Robotic Systems I: Homework II

Instructor: Konstantinos Chatzilygeroudis (`costashatz@upatras.gr`)

April 22 2024

Introduction

The homework concerns controlling a manipulator's end-effector to follow a path in 3D space. In particular, the homework is split in three parts: 1) modeling and simulation: we need to load the URDF of a Franka Panda manipulator (given as `franka.urdf`) and create a simulation loop using the pinocchio library, 2) task-space controller: we need to develop a task-space controller that controls the orientation and translation indepedently, and 3) evaluation: we need to evaluate the developed controller and simulator.

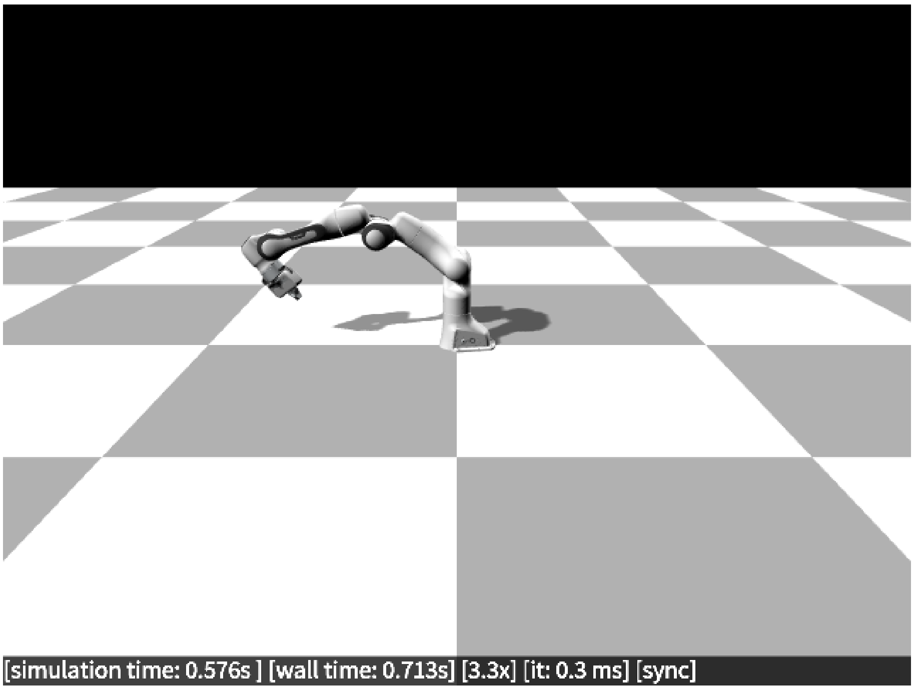

Figure 1: Visualization of the Franka manipulator

1 Modeling and Simulation (25%)

In this homework, we are going to use a Franka Panda manipulator (Fig. 1). The robot is a 7 degrees of freedom manipulator equipped with a gripper. In this homework, we have disabled the gripper.

In this part, you are asked to:

1. Write a function in Python called `load_franka()` that loads the URDF file of Franka (`franka.urdf`) and returns the model read by `pinocchio`.
2. Write a function in Python called `step_world()` that accepts the current state of the robot $(\boldsymbol{q} \in \mathbb{R}^{7\times 1}, \boldsymbol{v} \in \mathbb{R}^{7\times 1})_k$, the control inputs (torques) $\boldsymbol{\tau} \in \mathbb{R}^{7\times 1}$, a timestep $dt \in \mathbb{R}$ and outputs

the new state of the world $(\boldsymbol{q} \in \mathbb{R}^{7\times1}, \boldsymbol{v} \in \mathbb{R}^{7\times1})_{k+1}$ after applying the control inputs $\boldsymbol{\tau}$ to the system state. You need to make sure that the robot always stays inside the joint limits (in position and velocity). You need to use the `integrate()` function from the `pinocchio` library.

3. Visualize the system using a visualization library (you can use the provided `MeshCat` visualization scheme; `meshcat.ipynb`)

For all the above, you are allowed to use only native Python functions/classes, the `pinocchio` library, the `numpy` library and some visualization library (e.g. `matplotlib` or `MeshCat`).

2 Task-Space Controller (50%)

In this part, we want to create a task-space controller for torque control. In essence, we assume access to a desired end-effector pose profile $\boldsymbol{T}_{wd}(t) \in \mathbb{R}^{6\times1}$, and we want to create an end-effector *wrench* signal. For example, we can create a signal using a PID controller (in the world frame):

$$\mathcal{F}_w(t) = K_p\mathcal{X}_e(t) + K_i\int_0^t \mathcal{X}_e(t)dt + K_d\dot{\mathcal{X}}_e(t), K_p, K_i, K_d > 0 \tag{1}$$

where $\mathcal{X}_e(t) = \begin{bmatrix}\log(\boldsymbol{R}_{wd}(t)\boldsymbol{R}_{wb}(t)^T)\\ \boldsymbol{t}_{wd}(t) - \boldsymbol{t}_{wb}(t)\end{bmatrix}$, $\boldsymbol{T} = \begin{bmatrix}\boldsymbol{R} & \boldsymbol{t}\\ \boldsymbol{0} & 1\end{bmatrix}$, and $\boldsymbol{T}_{wb}(t)$ is the current end-effector pose.

We then need to transform this *wrench* signal $\mathcal{F}_w(t)$ to joint torques $\boldsymbol{\tau}$. For this, you are free to use any method: the easiest one is $\boldsymbol{\tau} = \boldsymbol{J}_w(\boldsymbol{q})\mathcal{F}_w$, where $\boldsymbol{J}_w(\boldsymbol{q})$ is the Jacobian of the end-effector expressed in the world frame.

Instructions:

1. You need to use the `panda_ee` frame as the end-effector
2. You are free to generate/compute the desired end-effector wrench signal, $\mathcal{F}(t)$, in any way that you wish
3. You can transform the end-effector signal to joint torques in any way that you wish
4. **The task-space controller needs to compute *the orientation and translation errors indepedently***
5. You need to add a regularization task with a null-space controller
6. You need to justify and describe all of your choices

For all the above, you are allowed to use only native Python functions/classes, the `pinocchio` library, the `numpy` library and some visualization library (e.g. `matplotlib` or `MeshCat`).

3 Evaluation (25%)

In this part, we will test the developed simulator and controller. In essence, you need to create several end-effector pose profiles and use the task-space controller to follow them in the simulator that you developed in the first part.

Instructions:

1. Use $dt = 0.001$ for the simulation
2. You need to use the `panda_ee` frame as the end-effector
3. You are free to calculate the desired end-effector pose profile $\boldsymbol{T}_{wd}(t)$ in any way that you wish. You need to create **at least four different profiles**
4. Discuss and visualize the results
5. You need to justify and describe all of your choices

For all the above, you are allowed to use only native Python functions/classes, the `pinocchio` library, the `numpy` library and some visualization library (e.g. `matplotlib` or `MeshCat`).

4 Deliverables

- Python file(s) with commented code[1]
- A short report

Bonus Points

- There is a 10% bonus if you implement the task-space controller that includes the task space inertia
- There is a 10% bonus if you implement your controller with a QP-based optimization scheme

[1] Jupyter notebooks are accepted as well.