UNIVERSITY OF TRENTO

Department of Industrial Engineering
Master's degree in Mechatronics Engineering

HUMANOID WALKING WITH TASK SPACE INVERSE DYNAMICS

First assignment report

Mattia Pettene 233944
Gabriele Poggesi 233220

Course of Optimization-based robot control

Academic Year 2022 - 2023

Introduction

In this assignment we are asked to practice on Task Space Inverse Dynamics (TSID) and understand the role of the task weights in order to tune them. The material provided allows us to control a simulated humanoid (Talos by PAL Robotics) and the main files we have worked on are the following three:

- `hw1_conf.py` contains all the parameters LIMP or TSID may need. In these parameters there are also the task weights and the proportional gains we will later tune;
- `hw1_LIPM_to_TSID_template.py` contains the functions that transform the LIPM trajectories in discrete functions;
- `hw1_tsid_biped_walking.py` contains the code to run the simulation of the humanoid walking.

The idea followed to simulate the behavior of the robot is based on two steps. The first is to run the `hw1_LIPM_to_TSID_template.py` in order to generate an .npz archive file containing the discretized trajectories for TSID. Then, we launched `hw1_tsid_biped_walking.py` to see the robot behavior in Gepetto viewer.

Interpolating function

Both the reference foot and CoM trajectories needed to perform a 4-step walk are computed using a Linear Inverted Pendulum Model (LIPM). Since the time step used in the reference trajectories is larger then the one of TSID, it is necessary to implement a third order interpolating function to generate new foot trajectories that will be used as reference in TSID. This function, called `compute_3rd_order_poly_traj()` has as inputs the current LIPM foot position, the next LIPM foot position, LIPM time step and TSID time step. The outputs of this function are the new foot sequences of position, velocity and acceleration values, considering the TSID time step. The first goal to achieve is to compute the coefficients a, b, c, d defining the third order polynomial:

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

To compute these coefficients, boundary conditions are defined. Considering the position value at the starting and ending point and imposing in the same

points the velocity values equal to zero, it is possible to obtain four equations for each direction x, y, z. The problem can be expressed in matrix form as:

$$A \cdot x = B \tag{2}$$

where x is the vector of the four unknowns $[a, b, c, d]$, B is the vector containing the the imposed values of boundary conditions and A is the coefficients matrix. The velocity equations are obtained deriving the polynomial time function of position (Eq 1):

$$\dot{x}(t) = b + 2ct + 3dt^2 \tag{3}$$

Substituting $t = 0$ at the starting point and $t = T$ at the final one to the equations 1 and 3, the problem 2 becomes:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & T & T^2 & T^3 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 2T & 3T^3 \end{bmatrix} \cdot \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix} = \begin{bmatrix} \alpha[0] \\ \alpha[1] \\ 0 \\ 0 \end{bmatrix} \tag{4}$$

where α is a general value indicating either $x(t)$ or $y(t)$ or $z(t)$.

Solving this problem imposing the boundary conditions previously explained, the four unknowns are computed for each polynomial describing $x(t)$ or $y(t)$ or $z(t)$ direction. At this point, vectors containing the values of the position, velocity and acceleration for each direction and at each time instant are filled using a `for` loop. These vectors have dimensions equal to:

$$size = \frac{T}{dt} \tag{5}$$

It is important to specify that the trajectory of $z(t)$ needs to be performed in two different steps, reaching a middle point before the final one. This division is necessary to take account of the ascend and the descend separately otherwise the solution would join the initial point and the end point with a straight line. For these reason inside the function `compute_3rd_order_poly_traj()` there is this type of distinction (carried out by an `if` condition): both x and y vectors are collected into three single matrix, one for the position steps and the others for the velocity and acceleration ones, while z and its derivatives are represented by three single vectors.

Therefore, calling this function, the outputs are values of position, velocity and acceleration for each little step dt.

Question 1

Running `hw1_tsid_biped_walking.py` with the default settings (`SQUAT=0`, `PUSH=0`, default weights and gains), the humanoid falls down before completing the 4-step walk.

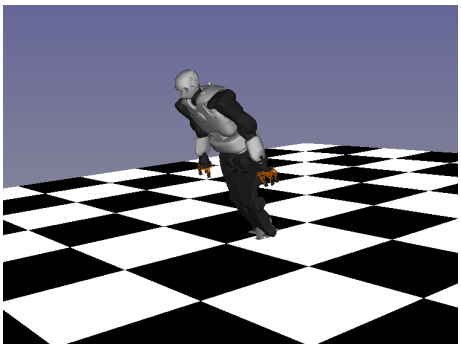


Figure 1: Humanoid robot falling

In order to allow the robot to complete the walk, it is necessary to tune the weights `w_com`, `w_foot` and `w_posture`. After a lot of attempts the following conclusion are reached: increasing `w_com`, the humanoid is more stable, because the CoM trajectory has a bigger weight in the total cost function and imbalances or falls are avoided. At the same time it is necessary to increase `w_foot`, otherwise the feet do not follow the desired trajectory and the humanoid starts crawling back instead of walking. The last parameter to tune is `w_posture`, which is responsible for the robot's posture: increasing this value the humanoid has a more elegant posture and it maintains a stiffer behavior. However, the value of `w_posture` is not too much influential on successfully performing the whole walk. The other two weights tuned have more importance.

As it is possible to observe in figure 2, using the default weights and gains the robot falls down: comparing the figure 2a with the figure 2b it is immediate to comply that in the first case the velocities diverge as a synonymous that the humanoid is falling down, while in the second case they remain bounded.

The same conclusions are observable in figures 2c and 2d, where in the first case the foots trajectories diverge and go out of bounds, while tuning the weight these trajectories remains bounded for the whole walk.

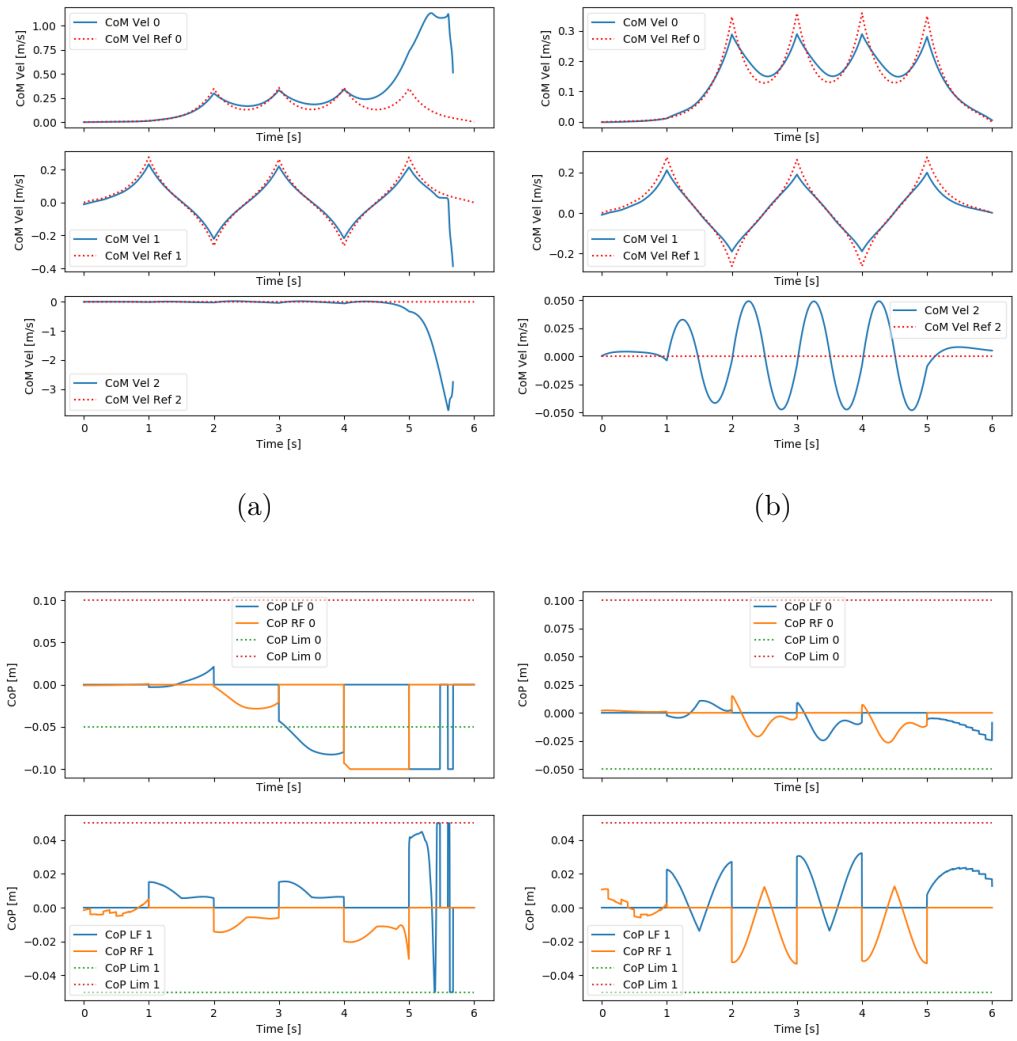


Figure 2: Plots of the velocities in x and y direction and of the foots trajectories, using the default weights (left) and the tuned ones (right).

The good results are reached imposing `w_com = 40`, `w_foot = 50` and `w_posture = 4`. These values are maintained for all the following analysis.

Question 2

The following objective is to simulate the walking while the robot is in squatting configuration. Selecting `SQUAT=1` in the configuration file, a lower reference value is given for the center of mass height. This new way of walking can be seen in Figure 3 and the values in Figure 4a.

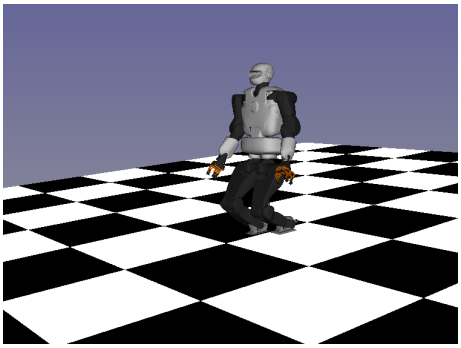
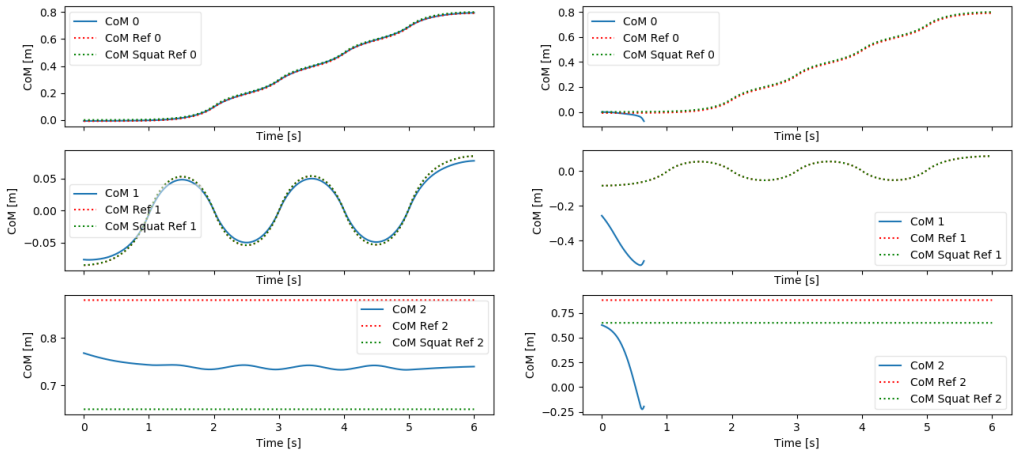


Figure 3: Humanoid robot in squatting configuration



(a) (b)

Figure 4: Humanoid center of mass when squatting.

The third plot in Figure 4a shows the value of the humanoid CoM in z-axis before the squatting blue, what it is able to reach red and the desired one green which is the lowest.
In order to reach the desired trajectory, we are suggest to tune `w_squat` or `kp_squat`. These parameters are both related to the achievement of the

desired height but they are useful in two different ways.

`kp_squat` is the proportional gain. As suggested by the definition, it is a scaling factor for the difference between the desired position and the real one.

`w_squat` is the weight of squat task. The difference with the previous parameter consists in the fact that this value select the importance of the squat task in the overall TSID problem.

Their effects can be seen in the resulting plots of the simulations. Increasing the proportional gain with low task weight, amplifies the position error but the whole problem will not consider the squat as a very important task and is not able to correct the unbalancing.

In Figure 4, the left plots show the reference trajectories for x, y and z axes for the whole simulation time. From them it is possible to see that the humanoid is able to complete the path. Nevertheless, the plots on the right show the trajectories followed when it falls. For this reason the functions end in less then a second. In the first case, figure4a, the task weight is bigger than the proportional gain and the solver try to achieve the lower level for the center of mass reaching quiet good results.

Lowering `w_squat` and increasing a lot the proportional gain, the robot starts from an unbalanced configuration and it is not able to achieve a stabilized configuration because even though the error is incremented inside the task, the overall problem do not consider it as very important issue due to the fact that it is scaled by a small weight. The humanoid center of mass in this case is plotted in figure 4b.

Question 3

Setting `SQUAT=0`, `PUSH=1`, an external push is applied to the humanoid: in particular, the CoM velocity push is equal to `push_robot_com_vel = [0.1, 0, 0]`. It is possible to look at the effect of this push in figure 5, where the velocity in x direction increases when the push is applied exactly of 0.1 m/s (the magnitude of the external push) with respect to the reference velocity.

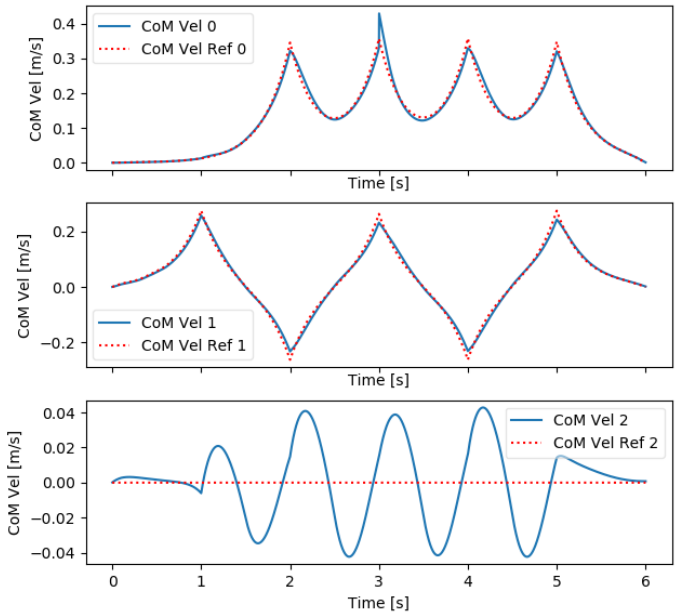


Figure 5: Trend of the velocities in the three axis when the push is applied.

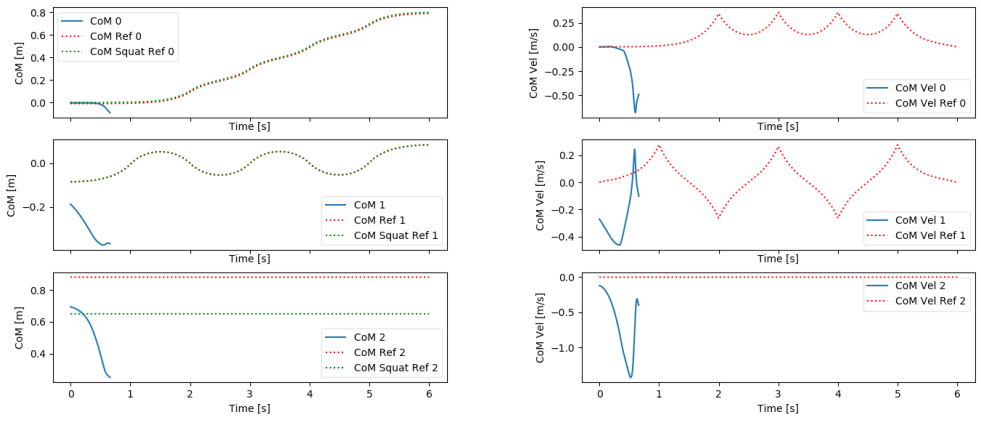
Keeping the tuned weights the humanoid does not fall down, because the applied push is very small and the parameters are well tuned to make the robot stable. In order to study how the chosen weights influence the effect of the external push, several attempts are carried out. The conclusion is that the humanoid falls if `w_com` and `w_foot` are decreased. In particular:

- decreasing a lot `w_com` but keeping `w_foot` at the same value the robot falls instantaneously, because trying to follow the correct foot trajectory it becomes unbalanced;
- decreasing both `w_com` and `w_foot` the humanoid does not fall at the beginning, but only as a consequence of the push, because not having to perform a correct movement of the foots, it keeps the balanced.

So, to avoid the humanoid falls down it is necessary to increase the weights `w_com` and `w_foot`, the first in order to keep balance and the second to allow the feet to follow the correct trajectory. There could be drawbacks in this, because increasing a lot the weights to allow the robot keeps balance also when it is pushed, other tasks may lose importance.

Question 4

The last part of the assignment is based on looking what happen when the humanoid walks in squat and endures an external push. These characteristics are enabled with `SQUAT=1` and `PUSH=1` in the configuration code. In the same file, in order to see the difference between `w_squat` and `kp_squat`, the assignment asks to set both equal to `100` and then run it again with `w_squat=10` and `kp_squat=1000`.



(a) (b)

Figure 6: Humanoid center of mass position and velocities while is squatting and reacting to an external push, tuning `w_squat`=10 and `kp_squat`=1000.

In figure 6 are shown the position and the velocity of the humanoid center of mass with the different weights. In particular, the figure 6a shows the actual values in the three axes with the desired ones. On the right, in figure

6b there are represented the velocities. The blue functions stop before one second because the robot falls down as in Question 2.
This is the same situation where the task weight is small and the solution for the actuators is obtained giving importance to other tasks instead of the squat one and although the error is enlarged as described in Question 2 with `kp_squat`, the resulting actions are not sufficient to stabilize the robot.

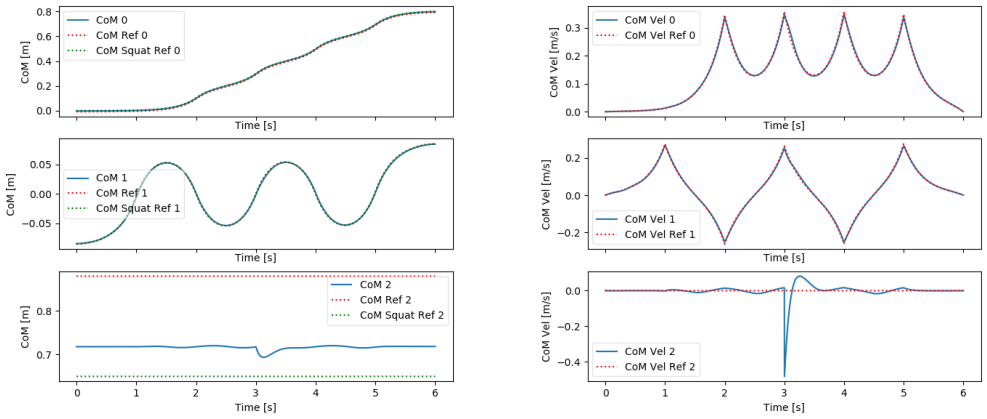


Figure 7: Humanoid center of mass position and velocities while is squatting and reacting to an external push, tuning `w_squat`=100 and `kp_squat`=100.

In the case of `w_squat=100` and `kp_squat=100` the humanoid is able to perform the whole walk without falling. This is due to the choice of the weights, because not having to minimize the error increasing `kp_squat` a lot, the importance of all the parameters is maintained. Looking at the figure 7, it is immediate to notice the presence of the external push received in z direction: the CoM trajectory decreases a little bit when the push is given to the robot, while the velocity presents a peak with the same intensity of the external push, `push_robot_com_vel = [0, 0, -0.5]`. After this, the robot reacts and tries to return stable, and these values become constant.