Legged Robots Practicals – Week 3

Contents

1 Introduction

In the first weeks we will explore the two-link pendulum to control a quadruped leg. Please refer to the slides for additional details. In the first week we derived the kinematics and dynamics of the double pendulum. In the second week we used the Jacobian and inverse kinematics to control the foot position of a quadruped leg in simulation. In this last ungraded assignment, we will combine our knowledge to make a hopping controller for a single leg.

2 Code Installation

The code is hosted at `https://gitlab.epfl.ch/lgevers/lr-practicals`. The `README.md` gives details on installation and code structure. The repository contains the code for all the practicals of week 1-3. In this session you will only work on `practical4_hopping.py` and `practical4_hopping_opt.py`.

3 Theory Review

3.1 Modeling

We consider the same quadruped leg in `pybullet` as last week. Here is a quick recap of the relevant relationships we have used so far:

Forward kinematics: The relationship between the joint angles $\boldsymbol{q}$ and foot position $\boldsymbol{p}$:

$$\boldsymbol{p} = f(\boldsymbol{q}) \tag{1}$$

where the function $f(\cdot)$ denotes the forward kinematics of a single leg with respect to the leg frame.

Inverse kinematics: The relationship between a desired foot position back to desired joint angles:

$$\boldsymbol{q} = f^{-1}(\boldsymbol{p}) \tag{2}$$

For general robotic systems this relationship will not be unique (i.e., there are multiple sets of joint configurations $\boldsymbol{q}$ that satisfy $\boldsymbol{p} = f(\boldsymbol{q})$). For the quadruped leg, we used either the iterative (numerical) inverse kinematics or the following analytical inverse kinematics:

$$q_2 = \mp \cos^{-1}\left(\frac{x^2 + z^2 - l_1^2 - l_2^2}{2 l_1 l_2}\right) \tag{3}$$

$$q_1 = \texttt{np.arctan2}(-x, -z) - \tan^{-1}\left(\frac{l_2 \sin q_2}{l_1 + l_2 \cos q_2}\right) \tag{4}$$

Foot linear velocity: Differentiating the forward kinematics relationship gives the foot velocity $\boldsymbol{v}$:

$$\boldsymbol{v} = \boldsymbol{J}(\boldsymbol{q})\dot{\boldsymbol{q}} \tag{5}$$

where $\boldsymbol{J}(\boldsymbol{q}) \in \mathbb{R}^{2\times 2}$ is the single-leg Jacobian of the foot position w.r.t. the leg frame.

Force relationship: The Jacobian can also be used to map a desired force $\boldsymbol{F}$ at the end effector (foot) to joint torques:

$$\boldsymbol{\tau} = \boldsymbol{J}^T(\boldsymbol{q})\boldsymbol{F} \tag{6}$$

3.2 Control

Last week we implemented controllers both in joint and Cartesian space. The position control in joint space is implemented as:

$$\boldsymbol{\tau}_{\text{joint}} = \boldsymbol{K}_{p,joint}(\boldsymbol{q}_d - \boldsymbol{q}) + \boldsymbol{K}_{d,joint}(\dot{\boldsymbol{q}}_d - \dot{\boldsymbol{q}}) \tag{7}$$

where $\boldsymbol{q}_d$ are the desired joint angles, $\dot{\boldsymbol{q}}_d$ are the desired joint velocities, and $\boldsymbol{q}$ and $\dot{\boldsymbol{q}}$ are the current joint angles and joint velocities, respectively. $\boldsymbol{K}_{p,joint}$ and $\boldsymbol{K}_{d,joint}$ are vectors of proportional and derivative gains.

The position control in Cartesian space is implemented as:

$$\boldsymbol{\tau}_{\text{Cartesian}} = \boldsymbol{J}^T(\boldsymbol{q})\left[\boldsymbol{K}_{p,Cartesian}(\boldsymbol{p}_d - \boldsymbol{p}) + \boldsymbol{K}_{d,Cartesian}(\boldsymbol{v}_d - \boldsymbol{v})\right] \tag{8}$$

where $\boldsymbol{p}_d$ is the desired foot position, $\boldsymbol{v}_d$ the desired foot velocity, $\boldsymbol{p}$ and $\boldsymbol{v}$ the current foot position and velocity, respectively. $\boldsymbol{J}(\boldsymbol{q})$ is the foot Jacobian at joint configuration $\boldsymbol{q}$, and $\boldsymbol{K}_{p,Cartesian}$ and $\boldsymbol{K}_{d,Cartesian}$ are diagonal matrices of proportional and derivative gains.

Recall that the Jacobian can be used to map a desired force $\boldsymbol{F}$ at the end effector (foot) to joint torques:

$$\boldsymbol{\tau}_{\text{FF}} = \boldsymbol{J}^T(\boldsymbol{q})\boldsymbol{F} \tag{9}$$

By designing a proper force profile $\boldsymbol{F}$ over time we can achieve a single upward jumping or continuous forward hopping.

4 Assignment: Single-Leg Hopping

Fill in `practical4_hopping.py` by implementing the desired force profiles and a leg controller to execute them. Explore the following questions:

1. What type(s) of control do you use (joint space, task space, force profile)? What is the role of each controller?
2. Do the motions look realistic (check torque limits)?
3. Can you achieve both a single jump and continuous jumping (forwards/backwards/in place)?

5 Assignment: Hopping Optimization

Transfer your solution to `practical4_hopping_opt.py` and fill in the optimization procedure to achieve (a) the highest jump, and (b) the fastest forward hopping. We provide an interface to pymoo (https://pymoo.org/index.html) using CMA-ES (https://pymoo.org/algorithms/soo/cmaes.html). Note that it is possible to use a different algorithm to tune the parameters. Explore the following questions:

1. Start optimizing the force profiles only. Which variables do you optimize? What is your objective function? How do they differ between the jumping and forward hopping tasks?
2. Are the optimization results realistic (torque limits)? How do you keep it realistic?
3. Can you think of additional variables to optimize (e.g., controller gains)? How does it (or does it not) improve the jumping/hopping performance?
4. (Bonus) Can you think of other tasks for the hopping controller to achieve? How would you design the force profiles and its optimization variables?