ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

PRACTICAL WEEKS 1-3

LEGGED ROBOTS

Camille Coppieters De Gibson, Chiara Evangelisti, Advaith Sriram

30th September 2024

PART 2 - JACOBIAN

1. Begin with the `on_rack` variable equal to true, to suspend the leg in the air. Fill in the code blocks to compute the absolute (double pendulum) and relative (as in the URDF) Jacobians, and use a Cartesian PD controller to keep the foot at $(x, z) = (0, -0.3)$ m. Try changing the gain matrices $K_{p,Cartesian}$ and $K_{d,Cartesian}$. How important are these? What happens if you set `on_rack` to false - can you use the same gains? How does this affect the foot position? How about when dragging the leg up and letting it fall? It will be helpful to plot the foot position/velocity, joint positions/velocities, and motor torques

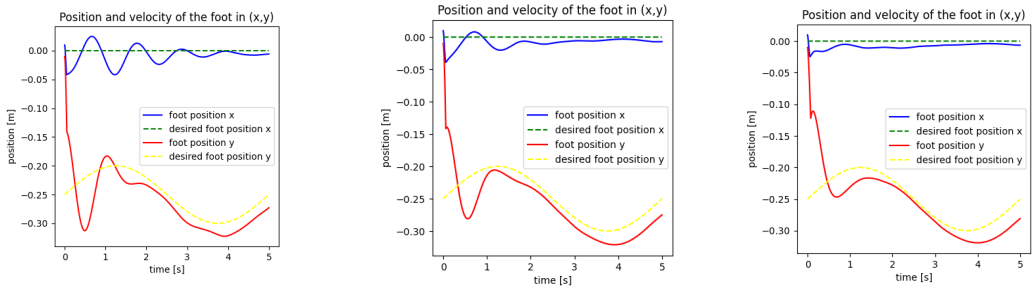
ANSWER

The gain matrices $K_{p,Cartesian}$ and $K_{d,Cartesian}$ play a key role in the stability of the controller. Their values must be tuned to achieve stability for a given application, this is the reason why when turning the `on_rack` variable equal to False, thus changing the dynamics of the system and the effect of gravity the same gains cannot be used anymore. Indeed as the robot is not hung up in the air anymore, but no gravity compensation has been added yet, the robot will follow any perturbation. If the foot position is modified (yet staying on the ground) the whole robot will move in that direction without being able to come back to the desired position, similarly when the leg is dragged up the whole leg will follow, performing a kind of parabolic motion in the air.
Gains for `on_rack` true : $K_{p,Cartesian} = [100, 100]$ and $K_{d,Cartesian} = [9.5, 9.5]$

2. With `on_rack` set to False, now try to apply force control by compensating for gravity and the mass of the system (use `env.robot.total_mass`, and be careful with the sign). What are the minimum gains you can use?

ANSWER

From a gain of $K_p = 200$ and $K_d = 20$ we have a good following of the foot position with the desired one. If we set K_p slightly lower it follows the desired foot position with some decay. If the $K_p < 75$ the foot just gets blocked in some position and doesn't let a go-return track possible.
For K_d we have to choose some value that is not too high, otherwise, the time that the foot gets in the right position would be long and it would never attend the desired trajectory. We have a value of $K_d = 20$ that works fine. If K_d is not high enough, the foot will never stabilize and it will keep oscillating a little bit around the desired point. We tried different $K_{d,Cartesian}$ in figure 1 to show the effect of the gain matrix. We see that in Subfigure B, the foot takes more time to stabilize, but at the end of the trajectory it's a little bit nearer to the desired position than in Subfigure C.



(A) $K_{d,Cartesian} = 10$ **(B)** $K_{d,Cartesian} = 20$ **(C)** $K_{d,Cartesian} = 40$

FIGURE 1
Setting initial joint angles at an unstable value $q^0 = [\frac{-\pi}{2}, \pi]$ to catch the effect of different $K_{d,Cartesian}$

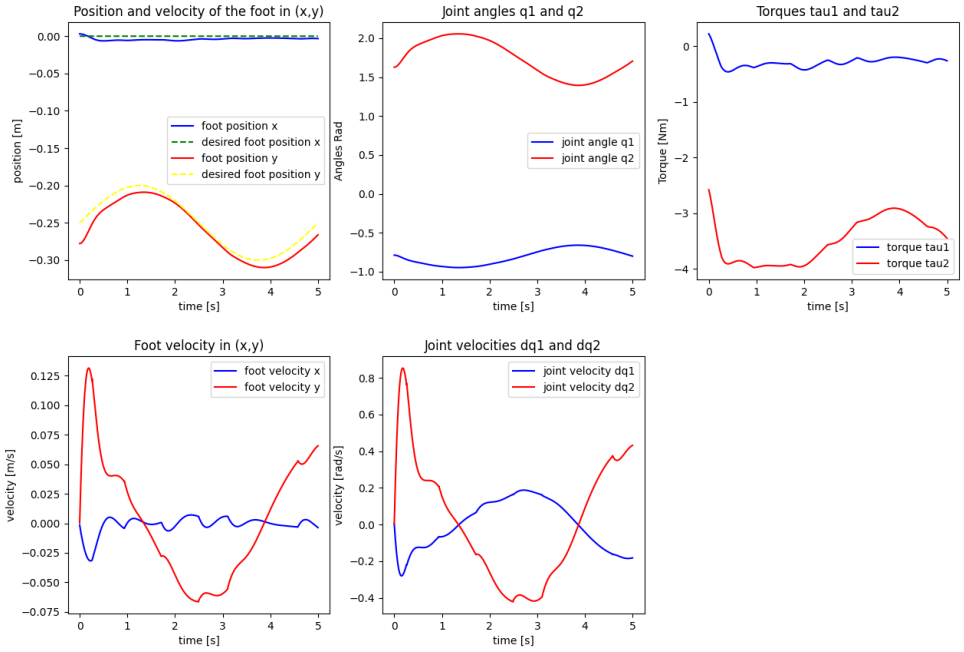


FIGURE 2
Plots of the foot position, motor angles and velocities, and the torques during a trajectory with the foot going from height 0.3 to 0.2 and vice versa

3. Plan a trajectory in task space and track it with the Cartesian PD controller. For example, decrease the hip height from $0.3\ m$ to $0.2\ m$ and vice versa over different time windows. Would you be able to plan such a trajectory in joint space?

   ANSWER

   We can see the results in figure 2 of a trajectory going from foot position height 0.3 to 0.2 and vice versa, for a smoother trajectory we used a sinus.
   We see that the torque magnitude when the hip goes up is higher than when it goes down. It is logical because when it goes up it has to counteract gravity. The reference along the x axis is constant, but we can observe some small deviations especially at the beginning of the trajectory, the leg then aligns with the tracked reference.

   In joint space, it is possible to plan such a trajectory, but it is less straightforward: to achieve the same vertical foot trajectory, one would need to calculate the inverse kinematics at each time step, to compute the joint angles q_1 and q_2 at each point in the trajectory to ensure the hip moves along the desired vertical path.

$$[x_2, y_2] = f(q_1, q_2) \Leftrightarrow [q_1, q_2] = f^{-1}(x_2, y_2)$$

4. What benefits do you see from using impedance control (i.e. Cartesian PD + force control)? What could be improved? (i.e. what if a particular joint configuration is required, for example, if the leg should look like < or >. Try starting with different initial angle configurations.)

ANSWER

If we set different initial angles we can observe different behaviours:

- With initial joint angles that are in the same order as $q = [-\frac{\pi}{4}, \frac{\pi}{2}]$ the reaction will be pretty similar to the reaction shown on figure 2
- With initial joint angles opposite to values as in the last situation we see the same sort of reaction the only difference is that the torque becomes positive.
- With initial joint angles far lower than $q = [-\frac{\pi}{4}, \frac{\pi}{2}]$ the foot begins to jump in the air because of repulsion to the ground. When it falls back towards the ground it stabilizes and gets a fine end of movement. We see the passage in the air mostly in a low absolute torque and a high difference in y position.
- With initial joint angles far higher than $q = [-\frac{\pi}{4}, \frac{\pi}{2}]$ the foot is in a first time more curled on itself, it then deploys and stabilizes itself round the desired foot position.

Using impedance control increases the robot's stability exploiting the information concerning the forces applied at the foot. Furthermore, it acts in task space, making it easier to plan and execute movements such as the aforementioned trajectory. However, as it focuses on forces and trajectories in task space, it does not enforce a specific joint configuration (like $>$ or $<$, to do so it would need to be combined with joint control and by adding constraints on the joint configuration.

PART 3 - INVERSE KINEMATICS

1. Iterative IK: How can we find both sets of solutions with the algorithm (i.e. leg looking like < or >)?

   ANSWER

   The iterative IK method requires an initial guess for the joint angles: initializing the algorithm with different starting configurations can introduce a bias leading it to provide one solution rather than the other. For the '<' configuration for example $q_0 = [\pi/4, -\pi/2]$ may be the initial guess, whereas for the '>' one we would rather choose $q_0 = [-\pi/4, \pi/2]$. Because the two solutions are quite far from each other (they are indeed opposite) the algorithm will tend toward the solution closer to the initial configuration provided. Therefore to obtain both the solutions it is sufficient to do multiple runs with suitable initial guesses.

2. What benefits do you see from tracking task space trajectories with inverse kinematics (and joint PD control) vs. impedance control (with Cartesian PD + force control)? When might you use one over the other? What if a particular joint configuration is required, for example, if the leg should look like < or >? What about motor-related considerations (torque sensing, bandwidth)?

   ANSWER

   Using inverse kinematics and joint PD control allows precise and simple control (just one PD per joint) over the leg position and configuration. This is a result of the mapping from the task space to the joint one. Therefore it is useful to apply when specific joint configurations are needed, when computational constraints require the use of simple control laws, and when the main concern is the position of joints and end-effector while the torque specifications are less critical. On the other hand, impedance control is preferred when the interactions with the environment play a key role and the control must be carefully adjusted to the external forces applied when disturbances or impacts are expected since controlling the torques helps to face them, and finally when tracking forces is crucial (maybe because of actuators limitations).

PART 4 - SINGLE-LEG HOPPING

1. Describe your methodology for implementing your jumping controller. What type of control do you use (joint space, task space, force profile)? Do the motions look realistic (torque limits)? Can you do both a single jump and continuous jumping (forwards/backwards/in place)?

ANSWER

The controller we adopted exploits all three types of control: joint space, task space, and force profile. The force profile is used to define the hopping motion: the force (both x and y profile) is a truncated sinusoidal having negative values when the leg is in contact with the ground and 0 when the leg is in the air. On top of that, a Cartesian PD controller is always active allowing us to control the overall leg motion making sure that the foot tracks the desired path. Finally, when the leg is in the air the ground interaction forces are not relevant anymore so it becomes useful to directly track the joint angles so that the robot will be in a correct configuration for landing.
For all four tasks the nominal foot position was maintained fixed, indeed this reference is set from the foot to the base of the leg, therefore the control using such a reference is used to keep the robot in the right configuration rather than making it trace a trajectory in the xy plane. As a consequence, the discriminant factor between the forward, backward, and in-place motion for continuous jumping was simply the direction and the magnitude of the force along the x-axis.
In the following more details concerning the parameters for each of the four cases considered:

	$f[Hz]$	$F_{z,max}[N]$	$F_{x,max}[N]$	$K_{p,Cart}[-]$	$K_{d,Cart}[-]$	$K_{p,Joint}[-]$	$k_{d,Joint}[-]$
Single	1.2	10*m*g	0	(200,300)	(20,30)	(55,55)	(0.8,0.8)
Forward	1.2	5*m*g	0.35*$F_{z,max}$	(500, 300)	(30,20)	(50,50)	(3.5, 3.5)
On place	1.2	3.5*m*g	0	(500,300)	(30,20)	(50,50)	(3.7,3.7)
Backwards	1.2	3*m*g	- 0.35*$F_{z,max}$	(500, 300)	(30,20)	(50,50)	(3.5, 3.5)

TABLE 1
Parameters for Different Implementation Types, m = `self.env.robot.totalmass`

The values from the previous table were found through trial and error, notably the forces in the x and z directions (`F_max,z` and `F_max,x`).

Regarding frequency and the force profiles we ran different optimizations with different cost functions. For instance we tried maximizing the maximum height (`f1 = -max_base_z`), minimizing the torque (`f1 = sum(abs(tau))`), maximizing the height to force ratio (`f1 = -max_base_z/F_max,z`) and minimizing the x deviation from the origin for continuous hopping on place
(`f1 = self.env.robot.GetBasePositions()[0]`). However each optimization run was very time-consuming, therefore due to lack of time we were unable to find more suitable value through optimization compared to the ones obtained from trial and error.

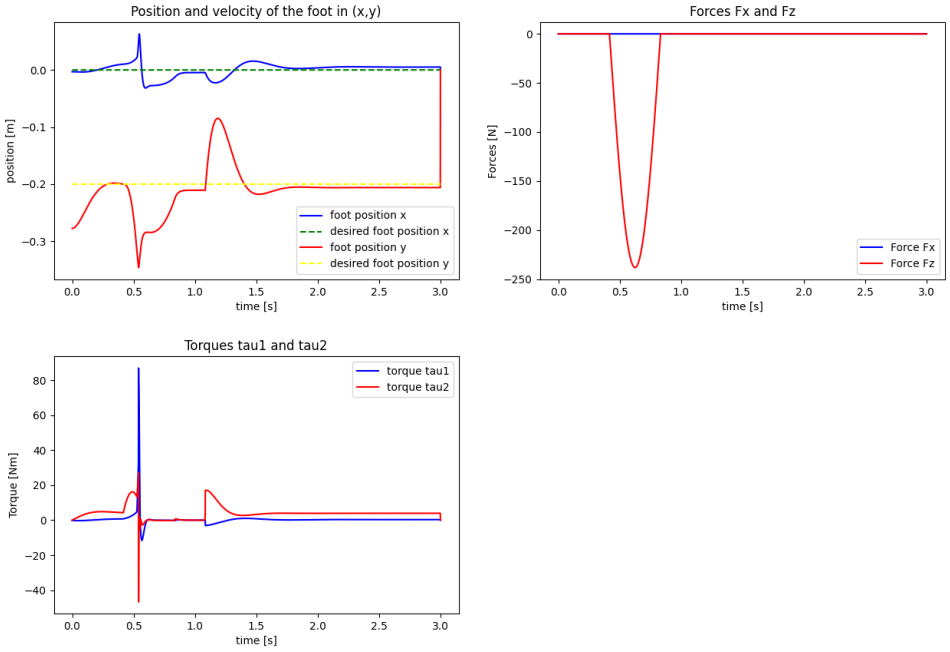


FIGURE 3
Position, torques and forces for the Single Jump

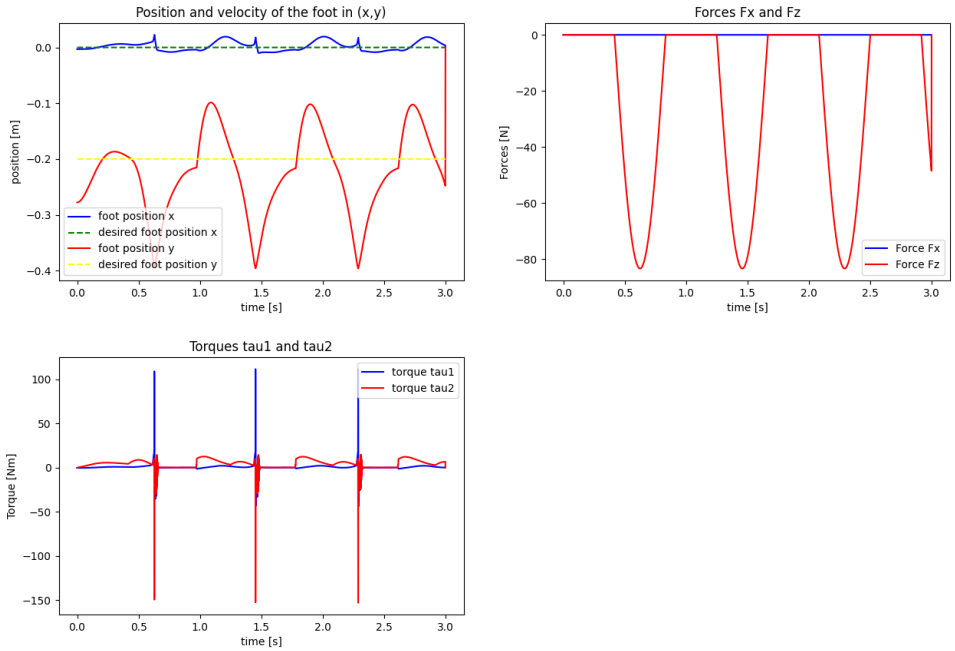


FIGURE 4
Position, torques and forces for the On Place Jumping

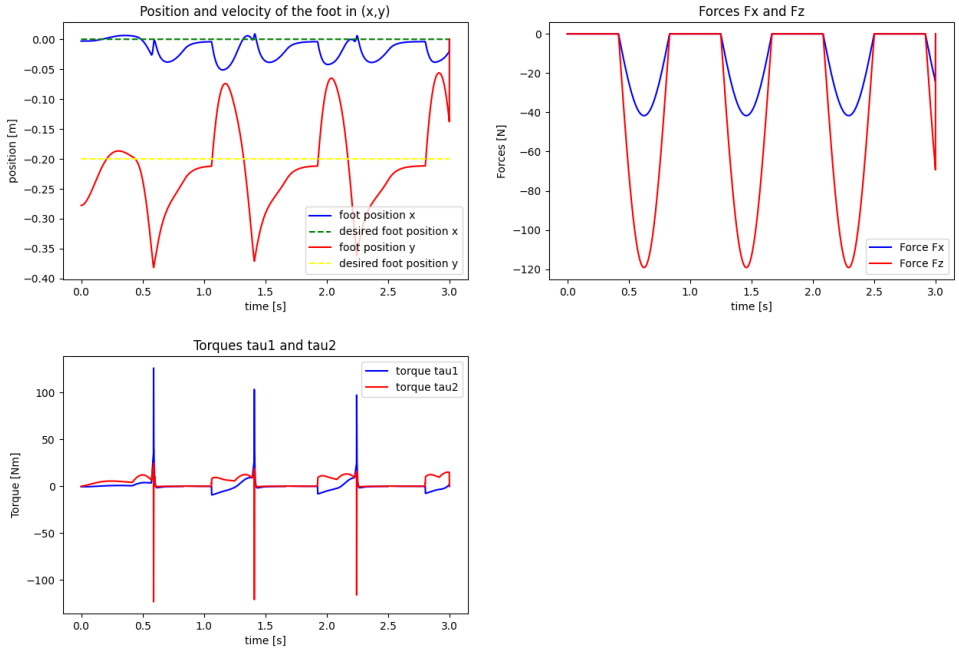


FIGURE 5
Position, torques and forces for the Forward Jumping

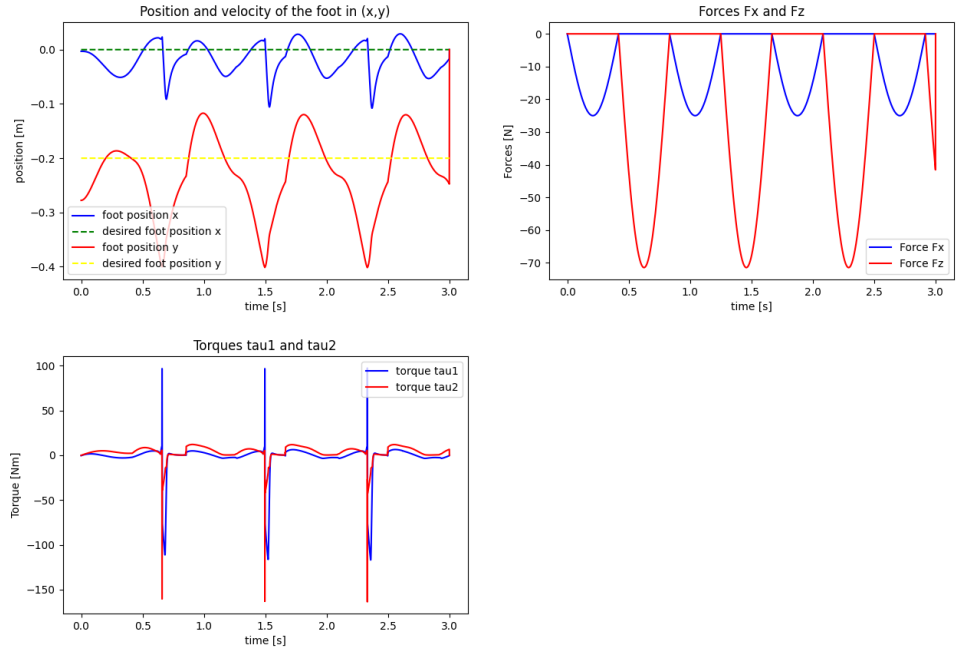


FIGURE 6
Position, torques and forces for the Backwards Jumping

As we can observe from the plots, on average the torques represent a realistic motion. Some spikes are registered when the robot enters in contact with the ground and even though it is an expected behavior , they are quite elevated. It is likely that perfecting the tuning of the controller a smoother hopping motion can be obtained, with consequently lower torque peaks.