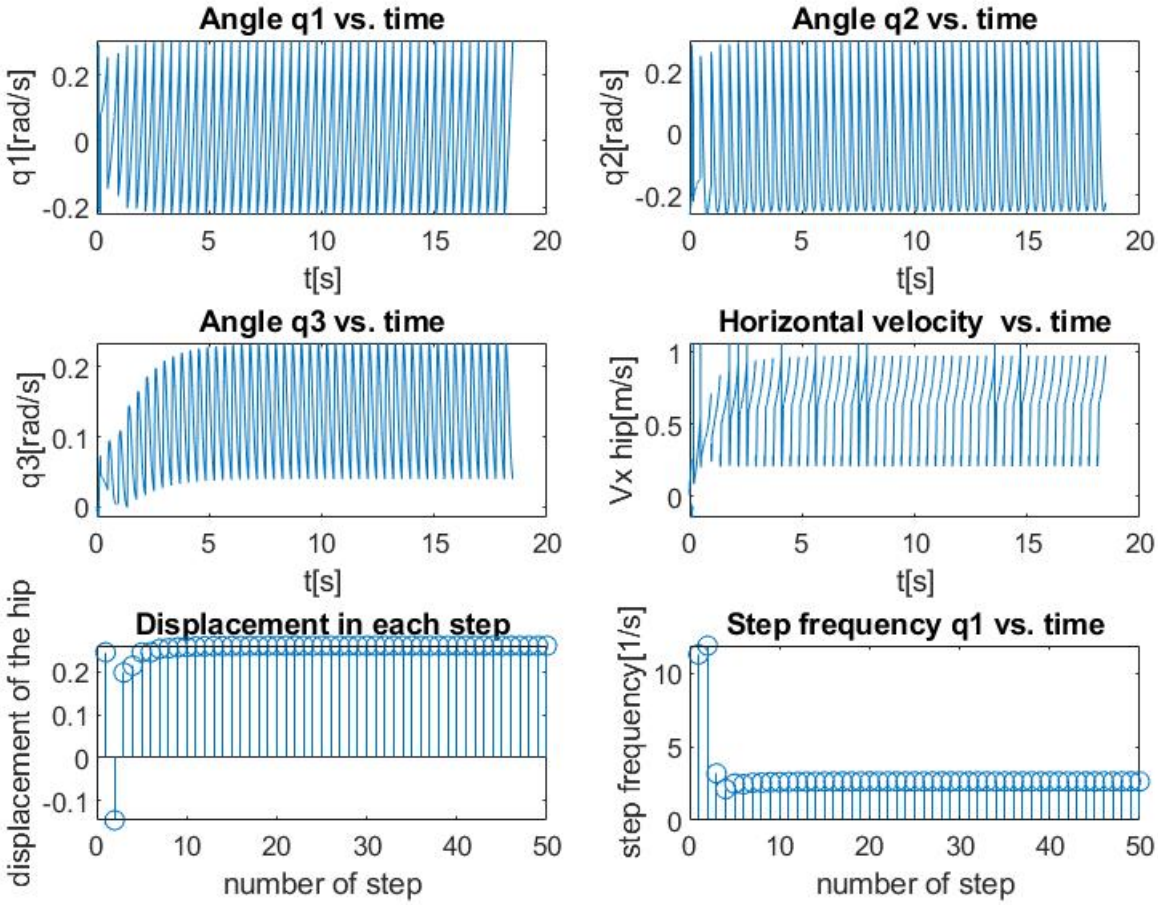
Table 1: Characteristics of the PD controller

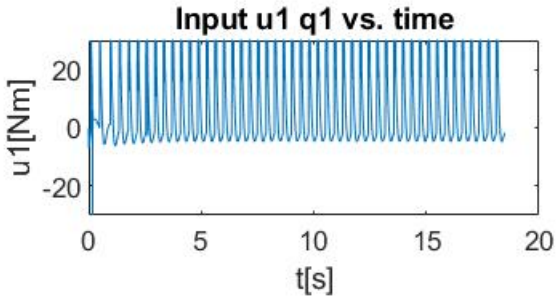
Gait	1	2	3
Target q1 [-]	$\pi/11$	$\pi/11$	$\pi/14$
Target q3 [-]	0	$-\pi/35$	$\pi/40$
Dist parameter [m]	0.12	0.16	0.08
Kp1 [-]	100	100	80
Kp2 [-]	-50	-50	-180
Kd1 [-]	20	20	10
Kd2 [-]	-5	-3	-12
Speed [m/s]	0.6	0.3	0.8
Max perturbation leg controller [Nm]	111	40	38
Min perturbation leg controller [Nm]	-16	-3	-50
CoT [-]	0.07	0.09	0.13

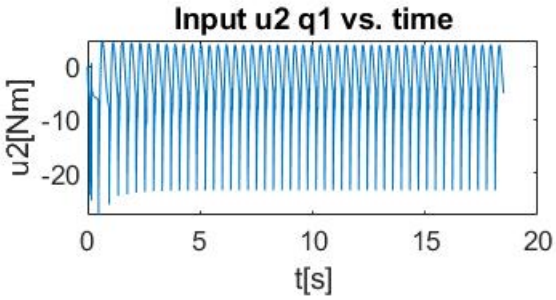
Table 2: Characteristics of the Trajectory planning controller

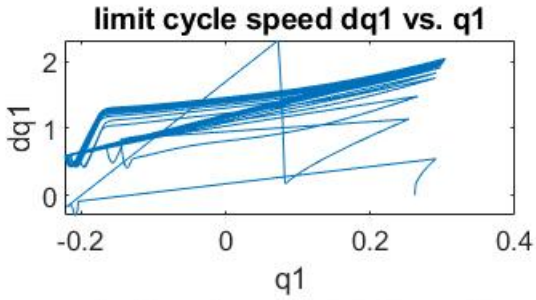
3 Results

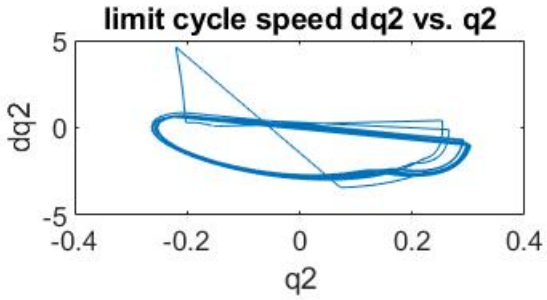
Here is the analysis of the 1st gait of the PD controller. The simulation have been done for 50 steps with initial conditions: $q_0 = [\pi/12, -\pi/12, 0]$ and $\dot{q_0} = [0, 0, 0]$. The first three graph show the angles of the robot and its horizontal velocity with respect to time. The fourth and fifth shows the displacement of each step and the step frequency. The sixth and the seventh are the power consumption of each actuator over time. The last ones are the limit cycles of the three angles.

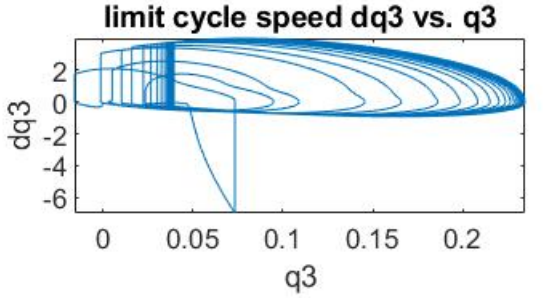



Input u1 q1 vs. time
u1[Nm]
20
0
-20
0
5
10
15
20
t[s]

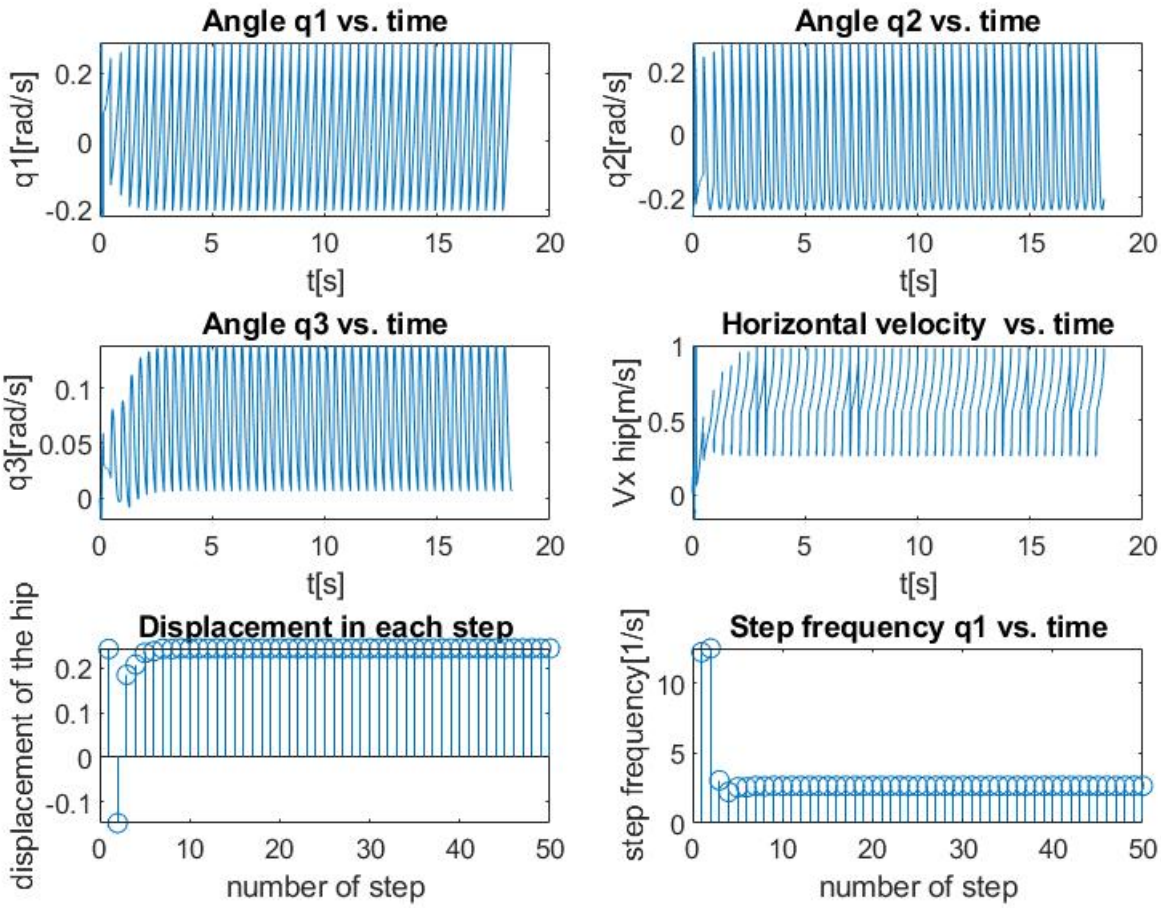

Input u2 q1 vs. time
u2[Nm]
0
-10
-20
0
5
10
15
20
t[s]

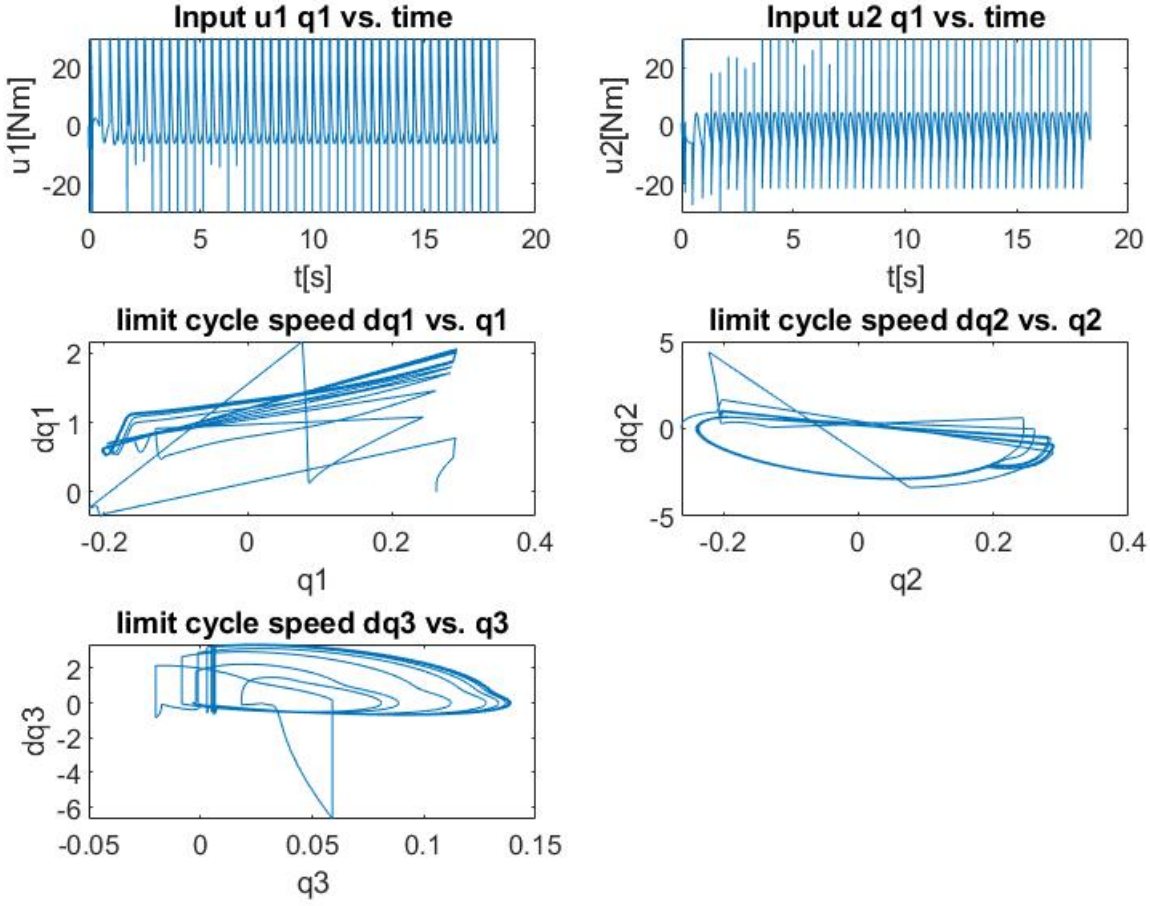

limit cycle speed dq1 vs. q1
dq1
2
1
0
-0.2
0
0.2
0.4
q1


limit cycle speed dq2 vs. q2
dq2
5
0
-5
-0.4
-0.2
0
0.2
0.4
q2


limit cycle speed dq3 vs. q3
dq3
2
0
-2
-4
-6
0
0.05
0.1
0.15
0.2
q3

And here is the analysis of the 1st gait of the Trajectory planning controller. The simulation have been done for 50 steps with the same initial conditions: $q_0 = [\pi/12, -\pi/12, 0]$ and $\dot{q_0} = [0, 0, 0]$.




Input u1 q1 vs. time
u1[Nm]
t[s]
Input u2 q1 vs. time
u2[Nm]
t[s]
limit cycle speed dq1 vs. q1
dq1
q1
limit cycle speed dq2 vs. q2
dq2
q2
limit cycle speed dq3 vs. q3
dq3
q3

4 Discussion

We chose to analyse the first two gaits of our controllers as they presents roughly the same characteristics (same speed, same target for the third angle and same gains).

4.1 Robustness

Looking at the tables, we can see that the second controller (based on trajectory planning) can withstand higher perturbations than the first one (111 N/m, -50 N/m) for some gaits but not all. We were surprised since we thought that the trajectory planning controller would be more robust than the PD for all gaits. Maybe the way we implemented the noise in the system was not that optimal.

4.2 Speed

The maximum speed and minimum speed we obtained with each controller were quite similar. If we look at the graph of the first gait of the first controller (which has a speed of 0.7 m/s) and the first gait of the second controller (which has a speed of 0.6 m/s), we can see that the horizontal velocity stabilizes for both around 3 seconds approximately. Also, this velocity varies over time between 0.4 and 0.8 for the first case and between 0.3 and 0.8 for the second case. So they are quite similar. The displacement between each step is approximately 0.3 in both cases. This shows that even though the two methods are different, for a same speed the system acts almost perfectly in the same way.
For both controller, we designed three different gaits so we could have three speeds and three costs of transports.

4.3 Stability

By looking at the limit cycles of every generalized coordinates, we can see that, they all converge to a specific closed trajectory. If we observe the plots of the angles, we see that the Trajectory planing controller stabilizes more rapidly the angle q_3 then the PD controller. This is a surprising result because, intuitively as we control q_3 similarly in both controller, we should have the same response. It can probably be explained by the fact that in the trajectory planing, the movements of the legs of the robot are smoother. Therefore, the angle q_3 is less affected.

4.4 Power consumption

A proper way to evaluate the power consumption of a robot is to look at his cost of transport (CoT). We have calculated this value for every gaits of the two controller. They are visible in the Table 2 and 1. It is important to state that the formula used for the calculation is the same as the one provided in the LR Assignment 4. However, we have divided that formula by the weight of the robot ($W = m \cdot g = (m_1+m_2+m_3)\cdot g = 304$ N), in order to obtain a dimensionless formula. We can see that, as a general manner, the PD controller exert a higher consumption than the Trajectory planning. Having a target that adapts itself along the trajectory probably reduces a little bit the overall consumption, as it avoids big overshoots under/above the angle reference.

4.5 Improvements

With more time, we surely could've designed quicker gaits or slower gaits, and find the real maximum speed and minimum speed for each controller. But designing each gait is very time-consuming, particularly finding accurate Kp1, Kp2, Kd1, Kd2 as well as applying correctly the design method. For example, for the trajectory planning, we had to find the trajectory and determine a function that is linked to this trajectory, which was quite time consuming.
For the trajectory planning, we're not sure if our trajectory is good enough to describe the evolution of the system. Maybe if we had more time and more data to analyze, we could have had a better model. Especially, it would have been interesting to have access to real data on the natural trajectories of humans when walking. It would have allow us to design a more accurate trajectory.

It would have been also interesting to implement another controller in order to compare it the two that we have designed.

5 Conclusion

This project was really interesting and taught us a lot about control, simulations and how to make a stick-man walk. It made us understand how important simulations are, in order to implement correctly the control and see how the robot will be affected by outer noise. Also, as we had previously worked on the robot Ranger from Cornell university presented in *Low-bandwidth reflex-based control for lower power walking: 65 km on a single battery charge*, working on the control of stick-man was very interesting. Indeed, these two robots are similar in the way that they have only a few degrees of freedom, nearly the same gaits and very limited actuation to reduce overall power consumption. It means that in some aspects, they can be compared. For instance, we were very surprised when we calculated the CoT of stick-man compared it to the one of ranger. They both had a CoT of the same order of magnitude: 0.04-0.015 for stick-man and 0.19 for ranger. This made us realize the incredible work that the University of Cornell had done, as their robot is much more complex than ours and still consumes less because 80% of their consumption is due to processors, communications and motor dissipation, which is not taken into account in the calculation of our CoT.