Pages 1–5
2022

Assignment 01: Humanoid walking with TSID

Marco Perini 229203 ,Erik Mischiatti 233242
Advanced Optimization-Based Robot Contro
Masters Degree, Mechatronic Engineering
28/10/2022

Keywords: Assignment, ORC, Industrial Engineering

GOALS OF THIS ASSIGNMENT:

- Making practice using TSID to make a humanoid robot walk;
- Understanding the role of the task weights and learning how to tune them

CONTENTS

1 INTRODUCTION

To solve the initial problem with the LIPM's step we have to implement a third order interpolating function to generate the new foot trajectories to be used as reference trajectories in TSID.

$$x(t) = a + bt + ct^2 + dt^3 \tag{1}$$

So we have to define the initial and final points x_0 and x_1, described by the same third order function, and apply the velocity constraints to have a smooth transition with the following trajectory.

$$\begin{cases} x_0(t_0) = a + bt_0 + ct_0^2 + dt_0^3 = x_0 \\ x_1(t_1) = a + bt_1 + ct_1^2 + dt_1^3 = x_1 \\ x_0'(t_0) = 0 \\ x_1'(t_1) = 0 \end{cases} \tag{2}$$

Now we can write it in the **matrix form**. In our case [A] is always **invertible** because t_0 and t_1 never coincide each other:

$$[A]\{\text{coeff.}\}^\top = \{b\}^\top \to \{\text{coeff.}\}^\top = [A]^{-1}\{b\}^\top \tag{3}$$

so in the **explicit form:**

$$\begin{Bmatrix} a \\ b \\ c \\ d \end{Bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & T & T^2 & T^3 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 2T & 2T^2 \end{bmatrix}^{-1} \begin{Bmatrix} x_0 \\ x_1 \\ 0 \\ 0 \end{Bmatrix} \tag{4}$$

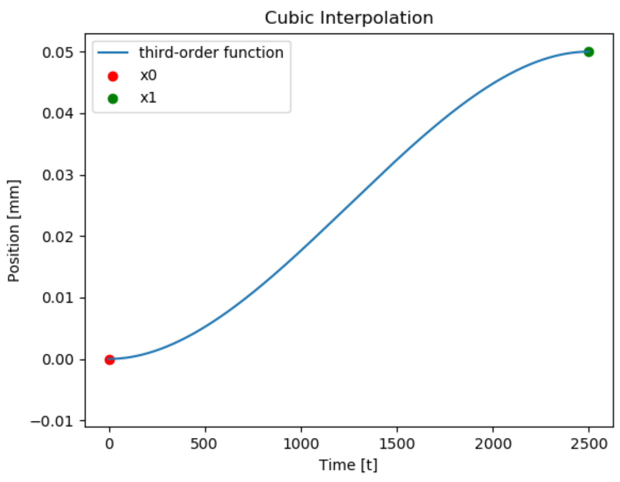

Figure 1: Third-order function plot

```
#Implemented code for the 3rd order interpolating
    function:
   def compute_3rd_order_poly_traj(x0, x1, T, dt):
    dim = x0.size #define the dimension of the input
    N = int(T/dt) #number of subdivisions
    A = np.array([[1, 0, 0, 0],
                 [1, T, T**2, T**3],
                 [0, 1, 0, 0],
                 [0, 1, 2*T, 3*T**2]])
    for s in range(dim):
      #b -> [P0, P1, P0_zero_vel, P1_zero_vel]
       b = np.array([x0[s], x1[s], 0, 0]).transpose()
      #compute coeff = A^-1*b
      #define computed coeff matrix as [a,b,c,d] x dim
       coeff[:,s] = np.dot(np.linalg.inv(A),b)

    x = np.empty((dim,N))*np.nan #pos
    dx = np.empty((dim,N))*np.nan #vel
    ddx = np.empty((dim,N))*np.nan #acc
    for i in range(dim):
      for j in range(N):
        #evaluate the 3rd order interp. function
        x[i,j]=coeff[0,i]+dt**j*coeff[1,i]+(dt*j)**2*
    coeff[2,i]+(dt*j)**3*coeff[3,i]
        dx[i,j]=coeff[1,i]+2*dt*j*coeff[2,i]+3*(dt*j)
    **2*coeff[3,i]
        ddx[i,j]=2*coeff[2,i]+6*dt*j*coeff[3,i]

  return x, dx, ddx

```

Listing 1: Python script

2 QUESTION 1:

- Run hw1_tsid_biped_walking.py with the default settings (SQUAT=0, PUSH=0 and default weights and gains). As you can notice, the humanoid falls down before completing the 6-steps walk. Tune the weights [w_com, w_foot, w_posture] to make the humanoid succeed in performing the whole walk. Describe how you tuned the weights and what the effects of such change are. Please refer to the plots generated by the code to argue your answer.

In order for the robot to perform the desired task, i.e. the six steps, we had to adjust the weights referring to the CoM, the centre of pressure and the robot's posture.The procedure applied was based on changing one parameter, specifically **w_foot**, while keeping the other two fixed.
Once the first satisfactory results were obtained, we focused on the other two, one at a time, repeating the same procedure, until we achieved this result:

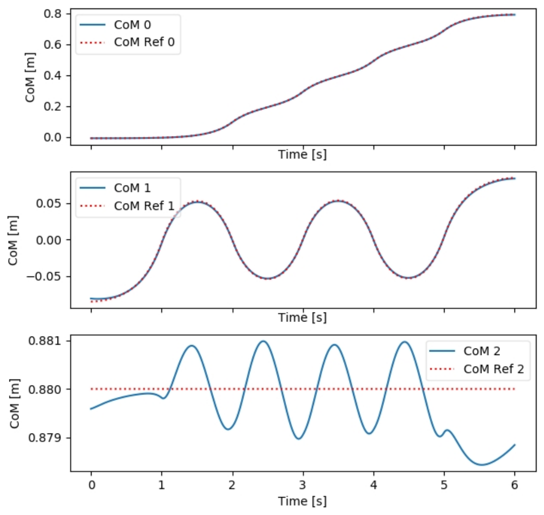

Figure 2: CoM Position

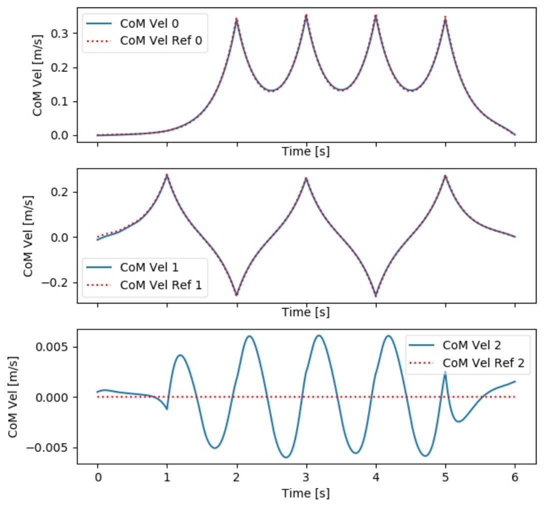

Figure 3: CoM Velocity

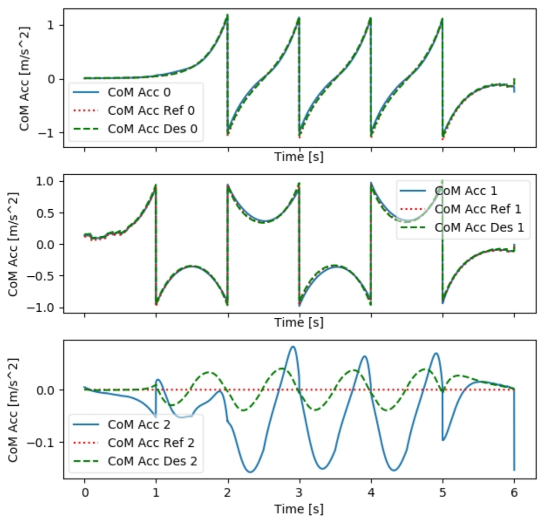

Figure 4: CoM Acceleration

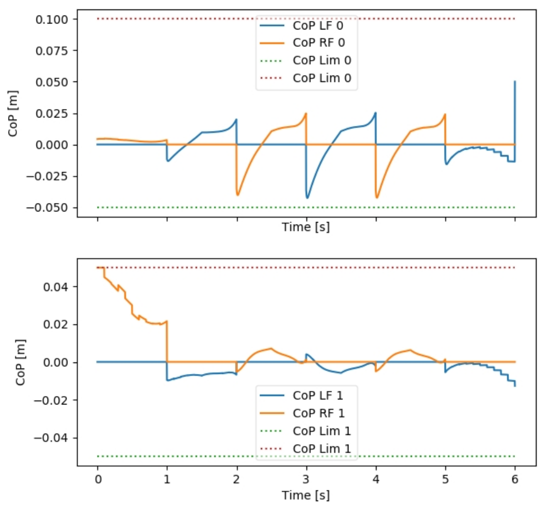

Figure 5: CoP plots

3 QUESTION 2:

- Set SQUAT=1 using the weights that you just tuned. Comment the plot that shows the tracking of the CoM reference trajectory (focus on what happens to the tracking of the z component). If we want the robot to squat more and have its CoM closer to the desired height we could either increase w_squat or kp_squat. Could you tell the difference (if any) between the two approaches? If you try them, do you notice any difference in the plots? If yes, what causes those differences?

Enabling the SQUAT parameter, the robot starts walking in squat position since the CoM is lower; the weights defined for the previous task are still good to be used.
Increasing kp_squat, the robot instantly follows the desired trajectory without delay because we are minimizing the error in position through a feedback loop. On the other hand, increasing w_squat makes the robot follow the desired trajectory in a incremental way; this is due to the importance of the weight in the cost function and, since there are multiple tasks, the TSID controller tries to take care of all the tasks making the process slower in stabilizing the z position.

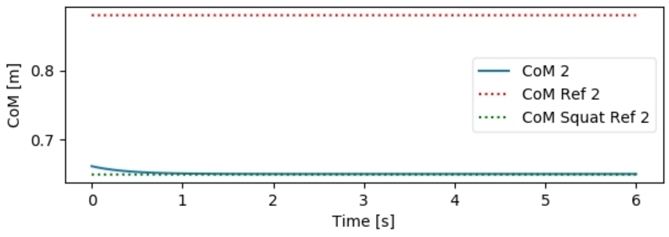

Figure 6: Z Axis: Position

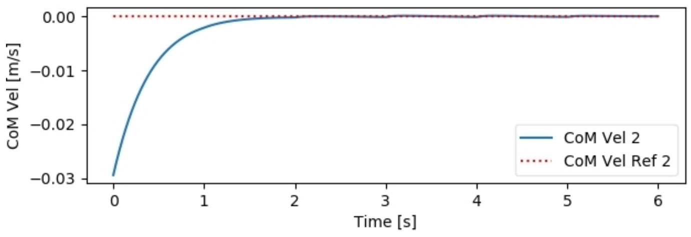

Figure 7: Z Axis: Velocity

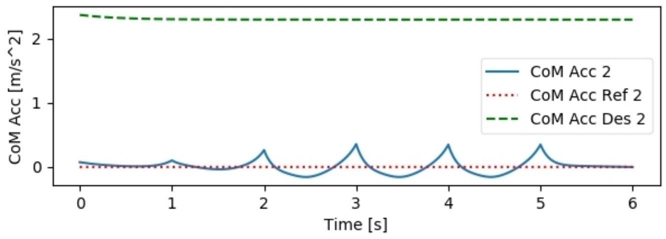

Figure 8: Z Axis: Acceleration

4 QUESTION 3:

- Set SQUAT=0 and PUSH=1 using your tuned weights. Does the robot fall down? If yes, what could you do to prevent it from doing it? Are there any drawbacks in doing what you suggested? If it does not fall, suppose it does and answer the previous questions.

The robot doesn't fall down with the previous weights when PUSH is enabled. We noticed that decreasing the weight of the center of mass, while keeping the others untouched, it falls. Therefore, increasing w_com could help with the walking. Increasing it too much would make the other weights less effective.
When PUSH is enabled kp and kd are initialized with higher values than the previous case. Kd helps more in compensating the push since it is proportional to the velocity.

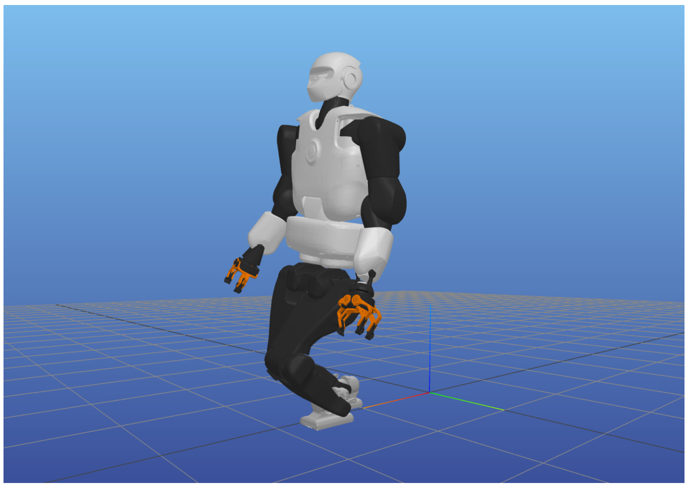

Figure 9: plots

5 QUESTION 4:

- Set SQUAT=1, PUSH=1 and push_robot_com_vel= [0, 0, -0.5]. Run two simulations, one with [w_squat=100, kp_squat=100] and another one with [w_squat=10, kp_squat=1000]. Comment on the differences that you notice and try to justify them.

We apply a vertical velocity at the center of mass after 3 seconds. Since the robot is walking and the feet are not in contact with the ground at the same time, the accelerations are not symmetric.
When the w_squat=100 and kp_squat=100, the robot keeps walking in squat position and after the push it recovers rapidly. As you can see in the Fig. [13], the foot position is negative because the controller doesn't see the ground as a physical object and the foot contact flag is temporary disabled to make the robot walk.
When the w_squat=10 and kp_squat=1000, we notice only a small difference in the acceleration of the joints, from the previous case, due to the fact that the proportional gain is higher. Usually, setting kp very high is not good because the robot becomes really stiff and, therefore, less compliant.

The upper plot referes to CASE 1: [w_squat = 100, kp_squat = 100]
The lower plot referes to CASE 2: [w_squat = 10, kp_squat = 1000]

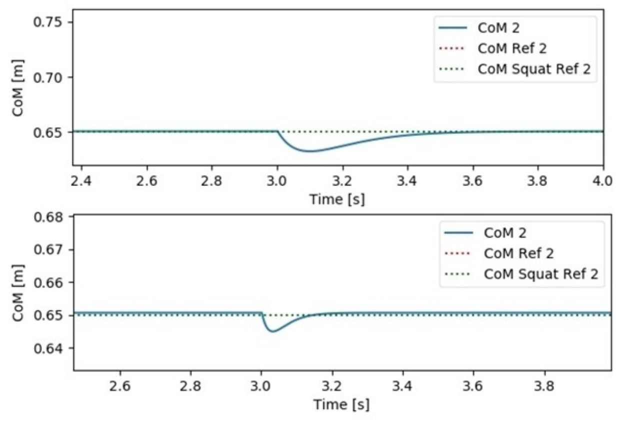

Figure 10: Z Axis: Position

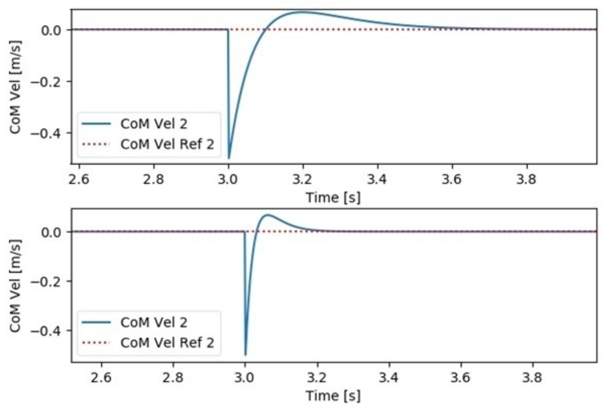

Figure 11: Z Axis: Velocity

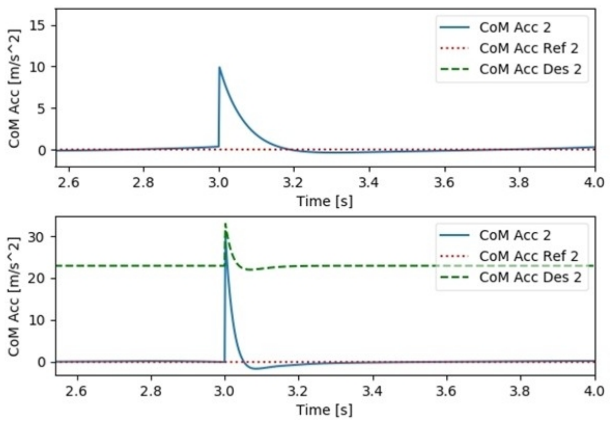

Figure 12: Z Axis: Acceleration

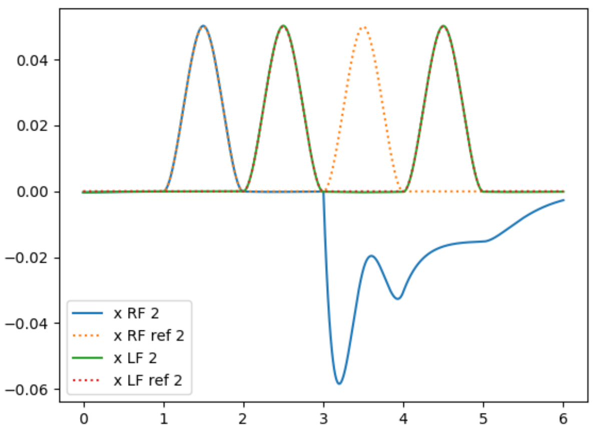

Figure 13: Tracking of the feet on Z Axis

6 EXTRA:

We discussed about some ways to fine tune the weights of the robot by just using the final errors in position, velocity and acceleration, "without knowing" the dynamics of the system. Since our objective was to find the values of the weights that minimized the errors, we decided to run many simulations (1000 times) using random weight values to see how the robot would have behaved.
In Fig.[14], the first hundred error samples are plotted and we can already notice some candidates that get close to zero.

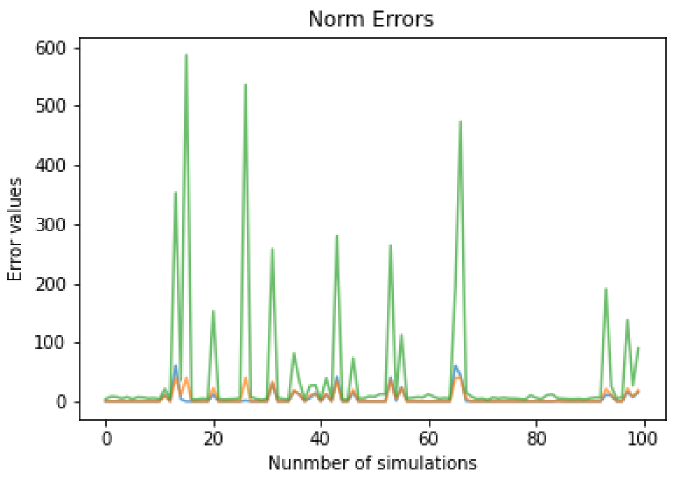

Figure 14

In FIg.[15], the distribution of the weights is plotted in a 3D space and the red points indicate weight combinations that lead to a norm error lower than 3.0.

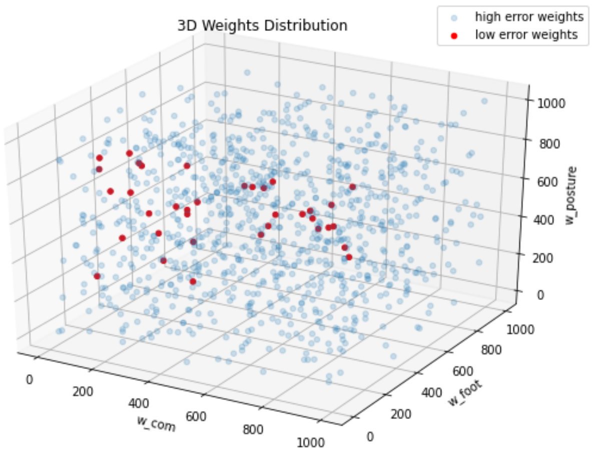

Figure 15

With this in mind, we opted to set up a simple **Multi-Layer Perceptron model (MLP)**, that would match the input and output with a nonlinear function that "describes" the system (similar to a system identification problem). The neural network is a sequential model with two dense hidden layers and three numbers as input and three as output (multi-input multi-output regression). No normalization was performed on the initial dataset (for example weights from 0 to 1) nor regularization (dropout layers etc)

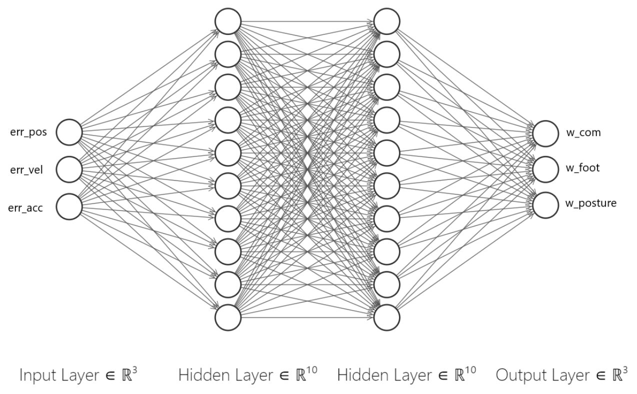

Figure 16

We trained the model for 200 epochs with a batch size equal to 3 and displayed in real time the loss function to check whether the results we were getting were good enough (the validation set is the 15% of the initial dataset).

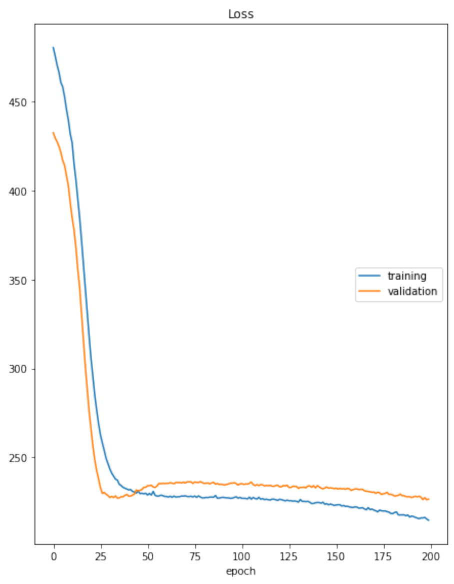

Figure 17

Since the NN was trained using only the final errors of the simulation and not the whole time window, we get some peculiar results when setting all the errors equal to zero during inference; the robot reaches the end of the simulation with low errors but with weird movement and not following the feet trajectories (it slides like a forward moonwalk). [1]

[1] Colab Notebook: `https://colab.research.google.com/drive/15RCFVscKQwH41kkYQc6iokhE2wHZUGqQ?usp=sharing`