Institute of Microengineering

Legged Robots MICRO-507

Ao.Univ.Prof. Dipl.-Ing. Dr.techn. Ijspeert Auke

Mini Bipedal Robot Project

Authors:
D. Bignet
S. Ben Ghorbel
C. Stocker

Scipper :
322826
261316
266575

Group 19

December 10, 2024

Contents

1 Part I: Theory **4**

2 Triple Critically Damped Virtual Oscillators Controller **18**

2.1 Starting point: Virtual Model Control: 18

2.1.1 Task space Kinematics 19

2.1.2 Task space Jacobian 19

2.1.3 Input transform: 20

2.2 Designing the controller 20

2.2.1 System separation 20

2.2.2 Shape of the applied forces: Triple harmonic oscillators 21

2.2.3 Final Virtual Forces 22

2.2.4 Definition of the working domain of the stance leg harmonic oscillator 23

2.2.5 Synchronization of the swing harmonic oscillator 23

2.2.6 Lifting of the swing foot when starting a step 24

2.3 Choosing the right gains 24

2.3.1 Critical damping 24

2.3.2 Global parameter optimization 25

2.3.3 Cost function 26

2.3.4 Maximum number of steps 27

2.3.5 Minimum and maximum steady-state gait velocity 27

2.3.6 Optimize for different step lengths 27

2.3.7 Optimize for different step frequencies 27

2.3.8 Examine the robustness of the controller against external perturbations 28

2.3.9 Examine the robustness of the controller against internal perturbations (sensory noise) 28

2.4 Overall assessment 29
2.4.1 Cons 29
2.4.2 Pros 29

3 MPC Controller Implementation 30

3.1 Introduction 30
3.2 Make the Model Continuous 30
3.2.1 Kinematics Equations 30
3.2.2 Equations of Dynamics 32
3.2.3 B matrix computation 33
3.2.4 Ground reaction force 34
3.2.5 Model Simulation and NMPC Controller Implementation 36
3.3 Variable Leg Length 39
3.4 NMPC Controller Information 40
3.5 Discussion about NMPC 41

1 Part I: Theory

Introduction: The relationship between man and his environment has always been the center of fascination of scientist. From Aristotle in antiquity to bio-mechanics today, man have tried to better understand the locomotion of humans. This growing field has recently attracted the interest of various disciplines as new tools have opened up new possibilities of bipedal robotics.

In this first section we will present the first modelling steps to simulate and control a biped walker. These stages touch on: Direct Kinematics, Inverse Kinematics, Dynamics, Impact Map, and Model Validation, Numerical Integration, and finally Control and Optimization. This is the basis needed for implementing a custom controller for the biped walker.

The archetypal model for kinematic arrangement of mostly all bipedal robots is that of human locomotion. The following stages follow this same kinematic approach.

Kinematics
In the kinematics we forget about the masses and focus only on the lengths and the types of degrees of freedom we want to define. The *Direct Kinematics* starts by considering as input: joint angles and as output: the position and orientation of the robot limbs. The *Inverse Kinematics* as the name suggests switches the input and output of the system.

Dynamics
In the dynamics we analyze the the relationship between the joint actuator torques and the resulting motion. To design controllers we use the *inverse dynamics* as this solves for the torques need for motion. This equation is useful as a controller can apply a torque. The dynamics equation is as follows:

$$\tau = \mathbf{M}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}} + \mathbf{G}(\mathbf{q}) + \mathbf{F}(\mathbf{q}, \dot{\mathbf{q}}) \tag{1}$$

where $M(q)$ is the mass matrix, $\ddot{q}$ is the joint space accelerations, $C(q, \dot{q})$ is the coriolis and centrifugal matrix, $\dot{q}$ is the joint velocities, $G(q)$ is the gravity matrix, and $F(q, \dot{q})$ joins together other types of external forces on the robot. However, we first exclude all external forces $F(q, \dot{q})$. On the other hand *direct forward dynamics* assumes the torques are known therefore the acceleration $\ddot{q}$ can be calculated. This equation is useful for simulation, as from the acceleration $\ddot{q}$, the position can be found using numerical integration.

$$\ddot{\mathbf{q}} = \mathbf{M}^{-1}(\mathbf{q})(\tau - \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}} + \mathbf{G}(\mathbf{q}) + \mathbf{F}(\mathbf{q}, \dot{\mathbf{q}})) \tag{2}$$

Equations of Motion
With these equation one can than solve for the equations of motion by many different approaches (Hamiltonian, Newton-Euler), but the one used in class was the Lagrangian method. The Lagrangian method uses the energy of the system. The Lagrangian finds the difference of the kinetic and potential energy.

$$L(\mathbf{q}, \dot{\mathbf{q}}) = T(\mathbf{q}, \dot{\mathbf{q}}) - V(\mathbf{q}) \qquad \frac{d}{dt}\left(\frac{\partial L}{\partial \dot{q}_i}\right) - \frac{\partial L}{\partial q_i} = 0 \quad \forall i \in [1, 2, 3] \tag{3}$$

where $L(q, \dot{q})$ is the Lagrangian, $T(q, \dot{q})$ is the kinetic energy, and $V(q)$ is the potential energy. Solving equation 3 gives the equations of motion. From those equations, one can extract matrices to end up with a matrix equation of the form 1.

The next step of the modelling consists of applying the *Kinematic Model* and *Dynamics Model* described previously to a three link biped as represented here in Figure 1.

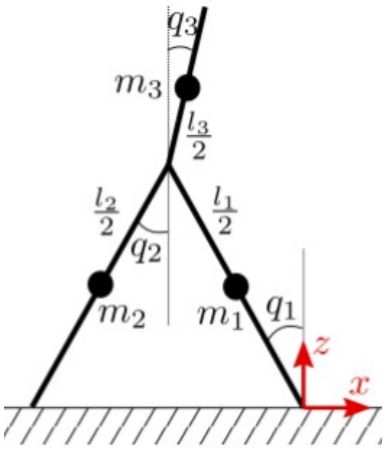

Figure 1: Three Link Biped

This model is a hybrid model between a swing phase model and an impact model. This hybrid model uses the swing phase model to describe the swing of the foot using the equations of motion and uses the impact model to describe the short moment when the swing leg touches the ground. The impact model is used to describe the interaction of the stance foot with the ground since it is hard to include discontinuous dynamics into the equations of motion. A function checks when the swing foot has hit the ground and stops integrating the equations of motion. At this point the impact model calculates the new state of the system and repeats again the swing model with the new parameters.

Swing Phase

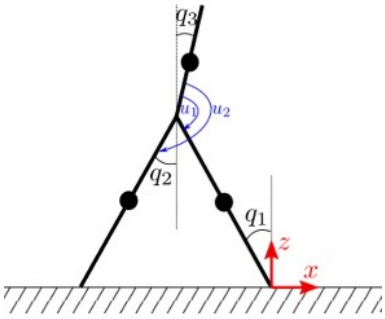

Figure 2: Swing Phase Model

The swing phase as presented here in Figure 2 assumes that one of the legs is the stance leg, while the other is the swing leg. The reference frame is set on the point of contact of the stance foot with the ground. As seen in the image q_1 ,q_2 and q_3 are the angles of legs and torso. The values m_1 ,m_2 and m_3 are the center of mass of each link. Then θ_1 is the angle between the torso and stance leg, and θ_2 is the angle between the torso and the trailing leg. This represents an under-actuated model because we have three degrees of freedom yet we have two motors at the hip for actuation. It means we have two control variables (u_1, u_2) for three degrees of freedom. B is the selection matrix that is equal to our torque τ if multiplied by the control vector u.

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

Impact Map

Upon impact our system experiences during a short duration abrupt changes in velocity. To find the new state of the system after the impact we use an *Impact Map*. At the moment of impact the swing leg changes to the stance leg and vice versa. At this point we would like to change the reference frame. This swap saves us from redoing all the calculations of motion but rather just change the initial conditions and the reference frame as shown in Figure 3.

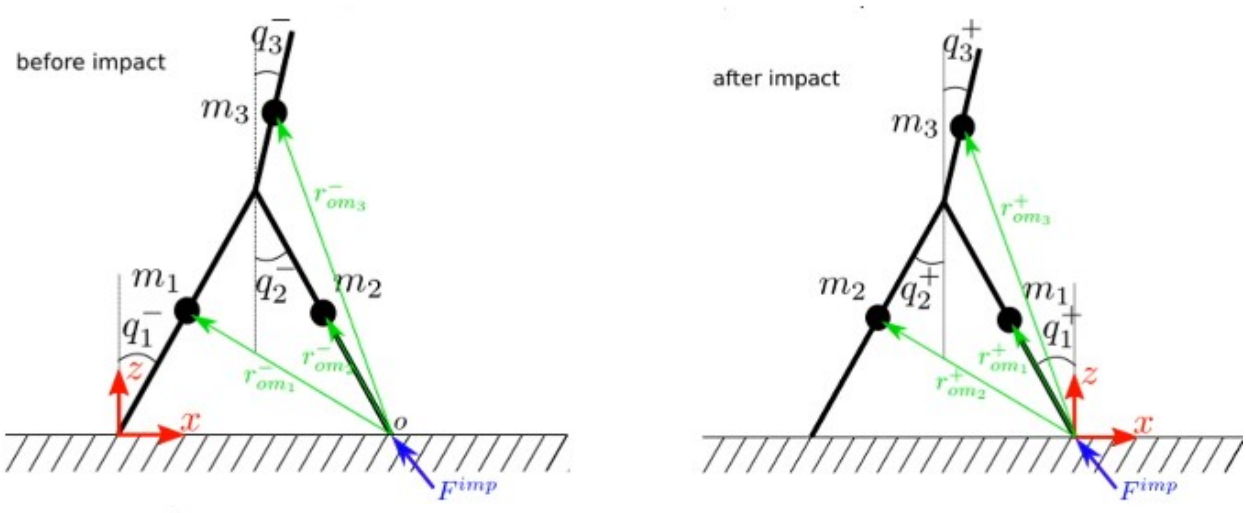

Figure 3: Before and After Impact

The trick is to swap the two legs as shown in eq. 5

$$\begin{bmatrix} q_1^+ \\ q_2^+ \\ q_3^+ \end{bmatrix} = \begin{bmatrix} q_2^- \\ q_1^- \\ q_3^- \end{bmatrix} \tag{5}$$

Conservation of Angular Momentum

Following the impact map, one must calculate the conservation of angular momentum. Upon impact we must consider the transfer of momentum before and after impact. Since we assume that the center of mass of each link is located at $m1$, $m2$ and $m3$ we can say that $H = m \cdot r \times \dot{r}$. The conservation of momentum must be calculated at three different points (impact point H_a, hip point H_b, and torso point H_c as expressed in eq.7). Since there exists conservation of momentum we have that the momentum before and after impact are the same as shown in eq. 6.

$$\mathbf{H}^- = [\mathbf{H}_a^-; \mathbf{H}_b^-; \mathbf{H}_c^-] \quad \mathbf{H}^+ = [\mathbf{H}_a^-; \mathbf{H}_b^+; \mathbf{H}_c^+] \quad \textit{and} \quad \mathbf{H}^+ = \mathbf{H}^- \tag{6}$$

$$\begin{cases} \mathbf{H}_a^- = m_1 \mathbf{r}_{om_1}^- \times \dot{\mathbf{r}}_1^- + m_2 \mathbf{r}_{om_2}^- \times \dot{\mathbf{r}}_2^- + m_3 \mathbf{r}_{om_3}^- \times \dot{\mathbf{r}}_3^- \\ \mathbf{H}_a^+ = m_1 \mathbf{r}_{om_1}^+ \times \dot{\mathbf{r}}_1^+ + m_2 \mathbf{r}_{om_2}^+ \times \dot{\mathbf{r}}_2^+ + m_3 \mathbf{r}_{om_3}^+ \times \dot{\mathbf{r}}_3^- \\ \mathbf{H}_b^- = m_1 \mathbf{r}_{om_1}^- \times \dot{\mathbf{r}}_1^- + m_2 \mathbf{r}_{om_2}^- \times \dot{\mathbf{r}}_2^- + m_3 \mathbf{r}_{om_3}^- \times \dot{\mathbf{r}}_3^- \\ \mathbf{H}_b^+ = m_1 \mathbf{r}_{om_1}^+ \times \dot{\mathbf{r}}_1^+ + m_2 \mathbf{r}_{om_2}^+ \times \dot{\mathbf{r}}_2^+ + m_3 \mathbf{r}_{om_3}^+ \times \dot{\mathbf{r}}_3^- \\ \mathbf{H}_c^- = m_1 \mathbf{r}_{om_1}^- \times \dot{\mathbf{r}}_1^- + m_2 \mathbf{r}_{om_2}^- \times \dot{\mathbf{r}}_2^- + m_3 \mathbf{r}_{om_3}^- \times \dot{\mathbf{r}}_3^- \\ \mathbf{H}_c^+ = m_1 \mathbf{r}_{om_1}^+ \times \dot{\mathbf{r}}_1^+ + m_2 \mathbf{r}_{om_2}^+ \times \dot{\mathbf{r}}_2^+ + m_3 \mathbf{r}_{om_3}^+ \times \dot{\mathbf{r}}_3^- \end{cases} \tag{7}$$

Equations 6 and 7 enables us to compute the next joint velocity state of the robot after impact as a function of the state velocity before impact. We can write $\mathbf{H}^+ = \mathbf{A}^+ \cdot \dot{\mathbf{q}}^+$ and $\mathbf{H}^- = \mathbf{A}^- \cdot \dot{\mathbf{q}}^-$. This means that we can solve for $\mathbf{q}^+$ as shown in eq.8 by multiplying $(\mathbf{A}^+)^{-1}$ on both sides.

$$\dot{\mathbf{q}}^+ = (\mathbf{A}^+)^{-1} \mathbf{A}^- \dot{\mathbf{q}}^- \tag{8}$$

On MATLAB this is done using the *collect* function which will find us A^+.

One can raise some interesting questions about Impact Map:

1. What can you say about the potential energy before and after impact?

   The potential energy remains constant before and after impact because mass points do not change in height during impact

2. Try $q_m = [\pi/6, -\pi/6, pi/10], dq_m = [1, 0.2, 0]$. What percentage of the kinetic energy of the biped is lost due to the impact?

   The impact map implies a 27.9% loss of kinetic energy because of the swing leg hits the ground with a non zero velocity before becoming the stance leg

3. Plot the percentage of the kinetic energy loss due to impact as a function of angle where and varies from 0 to 1.

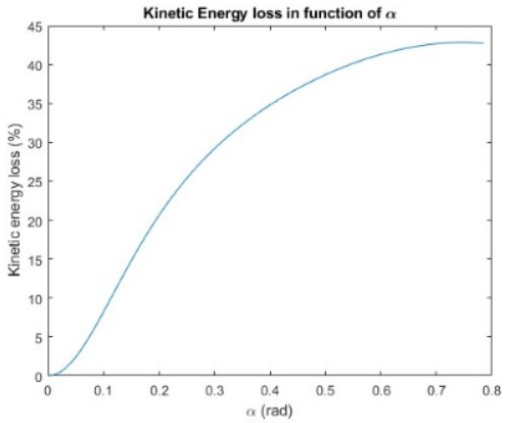

Figure 4: Kinetic Energy loss in function of α

The larger the angle, the greater the kinetic energy loss. If the angle is null, there is no displacement, thus no energy is dissipated

4. The bigger is the bigger is the step length. Based on your answer to question 3, what is the relation between step length and energy loss at impact given a fixed ?

   For a given velocity, the larger the step length the greater the kinetic energy loss because the bigger the α the bigger the step length

Animate

The next state of the project is to visualize the robot motion. In order to do so, we use a first function *Visualize* to plot one frame with the robot in one pose $\mathbf{q}$. Then, we use another function *Animate* to see the motion all throughout the simulation. A special care had been made to create continuity between steps so the stance position is increased at each impact. Indeed, the stance feet is previously always considered at 0. This way, we make the robot moving in the space. Moreover, it enables to define the robot position in the global frame.

Some questions are raised about those two functions:

1. In the animations, what does a real-time factor of 1 mean? How about a real-time factor less than 1?

   $real\_time\_factor$ characterizes the scale time factor between reality and animation. Then, if the $real\_time\_factor$, is equal to 1, the animation is exactly scaled to reality in terms of time duration. Finally, if the $real\_time\_factor$ is greater than 1, the animation is faster than reality (smaller animation time duration

2. How does "skip" in *animate.m* effect the real-time factor and the speed of the animation?

   'skip' enables us to skip animation frames, thus the animation is faster compared to the case where we consider al frames. Then, the $real\_time\_factor$ increases if the 'skip' value increases

3. What is the role of r0 in *animate.m*?

   r0 is the iterative stance foot position all throughout the simulation

Solving Equations of Motion

An important point to be completed now is to solve the equations of motion so the robot can be simulated. In order to do so, we need to numerically integrate the equations of motion to find the next state of the system. Thus, we looked at different methods for solving the ordinary differential equation (ODE).

1. Explicit Euler

It started by looking at the simplest numerical method, *Explicit Euler* shown on equation 9.

$$x(t_0 + dt) = x(t_0) + x'(t_0)dt + O(dt^2) \tag{9}$$

To solve a second order differential equation, the Euler method can be applied twice, by assuming that $u(t) = y'(t)$ and $a(t) = y''(t)$. This changes transform the equation to 10.

$$\begin{aligned} a(t_0) &= -\frac{k}{m}y(t_0) - \frac{d}{m}u(t_0) \\ u(t_0 + dt) &= u(t_0) + a(t_0)dt \\ y(t_0 + dt) &= x(t_0) + u(t_0)dt \end{aligned} \tag{10}$$

with dt the time step of the simulation.

We use this method on a simple second-order equation with a closed form solution:

$$m\ddot{y} + d\dot{y} + ky = 0; y(t_0) = y_0; \dot{y}(t_0) = u_0 \tag{11}$$

The closed form solution is then: $y(t) = Ae^{r-1t} + Be^{r_2t}$

This way we can have a good estimation of the performance of the integrator as the solution is known. This test equation is used in next sections about numerical integration methods.

We simulate the system and answer few questions about Explicit Euler:

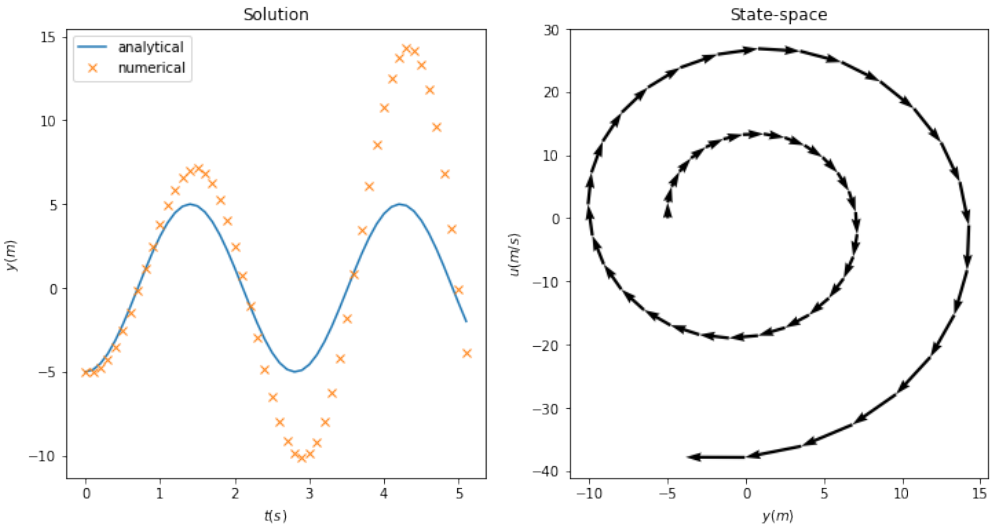

Figure 5: Anyalytical Solution and State-Space Plot

1. Explain the meaning of the state-space plot (right subfigure 5). What should be the expected state-space plot (hint: change visualize function)?

   The state-space plot visualizes the evolution of the velocity with respect to the position, which are the states of the system. The state space plot diverges from the analytical solution. The analytical solution has a plot of a perfect circle. The difference in approximation makes the state-space curve move astray from the perfect circle by a straight line for each timestep.

2. Why does the numerical solution diverges from the true solution (hint: Taylor expansion)?

   The explicit Euler method performs an approximation based on a first-order Taylor expansion. It only gives the perfect solution for functions with zero second derivative and higher. Error is of order $O(dt^2)$.

3. What can you do to better approximate the actual solution with this method?

   Error grows linearly with the timestep squared. We can minimize error by minimizing the timestep.

2. Semi-implicit Euler

The second method we looked at is the Semi-implicit Euler method. This method is similar to the Euler method except that it uses $u(t_0 + dt)$ in the equation for $y(t_0 + dt)$.

The same way, we can reflect on this method with some questions:

1. Is this method stable and when?

   Stability decreases with timestep. For dt>0.9, the method is no longer stable. In fact, the region of stability of the semi-implicit Euler method follows this region:

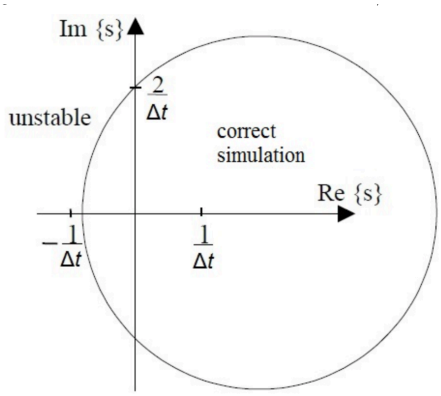

Figure 6: Region of Stability in space map

   The system is $m*s^2+d*s+k=0$ and its eigenvalues are : $s=+/-i\sqrt{20}/2=+/-i*2.236$. Because the eigenvalues are pure imaginary, the condition of stability for this method is $\sqrt{20}/2<2/dt \implies dt=0.894$.

2. What about the accuracy of the obtained solution?

   Semi-implicit Euler is more accurate than the explicit counterpart. Because the state-space diagram follows a elliptic path, error is bounded. However, approximation is not perfect as the state-space diagram follows an elliptic trajectory instead of a circular one. This is due to the fact that semi-implicit Euler relies on future approximations of the derivative values. Since we are approximating the derivative with its future values, the numerical output is time-shifted to the left compared to that of the analytical solution.

3. Runge-Kutta of Order 4 (Midpoint Method)

The final integration method we studied is the Runge-Kutta of Order 4. It is a great integration method which is more precise than previous ones. The method is expressed as follow:

$$\begin{aligned} k_1 &= f(x_0, t_0)dt \\ k_2 &= f(x_0 + \frac{k_1}{2}, t_0 + \frac{dt}{2})dt \\ k_3 &= f(x_0 + \frac{k_2}{2}, t_0 + \frac{dt}{2})dt \\ k_4 &= f(x_0 + k_3, t_0 + dt)dt \\ x(t_0 + dt) &= x_0 + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4) \end{aligned} \tag{12}$$

Some questions about this integration method are raised:

1. What is the error of this method?

   RK4 has an error of O(dt^4).

2. Do you expect the solution to diverge?

   Solution starts diverging for dt $>$ 1.25. The approximation is not always stable but the range of allowed timesteps is wider than before because the approximation is of higher order. A stable solution does not mean a good approximation. In fact, for dt $>$ 1, approximation gets brought down to zero over time.

This integration method is used in the function *ode45* which is the integrator we will use for this project.

4. Adaptive Time Stepping

One important point to be made is that every integration method uses a time step to compute the simulation. A lower time step will end up with a more precise simulation. However, this time step is mostly constant. It is possible to make it variable so it can adapt to the situation and better integrate the system. In order to understand the importance of an adaptive time step, we use the *odeint* function which is an adaptive time stepping integrator.

We answer in this section 3 questions:

1. Change the density of the time points in np.linspace. What do you observe?

   The approximation falls into same points as the analytical solution even for very large timesteps.

2. Do you expect the solution to diverge?

   Odeint performs adaptive time stepping in order to obtain stable results. The solution is not expected to diverge independently from the chosen timesteps.

3. Which algorithms are supported by odeint?

   Odeint uses LSODA algorithm, which solves the initial value problem for stiff or nonstiff systems of first order ordinary differential equations. Odeint performs adaptive time stepping in addition to the LSODA algorithm.

Introduction of a first controller: Method of virtual constraints

The goal of this section is then to build a first controller by virtual constraints. The model has three degrees of freedom with two actuators meaning we have under actuation. Let's recall that u_1 (resp. u_2) directly controls θ_1 (resp. θ_2). The consequences of this is that u_1 controls $\ddot{q}_1$ and $\ddot{q}_3$ while u_2 controls $\ddot{q}_2$ as shown in eq.1 thanks to torque τ. This means there our degrees of freedom are not independent but are coupled between each other. The torque τ applied to the system is computed as follow:

$$\tau = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ -1 & -1 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \end{bmatrix}$$

Ideally we would like to control the torso. To do this we mirror the swing and stance foot mirror each other. We apply the following virtual constraints for our controller to satisfy. To do this we differentiate our constraints, and use the notion of PD (proportional derivative control) to close the loop on the constraints (i.e. y_1, y_2)

$$\begin{array}{lll} q_3 \approx \alpha & y_1 = q_3 - \alpha & \dot{y}_1 = \dot{q}_3 \\ q_2 \approx -q_1 & y_2 = -q_2 - q_1 & \dot{y}_2 = -\dot{q}_2 - \dot{q}_1 \end{array}$$

The PD controller would follow eq.13. The dynamics of the error are such that they follow a critical damped system. This means the dynamics of the error would eventually go to zero.

$$\begin{aligned} u_1 &= k_1^P y_1 + k_1^D \dot{y}_1 \\ u_2 &= k_2^P y_2 + k_2^D \dot{y}_2 \end{aligned} \tag{13}$$

Now that the controller is well defined, it is possible to analyse its performances. First of all, it can be optimized regarding a reward function to reach optimal control and performances. The parameters to optimize are initial conditions and gains:

$$p = \left[q_0, \dot{q}_0, \alpha, k_1^P, k_1^D, k_2^P, k_2^D\right]^T$$

The reward function is defined as follow:

$$f = w_1 \left|\dot{x}_{target} - \bar{\dot{x}}\right| + w_2 CoT - w_3 d$$

where:

$$\begin{aligned} effort &= \frac{1}{2TU_{\max}} \sum_i^T \left(u_1^2(i) + u_2^2(i)\right) \\ CoT &= \frac{effort}{d} \end{aligned} \tag{14}$$

d is the distance traveled by the robot during the simulation.

Remark that this cost function needs some performance metrics such as Cost of Transport and Effort to be computed. We use then a function named *Analyse* to draw those metrics from simulation. The given reward function is a way to track a target velocity while minimizing the Cost of Transport and maximising the distance travelled by the robot. It enables to find a good balance between those performance metrics.

1. First optimization $w_1 = 1; w_2 = 0; w_3 = 0$

A first optimization possible is to fit the target velocity $\dot{x}_{target}$. Simulated performance metrics are summed up in the following table for different target velocities:

$\dot{x}_{target}$	$\bar{\dot{x}}$	**Effort**	**CoT**	**d**	t_f
0.05	0.05	3165	2865	1.1047	5.71s
0.2	0.2	2034	652	3.1175	8.78s
0.4	0.4	2762	891	3.0982	6.36s
0.6	0.6	3788	1248	3.0347	4.74s

Table 1: Optimal solution for a target velocity ($w_1 = 1; w_2 = 0; w_3 = 0$).

We can point that these weights ($w_1 = 1; w_2 = 0; w_3 = 0$) do not end up with a convergence of the mean velocity. If we look at some graphics 7 and 8, we can clearly see that this quantity decreases all during the time of the simulation. For instance, for a target velocity of 0.05, the robot stops and oscillates to fit the velocity at the last frame of the simulation. However, we can clearly see that the initial conditions are not well guest in this case. The robots initial angle velocities are far too high for such a small target velocity. It implies a transition from high speed to low speed movement until the robot stops. Therefore, this result may be explained by a local minima issue for solving the reward function. Taking only one weight fit only the last value of the mean velocity which implies a non-convergence of the trajectory. However, it perfectly fit the target mean velocity. Let's try with adding the weight $w_2 = 0.5$.

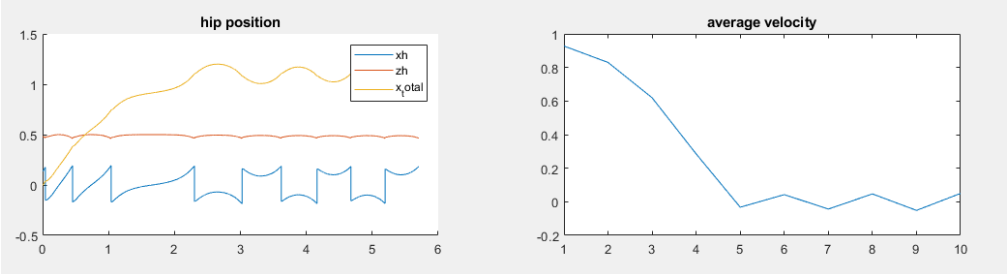

Figure 7: *target velocity* = 0.05

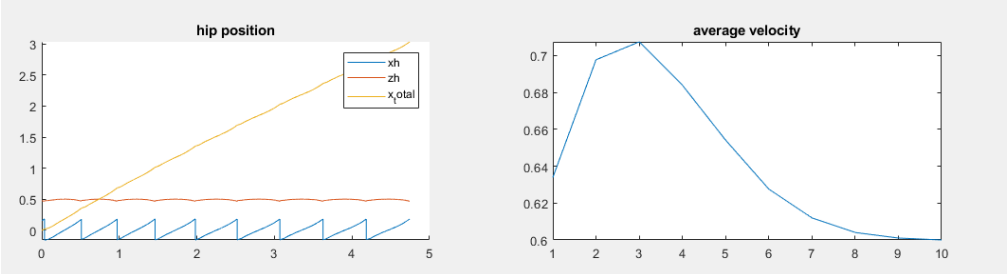

Figure 8: *target velocity* = 0.06

2. Second optimization $w_1 = 1; w_2 = 0.5; w_3 = 0$

Metrics for this simulation are shown below:

$\dot{x}_{target}$	$\bar{\dot{x}}$	**Effort**	**CoT**	**d**	t_f
0.02	0.186	1380	407	3.395	16.41s
0.4	0.186	1380	407	3.395	16.41s

Table 2: Optimal solution for a target velocity ($w_1 = 1; w_2 = 0.5; w_3 = 0$).

In this situation, we can clearly see that whatever the target velocity, the solution is the same. We end up with the optimal CoT (which is asked in the reward function). This is absolutely normal, and it could be expected in the sight of the previous case analysis where we can already guess a minimum of the CoT near a 0.2 velocity. The solution is no longer stabilized at the end of the solution, but it may stabilize at a non-zero velocity after some other simulation steps. Let's remark that the given walked distance is greater than all the previous ones. Thus, in this situation where we only care about w_1 and w_2, the maximum distance seems to be related to a minimum of the CoT. This trajectory is the more "effort efficient" in the sense that it is the best trajectory that use the less effort possible while walking as far as possible. It takes advantage of the dynamic of the walker and puts the less amount of power to walk. We can clearly see this dynamic by looking at the extremely slow movement of the legs when $q_1 \approx q_2 \approx 0$.

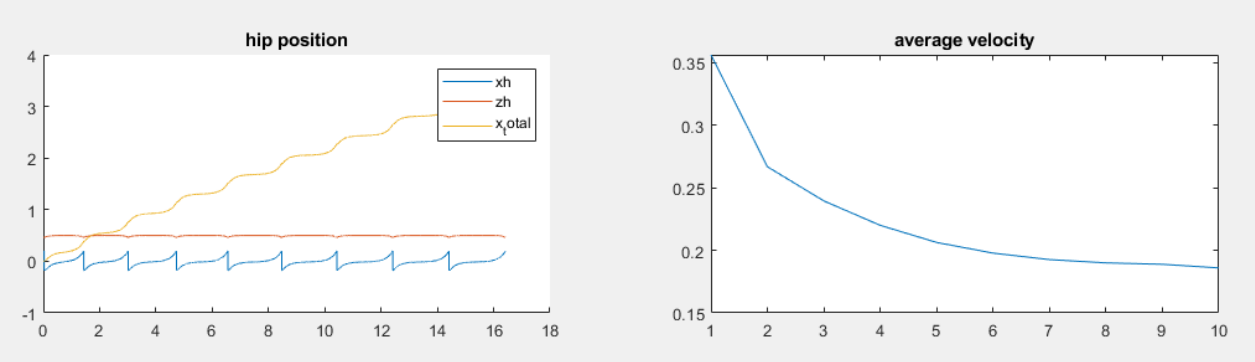

Figure 9: $w_1 = 1\,; w_2 = 0.5$

3. Third optimization $w_1 = 0; w_2 = 0; w_3 = 1$

Let's try now to optimize the distance travelled. We can guess that finding the maximum distance may imply finding the maximum mean velocity as well. In order to do so, we use $w_1 = w_2 = 0$ and $w_3 = 1$.

$\dot{x}_{target}$	**Effort**	**CoT**	**d**	t_f
0.865	11853	3061	3.8722	4.5s

Table 3: Solution for a maximum distance of travel ($w_1 = 0; w_2 = 0; w_3 = 1$)

The outcome of this simulation gives us the maximum distance of travel possible which is equal to 3.87. As expected, the mean velocity is higher than all the previously computed ones. We can reasonably think that this is the maximum velocity of the walker reachable. In this case, we can see on graphic 4 that the mean velocity finally stabilizes at the end of the situation. Therefore, the walker may walk indefinitely if it has enough energy. These maximum distance and velocity performances are only possible at a very high cost in terms of effort and CoT. Of course, walking faster and farther implies a greater CoT and Effort because more energy is needed to compensate the loss of energy at each leg step. All the more so the loss of energy is greater for large length step which is related to high velocity

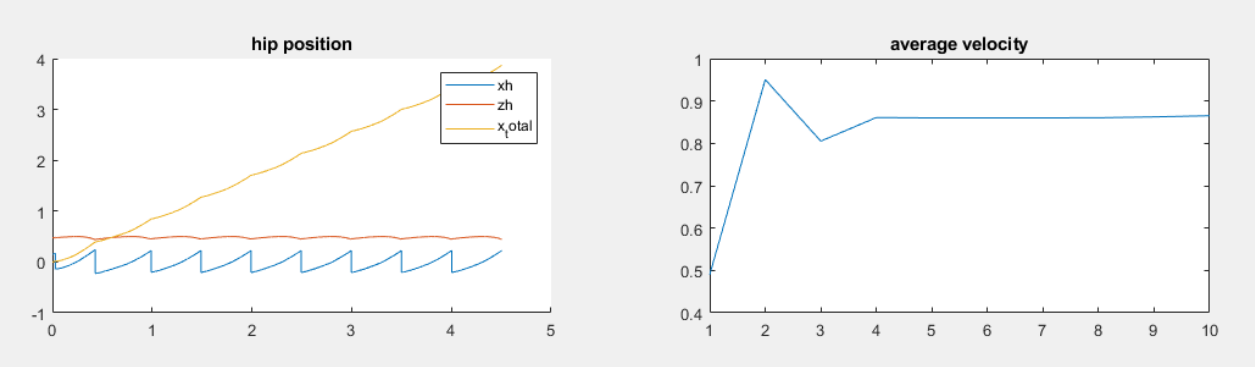

Figure 10: $w_3 = 1$

2 Triple Critically Damped Virtual Oscillators Controller

In this section we present a novel controller for the walking biped that is inspired by the virtual model controller and links each joint to a virtual critically damped tangential harmonic oscillator.

2.1 Starting point: Virtual Model Control:

Virtual Model Control is a creative technique that permits a projection from physical dynamics space to a task space where, instead of using hip torques u_1 and u_2 as system inputs, the engineer is able to design a controller by juxtaposition of separate forces applied on chosen key points of the system. In our case, we project onto three key points: top, hip and swing foot:

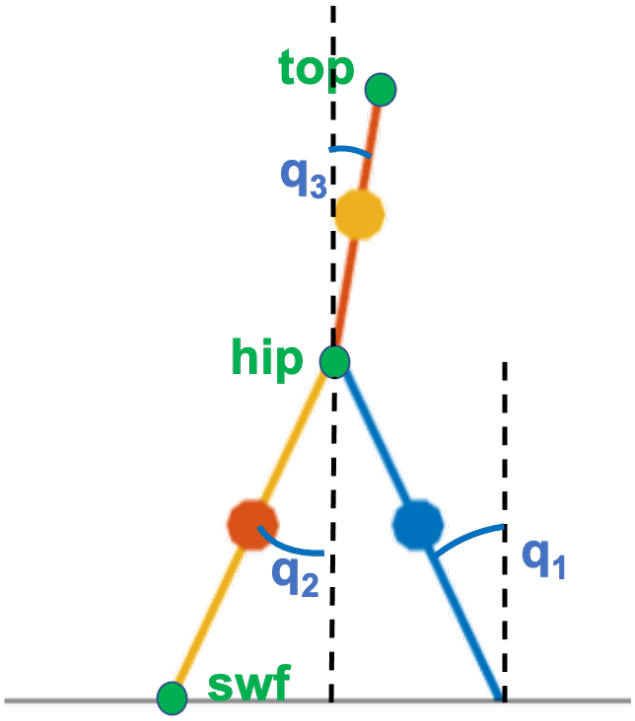

Figure 11: Biped Key Points.

2.1.1 Task space Kinematics

In the task space, the state vector is defined as follow:

$$\mathbf{q}_{task} = [x_h, z_h, x_{swf}, z_{swf}, x_t, z_t]^T \tag{15}$$

The kinematics of the target keypoints in the task space are defined as such:

$$\begin{aligned}
x_h &= l_1 \sin(q_1) \\
z_h &= l_1 \cos(q_1) \\
x_{swf} &= l_1 \sin(q_1) + l_2 \sin(q_2) \\
z_{swf} &= l_1 \cos(q_1) - l_2 \cos(q_2) \\
x_t &= l_1 \sin(q_1) + l_3 \sin(q_3) \\
z_t &= l_1 \cos(q_1) + l_3 \sin(q_3) \\
\dot{x_h} &= l_1 \sin(q_1) \\
\dot{z_h} &= -\dot{q_1}\, l_1 \sin(q_1) \\
\dot{x}_{swf} &= \dot{q_1}\, l_1 \cos(q_1) - \dot{q_2}\, l_2 \cos(q_2) \\
\dot{z}_{swf} &= \dot{q_2}\, l_2 \sin(q_2) - \dot{q_1}\, l_1 \sin(q_1) \\
\dot{x}_t &= \dot{q_1}\, l_1 \cos(q_1) + \dot{q_3}\, l_3 \cos(q_3) \\
\dot{z}_t &= l_1 \sin(q_1)
\end{aligned} \tag{16}$$

2.1.2 Task space Jacobian

In order to permit the change between the task space and the physical space, a Jacobian is computed as follow:

$$\begin{aligned}
J_{i,j} &= \left[\frac{\partial q_{task,i}}{\partial q_{phys,j}} \right] \\
J &= \begin{pmatrix}
l_1 \cos(q_1) & 0 & 0 \\
-l_1 \sin(q_1) & 0 & 0 \\
l_1 \cos(q_1) & -l_2 \cos(q_2) & 0 \\
-l_1 \sin(q_1) & l_2 \sin(q_2) & 0 \\
l_1 \cos(q_1) & 0 & l_3 \cos(q_3) \\
-l_1 \sin(q_1) & 0 & 0
\end{pmatrix}
\end{aligned} \tag{17}$$

where $q_{task,i}$ in $[x_h, z_h, x_{swf}, z_{swf}, x_t, z_t]^T$ and $q_{phys,j}$ in $[q_1, q_2, q_3]^T$.

2.1.3 Input transform:

Now that the task space Jacobian is computed, a transformation from input torques in the physical system to separate input forces in the task space is made possible as shown below:

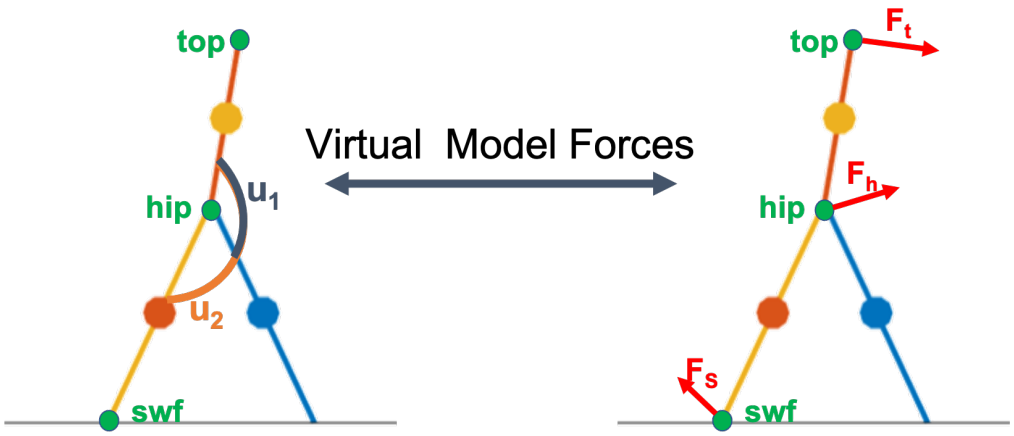

Figure 12: Transforming Physical Torques to Task Forces.

The equations that links the two input spaces can be extracted:

$$\begin{aligned} \tau^T \dot{\mathbf{q}}_{phys} &= \mathbf{F}^T \dot{\mathbf{q}}_{task} \\ \tau &= \mathbf{J}^T(\mathbf{q}_{phys})\mathbf{F} \\ u &= \mathbf{B}^+ \mathbf{J}^T(\mathbf{q}_{phys})\mathbf{F} \quad so\ \mathbf{Bu} = \tau \end{aligned} \tag{18}$$

where $\mathbf{B}^+$ is the pseudo-inverse of B.

2.2 Designing the controller

2.2.1 System separation

Thanks to the kinematic transformation stated above, a controller is designed such that the inputs are forces applied on each of the key points (top, hip and swing foot) instead of the two torques on the legs. This makes the control scheme much easier and much more intuitive to design, considering that custom inputs can appropriately be applied to each one of the task key points.

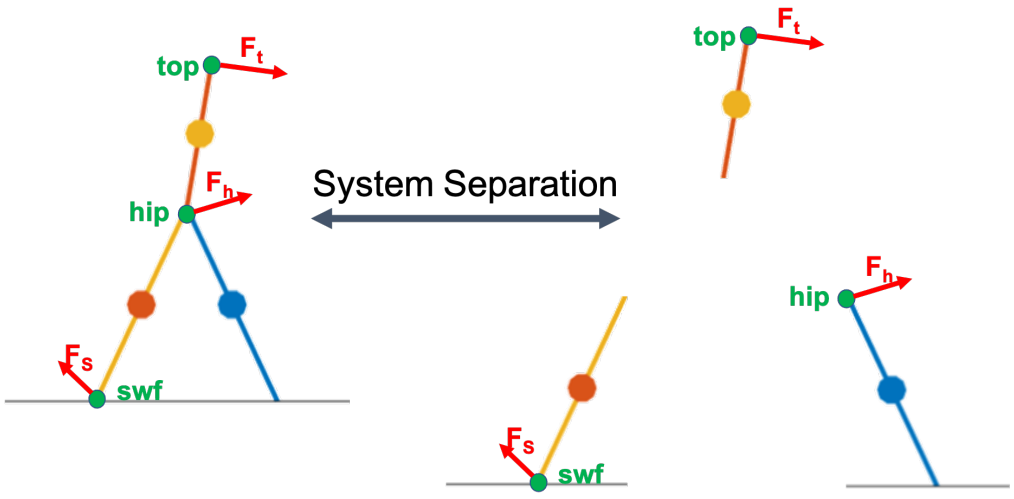

Figure 13: Separating inputs.

2.2.2 Shape of the applied forces: Triple harmonic oscillators

On each of the three task key points, a force is applied which is equivalent to a tangential spring and damper system. The force of each damper system $\mathbf{f}_h$, $\mathbf{f}_s$, $\mathbf{f}_t$ is controlled by the physical state angles q_1, q_2, q_3 respectively. The figure below illustrates the equivalence between $\mathbf{f}_t$ that is exerted on the top keypoint and a harmonic oscillator.

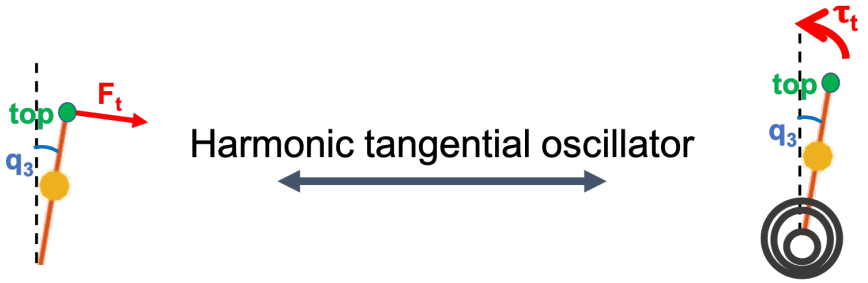

Figure 14: Equivalent top harmonic oscillator.

The forces are defined as such:

$$||\mathbf{f}_i||_2 = \frac{\tau_i}{l_i} = -k_i(q_i - q_{target,i}) - b_i\dot{q}_i \tag{19}$$

Where $i \in [h, s, t]$ and $i \in [1, 2, 3]$ for q_i.

System dynamics are then defined as follow:

$$\begin{aligned} \Sigma\tau &= Iq_i^2 \\ l_j||\mathbf{f}_j||_2 \pm l_j m_j g sin(q_j) &= m_j l_j^2 \ddot{q}_j \\ m_j l_j^2 \ddot{q}_j + l_j b_j \dot{q}_j + l_j k_j (q_j - q_{target,j}) \pm l_j m_j g sin(q_j) &= 0 \end{aligned} \tag{20}$$

Where $j \in [h, s, t]$ for $\mathbf{f}_j$, k_j and b_j, $j \in [1, 2, 3]$ for other parameters. Depending on the direction of the oscillator, the sign of the weight force can either be opposed or on the same side as $\mathbf{f}_i$. In fact, for top and hip, because zero position is facing up, the weight force is opposed to the oscillator force. As for the swing foot, zero position is facing downwards, making the weight and the spring force acting on the same direction. Finally, after using small angle approximation for sin function, the resulting dynamic equation can be written as:

$$k_{sys,j}(q_j - q_{target,j}) + b_{sys,j}\dot{q}_j + m_{sys,j}\ddot{q}_j = 0 \tag{21}$$

Where $j \in [h, s, t]$ and $j \in [1, 2, 3]$ for q_j, $\dot{q}_j$ and $\ddot{q}_j$. k_{sys}, b_{sys} and m_{sys} are the equivalent characteristics of the equivalent harmonic oscillator.
The table below shows the equivalent mass, damping and spring constants for each of the three systems:

Oscillator	k_{sys}	b_{sys}	m_{sys}
top	$k_3 l_3 - m_3 g \frac{l_3}{2}$	$b_3 l_3$	$m_3 \frac{l_3}{2}^2$
hip	$k_1 l_1 - m_1 g \frac{l_1}{2}$	$b_1 l_1$	$m_1 \frac{l_1}{2}^2$
swing foot	$k_2 l_2 + m_2 g \frac{l_2}{2}$	$b_2 l_2$	$m_2 \frac{l_2}{2}^2$

2.2.3 Final Virtual Forces

All in all, we can define the tangential virtual forces applied on the hip, top and swing foot thanks to the previous considerations.

$$\begin{aligned} \mathbf{f}_h &= \big[-(k_h(q_1 - q_{target,1}) + b_h\dot{q}_1)cos(q_1),\ (k_h(q_1 - q_{target,1}) + b_h\dot{q}_1)sin(q_1)\big]^T \\ \mathbf{f}_{swf} &= \big[\ (k_s(q_2 - q_{target,2}) + b_s\dot{q}_2)cos(q_2), -(k_s(q_2 - q_{target,2}) + b_s\dot{q}_2)sin(q_2)\big]^T \\ \mathbf{f}_t &= \big[-(k_t(q_3 - q_{target,3}) + b_t\dot{q}_3)cos(q_3),\ \ (k_t(q_3 - q_{target,3}) + b_t\dot{q}_3)sin(q_3)\big]^T \end{aligned} \tag{22}$$

Those forces are grouped in one force vector and applied on the system as follow:

$$\mathbf{u} = \mathbf{B}^+\mathbf{J}^T\mathbf{F} = \mathbf{B}^+\mathbf{J}^T[\mathbf{f}_h^T, \mathbf{f}_{swf}^T, \mathbf{f}_t^T]^T \tag{23}$$

2.2.4 Definition of the working domain of the stance leg harmonic oscillator

When simulating the whole system, we remarked that if the stance oscillator on the hip overpasses its target angle and moves back, it destabilizes the top oscillator by impelling an high acceleration on the torso mass because of the speed reversal. This behavior destabilizes the gait and the robot. Therefore, in order to avoid this behavior, we decided to adapt the force associated to the stance oscillator. The stance oscillator should only be active if $q_1 < 0$. If this condition is not satisfied, no force works. This enables to avoid any speed reversal which could destabilize the torso. All the more so it lets more time to the swing leg to go ahead. Finally, after some simulations with disturbances, it makes the robot more resilient against internal perturbations.

The new force applied on the hip is written as follow:

$$\mathbf{f}_h = (q_1 < 0) \times \left[-(k_h(q_1 - q_{target,1}) + b_h\dot{q}_1)cos(q_1), (k_h(q_1 - q_{target,1}) + b_h\dot{q}_1)sin(q_1)\right]^T \quad (24)$$

2.2.5 Synchronization of the swing harmonic oscillator

When simulating the system, we noticed that the swing leg starts its movement too fast, even before the stance leg is well positioned so the swing leg can well swing forward. Fully decoupled oscillators as it is sounds then non optimal. It would be better if the stance and the swing oscillators are coupled one to another so the swing one is only triggered when it is optimal. Then, when does the swing leg can start its movement? The perfect timing for the swing leg to move is when $q_1 \approx 0$. If it not the case, the swing leg will probably hit the ground before switching side which has to be avoided as much as possible.

Moreover, one important thing is that when the swing leg achieves to switch side, we still want it to track the angle target. At this moment, the stance leg will start falling forward so q_1 will become positive. As a consequence, the swing oscillator has to be triggered when $q_1 >\approx 0$. This condition couples the swing oscillator to the stance one which makes the transition between legs easier.

In order to implement that principle, we decided to use a sigmoid function so if q_1 is negative, it returns no force while if q_1 is positive, the oscillator is active. In between, a smooth transition can be tuned thanks to the sigmoid parameters. The new virtual force considered is written below:

$$\mathbf{f}_{swf} = \frac{1}{1 + e^{\frac{-q_1}{2(\sigma_{swf})^2}}} \begin{pmatrix} k_s(q_2 - q_{target,2}) + b_s\dot{q}_2)cos(q_2) \\ -(k_s(q_2 - q_{target,2}) + b_s\dot{q}_2)sin(q_2) \end{pmatrix} \quad (25)$$

2.2.6 Lifting of the swing foot when starting a step

Finally, one may notice that despite the coupling of the swing oscillator, the swing leg tends to fall because of its mass so it hits the ground before $q_1 \approx 0$. Therefore, it could be interesting to avoid this issue by lifting artificially the swing leg while the stance leg is not well positioned. After this, the swing oscillator can take the control and make the leg move. However, which lifting force to apply ? We don't want the robot to move back. Then, a tangential force to the back is not suitable. The simplest force that we can think of is a constant force in the z direction. This force enables to counteract the gravity dynamics by constantly lifting the swing leg while $q_1 <\approx 0$. A sigmoid function enables to consider this range of force activation. The new virtual swing force to be applied is then written as follow:

$$\mathbf{f}_{swf} = \frac{1}{1+e^{\frac{(q_1-q_{1,lifttrigger})}{2(\sigma_{swf})^2}}}\begin{pmatrix}0\\ f_{lift}\end{pmatrix} + \frac{1}{1+e^{\frac{-q_1}{2(\sigma_{swf})^2}}}\begin{pmatrix}k_s(q_2-q_{target,2})+b_s\dot{q}_2)cos(q_2)\\ -(k_s(q_2-q_{target,2})+b_s\dot{q}_2)sin(q_2)\end{pmatrix} \quad (26)$$

The same parameters as the previous section 2.2.5 are taken so a smooth transition between the two forces can be achieved.

2.3 Choosing the right gains

Now that the model is well defined, we have to select parameter values. In order to do so, we will first impose a harmonic regime of the oscillators for performances and dimension reduction purposes. Then, parameters will be found by hand and finally optimized to end up with optimized gaits.

2.3.1 Critical damping

Once the expression of the equivalent mass, damping and stiffness of each system is obtained, these gains can be tuned such that the masses are critically damped. This not only offers the fastest convergence to the target point but also offers a motion with no overshoot or oscillation. This aspect becomes more and more important if we consider that at the end of the control loop, only two torques will end up being applied on the system. Thus, in case one of the masses overshoots or oscillates, the other masses will be affected with this motion, especially that input is constrained to a certain limit. In fact, if one controller is using a large part of the maximum input allowed, there is a good chance that the other masses will not be able to track their desired motion.

The critical damping regime can be obtained by the following condition:

$$\begin{aligned}\tau_i &= \frac{2m_{sys,i}}{b_{sys,i}}\\ \frac{k_{sys,i}}{m_{sys,i}} &- 1/\tau_i^2 = 0\end{aligned} \quad (27)$$

Where τ_i is the time constant of each system i.
This time constant can be tuned such that the settling time of each oscillator follows a certain walking velocity. In other words, each oscillator settles at the desired time which makes the biped walk with a desired speed. q_{target} is also an important parameters that impacts the walking step length of the biped. In fact, the leg oscillators will move from -q_{target} to q_{target}, making the step angle $2q_{target}$. Finally, the **step length** is controlled by:

$$step\ length = 2l_1 sin(q_{target}). \tag{28}$$

After calculating the critical damping conditions, the values of b_{sys} and k_{sys} become linked, reducing by one the dimension of the controlled parameter space for each one of the oscillators:

Oscillator	k_{sys}
top	$\frac{2b_3^2+gl_3m_3^2}{2l_3m_3}$
hip	$\frac{2b_1^2+gl_1m_1^2}{2l_1m_1}$
swing foot	$\frac{2b_2^2-gl_2m_2^2}{2l_2m_2}$

As a consequence, only damper parameters have to be set. Spring parameters are computed according to those choices.

2.3.2 Global parameter optimization

The next step is to select gains. The idea is to first find parameters by hand so we can initialize an optimization solver to finer tune them afterwards. Let's recall the parameters to be found:

$$b_h, b_s, b_t, q_{target,1}, q_{target,2}, q_{target,3}, \sigma_{swf}, f_{lift} \text{ and } q_{1,lifttrigger}$$

After some tries and errors, we finally found some initial parameters which are summed up in the table below.

Then, after exploring a robust-enough combination of parameters that enables the biped to walk a few steps, a global optimization of a cost function is performed. The *GlobalSearch* module of **MATLAB Global Optimization Toolbox** is used to find global minimizers instead of using $fminsearch$, which looks for local solutions only. In fact, the *GlobalSearch* module uses a bench of local optimizers to find a global minima. Below is the first list of parameters optimized when seeking a maximum walked distance by the robot:

Parameter	*Description*	*Initial value*	*Optimized value*
b_h	Damping coefficient of the hip damper	100	99.9994
b_s	Damping coefficient of the swing foot damper	100	99.9995
b_t	Damping coefficient of the top damper	100	99.9993
$q_{target,1}$	Target angle of the hip oscillator	$\frac{\pi}{9}$	0.3227
$q_{target,2}$	Target angle of the swing foot oscillator	$-\frac{\pi}{9}$	-0.2674
$q_{target,3}$	Target angle of the top oscillator	0	0.0049
σ_{swf}	Controls the slope of the sigmoid of $\mathbf{f}_{swf}$, ensuring swing foot oscillator only acts for negative q_1	$\frac{\pi}{18}$	0.1450
f_{lift}	Peak value of the lifting force of the swing foot during early swing	100	99.9998
$q_{1,lift\ trigger}$	Angle for which lifting force of swing foot stops acting	$-\frac{\pi}{12}$	-0.2187

2.3.3 Cost function

In order to assess how the controller performs for multiple objectives, a cost function is used, consisting of a weighted sum of multiple gait metrics: target velocity, cost of transport, walking distance, step length and step frequency. This cost function is given as input to the global optimizer and the minimizing parameter combinations computed make the controller optimally better under the desired conditions. The cost function is expressed as follow:

$$\begin{aligned} f_{cost} &= w_1|v_{target} - v_{average}| + w_2 CoT - w_3 d_{total} + w_4 l_{step}^2 + w_5 f_{step}^2, \\ where\ CoT &= \frac{effort}{d_{total}} \\ and\ effort &= \frac{1}{2TU_{max}} \sum_{i=0}^{i=i_{end}} (u_1^2(i) + u_2^2(i)) \end{aligned} \tag{29}$$

If the objective was to walk the longest distance possible, w_3 is set to 1 and the rest of the weights are set to 0. Similarly, target velocities, step frequencies and step lengths can be reached.
Also, to keep the model from falling or from walking backwards, each time the cost function is evaluated, if the walker were falling or if it its velocity were to be negative, the cost function is by default evaluated to 10000.

2.3.4 Maximum number of steps

The controller easily manages to reach a walking periodic state. Without any disturbance, the walker is able to keep walking without falling until the end of the simulation. We tried simulating with increased total time but the model didn't fall and kept walking until the end. This is mainly due to how the controller is designed. In fact, if the model posture deviates from the desired state, the model is able to correct it with fast and successive steps, making it highly robust.

2.3.5 Minimum and maximum steady-state gait velocity

The controller succeeds to reach a broad range of walking velocities. When optimizing the cost function with only the target velocity weight set to 1, we obtain:

- The minimum average velocity reached is $0.69 m.s^{-1}$.
- The maximum average velocity reached is $0.9 m.s^{-1}$.
- The peak step velocity reached is $1.82 m.s^{-1}$.

2.3.6 Optimize for different step lengths

Because of the way the controller is designed, to correct its posture, the walker is able to perform fast small successive steps that keep it from falling. This greatly impacts its robustness but consequently reduces the average step lengths. Below are the results obtaining when setting the weight for maximizing step length:

- The minimum average step length reached is $0.05m$.
- The maximum average step length reached is $0.1m$.
- The peak step length reached is $0.34m$.

2.3.7 Optimize for different step frequencies

Because of the same phenomenon stated before, step frequencies are quite high for a given walking velocity. Optimizing the cost function with only the weight for step frequency set to 1 we obtain:

- The minimum average step frequency reached is $3.7Hz$.
- The maximum average step frequency reached is $24.9Hz$.

2.3.8 Examine the robustness of the controller against external perturbations

For external perturbations, a force is applied on the hip along the x direction, with a duration of 0.2s. Different values for push or pull forces are tried in order to assess the robustness of the model against external perturbations. We also tried reoptimizing the controller when applying the forces to test how far it can withstand external disturbance.
Below are the obtained results:

- The model manages to stabilize itself for push forces reaching up to $103N$. When reoptimizing, the forces reached $180N$.
- The model manages to stabilize itself for pull forces reaching up to $-29N$. When reoptimizing, the forces reached $-80N$.

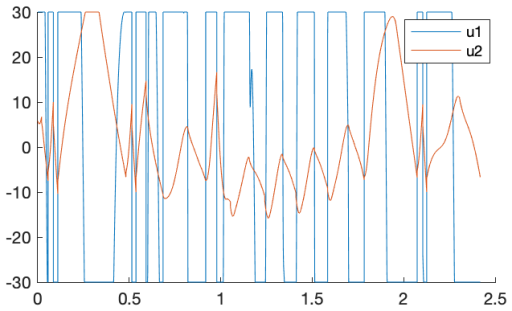

Figure 15: Input torques after simulation.

After analyzing the norm of input torques applied by the controller during walking, we see that the input u_1 is always saturated. This is a plausible cause for the robustness of the model against external perturbation. In other words, the input torques are always high such that they are able to easily oppose high perturbation forces.

2.3.9 Examine the robustness of the controller against internal perturbations (sensory noise)

The controller's robustness to internal perturbations is assessed by applying a Gaussian noise $N(0, \sigma)$ independently on each coordinate (e.g., only on q_1, then only on q_2, $\dot{q}_1$, etc.). We then note the maximum value of σ that the controller can tolerate for each coordinate independently. The noise is applied before evaluating the controller.
Below are the results obtained:

- The maximum standard deviation for $\mathbf{q}$: $\sigma_q = [0.2, 0.09, 0.07]^T$.
- The maximum standard deviation for $\dot{\mathbf{q}}$: $\sigma_{\dot{q}} = [9.9, 1.3, 1.9]^T$.

2.4 Overall assessment

2.4.1 Cons

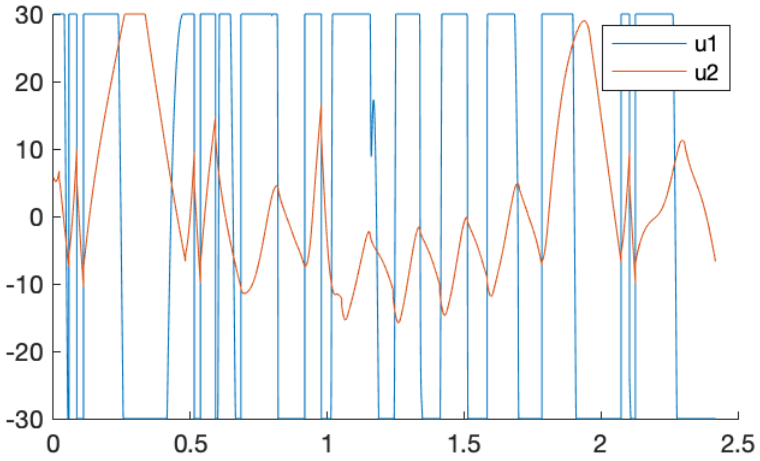

Figure 16: Input torques after simulation.

The plot above shows the shape of input torques applied during a simulation which aims to maximize walking distance only. We can clearly observe that u_1, the torque applied between the stance leg and the torso is always saturated and switches between the input extremities. This indicates a high amount of energy is being consumed during the whole walking cycle. In order to maximize walking distance and at the same time consume the least amount of energy, further investigations need to be undergone where different weights are to be assigned to the cost-of-transport and the total-distance components in the cost function.

2.4.2 Pros

Overall, the proposed controller presents a novel way of making this two-legged biped walk with fairly simple mechanical principles. By being able to extract all dynamic information from the model, we managed to easily tweak force gains such that the oscillators are critically damped. This not only enables a fast transition from a point to the following one in the gait cycle but also with little to no overshoot.
The model is not only robust to external perturbations but also to internal state noise. This mainly due to its ability to perform small corrective steps to straighten its posture before continuing the walking motion.
Also, setting oscillators to work in critical regime reduces by one the controllable degrees of freedom for each one of the oscillators. Moreover, global optimization of the cost function converges to global solutions relatively fast, indicating a low computation cost.

3 MPC Controller Implementation

3.1 Introduction

The first idea we think about when dealing with a biped walker is the implementation of an NMPC controller which can lead to great results in terms of robustness against perturbations, feasibility, performances and prediction of foot steps which can help to avoid "holes" or obstacles. Moreover, NMPC controllers are useful for fitting constraints. However, we quickly felt the need to change the robot model. We guessed that the model has to be continuous so prediction on several foot steps is possible.

First we tried to implement an NMPC controller using *nlmpc* (from the Matlab Model Predictive Control Toolbox) on the first model defined previously which uses an impact, so leads to a discontinuous model of the dynamics. As expected, the controller can only predict leg movements **before** the hit of the swing leg on the ground. Any further time simulation after the impact of the feet on the ground leads to simulation errors and can not be handled by our first NMPC controller because of the discontinuity.

Therefore, we decided to change the robot model into a continuous one to avoid using an impact map. The following section 3.2 presents the new continuous model.

3.2 Make the Model Continuous

In this section, the new continuous model will be explained step by step starting from the kinematics considerations in section 3.2.1.

3.2.1 Kinematics Equations

The first problem we have to deal with is that the robot is now "free" to move. The robot is not attached to any fixed point, contrary to the first model implemented which considers the stance feet as fixed. How is it then possible to express the kinematics equations without such a point ?

One guess is to use the center of mass of the robot to express the local relations. All in all, the center of mass $[x_g, z_g]^T$ defines the position of the robot in the space and parameters q_1, q_2 and q_3 defines the pose of the robot at that position. A new state vector is defined as follow:

$$\mathbf{q} = [x_g, z_g, q_1, q_2, q_3]^T \tag{30}$$

The robot structure considered in this MPC section is shown in figure 17. There is no swing foot or stance leg anymore but only legs number one and two:

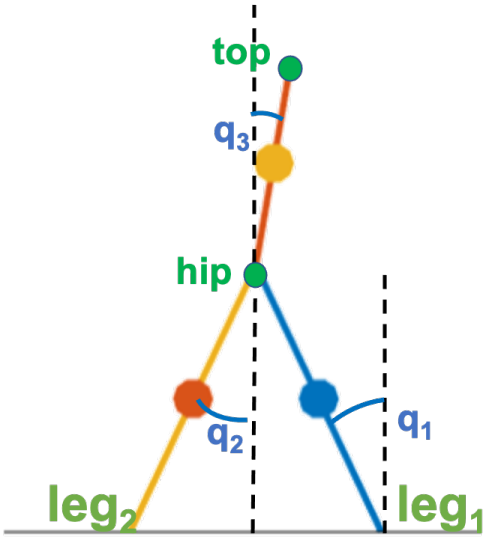

Figure 17: New biped model

Each mass point of the robot can be easily related to the hip point as follow:

$$\begin{aligned}
[x_1, z_1]^T &= [x_h - l_1 sin(q_1)/2, z_h - l_1 cos(q_1)/2]^T \\
[x_2, z_2]^T &= [x_h - l_2 sin(q_2)/2, z_h - l_2 cos(q_2)/2]^T \\
[x_3, z_3]^T &= [x_h + l_3 sin(q_3)/2, z_h + l_3 cos(q_3)/2]^T
\end{aligned} \tag{31}$$

Finally, the center of mass can be written using equations 31.

$$\begin{aligned}
[x_g, z_g]^t &= \frac{m_1[x_1, z_1]^T + m_2[x_2, z_2]^T + m_2[x_2, z_2]^T}{m_1 + m_2 + m_3} \\
&= [x_h + \Delta_x(q_1, q_2, q_3), z_h + \Delta_z(q_1, q_2, q_3)]^T
\end{aligned} \tag{32}$$

with:

$$\begin{aligned}
\Delta_x(q_1, q_2, q_3) &= \frac{m_3 l_3 sin(q_3) - m_1 l_1 sin(q_1) - m_2 l_2 sin(q_2)}{m_1 + m_2 + m_3} \\
\Delta_z(q_1, q_2, q_3) &= \frac{m_3 l_3 cos(q_3) - m_1 l_1 cos(q_1) - m_2 l_2 cos(q_2)}{m_1 + m_2 + m_3}
\end{aligned} \tag{33}$$

Therefore, we can express the hip point and the mass points as functions of the state vector $\mathbf{q}$:

$$\begin{aligned}
[x_h, z_h]^T(\mathbf{q}) &= [x_g - \Delta_x(q_1, q_2, q_3), z_g - \Delta_z(q_1, q_2, q_3)]^T \\
[x_1, z_1]^T(\mathbf{q}) &= [x_g - \Delta_x(q_1, q_2, q_3) - l_1 sin(q_1)/2, z_g - \Delta_z(q_1, q_2, q_3) - l_1 cos(q_1)/2]^T \\
[x_2, z_2]^T(\mathbf{q}) &= [x_g - \Delta_x(q_1, q_2, q_3) - l_2 sin(q_2)/2, z_g - \Delta_z(q_1, q_2, q_3) - l_2 cos(q_2)/2]^T \\
[x_3, z_3]^T(\mathbf{q}) &= [x_g - \Delta_x(q_1, q_2, q_3) + l_3 sin(q_3)/2, z_g - \Delta_z(q_1, q_2, q_3) + l_3 cos(q_3)/2]^T
\end{aligned} \tag{34}$$

Moreover, we can define the end points of the top (torso) and legs 1 and 2 (no swing nor stance leg anymore) the same way.

$$\begin{aligned}
[x_{l1}, z_{l1}]^T(\mathbf{q}) &= [x_g - \Delta_x(q_1, q_2, q_3) - l_1 sin(q_1), z_g - \Delta_z(q_1, q_2, q_3) - l_1 cos(q_1)]^T \\
[x_{l2}, z_{l2}]^T(\mathbf{q}) &= [x_g - \Delta_x(q_1, q_2, q_3) - l_2 sin(q_2), z_g - \Delta_z(q_1, q_2, q_3) - l_2 cos(q_2)]^T \\
[x_t, z_t]^T(\mathbf{q}) &= [x_g - \Delta_x(q_1, q_2, q_3) + l_3 sin(q_3), z_g - \Delta_z(q_1, q_2, q_3) + l_3 cos(q_3)]^T
\end{aligned} \tag{35}$$

The final step of this section is to compute the velocity of each previous point. In order to achieve those computations, we use the *diff* function and the symbolic representation in Matlab. The latter returns symbolic equations of velocities. Given a variable x, it computes:

$$\dot{x}(\mathbf{q}) = \frac{\partial x}{\partial x_g} dx_g + \frac{\partial x}{\partial z_g} dz_g + \frac{\partial x}{\partial q_1} dq_1 + \frac{\partial x}{\partial q_2} dq_2 + \frac{\partial x}{\partial q_3} dq_3 \tag{36}$$

We have now access to the kinematic equations of the new continuous model. The next step is to compute the dynamics equations of the robot in subsection 3.2.2 so it can be simulated.

3.2.2 Equations of Dynamics

In this section, the equations of the dynamics are drawn from the Lagrange method which follows the same considerations as the first non-continuous model. Using the equation 36, we can compute the potential energies V_1, V_2 and V_3 and kinematic energies T_1, T_2 and T_3 of each mass point such that:

$$\begin{aligned}
T_i &= m_i(\dot{x}_i^2 + \dot{z}_i^2) \\
V_i &= m_i z_i g0 \qquad\qquad \forall i \in [1, 2, 3]
\end{aligned} \tag{37}$$

Given those quantities, the Lagrangian of the system can be computed as follow:

$$L = (T_1 + T_2 + T_3) - (V_1 + V_2 + V_3) \tag{38}$$

This quantity is used to draw the five equations of motion:

$$Eq_i(\mathbf{q}) = \frac{d}{dt}\left(\frac{\partial L}{\partial \dot{\mathbf{q}}_i}\right) - \frac{\partial L}{\partial \mathbf{q}_i} = 0 \quad ; \forall i \in [1, 2, 3, 4, 5] \tag{39}$$

Given the five equations of the dynamics, we would like to have a matrix representation of the system as follow:

$$\mathbf{M}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}} + \mathbf{G}(\mathbf{q}) = \mathbf{0} \tag{40}$$

We can extract the matrices from equations 39 following the same strategy as explained in the first section of this report. Results are summed up below:

$$\mathbf{M}(\mathbf{q}) = \begin{pmatrix} m_1+m_2+m_3 & 0 & \frac{l_1 m_1 cos(q_1)}{2} & \frac{l_2 m_2 cos(q_2)}{2} & -\frac{l_3 m_3 cos(q_3)}{2} \\ 0 & m_1+m_2+m_3 & -\frac{l_1 m_1 sin(q_1)}{2} & -\frac{l_2 m_2 sin(q_2)}{2} & \frac{l_3 m_3 sin(q_3)}{2} \\ \frac{l_1 m_1 cos(q_1)}{2} & -\frac{l_1 m_1 sin(q_1)}{2} & \frac{l_1^2 m_1}{4} & 0 & 0 \\ \frac{l_2 m_2 cos(q_2)}{2} & -\frac{l_2 m_2 sin(q_2)}{2} & 0 & \frac{l_2^2 m_2}{4} & 0 \\ -\frac{l_3 m_3 cos(q_3)}{2} & \frac{l_3 m_3 sin(q_3)}{2} & 0 & 0 & \frac{l_3^2 m_3}{4} \end{pmatrix}$$

$$\mathbf{C}(\mathbf{q}, \dot{\mathbf{q}}) = \begin{pmatrix} 0 & 0 & -\frac{\dot{q_1} l_1 m_1 sin(q_1)}{2} & -\frac{\dot{q_2} l_2 m_2 sin(q_2)}{2} & \frac{\dot{q_3} l_3 m_3 sin(q_3)}{2} \\ 0 & 0 & -\frac{\dot{q_1} l_1 m_1 cos(q_1)}{2} & -\frac{\dot{q_2} l_2 m_2 cos(q_2)}{2} & \frac{\dot{q_3} l_3 m_3 cos(q_3)}{2} \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

$$\mathbf{G}(\mathbf{q}) = \begin{pmatrix} 0 & g(m_1+m_2+m_3) & -\frac{g l_1 m_1 sin(q_1)}{2} & -\frac{g l_2 m_2 sin(q_2)}{2} & \frac{g l_3 m_3 sin(q_3)}{2} \end{pmatrix}^T$$

Remark that the **C** matrix is significantly simpler than for the previous non continuous model, contrary to the **M** matrix. The interactions between the dynamics equations are mostly present in the **M** matrix as opposed to the first non continuous model.

Moreover, remark that the system is highly non linear, therefore, a non linear MPC controller is needed. The *nlmpc* function of the Matlab Model Predictive Control Toolbox is then used.

3.2.3 B matrix computation

The robot is expected to be controlled by two actuators represented by inputs $\mathbf{u} = [u_1, u_2]^T$ exactly the same way as in the previous model. We can define two angles θ_1 and θ_2 so input u_1 (resp. u_2) controls θ_1 (resp. θ_2). Those angles are defined as follow:

$$\begin{aligned} \theta_1 &= \pi + q_1 - q_3 \\ \theta_2 &= \pi + q_2 - q_3 \end{aligned} \tag{41}$$

Then, the new dynamics system controlled by **u** can be written as follow:

$$\mathbf{M}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}} + \mathbf{G}(\mathbf{q}) = \mathbf{B}\mathbf{u} \tag{42}$$

The matrix $\mathbf{B}$ can be computed thanks to virtual work considerations where:

$$\begin{aligned}\delta W &= \delta\theta_1 u_1 + \delta\theta_2 u_2 \\ &= (\delta q_1 - \delta q_3)u_1 + (\delta q_2 - \delta q_3)u_2 \\ &= \delta q_1 u_1 + \delta q_2 u_2 - \delta q_3(u_1 + u_2) \\ &= \delta x_g \times 0 + \delta z_g \times 0 + \delta q_1 u_1 + \delta q_2 u_2 - \delta q_3(u_1 + u_2) \\ &= \mathbf{F}.\delta\mathbf{q} \\ &= \mathbf{B}[u_1, u_2]^T.[\delta x_g, \delta z_g, \delta q_1, \delta q_2, \delta q_3]^T\end{aligned} \tag{43}$$

Therefore, matrix $\mathbf{B} = \begin{pmatrix} 0 & 0 \\ 0 & 0 \\ 1 & 0 \\ 0 & 1 \\ -1 & -1 \end{pmatrix}$ with $\mathbf{q} = [x_g, z_g, q_1, q_2, q_3]^T$.

3.2.4 Ground reaction force

The final step of the modelling is to define the ground reaction force so that the robot can stand up. The idea is to apply forces on the feet of the robot to avoid it going through the ground. The main difficulty here is to find a good force function shape. Before that, the first step is to be able to apply an external force on the system. In order to do so, the Jacobian of the system has to be defined and computed. We would like to be able to apply force vectors on hip, top and feet of the robot. The Jacobian matrix $\mathbf{J}$ is defined as follow using the kinematic equations:

$$\mathbf{J} = \begin{pmatrix} \frac{\partial x_h}{\partial x_g} & \frac{\partial x_h}{\partial z_g} & \frac{\partial x_h}{\partial q_1} & \frac{\partial x_h}{\partial q_2} & \frac{\partial x_h}{\partial q_3} \\ \frac{\partial z_h}{\partial x_g} & \frac{\partial z_h}{\partial z_g} & \frac{\partial z_h}{\partial q_1} & \frac{\partial z_h}{\partial q_2} & \frac{\partial z_h}{\partial q_3} \\ \frac{\partial x_{l1}}{\partial x_g} & \frac{\partial x_{l1}}{\partial z_g} & \frac{\partial x_{l1}}{\partial q_1} & \frac{\partial x_{l1}}{\partial q_2} & \frac{\partial x_{l1}}{\partial q_3} \\ \frac{\partial z_{l1}}{\partial x_g} & \frac{\partial z_{l1}}{\partial z_g} & \frac{\partial z_{l1}}{\partial q_1} & \frac{\partial z_{l1}}{\partial q_2} & \frac{\partial z_{l1}}{\partial q_3} \\ \frac{\partial x_{l2}}{\partial x_g} & \frac{\partial x_{l2}}{\partial z_g} & \frac{\partial x_{l2}}{\partial q_1} & \frac{\partial x_{l2}}{\partial q_2} & \frac{\partial x_{l2}}{\partial q_3} \\ \frac{\partial z_{l2}}{\partial x_g} & \frac{\partial z_{l2}}{\partial z_g} & \frac{\partial z_{l2}}{\partial q_1} & \frac{\partial z_{l2}}{\partial q_2} & \frac{\partial z_{l2}}{\partial q_3} \\ \frac{\partial x_t}{\partial x_g} & \frac{\partial x_t}{\partial z_g} & \frac{\partial x_t}{\partial q_1} & \frac{\partial x_t}{\partial q_2} & \frac{\partial x_t}{\partial q_3} \\ \frac{\partial z_t}{\partial x_g} & \frac{\partial z_t}{\partial z_g} & \frac{\partial z_t}{\partial q_1} & \frac{\partial z_t}{\partial q_2} & \frac{\partial z_t}{\partial q_3} \end{pmatrix}$$

This matrix is then used in the dynamics equations as follow:

$$\mathbf{M}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}} + \mathbf{G}(\mathbf{q}) = \mathbf{J}^T.[\mathbf{f}_h, \mathbf{f}_{l1}, \mathbf{f}_{l2}, \mathbf{f}_t]^T + \mathbf{B}\mathbf{u} \tag{44}$$

Now that we know how forces can be applied on foot, forces $\mathbf{f}_{l1}$ and $\mathbf{f}_{l2}$ have to be defined so ground reaction forces can be modeled. In this section, the ground is assumed at altitude $z_{ground} = 0$.

The main constraint is to define a function which only applies force when the foot point is close or below the ground horizon in the z axis to avoid it crossing the terrain. It can be seen as a spring force depending on the penetration distance of the foot in the ground. It forces the foot to go up and to stabilize at a given altitude. A damper in the z axis can be added to smooth the movement so big oscillations can be avoided.

Moreover, we have to avoid the foot to slip. As a consequence, a force in the x axis has to be applied. It can not be a spring force because it has to oppose the foot motion. A damper sounds like the best option in this situation.

Finally, remark that the spring and dampers only have to apply forces when the foot is extremely close to the ground. Therefore, the spring and damper constants should depend on the z position of the foot. Thus, if z is high enough, the constants should reach 0 so no forces are applied. With all these considerations, the ground reaction force is considered with the following shape:

$$\mathbf{f}_{ground}(x, \dot{x}, z, \dot{z}) = \begin{pmatrix} b_x(z)\dot{x} \\ k_z(z)z + b_z(z)\dot{z} \end{pmatrix} \tag{45}$$

Our first guess has been confirmed while reading at some scientific papers on the subject. We particularly refer to a paper from the ETH University [1]. The main idea is to use an exponential spring parameter and sigmoid based damper parameters to fit the requirements previously presented. The ground reaction force is then expressed as follow:

$$\mathbf{f}_{ground}(x, \dot{x}, z, \dot{z}) = \begin{pmatrix} -\frac{d_x}{1+e^{\alpha z}}\dot{x} \\ -ke^{-\beta z}z - \frac{d_z}{1+e^{\gamma z}}\dot{z} \end{pmatrix} \tag{46}$$

The main benefit of this shape is that the 2D damper is always active whatever the z position of the foot under the ground while no force is applied if the foot is above the terrain. Concerning the spring, the exponential parameter models the elastic property of the ground which quickly harden with the penetration distance. In addition, this force formulation facilitates the extraction process from the ground attraction field as the force decreases exponentially with z. However, because of the exponential shape of the spring parameter, it is possible that the force becomes extremely high during the simulation which can lead to odd results. As a consequence, the ground reaction force is saturated with minimum an maximum values.

In this project, we take the following ground reaction force model parameters:

$$k = 100 \ ; \ d_x = 10.000 \ ; \ d_z = 100.000 \ ; \ \alpha = \beta = \gamma = 1000$$

The d_x parameter is extremely high so no slip of the foot is possible. Indeed, a lower value of this parameter can lead to a slow slip of the foot. α, β and γ are set so the foot equilibrium points are near to $z \approx 0$.

The last step of this section is to apply the ground reaction model on each foot. Therefore, the following equations are drawn:

$$\begin{aligned} \mathbf{f}_{l1} &= \mathbf{f}_{ground}(x_{l1}, \dot{x}_{l1}, z_{l1}, \dot{z}_{l1}) \\ \mathbf{f}_{l2} &= \mathbf{f}_{ground}(x_{l2}, \dot{x}_{l2}, z_{l2}, \dot{z}_{l2}) \end{aligned} \tag{47}$$

3.2.5 Model Simulation and NMPC Controller Implementation

Now that the continuous model is fully described, it can be simulated the same way as the previous one. The dynamics is integrated using *ode45* Matlab function by considering the variable $\mathbf{x} = [\mathbf{q}, \dot{\mathbf{q}}]^T$ and the concatenated dynamics function $\dot{\mathbf{x}} = f_{dyn}(\mathbf{x})$ with:

$$f_{dyn}(\mathbf{x}) = \left[\dot{\mathbf{q}}, \mathbf{M}^{-1}\left(\mathbf{J}^T.[\mathbf{f}_h, \mathbf{f}_{l1}, \mathbf{f}_{l2}, \mathbf{f}_t]^T + \mathbf{B}.\mathbf{u} - \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}} - \mathbf{G}(\mathbf{q})\right)\right]^T \tag{48}$$

The system can now be simulated starting from initial coordinates $\mathbf{x}_0$. The continuous robot model behave as expected. The outcomes are quite impressive and look like natural and accurate. One frame of the simulation is shown in figure 18. Note that no input is applied in this simulation.

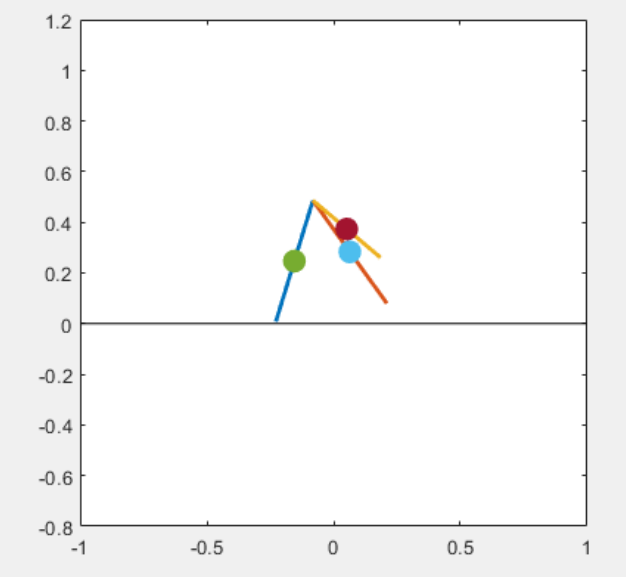

Figure 18: Continuous Model Simulation

Feet don't slip and don't go through the ground. There is no oscillation of the foot points either in the z axis which successfully stabilize close to $z \approx 0$. However, it seems that the force needed to extract from the ground attraction field is non negligible. The z damper prevents feet to gain altitude (when feet get a positive z velocity). Therefore, we change the shape of the ground reaction force so the z damper only applies force if the z velocity is negative. The new ground reaction force is written below:

$$\mathbf{f}_{ground}(x, \dot{x}, z, \dot{z}) = \begin{pmatrix} -\frac{d_x}{1+e^{\alpha z}}\dot{x} \\ -ke^{-\beta z}z - \frac{d_z}{1+e^{\gamma z}}min(\dot{z}, 0) \end{pmatrix} \tag{49}$$

All in all, the ground reaction force model can be validated and is considered in the following sections of this report.

Now that the model is well defined, it is time to implement our controller. We use the *nlmpc* function from the Matlab Model Predictive Control Toolbox which is well suited for control of constrained non linear systems while minimizing an objective function. The basis of an NMPC controller is to predict and take into consideration the future so it can adapt the control inputs. We will not go deep into the controller considerations, see section 3.4 for further information.

Unfortunately, the controller can not achieve great walking gaits. The controller is stuck before one foot step. The issue comes from the ground reaction force model. The swing leg can only switch side when $q_1 \approx q_2 \approx 0$. As a consequence, legs are collinear resulting in feet that are at the same altitude. Then, the same ground reaction force is applied which sticks

the swing leg (especially because of the x damper which prevents slipping) on the ground so the robot can not achieve one step. The ground reaction model is then not well adapted for such a simple biped walker model. Further adaptations are required. However, the controller still tries to go as far as possible without swinging legs. Therefore, one of the options to do so is to "jump" using the torso as a propeller. This is exactly the observed behavior of our controller's during simulation. Two simulation frames are shown in figures 19 and 20.

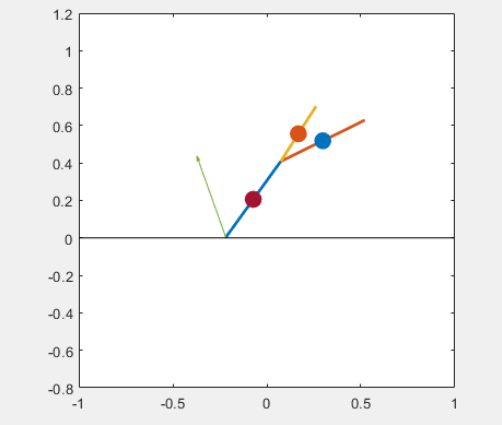

Figure 19: NMPC Controller Simulation - Frame 1

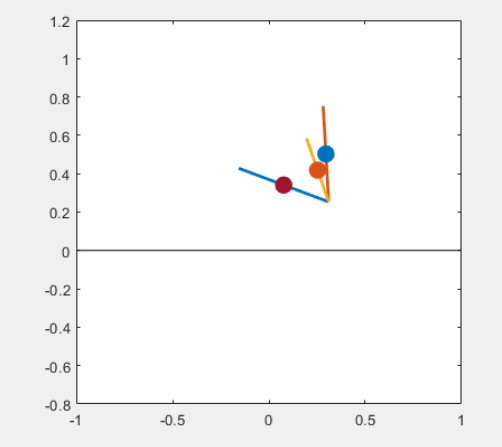

Figure 20: NMPC Controller Simulation - Frame 2

One other option we could expect (if no joint angle constraints are imposed) is to swing the leg with a clockwise movement even if it is not an acceptable walking gate. We find here one of the main limitations of NMPC. As explained before, an MPC controller forecasts the dynamics in the future for a given "horizon". Here, the horizon is not big enough so the controller can not simulate a complete or multiple steps. As a consequence, it doesn't seek for a final state which enables to perform another step. It doesn't seek for pattern reproducibility. Then, the controller only tries to go as far as possible before the swing leg

hits the ground. This explains in part the first jump outcome. In order to counteract this limitation, the horizon could be increased so that the controller can seek for pattern reproducibility. However, this is extremely heavy computation which means very long simulations.

Therefore, our best option is to enable the robot to swing legs as a natural walker. One possible consideration is that the leg length can be controlled and changed online so the swing leg can switch side without being influenced by the ground reaction force. This consideration is studied in the next section 3.3.

3.3 Variable Leg Length

In this section, we want to enable the controller to change leg lengths. One quick option is to set the leg length directly with the input $\mathbf{u}$ without any addition in the dynamics. This solution is not physically admissible as it is not even possible. One better option could be to add an independent dynamics for the leg extension. A control force to the leg extension via input $\mathbf{u}$ could then be added. However, more simply, we can "simulate" the leg extension dynamics by imposing a change rate of leg lengths to the controller.

As a consequence, the input of the system is now $\mathbf{u} = [u_1, u_2, l_{1c}, l_{2c}]^T$. Parameters l_{1c} and l_{2c} define the new variable leg lengths. A strong assumption is made here, we assume that those parameters don't affect the mass points coordinates. In other terms, the leg extension physical parts are assumed with no mass. Foot positions can be rewritten as follow:

$$\begin{aligned}[x_{l1}, z_{l1}]^T(\mathbf{q}) &= [x_g - \Delta_x(q_1, q_2, q_3) - l_{1c}sin(q_1), z_g - \Delta_z(q_1, q_2, q_3) - l_{1c}cos(q_1)]^T \\ [x_{l2}, z_{l2}]^T(\mathbf{q}) &= [x_g - \Delta_x(q_1, q_2, q_3) - l_{2c}sin(q_2), z_g - \Delta_z(q_1, q_2, q_3) - l_{2c}cos(q_2)]^T\end{aligned} \tag{50}$$

Moreover, this small change of coordinates entails a change of the Jacobian. The latter depends now on l_{1c} and l_{2c}. However, matrices $\mathbf{M}$, $\mathbf{C}$ and $\mathbf{G}$ remain the same. Remark that the additional components of input $\mathbf{u}$ are not used in the dynamics equation 48. So matrix $\mathbf{B}$ don't change either.

The NMPC controller can be tested on this new model. We can quickly notice a problem with this model, legs can extend (if previously retracted) in the ground which entails a huge reaction force following the exponential ground reaction force model. Therefore, the robot can abnormally jump by taking advantage of this feature. This behavior is not expected and should be avoided. One way to counteract this issue is to set a maximal extension rate to the legs so the leg penetration due to leg extension mechanism is below a given value to bound this jump effect. Nevertheless, this really limits the interest of the leg extension as more time may be needed to fully extend or retract a leg.

Therefore, this new model is not that good because of the strange jump behavior while the controller doesn't succeed to take advantage of the leg extension mechanism. We once more got odd results. Our robot resembled more to superman than to a walker robot because it took advantage of this jump behavior to take off.

To sum up, some adjustments are still needed. Multiple options can be drawn:

- One option is to add a force on feet to model the reaction force due to the leg extension. It could enables feet to go up a bit so legs can extend without having such a huge reaction force.
- Another option is to define an additional dynamics model of the leg extension mechanism. The control of this system could be simpler and more natural for the controller.
- A final option which can be considered is a change in the robot structure. Knee could be added to the structure to replace the leg extension mechanism. This model would be more accurate and easier to control but extensive reworking of the system is expected.

However, we didn't get time to go deeper into this controller and this continuous model of the robot. The outcomes look uncertain but encouraging at this level. That's why we decided to go back to a discontinuous model and to change our controller into a virtual model based controller which is presented in the previous section. More time is needed to test and validate new physical models from which the NMPC controller can take advantage of.

3.4 NMPC Controller Information

In this section, few considerations about the NMPC controller are explained. The *nlmpc* function from the Matlab Model Predictive Control Toolbox is used. It is a hand in solution for doing NMPC.

The dynamic function of the system with a simulation step and a given horizon is specified to the solver and some constraints on inputs and state variables are imposed. We first bound inputs $\mathbf{u}_1$ and $\mathbf{u}_2$ to saturate the torques. In addition, the leg length $\mathbf{l}_{1c}$ (resp. $\mathbf{l}_{2c}$) is bounded between $r_{red}l_1$ (resp. $r_{red}l_2$) and l_1 (resp. l_2) with $r_{red} \in [0, 1]$. This value is set to $r_{red} = 0.8$ in our project. Furthermore, constraints on state variables can be imposed as well. For instance, we can set an available range of angles for q_1, q_2 and q_3 while imposing the robot to stand up thanks to a constraint on z_g.

The next step of the configuration of the NMPC controller is to define a cost function. This is the quantity the solver tries to minimize which is useful to define an expected behavior. The cost function returns the sum of multiple parts:

- The first one is $-w_g x_g$ so the solver makes the robot walking in the right direction.
- The second part is $-w_l(l_{1c} + l_{2c})$ so the solver only retracts legs if needed.
- An additional one is $w_q q_3^2$ so the torso is kept vertical as much as possible.
- The final part of the cost function is $w_u(u_1^2 + u_2^2)$ so only minimum torques necessary are applied.
- We could add to this cost function the following part $w_v(\dot{x}_h - \dot{x}_{target})^2$ to track a reference velocity. However, this part is not yet implemented in our controller.

w_g, w_l, w_q, w_u and w_v are weighting factors so the behavior of the controller can be easily tuned.

Finally, initial conditions are imposed. The solver can iterate one simulation step further using the *nlmpcmove* function to find the optimal input $\mathbf{u}_{opt}$ to apply on the system so we respect the behavior translated into the cost function.

3.5 Discussion about NMPC

NMPC is a very interesting controller when dealing with motion optimization. (for example, it is used for foot planning by ANYmals [1]). In theory, NMPC can achieve incredible results by perfectly optimizing robot motion and footholds while fitting constraints so it can avoid obstacles and holes. Moreover, NMPC gets good results when facing disturbances. However, we faced multiple issues when dealing with NMPC in this project.

First of all, a more complex physical model is needed to simulate the robot. Moreover, to achieve those prediction outcomes, a long horizon (so it can forecast multiple steps) has to be set which entails very long simulations. Indeed, NMPC in our project was under very heavy computation stresses despite the simplicity of the model considered. As it is, the NMPC controller can not be used online on a real robot. It is right that some "linear explicit MPC" exist so it can be ran online. However, more work is needed to verify if the current NMPC controller can be translated into an "explicit NMPC" controller. In addition, the outcomes of the controller definitely depends on the accuracy of the physical continuous model considered.

Finally, one more thing that we noticed during the project is that because of the ground reaction force shape, computations are probably much slower. For instance, we first used a custom Runge Kutta 4 integrator to predict the next step of the system. We quickly remarked that this integrator is not accurate enough when considering such a dynamic ground reaction force. Therefore, we let *nlmpc* find the right integrator. Nevertheless, it is highly probable that non negligible amount of time is wasted to find the right integrator. It could be interesting to impose a high degree one to *nlmpc* to save time.

References

[1] M. Neunert et al. "Whole-Body Nonlinear Model Predictive Control Through Contacts for Quadrupeds". In: *IEEE Robotics and Automation Letters* 3.3 (2018), pp. 1458–1465. DOI: 10.1109/LRA.2018.2800124.