Solving Cartpole: A Comparison Study between Classical Controllers and Reinforcement Learning

Student Name: Sakshi Bhatia

ME5659: Control Systems Engineering
Course Project

on December 9, 2025

Submitted to:
Prof. Tong Ma

Northeastern University
Boston, MA

Abstract

Classical control methods have long been the foundation of optimal control for linear and non-linear dynamical systems. These methods use explicit system models and cost functions to compute optimal control laws analytically. In contrast, RL has emerged as a popular data-driven approach capable of learning optimal control policies directly from interactions with the environment, without requiring precise knowledge of system dynamics. This report presents a comparative study between classical controllers and RL agents on the CartPole control task. Our results highlight the strengths and limitations of both approaches, providing insights into their respective domains of applicability and the potential for hybrid control strategies that combine model-based and learning-based paradigms. Code is available on Github at https://github.com/sakshi79/ME5659.

Keywords: Control Systems, reinforcement learning, LQR, LQI, DQN

Contents

Chapter 1

Introduction

Classical control techniques, such as PID, LQR, and LQI, have formed the backbone of control engineering for decades. These methods are grounded in well-established mathematical theory, rely on explicit system modeling, and offer strong guarantees regarding stability, optimality, and robustness. Their interpretability and low computational cost have made them the preferred choice in safety-critical and industrial systems, including aircraft control, automotive systems, and robotics. However, many real-world control problems involve highly nonlinear, high-dimensional dynamics. As systems become more complex, deriving accurate analytical models or solving optimal control problems becomes increasingly challenging. Reinforcement Learning (RL) has emerged as a promising alternative, a data-driven framework capable of learning control policies directly from interaction with the environment. RL does not require an explicit model of the system and can, in principle, handle nonlinear, underactuated, or hybrid systems with complex constraints. Despite its flexibility, RL comes with its own challenges: long training times, limited stability guarantees, and a lack of interpretability compared to classical counterparts. These differences raise an important practical question: when is RL actually necessary or beneficial, and when are classical controllers sufficient, or even superior? This is the motivation for our comparison study. We examine the performance of two classical controllers, i.e, LQR and LQI, and one Reinforcement Learning algorithm, DQN on the OpenAI Gym CartPole environment. By examining both approaches under similar experimental conditions, this study provides a grounded understanding of their respective strengths and limitations. Such insight is essential not only for researchers exploring advanced control methods but also for practitioners choosing appropriate controllers for real-world systems.

Chapter 2

Methodology

This section describes the control problem, algorithms, and implementation details of this study. We implemented three controllers evaluated in this study: LQR, LQI, and Deep Q-Network(DQN). We detail the mathematical formulations, assumptions, parameter choices, and training procedures to ensure replicability of the comparison. [1]

2.1 Problem Setup: CartPole Environment

We use the standard CartPole-v1 environment from OpenAI Gym. The objective is to balance an inverted pendulum mounted on a frictionless cart by applying horizontal forces.

2.1.1 State Representation

The environment provides a 4-dimensional continuous state vector:

$$s = [x, \dot{x}, \theta, \dot{\theta}]$$

where

x : cart position

$\dot{x}$: cart velocity

θ : pole angle ($0 =$ upright)

$\dot{\theta}$: pole angular velocity

[1] The implementations of the Linear Quadratic Regulator (LQR), Linear Quadratic Integral (LQI) controller, and Deep Q-Network (DQN) used in this study are based on standard formulations from the control and reinforcement learning literature, as well as the canonical OpenAI Gym CartPole environment. This study primarily focuses on benchmarking and comparative analysis rather than introducing novel algorithms. Detailed references and formal citations will be included in future versions.

2.1.2 Action space

The action space is discrete:

$$a \in \{0, 1\}$$

mapped to forces:

$0 \rightarrow -F$

$1 \rightarrow +F$

where $F = 10N$, by default.

2.1.3 Reward Function

OpenAI Gym provides $r_t = 1$ for every time step the pole remains upright.
Episode termination occurs when one of these conditions is true:

$$|x| > 2.4m$$

$$|\theta| > 12^\circ$$

$$\text{episode length} \geq 500 \text{ (truncation)}$$

The environment is considered solved if the pole doesn't fall for at least 199 continuous timesteps.

2.2 System Modeling

Figure 2.1 illustrates the system dynamics of our CartPole. We have the following system parameters defined in the OpenAI environment:

$$l = 0.5m$$

$$g = 9.8m/s^2$$

$$m_p = 0.1m$$

$$J_p = m_p * (2l)$$

$$m_k = 1.0kg$$

$$m_p = 0.1kg$$

$$m_T = m_k + m_p = 1.1kg$$

$$\Delta t = 0.02s$$

From the system dynamics equations, we get the following:

$$\ddot{s} = \frac{u + m_p l \dot{\theta}^2 sin\theta - m_p g sin\theta cos\theta}{m_k + m_p - m_p cos^2\theta} \tag{2.1}$$

$$\ddot{\theta} = \frac{-ucos\theta - m_p l\dot{\theta}^2 sin\theta cos\theta + m_T g sin\theta}{l(m_c + m_p - m_p cos^2\theta)} \tag{2.2}$$

where u = F (external applied force), is the control input.

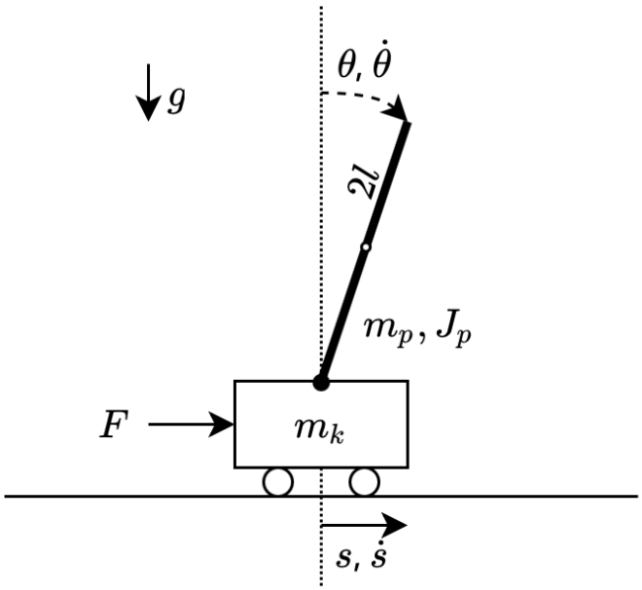

Figure 2.1: CartPole dynamics

2.2.1 Linearized Equations of Motion

To apply LQR/LQI, we linearize the nonlinear CartPole dynamics around the upright equilibrium ($\theta = 0$) using first-order Taylor expansion.

$$\text{Let } p = \frac{m_p}{m_k + m_p}$$

Linearized system model:

$$\dot{x} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & pg & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & \frac{g(1+p)}{l} & 0 \end{bmatrix} x + \begin{bmatrix} 0 \\ \frac{1}{m_c + m_k} \\ 0 \\ \frac{1}{l(m_c + m_k)} \end{bmatrix} u \tag{2.3}$$

This continuous system is discretized using zero-order hold with the environment timestep $\Delta t = 0.02s$.

2.3 Linear Quadratic Regulator (LQR)

LQR computes an optimal linear feedback law:

$$u_t = -Kx_t \tag{2.4}$$

by solving

$$\min_u \int (x^T Q x + u^T R u)\ dt \tag{2.5}$$

2.3.1 Cost Matrices

To prioritize angle stabilization, $Q = diag(1, 0.1, 10, 1)$.
For energy, efficiency, $R = I$

2.3.2 Riccati Equation

Solve the Discrete Algebraic Riccati Equation:

$$A^T P \ + \ PA \ - \ PBR^{-1}B^T P \ + \ Q \ = \ 0$$

$$\text{Then, } K = R^{-1}B^T P \tag{2.6}$$

We substitute this in 2.4, to get control input u_t.

2.4 Linear Quadratic Integral (LQI)

LQI augments LQR with integral action to eliminate steady-state error in cart position.

2.4.1 Error Integration

Define:

$$e_t = x_{ref} - x_t \tag{2.7}$$

$$\dot{z} = e_t \tag{2.8}$$

Augmented state,

$$x_a = \begin{bmatrix} x \\ z \end{bmatrix} \tag{2.9}$$

which leads to an augmented system:

$$A_a = \begin{bmatrix} A & 0 \\ -C & 0 \end{bmatrix}, \quad B_a = \begin{bmatrix} B \\ 0 \end{bmatrix} \tag{2.10}$$

We now solve the LQR problem with augmented matrices:

$$u_t = -K_a x_a \tag{2.11}$$

with integral compensation improving the long-term balancing stability.

2.5 Deep Q-Network (DQN)

DQN learns a parametric action-value function:

$$Q(s, a; \theta)$$

where θ denotes the parameters that the Neural Network (NN) learns during training, and selects the action to execute as

$$a_t = arg\max_a Q(s_t, a)$$

In each training episode, we first reset the environment, then take steps in the environment until we reach termination or truncation condition, and store these interaction steps in a memory buffer. After every few steps, we sample a batch from this memory buffer and update the NN parameters by doing on a gradient descent on this batch.

A Deep Q-Network (DQN) uses two separate neural networks during training:

2.5.1 Online (Policy) Network

This network has parameters θ, and it is used to select actions, compute Q-values for current state, and perform gradient updates. Formally, it can be represented as $Q_{online}(s, a; \theta)$. This is the network that actually learns and whose weights change at every training step.

2.5.2 Target Network

This network has parameters θ^-, and it is a delayed copy of the online network. It is used only for computing stable target values in the Bellman update. This network is represented as $Q_{target}(s, a; \theta)$. Crucially, its weights are frozen for many steps and updated only periodically. This stabilizes learning.

2.5.3 Bellman Update Using the Target Network

The target or desired value for a transition (s, a, r, s') is:

$$y \;=\; r + \gamma \max_{a'} Q_{target}(s', a'; \theta_-) \tag{2.12}$$

The online network update minimizes:

$$L(\theta) \;=\; (Q_{online}(s, a; \theta) - y)^2 \tag{2.13}$$

We only compute gradients w.r.t. θ. θ^- remains fixed.
We train our DQN model for 40k steps in the environment and report the following loss curves as shown in figure 2.2.

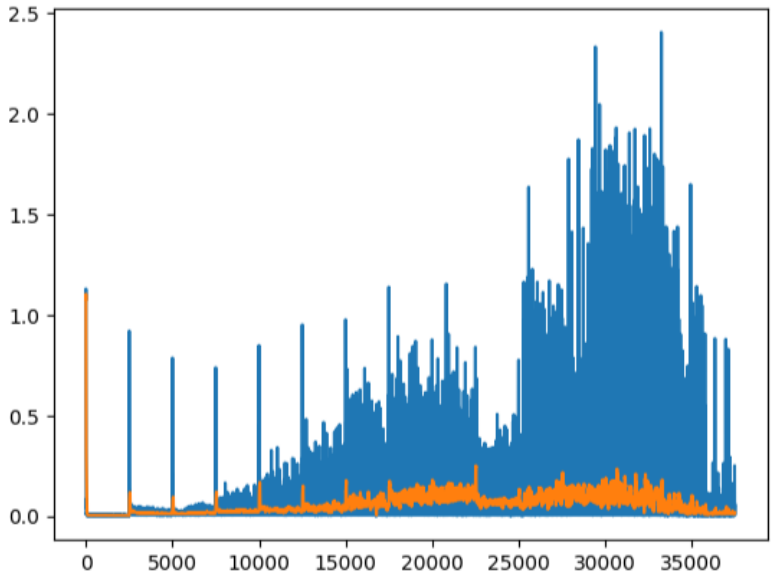

Figure 2.2: Training loss for DQN.

Chapter 3

Results

This section presents a comparative evaluation of LQR, LQI, and DQN controllers on the OpenAI Gym CartPole-v1 environment. Performance is analyzed in terms of control stability, success rate (defined as preventing the pole from falling), sample efficiency, and qualitative behavior. Success is defined as completing a minimum of 199 timesteps in the environment without the pole falling.
We present a comparison of the three control methods: LQR, LQI, and DQN.

3.1 Performance of LQR

Figures 3.1 and 3.2 show the performance of the LQR controller in solving the CartPole environment. We make the following observations with this controller:

1. 100% success rate from the first timestep.
2. Stabilizes the pole immediately.
3. Smooth and minimal-energy control inputs.
4. Does not regulate cart position; cart drifts over time.

3.2 Performance of LQI

Figures 3.3 and 3.4 show the performance of the LQI controller in solving the CartPole environment. We make the following observations with this controller:

1. 100% success rate from the first timestep.
2. Pole stabilizes immediately, while cart position is gradually driven toward the center.
3. Slightly higher control effort due to integral action.
4. Eliminates steady-state cart drift observed in LQR.

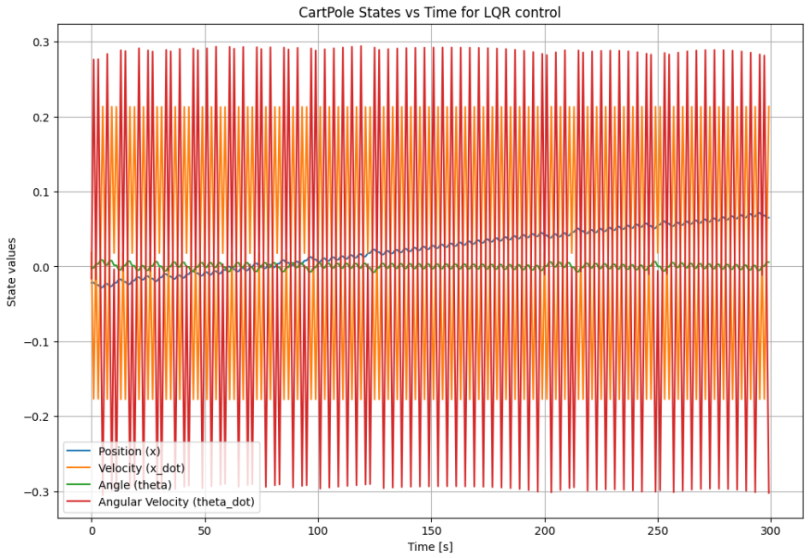

Figure 3.1: State variables vs Time: LQR

```
env = gym.make('CartPole-v1')
obs, info = env.reset()
states = []
times = []
dt = 0.2  # from cartpole github
t = 0.0
max_steps = 300

while True:
    states.append(obs)
    times.append(t)
    t += 1
    action, force = apply_state_controller(K, obs)
    abs_force = abs(float(np.clip(force, -10, 10)))
    env.env.force_mag = abs_force
    # apply action
    obs, reward, terminated, truncated, info = env.step(action)
    if terminated:
        print(f'Terminated on falling at timestep: {t}.')
        t = 0.0
    if t==max_steps:
        print(f"{max_steps} steps in the env without falling.")
        break
```

```
300 steps in the env without falling.
```

Figure 3.2: Steps completed in the environment by LQR.

3.3 Performance of DQN

Figures 3.3 and 3.4 show the performance of the trained DQN model in solving the CartPole environment. We make the following observations with this controller:

1. Trained for 40k environment steps.
2. Best model achieved only 125 steps on average before pole fell (environment not solved)
3. Observed high variability: some episodes terminate even earlier, others last slightly longer.
4. **Did not achieve reliable balancing** for this environment, unlike classical controllers

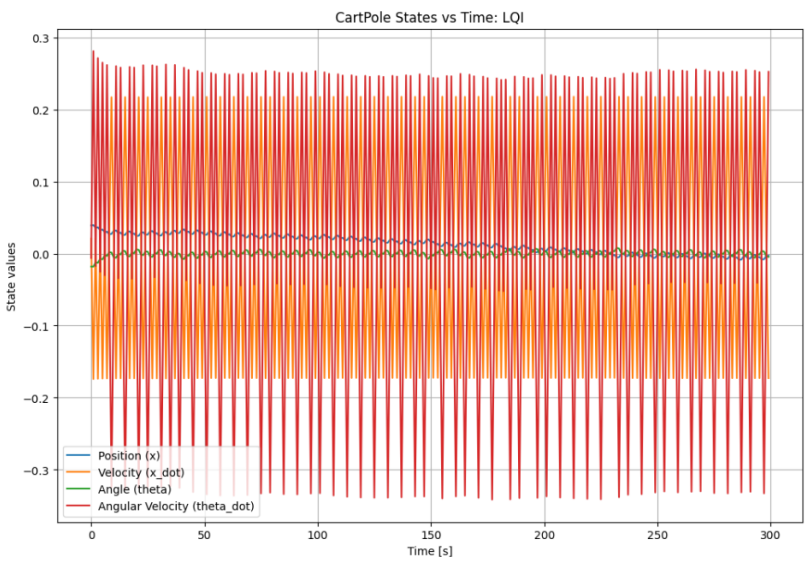

Figure 3.3: State variables vs Time: LQI

```python
env = gym.make('CartPole-v1')
obs, info = env.reset()
states = []
times = []
dt = 0.2  # from cartpole github
t = 0.0
max_steps = 300
while True:
    states.append(obs)
    times.append(t)
    t += 1
    action, force = lqi_controller(K, obs)
    abs_force = abs(float(np.clip(force, -10, 10)))
    # change magnitude of the applied force in CartPole
    env.env.force_mag = abs_force
    # apply action
    obs, reward, terminated, truncated, info = env.step(action)
    if terminated:
        print(f'Terminated on falling at timestep: {t}.')
        t = 0.0
    if t==max_steps:
        print(f"{max_steps} steps in the env without falling.")
        break
```

```
300 steps in the env without falling.
```

Figure 3.4: Steps completed in the environment by LQI.

```python
model_dict = final_model
loaded_model = DQN(*model_dict['args'], **model_dict['kwargs'])
loaded_model.load_state_dict(model_dict['state_dict'])
loaded_model.eval()
state, _ = env.reset()
done = False
t = 0.0
while not done and t<300:
    t += 1
    action = loaded_model(torch.tensor(state, dtype=torch.float32)).argmax().item()
    next_state, reward, done, trunc, _ = env.step(action)
    state = next_state
t
```

```
125.0
```

Figure 3.5: Steps completed in the environment by the trained DQN model.

3.4 Discussion and Conclusion

We observe that the classical controllers such as LQR and LQI outperform DQN in simple, linearizable tasks. LQI and LQR are both able to *solve* the environment (stay upright without falling), while DQN fails to do so. Additionally, Classical controllers require no training data and are immediately deployable, whereas DQN requires extensive interaction with the environment, yet fails to converge to a fully stable policy. Between LQR and LQI, we find LQI to be better since LQI improves upon LQR by regulating cart position without compromising stability.

We conclude that RL is not universally superior, and analytical controllers are highly effective in low-dimensional, well-understood systems. Classical controllers are, in addition, also sample-efficient. While DQN is theoretically capable of learning nonlinear recovery strategies, the CartPole environment is simple enough that the advantages of nonlinear policies do not outweigh the challenges of learning from sparse rewards and limited exploration.