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:

- $M(q) \in \mathbb{R}^{2\times2}$ is the symmetric, positive-definite mass matrix.
- $C(q, \dot{q}) \in \mathbb{R}^{2\times2}$ represents Coriolis and centrifugal forces.
- $F \in \mathbb{R}^{2\times2}$ 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. 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:

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

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 Affine LQR)

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 **Affine Linear Quadratic Regulator (LQR)** problem to find the optimal control deviation, this sequence of solved subproblems forms the Newton-type update. [**?**].

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 $\bar{u}$ to generate the nominal trajectory.

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

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 Affine LQR)

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 using the terminal cost Hessian and gradient.

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

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`).
- Cost Gradients (q_t, r_t): `q_t`, `r_t` (via `Q @ dx`, `R @ du`).
- 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:

- $S_t \to$ `S_mat = self.R + B_t.T @ P @ B_t`
- $S_t^{-1} \to$ `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 [**?**]:

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

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

Code Mapping:

- $K_t^* \to$ `K_t` (stored in `K_gains`)
- $\sigma_t^* \to$ `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:

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

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

Code Mapping:

- $K_t^{*\top} S_t K_t^* \to$ `term_quad` (The quadratic reduction)

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

$$u_{new}(t) = \bar{u}_t + \alpha \cdot \sigma_t^* + K_t^* \cdot (x_{new}(t) - \bar{x}_t) \tag{2.11}$$

Code Mapping:

- $\alpha \to$ `alpha` (Determined via Armijo Line Search)
- $x_{new}(t) - \bar{x}_t \to$ `dx_deviation`
- Update $\to$ `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_k^*, u_k^*) generated in the previous chapter serves as the nominal path. To handle disturbances and model uncertainties, we linearize the discrete-time dynamics $x_{k+1} = f_d(x_k, u_k)$ around this nominal trajectory:

$$\delta x_{k+1} \approx A_k \delta x_k + B_k \delta u_k \tag{3.1}$$

where $\delta x_k = x_k - x_k^*$, $\delta u_k = u_k - u_k^*$, and the Jacobian matrices are defined as:

$$A_k = \left.\frac{\partial f_d}{\partial x}\right|_{x_k^*, u_k^*}, \quad B_k = \left.\frac{\partial f_d}{\partial u}\right|_{x_k^*, u_k^*} \tag{3.2}$$

These Jacobians are computed numerically using finite differences to account for the complexity of the RK4 integration.

3.2 Discrete-Time Finite-Horizon LQR

We formulate a tracking problem to minimize the deviation from the nominal trajectory:

$$\min_{\delta u_k} \sum_{k=0}^{N-1} (\delta x_k^\top Q \delta x_k + \delta u_k^\top R \delta u_k) + \delta x_N^\top Q_{final} \delta x_N \tag{3.3}$$

The optimal control law is a time-varying linear feedback:

$$\delta u_k = -K_k \delta x_k \tag{3.4}$$

where the feedback gain K_k is computed by solving the discrete-time Riccati difference equations backwards in time:

$$P_N = Q_{final} \tag{3.5a}$$
$$K_k = (R + B_k^\top P_{k+1} B_k)^{-1} B_k^\top P_{k+1} A_k \tag{3.5b}$$
$$P_k = Q + A_k^\top P_{k+1} A_k - A_k^\top P_{k+1} B_k K_k \tag{3.5c}$$

The control input applied to the robot is then $u_k = u_k^* - K_k(x_k - x_k^*)$. This approach provides local stability and robust tracking performance.

Chapter 4

Task 4: Trajectory Tracking via MPC

This chapter discusses the implementation of Model Predictive Control (MPC) to track the optimal reference, accounting for system constraints and performance.

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.

(a) Optimal vs Desired (q_1)

(b) Optimal vs Desired (q_2)

(c) Intermediate Trajectories (q_1)

(d) Intermediate Trajectories (q_2)

(e) Armijo Direction (Initial)

(f) Armijo Direction (Final)

(g) Norm of Descent (Log Scale)

(h) Cost along Iterations (Log Scale)

(i) Optimal Control $u(t)$

(j) Phase Portrait

Figure 5.1: Trajectory Generation Results for Task 1: 10 required plots illustrating convergence and optimality.

5.2 Task 2: Trajectory Generation (Smooth Reference)

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

(a) Optimal vs Desired (q_1)

(b) Optimal vs Desired (q_2)

(c) Intermediate Trajectories (q_1)

(d) Intermediate Trajectories (q_2)

(e) Armijo Direction (Initial)

(f) Armijo Direction (Final)

(g) Norm of Descent (Log Scale)

(h) Cost along Iterations (Log Scale)

(i) Optimal Control $u(t)$

(j) Phase Portrait

Figure 5.2: Trajectory Generation Results for Task 2: 10 required plots illustrating improved convergence with smooth reference.

5.3 Task 3: LQR Tracking Performance

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

(a) System Trajectory vs Desired Optimal Trajectory

(b) Error (Perturbation 1) (c) Error (Perturbation 2) (d) Error (Perturbation 3)

Figure 5.3: LQR Tracking Results: System behavior and error analysis for different initial conditions.

5.4 Task 4: MPC Tracking Performance

Performance of the Model Predictive Controller in tracking the reference trajectory.

(a) System Trajectory vs Desired Optimal Trajectory

(b) Error (Perturbation 1) (c) Error (Perturbation 2) (d) Error (Perturbation 3)

Figure 5.4: MPC Tracking Results: System behavior and error analysis for different initial conditions.

5.5 Animation

Conclusions