Optimisation Based Robot Control
Assignment 1

Nicola Farina
ID 229296
Dept. of Information Engineering and Computer Science
nicola.farina@studenti.unitn.it

Paolo Furia
ID 239451
Dept. of Industrial Engineering
paolo.furia@studenti.unitn.it

I. First answer

Once we have implemented the function `compute_3rd_order_poly_traj()` (whose documentation is available in the code), the humanoid barely manages to take four steps before falling to the ground, using the default weights. We examined the resulting plots, ignoring the last second when the robot falls, and figured out the following things:

- the CoM position and velocity trajectory, while not perfectly tracking the reference, is still decent as can be seen in Figures 1a, 1b;
- the CoP goes beyond the limits for the x component as can be seen in Figure 1g;
- the foot trajectory is very imprecise, in particular the z component.

In order to understand how to tune the weights, we proceeded by increasing or decreasing the various weights while keeping the rest with the default values.

- Since the robot with the default values didn't follow the CoM trajectory very well, we increased its weight `w_com` with the values $[1, 10, 100]$ while keeping the other weights as default. As expected, the robot followed the reference trajectory much better. Increasing to higher values only yielded better results. The robot managed to finish the 4 steps, but the foot lagged behind and it was evident that the robot would fall if the simulation did not end immediately after the 4th step. Thus, we proceeded to tune the foot-task weights.
- In order to improve the foot trajectory, we increased the foot weight `w_foot` with the values $[1, 10, 100]$. We noticed that the more the weight increases, the more the trajectory of the CoM deviated from the reference one, especially along the y component. This is explained by the fact that, when increasing the weight of a task, the relative importance of the other tasks diminishes.
  Instead, the trajectory of the feet along z changed for the better: increasing the weight brought the trajectory closer to the reference one.
  The most important result, though, is that a combination of higher-than-default CoM task weight and foot task weight resulted in a much better walk: the feet did not lag behind the CoM anymore, and the robot managed to keep its balance both during and after the walk.
- Finally we tuned the posture weight `w_posture` with the values $[2 \cdot 10^{-5}, 2 \cdot 10^{-3}, 2 \cdot 10^{-1}]$ and we noticed that the lower the weight, the greater the flexibility of the robot's joints, causing it to have an unnatural and shaky, drunk-like posture during the walk. Despite this, the greater flexibility allowed the robot to better follow the reference trajectory, by making it less stiff.
  Asymptotically speaking, high weights lead the robot to act somewhat like a rigid body, making it fall very fast when the CoM started moving. On the contrary, very low weights lead the robot to move very unnaturally and out of control, with frequent self-collisions and impossible real-life movements.

After trying different combinations of weights, following the reasoning explained in the previous points, we chose the following values, which lead us to the solution reported in the Figure 2.

$$\texttt{w\_com} = 100$$
$$\texttt{w\_foot} = 3$$
$$\texttt{w\_posture} = 6 \cdot 10^{-4}$$

While `w_com` was the easiest to tune (essentially, it is the most important task when it comes to keeping the robot's balance), we ended up with those exact values of `w_foot` and `w_posture` by making several tweaks. In particular, we tuned the posture in order to have a lower tracking error of the z component of the CoM acceleration, which proved to be the most difficult to deal with (we went from a maximum error of around 0.6 to 0.1). Also, tuning the posture allowed to keep the CoP graphs between the upper and lower limits.

II. Second answer

The most obvious change when setting `SQUAT=1` is in the tracking of the `z` component of the CoM trajectory. In particular, it never reaches the reference height of 0.88, and neither the desired squatting height of 0.65: the height remains stable at around 0.86.

This is to be expected, since setting the squat flag activates a task that pushes the robot into maintaining its CoM at height 0.65. This is directly in conflict with the task of tracking the CoM reference trajectory, which is constant at height 0.88. The fact that the robot is closer to the reference height than the desired squatting height is backed up by the fact that, with the same proportional gains for the CoM and squat (set at 10), our weight for the CoM task (100) is bigger than the default weight of the squatting task (10).

To reach the desired squatting height, we can either increase `kp_squat` or `w_squat`. In the end, they both drive the robot towards the same goal, but with important differences. To show them, we first increase the gain from 10 to 100 while keeping the weight fixed; and then we increase the weight from 10 to 100 while keeping the gain fixed.

When we increase the gain, we directly intervene on the PD controller. Increasing a gain means proportionally increasing the control signal for a given tracking error. In practice, this results in faster reaction times and "stronger" control inputs. This means that the desired trajectory is reached faster, but usually with overshooting. Also, such inputs result in less smooth trajectories, which are not desirable in robots as it could lead to damages to their parts.

If we increase the weight instead, we just give more importance to the squatting task relative to the CoM tasks (and the others) without directly increasing the sensitivity of the controller to the errors. These theoretical concepts can be observed in Fig. 4, where the height takes about 0.2 seconds (even after the 1.5 seconds stabilization phase) to stabilize with the increased weight, whereas it is already stable with the increased gain; and in Fig. 5, where there is an overshoot with the increased gain, as opposed to the gradual convergence with the increased weight.

III. Third answer

Although the robot is pushed, it does not fall. Assuming this happens, we believe the best way to prevent this is to act on the CoM in such a way as to keep the robot balanced and not allow it to fall, as a human would do.

Our first choice would be to increase the `kp_com` gain in order to be more reactive to external forces. This affects the motors in terms of reaction speed, and therefore drive torque, causing high control signals as the gain increases. Therefore, the robot is more rigid and recovers a balanced stance quickly enough, before its CoM falls outside of the balance boundaries.

However, as already covered in the previous answer, the gain should not be set excessively high, otherwise there is the risk to incur in a situation as in Fig. 6 with a gain set at 1000, with unstable, over-the-limits behavior, which would damage the motors in a real-life scenario.

We tried to simulate the robot with three values for `kp_com`: 10, 100, 1000. We saw that the best trade-off between reaction time to the push and low-stiffness is the value 100.

Another choice we would try to implement if the robot fell is to very slightly reduce the posture task weight or its gain. The logic behind this is that, with a high priority for posture, the robot acts similar to a rigid body. Therefore, when it is pushed, the robot does not have any flexibility and starts falling. The drawback would be that, with low importance on maintaining posture, the robot joints are less constrained and may produce non-natural movements, self-collisions, and movements not possible in real life. Additionally, if set too low, the robot would not oppose the push at all, bend, and fall. This means that we would need to tune the parameter very carefully.

IV. Fourth answer

Setting `SQUAT=1`, `PUSH=1`, `push_robot_com_vel=[0,0,-0.5]` and switching between the two configurations of weight `w_squat` and gain `kp_squat`, we noticed that the only differences were:

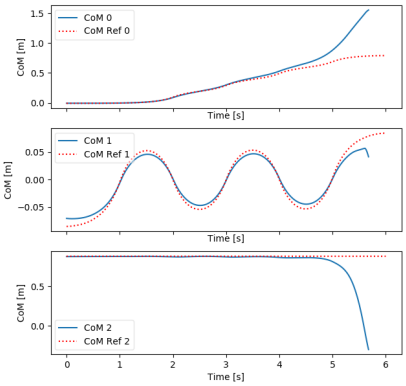
- the acceleration of the center of mass, in the z component;
- the torque exerted by the motors.

By increasing the gain `kp_squat` to 1000 (and decreasing the weight `w_squat` to 10) it is possible to notice, as shown in the Figure 7b, the presence of a slightly higher acceleration peak along the z component, and a slightly faster return to the reference trajectory in terms of time.

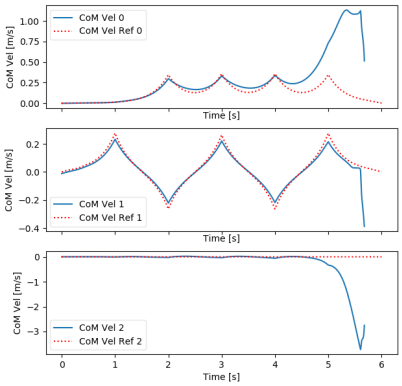
Similarly, Figures 7c and 7d show that the torque needed to balance the thrust is more pronounced than with a lower gain.

The justification for this behavior is the definition of gain and weight. As argued in the previous answers, increasing the gain gives a stronger, faster signal response. The motors exert a greater torque more quickly in order to counteract the thrust coming from the external push.

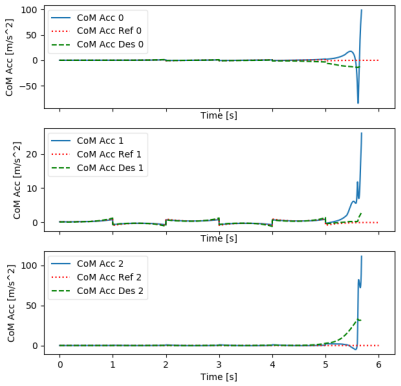
V. PLOTS



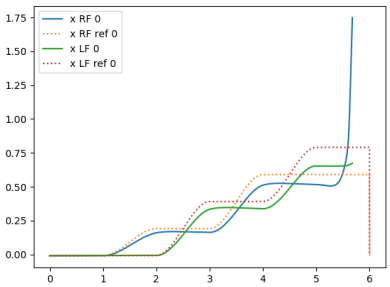
(a) CoM position



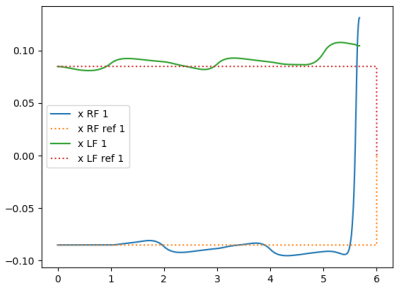
(b) CoM velocity



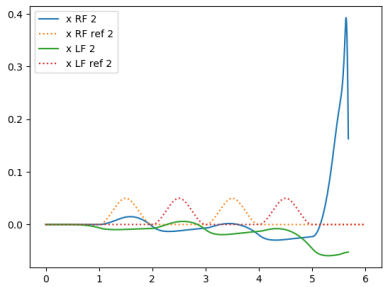
(c) CoM acceleration



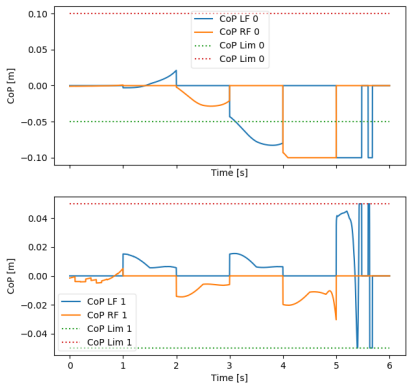
(d) Foot x position



(e) Foot y position

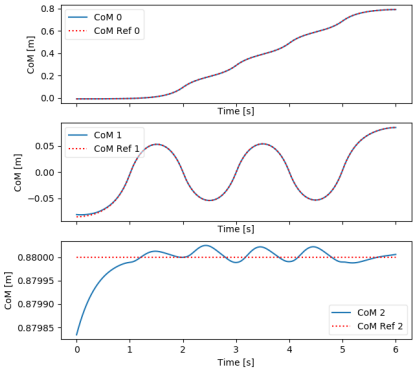


(f) Foot z position

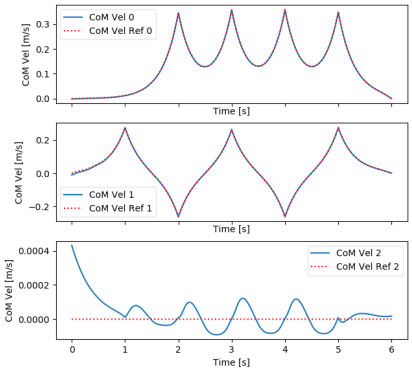


(g) CoP results

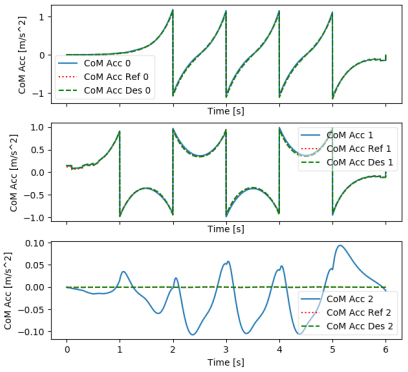
Fig. 1: Default weights results



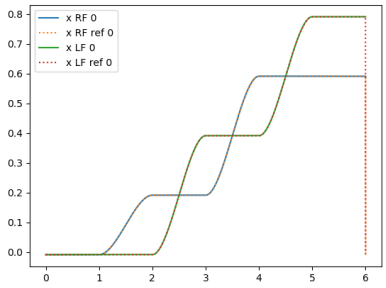
(a) ComM position



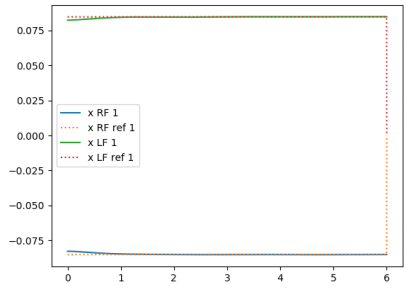
(b) CoM velocity



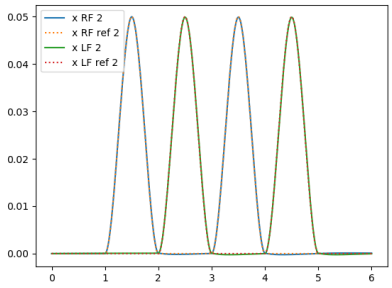
(c) CoM acceleration



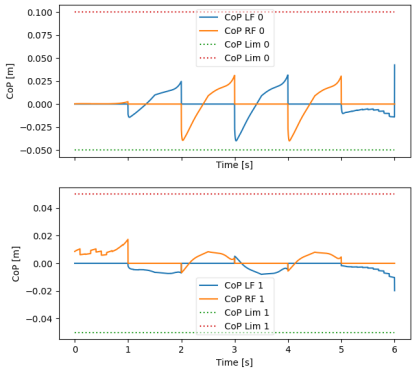
(d) Foot x position



(e) Foot y position

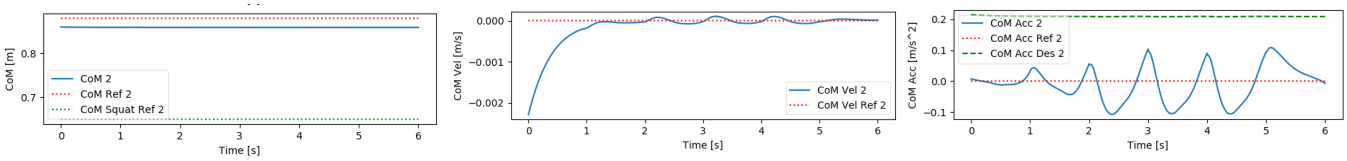


(f) Foot z position



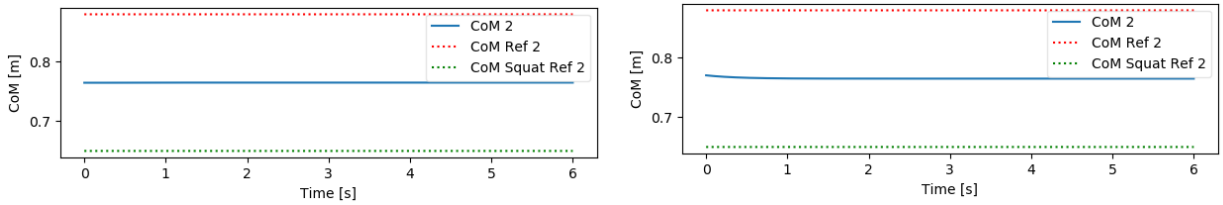
(g) CoP results

Fig. 2: Final weights results



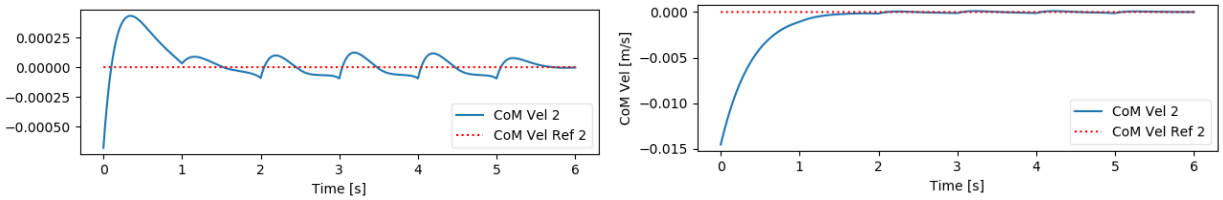
(a) Position (b) Velocity (c) Acceleration

Fig. 3: CoM trajectory when squatting with tuned weights (z component)



(a) Increased gain (b) Increased weight

Fig. 4: CoM position when increasing squat gain and weight (z component)



(a) Increased gain (b) Increased weight

Fig. 5: CoM velocity when increasing squat gain and weight (z component)

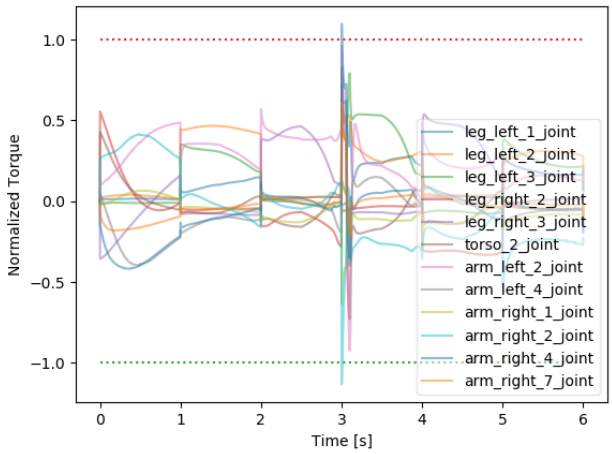
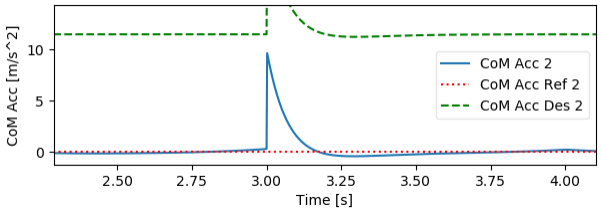
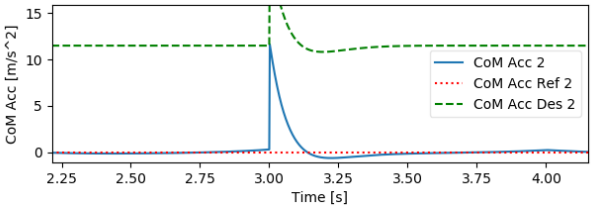


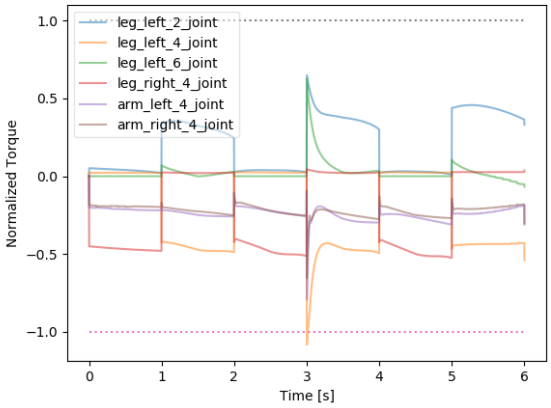
Fig. 6: Joint torques with `kp_com=1000`



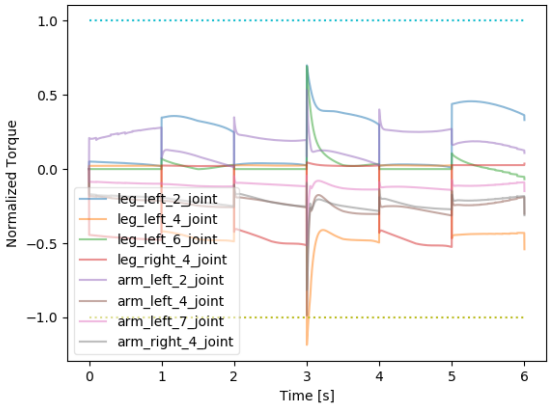
(a) CoM acceleration along z with `w_squat=100`, `kp_squat=100`



(b) CoM acceleration along z with `w_squat=10`, `kp_squat=1000`



(c) Torques with `w_squat=100`, `kp_squat=100`



(d) Torques with `w_squat=10`, `kp_squat=1000`

Fig. 7: Differences between setting [`w_squat=100`, `kp_squat=100`] and [`w_squat=10`, `kp_squat=1000`] with `PUSH=1` and `SQUAT=1`.