UNIVERSITÀ DI BOLOGNA

School of Engineering

Master Degree in Automation Engineering

Optimal Control

Optimal Control of a Ball and Beam system

Professor: **Giuseppe Notarstefano**

Students:
Nicholas Baraghini
Fabio Curto
Federico Iadarola

Academic year 2021/2022

Abstract

In this paper, an optimal control method for a ball and beam system is presented. The main goal is to feed the system with an arbitrarily reference trajectory and to control it in order to execute the optimal trajectory. To achieve it, a differential dynamic programming algorithm (DDP) has been implemented. First, the algorithm has been employed to calculate the optimal trajectory of the ball moving from one equilibria to another. Next, exploiting the DDP algorithm, from the desired trajectory an optimal trajectory has been generated for the system to follow. Finally, using a Linear Quadratic Regulator (LQR) algorithm, the previously obtained optimal trajectory has been tracked by the ball. For all the tasks, an animation showing the results has been displayed. The project has been carried out in Python language

Contents

Introduction

Motivations

The ball and beam system is one of the most enduringly popular and important laboratory models for teaching control systems engineering. It is a very simple system to be understood and implemented yet interesting to be studied since the system is open-loop unstable. The system is governed by non-linear dynamics equations and has two independent degrees of freedom, the position of the ball on the beam and the angle of the beam respect a reference frame. The control objective is to control the torque τ applied at the pivot of the beam, such that the ball can roll on the beam and compute correctly a desired trajectory. Even though several control techniques have been already implemented for this problem, exploiting an optimal control approach on this system can be a good teaching exercise, giving also the possibility to test the results obtained on a prototype.

To achieve the objective, the model of the system has been discretized by means of the Euler-Lagrange method. In the first task, the DDP algorithm has been exploited to find the optimal transition of the ball stepping from an equilibria to another one. Then, after having designed an arbitrary reference trajectory for the ball, the DDP algorithm is again used to compute the optimal trajectory following the desired one. In conclusion, the ball is called to track the reference trajectory defined. This is achieved by linearizing the system dynamics about the optimal trajectory, obtained in the previous task, and exploiting the LQR algorithm to obtain the optimal feedback controller. Plots and visual animation of the system are produced during the task progress, to show the results and the system evolution.

Contributions

- **Nicholas Baraghini**: Write the proposal of the project. Implemented the dynamics and contributed to DDP algorithm coding in python. Implemented task1 and task1.2, contributed to task2
- **Fabio Curto**: Implemented the Visualization, the trajectory desired in python and MATLAB. Completed the task2
- **Federico Iadarola**: Contributed to the definition of the dynamics, to the DDP algorithm and Armijo code in python. Implemented the task3, contributed to task2

Chapter 1

Task 0 - Dynamics setup

The control objective is to control the torque τ applied at the pivot of the beam, such that the ball can roll on the beam and track the desired trajectory. The torque causes thus a change of the beam angle and a movement in the position of the ball. The ball rolls on the beam without slipping under the action of the force of gravity. Defining the generalized coordinate:

$$q(t) = \begin{bmatrix} p(t) \theta(t) \end{bmatrix}$$

containing the 2 D.o.F. of the system: the position of the ball and the angle of the beam, respectively. Exploiting the Lagrange method, the dynamics of the system are extracted:

$$(\frac{J_b}{r^2} + m)\ddot{p} + mg\sin(\theta) - mp\dot{\theta^2} = 0$$

$$(mp^2 + J)\ddot{\theta} + 2mp\dot{p}\dot{\theta} + mgp\cos(\theta) = \tau$$

where:

- J_b :inertia matrix of the ball
- J: inertia matrix of the beam
- r: radius of the ball
- m : mass of the ball

The equations of motion we derived for the Ball and Beam system can be written in state-variable representation. Defining the state vector:

$$x(t) = \begin{bmatrix} x_1(t) \\ x_2(t) \\ x_3(t) \\ x_4(t) \end{bmatrix} = \begin{bmatrix} p(t) \\ \dot{p}(t) \\ \theta(t) \\ \dot{\theta}(t) \end{bmatrix}$$

The equation of motion can be written in terms of the state variable as

$$\dot{x}(t) = \begin{bmatrix} \dot{x_1}(t) \\ \dot{x_2}(t) \\ \dot{x_3}(t) \\ \dot{x_4}(t) \end{bmatrix} = \begin{bmatrix} x_2 \\ \frac{m(x_1 x4^2 - g\sin x_3)}{\frac{J_b}{r^2}+m} \\ x_4 \\ \frac{-2mx_1x_2x_4 - mgx_1\cos(x_3)+\tau}{mx_1^2+J} \end{bmatrix} = f(x,\tau)$$

Applying direct Euler, the resulting discrete-time system is:

$$\begin{bmatrix} x_{1,t+1} \\ x_{2,t+1} \\ x_{3,t+1} \\ x_{4,t+1} \end{bmatrix} = \begin{bmatrix} x_{1,t} \\ x_{2,t} \\ x_{3,t} \\ x_{4,t} \end{bmatrix} + \delta t \begin{bmatrix} \dot{x_1}(t) \\ \dot{x_2}(t) \\ \dot{x_3}(t) \\ \dot{x_4}(t) \end{bmatrix}$$

Through the gradient respect the input and the state we linearize the dynamics:

$$A = \nabla_x f(x,\tau) = \begin{bmatrix} 1 & \frac{mx_3^2}{\frac{J_b}{r^2}+m}\delta t & 0 & \frac{d_1}{(mx_1^2+J)^2}\delta t \\ \delta t & 1 & 0 & \delta t\frac{-2mx_1mx_4}{(mx_1^2+J)^2} \\ 0 & \frac{\delta t}{\frac{J_b}{r^2}+m}(-mg\cos x_3 & 1 & \delta t\frac{mgx_1\sin x_3}{(mx_1^2+J)^2} \\ 0 & \frac{\delta t}{\frac{J_b}{r^2}+m}(2mx_1x_3 & \delta t & 1-\delta t\frac{-2mx_1mx_4}{(mx_1^2+J)^2} \end{bmatrix}$$

where:

$$d1 = -(2mx_2x_4 - mgx_1\cos(x_3)(mx_1^2+J) - (-2mx_1x_2x_4 - mgx_1\cos(x_3)+\tau)2mx_1$$

$$B = \nabla_u f(x,\tau) = \begin{bmatrix} 0 \\ 0 \\ 0 \\ \frac{\delta t}{(mx_1^2+J)^2} \end{bmatrix}$$

Obtaining the linear system: $\Delta x_{t+1} = A\Delta x_t + B\Delta u_t$

For the formulation and solution of the DDP algorithm, we also compute the dynamic gradients and implement the tensor product. All this element are computed and returned from the `BBDynamics()` function in the module `systemdynamic.py`

Chapter 2

Task 1 - Trajectory exploration

In the first task, two arbitrary points along the beam were chosen and an optimal transition was computed using a step-trajectory as reference.

The algorithm proposed to solve this task calls three main functions:

- **DDP_comp_t_k()**: This function is responsible for calculating the matrices K, σ, P, and p defined by the DDP algorithm, designated at the k-th iteration.
  - $K_t^k = -(l_{uu,t}^k + f_{u,t}^k \cdot P_{t+1}^k \cdot f_{u,t}^{k,T} + f_{uu,t}^k \cdot p_{t+1}^k)^{-1} \cdot (l_{ux,t}^k + f_{u,t}^k \cdot P_{t+1}^k \cdot f_{x,t}^{k,T} + f_{ux,t}^k \cdot p_{t+1}^k)$
  - $\sigma_t^k = -(l_{uu,t}^k + f_{u,t}^k \cdot P_{t+1}^k \cdot f_{u,t}^{k,T} + f_{uu,t}^k \cdot p_{t+1}^k)^{-1} \cdot (l_{u,t}^k + f_{u,t}^k \cdot p_{t+1}^k)$
  - $P_t^k = -(l_{xx,t}^k + f_{x,t}^k \cdot P_{t+1}^k \cdot f_{x,t}^{k,T} + f_{xx,t}^k \cdot p_{t+1}^k) - K_t^{k,T} \cdot (l_{uu,t}^k + f_{u,t}^k \cdot P_{t+1}^k \cdot f_{u,t}^{k,T} + f_{uu,t}^k \cdot p_{t+1}^k) \cdot K_t^k$
  - $p_t^k = -(l_{x,t}^k + f_{x,t}^k \cdot p_{t+1}^k) - K_t^{k,T} \cdot (l_{uu,t}^k + f_{u,t}^k \cdot P_{t+1}^k \cdot f_{u,t}^{k,T} + f_{uu,t}^k \cdot p_{t+1}^k) \cdot \sigma_t^k$

  for t = T-1, ..., 0 With :
  - $P_T^k = l_{xx,T}^k$
  - $p_T^k = l_{x,T}^k$
- **Armijo()**: This function has been implemented to search for an approximate optimal step size at each iteration. By making use of the implementation of the Armijo algorithm.
  Therefore setting:
  - $\gamma^0 = 1 (> 0)$
  - $\beta = 0.5 (\in [0, 1])$
  - $c = 0.05 (\in [0, 1])$

Then: $\gamma^{i+1} = \beta \cdot \gamma^i$
Until: $l(x^k + \gamma^i \cdot d^k) \leq l(x^k) + c \cdot \gamma^i \nabla l(x^k)^T \cdot d^k$

- **Trajectory_Update()**: This function updates the optimal trajectory that occurs between one iteration and its next one. Therefore, making use of the fundamental DDP matrices calculated by DDP_comp_t_k and by the optimal step-size computed by Armijo, this function takes care of the calculation of the input and the optimal state of the system at the (k + 1)-th iteration.
  Thus for t = 0, ..., T-1:
  - $u_t^{k+1} = u_t^k + K_t^k \cdot (x_t^{k+1} - x_t^k) + \sigma_t^k$
  - $x_{t+1}^{k+1} = f(x_t^{k+1}, u_t^{k+1})$
  - $x_{0,t}^{k+1} = x_{init}$

The three functions just defined are then performed iteratively until the descent d^k falls below a certain threshold or until the number of iterations does not exceed a certain maximum value of imposed cycles.
The function above presented, belongs to the module `optcon.py`.
For investigation purposes, the optimal trajectory has been computed also for a multi-step reference trajectory of the ball position along the beam.

Single Step

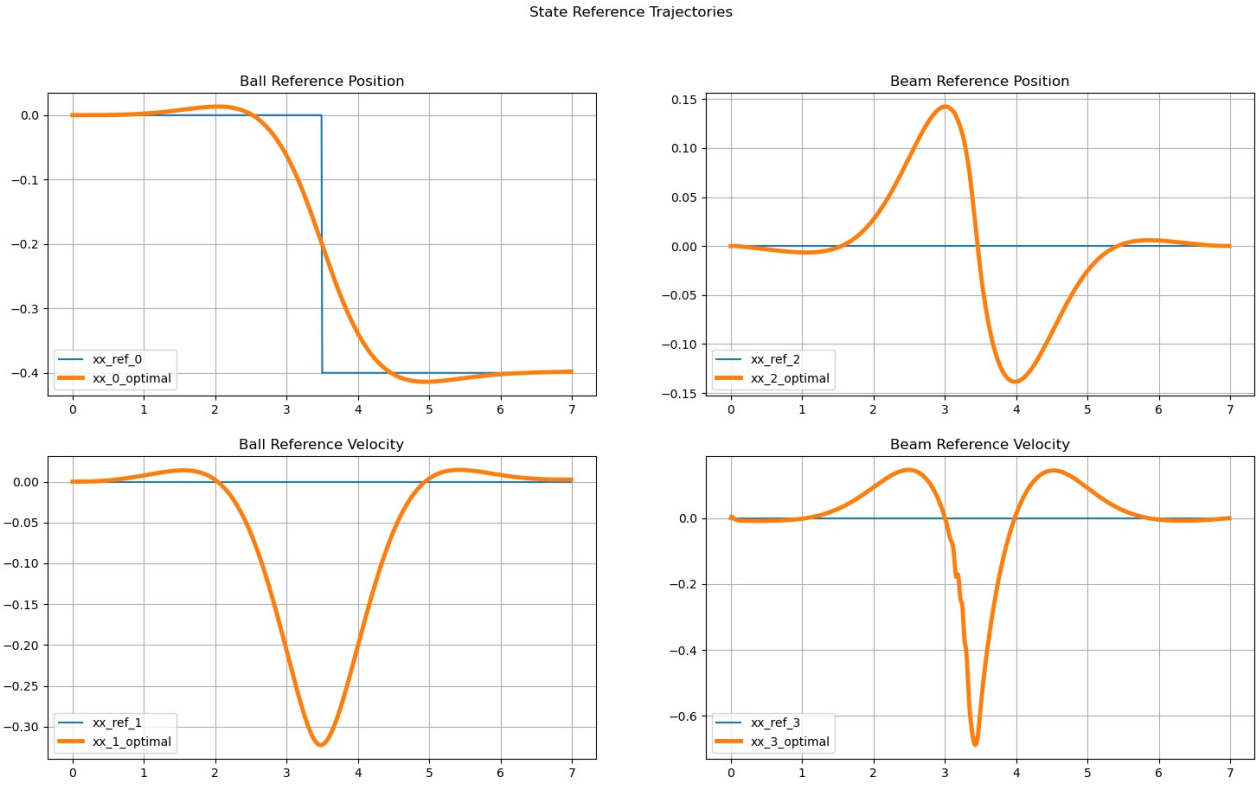

Figure 2.1: Comparison between the reference state and the optimal state computed by the DDP algorithm

In this case, a step reference trajectory was created for the position of the ball. Over 7 seconds of the experiment, the ball is required to stop in the central position of the beam, for the first half of the period; While for the remaining half of the time, the ball should be in a position close to the left end of the beam. Considering a reference system fixed in the beam pivot and oriented as the slope of the beam itself, then the reference step trajectory has an initial value of 0 and a final value of $-0.8 \cdot \frac{L}{2}$, of which L is considered the length of the beam.

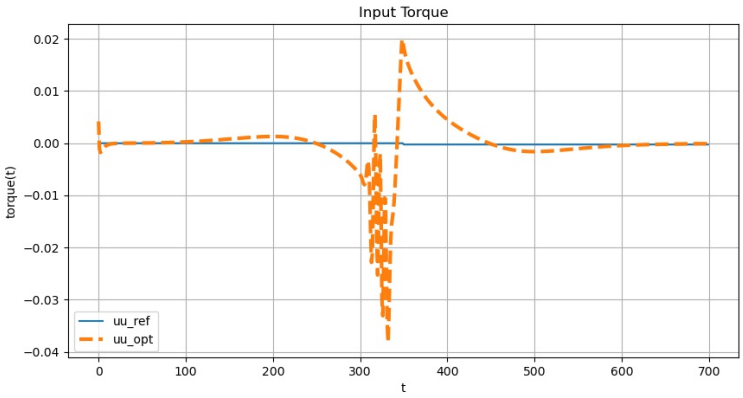

Figure 2.2: Input torque

As can be seen from the graphs, the trajectory of both the input and the speeds are subjected to zero, and therefore properly weighing the contribution to these states in the cost function, the optimal sector will not only try to position the ball precisely but also execute the trajectory thus minimizing both speed and input torque.

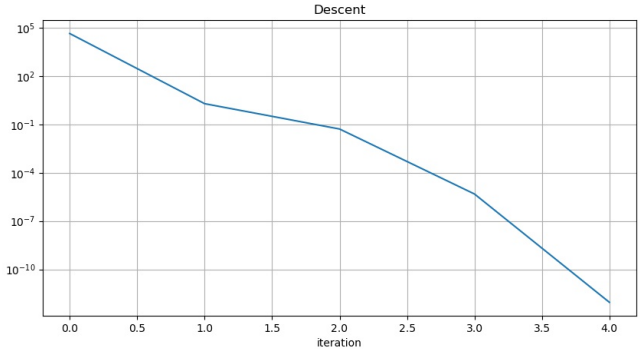

Figure 2.3: Descent

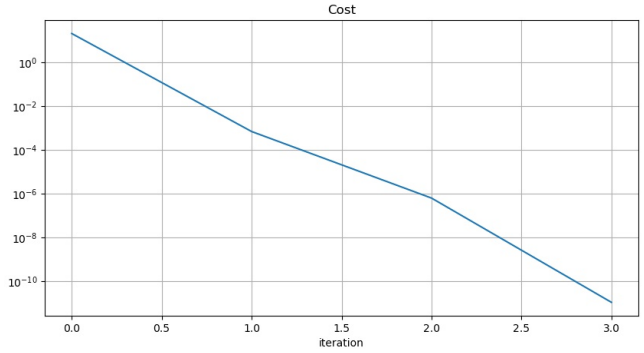

Figure 2.4: Cost

From the descent and cost graphs, it can be seen that the algorithm reaches very low descent values in a small number of iterations thus leading to fast convergence of the algorithm to the optimal trajectory.

Multiple Step

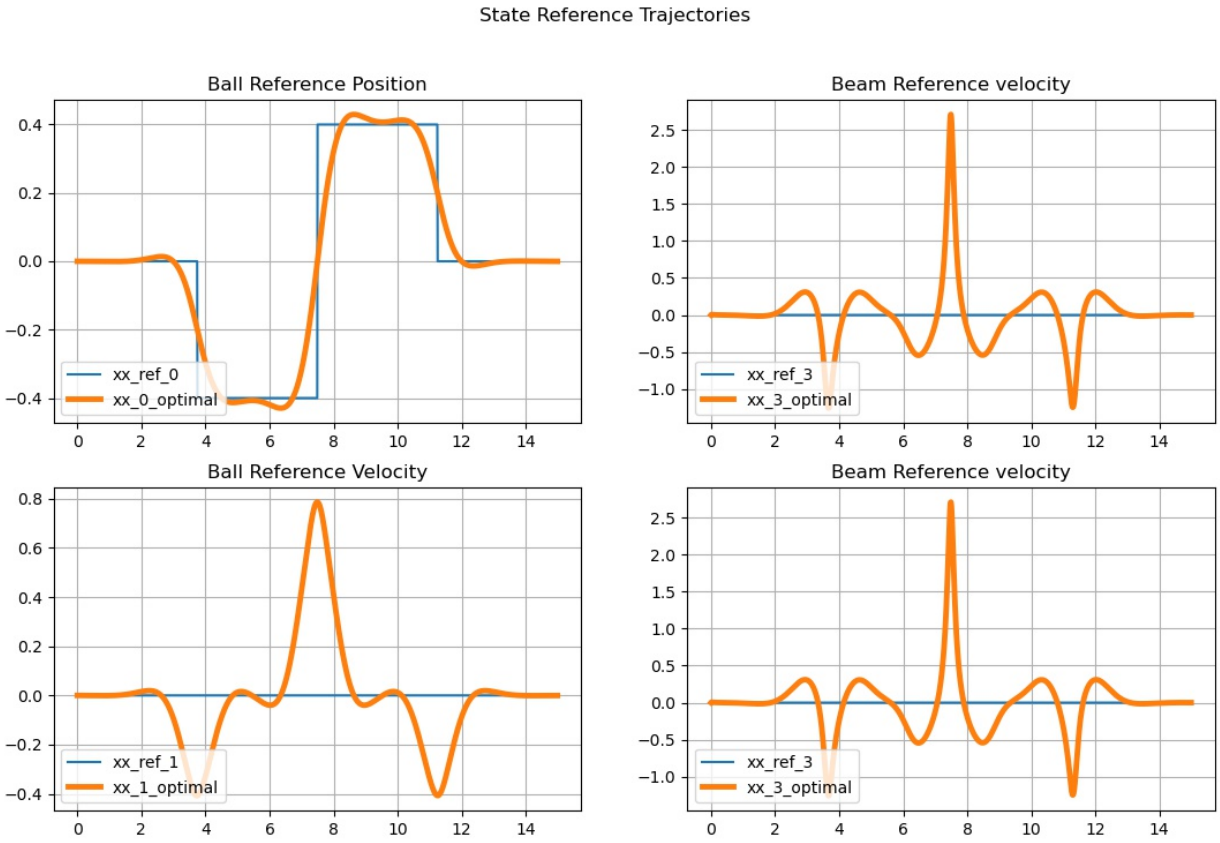

Figure 2.5: Comparison between the reference state and the optimal state computed by the DDP algorithm

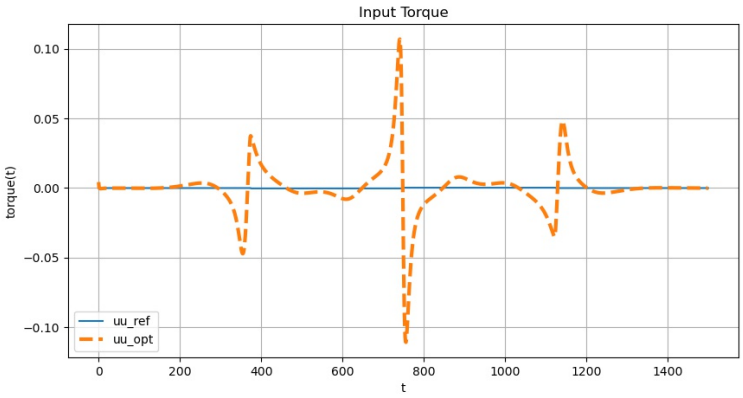

Figure 2.6: Input torque

Similarly to what was done for a single-step trajectory, it was generalized for a multiple-step reference trajectory, where the ball was required to be in different positions on the shaft, in sundry homogeneous time intervals, for a total duration of the experiment of 15 seconds

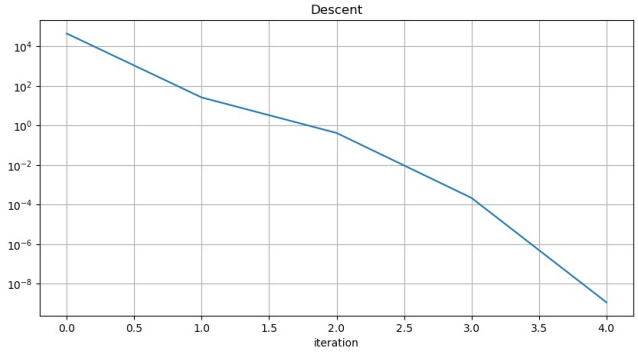

Figure 2.7: Descent

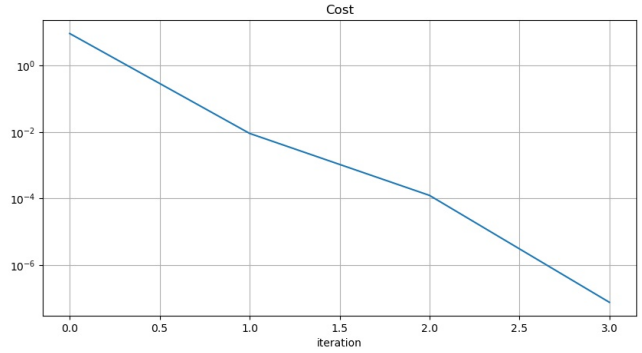

Figure 2.8: Cost

Even for a multi-step trajectory, there is a fast convergence of the optimization algorithm

Chapter 3

Task 2 - Trajectory Optimization

In the second task the DDP algorithm has been exploited in order to obtain an optimal trajectory, from a properly design reference trajectory of the ball.

Quasi-Stationary Trajectory

The reference curve has been designed inside the module `ReferenceTrajectory.py` by the function `SplineRef()`. This function require:

- waypoints: points on the x axes that will be interpolated
- time points: time instants when the waypoints are reached

and return a cubic polynomial, which is twice continuously differentiable, obtained interpolating the given data.

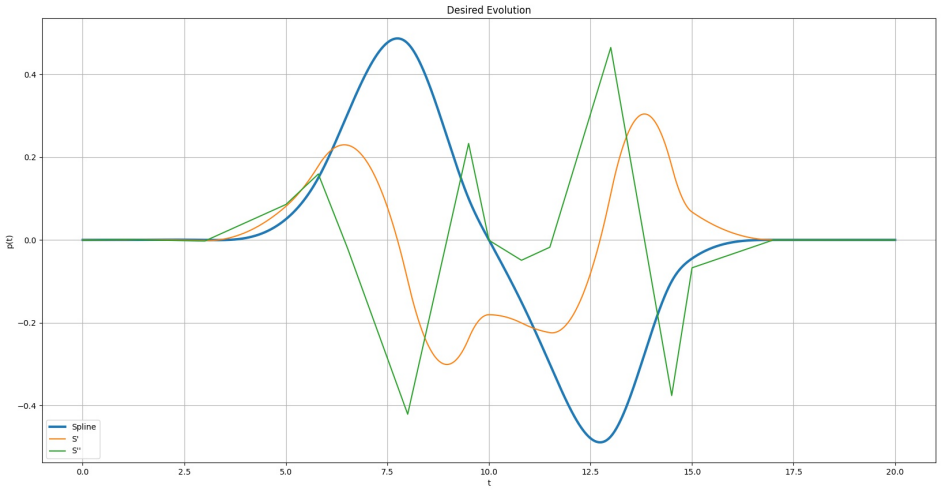

Figure 3.1: Reference trajectory, reference velocity and acceleration

Even if the evolution of the beam position, of the beam velocity and of the input are not known, computing the input in quasi static condition make possible to exploit and find the optimal trajectory through the DDP algorithm.
Starting from $p(t)$, $\dot{p}(t)$, $u_{qs}(t)$ for t= 0, ..., T the otpimal trajectory $(\mathbf{x}^{opt}, \mathbf{u}^{opt})$ is computed.
The following plots show the results computed with the algorithm.

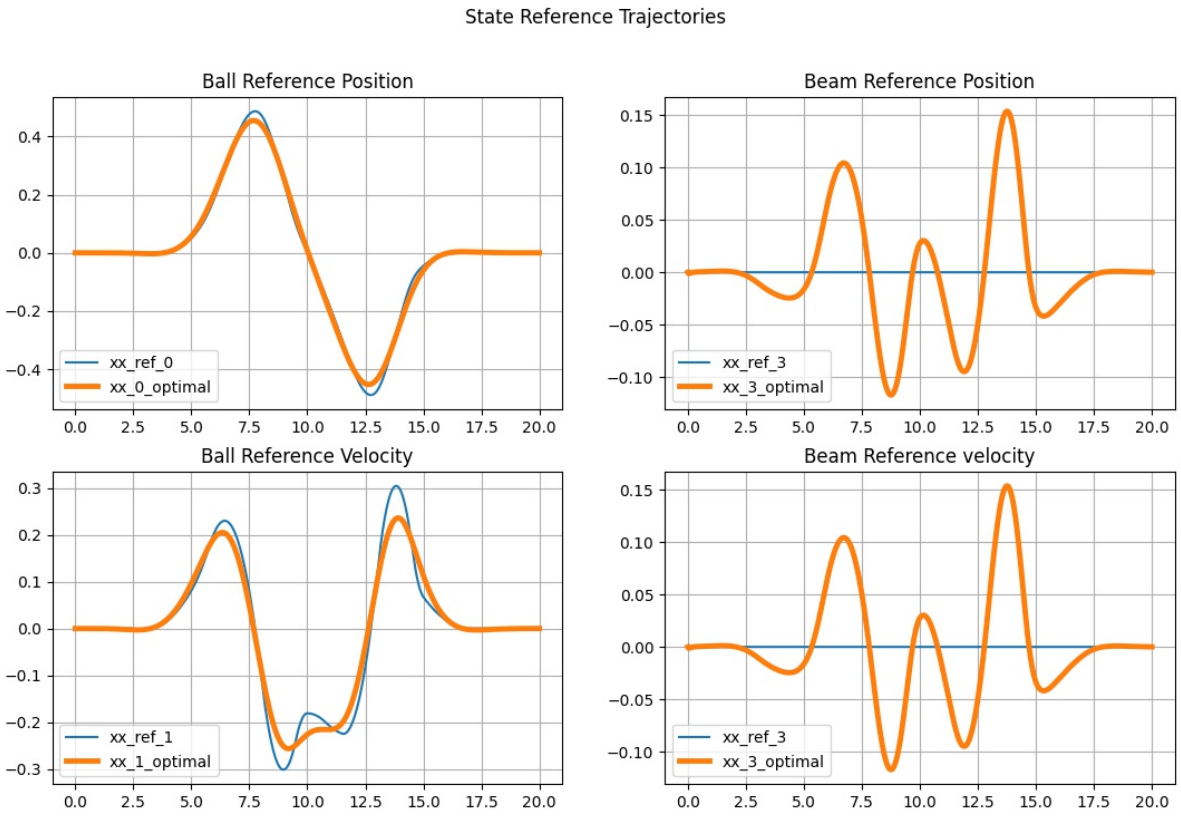

Figure 3.2: Comparison between the reference state to follow and the optimal one computed using DDP

Since the cost matrix gives more weights to position of the ball, the optimal trajectory try to follow it as close as possible, adapting the missing state in order to have a trajectory feasible and optimal.

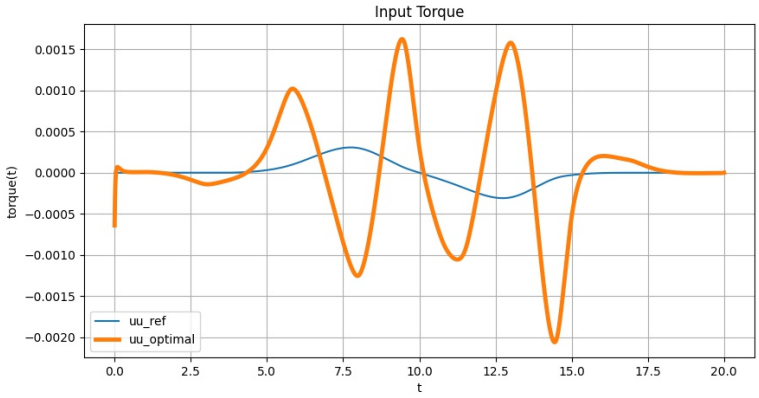

Figure 3.3: optimal input reference to apply in order to obtain the optimal trajectory

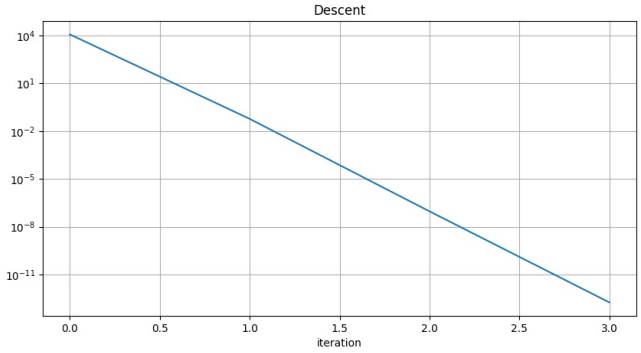

Figure 3.4: Descent

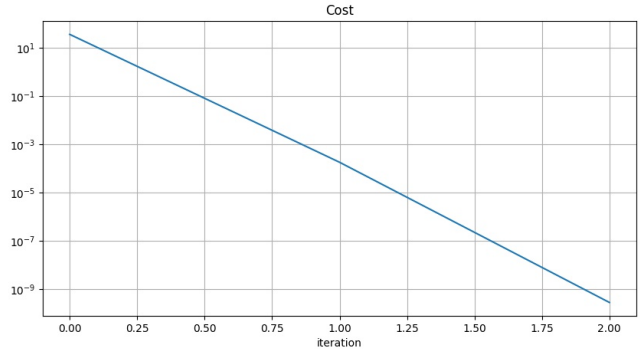

Figure 3.5: Cost

Refined Trajectory

Alternatively, to overcame the absence of state evolution, a regulator exploiting the dynamic of the system has been implemented; Therefore feeding the controller with the position and the velocity of the ball the regulator finds the most close beam angle and velocity that satisfies the dynamics equations; The simulation has been implemented in Simulink.
Also in this case, a cubic polynomial has been used as reference curve, exploiting the Simulink block `Polynomial Trajectory`.
The controller implemented is a PID controlled properly tuned.
In particular:

- Starting from the desired acceleration of the ball, an angular reference is calculated. The angle to follow is the one giving the ball the gravitational acceleration matching the desired trajectory.
- The inner PID loop controls the angular position θ without considering the final trajectory of the ball.

This method can be seen as feed-forward from the point of view of the ball position, in fact it is based on the perfect knowledge of the system model and does not control in any way the ball position in the presence of any uncertainties or disturbances.
In the case under study, this type of controller is sufficient because the only purpose is to generate a reference, complete of all states and input, which will be processed by the optimal control algorithm.

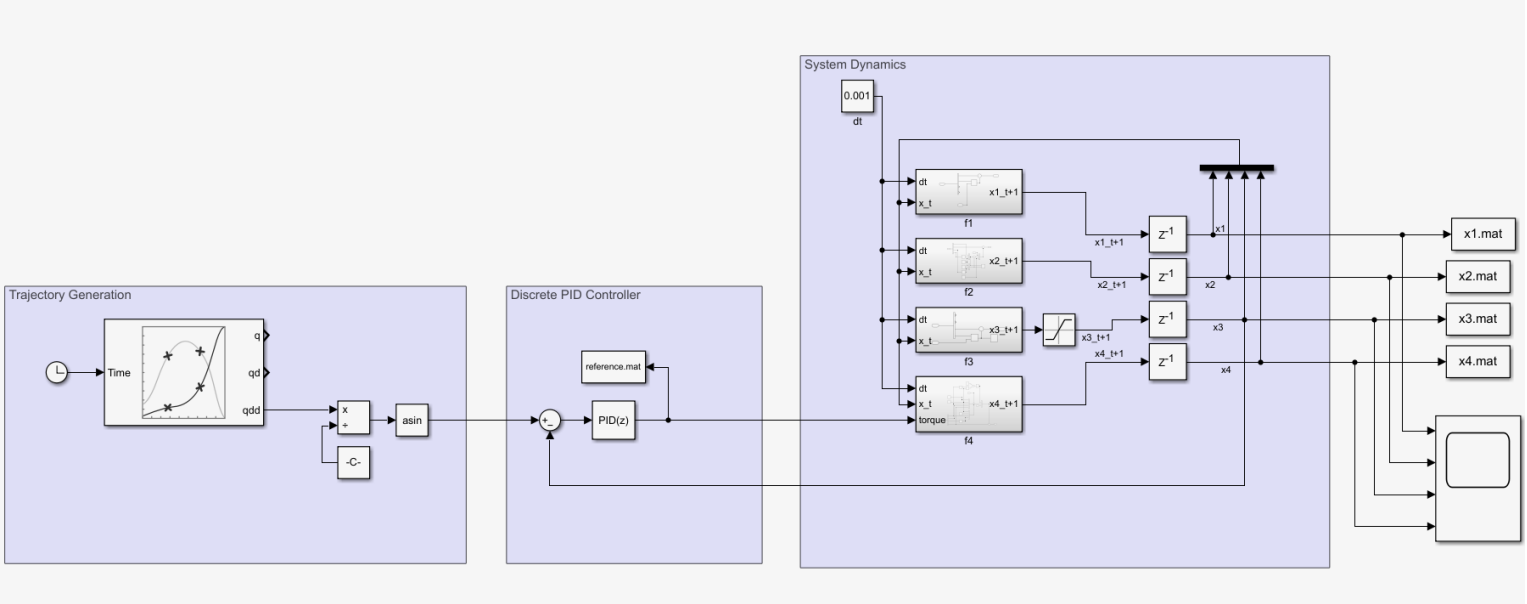

Figure 3.6: Simulink Scheme

As mentioned earlier, after extracting the refined trajectory, the DDP algorithm was applied over the new desired reference and the results are show below.

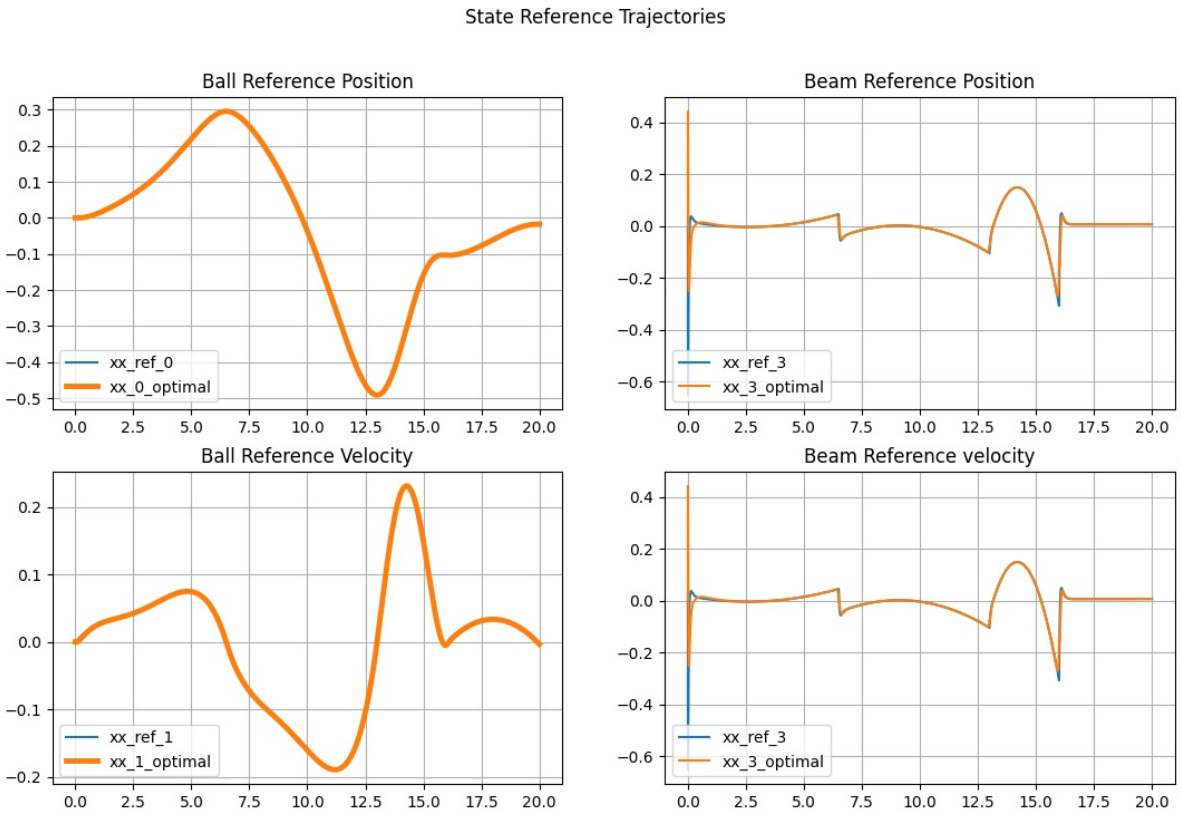

Figure 3.7: Comparison between the reference state to follow and the optimal one computed using DDP

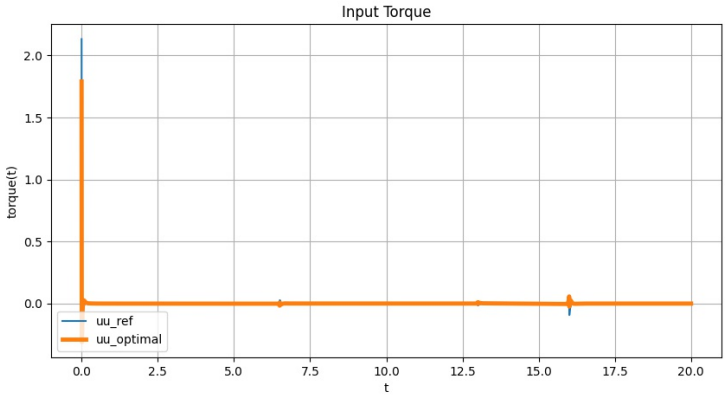

Figure 3.8: optimal input reference to apply in order to obtain the optimal trajectory

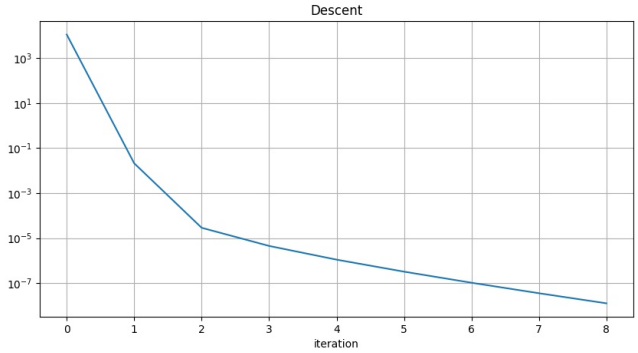

Figure 3.9: Descent

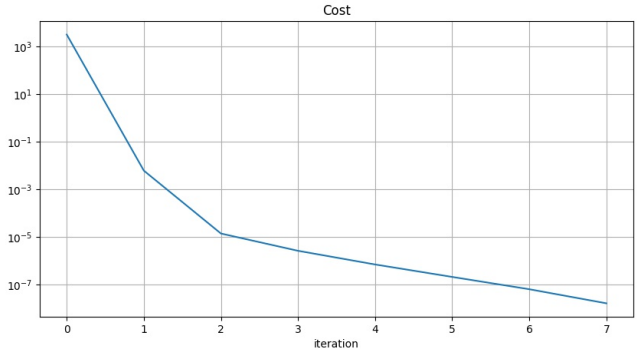

Figure 3.10: Cost

Chapter 4

Task 3 - Trajectory Tracking

The objective of this task is to control the non-linear ball and beam system along an evolution, also satisfying some performance criteria. The idea is to track the optimal trajectory $(\mathbf{x}^{opt}, \mathbf{u}^{opt})$ computed in Task 2 exploiting LQR to define the optimal feedback controller.

Step 1 - Linearization of the system
The dynamics have been approximated about the trajectory $(\mathbf{x}^{opt}, \mathbf{u}^{opt})$ via the linear system:

$$\Delta x_{t+1} = A_t^{opt} \Delta x_t + B_t^{opt} \Delta u_t$$

where $A_t^{opt} \in \Re^{4x4}$, $B_t^{opt} \in \Re^{4x1}$ are defined as:

$$A_t^{opt} = \nabla_{x_t^{opt}} f(x_t^{opt}, u_t^{opt})^T$$

$$B_t^{opt} = \nabla_{u_t^{opt}} f(x_t^{opt}, u_t^{opt})^T$$

for all (x_t^{opt}, u_t^{opt}) with t = 0,....,T state-input pairs at time t of trajectory $(\mathbf{x}^{opt}, \mathbf{u}^{opt})$ with lenght T.

Step 2 - Calculate the LQR optimal controller
The following optimal control problem has been solved:

$$\begin{aligned} \min_{\substack{\Delta x_1,\dots,\Delta x_T \\ \Delta u_0,\dots,\Delta u_{T-1}}} \quad & \sum_{t=0}^{T-1} \Delta x_t^\top Q_t^{\text{reg}} \Delta x_t + \Delta u_t^\top R_t^{\text{reg}} \Delta u_t + \Delta x_T^\top Q_T^{\text{reg}} \Delta x_T \\ \text{subj.to } & \Delta x_{t+1} = A_t^{\text{opt}} \Delta x_t + B_t^{\text{opt}} \Delta u_t \qquad t = 0, \dots, T-1 \\ & \Delta x_0 = 0 \end{aligned}$$

for some cost matrices $Q_t^{reg} \geq 0 \in \Re^{4x4}$, $R_t^{reg} \geq \in \Re$ $Q_T^{reg} \geq 0 \in \Re^{4x4}$
Obtaining as return K_t^{reg} for all t = 0, ..., T-1, with shape $K_t^{reg} \in \Re^{4x4}$
The latter is the feedback gain needed to track the optimal reference curve.

Step 3 - Tracking the optimal trajectory
The feedback controller generated has been applied on the linearization to the nonlinear system in order to track $(\mathbf{x}^{opt}, \mathbf{u}^{opt})$ For all t = 0, ... ,T

$$u_t = u_t^{opt} + K_t^{opt}(x_t - x_t^{opt})$$

$$x_{t+1} = f(x_t, u_t)$$

starting from an arbitrary x_0.
This steps are performed by the function `TrajectoryTracking()` inside the module `optcon.py`

Results show that starting from an arbitrary x_0 and after a transient phase,the system reaches and follows the optimal trajectory $(\mathbf{x}^{opt}, \mathbf{u}^{opt})$ computed in task 2. Modifying the cost matrices, which are the degree of freedom to control the performance of the system, is possible to reduce the peak of the input at the beginning but obtaining a bigger transient phase in the states and a larger cost. Alternatively the cost matrices can be modified in order to reduce the transient phase and reach in a smaller time the optimal reference. The following plots shows the tracked results for the system, starting from an x_0 perturbed respect the $x_{0_{opt}}$. The torque has a peak in first instant. This is caused by the wrong initial condition, disturbed by random normal noise.

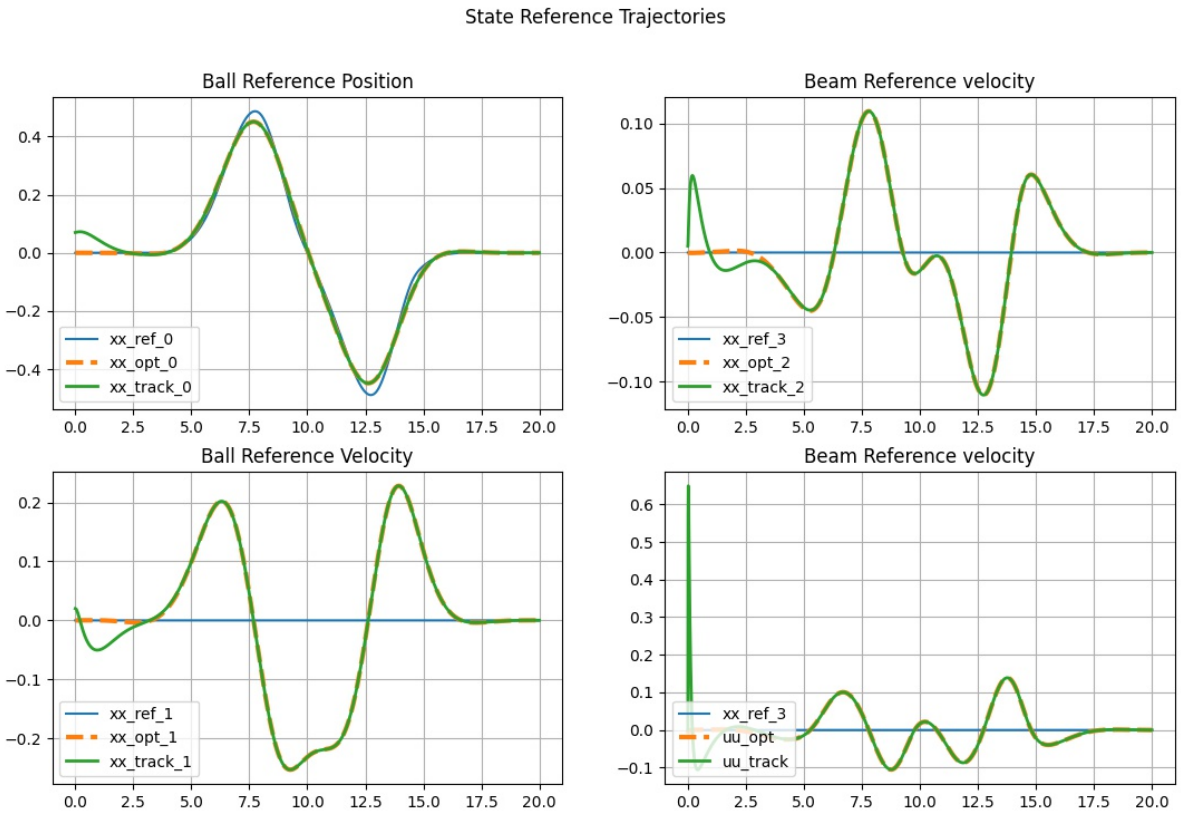

Figure 4.1: Comparison between the tracked state and the optimal state computed by the DDP algorithm, when the initial state is affected by noise

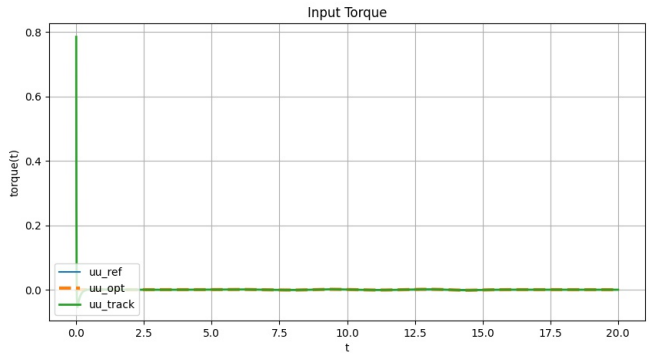

Figure 4.2: Comparison between the tracked input and the optimal input computed by the DDP algorithm, when the initial state is affected by noise

Case with noise affecting the dynamics

This plots shows that, even if the dynamics is affected by noise, the optimal feedback controller is able to track the optimal trajectory correctly. The input torque is disturbed by noise due to the fluctuation of the beam velocity. Anyway input torque is clearly following the optimal trajectory.

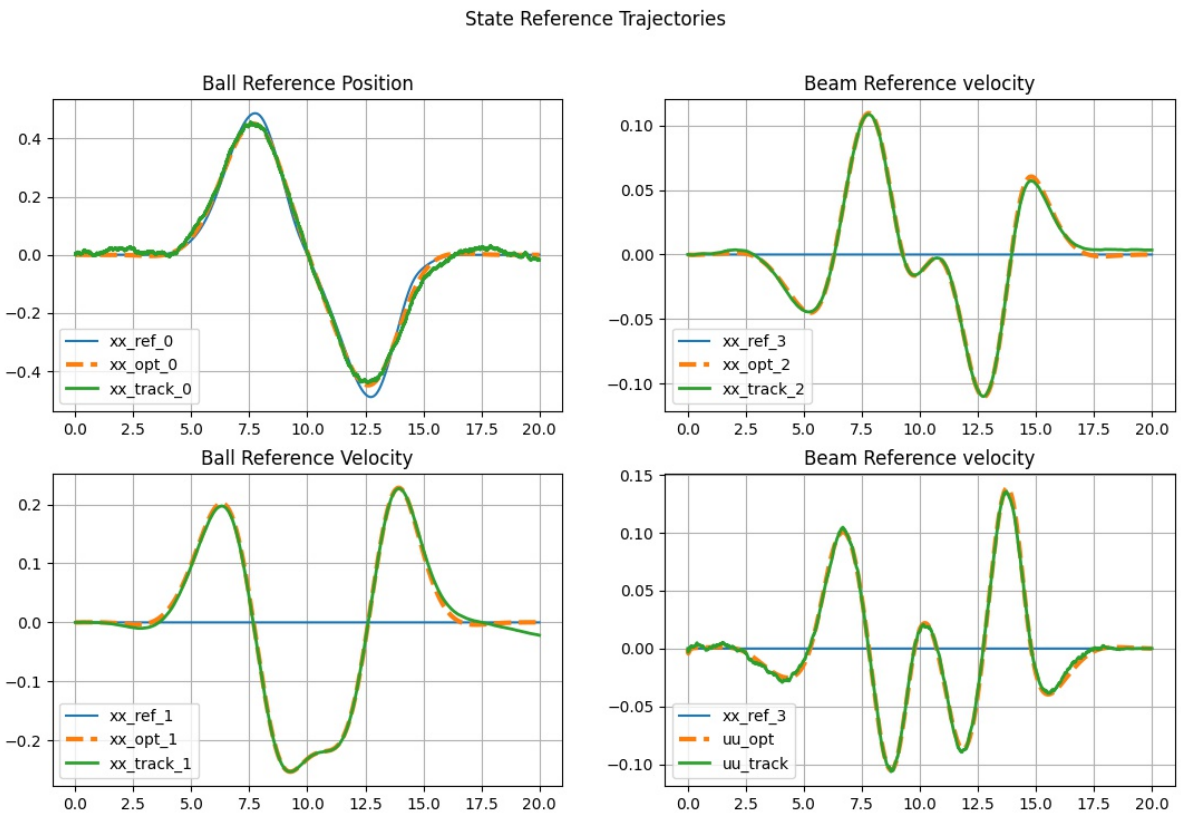

Figure 4.3: Comparison between the tracked state and the optimal state computed by the DDP algorithm, when the dynamics are affected by noise

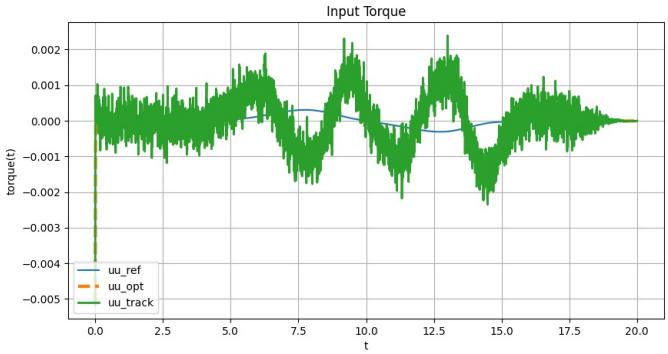

Figure 4.4: Comparison between the tracked input and the optimal input computed by the DDP algorithm, when the dynamics are affected by noise

Tracking over Refined Trajectory

In conclusion the LQR algorithm has been implemented over the reference optimal trajectory generated from the refined curve. As expected, the trajectory is tracked correctly. The beam position, velocity and the input, are slightly different from the optimal one since the performance requested give more importance to the position of the ball.

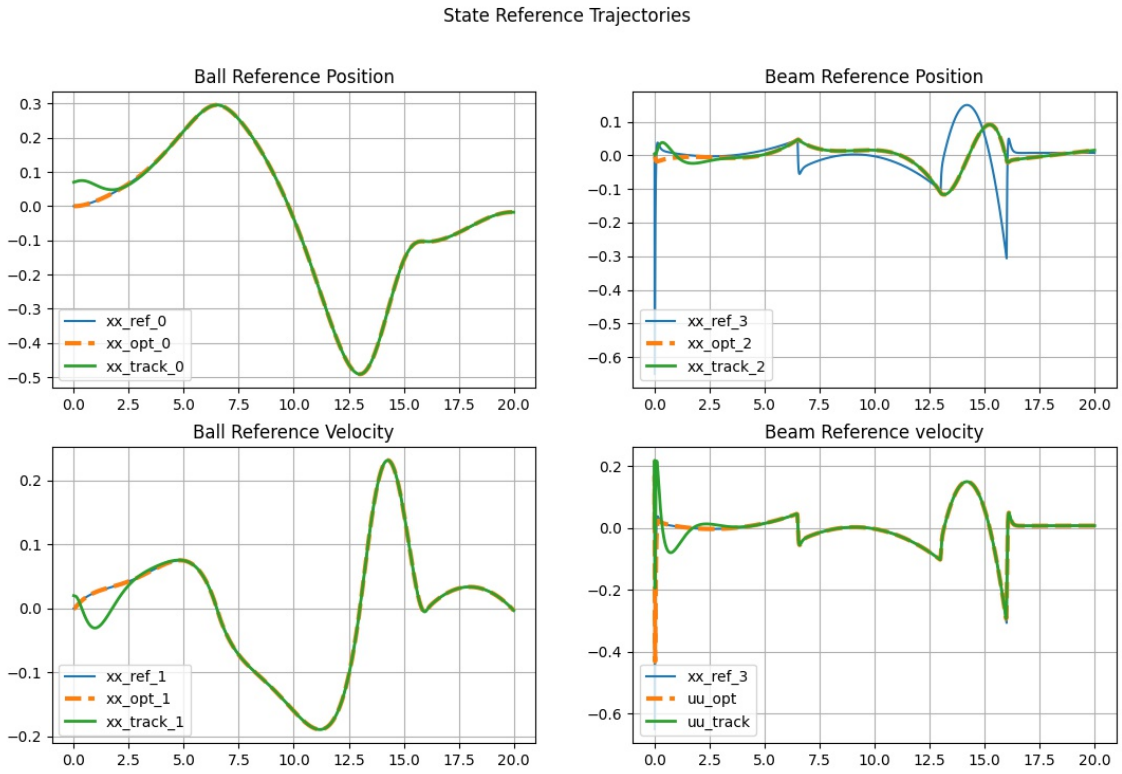

Figure 4.5: Comparison between the tracked state and the optimal state computed by the DDP algorithm, in case of refined reference with perturbed initial state

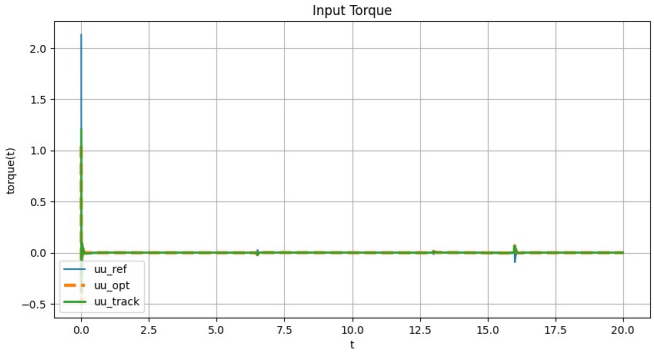

Figure 4.6: Comparison between the tracked input and the optimal input computed by the DDP algorithm, in case of refined reference with perturbed initial state

Conclusions

Using differential dynamic programming algorithm the objective of optimizing a trajectory for a non-linear system has been accomplished. In this paper are reported the results of the implementation of a DDP algorithm in python code. First, the dynamics have been discretized and the high order derivatives computed. Then the DDP algorithm has been applied firstly over a simple step function between two equilibria, then over a more complex and desired trajectory. This produced the desired optimal trajectory of the curve under inspection. In conclusion, by implementing an LQR algorithm the system tracked the optimal trajectory, with results that are not affected significantly by noise.

Bibliography

[1] D.P BERTSEKAS. *Dynamic programing and stochastic control*. Mathematics in Science and Engineering. Academic Press, 1976.

[2] G. BEAUCHAMP-BÃEZ C. G. BOLÃVAR-VINCENTY. *Modelling the Ball-and-Beam System*. URL: `http://www.laccei.org/LACCEI2014-Guayaquil/RefereedPapers/RP176.pdf`.

[3] G. NOTARSTEFANO. *Optimal control*. Slide for Optimal Control study course. 2021.