UNIVERSITÀ DI BOLOGNA

School of Engineering

Master Degree in Automation Engineering

Optimal Control and Reinforcement Learning

OPTIMAL CONTROL OF A GYMNAST ROBOT

Professor: **Giuseppe Notarstefano**

Students:

Rubén Gil Martínez

Guillermo López Pérez

Academic year 2025/2026

Abstract

This project focuses on the optimal control of a planar gymnast robot, modeled as a double pendulum with torque applied only at the hip (Acrobot). We implement trajectory generation using Newton-like algorithms and tracking via LQR and MPC techniques.

Contents

List of Figures

Introduction

The Acrobot, a planar gymnast robot, serves as a classic benchmark for underactuated mechanical systems. Comprising two links with torque applied solely at the hip joint, it presents a significant control challenge due to its highly non-linear dynamics and the lack of direct actuation at the first joint.

The primary objective of this project is to design and implement an optimal control framework capable of driving the system from its stable downward hanging state to the unstable upright equilibrium through a coordinated swing-up maneuver. This involves several integrated technical phases:

- **Task 0: Modeling and Discretization:** Derivation of the system's equations of motion and implementation of the 4th-order Runge-Kutta (RK4) scheme for high-fidelity numeric integration.
- **Task 1 & 2: Trajectory Generation:** Development of an Iterative Linear Quadratic Regulator (iLQR) algorithm. We explore the impact of reference choice by comparing a naive step reference with a physically-motivated smooth reference that leverages the system's natural frequency to minimize control effort.
- **Task 3: Feedback Tracking via LQR:** Implementation of a Time-Varying LQR controller designed to follow the optimal trajectory while rejecting disturbances and ensuring local stability.
- **Task 4: Predictive Control via MPC:** Implementation of Model Predictive Control (MPC) to enhance tracking performance and robustness by explicitly considering future system behavior.

By combining trajectory optimization with robust feedback control, we aim to achieve a seamless transitions between equilibria, demonstrating the efficacy of modern optimal control techniques in handling complex robotic dynamics.

Chapter 1

Task 0: Problem Setup and Discretization

1.1 System Description

The Acrobot is a generic planar gymnast robot consisting of two links and two joints. It is a canonical example of an underactuated mechanical system, where control input is only available at the second joint (the "hip"), while the first joint (the "shoulder") is passive.

The state of the system is defined by the generalized coordinates $q = [\theta_1, \theta_2]^\top$, where θ_1 is the angle of the first link with respect to the vertical axis, and θ_2 is the relative angle of the second link with respect to the first. The full state vector is given by $x = [\theta_1, \theta_2, \dot{\theta}_1, \dot{\theta}_2]^\top \in \mathbb{R}^4$. The control input $u = \tau \in \mathbb{R}$ represents the torque applied at the hip joint.

1.2 Equations of Motion

The continuous-time dynamics of the Acrobot are derived using the Euler-Lagrange formalism and can be expressed in the standard manipulator form:

$$M(q)\ddot{q} + C(q, \dot{q})\dot{q} + F\dot{q} + G(q) = Bu \tag{1.1}$$

where:

- $M(q) \in \mathbb{R}^{2\times 2}$ is the symmetric, positive-definite mass matrix.
- $C(q, \dot{q}) \in \mathbb{R}^{2\times 2}$ represents Coriolis and centrifugal forces.
- $F \in \mathbb{R}^{2\times 2}$ is the diagonal matrix of viscous friction coefficients.
- $G(q) \in \mathbb{R}^2$ is the gravity vector.
- $B = [0, 1]^\top$ is the actuation matrix.

For simulation and control design purposes, we solve for the joint accelerations $\ddot{q}$:

$$\ddot{q} = M(q)^{-1} \left(Bu - C(q, \dot{q})\dot{q} - F\dot{q} - G(q) \right) \tag{1.2}$$

1.3 Physical Parameters

The project employs Parameter Set 3, which defines the following physical constants:

Parameter	Symbol	Value
Mass of link 1	m_1	1.5 kg
Mass of link 2	m_2	1.5 kg
Length of link 1	l_1	2.0 m
Length of link 2	l_2	2.0 m
COM distance link 1	l_{c1}	1.0 m
COM distance link 2	l_{c2}	1.0 m
Inertia of link 1	I_1	2.0 kg·m^2
Inertia of link 2	I_2	2.0 kg·m^2
Gravity	g	9.81 m/s^2
Viscous friction	f_1, f_2	1.0 N·m·s/rad

Table 1.1: Acrobot Physical Parameters (Set 3)

1.4 Discretization via Runge-Kutta 4

For numerical simulation and discrete-time optimal control, the continuous-time dynamics $\dot{x} = f_c(x, u)$ must be discretized. We employ the 4th-order Runge-Kutta (RK4) integration scheme with a constant sampling period $d_t = 0.01$ s. Given the state at time t, the next state x_{t+1} is computed as:

$$k_1 = f_c(x_t, u_t) \tag{1.3a}$$

$$k_2 = f_c(x_t + \frac{d_t}{2} k_1, u_t) \tag{1.3b}$$

$$k_3 = f_c(x_t + \frac{d_t}{2} k_2, u_t) \tag{1.3c}$$

$$k_4 = f_c(x_t + d_t k_3, u_t) \tag{1.3d}$$

$$x_{t+1} = x_t + \frac{d_t}{6}(k_1 + 2k_2 + 2k_3 + k_4) \tag{1.3e}$$

This numeric integration method ensures robust stability and accuracy, facilitating the application of gradient-based optimization algorithms.

Chapter 2

Task 1 and 2: Trajectory Generation.

In this chapter, we address the core problem of trajectory generation: finding an optimal control sequence $u(t)$ and state trajectory $x(t)$ that transitions the Acrobot from a stable hanging position to the unstable upright equilibrium [3]. We employ a Newton-type optimal control algorithm (specifically, Differential Dynamic Programming making use of Iterative LQR method) to solve this non-linear optimization problem.

The process is divided into three logical steps: computing the boundary conditions (equilibria), defining a reference trajectory to guide the solver, and executing the iterative optimization loop.

2.1 Step 1: Computation of Equilibria

The first requirement is to rigorously define the start and end points of the maneuver. An equilibrium is defined as a state where the system remains stationary if no external force is applied (or a constant holding torque is maintained). Mathematically, this implies that the acceleration and velocity are zero:

$$\ddot{q} = 0, \quad \dot{q} = 0 \implies \dot{x} = 0 \tag{2.1}$$

We must solve the static equation derived from the dynamics $f(x, u)$:

$$f(x_{eq}, u_{eq}) = 0 \tag{2.2}$$

Given that the gravity vector $G(q)$ and other matrices contain non-linear trigonometric terms $(\sin(\theta_1), \sin(\theta_1+\theta_2), \sin(\theta_2), \cos(\theta_1), \cos(\theta_2))$, an analytical solution is non-trivial. Therefore, we employ a numerical root-finding algorithm to solve for the configuration angles.

We identified two key equilibria:

- **Stable Equilibrium (x_{eq1}):** The "Down" position where the robot hangs vertically under gravity ($x \approx [0, 0, 0, 0]^\top$).
- **Unstable Equilibrium (x_{eq2}):** The "Up" position where the robot is balanced perfectly inverted ($x \approx [\pi, 0, 0, 0]^\top$).

2.2 Step 2: Definition of Reference Curves

The optimization algorithm requires a reference trajectory $x_{ref}(t)$ to define the cost function. We explored two different approaches to generating this reference, highlighting the impact of physical consistency on solver convergence.

2.2.1 Task 1: The Step Reference

In the first approach, we define a "Step" reference. The time horizon T is split into two halves:

$$x_{ref}(t) = \begin{cases} x_{eq1} & \text{if } 0 \leq t < T/2 \\ x_{eq2} & \text{if } T/2 \leq t \leq T \end{cases} \tag{2.3}$$

This reference commands the robot to stay at the bottom for the first half and immediately teleport to the top for the second half.

Limitations: This creates a conflict between the *optimization objective* and the *system physics*. To reach the top, an underactuated robot must swing back and forth ("pump" energy). However, the cost function J penalizes any deviation from zero during the first half. The solver is forced to fight this penalty to discover the necessary swinging motion, often resulting in slower convergence.

2.2.2 Task 2: The "Natural" Smooth Reference

In the second approach, we construct a "Physically Inspired" reference that anticipates the system's dynamics. This reference is composed of two phases:

1. **Phase 1 (Energy Pumping):** We inject sinusoidal oscillations into the reference trajectory. The frequency of these oscillations is tuned to the natural frequency of a single-arm pendulum, approximated by the simplified model $\omega_n \approx \sqrt{g/L_{eff}}$. Where g is the acceleration due to gravity ($g \approx 9.81$ m/s^2) and L_{eff} is the effective length of the first part of the pendulum ($L_{eff} \approx L_1$).
2. **Phase 2 (Smooth Rise):** We utilize a sigmoid function (logistic curve) to transition smoothly from the oscillating bottom state to the inverted equilibrium, avoiding the infinite derivatives associated with a step change.

Advantages for Convergence: By adding oscillations, we align the cost function with the physics. We effectively signal to the solver: *"It is optimal to swing here."* This reduces the conflict between the objective function and the dynamic constraints, creating a smoother optimization landscape and facilitating faster, more robust convergence.

2.3 Step 3: The Optimization Loop (Newton via iLQR)

With the boundary conditions and reference defined, we implement the Newton-type optimal control algorithm. In every iteration k, we locally approximate the problem by solving a finite-horizon **LQR Affine** problem to find the optimal control deviation, this sequence of solved subproblems forms the Newton-type update. [1].

The process consists of three distinct phases. Below, we detail the mathematical formulation and provide a direct mapping to the variables used in our Python implementation.

2.3.1 Phase 1: Forward Pass (Nominal Trajectory)

We simulate the non-linear dynamics using the current control sequence $\bar{u} = 0$ and $\bar{x_0} = x_{eq1}$, here, we obtain this first feasible trajectory via **Shooting**.

$$\bar{x}_{t+1} = f(\bar{x}_t, \bar{u}_t) \tag{2.4}$$

Code Mapping:

- $\bar{x}_t, \bar{u}_t \rightarrow$ `x_traj[i]`, `u[i]`
- $f(\cdot) \rightarrow$ `discrete_step_rk4(x, u, dt)`

2.3.2 Phase 2: Backward Pass (Solving LQR Affine)

This phase computes the feedback gains and feedforward corrections by propagating the computation of the Riccati difference equations backwards in time.

1. Initialization ($t = T$)

We initialize the Cost-to-Go vector and matrix.

$$P_T = Q_T, \quad p_T = Q_T(x_T - x_{ref,T}) \tag{2.5}$$

Code Mapping:

- $P_T \rightarrow$ `P = self.Q_final`
- $p_T \rightarrow$ `p = self.Q_final @ dx_terminal`

2. Linearization and Approximation

For each step t, we compute the local derivatives.

- Dynamics (A_t, B_t): `A_t`, `B_t` (via `get_derivatives_fd`).
- Taylor Linear Terms (q_t, r_t): `q_t`, `r_t` (via `Q @ dx`, `R @ du`).
- Taylor Quadratic Terms (Cost Hessians) (Q_t, R_t): `self.Q`, `self.R`.

3. The "S" Matrix (auxiliary term)

To simplify the gain computation, we define the matrix S_t, which represents the Hessian of the Action-Value function with respect to the control input u.

$$S_t = R_t + B_t^\top P_{t+1} B_t \tag{2.6}$$

Code Mapping:

- $S_t \rightarrow$ `S_mat = self.R + B_t.T @ P @ B_t`
- $S_t^{-1} \rightarrow$ `S_inv` (Regularized inverse)

4. Computing the Gains

We calculate the optimal **Feedback Gain** K_t^* and **Feedforward Correction** σ_t^* by minimizing the local quadratic model [1]:

$$K_t^* = -S_t^{-1}(B_t^\top P_{t+1} A_t) \tag{2.7}$$

$$\sigma_t^* = -S_t^{-1}(r_t + B_t^\top p_{t+1}) \tag{2.8}$$

Code Mapping:

- $K_t^* \rightarrow$ `K_t` (stored in `K_gains`)
- $\sigma_t^* \rightarrow$ `sigma_t` (stored in `sigma_vec`)

5. Difference Riccati Equation Update

Finally, we update the Cost-to-Go parameters (P_t, p_t) for the previous time step. In the code, we utilize the computed gains to perform this update efficiently. We have to keep in mind that c_t, which is a offset term for the linearization, can be set to zero becuase we define the problem in terms of deviations from the nominal trajectory, thus, the variables that the optimal control problem depends on are $x_t - \bar{x}_t$ and $u_t - \bar{u}_t$.

$$P_t = Q_t + A_t^\top P_{t+1} A_t - K_t^{*\top} S_t K_t^* \tag{2.9}$$

$$p_t = q_t + A_t^\top p_{t+1} - K_t^{*\top} S_t \sigma_t^* \tag{2.10}$$

$$c_t = 0 \tag{2.11}$$

Code Mapping:

- $K_t^{*\top} S_t K_t^* \rightarrow$ `term_quad` (The quadratic reduction)
- $K_t^{*\top} S_t \sigma_t^* \rightarrow$ `term_lin` (The linear reduction)
- $P_t \rightarrow$ `P` (updated for next loop iteration)
- $p_t \rightarrow$ `p` (updated for next loop iteration)

2.3.3 Phase 3: Update (Closed-Loop Rollout)

We generate the new control sequence. Crucially, we employ a **Closed-Loop** update rule. This means we apply the feedforward correction σ_t^* scaled by a step size α, plus the feedback reaction to the *actual* state deviation experienced during the new simulation.

$$u_{new}(t) = \bar{u}_t + \alpha \cdot \sigma_t^* + K_t^* \cdot (x_{new}(t) - \bar{x}_t) \tag{2.12}$$

Code Mapping:

- $\alpha \rightarrow$ `alpha` (Determined via Armijo Line Search)
- $x_{new}(t) - \bar{x}_t \rightarrow$ `dx_deviation`
- Update $\rightarrow$ `u_new[i] = u[i] + du`

The feedback term K_t^* stabilizes the unstable Acrobot dynamics during the rollout, ensuring that the new trajectory remains valid even when moving far from the equilibrium.

Chapter 3

Task 3: Trajectory Tracking via LQR

3.1 Linearization around the Optimal Generated Trajectory

The optimal trajectory (x_t^*, u_t^*) generated in the previous chapter serves as the nominal path [3]. To handle disturbances and model uncertainties, we linearize the discrete-time dynamics $x_{t+1} = f_d(x_t, u_t)$ by performing a first-order Taylor expansion about this optimal nominal trajectory. This produces a time-varying linear approximation of the system that is locally valid for small deviations from the reference trajectory:

$$\Delta x_{t+1} \approx A_t^{\text{traj}} \Delta x_t + B_t^{\text{traj}} \Delta u_t \quad t = 0, \ldots, T-1 \tag{3.1}$$

where $\Delta x_t = x_t - x_t^*$ and $\Delta u_t = u_t - u_t^*$. In this formulation, the Jacobian matrices A_t^{traj} and B_t^{traj} represent the system's sensitivity to small perturbations from the planned path, and are defined as:

$$A_t^{\text{traj}} = \frac{\partial f_d}{\partial x}\Big|_{x_t^*, u_t^*}, \quad B_t^{\text{traj}} = \frac{\partial f_d}{\partial u}\Big|_{x_t^*, u_t^*} \tag{3.2}$$

By updating these matrices at every time step t, the controller accounts for the changing geometry and velocity of the Acrobot, effectively "straightening" the nonlinear physics into a sequence of linear sub-problems that can be solved efficiently.

3.2 Discrete-Time Finite-Horizon TV-LQR Optimization

The core of the tracking problem is formulated as an optimization task over a finite time horizon T. Given the nominal trajectory (x_t^*, u_t^*), we define the

tracking error states $\Delta x_t = x_t - x_t^*$ and control deviations $\Delta u_t = u_t - u_t^*$. The goal is to find an optimal sequence of control deviations that minimizes a quadratic cost function:

$$\sum_{t=0}^{T-1} (\Delta x_t^\top Q_{\text{reg}} \Delta x_t + \Delta u_t^\top R_{\text{reg}} \Delta u_t) + \Delta x_T^\top Q_{\text{final}} \Delta x_T \tag{3.3}$$

subject to: $\Delta x_{t+1} = A_t^{\text{traj}} \Delta x_t + B_t^{\text{traj}} \Delta u_t$ t=0,...,T-1 $\Delta x_0 = 0$

The solution to this problem is obtained via dynamic programming, specifically by solving the discrete-time Riccati difference equations backwards in time. This process yields a sequence of optimal cost-to-go matrices P_t, which are used to compute the feedback gains.

3.3 Implementation of the Time-Varying Feedback Controller

A crucial aspect of our implementation is the separation between the optimization phase (computing gains) and the control phase (applying the feedback law).

3.3.1 Phase 1: Computation of Optimal Feedback Gains

Before starting the tracking experiment, we perform a backward pass from $t = T$ down to $t = 0$. In this phase, we pre-compute and store the optimal feedback gains K_t for every time step:

$$P_T = Q_{\text{final}} \tag{3.4a}$$

$$K_t^{\text{reg}} = -(R_{\text{reg}} + B_t^{\text{traj, T}} P_{t+1} B_t^{\text{traj}})^{-1} (B_t^{\text{traj, T}} P_{t+1} A_t^{\text{traj}}) \tag{3.4b}$$

$$P_t = Q_{\text{reg}} + A_t^{\text{traj},T} P_{t+1} A_t^{\text{traj}} + (A_t^{\text{traj},T} P_{t+1} B_t^{\text{traj}}) K_t^{\text{reg}} \tag{3.4c}$$

This pre-computation ensures that the controller can operate in real-time during the forward simulation, as the heavy matrix operations are already solved.

3.3.2 Phase 2: State Feedback Control Law

Once the gains K_t are available, the physical system is controlled using a feedback controller. At each sampling instant t, the controller measures the actual state x_t, computes the error Δx_t, and applies the following control command:

$$u_t = u_t^{traj} + K_t^{reg} (x_t - x_t^{traj}) \tag{3.5}$$

$$x_{t+1} = f_d(x_t, u_t), \quad t = 0, \ldots, T-1 \tag{3.6}$$

Here, u_t^{traj} acts as the *feedforward* nominal torque required to maintain the ideal trajectory, while the term $K_t \Delta x_t$ provides the *feedback* corrective action, after applying the corrective action, we compute the new state x_{t+1} which will be more close to the desired trajectory. This dual structure allows the Acrobot to stay close to the optimized swing-up path even when faced with initial condition perturbations or unmodeled dynamics, leveraging the local optimality of the LQR design.

Chapter 4

Task 4: Trajectory Tracking via MPC

While Task 3 focused on a pre-computed, offline feedback law (LQR), Task 4 introduces **Model Predictive Control (MPC)** for trajectory tracking [2]. The fundamental difference lies in the fact that MPC does not rely on a fixed sequence of gains. Instead, at every single time step t, it solves an **online optimization problem** to find the best possible control action. This receding-horizon approach makes MPC more robust and flexible than a standard LQR tracker, particularly for large deviations from the nominal trajectory.

4.1 The Receding Horizon Strategy

The implemented MPC controller employs a **Linear Time-Varying (LTV)** formulation with a receding horizon strategy. At each time step t, the controller plans over a finite prediction horizon of T steps into the future, but only executes the very first control command. This process is then repeated at the next sampling instant, incorporating new state information.

This principle allows the controller to continuously integrate new measurements at every sampling instance, providing a natural feedback mechanism that compensates for errors introduced by the first-order Taylor linearization of the Acrobot's nonlinear dynamics.

4.2 MPC Problem Formulation

At each time step t, given the measured state x_t^{meas}, the MPC controller solves the following finite-horizon optimal control problem:

$$\min_{\Delta x, \Delta u} \sum_{k=0}^{T-1} \left(\Delta x_{k|t}^\top Q_{\text{mpc}} \Delta x_{k|t} + \Delta u_{k|t}^\top R_{\text{mpc}} \Delta u_{k|t} \right) + \Delta x_{T|t}^\top P_{\text{mpc}} \Delta x_{T|t} \quad (4.1)$$

subject to the linearized dynamics constraints:

$$\Delta x_{k+1|t} = A_{t+k} \Delta x_{k|t} + B_{t+k} \Delta u_{k|t}, \quad k = 0, \ldots, T-1 \quad (4.2)$$

and the initial condition constraint:

$$\Delta x_{0|t} = x_t^{\text{meas}} - x_t^* \quad (4.3)$$

Here, $\Delta x_{k|t}$ and $\Delta u_{k|t}$ represent the predicted state and control deviations from the reference trajectory at time $t+k$, as predicted at time t. The notation $k|t$ emphasizes the predictive nature of MPC.

4.3 Terminal Cost Design for Stability

A critical component of the MPC formulation is the terminal cost matrix P_{mpc}. To ensure closed-loop stability, this matrix is obtained by solving the **Discrete Algebraic Riccati Equation (DARE)** at the target equilibrium x_{eq2}:

$$P_{\text{mpc}} = A^\top P_\infty A - A^\top P_\infty B (R + B^\top P_\infty B)^{-1} B^\top P_\infty A + Q \quad (4.4)$$

The resulting matrix $P_{\text{mpc}} = P_\infty$ represents an infinite-horizon cost-to-go at the terminal state. From a theoretical perspective, this terminal cost acts as a Lyapunov function, providing guarantees of closed-loop stability for the MPC controller when combined with the finite horizon.

4.4 Local Linearization within the Horizon

Since the Acrobot is a highly nonlinear system, the MPC controller performs local linearization at each step of the prediction horizon. The Jacobian matrices A_{t+k} and B_{t+k} are computed using finite differences about the reference trajectory points (x_{t+k}^*, u_{t+k}^*):

$$A_{t+k} \approx \frac{\partial f_d}{\partial x}\Big|_{(x_{t+k}^*, u_{t+k}^*)}, \quad B_{t+k} \approx \frac{\partial f_d}{\partial u}\Big|_{(x_{t+k}^*, u_{t+k}^*)} \quad (4.5)$$

This time-varying linearization allows the MPC to capture the local physics of the system across the entire prediction horizon, adapting to the changing dynamics along the swing-up trajectory.

4.5 Online Optimization via Quadratic Programming

The optimization problem is structured as a **Quadratic Program (QP)** and solved at each time step using an efficient solver (OSQP). The QP structure consists of:

- **Hessian Matrix** H: A large, sparse, block-diagonal matrix that aggregates all stage costs (Q_{mpc}, R_{mpc}) and the terminal cost (P_{mpc}) over the prediction horizon.
- **Equality Constraints**: The linearized dynamics $\Delta x_{k+1|t} = A_{t+k}\Delta x_{k|t} + B_{t+k}\Delta u_{k|t}$ are enforced as linear equality constraints, ensuring that the predicted trajectory remains dynamically consistent.
- **Initial Condition**: The optimization is constrained to start at the current measured deviation: $\Delta x_{0|t} = x_t^{\text{meas}} - x_{t,ref}$.

The QP solver returns the optimal sequence of control corrections $\{\Delta u_{t|t}^*, \ldots, \Delta u_{t+T-1|t}^*\}$. Following the receding horizon principle, only the first correction $\Delta u_{t|t}^*$ is applied.

4.6 Closed-Loop Simulation

The MPC controller is tested in closed-loop simulation to evaluate its robustness against perturbations. At each time step t, the control loop executes the following steps:

1. **Measurement**: Obtain the current simulated state x_t^{meas}.
2. **Optimization**: Solve the MPC QP problem to compute $\Delta u_{t|t}^*$.
3. **Actuation**: Apply the control input to the full nonlinear model:

$$u_{\text{applied}} = u_t^* + \Delta u_{t|t}^* \tag{4.6}$$

4. **Integration**: Compute the next state using the nonlinear dynamics:

$$x_{t+1} = f_d(x_t^{\text{meas}}, u_{\text{applied}}) \tag{4.7}$$

By continuously re-optimizing and incorporating feedback, the MPC controller successfully rejects large perturbations and ensures swing-up and stabilization of the Acrobot at the upright equilibrium.

Chapter 5

Task 5: Animation and Results

In this chapter, we present the numerical results for all tasks, including trajectory generation performance and tracking effectiveness under various conditions.

5.1 Task 1: Trajectory Generation (Step Reference)

Performance metrics for the iLQR algorithm using a simple step reference.

5.1.1 Trajectories and Convergence Metrics

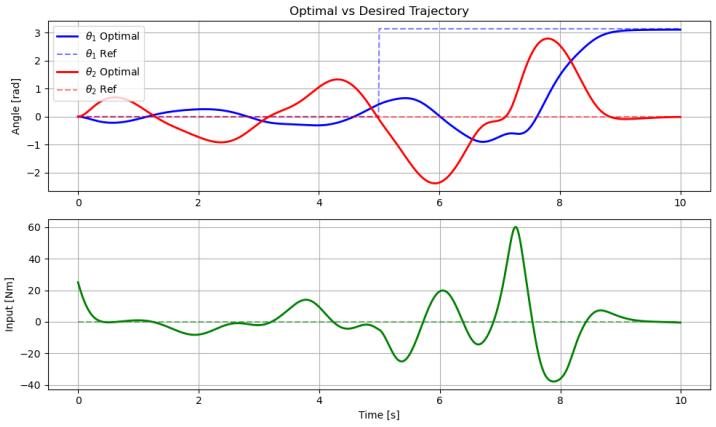


Figure 5.1: Task 1: Optimal vs Desired Trajectories.

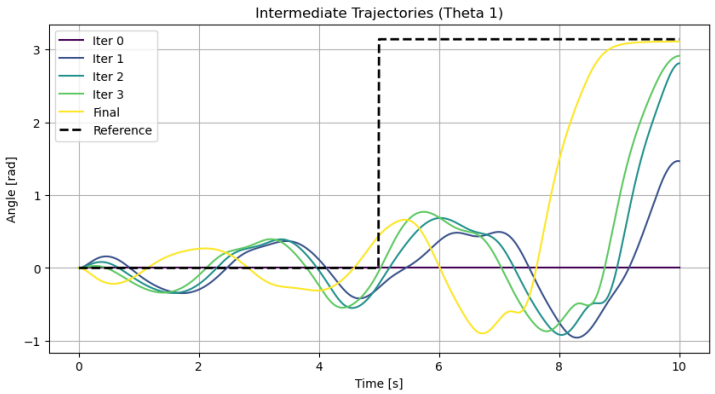


Figure 5.2: Task 1: Intermediate Trajectories during optimization.

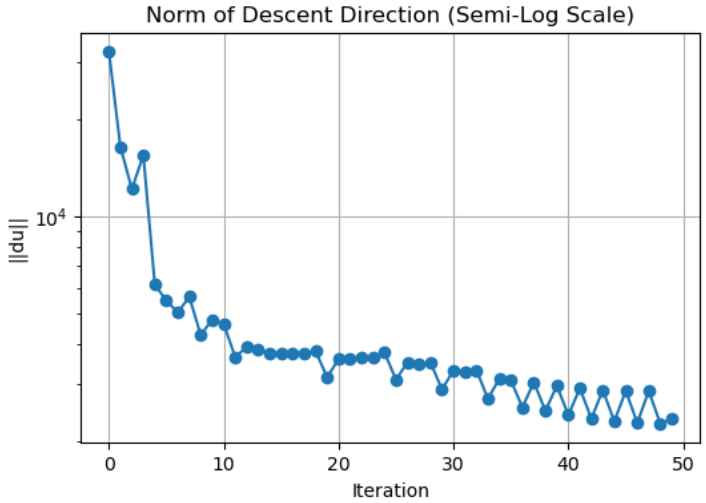


Figure 5.3: Task 1: Norm of Descent Direction (Semi-Logarithmic Scale).

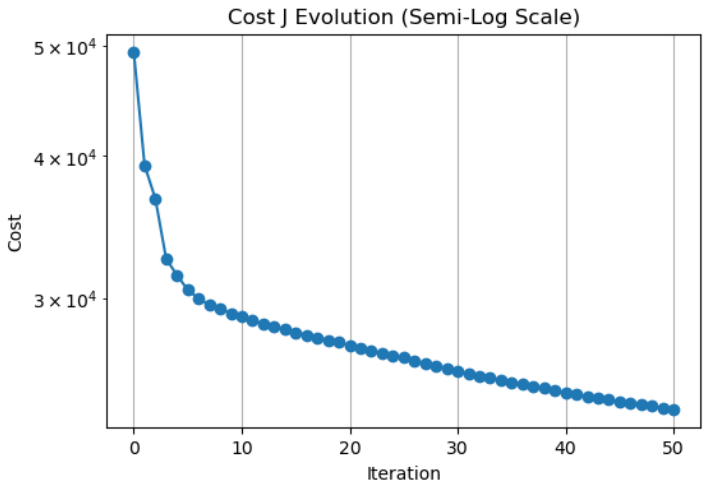


Figure 5.4: Task 1: Cost Evolution (Semi-Logarithmic Scale).

5.1.2 Armijo Landscapes Across Iterations

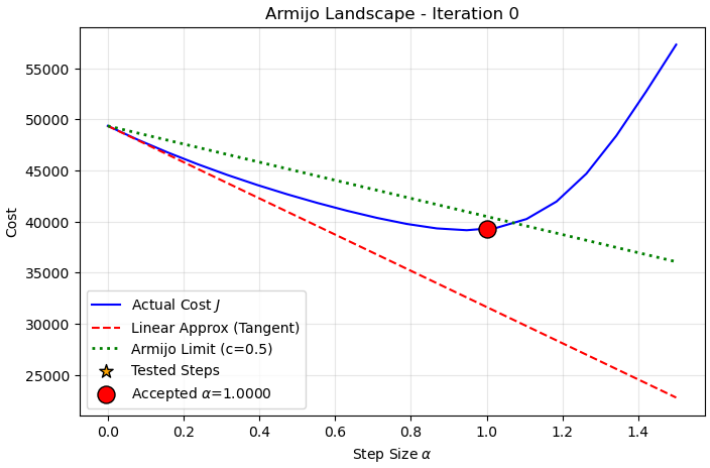


Figure 5.5: Task 1: Armijo Landscape (Iteration 0).

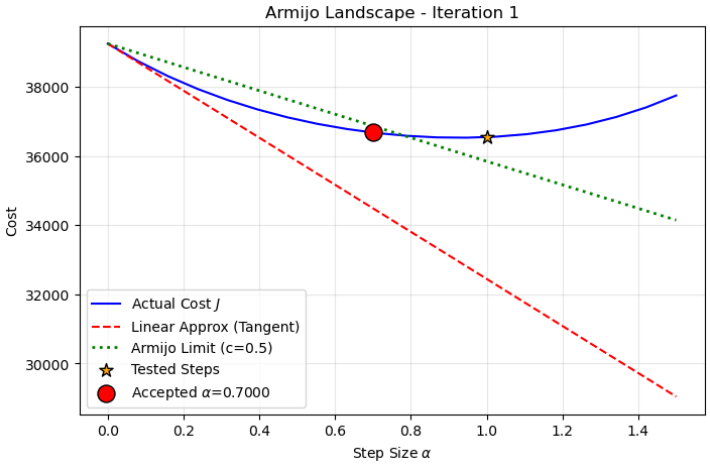


Figure 5.6: Task 1: Armijo Landscape (Iteration 1).

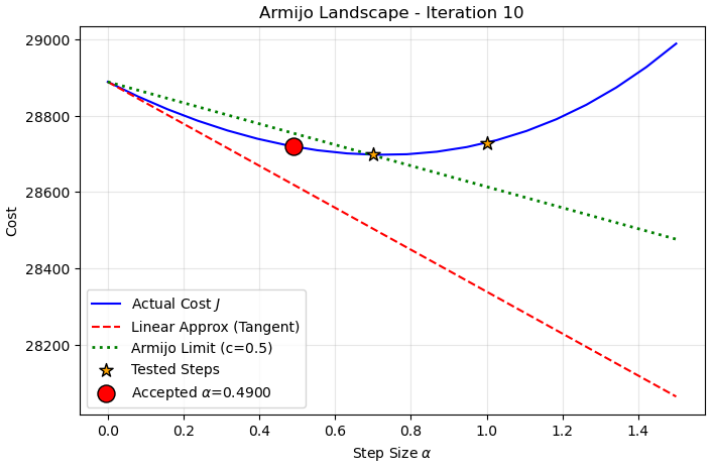


Figure 5.7: Task 1: Armijo Landscape (Iteration 10).

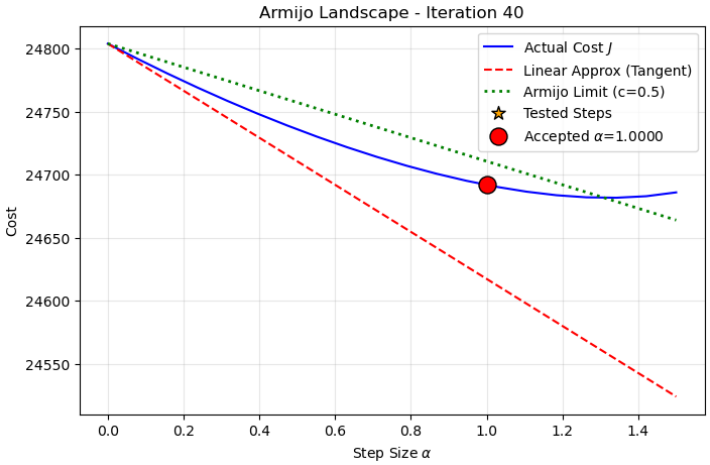


Figure 5.8: Task 1: Armijo Landscape (Iteration 40).

5.2 Task 2: Trajectory Generation (Smooth Reference)

Comparison of the iLQR performance when initiated with a physically-inspired smooth reference.

5.2.1 Trajectories and Convergence Metrics

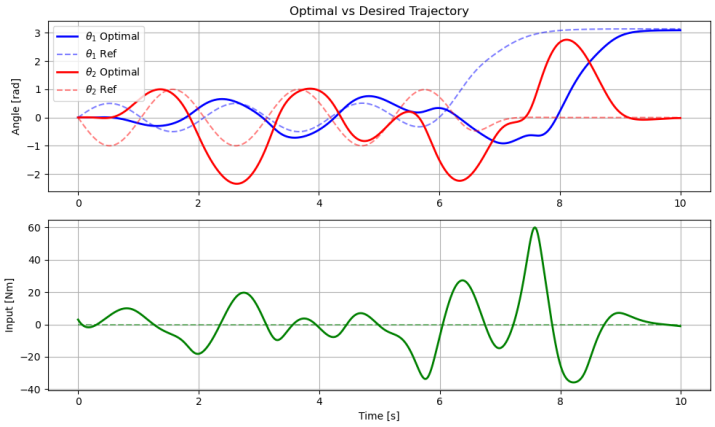


Figure 5.9: Task 2: Optimal vs Desired Trajectories.

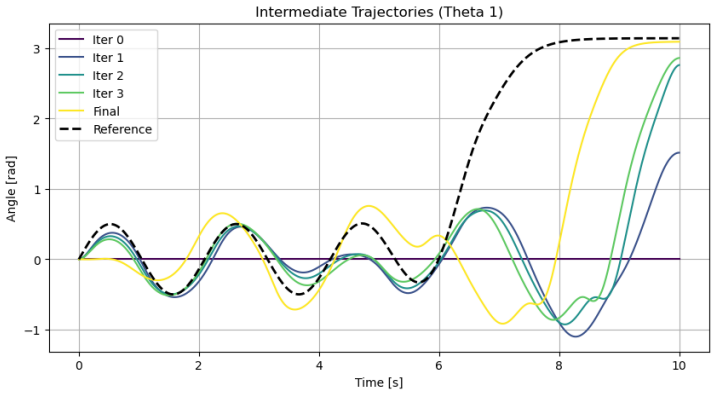


Figure 5.10: Task 2: Intermediate Trajectories during optimization.

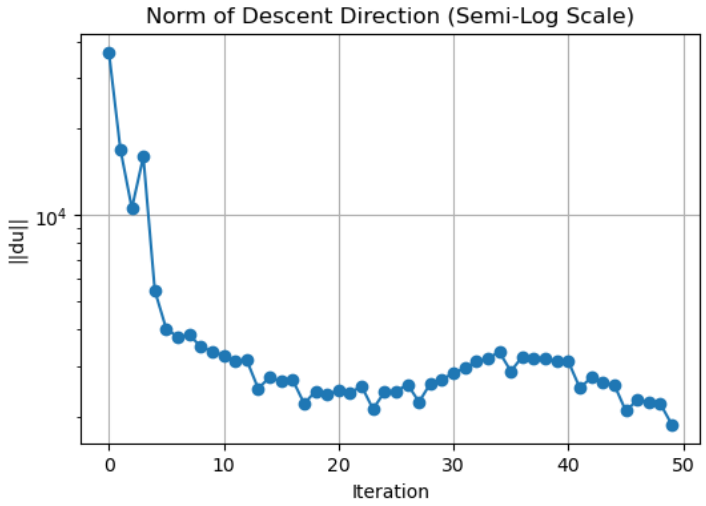


Figure 5.11: Task 2: Norm of Descent Direction (Semi-Logarithmic Scale).

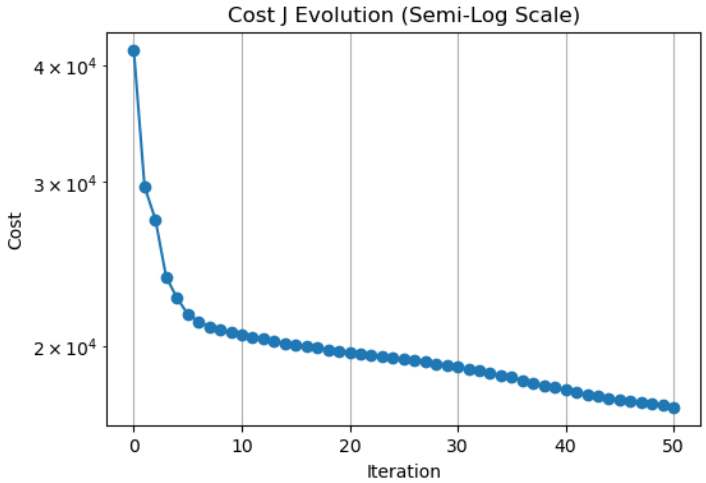


Figure 5.12: Task 2: Cost Evolution (Semi-Logarithmic Scale).

5.2.2 Armijo Landscapes Across Iterations

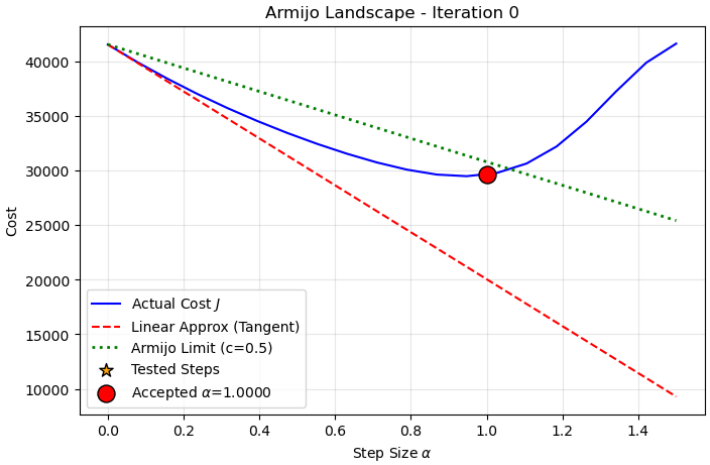


Figure 5.13: Task 2: Armijo Landscape (Iteration 0).

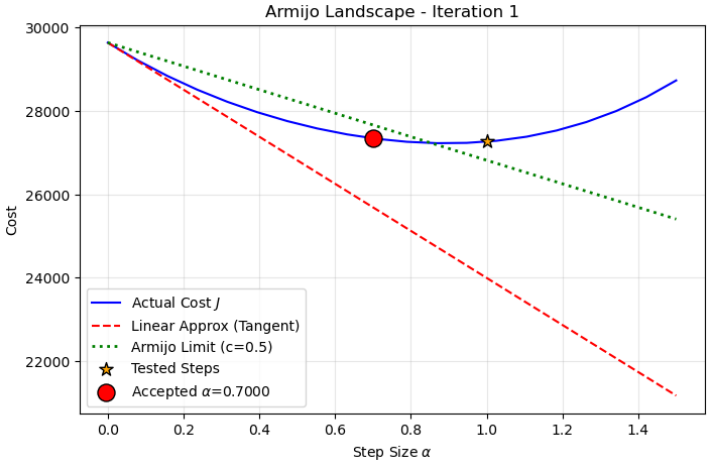


Figure 5.14: Task 2: Armijo Landscape (Iteration 1).

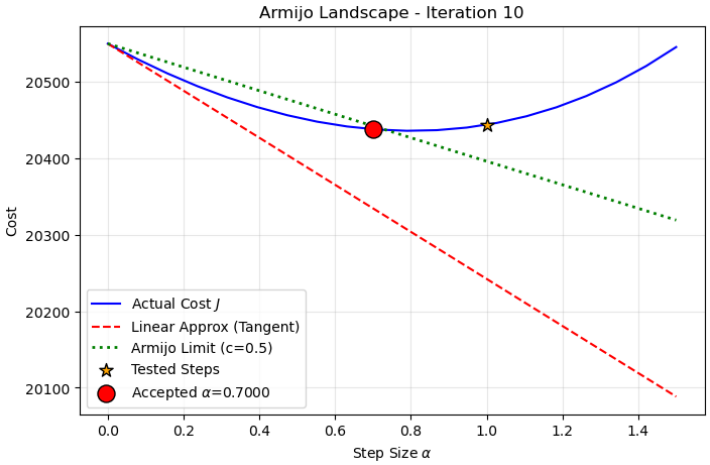


Figure 5.15: Task 2: Armijo Landscape (Iteration 10).

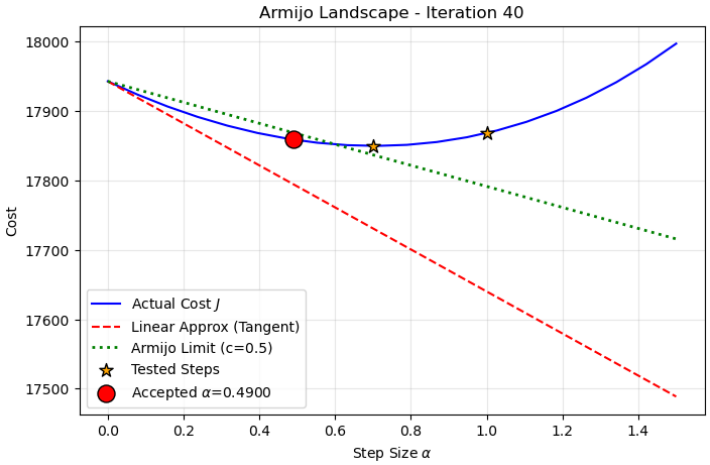


Figure 5.16: Task 2: Armijo Landscape (Iteration 40).

5.3 Task 3: LQR Tracking Performance

Tracking capability of the Time-Varying LQR controller under different initial condition perturbations.

5.3.1 System Trajectory vs Desired Optimal Trajectory

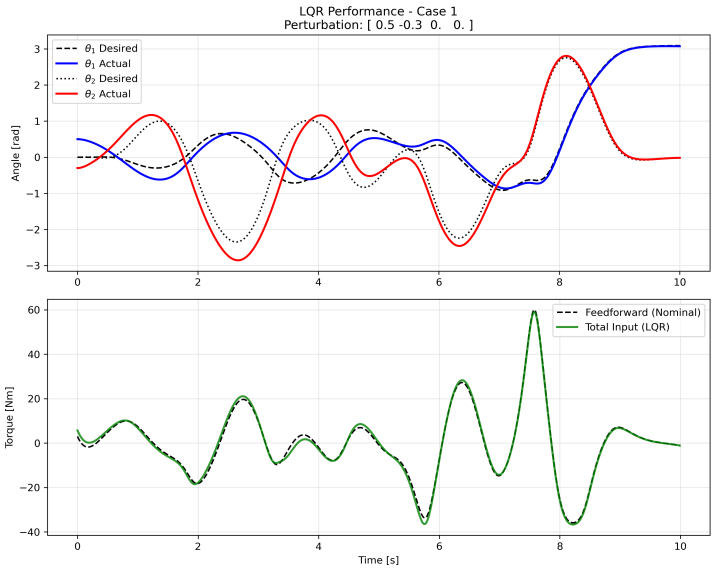


Figure 5.17: Task 3: System Trajectory vs Optimal Trajectory (Perturbation 1).

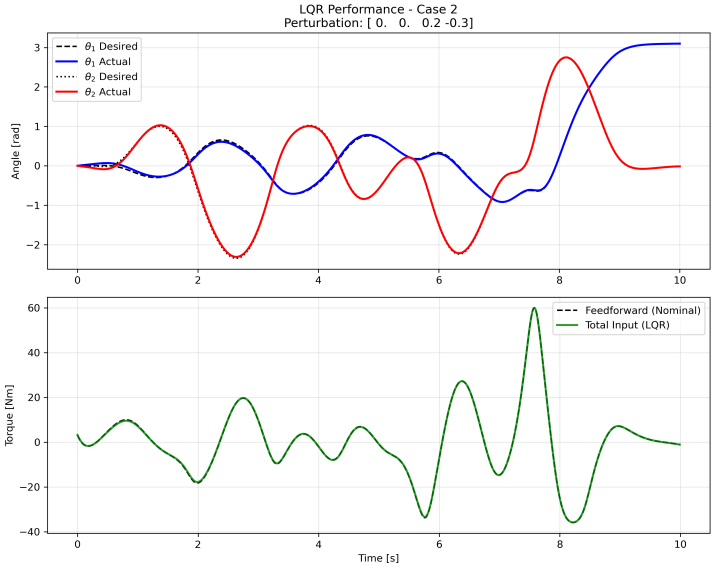


Figure 5.18: Task 3: System Trajectory vs Optimal Trajectory (Perturbation 2).

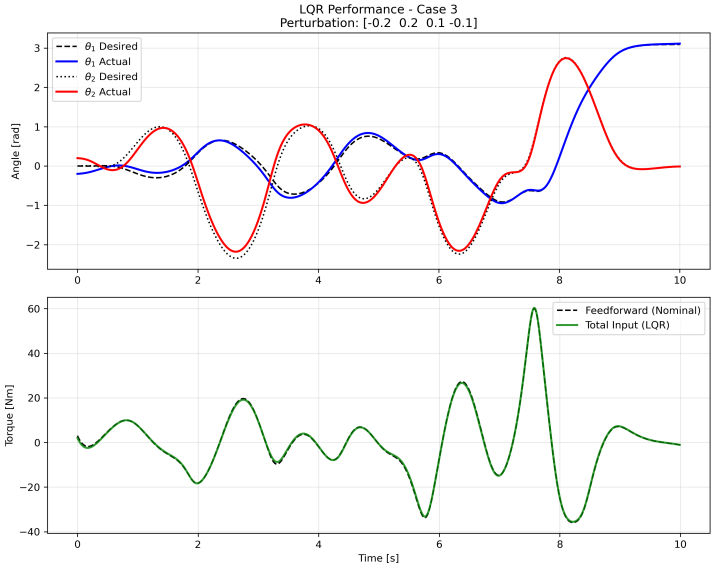


Figure 5.19: Task 3: System Trajectory vs Optimal Trajectory (Perturbation 3).

5.3.2 Tracking Error

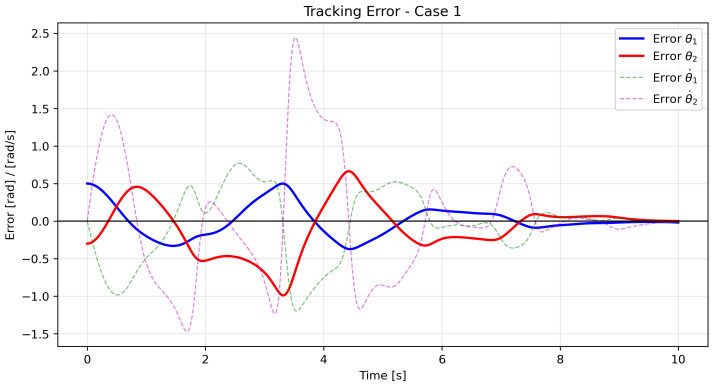


Figure 5.20: Task 3: Tracking Error (Perturbation 1).

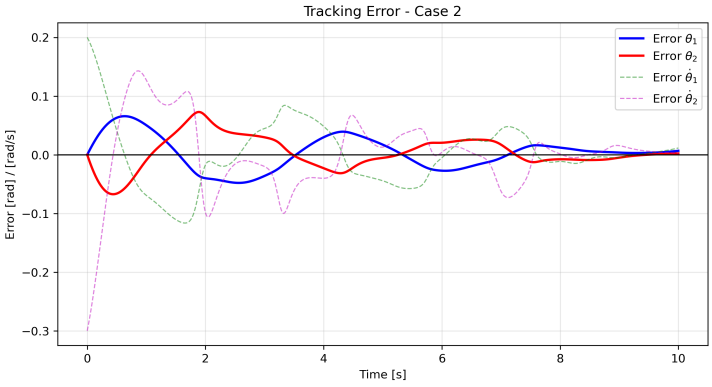


Figure 5.21: Task 3: Tracking Error (Perturbation 2).

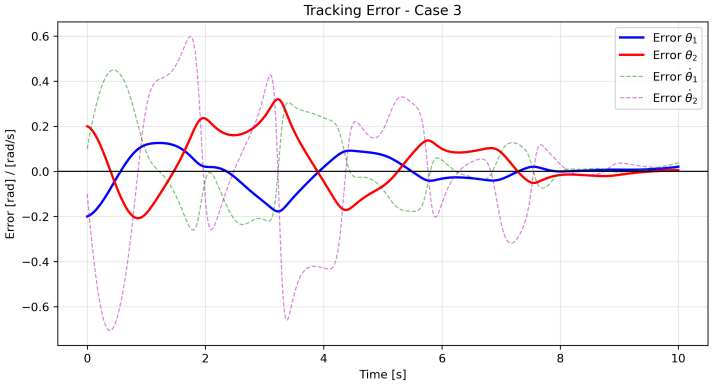


Figure 5.22: Task 3: Tracking Error (Perturbation 3).

5.4 Task 4: MPC Tracking Performance

Tracking capability of the Model Predictive Controller under different initial condition perturbations.

5.4.1 System Trajectory vs Desired Optimal Trajectory

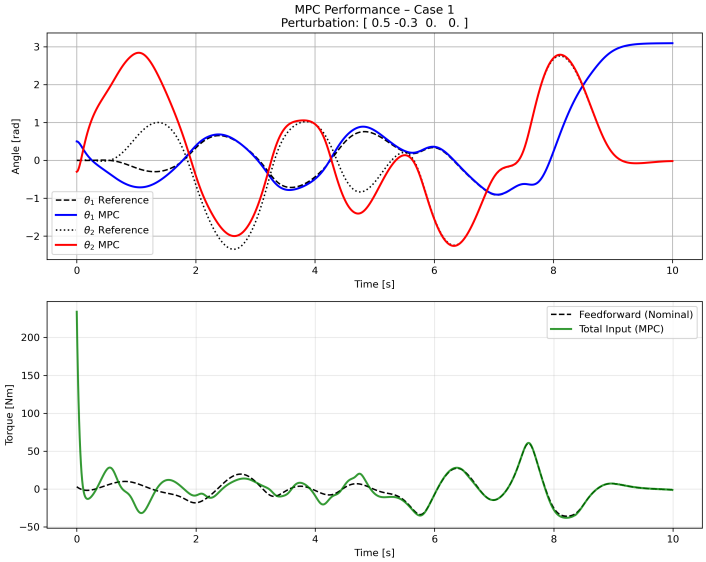


Figure 5.23: Task 4: System Trajectory vs Optimal Trajectory (Perturbation 1).

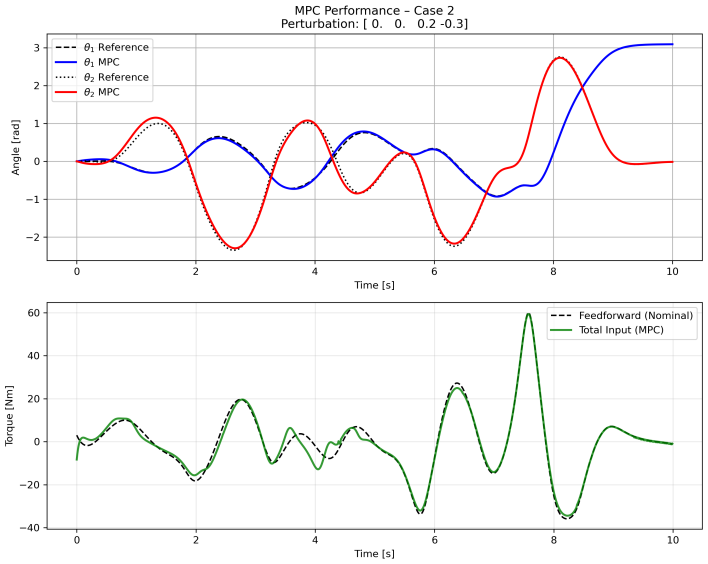


Figure 5.24: Task 4: System Trajectory vs Optimal Trajectory (Perturbation 2).

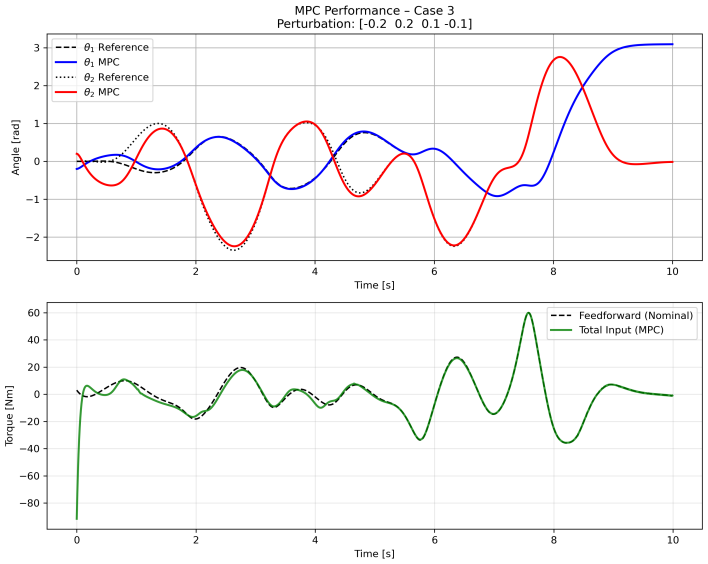


Figure 5.25: Task 4: System Trajectory vs Optimal Trajectory (Perturbation 3).

5.4.2 Tracking Error

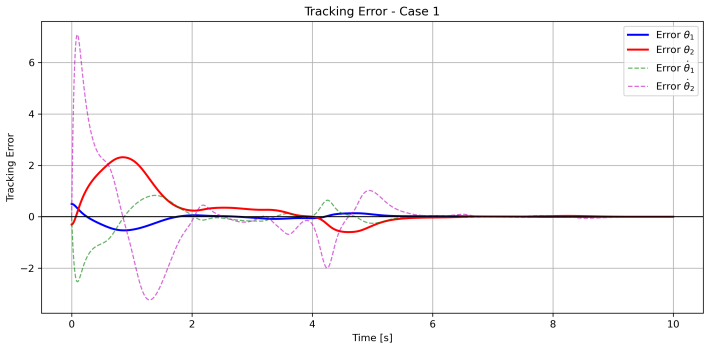


Figure 5.26: Task 4: Tracking Error (Perturbation 1).

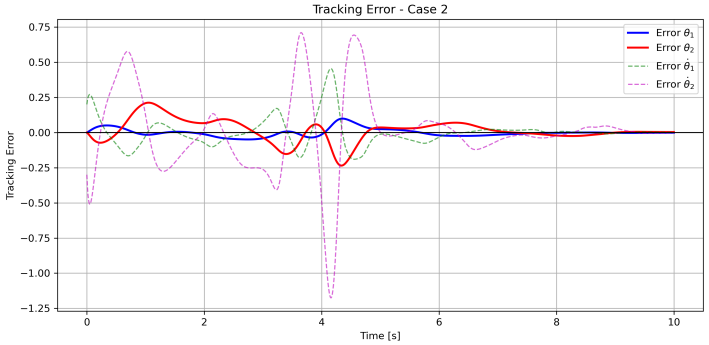


Figure 5.27: Task 4: Tracking Error (Perturbation 2).

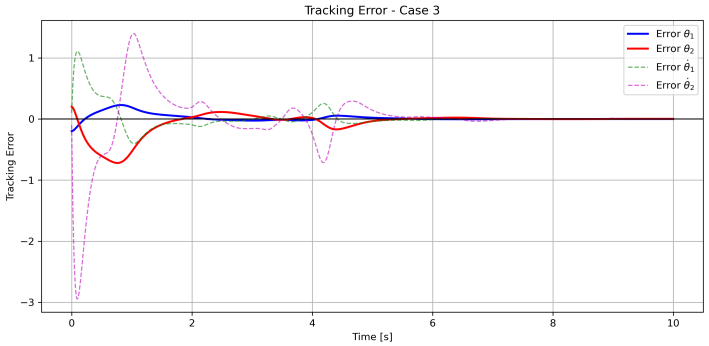
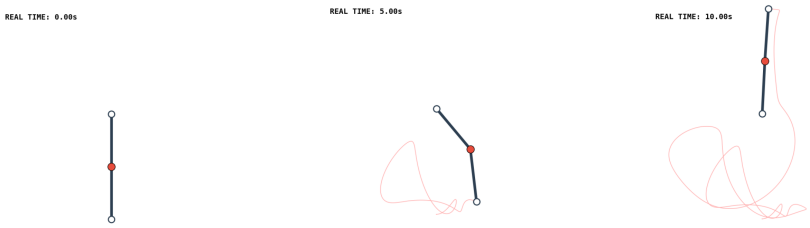


Figure 5.28: Task 4: Tracking Error (Perturbation 3).

5.5 Animation

The following sequence illustrates the Acrobot's swing-up maneuver as computed by the trajectory generation algorithm. Starting from the stable downward equilibrium (left), the robot pumps energy through coordinated hip torques, reaching a mid-swing configuration (center), and finally stabilizes at the unstable inverted equilibrium (right). This demonstrates the successful execution of the optimal control strategy developed throughout this project.



(a) Initial position ($t = 0$) (b) Mid-swing ($t = 5$ s) (c) Final position ($t = T$)

Figure 5.29: Acrobot swing-up animation sequence: from stable hanging to inverted equilibrium.

Note that the animation is stored in gif format in the figs folder.

Conclusions

This project has successfully demonstrated a complete optimal control framework for the swing-up and stabilization of the Acrobot, an underactuated nonlinear robotic system. The development progressed through four interconnected phases, each building upon the previous to achieve the final objective.

The Role of Discretization

The accurate discretization of the continuous-time dynamics proved to be a fundamental prerequisite for the entire project. By employing a 4th-order Runge-Kutta (RK4) scheme with a sufficiently small sampling time ($d_t = 0.01$ s), we ensured that the discrete-time model faithfully captures the nonlinear behavior of the Acrobot, including the Coriolis, centrifugal, and gravitational effects. This high-fidelity discretization is essential not only for realistic forward simulation but also for obtaining accurate Jacobian matrices (A_t, B_t) via finite differences. A poor discretization would have introduced numerical errors that propagate through the optimization and control stages, leading to suboptimal or even divergent solutions.

Trajectory Generation: iLQR as an Efficient Planning Strategy

For the trajectory generation phase (Tasks 1 and 2), we adopted the iterative Linear Quadratic Regulator (iLQR) algorithm, which can be viewed as a computationally efficient variant of Differential Dynamic Programming (DDP). While DDP forms a second-order approximation of the value function and involves computing Hessians of the dynamics, iLQR simplifies the problem by neglecting the second-order terms—specifically the products of Hessians with the cost-to-go—and relies solely on first-order (Jacobian) linearizations. This simplification results in significant computational savings without a substantial loss in convergence speed for many practical systems.

The essence of iLQR lies in its dynamic programming structure: at each iteration, a backward pass solves a sequence of local LQR subproblems to

compute optimal feedback gains (K_t) and feedforward corrections (σ_t), followed by a forward pass that simulates the system with the updated control law. This approach efficiently generates a feasible and locally optimal trajectory, serving as the *planning phase* of the control problem. The choice of reference trajectory (step vs. smooth) significantly affected convergence, highlighting the importance of providing the optimizer with a physically consistent initial guess.

It is important to note that this task could have been performed using other types of closed-loop optimization algorithms.Ultimately, we chose this approach to establish a direct connection with Task 3. In terms of implementation, it is crucial to recognize that the feedback gain K_T from the final iteration of the iLQR (used for trajectory generation in Task 2) will be identical to the optimal gain K^*_{reg} obtained when solving the standard LQR for trajectory tracking. This mathematical equivalence occurs because, in both cases, the linearization of the nonlinear dynamics is performed around the exact same optimal reference trajectory.

Trajectory Tracking: Offline LQR Design for Real-Time Execution

The tracking problem (Task 3) fundamentally differs from the generation problem in its objective and structure. Once the optimal trajectory (x^*_t, u^*_t) has been generated, the goal shifts to *maintaining* the system on this reference path despite perturbations and model uncertainties. We formulated this as a standard Time-Varying LQR problem with the following key characteristics:

- **Linearization Over the Optimal Trajectory:** The system dynamics are linearized around the pre-computed optimal trajectory (not around equilibrium points), yielding time-varying Jacobians A_t^{traj} and B_t^{traj}.
- **Incremental Formulation:** The optimization variables are the state and control *deviations* from the nominal path (Δx_t, Δu_t), not the absolute states.
- **Quadratic Cost, Linear Constraints:** The resulting problem is a standard LQR with a quadratic objective and linear dynamics, which admits a closed-form solution via the Riccati difference equations.
- **One-Shot Offline Computation:** Unlike iLQR which iterates until convergence, the tracking LQR requires only a *single backward pass* to compute all feedback gains K_t^{reg}. This is because the problem is already linear-quadratic by construction—there is no need for iterative linearization.

This offline pre-computation of the gains enables real-time control execution: during operation, the controller simply reads the current state, computes the deviation, and applies the pre-stored gain to generate the corrective input with minimal computational overhead.

LQR vs. MPC: A Comparative Perspective

Both the Time-Varying LQR and Model Predictive Control (MPC) are trajectory-tracking strategies, but they differ significantly in philosophy and implementation:

Aspect	**LQR Tracking**	**MPC Tracking**
Computation	All gains computed *offline* in a single backward pass.	Optimization solved *online* at each time step.
Constraint Handling	Cannot explicitly handle state/input constraints (relies on linear assumptions).	Naturally incorporates inequality constraints (e.g., torque limits, joint angle limits).
Receding Horizon	Uses a fixed terminal time T matching the trajectory length.	Uses a rolling prediction horizon that advances with time.
Real-Time Feasibility	Extremely fast execution (simple matrix multiplication).	Computationally heavier; requires efficient solvers for real-time use.
Robustness	Degrades for large perturbations (linear approximation breaks down).	Re-optimizes at each step, adapting to the current state more robustly.

Table 5.1: Comparison between LQR and MPC for Trajectory Tracking

In essence, LQR is optimal for small perturbations and when computational resources are limited, while MPC offers greater flexibility and robustness at the cost of increased online computation. For the Acrobot swing-up task, both approaches successfully tracked the reference trajectory, but MPC demonstrated superior performance under larger initial perturbations where the LQR's linear assumptions become less accurate.

Bibliography

[1] Giuseppe Notarstefano. Optimal control and reinforcement learning: Lq optimal control. Lecture Slides, University of Bologna, 2025. Course Material.

[2] Giuseppe Notarstefano. Optimal control and reinforcement learning: Model predictive control. Lecture Slides, University of Bologna, 2025. Course Material.

[3] Giuseppe Notarstefano. Optimal control and reinforcement learning: Optimal control based trajectory generation and tracking. Lecture Slides, University of Bologna, 2025. Course Material.