Robot Learning

Winter Semester 2022/2023, Homework 1

Prof. Dr. J. Peters, N. Funk, T. Schneider, J. Ritter, K. Hansel and D. Rother

TECHNISCHE UNIVERSITÄT DARMSTADT

Total points: 42 + 4 bonus
Due date: 23:59, Monday, 14 November 2022

Note: Inside the homework folder, there are:

- A Latex template for this exercise that you can use to fill in your answers and that you can also directly use in sharelatex to collectively write the answers!
- There is also a Conda environment prepared in the "code" folder. To install this, follow the instructions in the README file.

Important: For every answer, please describe precisely how you obtain the results and justify every answer. Otherwise, we cannot give you the points (as in the exam)!

Problem 1.1 Robotics in a Nutshell [12 Points + 2 Bonus]

The robot shown in Figure 1 is a model of an articulated robot. An articulated robot is widely applied in industry. Its kinematics chain has three rotational joints $\theta_{1,2,3}$. We will consider only the first two joints $\theta_{1,2}$ to keep the task simpler.

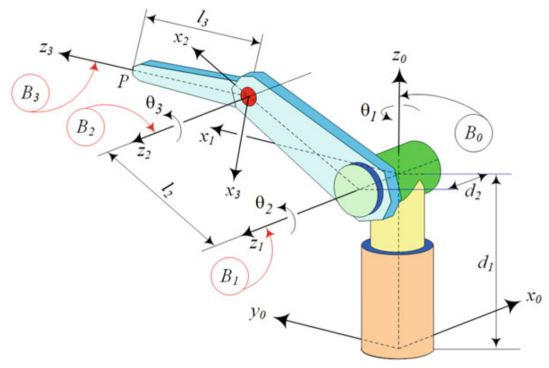

Figure 1: A model of an articulated arm

joint i	θ_i	d_i	a_i	α_i
1	θ_1	d_1	0	$-\frac{\pi}{2}$
2	θ_2	d_2	l_2	0
3	θ_3	l_3	0	$\frac{\pi}{2}$

Table 1: DH parameter table for setting up the link frames

We want to compute the forward kinematics using the Denavit-Hartenberg convention. For that, the DH parameter table of the robot at rest position is set up as indicated in Table 1.

a) Forward Kinematics [4 Points]

Compute the transformations ${}^0T_1$, ${}^1T_2$ and then set up the forward kinematics model ${}^0T_2$.

b) Trajectories [3 Points]

Please describe in words: What is a trajectory in robotics? Why should any trajectory representation be twice differentiable? Are cubic splines suitable for obtaining trajectories?

c) Differential Kinematics [4 Points]

Assume $\boldsymbol{x}_{\text{end-eff}} = [x, y, z]^T$ (tip point P) given by:

$$\boldsymbol{x}_{\text{end-eff}} = {}^0r_3 = \begin{pmatrix} {}^0r_{3,x} \\ {}^0r_{3,y} \\ {}^0r_{3,z} \end{pmatrix} = \begin{pmatrix} -d_2 \sin\theta_1 + l_2 \cos\theta_1 \cos\theta_2 + l_3 \cos\theta_1 \sin(\theta_2+\theta_3) \\ d_2 \cos\theta_1 + l_2 \cos\theta_2 \sin\theta_1 + l_3 \sin\theta_1 \sin(\theta_2+\theta_3) \\ d_1 - l_2 \sin\theta_2 + l_3 \cos(\theta_2+\theta_3) \end{pmatrix}$$

for the sake of simplicity, we additionally assume that $d_1 = d_2 = l_2 = l_3 = 1$.
First, compute the Jacobian matrix $\boldsymbol{J}(\boldsymbol{\theta})$ of the robot. Second, explain in a sentence the physical meaning of the Jacobian.

Hint: You can use the abbreviations $c(\theta)$ and $s(\theta)$ for $\cos(\theta)$ and $\sin(\theta)$ values.

d) Singularities [1 Points]

Please explain, what is a singularity in robotics?

e) Workspace [2 Bonus Points]

Sketch schematically a robot arm matching the workspace shown in figure 3 with as few rotational and linear joints as possible. The origin of the coordinate system and the robot base are located at position (0, 0) in the plot. Figure 2 shows an example of the workspace of a 2-DoF robot with q_1 rotational and q_2 linear joints.

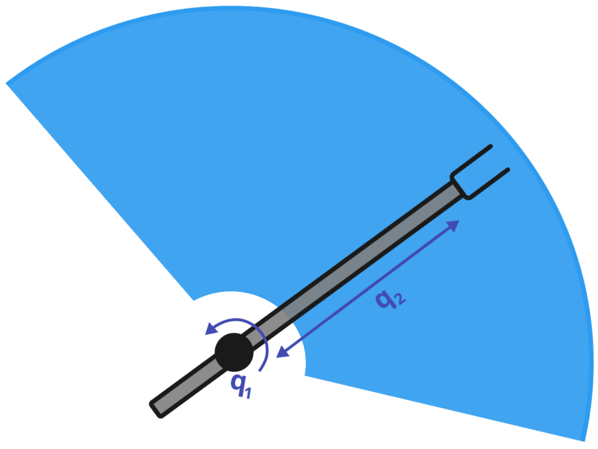

Figure 2: An example of workspace for a robot with one rotational and one linear joint

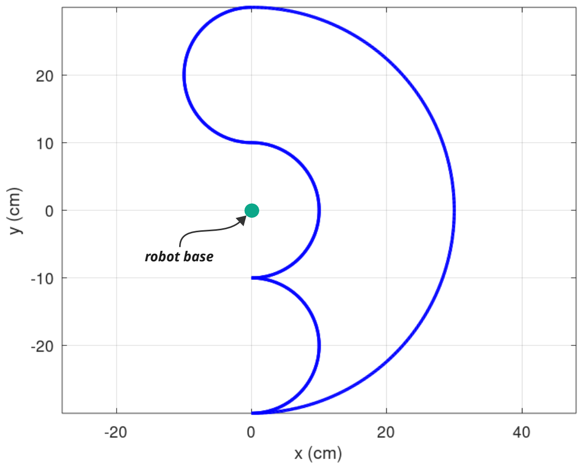

Figure 3: A workspace for a robot

Problem 1.2 Control [30 Points + 2 Bonus]

In robotic locomotion it is common to abstract from the robot by using inverted pendulum models. In this exercise we will use a planar double inverted pendulum to test different control strategies. Our robot can be controlled by specifying the torque $\mathbf{u} = [u_1, u_2]$ of its motors. Consider that in mechanical systems the torque $\boldsymbol{u}$ is a function of the joint positions $\boldsymbol{q}$, velocities $\dot{\boldsymbol{q}}$ and accelerations $\ddot{\boldsymbol{q}}$, as given by

$$\boldsymbol{u} = \boldsymbol{M}(\boldsymbol{q})\ddot{\boldsymbol{q}} + \boldsymbol{c}(\boldsymbol{q}, \dot{\boldsymbol{q}}) + \boldsymbol{g}(\boldsymbol{q}),$$

where $\boldsymbol{M}$ denotes the inertial matrix, $\boldsymbol{c}(\boldsymbol{q}, \dot{\boldsymbol{q}})$ the Coriolis and centripetal forces, and $\boldsymbol{g}$ the gravity terms.

For the programming part of this exercise (starting from question b)), you will use the code that is attached to the exercise. The code is organized as follows: By commenting / uncommenting the specific lines in `example.py` you can choose which controller configuration to run. Already after downloading the files, running `example.py` should generate some plots, however, without the robot moving at all as the controllers currently just apply a constant action of all zeros. Thus, you have to modify the controllers, inside `my_ctrl.py` (for b), c), d)), and inside `my_taskSpace_ctrl.py` for the last part. If you want to dive deeper into the code, you can start from `my_ctl.py` / `my_taskSpace_ctl.py`. Please attach figures of the plots that you generated in here, and provide short snippets of your code! Also, please submit your code with the exercise. Note: please leave all the files unchanged and only modify `example.py` (commenting / uncommenting depending on which configuration you want to run), as well as `my_ctrl.py` and `my_taskSpace_ctrl.py`.

a) Gravity Compensation and Inverse Dynamics Control [5 Points]

Suppose that you would like to create a control law to set the joint angles on the double inverted pendulum model by controlling the torque of the motors. First, provide a PD-feedback control law which additionally compensates for gravity. Second, extend the previous control law to full inverse dynamics control. Lastly, will the PD control law with gravity compensation perfectly reach a desired setpoint when having an imperfect robot model?

b) Comparison of Different Control Strategies [15 Points]

In the following exercise you will investigate the differences of the following control algorithms, P, PD, PID, PD with gravity compensation, and full inverse dynamics. The double pendulum is initiated hanging down, with state $\boldsymbol{q}_{\text{start}} = [-\pi, 0]$. We simulate the system with a time-step $dt = 0.002$ seconds using symplectic Euler integration and run the simulation for $t_{\text{end}} = 3s$.

Implement the control laws by filling the skeleton file `my_ctl.py`. Use the following feedback gains $K_P = 65, K_D = 12, K_I = 0.1$ for the first joint and $K_P = 35, K_D = 8, K_I = 0.1$ for the second one. The target state of the double pendulum is set to $\boldsymbol{q}_{\text{des}} = [-\pi/2, 0]$.

Create plots that compare the different control strategies and analyze the results. It is your choice how to illustrate your results. However, we already provide code to generate the plots, which is sufficient. We thus stronly advise to leave the code untouched (this also makes correction way easier, when we re-run the code that you submitted). Do not forget to include your source code in your solutions.

To modify and/or run the code, see the "Additional Information" section in the README file.

c) Tracking Trajectories [5 Points]

Repeat the same experiment but this time use the provided time-varying target trajectory. Create plots that compare the different control strategies and analyze the results.

d) Tracking Trajectories — High Gains [2 Bonus Points]

Repeat the previous experiment for the PD controller, but now with all gains multiplied by 5. What do you observe? Are there any drawbacks of using high gains?

e) Task Space Control [5 Points]

The robot must now reach a desired position in task space $\boldsymbol{x}_{\text{end}} = [0.0, 1.2]$. In class we derived the Jacobian transpose, Jacobian pseudo-inverse, and Jacobian pseudo-inverse with damping methods. All of them are implemented in `my_taskSpace_ctl.py`. You are asked to implement also the null-space task prioritization method with a null-space resting posture $\boldsymbol{q} = [0, \pi]$. Run the simulation and plot the initial and final configuration of the robot. Then, change the resting posture to $\boldsymbol{q} = [0, -\pi]$ and redo the plots. Analyze in a couple of sentences your observation. Use the same damping coefficient 10^{-6} and include a code snippet to your solutions.