Robotic Systems II: Homework III

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

December 8 2025

Introduction

The homework concerns the modeling and control of non-linear systems by combining offline Trajectory Optimization and online Time-Variant LQR (TVLQR). In particular, we are going to model, simulate and control a swing-up behavior for a double-pendulum system (Fig. 1). The homework is split into three parts/sub-questions: 1) modeling and discretization of the system, 2) formulating the task and solving it as a non-linear trajectory optimization problem, and 3) controlling the system under disturbances via TVLQR.

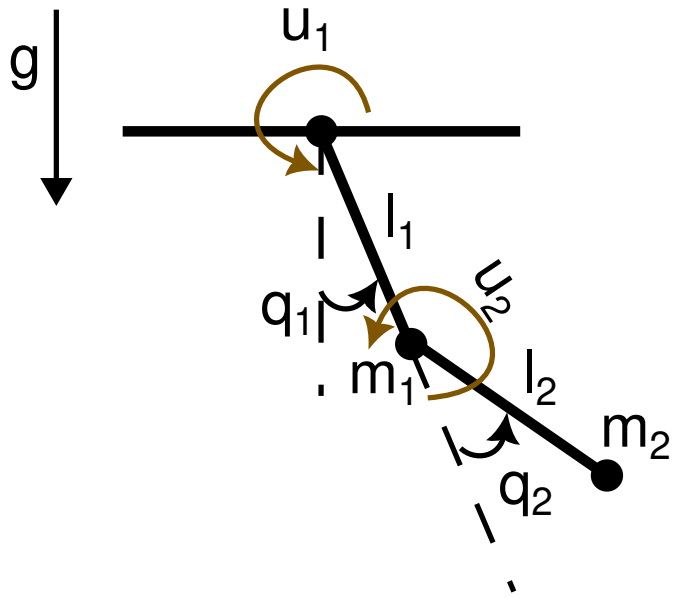

Figure 1: Double Pendulum System.

1 Modeling (20%)

For this homework we are going to use the double pendulum system (Fig. 1). The system is a two-link (masses m_1 and m_2, and lengths l_1 and l_2 respectively), robot (see Fig. 1). Both joints exert torque (u_1 and u_2 respectively). The system has four continuous state variables: two joint positions (q_1, q_2) and two joint velocities $(\dot{q}_1, \dot{q}_2)$.
The state of the system is as follows:

$$\boldsymbol{x} = \begin{bmatrix} q_1 \\ q_2 \\ \dot{q}_1 \\ \dot{q}_2 \end{bmatrix} \in \mathbb{R}^4$$

And the control inputs:

$$\boldsymbol{u} = \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} \in \mathbb{R}^2$$

The continuous dynamics of the system are given by the following equations:

$$\dot{\boldsymbol{x}} = f(\boldsymbol{x}, \boldsymbol{u}) = \begin{bmatrix} \dot{q}_1 \\ \dot{q}_2 \\ \ddot{q}_1 \\ \ddot{q}_2 \end{bmatrix} \in \mathbb{R}^4$$

By defining $\boldsymbol{q} = \begin{bmatrix} q_1 \\ q_2 \end{bmatrix}$, we can derive the following equation:

$$\ddot{\boldsymbol{q}} = \boldsymbol{M}^{-1}\Big(\boldsymbol{u} - \boldsymbol{C}(\boldsymbol{q}, \dot{\boldsymbol{q}})\dot{\boldsymbol{q}} + \boldsymbol{G}(\boldsymbol{q})\Big)$$

where

$$\boldsymbol{M} = \begin{bmatrix} l_1^2 m_1 + l_2^2 m_2 + l_1^2 m_2 + 2 l_1 m_2 l_2 \cos q_2 & l_2^2 m_2 + l_1 m_2 l_2 \cos q_2 \\ l_2^2 m_2 + l_1 m_2 l_2 \cos q_2 & l_2^2 m_2 \end{bmatrix} \in \mathbb{R}^{2\times 2}$$

$$\boldsymbol{C} = \begin{bmatrix} -2\dot{q}_2 l_1 m_2 l_2 \sin q_2 & -\dot{q}_2 l_1 m_2 l_2 \sin q_2 \\ \dot{q}_1 l_1 m_2 l_2 \sin q_2 & 0 \end{bmatrix} \in \mathbb{R}^{2\times 2}$$

$$\boldsymbol{G} = \begin{bmatrix} -g m_1 l_1 \sin q_1 - g m_2 (l_1 \sin q_1 + l_2 \sin(q_1 + q_2)) \\ -g m_2 l_2 \sin(q_1 + q_2) \end{bmatrix} \in \mathbb{R}^{2\times 1}$$

and $g = 9.81\, m/s^2$, $m_1 = m_2 = 1\, kg$, $l_1 = l_2 = 0.5\, m$.

In this part, you are asked to:

1. Write a function in Python called `dynamics()` that takes as input the state $\boldsymbol{x} \in \mathbb{R}^{4\times 1}$ and controls $\boldsymbol{u} \in \mathbb{R}^{2\times 1}$ and returns the $\dot{\boldsymbol{x}} \in \mathbb{R}^{4\times 1}$
2. Create a simple visualization of the system (e.g. by using the `matplotlib` library); you are free to use any library for the visualization
3. Discretize the system using **Runge-Kutta 4th Order integration**
4. Make sure that the integration works and is correct

For all the above (apart from the visualizations), you are allowed to use only native Python functions/classes, and the `numpy` and `jax` libraries.

2 Trajectory Optimization (40%)

In this part, you are asked to:

1. Formulate the problem as a non linear trajectory optimization using either numerical quadrature (using the trapezoidal or hermite simpson rule) or the cubic splines formulation
2. Define an appropriate cost function and solve the problem using the `cyipopt` library
3. Visualize the resulting trajectory

Use the following bounds/constraints:

- **Initial state:** $\boldsymbol{x}_0 = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$

- **Final state:** $\boldsymbol{x}_f = \begin{bmatrix} \pi \\ 0 \\ 0 \\ 0 \end{bmatrix}$
- **Control inputs:** $\begin{bmatrix} -10 \\ -10 \end{bmatrix} \leq \boldsymbol{u} \leq \begin{bmatrix} 10 \\ 10 \end{bmatrix}$
- **Total time:** $5\,s$
- **Number of knot points:** 101
- **Timestep (dt):** $0.05\,s$

For all the above (apart from the visualizations), you are allowed to use only native Python functions/classes, and the `numpy` and `jax` libraries.

3 Controlling via TVLQR (40%)

In this part, you are asked to implement a discrete-time Time-Variant LQR (TVLQR) controller for the system. In particular:

1. Write a function in Python called `my_controller()` that takes as input the current state of the system, and outputs the controls to be applied to the system. You should linearize around the trajectory that the previous step outputs.
2. Add noise to the observations (Gaussian with zero mean and diagonal covariance). How does increasing the noise affect the controller?

For all the above (apart from the visualizations), you are allowed to use only native Python functions/classes, and the `numpy`, `jax` and `proxsuite` libraries.

4 Deliverables

- 3 python files with commented code (one for each subquestion)[1]
- A detailed report

Bonus Points

1. There is a 15% bonus if you do not use a list of $\boldsymbol{K}$ matrices, but perform some interpolation/regression such that we can continuously control the robot.
2. There is a 5% bonus if one writes the derivatives by hand and does not use `jax` (autodiff).

[1] Jupyter notebooks are accepted as well.