UNIVERSITÀ DI BOLOGNA

School of Engineering

Master Degree in Automation Engineering

Optimal Control

Course Project 1
Optimal Control of a Flexible Robotic Arm

Professor: **Giuseppe Notarstefano**

Students:
Vittorio Caputo
Marco Drammis
Sunil Shrikant Shikalgar

Academic year 2024/2025

Abstract

Flexible robotic arms are gaining prominence in advanced robotics due to their versatility and precision, with applications in medical assistance, automation, and precision manufacturing. However, their inherent flexibility poses significant challenges in control and trajectory planning. This project focuses on the optimal control of a flexible robotic arm, modeled as a planar two-link system with torque applied at the first joint.

The study begins with the discretization of the robot's dynamics, forming a mathematical foundation for simulation and control. Trajectory generation tasks include computing equilibrium states and designing smooth transitions between them using Newton's optimization algorithm, as well as tackling the challenging swing-up motion. To ensure accurate trajectory tracking, both Linear Quadratic Regulator (LQR) and Model Predictive Control (MPC) strategies are implemented, with their robustness tested under perturbed conditions. Finally, an animation of the robotic arm executing optimal control tasks is developed, showcasing the effectiveness of the proposed methods. This work contributes to advancing control strategies for flexible robotic systems in dynamic environments.

Contents

Introduction

Flexible robotic arms are increasingly significant in advanced robotics, offering versatility and precision for tasks in fields such as medical assistance, automation, and precision manufacturing. These robotic systems, characterized by their lightweight structures and flexibility, enable improved adaptability for complex operations. However, their inherent flexibility also introduces challenges in control and trajectory planning, necessitating sophisticated strategies to ensure stability, accuracy, and efficiency.

This project focuses on the **optimal control of a flexible robotic arm**, modeled as a planar two-link system with torque applied at the first joint. The tasks undertaken are detailed as follows:

Task 0: Problem Setup

The first step involves discretizing the robot's dynamics to derive discrete-time state-space equations. This forms the mathematical foundation for simulating and controlling the robotic arm. A coded dynamics function ensures an accurate representation of the system's behavior for subsequent tasks.

Task 1: Trajectory Generation (I)

Two equilibrium states, representing the start and end points of motion, are computed. A reference curve is generated to connect these equilibria, ensuring smooth and symmetric transitions. Newton's optimization algorithm is used to generate the optimal transition trajectory between these two equilibria.

Task 2: Trajectory Generation (II)

In this task, we developed the challenging task of swing up by also exploiting the Newton's method.

Task 3: Trajectory Tracking via LQR

In this task, the dynamics of the robotic arm are linearized around the optimized trajectory. The LQR algorithm is applied to develop an optimal feedback controller for tracking a reference optimal trajectory. The robustness of this controller is evaluated under perturbed initial conditions, demonstrating its ability to maintain precise trajectory tracking.

Task 4: Trajectory Tracking via MPC

The Model Predictive Control (MPC) technique is employed to track a reference optimal trajectory. The method's effectiveness is tested by introducing perturbations to the initial state.

Task 5: Animation

An animation of the robot executing task 3 has been developed.

Contributions

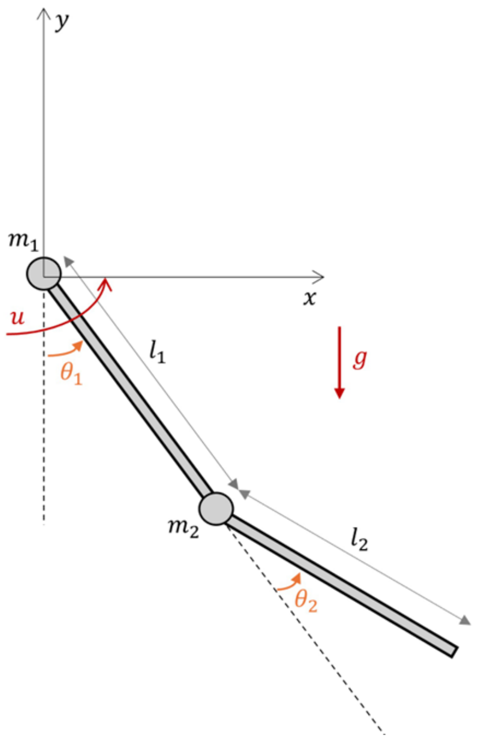
This project contributes by showcasing the power and effectiveness of optimal control techniques in addressing the challenges posed by underactuated systems. Through the application of various optimal control methods, such as Newton's optimization, Linear Quadratic Regulator (LQR), and Model Predictive Control (MPC), it demonstrates how these strategies can be effectively employed to maintain stability, precision, and performance in robotic systems with fewer actuators than degrees of freedom, particularly in flexible robotic arms.

Chapter 1

Task 0 - Problem setup

This chapter discusses the setup of the problem. In particular, in order to design an optimal trajectory for our flexible robotic it's necessary to derive the state-space model of the manipulator from the given Euler-Lagrange dynamic model.

1.1 Euler-Lagrange dynamic model



The state space consists in

$$\mathbf{x} = \begin{bmatrix} \theta_1 & \theta_2 & \dot{\theta}_1 & \dot{\theta}_2 \end{bmatrix}^T,$$

where θ_1 represents the angle of the first link with respect to the vertical direction, θ_2 represents the angle of the second link with respect to the first link, $\dot{\theta}_1$ and $\dot{\theta}_2$ are the angular rates of changes associated to θ_1 and θ_2, respectively.
The input is the torque u on the first link. The system dynamics are described as:

$$\mathbf{M}(\theta_1, \theta_2) \begin{bmatrix} \ddot{\theta}_1 \\ \ddot{\theta}_2 \end{bmatrix} + \mathbf{C}(\theta_1, \theta_2, \dot{\theta}_1, \dot{\theta}_2) \begin{bmatrix} \dot{\theta}_1 \\ \dot{\theta}_2 \end{bmatrix} + \mathbf{F} \begin{bmatrix} \dot{\theta}_1 \\ \dot{\theta}_2 \end{bmatrix} + \mathbf{G}(\theta_1, \theta_2) = \begin{bmatrix} u \\ 0 \end{bmatrix},$$

where

$$\mathbf{M} = \begin{bmatrix} l_1 + l_2 + m_1 r_1^2 + m_2\left(l_1^2 + r_2^2\right) + 2m_2 l_1 r_2 \cos(\theta_2) & l_2 + m_2 r_2^2 + m_2 l_1 r_2 \cos(\theta_2) \\ l_2 + m_2 r_2^2 + m_2 l_1 r_2 \cos(\theta_2) & l_2 + m_2 r_2^2 \end{bmatrix},$$

$$\mathbf{C} = \begin{bmatrix} -m_2 l_1 r_2 \sin(\theta_2)\left(\dot{\theta}_2 + 2\dot{\theta}_1\right) \\ m_2 l_1 r_2 \sin(\theta_2)\dot{\theta}_1 \end{bmatrix},$$

$$\mathbf{G} = \begin{bmatrix} g\left(m_1 r_1 + m_2 l_1\right)\sin(\theta_1) + g m_2 r_2 \sin(\theta_1 + \theta_2) \\ g m_2 r_2 \sin(\theta_1 + \theta_2) \end{bmatrix},$$

$$\mathbf{F} = \begin{bmatrix} f_1 & 0 \\ 0 & f_2 \end{bmatrix}.$$

Here, r_1 and r_2 are the distances between the pivot points of the link and their center of mass, m_1 and m_2 are the respective masses, f_1 and f_2 are the viscous friction coefficients, and g is the gravitational acceleration. The parameters for Set 3 are provided in the table below.

Parameter	**Value**
m_1	1.5
m_2	1.5
l_1	1
l_2	1
r_1	0.5
r_2	0.5
I_1	0.33
g	9.81
f_1	0.1
f_2	0.1

Table 1.1: Model parameters for Set 3.

1.2 Continuous time state-space model

The state-space consists of the vector $\mathbf{x} = [\theta_1, \theta_2, \dot{\theta}_1, \dot{\theta}_2]^\top = [x_1, x_2, x_3, x_4]^\top$, as already discussed in the previous section.

The first two state-space equations are trivial, since the angular velocity is simply the time derivative of the angular position:

$$\dot{x}_1 = x_3, \quad \dot{x}_2 = x_4$$

The remaining two state-space equations can be derived from the Euler-Lagrange dynamic model:

$$M(x_1, x_2)\begin{bmatrix} \dot{x}_3 \\ \dot{x}_4 \end{bmatrix} + C(x_1, x_2, x_3, x_4) + F\begin{bmatrix} x_3 \\ x_4 \end{bmatrix} + G(x_1, x_2) = \begin{bmatrix} u \\ 0 \end{bmatrix}$$

It turns out that:

$$\begin{bmatrix} \dot{x}_3 \\ \dot{x}_4 \end{bmatrix} = -M(x_1, x_2)^{-1}\left[C(x_1, x_2, x_3, x_4)\begin{bmatrix} x_3 \\ x_4 \end{bmatrix} + F\begin{bmatrix} x_3 \\ x_4 \end{bmatrix} + G(x_1, x_2) - \begin{bmatrix} u \\ 0 \end{bmatrix}\right]$$

Defining the following constants for compact notation:

$$
\begin{aligned}
a &= I_1 + I_2 + m_1 r_1^2 + m_2(l_1^2 + r_2^2) \\
b &= m_2 l_1 r_2 \\
c &= I_2 + m_2 r_2^2 \\
d &= g(m_1 r_1 + m_2 l_1) \\
e &= g m_2 r_2
\end{aligned}
$$

After some computations, it is possible to obtain the following expressions for $\dot{x}_3$ and $\dot{x}_4$:

$$
\begin{aligned}
\dot{x}_3 = {} & \frac{-c}{c(a-c) - b^2\cos^2(x_2)} \left[-b x_4 \sin(x_2)(x_4 + 2x_3) + f_1 x_3 + d\sin(x_1) \right. \\
& \left. + e\sin(x_1 + x_2) - u \,\right] + \frac{c + b\cos(x_2)}{c(a-c) - b^2\cos^2(x_2)} \left[b\sin(x_2) x_3^2 + f_2 x_4 + e\sin(x_1 + x_2) \right]
\end{aligned}
$$

$$
\begin{aligned}
\dot{x}_4 = {} & + \frac{c + b\cos(x_2)}{c(a-c) - b^2\cos^2(x_2)} \left[-b x_4 \sin(x_2)(x_4 + 2x_3) + f_1 x_3 + d\sin(x_1) \right. \\
& \left. + e\sin(x_1 + x_2) - u \,\right] - \frac{a + 2b\cos(x_2)}{c(a-c) - b^2\cos^2(x_2)} \left[b\sin(x_2) x_3^2 + f_2 x_4 + e\sin(x_1 + x_2) \right]
\end{aligned}
$$

1.3 Discrete time state-space model

In order to derive the discrete time state-space model from the continuous time one, it's possible to exploit the Euler discretization method that approximates the continuous time derivative with a finite difference:

$$
\dot{x}(t) = \frac{x(t+1) - x(t)}{dt}
$$

where dt is the discretization step.

We can apply Euler discretization method to the continuous time state-space equation to get the following discrete time state space model:

$$
\begin{cases}
x_{1,t+1} = x_{1,t} + dt x_{3,t} \\
x_{2,t+1} = x_{2,t} + dt x_{4,t} \\
x_{3,t+1} = x_{3,t} + dt \dfrac{-c}{c(a-c) - b^2\cos^2(x_{2,t})} \left[-b x_{4,t} \sin(x_{2,t})(x_{4,t} + 2x_{3,t}) + f_1 x_{3,t} + d\sin(x_{1,t}) \right. \\
\qquad \left. + e\sin(x_{1,t} + x_{2,t}) - u \,\right] + dt \dfrac{c + b\cos(x_{2,t})}{c(a-c) - b^2\cos^2(x_{2,t})} \left[b\sin(x_{2,t}) x_{3,t}^2 + f_2 x_{4,t} \right. \\
\qquad \left. + e\sin(x_{1,t} + x_{2,t}) \,\right] \\
\\
x_{4,t+1} = x_{4,t} + dt \dfrac{c + b\cos(x_2)}{c(a-c) - b^2\cos^2(x_2)} \left[-b x_{4,t} \sin(x_{2,t})(x_{4,t} + 2x_{3,t}) + f_1 x_{3,t} + d\sin(x_{1,t}) \right. \\
\qquad \left. + e\sin(x_{1,t} + x_{2,t}) - u \,\right] - dt \dfrac{a + 2b\cos(x_{2,t})}{c(a-c) - b^2\cos^2(x_{2,t})} \left[b\sin(x_{2,t}) x_{3,t}^2 + f_2 x_{4,t} \right. \\
\qquad \left. + e\sin(x_{1,t} + x_{2,t}) \,\right]
\end{cases}
$$

Where in compact form we can write: $x_{t+1} = f(x_t, u_t)$.

This is what is implemented in the function $f(x, u)$ in the dynamics.py file.

1.4 Linearization matrices

In the same dynamics.py file there is also the implementation of the linearization matrices A and B, that will be used in the next tasks.

The state linearization matrix A is defined as the partial derivative of the function f with respect to x:

$$A(x,u) = \frac{\partial f(x,u)}{\partial x} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \frac{\partial f_1}{\partial x_3} & \frac{\partial f_1}{\partial x_4} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \frac{\partial f_2}{\partial x_3} & \frac{\partial f_2}{\partial x_4} \\ \frac{\partial f_3}{\partial x_1} & \frac{\partial f_3}{\partial x_2} & \frac{\partial f_3}{\partial x_3} & \frac{\partial f_3}{\partial x_4} \\ \frac{\partial f_4}{\partial x_1} & \frac{\partial f_4}{\partial x_2} & \frac{\partial f_4}{\partial x_3} & \frac{\partial f_4}{\partial x_4} \end{bmatrix}$$

The input linearization matrix B is defined as the partial derivative of the function f with respect to u:

$$B(x,u) = \frac{\partial f(x,u)}{\partial u} = \begin{bmatrix} \frac{\partial f_1}{\partial u} \\ \frac{\partial f_2}{\partial u} \\ \frac{\partial f_3}{\partial u} \\ \frac{\partial f_4}{\partial u} \end{bmatrix}$$

This is what is implemented in the functions A(x,u) and B(x,u) in the dynamics.py file.

Chapter 2

Task 1 - Trajectory generation (I)

For Task 1, the goal is to compute two equilibria of the system, define a reference trajectory between them, and determine the optimal transition between these points using a closed-loop version of the Newton-like algorithm for optimal control.

2.1 Equilibria of the system

Considering the Euler-Lagrange dynamic model:

$$M(x_1, x_2)\begin{bmatrix}\dot{x}_3\\ \dot{x}_4\end{bmatrix} + C(x_1, x_2, x_3, x_4) + F\begin{bmatrix}x_3\\ x_4\end{bmatrix} + G(x_1, x_2) = \begin{bmatrix}u\\ 0\end{bmatrix}$$

To compute the equilibria of the system, it is possible to set to zero the angular velocities x_3 and x_4 (and so also the angular accelerations $\dot{x}_3$ and $\dot{x}_4$). As a consequence, we get the following result:

$$\begin{bmatrix}u_E\\ 0\end{bmatrix} = G(x_{1E}, x_{2E}) = \begin{bmatrix}d\sin(x_{1E}) + e\sin(x_{1E} + x_{2E})\\ e\sin(x_{1E} + x_{2E})\end{bmatrix}$$

Solving this system of equations, we obtain that at the equilibrium the following relations must be satisfied:

$$\begin{cases}u_E = d\sin(x_{1E})\\ x_{2E} = k\pi - x_{1E}\end{cases} \qquad k = 0, 1, 2...$$

In the design of the reference curve we will choose $x_{2E} = -x_{1E}$.

2.2 Reference curve

2.2.1 Cubic polynomial transition

In order to define a smooth transition between two equilibria, pos_start and pos_end, for our reference curve, we decided to use a cubic polynomial function, The polynomial function is defined as a function of normalized time t, where:

$$t \in [0, 1]$$

The position at time t, denoted by $\theta(t)$, is given by the cubic polynomial equation in parametric form:

$$\theta(t) = \text{pos\_start} + (\text{pos\_end} - \text{pos\_start}) \cdot \left(3t^2 - 2t^3\right)$$

The trajectory satisfies the following conditions:

- At $t = 0$:
$$\theta(0) = \text{pos\_start}, \quad \dot{\theta}(0) = 0$$
This ensures that the trajectory starts at pos_start with zero initial velocity.
- At $t = 1$:
$$\theta(1) = \text{pos\_end}, \quad \dot{\theta}(1) = 0$$
This ensures that the trajectory ends at pos_end with zero final velocity.

The coefficients of the cubic polynomial are chosen specifically to satisfy the boundary conditions on position and velocity.

2.2.2 Reference curve design

The reference state-input curve is constructed defining two long constant parts between two equilibria with a smooth and shorter transition in between realized with the cubic polynomial function described in the previous subsection.

The total time, T, is divided into three distinct segments:

- The initial and final constant segments each have a duration of $t\_const = \frac{T - t\_transition}{2}$
- The transition segment has a duration of $t\_transition = \frac{T}{4}$

The reference curve for the state x_1 is the following:

- In the first segment segment, x_{1ref} is set constant to a certain equilibrium x_{1eq_1} chosen as $-20°$
- In the second segment, we have a cubic polynomial transition from the first equilibrium $-20°$ to the second equilibrium x_{1eq_2} chosen as $30°$
- In the third segment, x_{1ref} is set constant to the second equilibrium value chosen as $30°$

The reference curve for the state x_2 is the following:

- In the first segment segment, x_{2ref} is set constant to $-x_{1eq_1}$ according to the equilibrium equations
- In the second segment, we have a cubic polynomial transition from the first equilibrium to the second equilibrium
- In the third segment, x_{2ref} is set constant to the second equilibrium $-x_{1eq_2}$ according to the equilibrium equations

The reference curve for the input u is the following:

- In the first segment segment, u_{ref} is set constant to $d \sin(x_{1eq_1})$ according to the equilibrium equations
- In the second segment, we have a cubic polynomial transition from the first equilibrium to the second equilibrium
- In the third segment, u_{ref} is set constant to the second equilibrium $d \sin(x_{1eq_2})$ according to the equilibrium equations

Regarding the angular velocities x_3 and x_4, for them a constant zero reference is chosen.

2.3 Newton's method

The reference curve that we've designed up to now, it's not a trajectory, it doesn't satisfy the dynamics of the system, so we can use the Newton's method to find the optimal trajectory that minimizes a certain cost function and satisfies also the constraint due to the dynamics.

2.3.1 Trajectory generation cost function

The trajectory generation cost function is designed to evaluate the deviation between the system's state and control inputs, compared to a reference trajectory, over both the stage and terminal phases. It consists of two components: the stage cost and the terminal cost.

Stage Cost: The stage cost measures the difference between the current state x and the reference state x_{ref}, as well as the difference between the current control input u and the reference control input u_{ref}, at each time step. This cost function penalizes the deviations from the desired trajectory, and it is calculated using quadratic terms with weighting matrices Q_t and R_t for the states and inputs, respectively. The stage cost function is defined as:

$$\ell_t(x_t, u_t) = 0.5 \cdot (x_t - x_{\text{ref}})^T Q_t (x_t - x_{\text{ref}}) + 0.5 \cdot (u_t - u_{\text{ref}})^T R_t (u_t - u_{\text{ref}})$$

Terminal Cost: The terminal cost evaluates the difference between the final state x_T and the reference terminal state $x_{T_{\text{ref}}}$. It is intended to penalize any deviation from the desired final state at the end of the trajectory. The terminal cost function is given by:

$$\ell_T(x_T) = 0.5 \cdot (x_T - x_{T_{\text{ref}}})^T Q_T (x_T - x_{T_{\text{ref}}})$$

2.3.2 Regularized Newton's method algorithm (closed loop)

The main steps of Newton's method algorithm are the following:

Initialization: Choice of an initial guess trajectory $(\mathbf{x}^0, \mathbf{u}^0)$.

For $k = 0, 1, \ldots$:
Step 1: Descent direction computation

Evaluate $\nabla_1 f_t(x_t^k, u_t^k)$, $\nabla_2 f_t(x_t^k, u_t^k)$, $\nabla_1 \ell_t(x_t^k, u_t^k)$, $\nabla_2 \ell_t(x_t^k, u_t^k)$, $\nabla \ell_T(x_T^k)$.
Solve backward the co-state equation with $\lambda_T^k = \nabla \ell_T(x_T^k)$ and compute Q_t^k, R_t^k, S_t^k, and Q_T^k.
Compute K_t^k, σ_t^k, for all $t = 0, \ldots, T-1$, solving the following lqr affine optimization problem:

$$\min_{\Delta x, \Delta u} \sum_{t=0}^{T-1} \begin{bmatrix} \nabla_1 \ell_t(x_t^k, u_t^k) \\ \nabla_2 \ell_t(x_t^k, u_t^k) \end{bmatrix}^T \begin{bmatrix} \Delta x_t \\ \Delta u_t \end{bmatrix} + \frac{1}{2} \begin{bmatrix} \Delta x_t \\ \Delta u_t \end{bmatrix}^T \begin{bmatrix} Q_t^k & S_t^{kT} \\ S_t^k & R_t^k \end{bmatrix} \begin{bmatrix} \Delta x_t \\ \Delta u_t \end{bmatrix} + \nabla \ell_T(x_T^k)^T \Delta x_T + \frac{1}{2} \Delta x_T^T Q_T^k \Delta x_T$$

subject to the dynamics constraint:

$$\Delta x_{t+1}^{k+1} = \nabla_1 f_t(x_t^k, u_t^k)^T \Delta x_t^k + \nabla_2 f_t(x_t^k, u_t^k)^T \Delta u_t^k, \quad \Delta x_0 = 0$$

Step 2: New state-input trajectory computation using Armijo step-size selection rule

Forward integration (closed-loop), for all $t = 0, \ldots, T-1$, with:

$$x_0^{k+1} = x_0$$

$$u_t^{k+1} = u_t^k + K_t^k(x_t^{k+1} - x_t^k) + \gamma^k \sigma_t^k$$
$$x_{t+1}^{k+1} = f_t(x_t^{k+1}, u_t^{k+1})$$

This is basically what has been implemented in the `Newton_reg` function in the Reg_Newton_Method.py file. Actually, the regularized version has been implemented by considering the following matrices for the LQR affine problem to be solved at each iteration:

$$\tilde{Q}_t^k = \nabla_{11}^2 \ell_t(x_t^k, u_t^k), \;\; \tilde{R}_t^k = \nabla_{22}^2 \ell_t(x_t^k, u_t^k), \;\; \tilde{S}_t^k = \nabla_{12}^2 \ell_t(x_t^k, u_t^k), \;\; \tilde{Q}_T^k = \nabla^2 \ell_T(x_T^k)$$

In few words, only the hessian with respect the cost function is considered (not the dynamic function f). Moreover, according to the chosen cost function described in the previous subsection , the matrices to be used for the LQR affine problem are the following:

$$\tilde{Q}_t^k = Q_t, \;\; \tilde{R}_t^k = R_t, \;\; \tilde{S}_t^k = 0, \;\; \tilde{Q}_T^k = Q_T$$

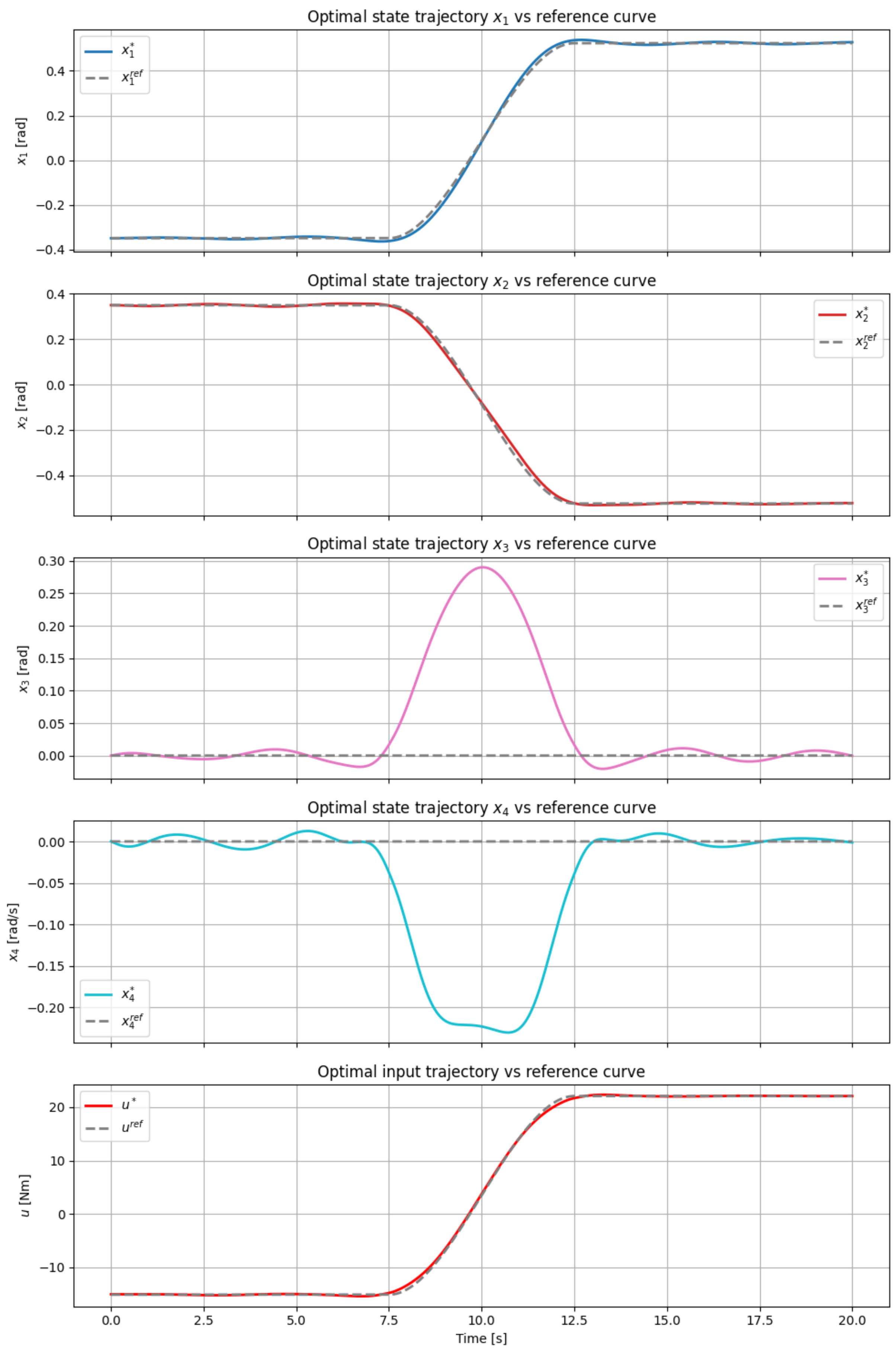
Where Q_t, R_t and Q_T are the weight matrices of the trajectory generation cost function

2.3.3 Optimal trajectory and desired curve plots

In the plots below, it is possible to observe the result of the Newton's method where there is the comparison between the optimal state-input trajectory generated by the algorithm (continuous line) and the desired state-input curve (dashed line). These results have been obtained by considering a terminal condition based on the descent direction norm. In particular, we have decided to stop the Newton's algorithm when the descent direction norm is less than 10^{-5}. According to this stopping criterium, we got a number of iterations equal to 13. We chose as initial state-input guess given the first value of the reference curve and costant for all the time duration of 20s. Furthermore, the chosen weight matrices for the trajectory generation cost function are the following:

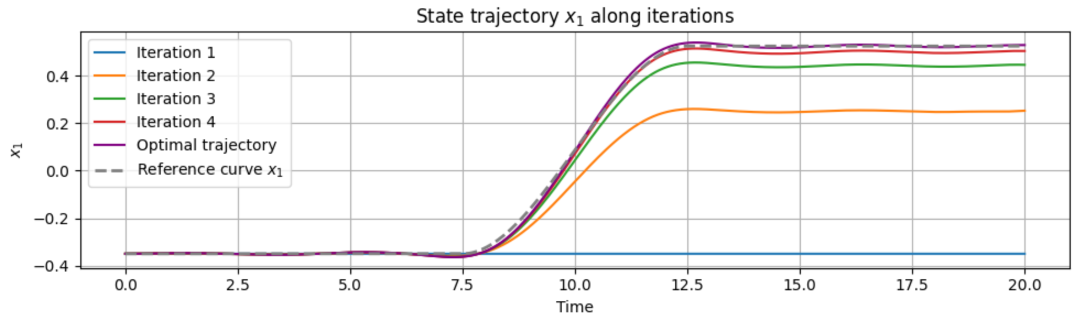
$$Q_t = \text{diag}(1000, 1000, 10, 10)$$
$$R_t = 1 \cdot I_{n_i}$$
$$Q_T = Q_t$$

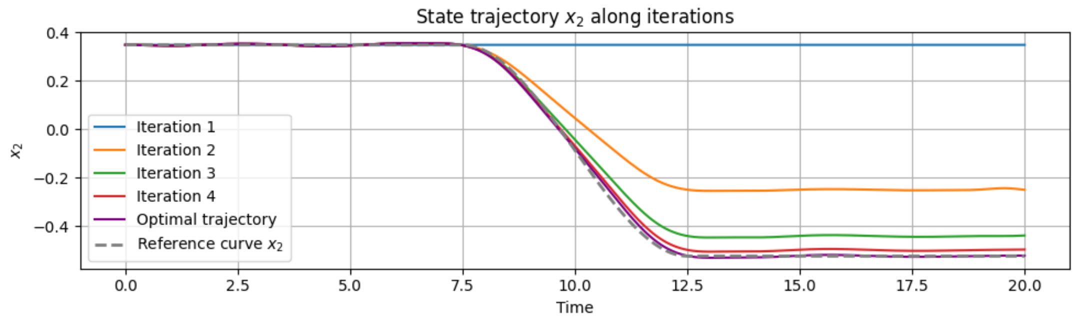
An higher weight has been chosen for the angular positions x_1 and x_2, since are the references that we are more interested to follow.

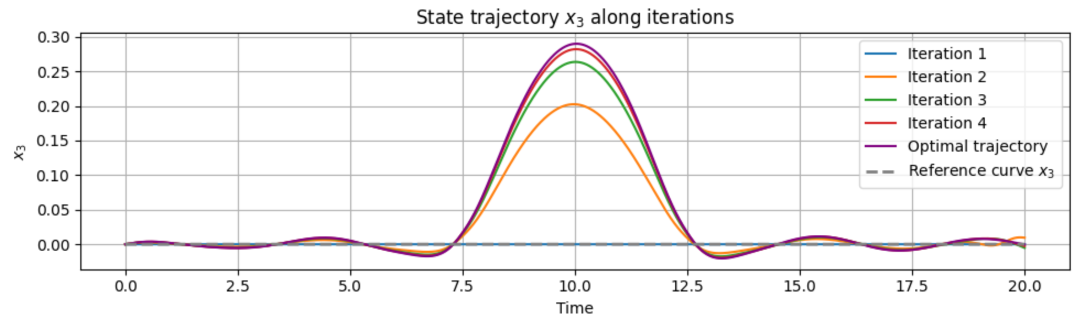

Optimal state trajectory x_1 vs reference curve
x_1^*
x_1^{ref}
x_1 [rad]
Optimal state trajectory x_2 vs reference curve
x_2^*
x_2^{ref}
x_2 [rad]
Optimal state trajectory x_3 vs reference curve
x_3^*
x_3^{ref}
x_3 [rad]
Optimal state trajectory x_4 vs reference curve
x_4^*
x_4^{ref}
x_4 [rad/s]
Optimal input trajectory vs reference curve
u^*
u^{ref}
u [Nm]
Time [s]

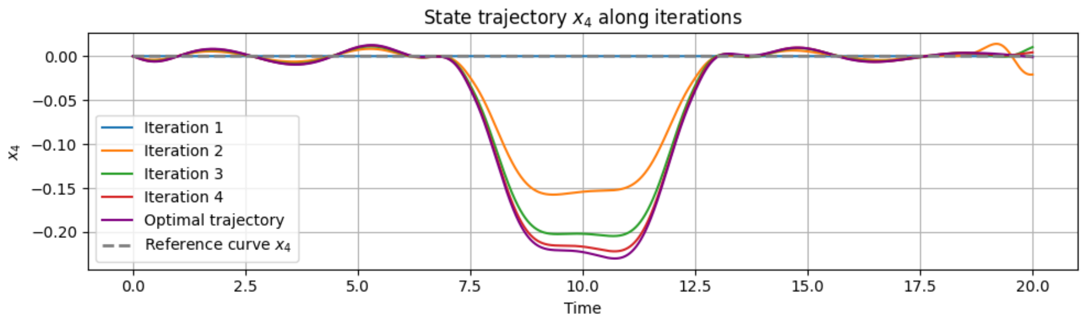
2.3.4 Trajectory plots along iterations

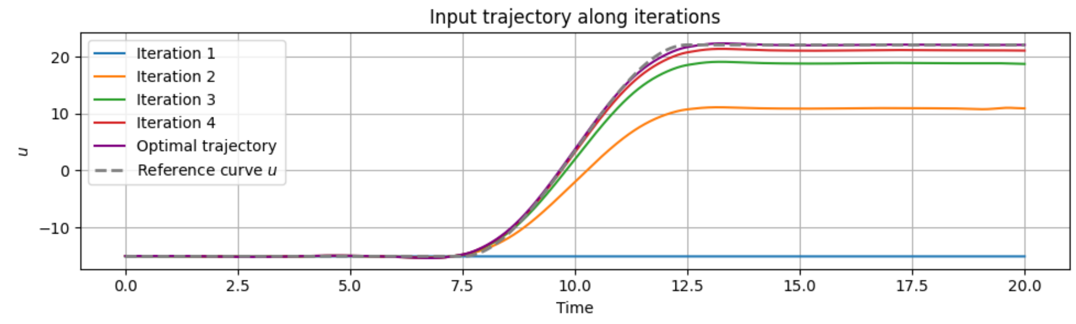
In the plots below, the the first four intermediate trajectory, the optimal trajectory and the reference curve are reported in order to show the effectiveness of Newthon's method that from a constant state-input initial guess is able to reach the optimal one very close to the reference (in particular for the angular positions and the input).





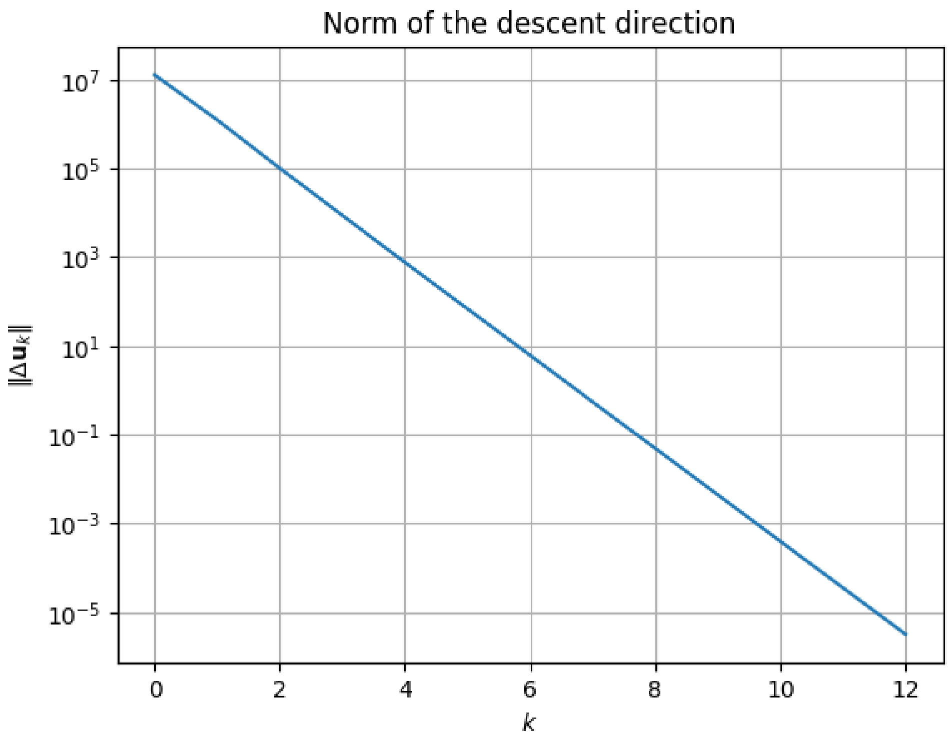


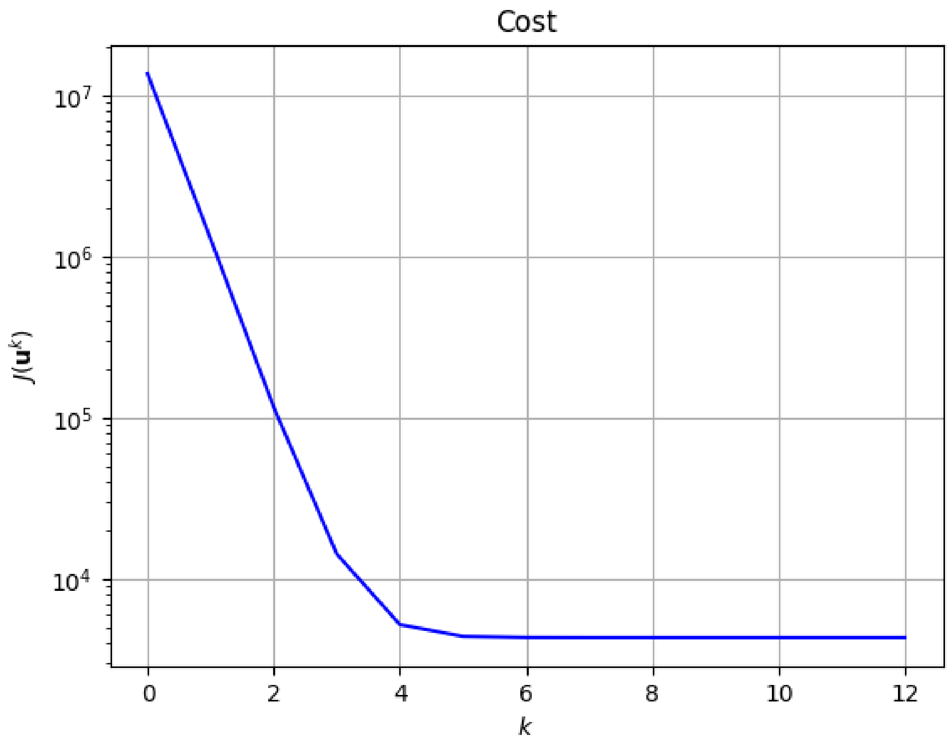




2.3.5 Descent direction norm and cost along iterations

In the plots below, it is possible to see the behaviour of the descent the direction norm and the trajectory generation cost along the iterations. The behaviour seems reasonable due to the fact that both the descent norm and the cost decrease iteration after iteration, meaning that the algorithm is moving toward a minimum of the cost. Moreover, we can notice that the norm of the descent direction at the last iteration reaches a value less than 10^{-5} as desired.





2.3.6 Armijo stepsize selection rule

Since in the Newton's method it is required to select the stepsize at each iteration according to the Armijo rule, we needed also to implement the `armijo_stepsize` function in the armijo.py file according to the procedure described below:

1. Set $\bar{\gamma}^0 > 0$, $\beta \in (0,1)$, $c \in (0,1)$.
2. While $J(\mathbf{u}^k + \bar{\gamma}^i \Delta\mathbf{u}^k) \geq J(\mathbf{u}^k) + c\bar{\gamma}^i \nabla J(\mathbf{u}^k)^T \Delta\mathbf{u}^k$:

   $\bar{\gamma}^{i+1} = \beta\bar{\gamma}^i$.
3. Set $\gamma^k = \bar{\gamma}^i$.

In particular, in the code the following values have been used:

$$\bar{\gamma}^0 = 1, \;\; \beta = 0.7, \;\; c = 0.5$$

2.3.7 Armijo descent direction plots

Below the armijo descent direction plots in the first two iterations and in the last two iterations of the Newton's method are reported.

These plots describe the procedure of Armijo stepsize selection rule: while the cost function at a certain stepsize (continuous green line) is greater or equal to the dashed green line, then the stepsize is updated through the factor beta. In fact, in all the plots below the explored stepsizes are 1 (initial value) and 0.7 (indicated with stars). Since the cost corresponding to stepsize 0.7 is below the dashed green line, then it is chosen as stepsize for the related iteration.

Furthermore, the armijo descent direction plot is also very useful to have some idea on the correctness of the implemented code. In this case, the plots seem reasonable because the red line appears indeed tangent to the cost. Morevore, another good thing that is possible to observe is that the cost decreases along the iterations.

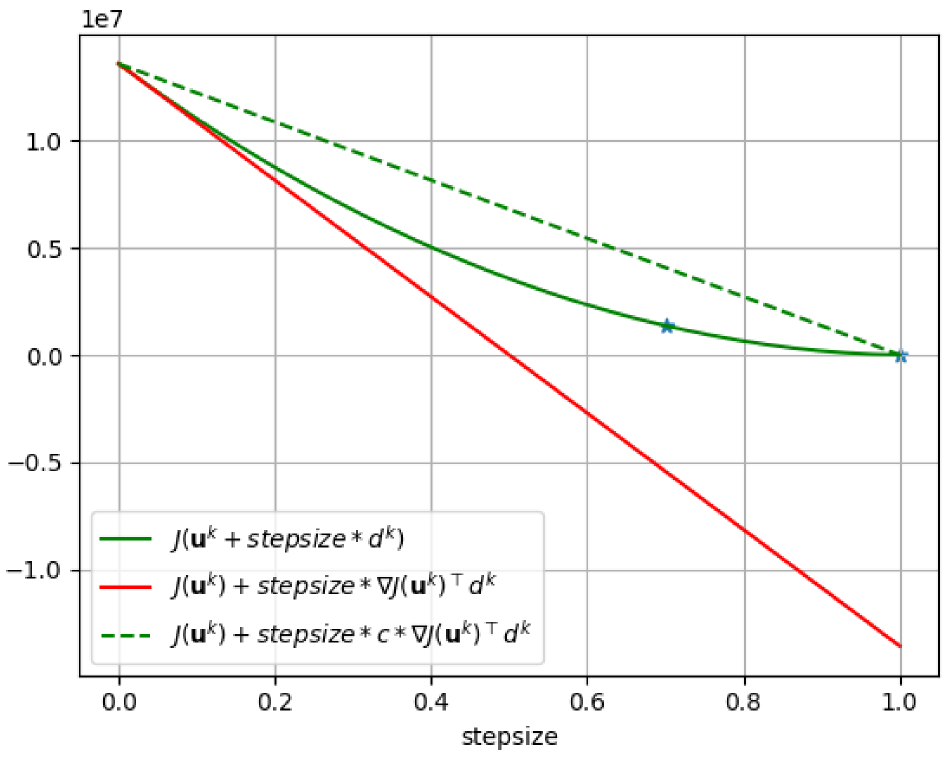


Figure: Armijo descent direction plot iteration 1

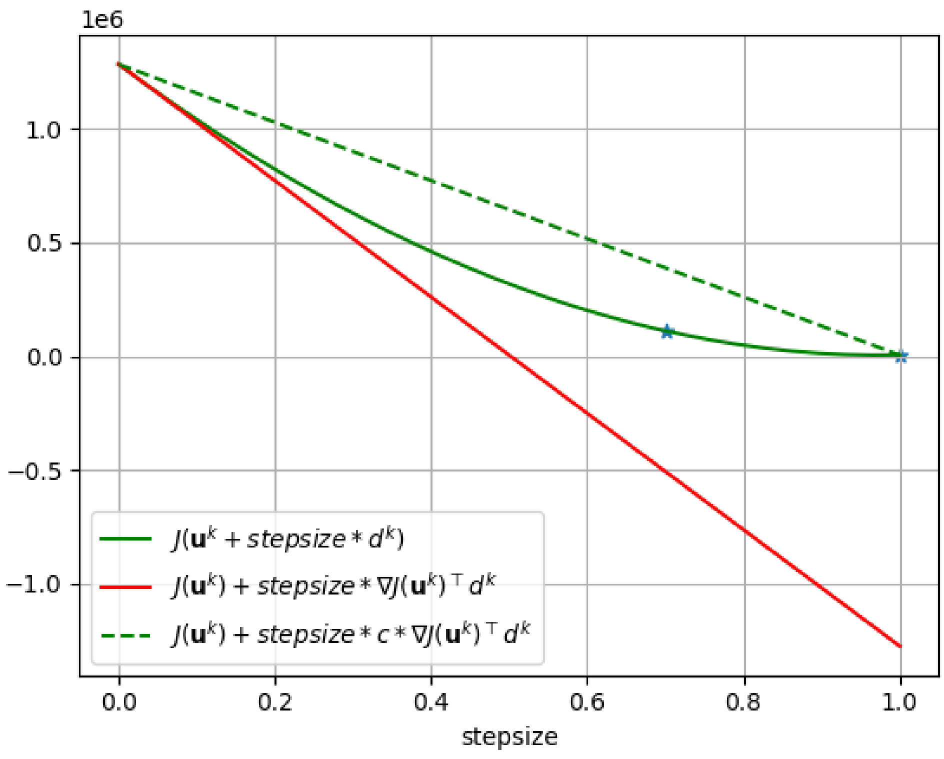


Figure: Armijo descent direction plot iteration 2

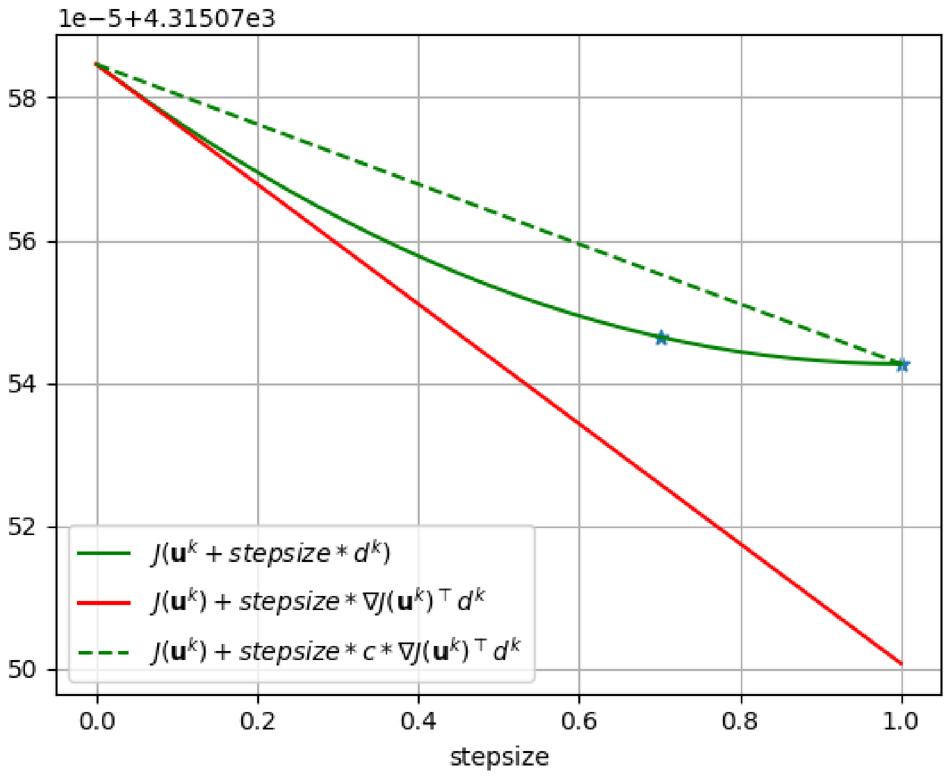


Figure: Armijo descent direction plot iteration 12

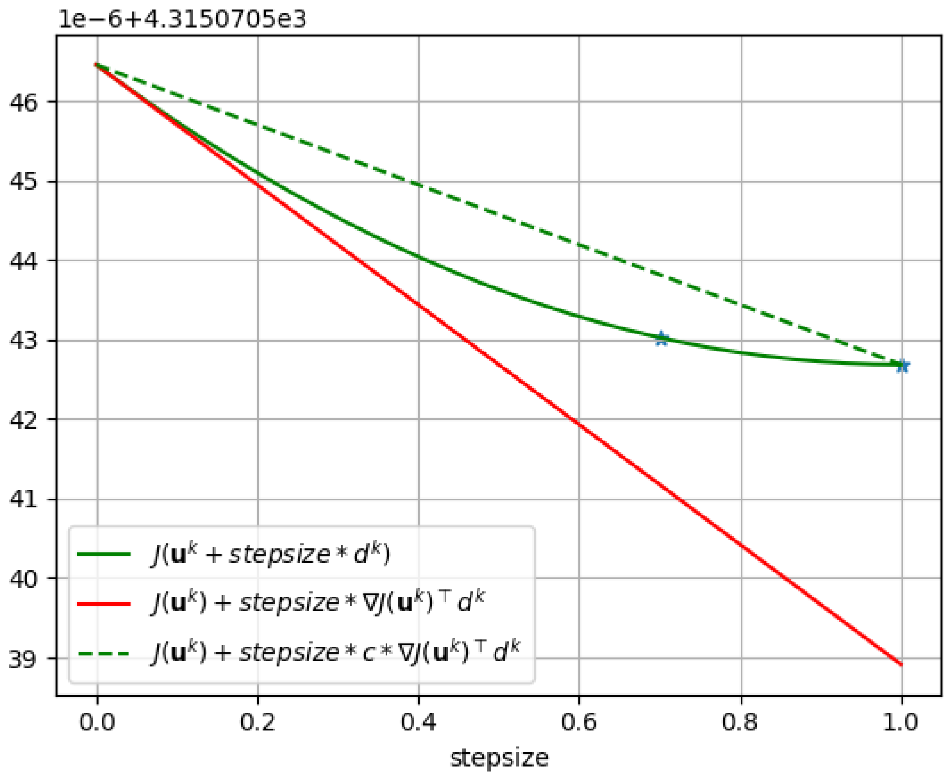


Figure: Armijo descent direction plot iteration 13

Chapter 3

Task 2 - Trajectory generation (II)

For Task 2, we encountered the challenging objective of achieving the swing-up motion for the double pendulum.

3.1 Reference curve

We have analyzed the dynamic equation of the plant, which describes the system's behavior as a function of the state variables θ_1 and θ_2.

$$M(\theta_1, \theta_2) \begin{bmatrix} \ddot{\theta}_1 \\ \ddot{\theta}_2 \end{bmatrix} + C(\theta_1, \theta_2, \dot{\theta}_1, \dot{\theta}_2) \begin{bmatrix} \dot{\theta}_1 \\ \dot{\theta}_2 \end{bmatrix} + F \begin{bmatrix} \dot{\theta}_1 \\ \dot{\theta}_2 \end{bmatrix} + G(\theta_1, \theta_2) = \begin{bmatrix} u \\ 0 \end{bmatrix}$$

The equation is a vectorial equation consisting of two scalar equations. The second equation expresses θ_1 as a function of θ_2, so, by defining $\theta_1(t)$, we can numerically compute $\theta_2(t)$. Then, with θ_1 and θ_2 in hand, we can then calculate the input u using the first equation. We defined θ_1 as a cubic polynomial transition (as the one described in the chapter related to task 1) from 0 to π over a specified time interval, and chose this time such that θ_2 also transitions from 0 to π, achieving the swing-up.

While method isn't perfectly precise, as we'd need to set the exact time for an ideal swing-up but the discrete nature of the trajectory makes this impossible. Instead, we found the closest feasible time and computed the trajectory numerically until the pendulum was near the desired final configuration. Afterward, we elongated the computed reference curve and imposed the state and input to match the exact final configuration.

At the end, we got a reference curve for θ_1, θ_2 and u, while for the two angular velocities we used a constant zero reference.

3.2 Newton's method

3.2.1 Optimal trajectory and desired curve plots

In the plots below, it is possible to observe the result of the Newton's method where there is the comparison between the optimal state-input trajectory generated by the algorithm (continuous line) and the desired state-input curve (dashed line). These results have been obtained by considering, as we did for task 1, a terminal condition based on the discent direction norm. This time the condition of stopping the Newton's algorithm when the descent direction norm is less than 10^{-5} led to 21 iterations of the algorithm. The initial state-input guess has been chosen as the first value of the reference curve and costant for all the time duration. Furthermore, the chosen weight matrices for the trajectory generation cost function are the following:

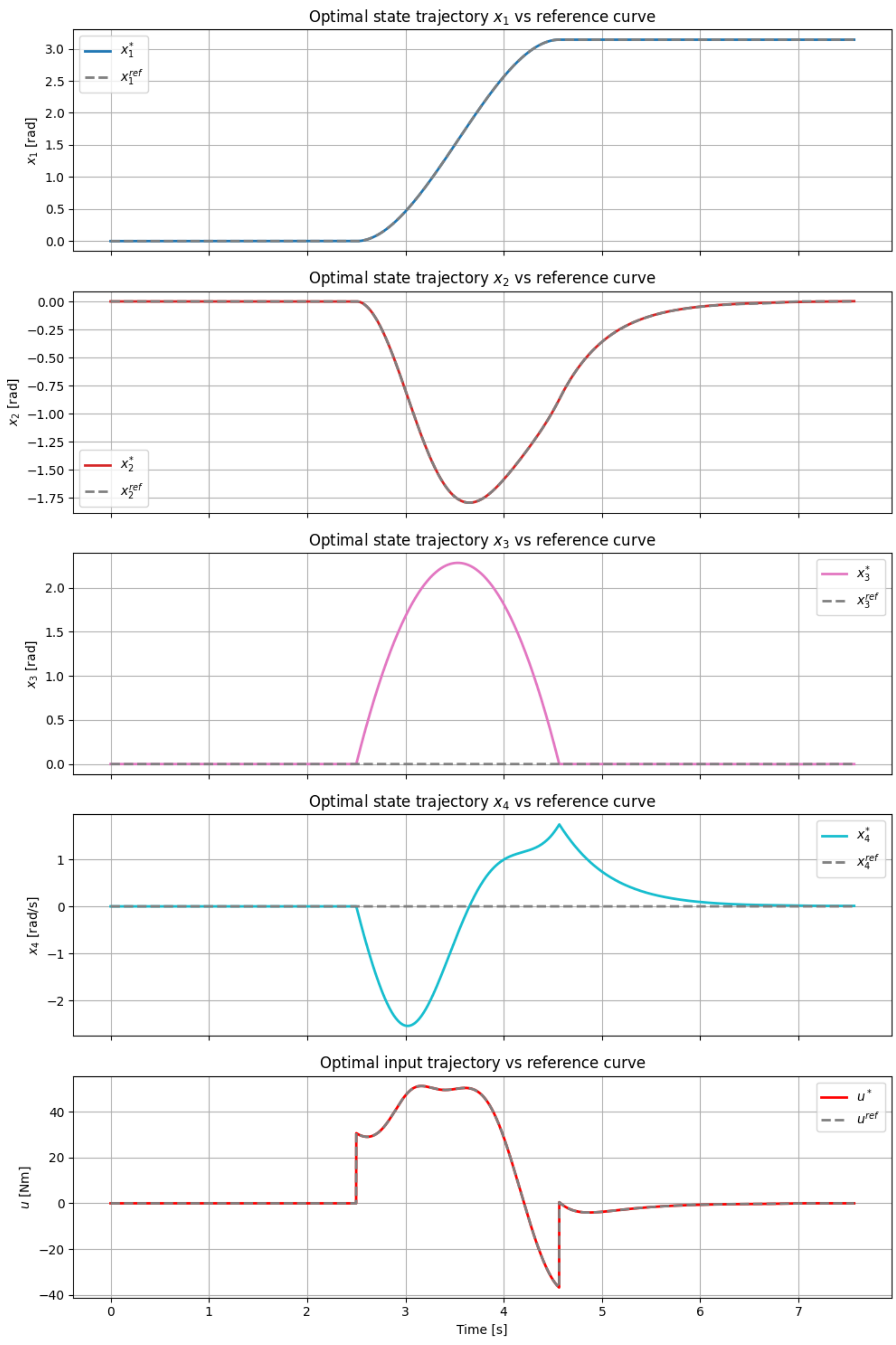
$$Q_t = \text{diag}(100000, 100000, 0.01, 0.01)$$

$$R_t = 100 \cdot I_{n_i}$$

$$Q_T = Q_t$$

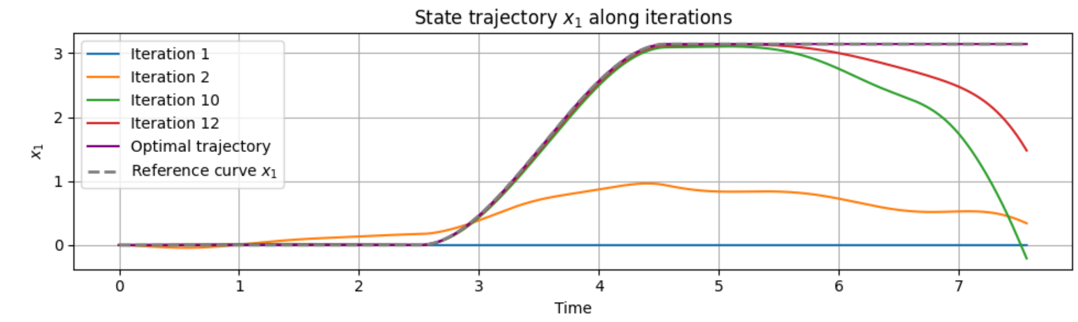
We used higher weights for the angular positions and the input since are the references that we are more interested in since we got them from the dynamics equations. While, we used lower weights for the angular velocities, since for them simply we chose a constant zero reference.

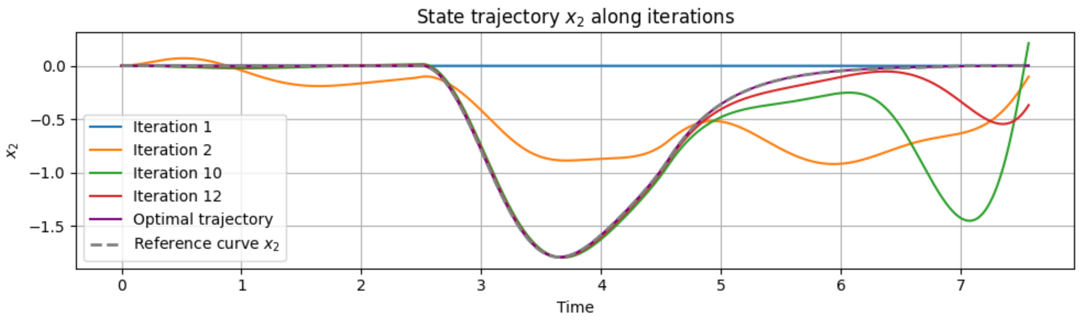
It is important to observe that in this case the state trajectory related to the positions and the input trajectory seem almost coincide with the reference curve, since these have been obtained from the dynamics equations (except the last elongated part). While this is not the case for the reference angular velocities.

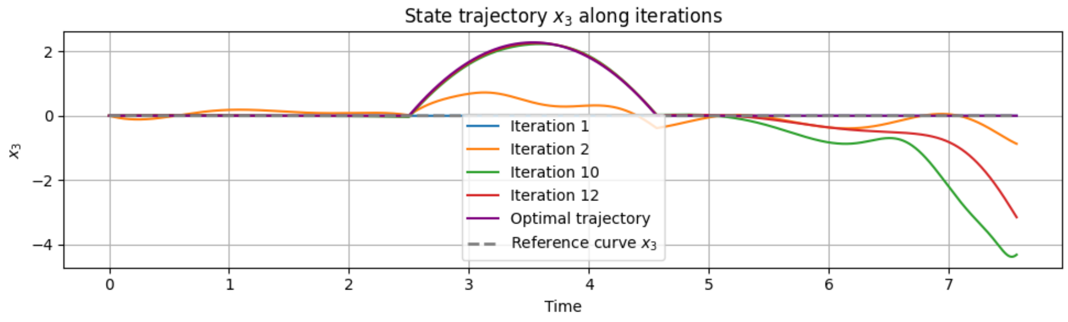

Optimal state trajectory x_1 vs reference curve
x_1 [rad]
x_1^*
x_1^{ref}
Optimal state trajectory x_2 vs reference curve
x_2 [rad]
x_2^*
x_2^{ref}
Optimal state trajectory x_3 vs reference curve
x_3 [rad]
x_3^*
x_3^{ref}
Optimal state trajectory x_4 vs reference curve
x_4 [rad/s]
x_4^*
x_4^{ref}
Optimal input trajectory vs reference curve
u [Nm]
u^*
u^{ref}
Time [s]

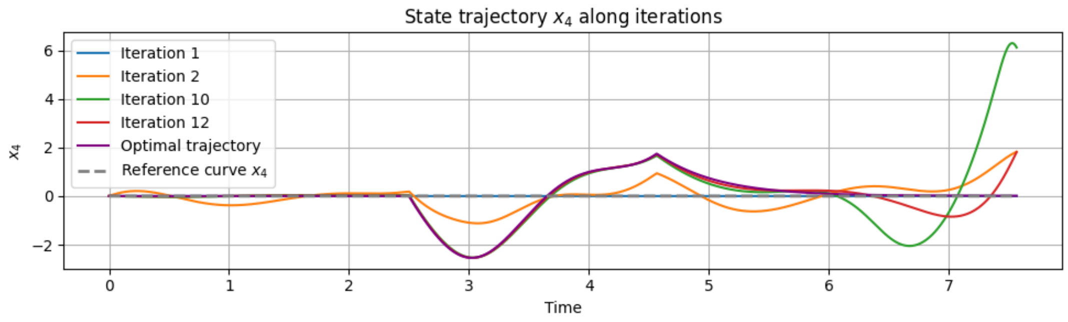
3.2.2 Trajectory plots along iterations

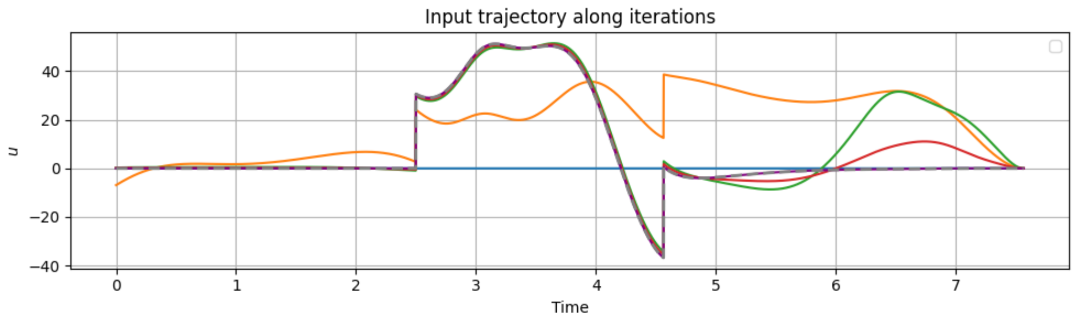
In the plots below, some intermediate trajectories, the optimal trajectory and the reference curve are reported in order to show the effectiveness of Newthon's method that from a constant state-input initial guess is able to reach the optimal one very close to the reference.





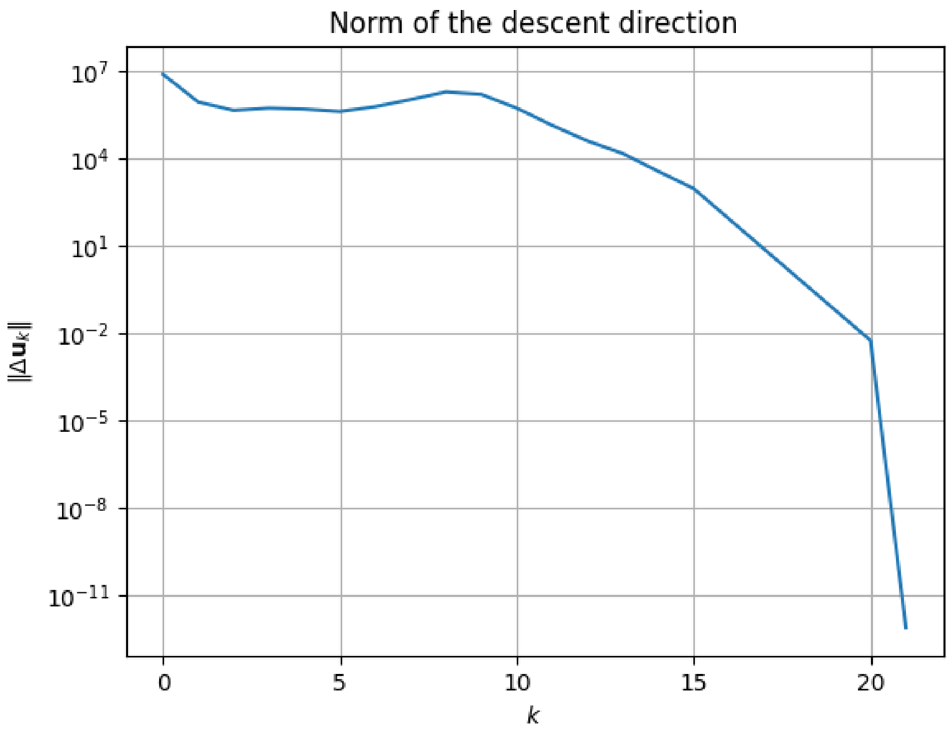


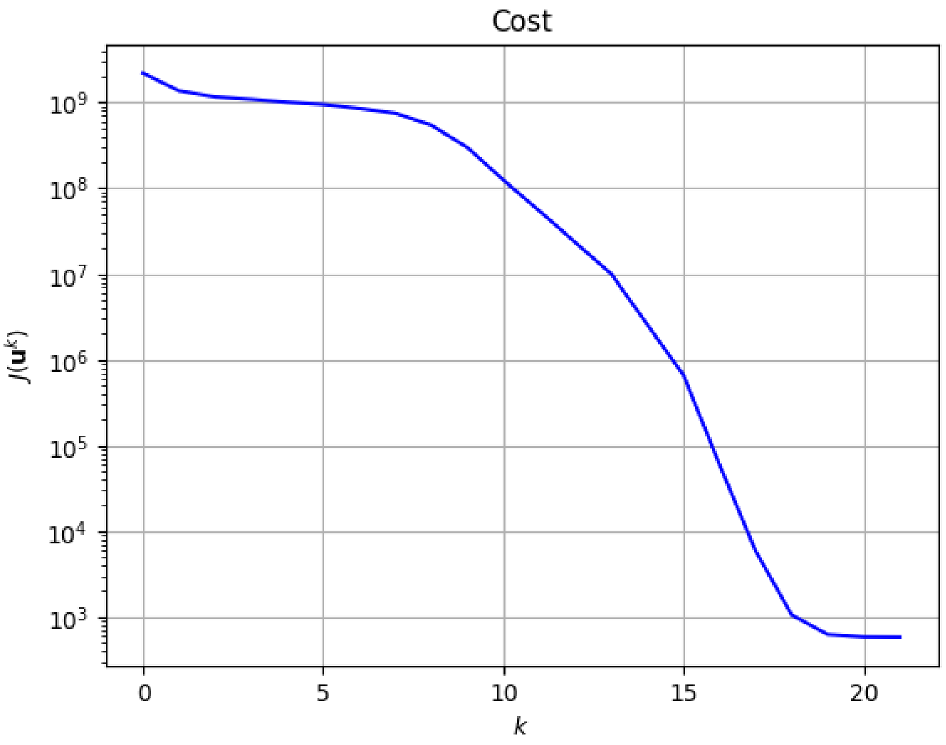




3.2.3 Descent direction norm and cost along iterations

In the plots below, it is possible to see the behaviour of the descent the direction norm and the trajectory generation cost along the iterations. The behaviour seems reasonable due to the fact that both the descent norm and the cost have a decrease from the first to the last iteratiom, meaning that the algorithm is moving toward a minimum of the cost. Moreover, we can notice that the norm of the descent direction at the last iteration reaches a value less than 10^{-5} as desired.





3.2.4 Armijo descent direction plots

The Armijo descent direction plot is also very useful to have some idea on the correctness of the implemented code as we have already discussed for task 1. Also in this case, the plots seem reasonable because the red line appears indeed tangent to the cost. Moreover, another good thing that is possible to observe is that the cost decreases along the iterations.

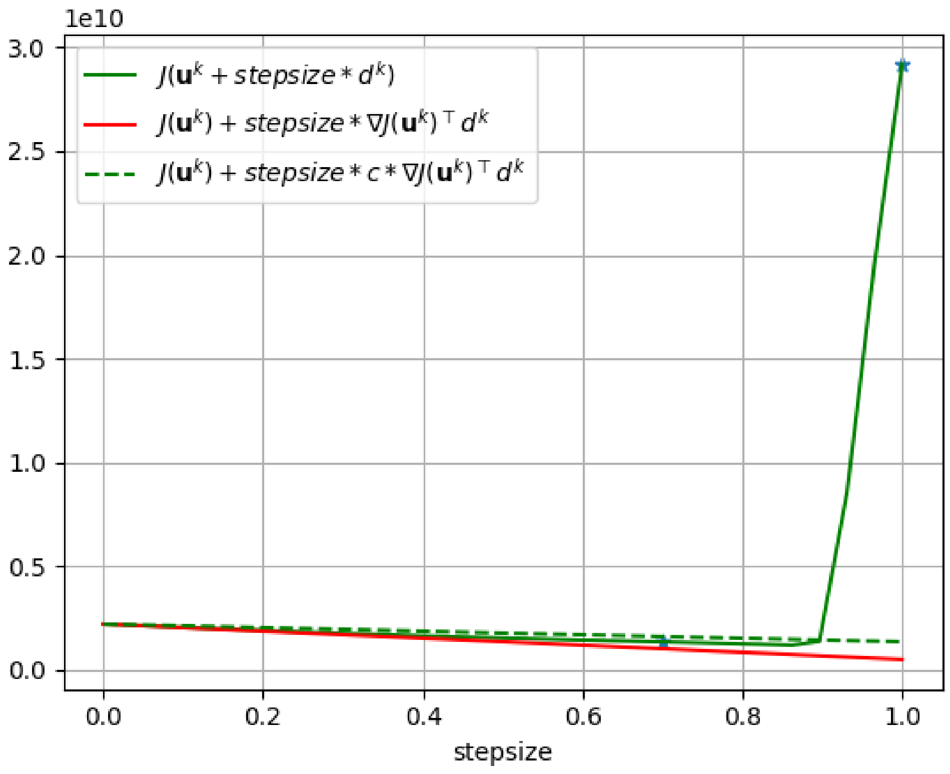


Figure: Armijo descent direction plot iteration 1

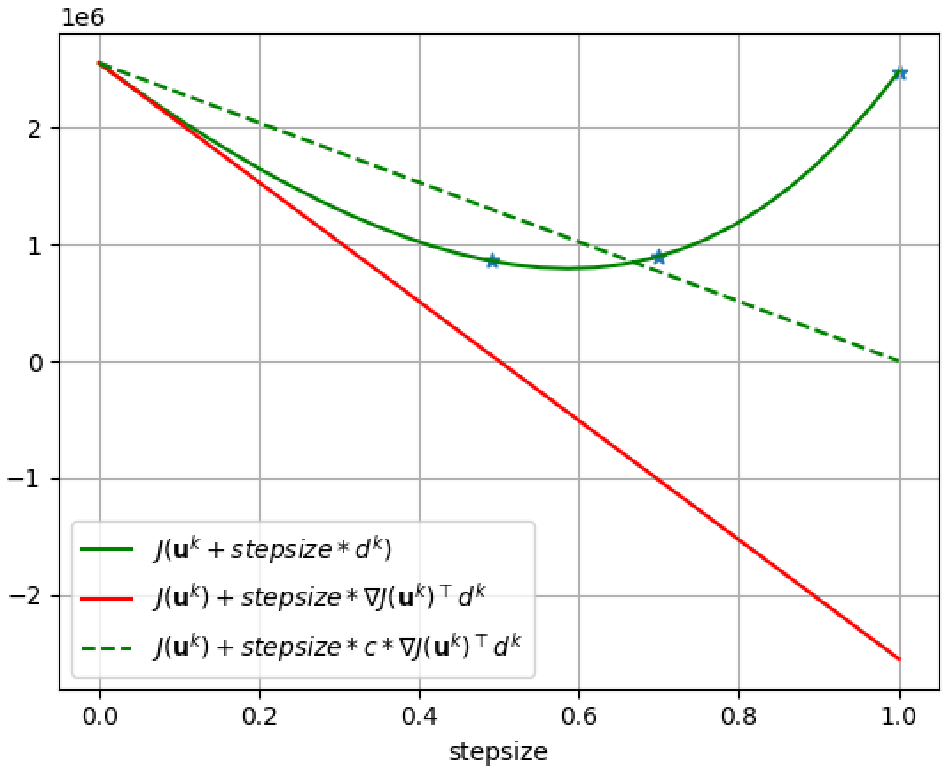


Figure: Armijo descent direction plot iteration 15

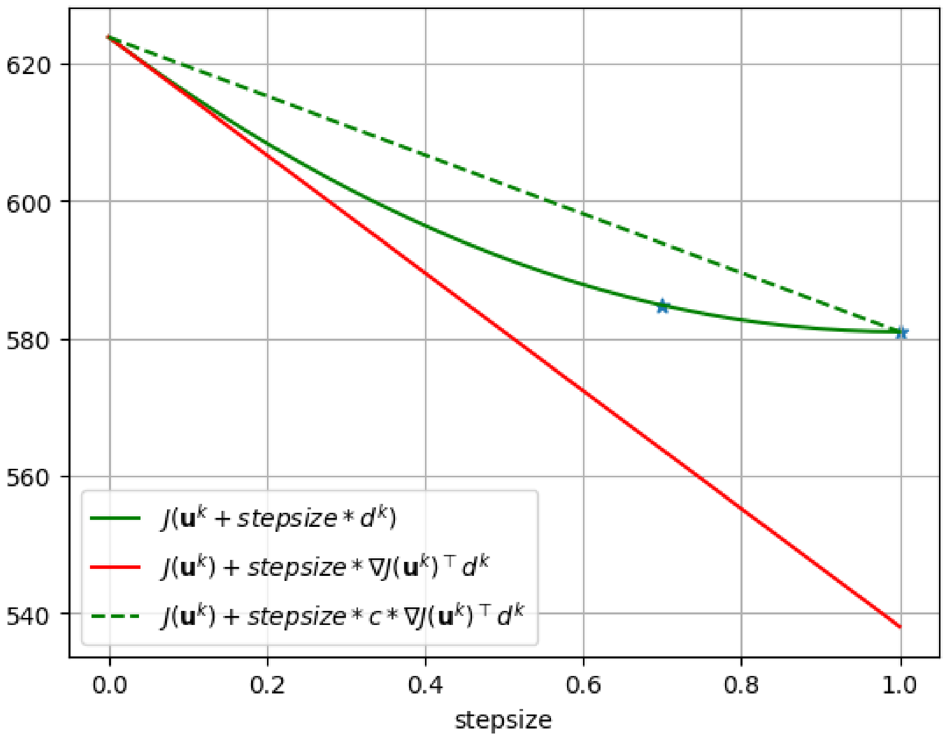


Figure: Armijo descent direction plot iteration 19

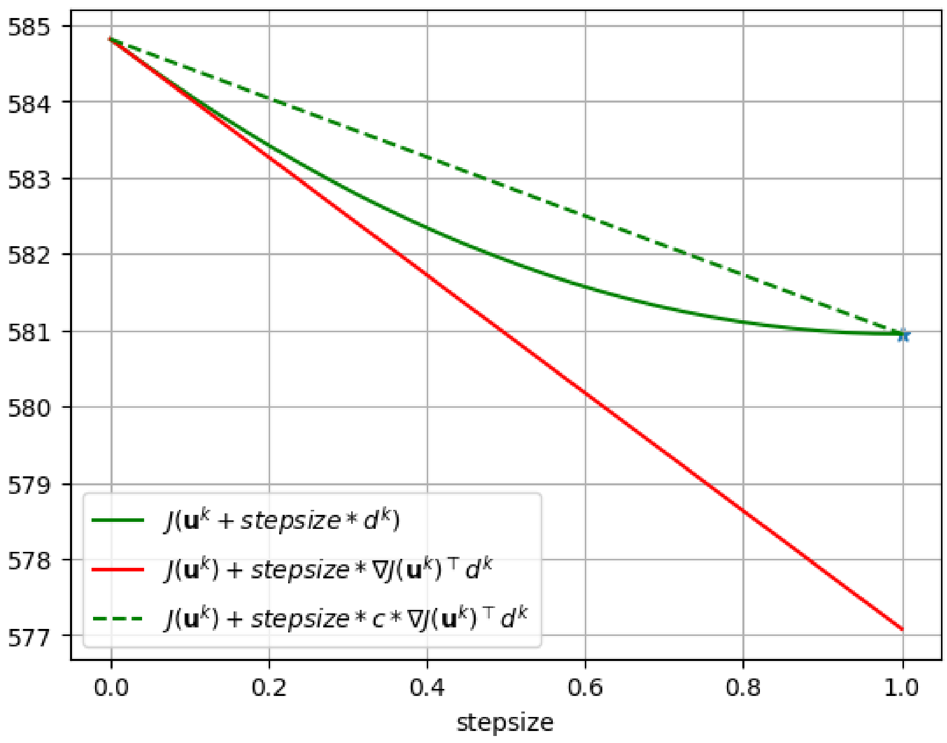


Figure: Armijo descent direction plot iteration 20

Chapter 4

Task 3 - Trajectory tracking via LQR

For task 3, the objective is to exploit trajectory tracking via LQR in order to track the optimal desired trajectory generated in task 2 showing tracking performance considering a perturbed initial condition with respect to the one of the optimal trajectory.

4.1 LQR-based trajectory tracking algorithm

The idea of trajectory tracking via LQR is to exploit the LQR algorithm to define the optimal feedback controller, based on the linearization of the system around the optimal reference trajectory, to track this reference trajectory.

The main steps of the algorithm, implemented in the `lqr_track` function in the LQR_tracking.py file, are the following:

Step 1: System linearization

Linearization of the dynamics about the generated optimal trajectory $(\mathbf{x}^{\text{traj}}, \mathbf{u}^{\text{traj}})$ by computing the time-varying state and input linearization matrices A and B:

$$A_t^{traj} = A(x_t^{traj}, u_t^{traj}), \quad B_t^{traj} = B(x_t^{traj}, u_t^{traj}) \qquad t = 0, 1, ..., T-1$$

Where A and B are the linearization matrices computed in the dynamics.py file in task 0 and they must be computed at the state-input optimal trajectory.

Step 2: Computation of the optimal feedback gain

Solve the optimal control problem:

$$\min_{\Delta x_1,\ldots,\Delta x_T,\Delta u_0,\ldots,\Delta u_{T-1}} \sum_{t=0}^{T-1} \Delta x_t^\top Q_t^{\text{reg}} \Delta x_t + \Delta u_t^\top R_t^{\text{reg}} \Delta u_t + \Delta x_T^\top Q_T^{\text{reg}} \Delta x_T$$

subject to the linearized system dynamics:

$$\Delta x_{t+1} = A_t^{\text{traj}} \Delta x_t + B_t^{\text{traj}} \Delta u_t, \quad t = 0, \ldots, T-1,$$

$$\Delta x_0 = 0$$

Set $P_T = Q_T^{\text{reg}}$ and solve backward in time for $t = T-1, \ldots, 0$ the difference Riccati equation to get the values of the matrix P_t at each time:

$$P_t = Q_t^{\text{reg}} + (A_t^{\text{traj}})^\top P_{t+1} A_t^{\text{traj}} - \big((A_t^{\text{traj}})^\top P_{t+1} B_t^{\text{traj}}\big)\big(R_t^{\text{reg}} + (B_t^{\text{traj}})^\top P_{t+1} B_t^{\text{traj}}\big)^{-1}\big((B_t^{\text{traj}})^\top P_{t+1} A_t^{\text{traj}}\big),$$

and use the matrix P_t to compute for all $t = 0, \dots, T-1$, the optimal feedback gain K_t^{reg}:

$$K_t^{\text{reg}} := -\big(R_t^{\text{reg}} + (B_t^{\text{traj}})^\top P_{t+1} B_t^{\text{traj}}\big)^{-1} \big((B_t^{\text{traj}})^\top P_{t+1} A_t^{\text{traj}}\big).$$

The difference Riccati equation and the optimal feedback gain is what is implemented in the function lqr in the LQR.py file.

Step 3: Application of the optimal LQR controller to the non linear system

At the end the control law based on the optimal feedback gain is applied to the original non linear dynamics in order to track the reference trajectory and so also the evolution of the state is simulated according to the feedback control law:

$$u_t = u_t^{\text{traj}} + K_t^{\text{reg}}(x_t - x_t^{\text{traj}})$$

$$x_{t+1} = f_t(x_t, u_t), \quad t = 0, 1, \dots$$

with x_0 given.

4.2 LQR tracking plots

In order to show the tracking performance of the LQR controller, we considered a perturbed initial condition $x_0^{perturbed}$ with respect to the initial condition x_0^{traj} of the reference optimal trajectory:

$$x_0^{perturbed} = x_0^{traj} + noise$$

Furthermore, the chosen weight matrices for the LQR are the following:

$$Q_t^{reg} = \text{diag}(10000, 10000, 100, 100)$$

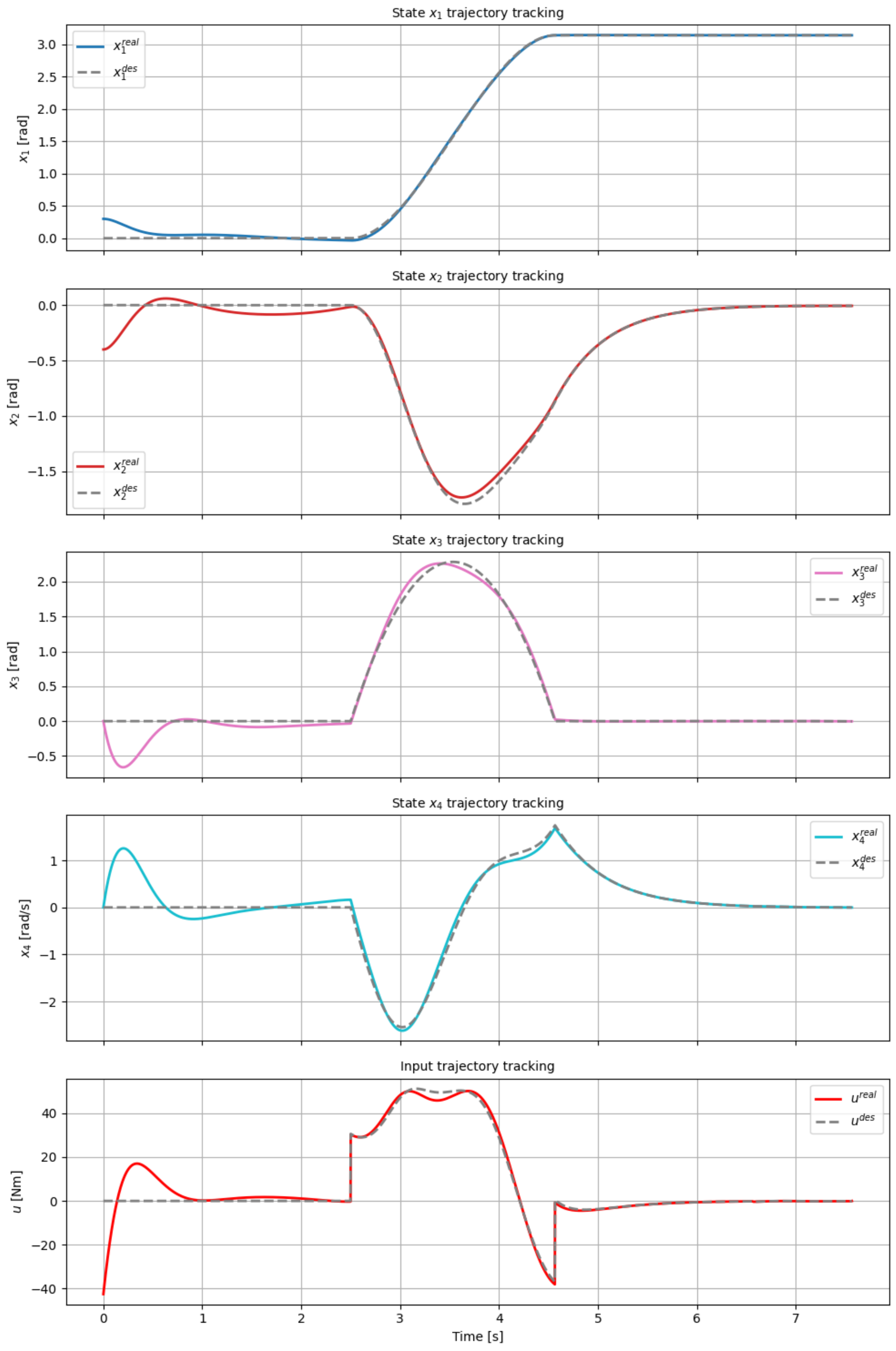
$$R_t^{reg} = 1 \cdot I_{n_i}$$

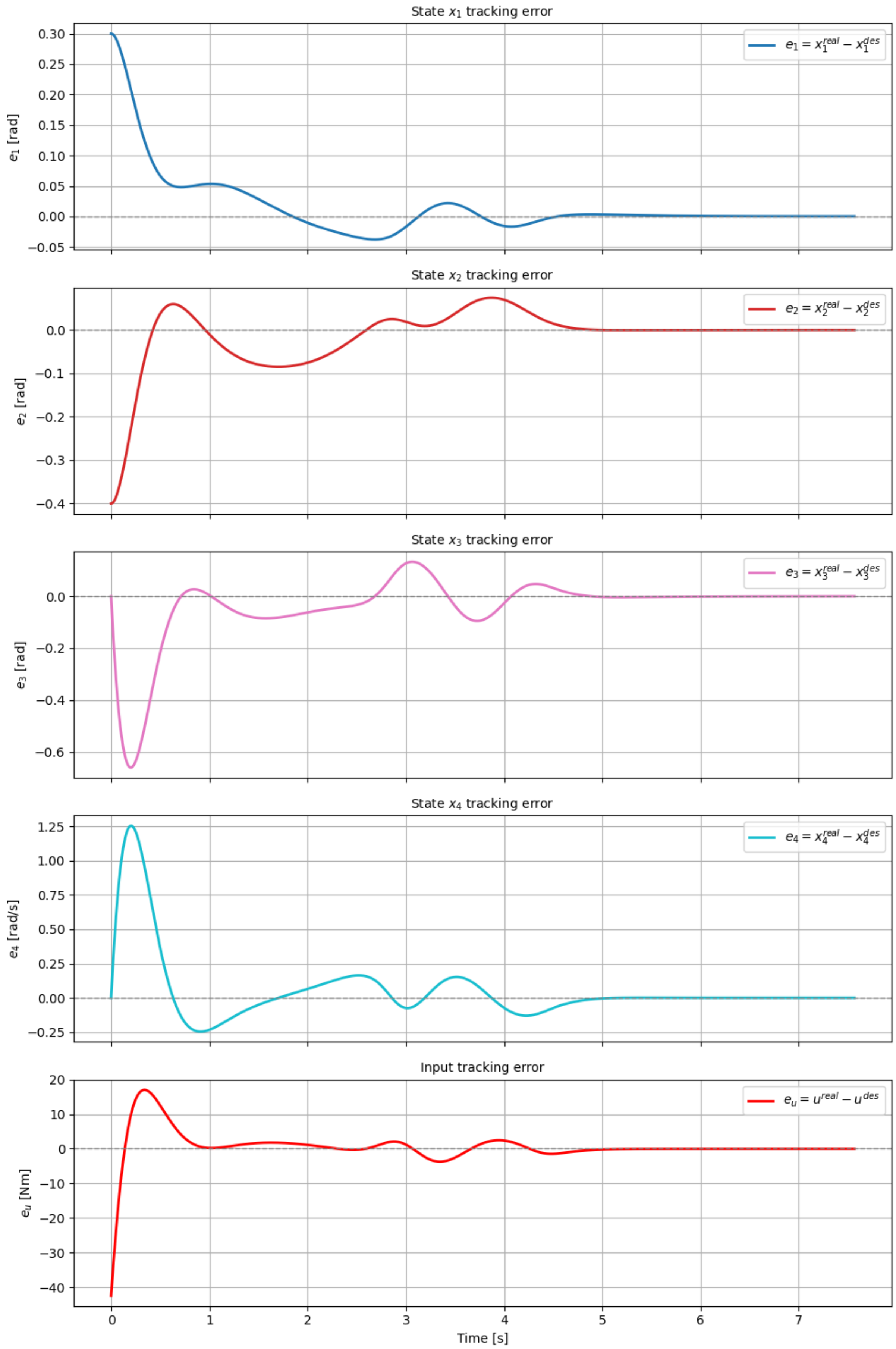
$$Q_T^{reg} = Q_t^{reg}$$

The plots regarding the real state-input trajectory versus the desired optimal one are reported below together with the related tracking error plots in the case in which the noise is:

$$noise = \begin{bmatrix} 0.3 \\ -0.4 \\ 0 \\ 0 \end{bmatrix}$$

In this experiment, the noise doesn't affect the angular velocity and the tracking performances look very good.

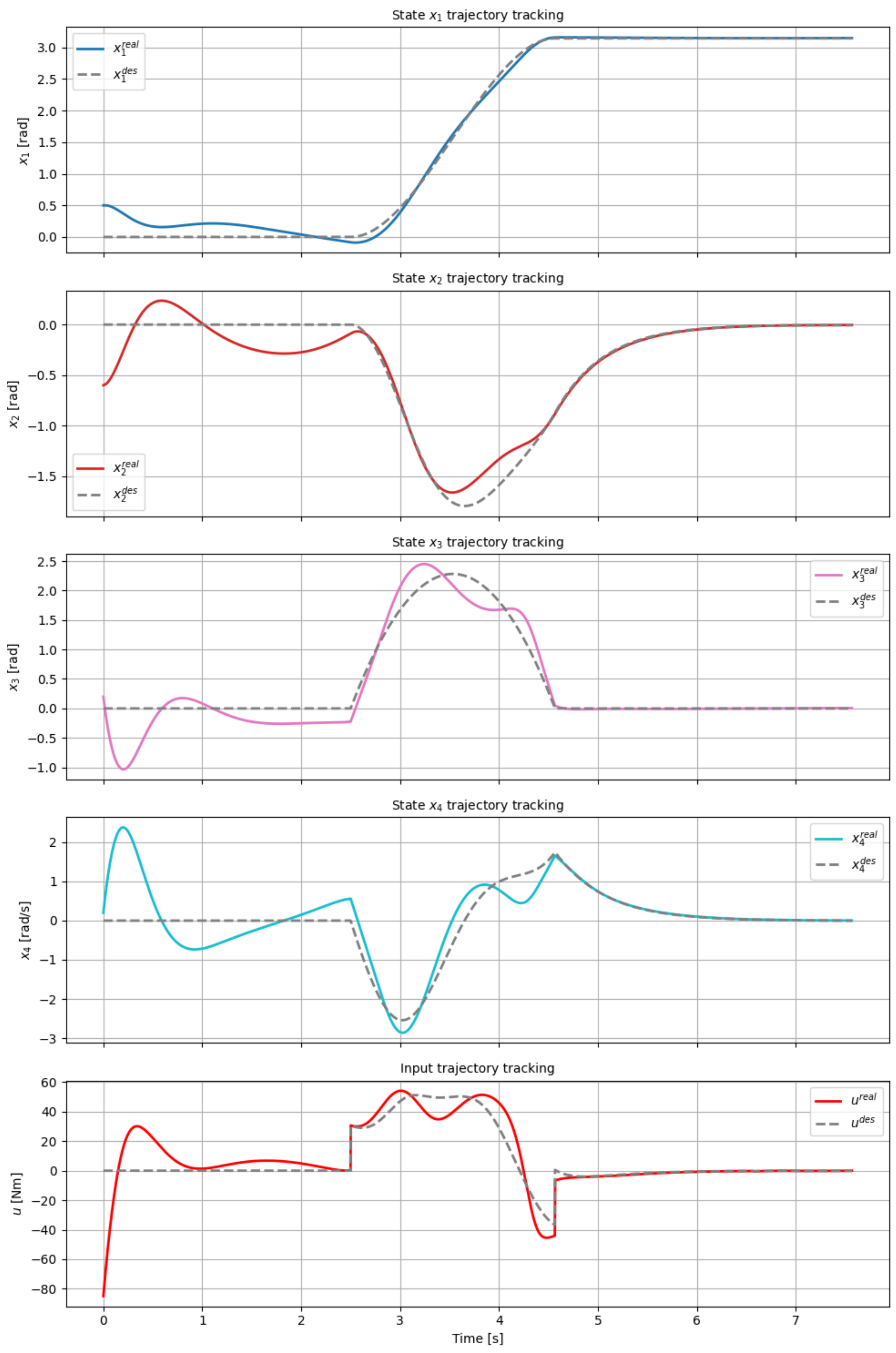

State x_1 trajectory tracking
x_1 [rad]
x_1^{real}
x_1^{des}
State x_2 trajectory tracking
x_2 [rad]
x_2^{real}
x_2^{des}
State x_3 trajectory tracking
x_3 [rad]
x_3^{real}
x_3^{des}
State x_4 trajectory tracking
x_4 [rad/s]
x_4^{real}
x_4^{des}
Input trajectory tracking
u [Nm]
u^{real}
u^{des}
Time [s]


State x_1 tracking error
$e_1 = x_1^{real} - x_1^{des}$
e_1 [rad]
State x_2 tracking error
$e_2 = x_2^{real} - x_2^{des}$
e_2 [rad]
State x_3 tracking error
$e_3 = x_3^{real} - x_3^{des}$
e_3 [rad]
State x_4 tracking error
$e_4 = x_4^{real} - x_4^{des}$
e_4 [rad/s]
Input tracking error
$e_u = u^{real} - u^{des}$
e_u [Nm]
Time [s]

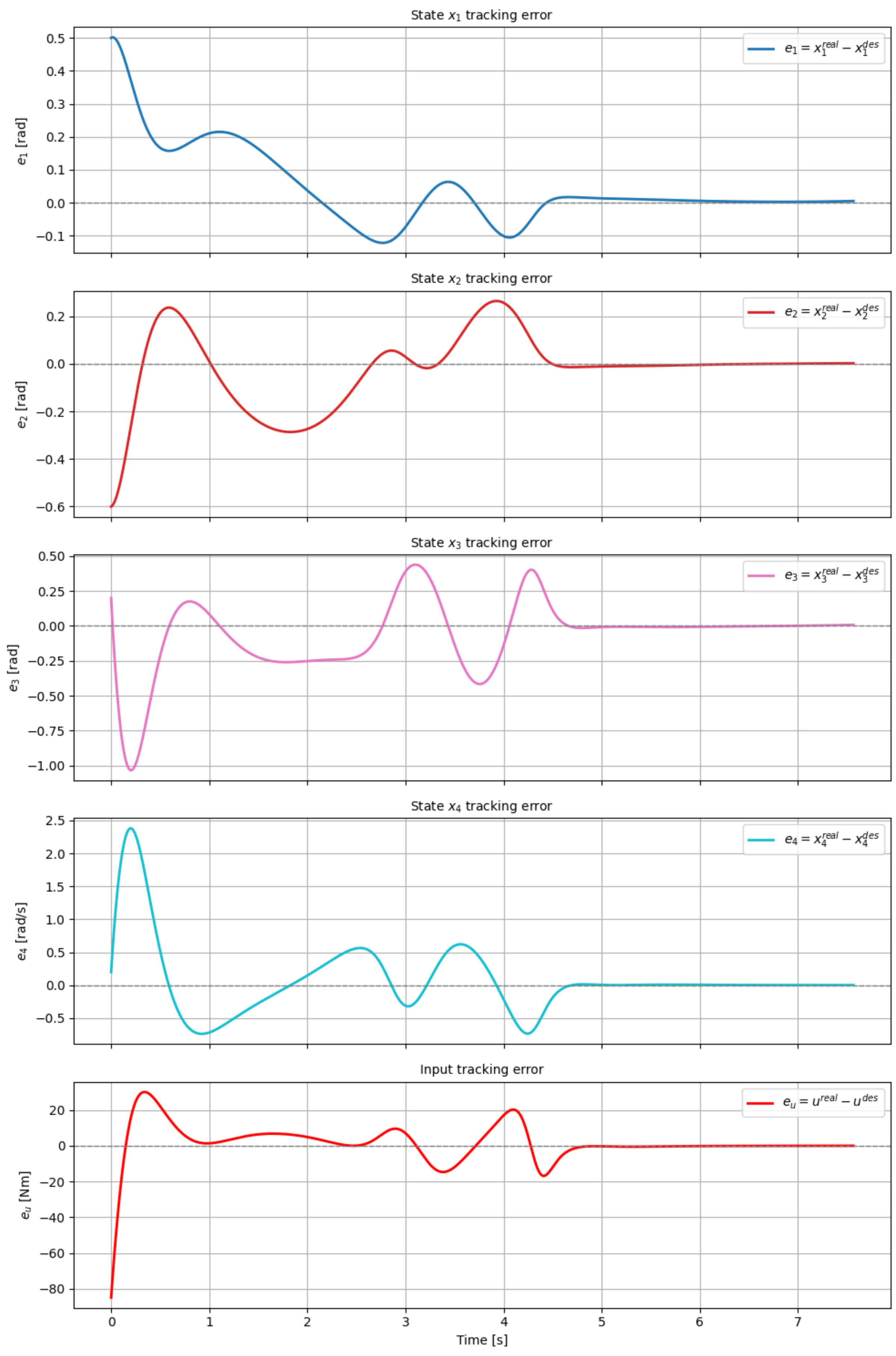
We report below also the same type of plots in case the noise that affects the ideal initial condition is:

$$noise = \begin{bmatrix} 0.5 \\ -0.6 \\ 0.2 \\ 0.2 \end{bmatrix}$$

In this case, we have considered a slight higher noise for the angular positions and also a noise in the angular velocities not present in the previous case. This time the LQR has more difficulty to track the reference due to the worse noise condition, but still the tracking performances are good.



State x_1 trajectory tracking
x_1 [rad]
x_1^{real}
x_1^{des}
State x_2 trajectory tracking
x_2 [rad]
x_2^{real}
x_2^{des}
State x_3 trajectory tracking
x_3 [rad]
x_3^{real}
x_3^{des}
State x_4 trajectory tracking
x_4 [rad/s]
x_4^{real}
x_4^{des}
Input trajectory tracking
u [Nm]
u^{real}
u^{des}
Time [s]


State x_1 tracking error
$e_1 = x_1^{real} - x_1^{des}$
e_1 [rad]
State x_2 tracking error
$e_2 = x_2^{real} - x_2^{des}$
e_2 [rad]
State x_3 tracking error
$e_3 = x_3^{real} - x_3^{des}$
e_3 [rad]
State x_4 tracking error
$e_4 = x_4^{real} - x_4^{des}$
e_4 [rad/s]
Input tracking error
$e_u = u^{real} - u^{des}$
e_u [Nm]
Time [s]

Chapter 5

Task 4 - Trajectory tracking via MPC

For task 4, the objective is to exploit trajectory tracking via MPC in order to track the optimal desired trajectory generated in task 2 showing tracking performance considering a perturbed initial condition with respect to the one of the optimal trajectory.

5.1 MPC-based trajectory tracking controller

MPC (Model Predictive Control) is a state feedback controller based on the following steps for each t:

1. Measure the current state x_t.
2. Compute the optimal trajectory solving an optimal control problem in the prediction horizon T:
$$x^*_{t|t}, \dots, x^*_{t+T|t},\ u^*_{t|t}, \dots, u^*_{t+T-1|t} \quad \text{with initial condition } x_t.$$
3. Apply the first control input $u^*_{t|t}$.
4. Measure x_{t+1} and repeat.

The optimal control problem to be solved t each time instant t is the following:

$$\begin{aligned}
\min_{x_t,\dots,x_{t+T},u_t,\dots,u_{t+T-1}} \quad & \sum_{\tau=t}^{t+T-1} \ell_\tau(x_\tau, u_\tau) + \ell_{t+T}(x_{t+T}) \\
\text{subj. to} \quad & x_{\tau+1} = f(x_\tau, u_\tau) \quad \forall \tau = t, \dots, t+T-1, \\
& x_\tau \in \mathcal{X},\ u_\tau \in \mathcal{U} \quad \forall \tau = t, \dots, t+T, \\
& x_t = x_t^{\text{meas}}.
\end{aligned}$$

In the case where the dynamics is linear and the cost quadratic, at each time instant t the following LQ problem has to be solved:

$$\begin{aligned}
\min_{x_t,\dots,x_{t+T},u_t,\dots,u_{t+T-1}} \quad & \sum_{\tau=t}^{t+T-1} \Big(x_\tau^\top Q_\tau x_\tau + u_\tau^\top R_\tau u_\tau \Big) + x_{t+T}^\top Q_{t+T} x_{t+T} \\
\text{subj. to} \quad & x_{\tau+1} = A_\tau x_\tau + B_\tau u_\tau \quad \forall \tau = t, \dots, t+T-1, \\
& x_\tau \in \mathcal{X},\ u_\tau \in \mathcal{U} \quad \forall \tau = t, \dots, t+T, \\
& x_t = x_t^{\text{meas}}.
\end{aligned}$$

In our case, we implemented the MPC based on solving an LQ problem considering the linearization of the system around a reference generated trajectory and without taking into account constraints on the state and input.

5.2 MPC tracking plots

We developed the basic mpc control scheme with two different implementations: the first based on the LQR function implemented by us and the other exploiting the library cvxpy. In particular, the first implementation is based on solving at each time t an LQ problem on the appropriate prediction horizon, getting the optimal gain feedback from our implemented LQR function and applying only the first control input by selecting only the first optimal gain matrix. Both the two implementations lead to the same results/plots even if the one based on cvxpy is computationally very slow. We found out that these implementations are not able to track the swing up trajectory of task 2 if there is a perturbed initial condition, as it's possible to see in the plots below obtained by considering a prediction horizon of 5, the following noise added to the ideal initial condition:

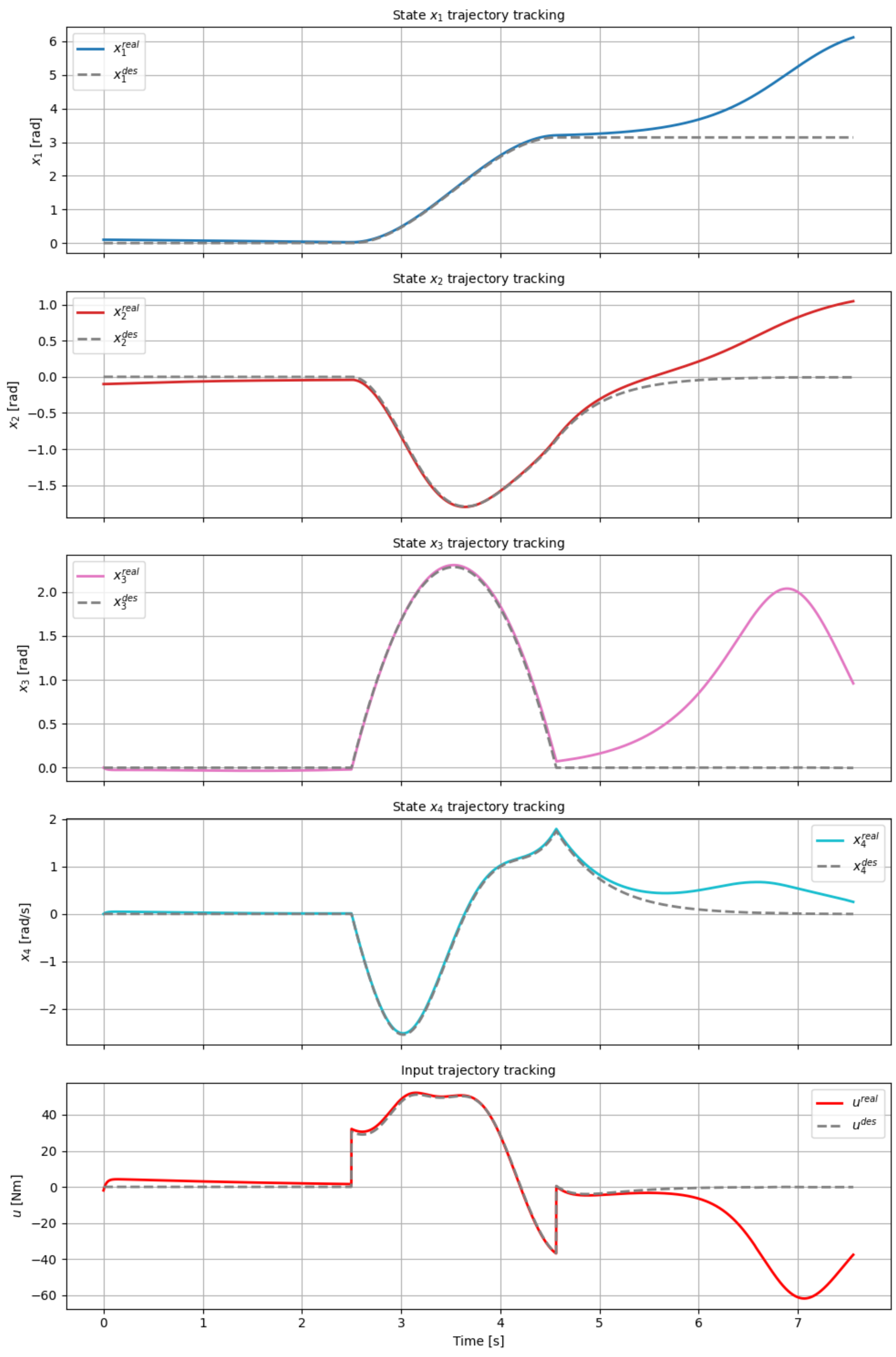
$$noise = \begin{bmatrix} 0.1 \\ -0.1 \\ 0 \\ 0 \end{bmatrix}$$

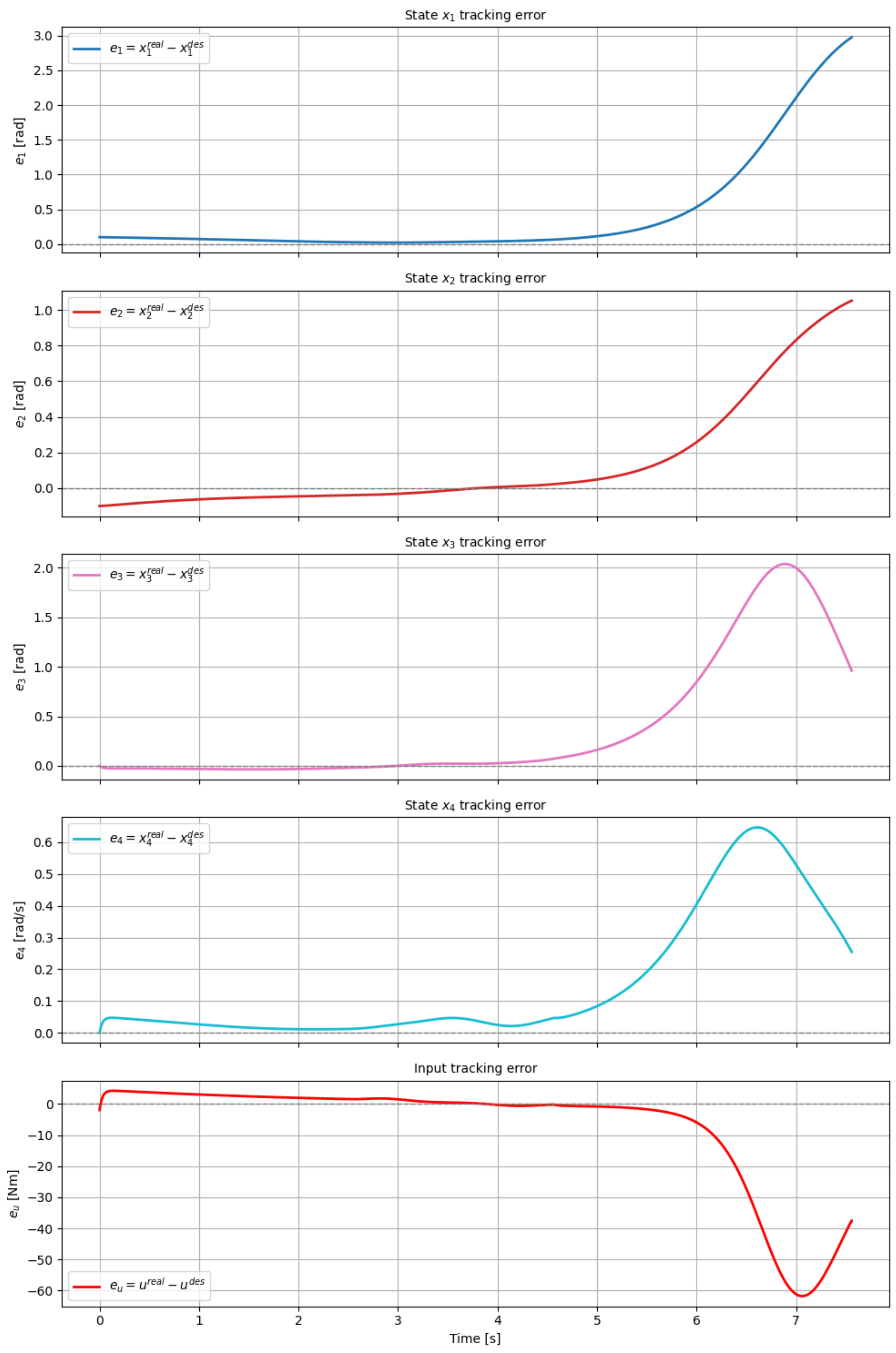
and the following regulator weight matrices:

$$Q_t^{reg} = \text{diag}(10000, 10000, 100, 100)$$

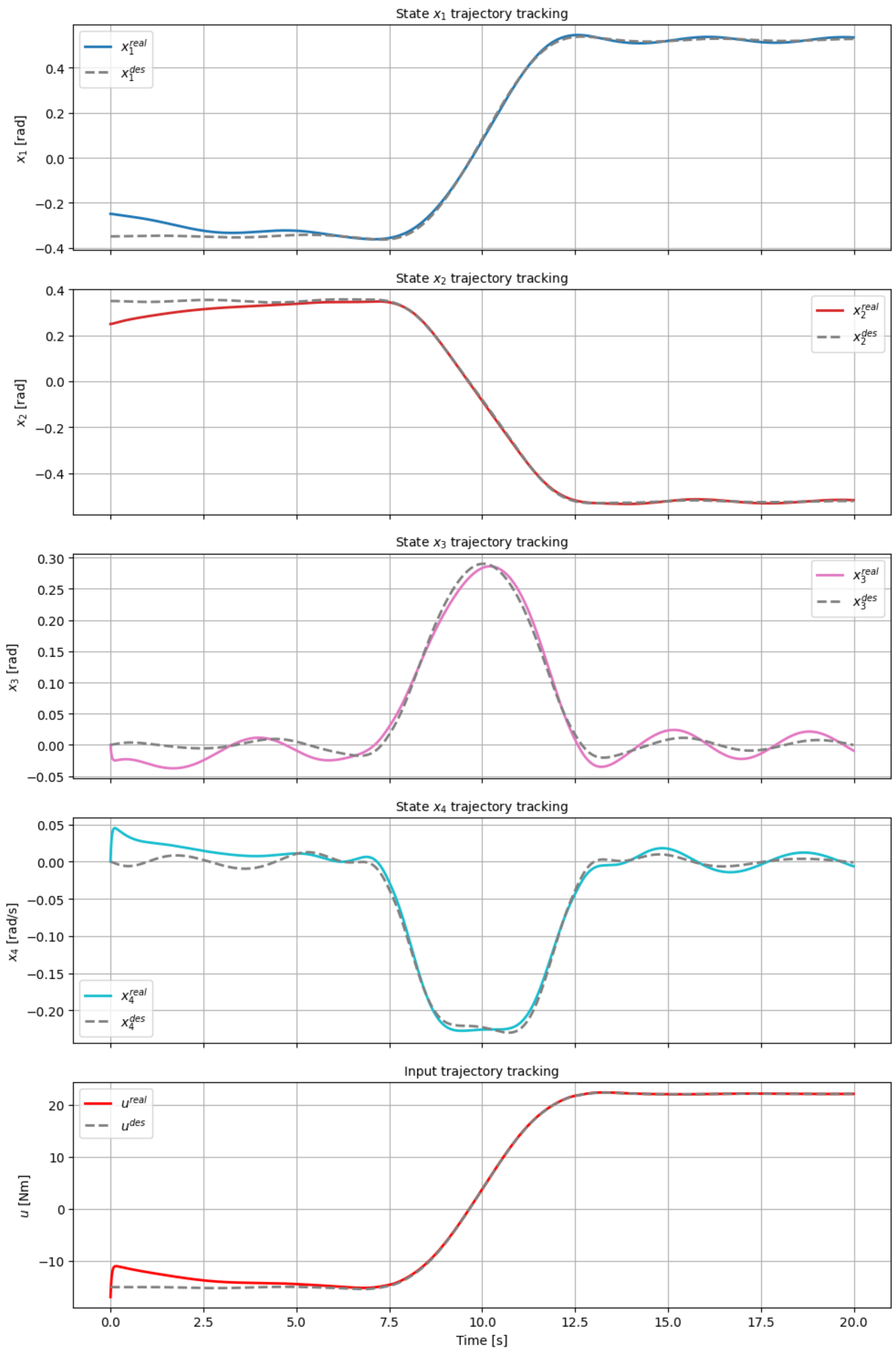
$$R_t^{reg} = 0.001 \cdot I_{n_i}$$

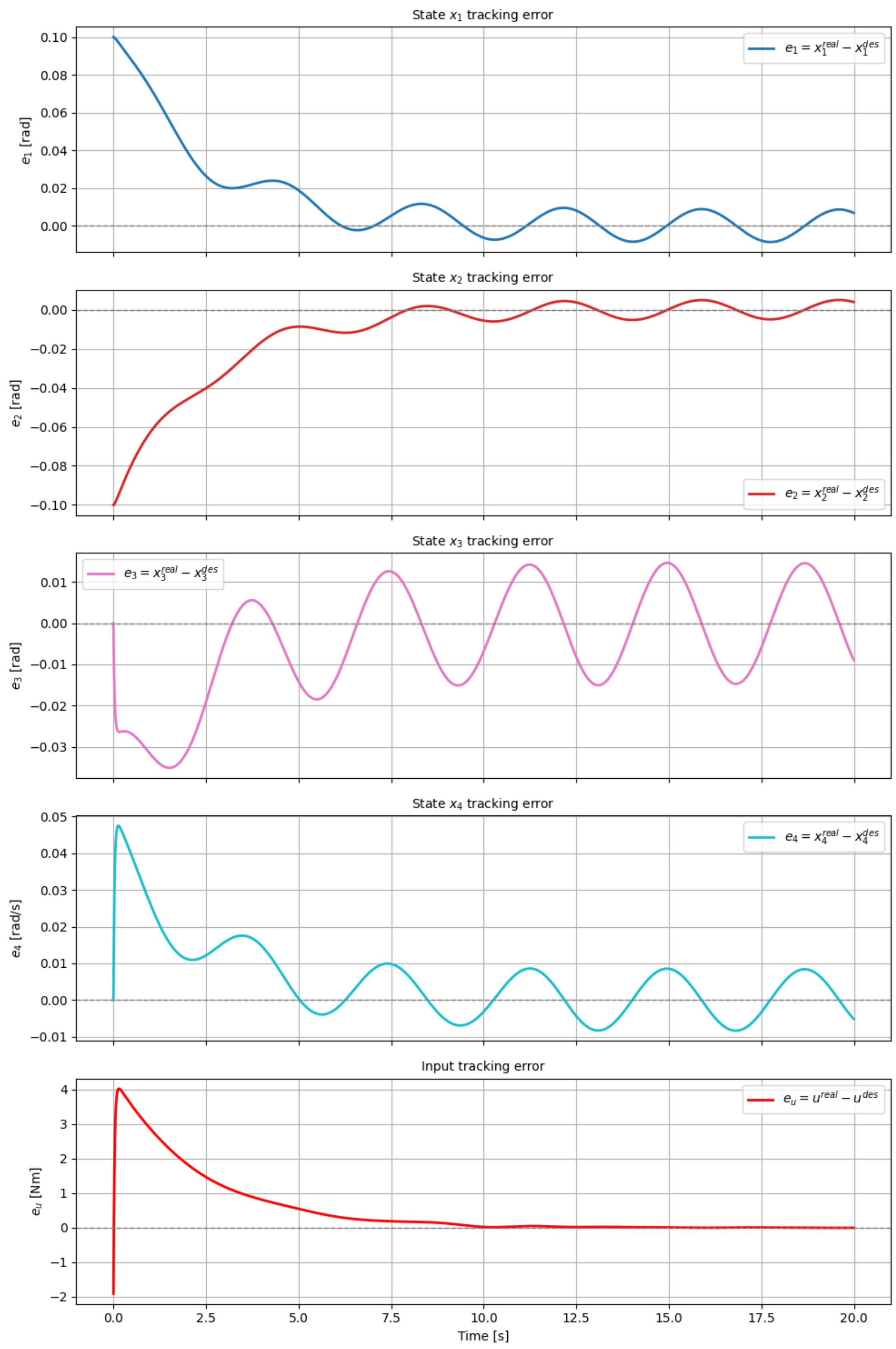
$$Q_T^{reg} = Q_t^{reg}$$


State x_1 trajectory tracking
x_1^{real}
x_1^{des}
x_1 [rad]
State x_2 trajectory tracking
x_2^{real}
x_2^{des}
x_2 [rad]
State x_3 trajectory tracking
x_3^{real}
x_3^{des}
x_3 [rad]
State x_4 trajectory tracking
x_4^{real}
x_4^{des}
x_4 [rad/s]
Input trajectory tracking
u^{real}
u^{des}
u [Nm]
Time [s]


State x_1 tracking error
$e_1 = x_1^{real} - x_1^{des}$
e_1 [rad]
State x_2 tracking error
$e_2 = x_2^{real} - x_2^{des}$
e_2 [rad]
State x_3 tracking error
$e_3 = x_3^{real} - x_3^{des}$
e_3 [rad]
State x_4 tracking error
$e_4 = x_4^{real} - x_4^{des}$
e_4 [rad/s]
Input tracking error
$e_u = u^{real} - u^{des}$
e_u [Nm]
Time [s]

With our implemented mpc control algorithms we got better performance considering the trajectory generated in task 1, even if not as good as the ones obtained for lqr regulator. This is possible to see in the plots below (corresponding to the same parameters described previously for swing-up trajectory) where, in particular for the states 1 and 2 where the noise is applied, the tracking error decreases and goes oscillating toward zero.

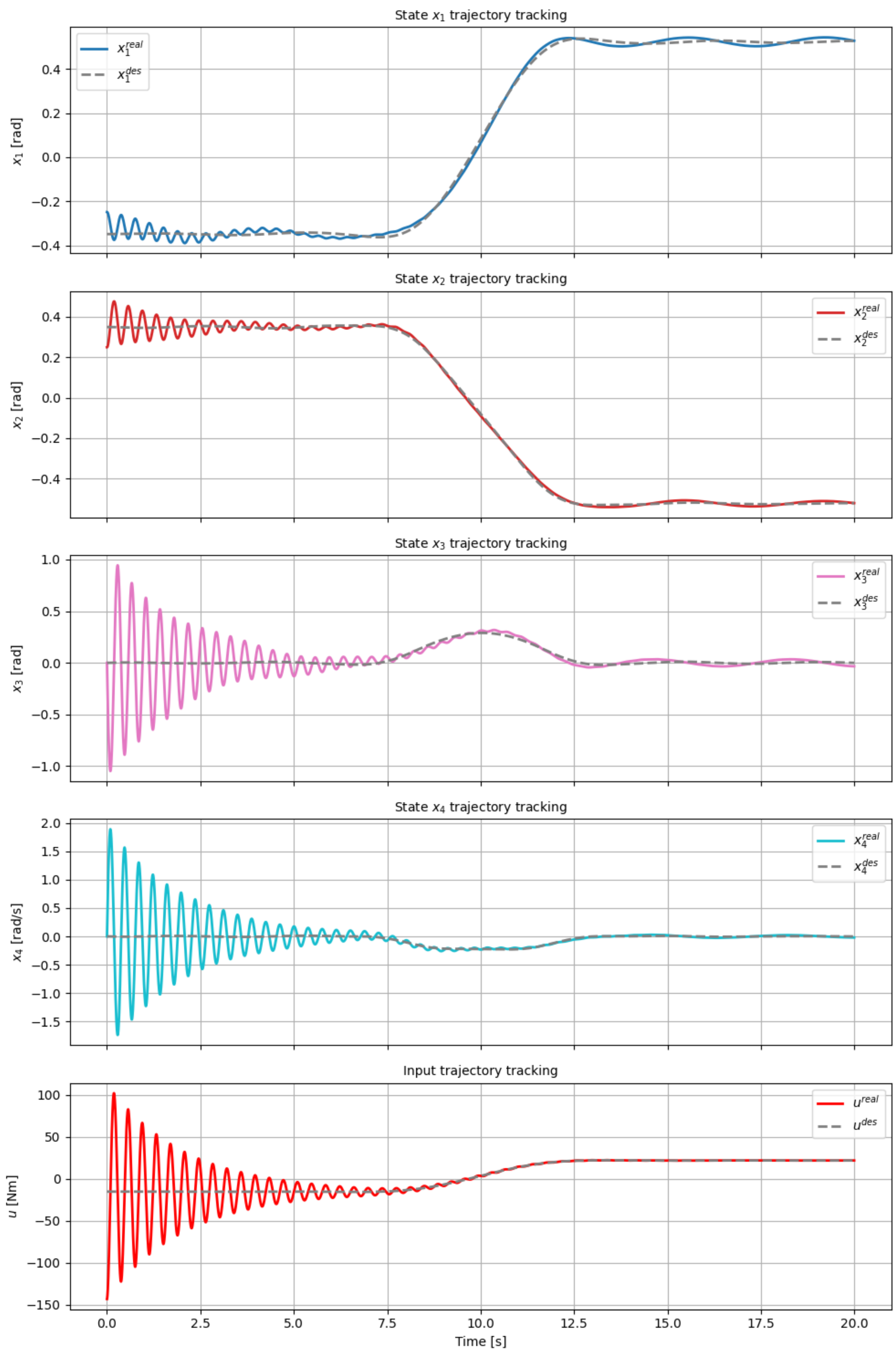




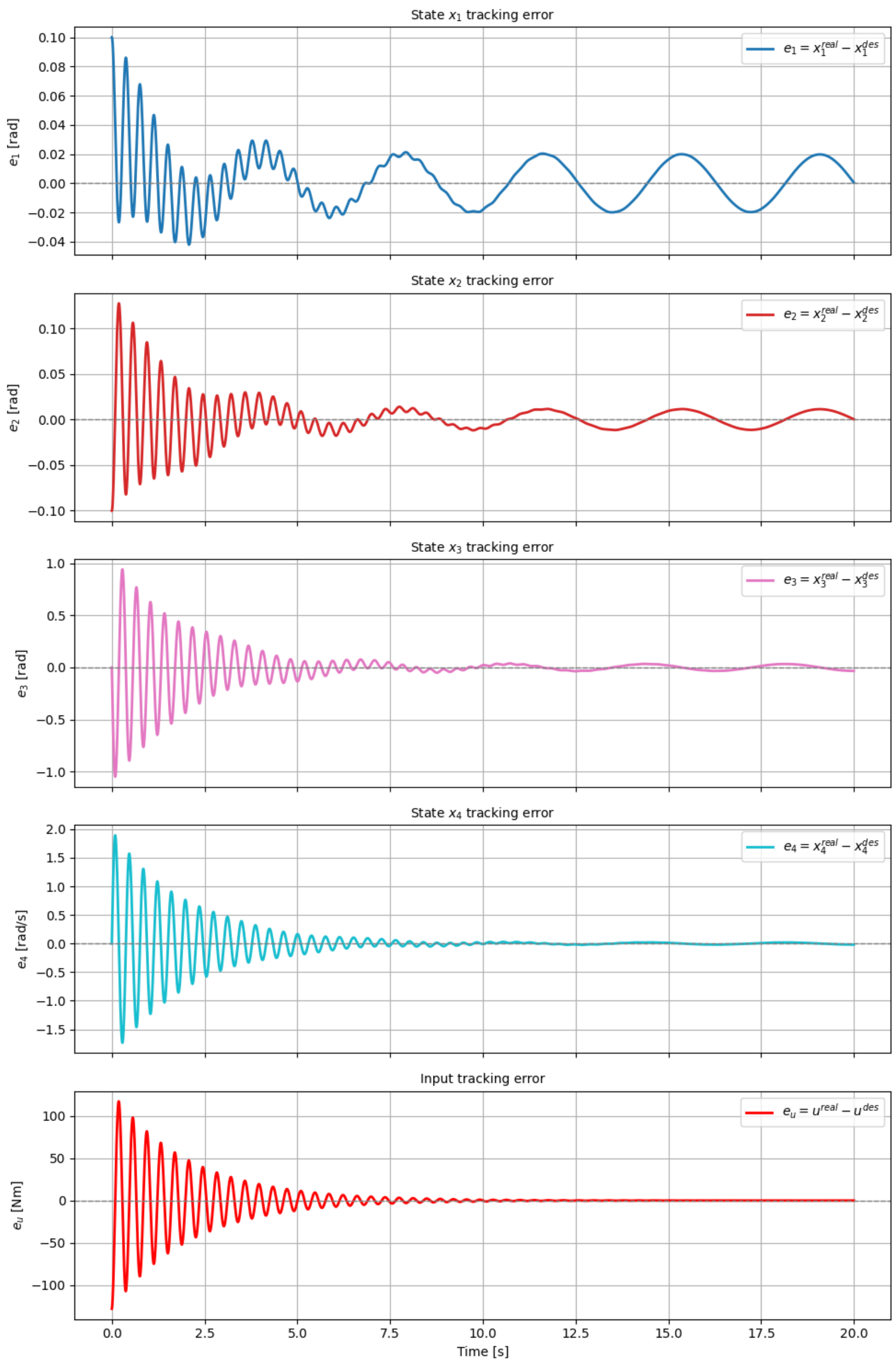
If we want to get better resukts at the final time, in particular for the angular positions, it is possible to decrease the weights for the angular velocities and increase the weight at the final time. Considering the following weight matrices:

$$Q_t^{reg} = \mathrm{diag}(100000, 100000, 0.1, 0.1)$$
$$R_t^{reg} = 0.001 \cdot I_{n_i}$$
$$Q_T^{reg} = 10 \cdot Q_t^{reg}$$

we get the tracking plots below where it is possible to observe a zero tracking error of the angular positions at the final time:



State x_1 trajectory tracking
x_1^{real}
x_1^{des}
x_1 [rad]
State x_2 trajectory tracking
x_2^{real}
x_2^{des}
x_2 [rad]
State x_3 trajectory tracking
x_3^{real}
x_3^{des}
x_3 [rad]
State x_4 trajectory tracking
x_4^{real}
x_4^{des}
x_4 [rad/s]
Input trajectory tracking
u^{real}
u^{des}
u [Nm]
Time [s]


State x_1 tracking error
$e_1 = x_1^{real} - x_1^{des}$
e_1 [rad]
State x_2 tracking error
$e_2 = x_2^{real} - x_2^{des}$
e_2 [rad]
State x_3 tracking error
$e_3 = x_3^{real} - x_3^{des}$
e_3 [rad]
State x_4 tracking error
$e_4 = x_4^{real} - x_4^{des}$
e_4 [rad/s]
Input tracking error
$e_u = u^{real} - u^{des}$
e_u [Nm]
Time [s]

Chapter 6

Task 5 - Animation of the robot executing task 3

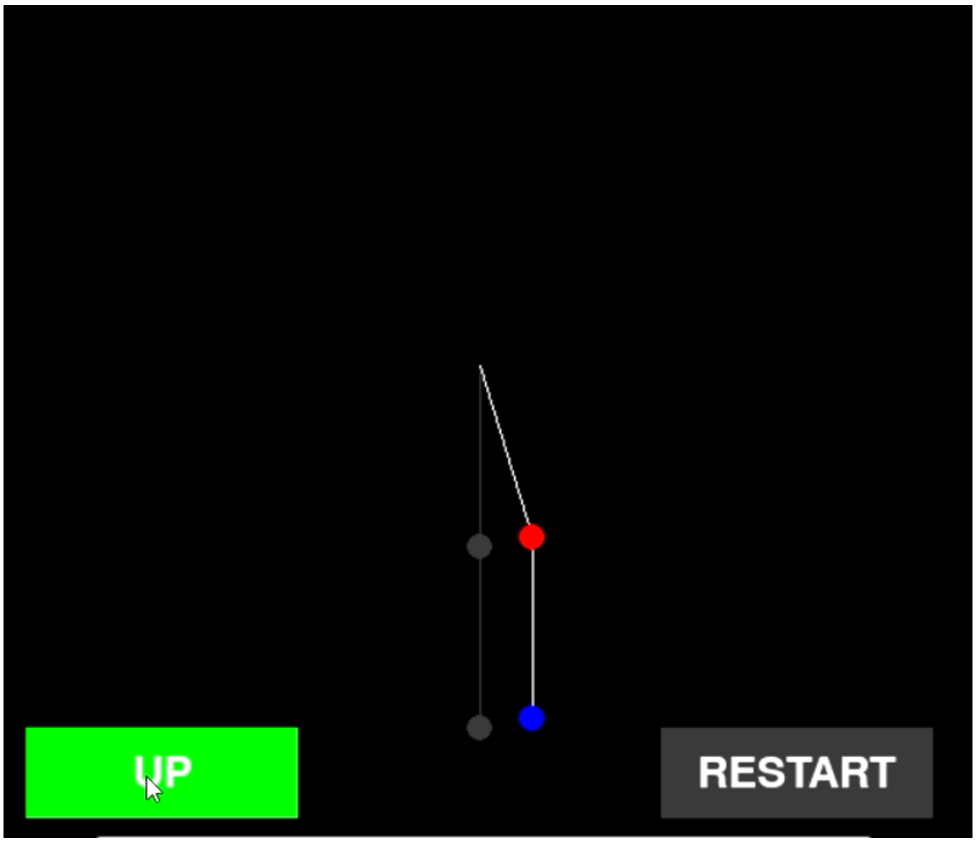
For task 5, the objective is to create an animation of the robot that executes task 3.

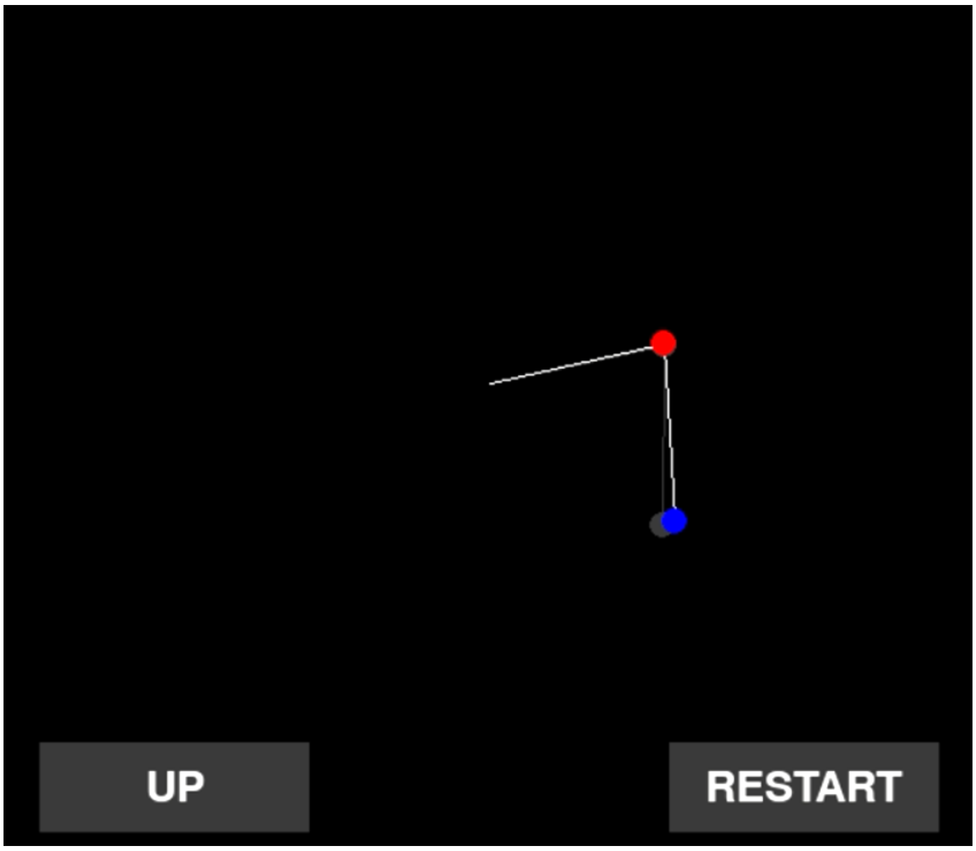
The animation has been realized by using the Python library Pygame. The simulation includes interactive buttons: "UP" to start or the motion and "RESTART" to start again the simulation.

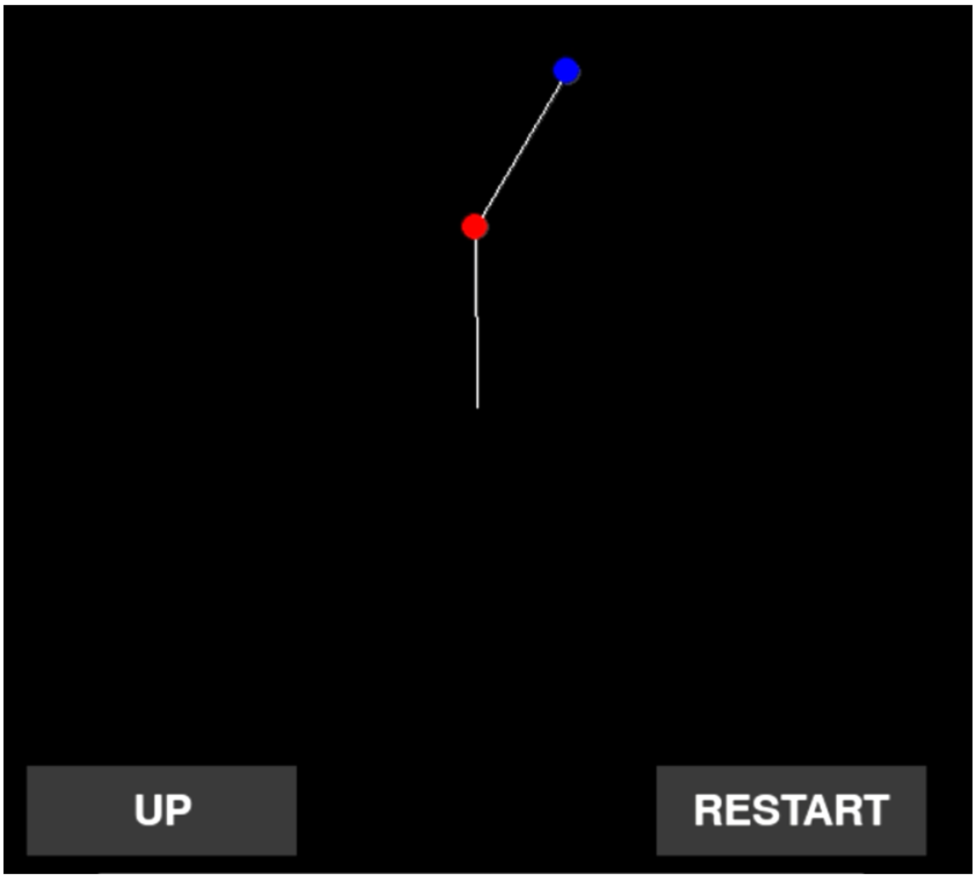
In the pictures below are reported four frames of the robot executing task 3 in case the noise in the initial condition is:

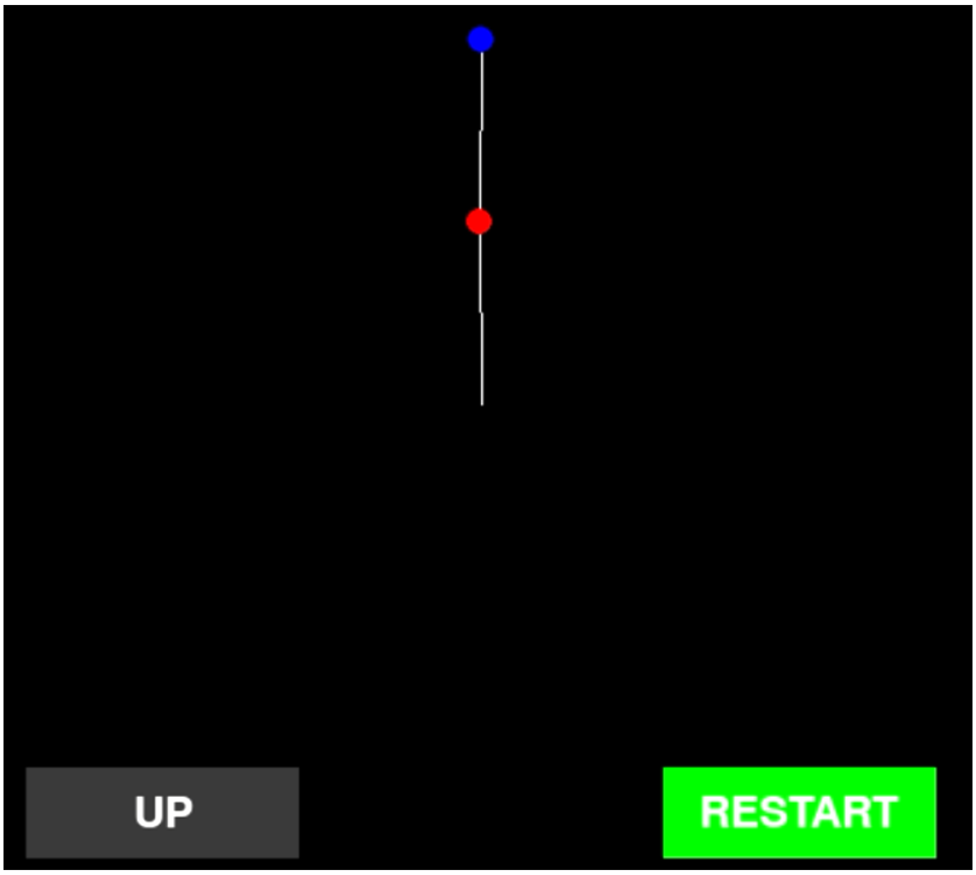
$$noise = \begin{bmatrix} 0.3 \\ -0.4 \\ 0 \\ 0 \end{bmatrix}$$

The coloured pendolum represents the real trajectory, while the gray one is related to the optimal reference trajectory. According to his, the first frame corresponds to the perturbed initial condition (colured) and the ideal one (gray).


UP
RESTART


UP
RESTART


UP
RESTART


UP
RESTART

Conclusions

In this project, we applied optimal control techniques, in particular Newton's method, LQR and MPC, to design and track an optimal trajectory for a flexible robotic arm. We have to underline the power of optimal control that allowed us to control an underactuated system, since the planar-two links robot was actuated with torque only at the first joint.

Beyond the specific case of the flexible robotic arm, the versatility of optimal control is evident in its broader applicability. These methods are not only crucial for robotics but can be extended to various domains requiring precise control over complex systems. For example, in aerospace engineering, optimal control is used to design flight paths and optimize fuel consumption in underactuated aircraft. In the automotive industry, it plays a key role in autonomous driving, where real-time trajectory planning and path tracking are essential for safety and efficiency. Additionally, in medical robotics, optimal control techniques can be applied to guide surgical robots with high accuracy, ensuring patient safety and precision.

In conclusion, this project reinforces the critical role of optimal control in modern engineering, showcasing how it enables the control of systems that would otherwise be challenging to manage.

Bibliography

[1] D. Bertsekas, *Nonlinear Programming*, Athena Scientific, 1999.

[2] A. E. Bryson and Y. Ho, *Applied Optimal Control: Optimization, Estimation, and Control*, CRC Press, 1975.

[3] Slides of the *Optimal Control M* course