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[cite: 37, 51, 61].

Contents

List of Figures

Introduction

The goal of this project is to design an optimal trajectory for a planar gymnast robot[cite: 3]. The system consists of two links where actuation is provided only at the second joint (the hip)[cite: 3, 12]. The dynamics involve nonlinear interactions described by mass, Coriolis, and gravity matrices[cite: 13, 16].

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. We employ a Newton-type optimal control algorithm (specifically, Differential Dynamic Programming or Iterative LQR) 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 $G(q)$ contains non-linear trigonometric terms $(\sin(\theta_1), \sin(\theta_1 + \theta_2))$, 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

With the boundary conditions and reference defined, we implement the Newton-type optimal control algorithm. This is an iterative method that improves a control sequence u by minimizing a quadratic approximation of the cost function. The process consists of three distinct phases per iteration.

2.3.1 Phase 1: Forward Pass (Simulation)

In the forward pass, we simulate the non-linear dynamics of the robot using the current control sequence. For the first iteration, we initialize with $u = 0$ (passive dynamics).

$$x_{t+1} = f(x_t, u_t), \quad t = 0, \dots, N-1 \tag{2.4}$$

This step generates the nominal trajectory $\bar{x}, \bar{u}$ around which we will linearize.

2.3.2 Phase 2: Backward Pass (Riccati Recursion)

This phase is the core of the Newton algorithm. We compute the optimal control modification by solving the problem backwards in time, from $t = N$ to 0.

First, we compute the derivatives of the dynamics (A_t, B_t) and the quadratic expansion of the cost function $(l_x, l_u, l_{xx}, l_{uu})$ around the nominal trajectory. We then define the **Action-Value Function** $Q(x, u)$, which represents the cost of taking action u at state x and acting optimally thereafter:

$$Q(x, u) = L(x, u) + V_{next}(f(x, u)) \tag{2.5}$$

Applying the quadratic expansion, the gradients and Hessians of Q are computed as:

$$Q_x = l_x + A_t^\top V_x' \tag{2.6}$$
$$Q_u = l_u + B_t^\top V_x' \tag{2.7}$$
$$Q_{xx} = l_{xx} + A_t^\top V_{xx}' A_t \tag{2.8}$$
$$Q_{uu} = l_{uu} + B_t^\top V_{xx}' B_t \tag{2.9}$$
$$Q_{ux} = l_{ux} + B_t^\top V_{xx}' A_t \tag{2.10}$$

where V' denotes the Value function at the next time step.

To find the optimal control modification δu, we minimize the quadratic model with respect to u:

$$\frac{\partial Q}{\partial u} = Q_u + Q_{uu}\delta u + Q_{ux}\delta x = 0 \tag{2.11}$$

Solving for δu yields the optimal control law consisting of a feedforward term k and a feedback gain K:

$$\delta u^* = \underbrace{-Q_{uu}^{-1}Q_u}_{k \text{ (Feedforward)}} + \underbrace{-Q_{uu}^{-1}Q_{ux}}_{K \text{ (Feedback)}} \delta x \tag{2.12}$$

Finally, we update the Value function $V(x)$ for the previous time step to propagate the cost information backwards.

2.3.3 Phase 3: Update (Closed-Loop Rollout)

Once the gains k and K are computed, we apply them to generate the new control sequence. Crucially, we use a **Closed-Loop** update rule during the new forward simulation.

Instead of simply applying $u_{new} = u_{old} + \alpha k$, we calculate:

$$u_{new}(t) = u_{old}(t) + \alpha \cdot k_t + K_t \cdot (x_{new}(t) - x_{old}(t)) \tag{2.13}$$

where α is a step size determined by an Armijo line search to ensure cost reduction.

The inclusion of the feedback term $K_t(x_{new} - x_{old})$ is vital. Since the Acrobot is highly unstable, a purely open-loop update would cause the new trajectory to diverge exponentially from the linearized path, rendering the quadratic approximation invalid. The feedback term stabilizes the rollout, keeping the robot near the planned path and ensuring the convergence of the Newton algorithm.

Chapter 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:

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

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

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[cite: 61].

Chapter 5

Results and Animation

5.1 Simulation Environment

The performance of the proposed control strategies was evaluated in a simulated environment using the physical parameters specified in Table 1.1. The simulation was initialized at the downward equilibrium and commanded to reach the inverted position.

5.2 Task 1: Trajectory Generation Performance

In this section, we analyze the convergence and optimality of the Newton-type optimization algorithm for the swing-up maneuver.

Plot 1-1 *Plot 1-2*
Plot 1-3 *Plot 1-4*
Plot 1-5 *Plot 1-6*
Plot 1-7 *Plot 1-8*
Plot 1-9 *Plot 1-10*

Figure 5.1: Performance metrics for Task 1 (Trajectory Generation).

5.3 Task 2: LQR Tracking Results

The effectiveness of the LQR controller in tracking the nominal swing-up trajectory is demonstrated through the following set of results.

Plot 2-1 *Plot 2-2*
Plot 2-3 *Plot 2-4*
Plot 2-5 *Plot 2-6*
Plot 2-7 *Plot 2-8*
Plot 2-9 *Plot 2-10*

Figure 5.2: Tracking performance and error metrics for Task 2 (LQR Tracking).

5.4 Task 3: Robustness Analysis

A specific scenario evaluating the robustness of the controller is presented here.

Figure 5.3: Robustness metric for Task 3.

5.5 Task 4: Comparative Study

Final comparison between the implemented control techniques.

Figure 5.4: Comparative analysis for Task 4.

5.6 Animation

Conclusions