Learning for Adaptive and Reactive Robot Control
Instructions for exercises of lecture 1

Professor: Aude Billard
Contact: aude.billard@epfl.ch

Introduction

The goal of this exercise session is to familiarize you with the simple problem of how to plan a trajectory in free space for a robot arm. You will develop different objective functions and observe the different paths it generates.

This exercise session is composed of two parts. In the first part, you are to solve an exercise with pen and paper. This exercise requires you to find the optimal path for three different cost functions. In the second part of the exercise, you will program these costs functions in MATLAB.

This part of the course follows *exercise* 1.1 and *programming exercises* 1.1 *and* 1.2 of the book "*Learning for Adaptive and Reactive Robot Control: A Dynamical Systems Approach. MIT Press, 2022*".

1 Theoretical exercises [1h]

1.1 Optimal trajectories formulation

Book correspondence: Ex1.1, p8

Given a 2-joint planar robotic arm with unit link length, we are interested in its end-effector position along X direction, i.e. with forward kinematics:

$$x(t) = \cos(q_1(t)) + \cos(q_2(t)) = h(\mathbf{q}(t))$$

A trajectory is a path with $N+1$ waypoints $\{(q_1(0), q_2(0)), (q_1(1), q_2(1)), \ldots, (q_1(N), q_2(N))\}$, and a time-indexing $\{0, t(1), \ldots, t(N)\}$ that specifies the time at which the robot arrives the corresponding waypoint.

Write the optimisation problem to get the trajectories with $N+1$ waypoints and uniform time indexing, i.e. $\delta t = t(n) - t(n-1) = t(N)/N, \forall\, n \in [1, \ldots, N]$ that moves the robot from initial configuration x_0 given by $(q_1(0), q_2(0))$ to a desired position x^* subject to joint maximum increment $\delta q_{\max}$ per second with following desired properties:

(a) Minimizing the time to reach x^*

(b) Following the shortest path in Cartesian space

(c) Following the shortest path in joint space

1.2 Supplementary exercise: (OPTIONAL)

Generalize the above optimization to a K-joint robot with end-effector moving in 3D Cartesian space, given the forward kinematics $h(\cdot) : \mathbb{R}^K \rightarrow \mathbb{R}^3, \mathbf{x} = h(\mathbf{q})$.

2 Programming exercises [1h]

Preliminaries: Instructions for proper installation of MATLAB libraries

For this first session, you will need to download the `book-code` folder from the Github repository. This contains all the code you will need for the semester. Please follow the installation instructions to install the required Matlab Toolboxes.
Each week, you will gain access to the previous week's `solutions` folder on Moodle. You will need to download it, unzip it and place it in its corresponding `lecture-xxx` folder. Your architecture should have the following structure:

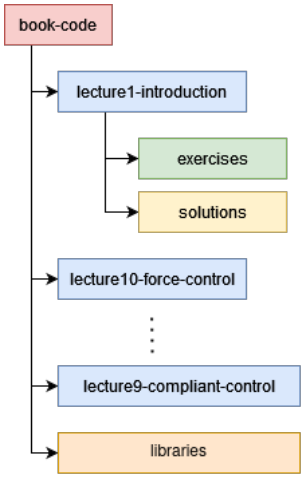

Figure 1: Workspace directory architecture

It is important to have this exact structure, so that the exercises and solution files can properly access the `libraries` folder.

To make sure everything works properly, you will need to use at least MATLAB R2020a, but we recommend R2022b. You will also need to install the following MATLAB toolboxes:

- Control system
- Curve Fitting
- Deep Learning
- Model Predictive Control
- Optimization
- Robotics System
- ROS toolbox
- Signal Processing
- Statistics and Machine Learning

2.1 Programming Exercise 1.1: Computing optimal trajectories

Book correspondence: Programming Ex1.1.2, p.8

The aim of this exercise is to familiarize the reader with basic optimization methods to the motion of a manipulator. Open `chp1_ex1_2.m` *for exercise 1.1. The code generates a drawing with a four-joint manipulator. Edit the code to do the following:*

1. Write the optimization problems of the theoretical exercise for this redundant manipulator moving in three dimensions, hence with a third-dimensional (3D) attractor. Assume that the joints are attached serially. Try moving the target position at different locations in space and compare the trajectory solutions for the three optimizations. Can you find a configuration when the solution to (b) and (c) are identical?
(a) Minimizing the time T_f taken to reach the target
(b) Following the shortest path in Cartesian space
(c) Following the shortest path in joint space

We use the Model Predictive Control class of MATLAB to take care of the numerical optimization. You only have to define a cost function corresponding to each problem of the book's exercise 1.1. To do this, you can fill the `cost` variable with your own combination of the variables `X` and `U` which are the optimal trajectory and the optimal input respectively, stored as row-vectors.

$$\mathbf{X} = \begin{bmatrix} \mathbf{X}_0 \\ \dots \\ \mathbf{X}_N \end{bmatrix} = \begin{bmatrix} q_0^1 & q_0^2 & q_0^3 & q_0^4 \\ & \dots & & \\ q_N^1 & q_N^2 & q_N^3 & q_N^4 \end{bmatrix} \quad \mathbf{U} = \begin{bmatrix} \mathbf{U}_0 \\ \dots \\ \mathbf{U}_N \end{bmatrix} = \begin{bmatrix} \dot{q}_0^1 & \dot{q}_0^2 & \dot{q}_0^3 & \dot{q}_0^4 & T_f \\ & \dots & & & \\ \dot{q}_N^1 & \dot{q}_N^2 & \dot{q}_N^3 & \dot{q}_N^4 & T_f \end{bmatrix} \tag{1}$$

Here $[q^1, q^2, q^3, q^4]$ are the four joint positions of the robot, and $[\dot{q}^1, \dot{q}^2, \dot{q}^3, \dot{q}^4]$ are the four joint velocities. T_f is the final time of the trajectory. This trajectory is made of N points, that you can retrieve with:

```
% Number of waypoints of the trajectory:
N = data.PredictionHorizon
```

The three functions to modify are `minimumTime()`, `minimumTaskDistance()` and `minimumJointDistance()` at the bottom of the file. You can play with different cost functions and launch the appropriate MATLAB section to see its effect on the final trajectory.
WARNING Some objective function might take a few seconds to be solved.

You can use the Jacobian matrix $J(\mathbf{q})$ to compute the small displacement $\delta\mathbf{x}$ from the joint velocity: $\delta\mathbf{x} = J(\mathbf{q}) * \dot{\mathbf{q}} * \delta t$. The Jacobian matrix can be computed with the following MATLAB function:

```
% Input 4x1 joint position vector q, output 3x4 Jacobian matrix:
jacobian = robot.fastJacobian(q);
```

2.2 Programming Exercise 1.2: Disturbed trajectory [TO BE CONTINUED IN WEEK 2 OF THE CLASS]

Book correspondence: Programming Ex1.2, p.9

The aim of this programming exercise is to determine how to recompute a path under disturbances with the optimization techniques seen in programming exercise 1.1. Open `chp1_ex2.m`.

1. Initialize the end-effector at one of the locations chosen in the previous programming exercise and generate a path following optimization (c).

You can modify the variable `initial_joint_configuration` to define the initial configuration. Fill the cost function at the end of the script to solve for minimum joint distance like in exercise 1.

2. Generate a disturbance midpath by suddenly displacing one of the joints of the robot by 10 degrees.

Choose an index mid-trajectory (`disturbance_idx`), and apply the disturbance at this time by modifying the joint configuration at this index (`q_mid`). The updated joint configuration will be used as the initial configuration for the next optimization.

3. Redo the optimization to generate a new path to the target using the residual time to the target.

A new trajectory is generated from the disturbed configuration, and stored in `optimal_solution_after_disturbance`. Create the complete trajectory by stitching the two solutions (line 72-76). The function `optimal_control.solveOptimalTrajectory()` returns the solution as a structure with the following content:

- `Topt`: Time vector as 1xN vector
- `Xopt`: Joint position as 4xN array.
- `Yopt`: Cartesian position as 3xN array.
- `MVopt`: Joint speed and final time as 5xN array.

Here N is the trajectory length (horizon length).

$$\mathbf{X}_{opt} = \begin{bmatrix}\mathbf{X}_{opt,0} & \dots & \mathbf{X}_{opt,N}\end{bmatrix} = \begin{bmatrix} q_0^1 & & q_N^1 \\ q_0^2 & \dots & q_N^2 \\ q_0^3 & & q_N^3 \\ q_0^4 & & q_N^4 \end{bmatrix} \tag{2}$$

$$\mathbf{Y}_{opt} = \begin{bmatrix}\mathbf{Y}_{opt,0} & \dots & \mathbf{Y}_{opt,N}\end{bmatrix} = \begin{bmatrix} x_0 & & x_N \\ y_0 & \dots & y_N \\ z_0 & & z_N \end{bmatrix} \tag{3}$$

$$\mathbf{MV}_{opt} = \begin{bmatrix}\mathbf{MV}_{opt,0} & \dots & \mathbf{MV}_{opt,N}\end{bmatrix} = \begin{bmatrix} \dot{q}_0^1 & & \dot{q}_N^1 \\ \dot{q}_0^2 & & \dot{q}_N^2 \\ \dot{q}_0^3 & \dots & \dot{q}_N^3 \\ \dot{q}_0^4 & & \dot{q}_N^4 \\ T_f & & T_f \end{bmatrix} \tag{4}$$

2.3 Programming Exercise 1.3: Closed-from trajectory [Optional]

Book correspondence: Programming Ex1.1.1, p.8

1. Compute a closed-form time-dependent trajectory generator based on a third-order polynomial.

You can add your implementation line 51 in `chp1_ex1_1.m`. You have to manually create a trajectory in position (Cartesian space) that starts at `initial_position` and ends at `target_position`, based on a third-order polynomial (see figure 2).

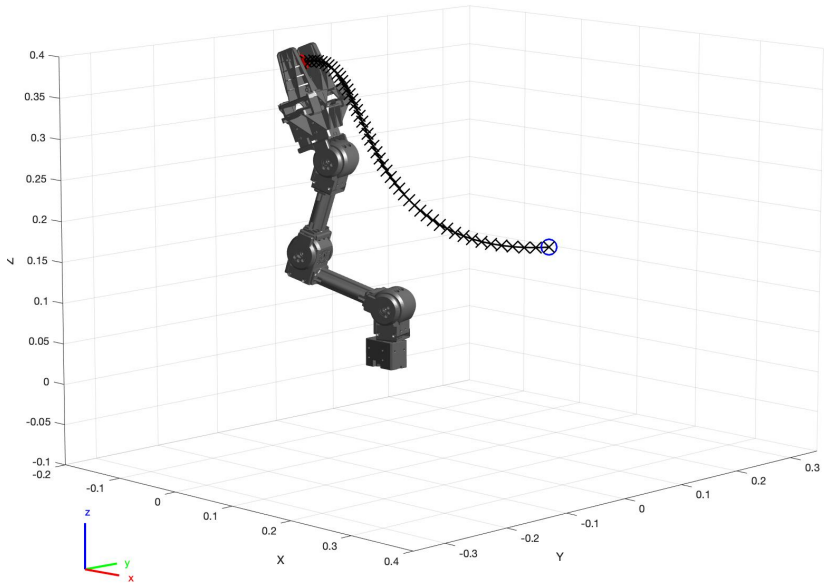

Figure 2: Example of a Cartesian trajectory based on a third-order polynomial. The red circle is the `target_position`, and the blue circle is the `initial_position`.

References

[1] Aude Billard, Sina Mirrazavi, and Nadia Figueroa. *Learning for Adaptive and Reactive Robot Control: A Dynamical Systems Approach*. MIT press, 2022.