EPFL, Lausanne
Legged Robots, Fall 2018

Design of a walking controller for a three-link biped

Teacher: Prof. Auke Ijspeert
Group 14
Victor Delafontaine, Nathanael Ferraroli and Valentin Kindschi

December 19, 2018

Abstract

The goal of the project is to implement a controller for a three-link bipedal robot composed of two legs and a torso. The robot should be ableto walk on a 2D plane with flat ground, thanks to its two actuators. We decided to implement the controller using the virtual constraints method, using one constraint to control the inclination of the torso and another one to control the angle between the two legs.

The result is a robot able to walk at speeds between 0.4 and 1.2 m/s. The robot start at an initial position where all the angles and angular velocities are null. It can sustain a perturbation of 50 N at all speeds, and 450 N at 1.2 m/s. The required torque was contained under the required 30 Nm for the actuator controlling the legs however, some spikes and higher values could not be avoided for the controller of the torso.

Contents

1 Introduction

Our three-link biped, who we will call Bob from now on, has a total of three degrees of freedom (DOF) defined as q_i (i=1,2,3). They are defined as the angular position of each legs as well as the torso rotating at the hip, 0 being a vertical position. All three limbs have a discrete mass positioned at their middle. It has two degrees of actuation u_1 and u_2. We can see a representation of Bob in Figures 1 and 2 below:

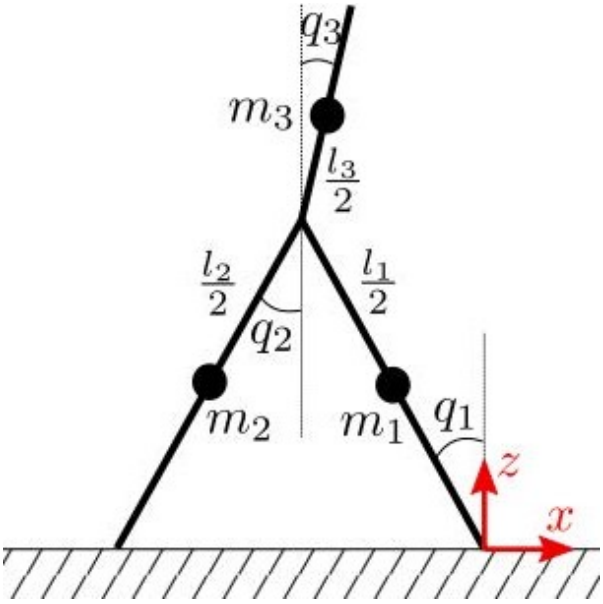

Figure 1: Bob's geometry

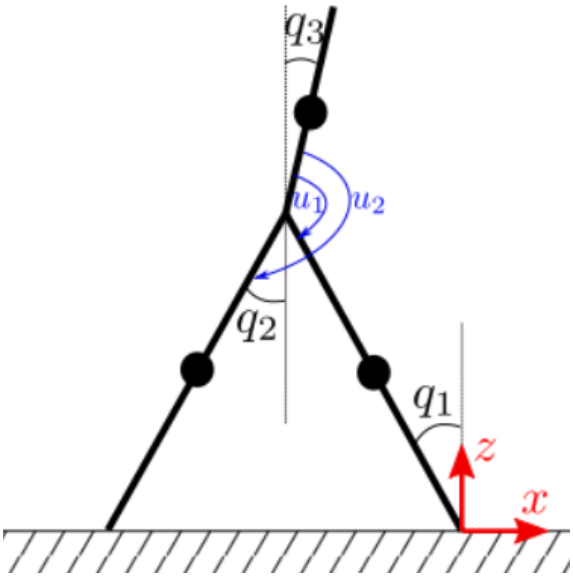

Figure 2: Bob's actuation

In the first assignment, we created the functions behind Bob's dynamic and kinematics. In the second one we developed the one to simulate steps and display them in an animation. Missing to this is the controller: an empty vector of size two was used until now.
The goal of this final assignment is to design Bob's walking controller.

2 Control design

We decided to control Bob using the virtual constraints method. This choice was done as the method is both intuitive and complete. Moreover, a step-by-step method was defined precisely in the lectures, which reduced the error we could do while implementing it.
In contrast, Zero-Moment Point (ZMP) based method were discarded as Bob doesn't have any feet.

We could have used trajectory based method, but we decided to use the virtual constraint method as it seemed more straightforward for us.

Our controller has a degree of two, while we have three angles to actuate. Hence the controller will have one degree of underactuation.
We decided to design our constraints y as follows, $q1$ being our free variable:

$$y = \begin{bmatrix} q_3 - c_1 \\ q_2 - (q_1 - f(q_1, q_2, c_2)) \end{bmatrix} \tag{1}$$

The constants c_1 and c_2 govern the goal position. The first one will decide how much Bob leans forward or back while the second will influence the angle between the two legs. The function $f(q_1, q_2, c_2)$ was first chosen as a constant, independent of q_1 and q_2. While this solution worked, we decided to match the leg movement with a sinusoidal depending on both legs position, as we estimated that it would be more efficient. Hence this function was defined as:

$$f(q_1, q_2, c_2) = \frac{c_2}{2} \cdot \left(1 + cos(\frac{q_2 - q_1}{c_2} \cdot \pi)\right) \tag{2}$$

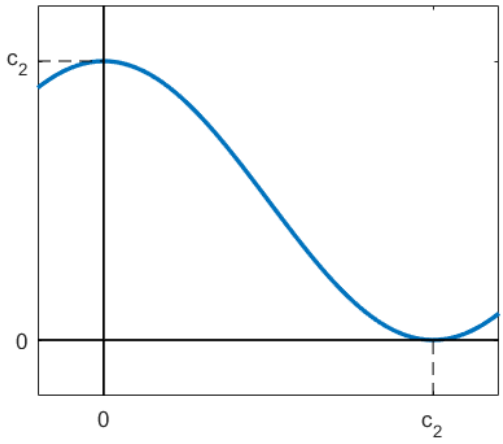

Figure 3: $f(q_1, q_2, c_2)$

When the difference between q_2 and q_1 is near 0, this function is equal to the maximum angle between the legs, defined by c_2, whereas when the the angle between the legs is c_2, the function is equal to 0. This will ensure that the legs are continually getting closer or apart from each other and the two constants c_1 and c_2 can be tuned depending on the required speed.

We decided to use the linearized version of the virtual constraints method. As a result, our controller was obtained as follows:

$$u = \begin{bmatrix} k_{p,1}y_1 + k_{d,1}\dot{y_1} \\ -k_{p,2}y_2 - k_{d,2}\dot{y_2} \end{bmatrix} \tag{3}$$

The second component of u is negative due to the axis representation shown in Figure 1.
The main difficulty to define this controller will be the fine tuning of parameters k_p and k_d for $u_{1,2}$, as well as for the constants found in equation 1.

2.1 Parameter tuning

Bob's walking controller goes through two phases. The first phase corresponds to a number of step inferior than *nb_step_begins* and is used to start a stable gait, always using the same parameters regardless of the targeted walking speed. After *nb_step_begins*, the controller switches to parameters that are chosen to reach the desired speed.

Tuning the parameters proved to be mainly a trial and error process. Since this could be highly time consuming, we decided to use a 3-step strategy to find efficient parameters :

1. Design of a robust controller that creates stable gait, without any targeted speed or maximum control torques.

2. Starting from the parameters found in step 1, decrease the values of the control parameters (k_p and k_d for $u_{1,2}$) to decrease the generated torques while adapting the constants to keep a stable gait and reach the highest desired speed ($1.2m/s$).

3. Decrease the walking speed by first trying to modify the constants, only changing control parameters if this wasn't enough to reach the goal.

Trying to reach the relatively high speed of $1.2m/s$ in step 2 showed that some vague "ground rules" could be observed :

1. Faster walking speed means the torso needs to lean further forward to keep balance.

2. Step length (and consequently the constant controlling the angle between legs) has to be as high as possible, while staying low enough for the swing foot to overshoot as little as possible over the ground.

3. k_p and k_d for u_1 need to be high enough for the torso to be stable on desired position without oscillating (which would unbalance the system, lowering the walking speed or even making the robot unable to perform a stable gait).

4. k_p for u_2 needs to be high to make the swing phase fast, but too fast means the inertia of the leg makes Bob fall backward.

5. k_d for u_2 has to be high enough to prevent oscillation (which would once again unbalance the robot) but a high value prevents the leg from reaching desired position in time to keep a stable gait.

While prioritizing rules 1 and 2, all these principles were widely used to reach all the targeted speeds of the second set (0.4, 0.6, 0.8, 1.0, 1.2 m/s) within the required 10% of error. In Table 1 below, P_{max} is the maximum perturbation the robot can sustain in each gait and one can find the resulting parameters of the controller:

Speed [m/s]	$k_{p,1}$	$k_{d,1}$	$k_{p,2}$	$k_{d,2}$	c_1	c_2	$P_{max}[N]$	CoT
0.4	100	50	7	1.3	0	$\pi/4.5$	35	0.17
0.6	80	40	8	1.8	$\pi/36$	$\pi/3$	100	0.23
0.8	80	40	12	1.2	$\pi/10$	$\pi/3$	100	0.44
1.0	80	40	30	1.2	$\pi/5$	$\pi/4.5$	450	0.45
1.2	100	50	30	1.2	$\pi/3$	$\pi/4.5$	480	0.42

Table 1: Control parameters and performances for achieved speeds

3 Results

We created different plotting functions to analyze Bob's behavior. These are defined as follows:

- angles versus time: *plotQ*
- robot velocity versus time: *plotSpeed*
- displacement in each step versus step number: *plotStepLength*
- step frequency versus step number: *plotStepFrequency*
- torques versus time: *plot_u*
- angles versus angular velocities: *plotQvsDQ*
- cost of transport versus step number: *plotCot*

We also added two additional functions we deemed useful:

- *plotStepVect*: to visualize the time spent in each step
- *plotHipPos*: to see the shape of hip movement

- *printResults*: prints the CoT, target and obtained speed and maximum torque produced

Overall these functions enabled us to have a more analytic view on Bob's movement and behavior.

3.1 Range of motion

Our robot is capable of walking at speeds between 0.4 and 1.2 m/s in increments of 0.2m/s (0.4; 0.6; 0.8; 1.0; 1.2 m/s).
The chosen speed can be changed inside the *simulation.m* file. The global variable *desired_speed* is then retrieved when necessary during the parameter choice.

The obtained speeds as well as the error relative to the targeted speed can be found in Table 2 below. The function *printResults* was used to obtain these values.

Desired speed [m/s]	0.4	0.6	0.8	1.0	1.2
Obtained speed [m/s]	0.42	0.60	0.80	1.06	1.19
Error [%]	4.02	0.36	0.27	5.91	0.59

Table 2: Desired and obtained speeds

All obtained values are smaller than 10% as per requirements.

3.2 Robot speed and position, angle q_i

The robot hip velocities can be seen in Figure 4. The hip vertical velocity is plotted in red, and oscillate around 0 (up and down movements cancel themselves). The blue curve represents the horizontal velocity, that also oscillates in sync with each step but always keeping a positive value.
The yellow line is the average horizontal speed of the hip, which we want to reach the targeted values between $0.4m/s$ and $1.2m/s$. Its final value can be found in the legend.

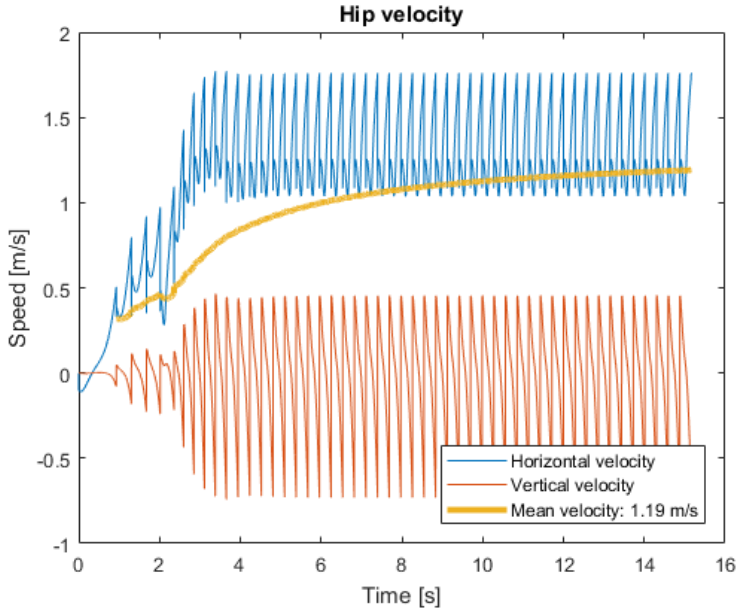

Figure 4: Hip horizontal and vertical velocities for 1.2 m/s

In addition to that, we can plot the hip position. The result can be seen in Figure 5. This makes it possible to check its vertical movement to try to match it to a human-like oscillating behaviour during gait.

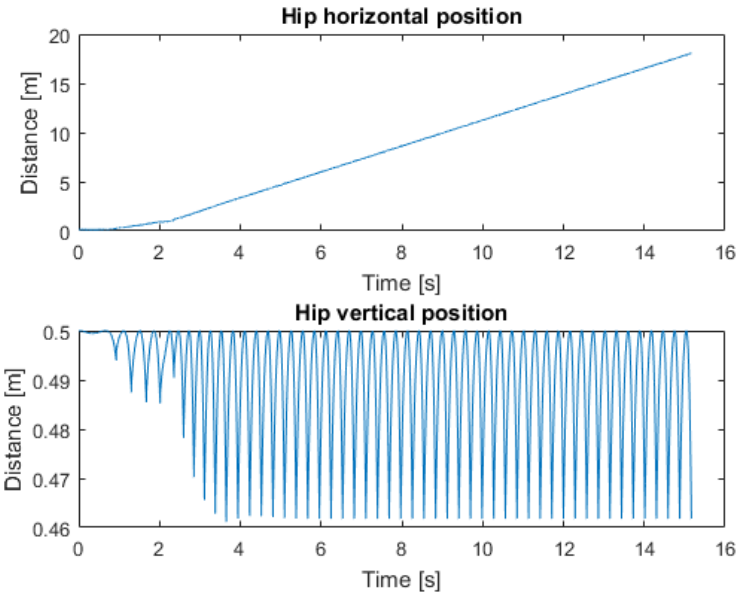

Figure 5: Hip horizontal and vertical position for 1.2 m/s

In the graph of Figure 5, we can see that the horizontal position is steadily growing

while the hip goes up and down at each step. This motion is expected as the hip follows an inverted pendulum model.

In Figure 6 below we can see a plot of the three angles defined as in Figure 1.

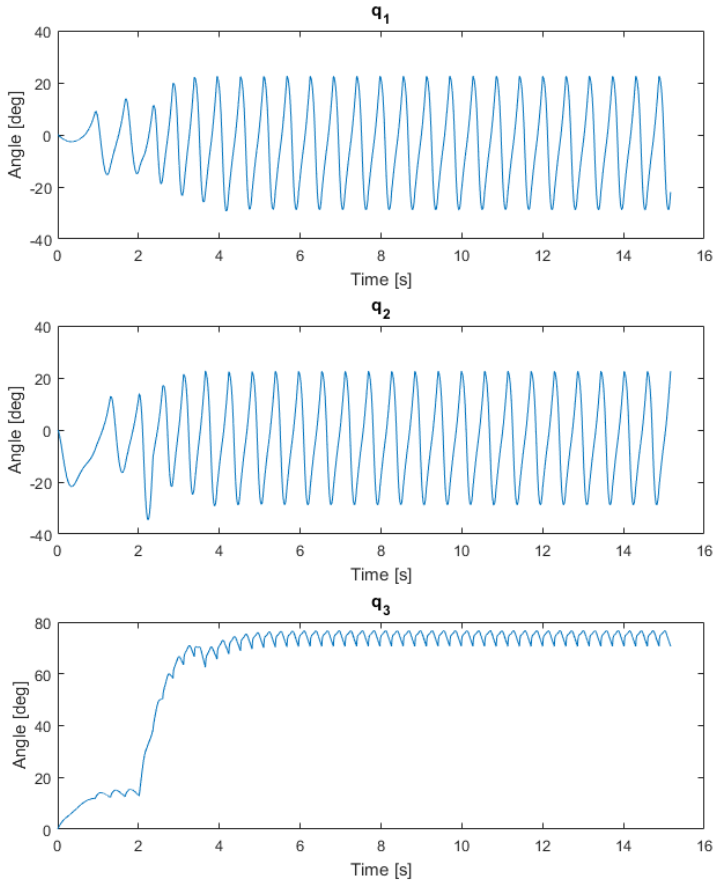

Figure 6: Actuation angle q_i (i=1,2,3) for 1.2 m/s

We can see that the angle each follow a periodic pattern in time. At an high speed (figure 6, 1.2 m/s), the legs have an high range of motion: between 25° and -25° approximately. We can also see that the torso is leaned further forward, at 75°. This is because Bob's center of mass needs to be forward to bring the swing foot into contact after its swing. Without this forward centered robot, the foot would take too much time to come into contact and the robot wouldn't be able to reach fast speeds.
We can also see the effect of the variable *nb_step_begins*. It was set at 5 during this simulations. On the graph of Figure 6 showing q_3, we can easily see that the robot first moves at lower forward angle, then leans forward to reach the desired speed.

By plotting the same graph for lower speeds, we could see that Bob's legs have a smaller range of motion, usually between 15° and by -15° approximately. The torso is angled forward with an small angle, for example approximately 8°for 0.8 m/s.

3.3 Step characteristics

The first function of interest on the subject of step is *plotStepVect*. An example can be seen in Figure 7. This function allows us to see the time the robot spends in each step. We can see that the first step is relatively long and then the steps are all approximately of the same length. This is easily explained by the fact that the robot starts at zero velocities but can, after the first step, use inertia from last step to perform the next.

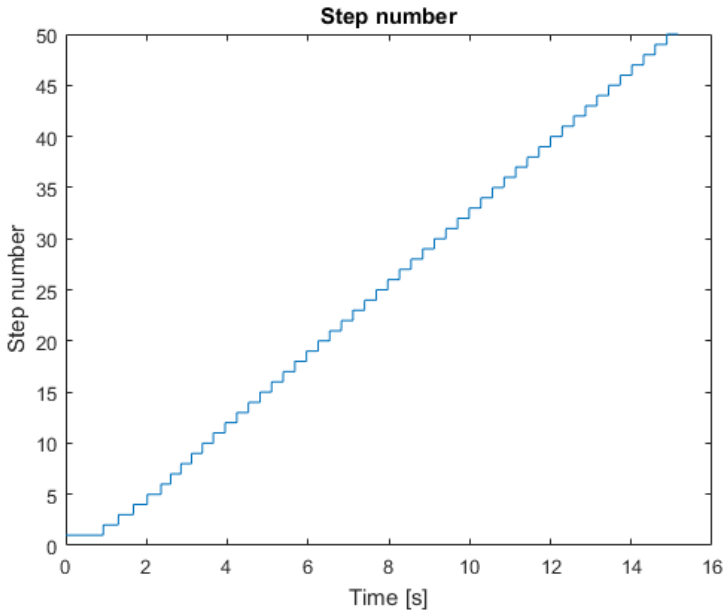

Figure 7: Step length in time

Building on this function, we can plot the step frequency and length based on step number. The result of these two functions (*plotStepLength* and *plotStepFrequency*) can be seen in Figures 8 and 9 respectively.

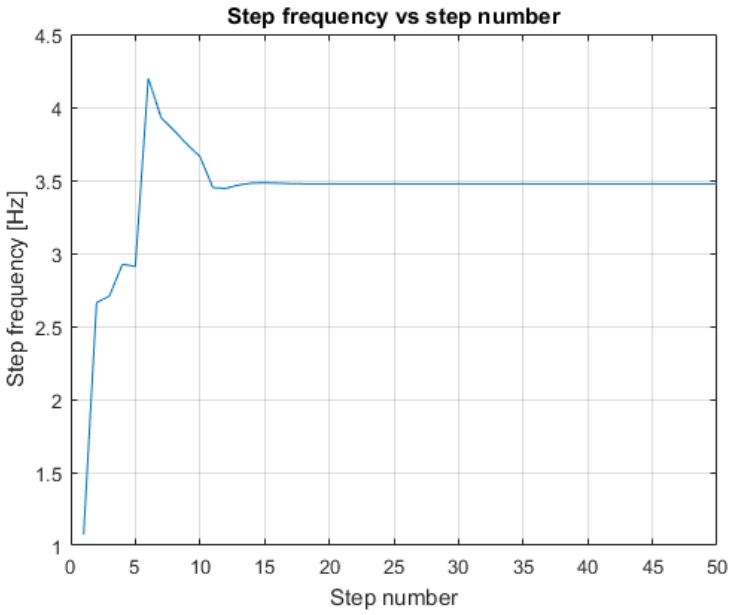

Figure 8: Step frequency vs step number for 1.2 m/s

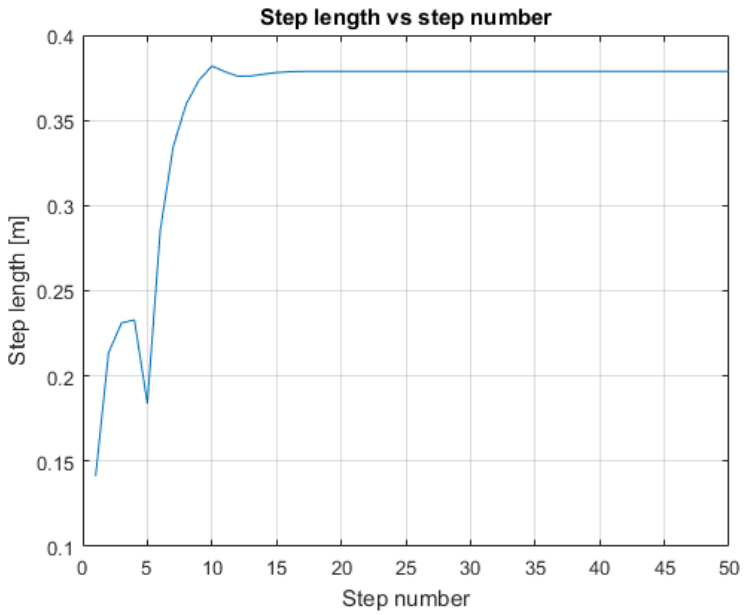

Figure 9: Displacement in each step vs step number for 1.2 m/s

The first thing we can observe is that the robot reaches an equilibrium after between 10 and 15 steps depending on the desired speed. We also notice the effect of our starting gait for the first 5 steps.

A thing we could observe by plotting these graphs for multiple speed is that the faster the robot is, the longer the step length and the higher its frequency have to be. This makes sense since to run faster a biped should increase its footstep length and/or its step frequency.

3.4 Actuation torque

The simulator could produce infinite torques, but this wouldn't be possible to reproduce in real world situation. We were fixed a maximum torque of 30 Nm to simulate an actuator maximum power. In Figure 10, we can find the torque plot for the fastest and speed of our motion range, 1.2 m/s.

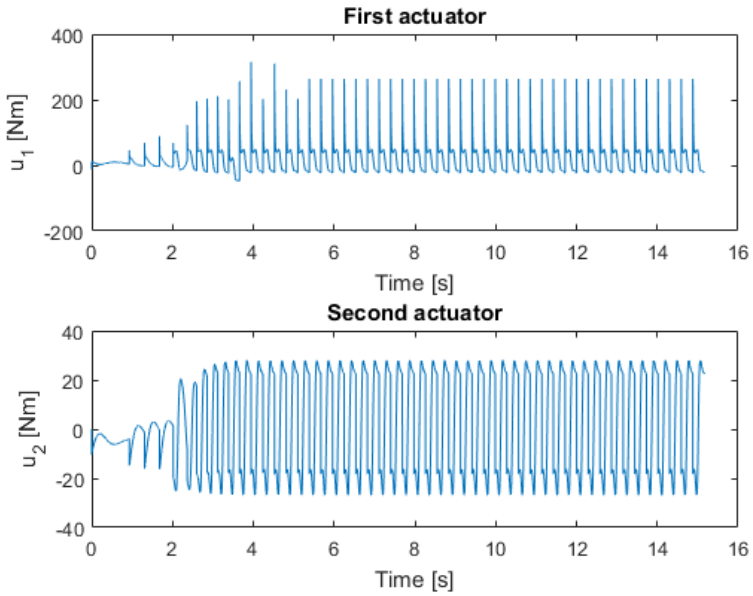

Figure 10: Torque produced by each actuators for 1.2 m/s

We can see that the torque sometimes go over 30 Nm. However, the spikes produced are very short (less than 10 ms). We concluded that this came from the PD controller reacting on a step. As the two foot change label (q_1 is always for the stance foot, q_2 always for the swing foot), the derivative of the angle becomes very high. If we neglect the peaks, the value is still around 50 Nm and with our simple constraint y_1, we could not achieve better results.

3.5 Cost of Transport

The Cost of Transport (CoT) is a good indicator of the effectiveness of the gait. It can be computed using formula 4.

$$CoT = \frac{E}{m \cdot g \cdot d} \tag{4}$$

In this equation, E is the total energy consumed by the system to travel distance d, m is the total mass of the system and g the standard gravity . Since the simulation only considers conservative forces in the calculation (e.g. no friction forces), the overall mechanical energy of the system remains constant without actuation. Thus, any increase or decrease in the mechanical energy is due to actuation. The consumed energy can be computed by finding out if the difference in potential energy differs from the difference in kinetic energy for each simulation step. In case it differs, we can add the absolute value of the difference to the total energy consumed. For each step n we can compute the total energy using equations 5 to 7.

$$E_{pot}(n) = g \cdot (m_1 \cdot z_1(n) + m_2 \cdot z_2(n) + m_3 \cdot z_3(n)) \tag{5}$$

$$E_{kin}(n) = \frac{1}{2}(I_1 \dot{q_1}(n)^2 + I_2 \dot{q_2}(n)^2 + I_3 \dot{q_3}(n)^2) \tag{6}$$

$$E_{tot}(n) = E_{tot}(n-1) + \mid (E_{kin}(n) - E_{kin}(n-1)) - (E_{pot}(n) - E_{pot}(n-1)) \mid \tag{7}$$

We obtain the CoT shown in Table 3 for each targeted speeds:

Desired speed	0.4	0.6	0.8	1.0	1.2
Cost of transport	0.17	0.23	0.44	0.45	0.42

Table 3: Cost of transport for set speeds

As expected, the cost of transport is lower for lower speeds. It makes sense since a smaller speed needs a smaller footstep and a lower step frequency, as shown in 3.3, which requires less energy. Note that we could expect a higher cost for 1.2 m/s, but the hyper-parameters of our controller might be more optimized for this speed than for lower ones, which could explain the slightly smaller cost of transport. In order to decrease the CoT for faster speeds, another type of gait such as running could be used.

As we can see on Figure 11, representing the CoT for each steps of the robot, the cost starts at zero and then varies a lot before reaching a stable curve that converges to the value in Table 3. The variation at the beginning are due to the lack of a stable gait at this point, each step can thus have very different length and energy cost.

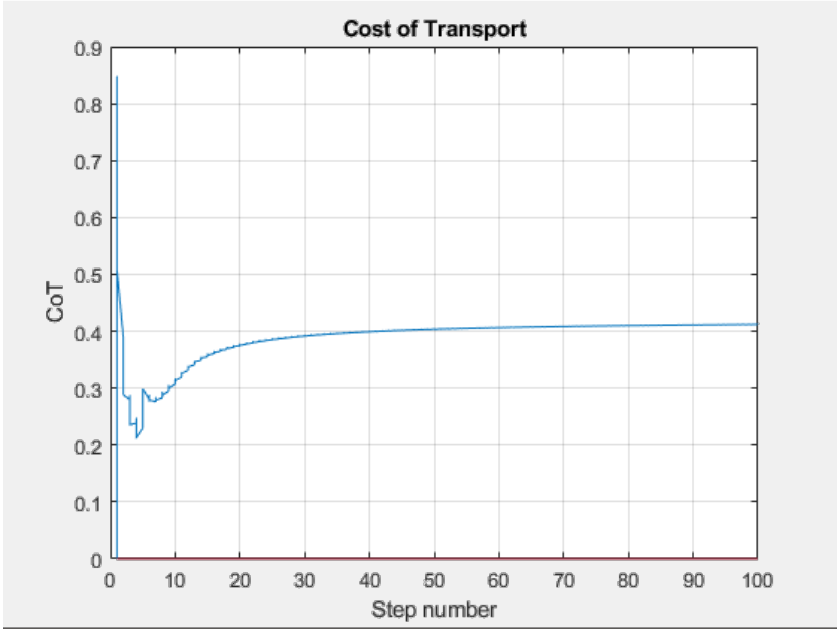

Figure 11: Cost of Transport at each step for 1.2 m/s

3.6 Relation of angle versus angle rate

We can plot each angle against its relative rate. For a stable gait, each angle should reach a limit cycle behaviour. Figure 12 shows the obtained phase plot.

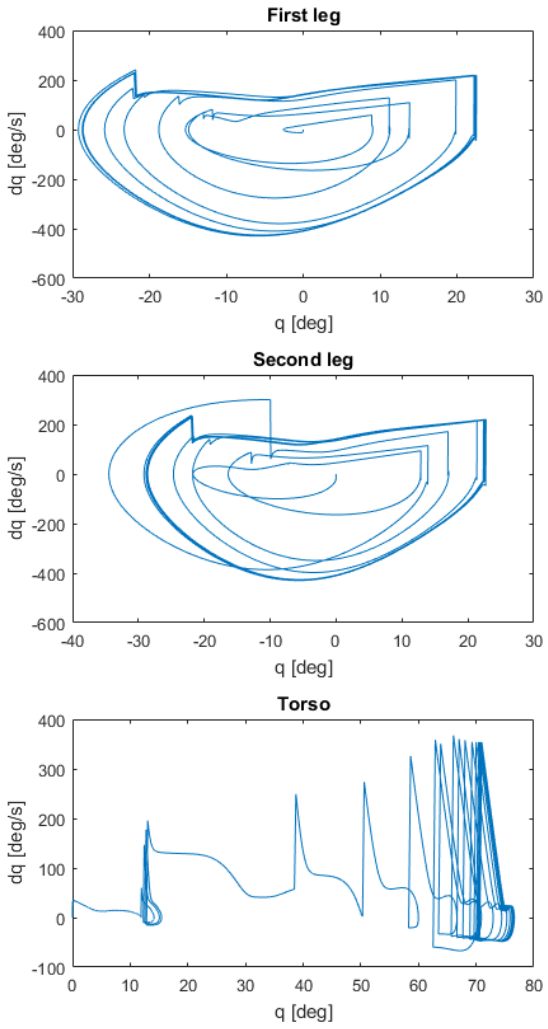

Figure 12: Angle versus angle rate for 1.2 m/s

We can see the limit cycle in Figure 12. The most obvious of the three angle to observe this is q_3. In the corresponding graph, the curve begins on the left of the graph, then the limit cycle shows its form and finally centers itself around 70°.

3.7 Resistance to perturbations

We can introduce a perturbation in our system by tuning two variables to change the perturbation force amplitude and the step label at which it is applied.
We chose to disturb the robot with a positive and negative force. The perturbation was

done on step 20 of a 40-step simulation with an intensity of ±480N.

The main difference is seen in the graph of step length as seen in Figure 13 below:

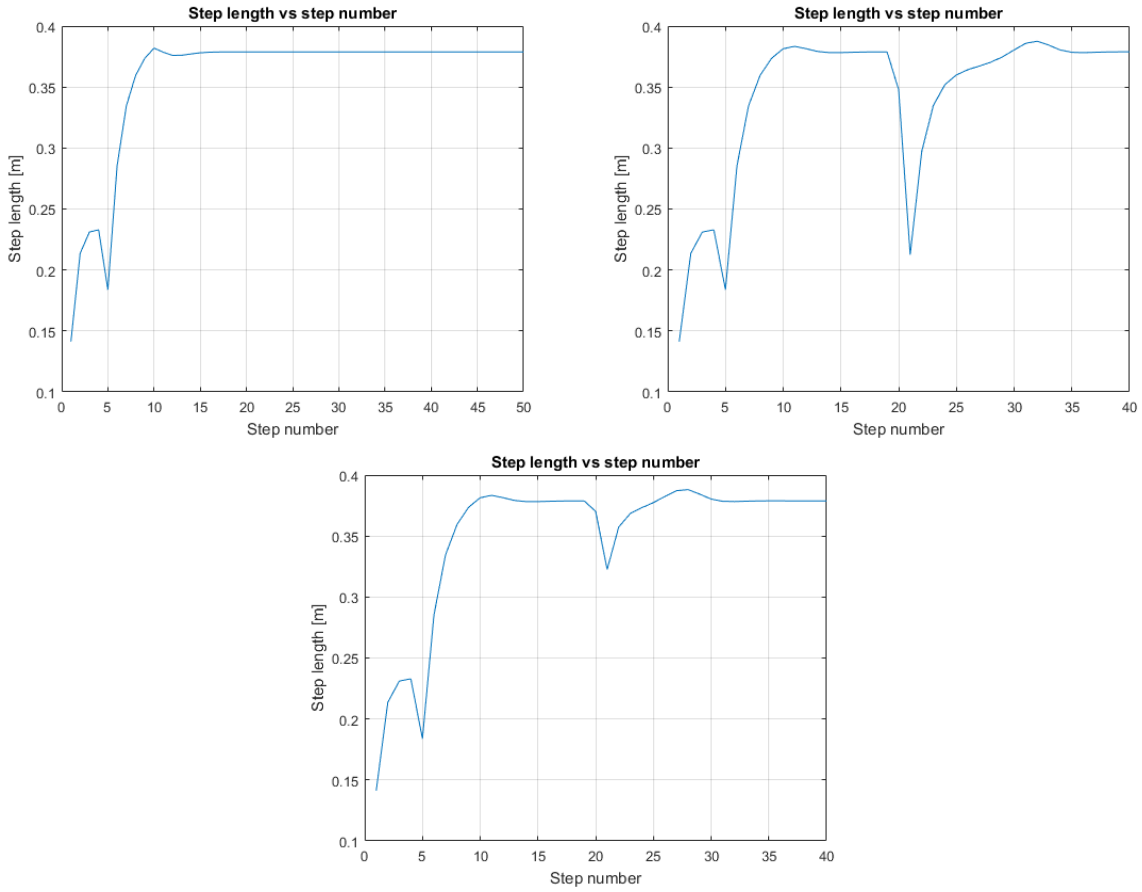

Figure 13: No perturbation, perturbation forward and backward

The threshold to make the robot fall is very high as he his angle forward in any case to reach the required high speed. A push forward will result in his torso to straighten momentarily before going back into running gait. The limit is high as it requires a lot of force to make the torso go behind the hip vertical point. It takes approximately 5 steps to get back into the previous gait.
Pushing backward will bend Bob forward, just as someone who gets punched in the gut. The limit is defined by when the torso goes low enough so that the controller can't bring it back up in time.

For our slowest speed of 0.4 m/s, the range is [-40; 35]N. As the robot torso is close to vertical (PD control tries to make q_3 go to 0) to sustain the low speed, pushing it too much forward will cause its torso to go behind his hip, which causes the fall. Pushing it backward makes Bob lean too much forward which also precipitates his fall.

Intermediate speeds can withstand perturbations of variable amplitude, always above $\pm 50N$.

4 Discussion

Even if Bob is capable of walking, the designed controller is not perfect. Some improvements could be made to improve it.

For example, we could use the nonlinear version of the virtual constraints method. This could make the controller more robust and versatile.

Another improvement could be changing the constant values found in the output equation 1. We could for example make the torso follow a sinusoidal instead of trying to reach a fixed value. By matching its frequency with the step frequency, this could make it so that the robots leans forward when moving a foot forward. This is a comportment found in human gait, and is found to be more efficient and could also decrease the torque peaks of $u1$.

The parameters could also be tuned more optimally by running multiple loops on all parameters. However, this would be very time consuming for this project.

It could also be interesting to implement reflexes in the controller. Bob could for example, lean its torso forward or back if the robot's center of mass horizontal position goes too far away from the contact foot.

All these methods would allow to create a more stable and robust to perturbation gait, allowing Bob to move in a wider variety of terrain and conditions.

5 Conclusion

Building a simulation of a simple system turned out to be an interesting task which required many engineering skills from programming to physics. Starting almost from scratch and building the whole program seemed to be complicated at first but was in the end mostly a succession of relatively simple and well defined tasks.

The controller on the other hand was simple to implement in the first place but it was then complicated to find stable and robust gaits for different speeds, which is why it had to be modified along the way. Tuning the parameters was quite frustrating in the beginning, feeling like a completely random task. After a few trials, it was

still possible to highlight a few principle that made the process closer to a logical method.

Overall, this project allowed us to acquire a wider view of what the work of an engineer can be when working on large projects by making us work on a simple model from the creation of the simulation to the design of a controller and the tuning of its parameters.