Humanoid Walking with Whole Body Control

Ntagkas Alexandros

July 16, 2024

Contents

1 Introduction

Humanoid walking with whole body control involves the coordinated management of all joints and limbs to achieve a stable walking motion. The main parts of such a task are:

- Footstep generation
- Center of Mass(CoM) trajectory Planning
- Foot trajectory planning
- Quadratic Programming (QP) Controller to track the above trajectories.
- Finite State Machine (FSM) to coordinate between the different states of the movement.

I will be using **RobotDART** [1] for dynamic simulation and **pinocchio** [2] for the computation of the robot dynamics and kinematics.

2 RobotDART-Pinocchio Compatibility

RobotDART and pinocchio have some incompatibilities. The orientation in RobotDART for the floating base is expressed in axis-angle while the orientation in pinocchio in quaternions.

$$q_{pin} = \begin{bmatrix} x_{float} \\ y_{float} \\ z_{float} \\ q_x \\ q_y \\ q_z \\ q_w \\ joint_1 \\ \dots \\ joint_n \end{bmatrix}, q_{dart} = \begin{bmatrix} a_x\theta \\ a_y\theta \\ a_z\theta \\ x_{float} \\ y_{float} \\ z_{float} \\ joint_1 \\ \dots \\ joint_n \end{bmatrix}$$

the conversion from axis-angle to quaternions can be done as follows:

$$\theta = \sqrt{a_x^2 + a_y^2 + a_z^2}$$

if $\theta = 0$ we get the unit quaternion $q_x = q_y = q_z = 0, q_w = 1$, else

$$q_w = cos(\frac{\theta}{2}), q_x = a_x sin(\frac{\theta}{2}), q_y = a_y sin(\frac{\theta}{2}), q_z = a_z sin(\frac{\theta}{2})$$

3 Footstep Generation

With generate_footsteps(distance, step_length, foot_spread) I can produce footsteps with the desired characteristics.

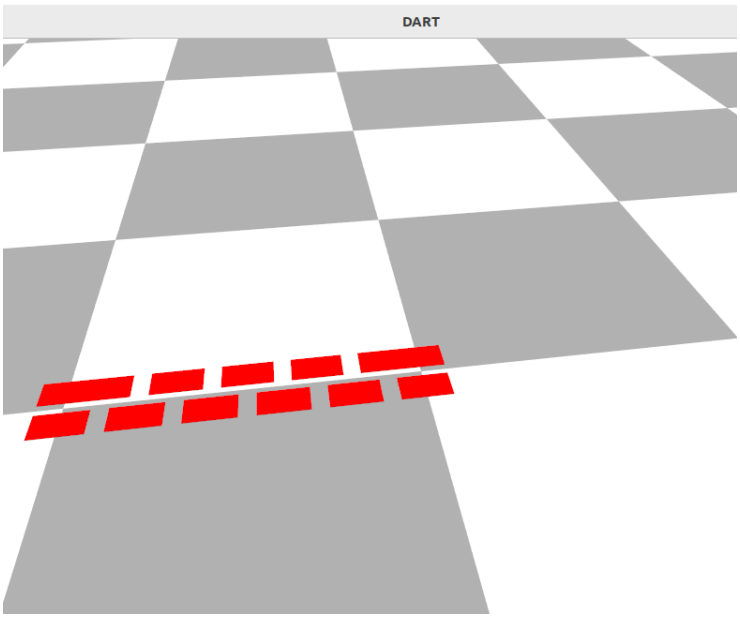


Figure 1: d=1 , lenth=0.1, spread =0.07

4 Finite State Machine

The FSM for a humanoid walking algorithm is one of the most important aspects. I will be using something similar to this:

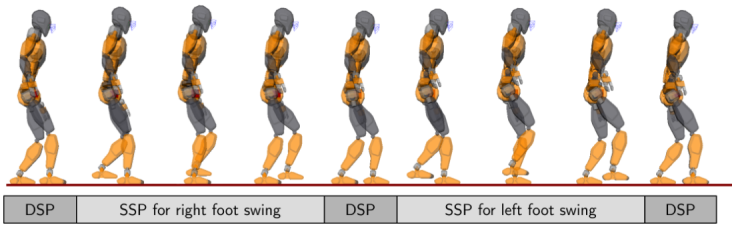


Figure 2: FSM

DSP is a double support phase where both feet stay on the ground. For me this will be shifting the CoM to the opposite side of the next step's side.

SSP is the single support phase where we must lift the respective leg and move it to the next footstep position , retaining the robot's balance in the process.

In addition to the foot movement we have to move the CoM as well, so that it always stays inside the support polygon shown below. This is essential to keep the robot stable throughout the movement.

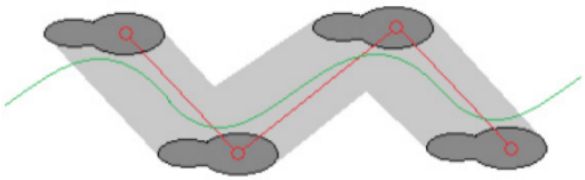


Figure 3: Support Polygon in grey

The timings and durations of each state will be discussed in the results section.

5 Trajectories

5.1 Cubic Hermite Splines

Cubic Hermite splines are a type of spline where each piece is a third-degree polynomial specified in Hermite form. The spline ensures continuity of both the function and its first derivative.

On the unit interval $[0, 1]$, given a starting point $\mathbf{p}_0$ at $t = 0$ and an ending point $\mathbf{p}_1$ at $t = 1$ with starting tangent $\mathbf{m}_0$ at $t = 0$ and ending tangent $\mathbf{m}_1$ at $t = 1$, the polynomial can be defined by

$$H(t) = h_{00}(t)p_0 + h_{10}(t)m_0 + h_{01}(t)p_1 + h_{11}(t)m_1,$$

where

$$\begin{aligned} h_{00}(t) &= 2t^3 - 3t^2 + 1, \\ h_{10}(t) &= t^3 - 2t^2 + t, \\ h_{01}(t) &= -2t^3 + 3t^2, \\ h_{11}(t) &= t^3 - t^2. \end{aligned}$$

Because we do not want to always work in the unit interval we can time-scale the above at any interval T we want , by mapping $[0, T]$ to $[0, 1]$ This will be done setting $t = \frac{t}{T}$ and $m_0 = m_0 T$, $m_1 = m_1 T$

5.2 CoM Trajectory

In all of the movements the CoM will have zero velocity at start and finish. The start point p_0 will always be the current point of the CoM which we can compute with *pin.centerOfMass*. The final point p_1 will be the desired point which can vary based on the state of the robot (SSP or DSP).

On SSP, as we see below, we have to move the CoM towards the landing stop of the opposite foot.

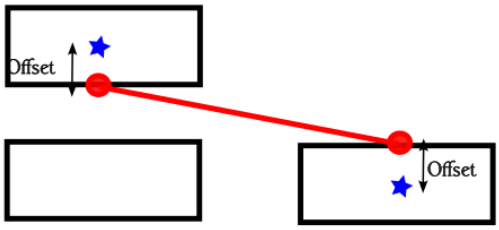


Figure 4: SSP CoM Trajectory

However, driving the leg exactly to the **center of the foot (blue star),** will create a big movement, which is undesirable because the robot may lose stability easily. To avoid that we introduce an **offset** that places the CoM desired position more centrally.

On DSP the CoM has to remain on the offset point and wait for the next step.

Now that we have p_0 and p_1 for each case we can create 3 cubic hermite splines one for each axis (x,y,z) that describe the movement.

5.3 Foot Trajectory

The foot has to move only on SSP. The starting point will be the current position of the foot calculated by the forward Kinematics and the desired position will be the next footsteps placement. However, we need to add one more point, in order to lift the foot of the ground. Suppose T_{SSP} is the duration of SSP. Then we need

$$p(\frac{T_{SSP}}{2}) = \begin{bmatrix} \frac{p_{0x}+p_{1x}}{2} \\ \frac{p_{0y}+p_{1y}}{2} \\ H \end{bmatrix}$$

where H is the midpoint height that we choose.

So we will create 2 cubic-hermite splines:

- From p_0 to p_m with velocities $v_0 = 0$ and v_m
- From p_m to p_1 with velocities v_m and $v_1 = 0$

To get the trajectory point at a given time t, we first determine whether t falls in the first or second segment. Then we evaluate the appropriate spline.

For $t \leq 0.5T_{\text{SSP}}$:

$$\text{Use the spline from } p_0 \text{ to } p_m : \quad p_1(t)$$

For $t > 0.5T_{\text{SSP}}$:

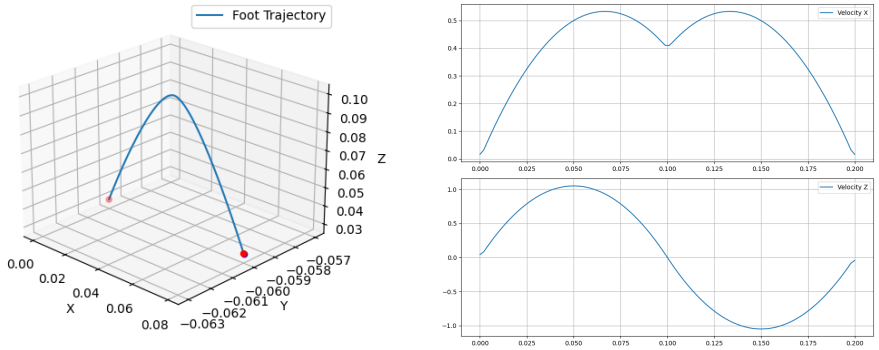
$$\text{Use the spline from } p_m \text{ to } p_1 : \quad p_2(t - 0.5T_{\text{SSP}})$$

5.3.1 Finite Difference

The choice of v_m can be done with finite differences :

$$v_m = \frac{p_1 - p_0}{T_{SSP}}$$

This produces the following trajectories.



(a) Trajectory 3D

(b) Velocities

We can see that the velocities and the trajectory are continuous as expected however at $T_{SSP}/2$ the acceleration on the x-axis is not discontinuous. This is not a serious issue since our controller can still follow the trajectory.

5.3.2 Continuous Acceleration

To avoid the discontinuous acceleration and not choice a fifth degree polynomial we can find the value v_m^* that creates a continuous acceleration profile.

We have the equations for the splines:

$$p_1(t) = h_{00}p_0 + h_{01}p_m + h_{11}Tu_m$$
$$p_2(t) = h_{00}p_m + h_{10}Tv_m + h_{01}p_1$$

and the second derivatives:

$$\ddot{h_{00}}(t) = 12t - 6,$$
$$\ddot{h_{10}}(t) = 6t - 4,$$
$$\ddot{h_{01}}(t) = -12t + 6,$$
$$\ddot{h_{11}}(t) = 6t - 2$$

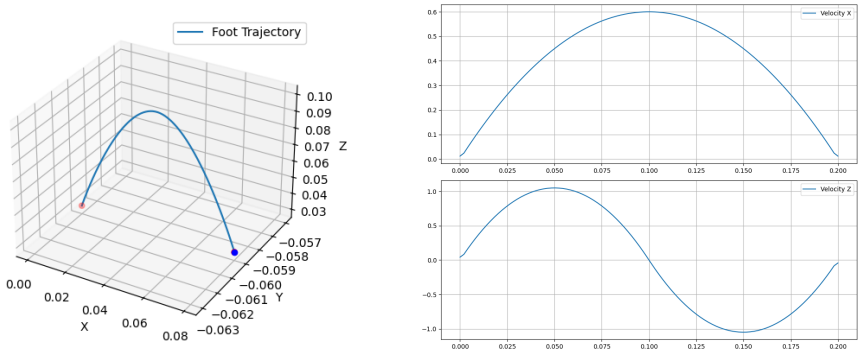
So we can calculate the derivatives of the splines $\ddot{p_1}(t), \ddot{p_2}(t)$.

To have constant acceleration we must have $\ddot{p_1}(t) = \ddot{p_2}(t)$ at the point of change of splines. This point is $T = \frac{T_{SSP}}{2} \Rightarrow \ddot{p_1}(T) = \ddot{p_2}(0)$

And we calculate the desired velocity

$$v_m^* = \frac{6}{8T}(p_1 - p_0)$$

This produces the following trajectories.



(a) Trajectory 3D

(b) Velocities

We can see that now we have constant acceleration.

6 QP controller

Now that we have the trajectories for our points of interest we need a way to track them.

I will be using an optimization based controller which uses the proxsuite optimization library and calculate the desired torques τ that minimize the error to the current trajectory point.

Another advantage of the QP controller is that we can introduce constrains for our physical system that have nothing to do with the task at hand but improve the quality of the solutions over all.

6.1 Friction Cones

Such a constraint are friction cone constrains. On each leg we place 4 contact frames.

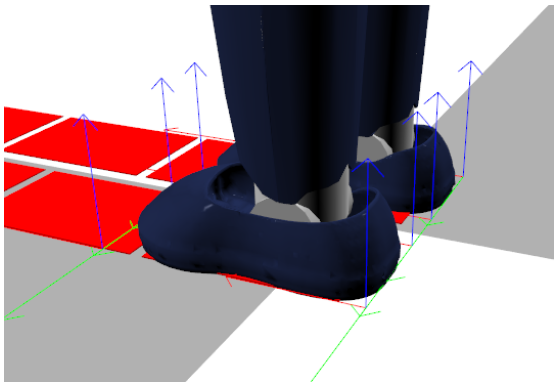


Figure 7: Contacts

For each contact we have a force f being applied. We split the force

$$f =< f_n, f_t, f_b >$$

in normal and tangental components.

The friction constrains imply that that force cannot pull/grasp the ground so it must point upwards:$f \cdot n > 0$ and satisfy 2 conditions to avoid slip:

$$||\mathbf{f} \cdot \mathbf{t}||_2 \leq \mu(\mathbf{f} \cdot \mathbf{n})$$
$$||\mathbf{f} \cdot \mathbf{b}||_2 \leq \mu(\mathbf{f} \cdot \mathbf{n})$$

with μ the Coulomb static friction coefficient of the contact/ground.

To use these constrains in QP they have to be linear so we approximate them like so :

$$|\mathbf{f} \cdot \mathbf{t}| \leq \tilde{\mu}(\mathbf{f} \cdot \mathbf{n})$$
$$|\mathbf{f} \cdot \mathbf{b}| \leq \tilde{\mu}(\mathbf{f} \cdot \mathbf{n})$$

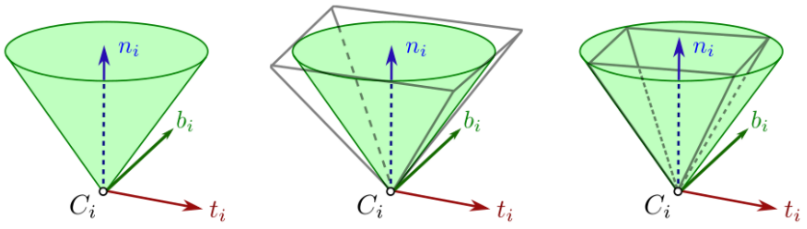


Figure 8: Cones

if $\tilde{\mu} = \mu$ we get the more relaxed friction cones(outer) and if $\tilde{\mu} = \frac{\mu}{\sqrt{2}}$ we get the inner planes.

6.2 QP formulation

$$\begin{aligned} \min_x f(x) &= \frac{1}{2} x^T Q x + q^T x \\ \text{s.t. } & Ax - b = 0 \\ & Cx - d \leq 0 \end{aligned} \tag{1}$$

$$\text{where } x, q \in \mathcal{R}^N,\ Q \succeq 0 \in \mathcal{R}^{N \times N}.$$

$$x = \begin{bmatrix} \dot{\nu} \\ u \\ f_1 \\ \vdots \\ f_{\text{Ncontact}} \end{bmatrix},\ A = \begin{bmatrix} M(q) & -S & -J_1(q)^T & \cdots & -J_{\text{Ncontact}}(q)^T \end{bmatrix},$$

$$b = -C(q, \nu)$$

- Multiple tasks are combined with a weighted sum: $\sum_{i=0}^{N_{\text{tasks}}} w_i ||W_i x - t_i||^2$
- Practically, we define $W = \begin{bmatrix} w_1 W_1 \\ \vdots \\ w_{N_{\text{tasks}}} W_{N_{\text{tasks}}} \end{bmatrix}$ and $t = \begin{bmatrix} w_1 t_1 \\ \vdots \\ w_{N_{\text{tasks}}} t_{N_{\text{tasks}}} \end{bmatrix}$
- We set $Q = W^T W + r$ and $q = -W^T t$ where r is a small regularization term
- C, d are referring to the friction cone constrains

6.3 Tasks

A task can be described by the pair (W, t). For every end effector we have:

$$\dot{\mathbf{p}} = \mathbf{J}(q)\mathbf{v} \quad \Longrightarrow \quad \ddot{\mathbf{p}} = \mathbf{J}(q)\dot{\mathbf{v}} + \dot{\mathbf{J}}(q)\mathbf{v}$$

Suppose we have a trajectory for this end effector. At time t we want it to have an acceleration $\ddot{p_d}$ and thus the task can be:

$$||\mathbf{J}(q)\dot{\mathbf{v}} - (\ddot{\mathbf{p}}_d - \dot{\mathbf{J}}(q)\mathbf{v})||^2$$

so we get:

$$\mathbf{W}_i = \begin{bmatrix} \mathbf{J}(q) & \mathbf{0} & \mathbf{0} \end{bmatrix}$$

and

$$\mathbf{t}_i = \ddot{\mathbf{p}}_d - \dot{\mathbf{J}}(q)\mathbf{v}$$

The desired acceleration given a trajectory can be computed as:

$$\ddot{p_d} = K_p X_e - K_d \mathcal{V}$$

where $X_e \in SE(3)$,$X_e = \begin{bmatrix} t_d(t) - t_b(t) \\ log(R_d R_b^T) \end{bmatrix}$ and $\mathcal{V}$ is the spatial velocity

For some end effector we may want a specific orientation or translation not both . Thus we will modify the above:

- For translation use only the translation part of $J(q)$,$X_e = t_d(t) - t_b(t)$ and $\mathcal{V} = v$
- For orientation use only the orientation part of $J(q)$,$X_e = log(R_d R_b^T)$ and $\mathcal{V} = \omega$

6.4 Controller Variety

At the SSP we always have one foot off the ground so it is important to modify the QP formulation. That is removing the friction cone constrains for the lifted foot.

At the DDP we want both foot's friction constrains to apply.

To do this we will create 3 QP controllers: $controllerR$ where the constrains apply to the right leg, $controllerL$ where the constrains apply to the left leg,$controller$ where both constrains are active.

6.5 Results

We will set $T_{DSP} = 0.1$, $T_{SSP} = 0.1$, $offset = 0.05$, $com_z = 0.47$, $p_{mz} = 0.1$, $distance = 0.5$, $step\_spread = 0.06$, $step\_length = 0.08$

The tasks throughout the movement will be about

- A translation task for the CoM .
- An orientation task for the chest to keep the posture upright.
- A translation and orientation task for each leg, either to keep them in place in DSP or to move them in SSP.

The weights will be set as $2, 5, 1, 1$ for the CoM, $3, 5, 3, 1$ for the right leg and $3, 5, 1, 3$ for the left leg. The sequence of walking should look like this:

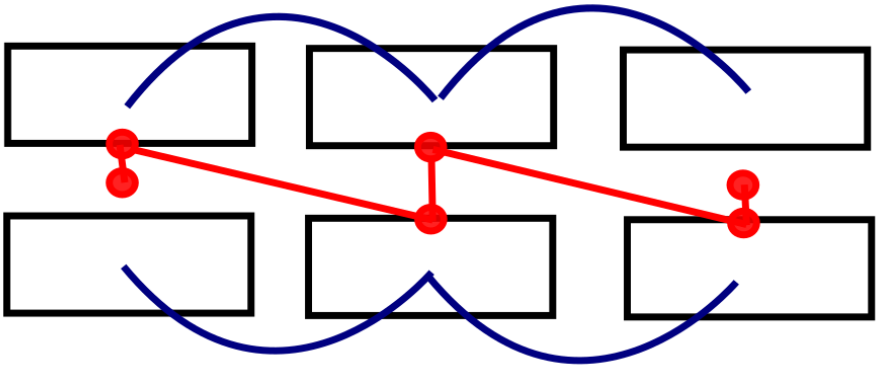


Figure 9: Sequence

The result can be seen in the relevant video.
The positions of the CoM and the legs can be seen below.

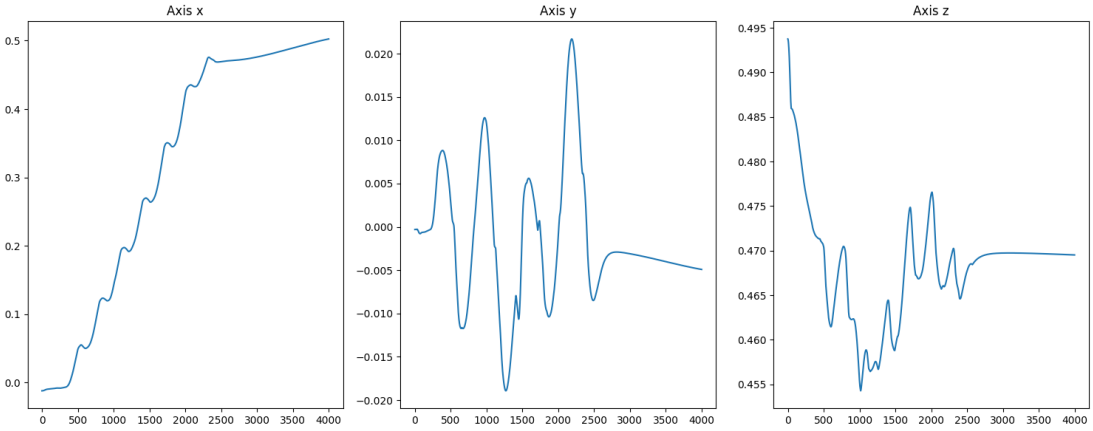


Figure 10: CoM actual Trajectory

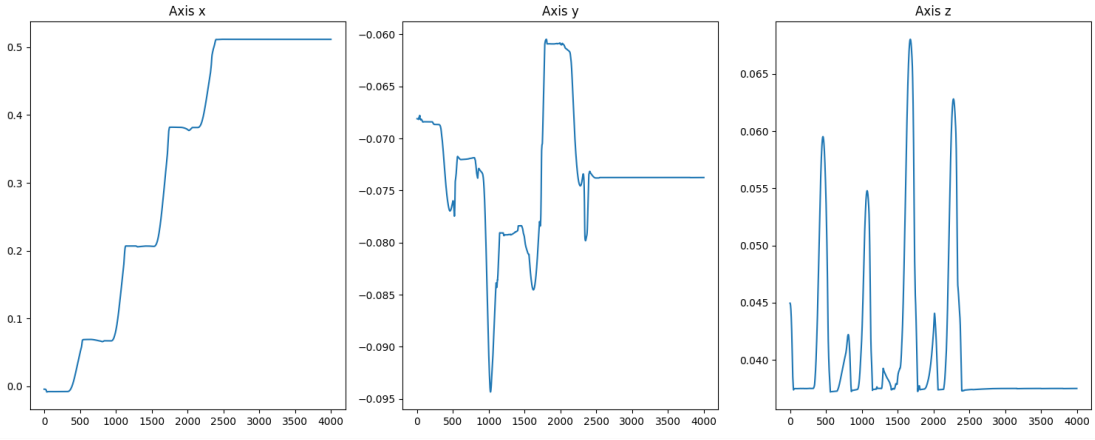


Figure 11: Right Leg actual Trajectory

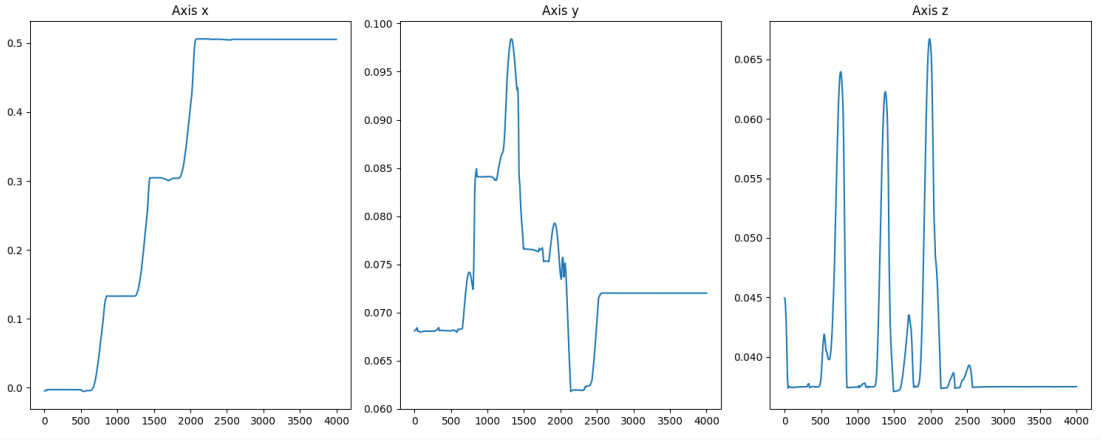


Figure 12: Left Leg actual Trajectory

We see that for the most part the trajectories are identical to the desired. The CoM swings around the y axis and has almost constant z-axis value.

The legs' x-axis has step increments like expected the y axis is pretty much constant and the z value has some spikes which are the lift phases.

7 Bibliography

References

[1] RobotDART: a versatile robot simulator for robotics and machine learning researchers, *Chatzilygeroudis, Konstantinos and Dionis, Totsila and Mouret, Jean-Baptiste*,2024

[2] Pinocchio: fast forward and inverse dynamics for rigid body models, *Carpentier, Justin and Saurel, Guilhem and Mirabel, Joseph and Lamiraux, Florent and Stasse, Olivier and Mansard, Nicolas*, 2019