HW2: LQR and DQN and PPO Based Control for Cart-Pole Stabilization Under Earthquake Disturbances

RAS 598: Space Robotics and AI

Aldrin Inbaraj A
ASU ID: 1226200393

1 LQR Controller

1.1 Problem Statement

The cart-pole system consists of an inverted pendulum mounted on a cart, moving along a track. The objective is to use an LQR controller to stabilize the pole while keeping the cart within ±2.5 m, even under earthquake-like disturbances.

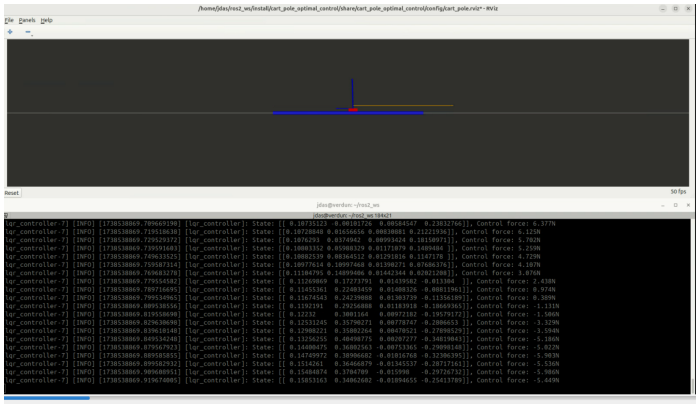

Figure 1: Cart-Pole System Under Earthquake Disturbances

1.2 System Constraints

- Cart range: ±2.5 m
- Pole length: 1.0 m
- Cart mass: 1.0 kg
- Pole mass: 1.0 kg

Disturbance:

- Earthquake force generator
- Amplitude: 15.0 N
- Frequency range: 0.5 - 4.0 Hz
- Includes Gaussian noise

1.3 Tuning of LQR

The Linear Quadratic Regulator (LQR) is an optimal control approach that minimizes the following cost function:

$$J = \int_0^\infty \left(x^T Q x + u^T R u \right) dt$$

where:

- x is the state vector $[x, \dot{x}, \theta, \dot{\theta}]$.
- u is the control input (force applied to the cart).
- Q is the state cost matrix, penalizing deviations from the desired state.
- R is the control effort cost, penalizing excessive force inputs.

1.4 Analysis of LQR Controller with Varying Q and R Matrices

1.4.1 Logic for Calculating the Stability Score

The stability score is computed based on key performance metrics: cart displacement, pole angle deviation, and control effort. The score penalizes excessive movements and inefficient control, ensuring that a well-balanced controller achieves the highest rating.

$$\text{stability\_score} = \max\left(0, 10 - (\text{max\_cart\_displacement} \times 2) - \frac{\text{max\_pole\_deviation}}{5} - \frac{\text{avg\_control\_effort}}{20}\right)$$

1.4.2 Time Limit

A time limit of 120 seconds is set to ensure that if the model remains stable for an extended period, it exits the execution and records performance data.

1.4.3 Varying Q while keeping R constant at 0.1

We first analyze the effect of different Q matrices on system performance while keeping $R = 0.1$. The goal is to find the optimal Q matrix that maximizes stability while maintaining a reasonable control effort.

Q Values	Max Pole Angle Deviation (°)	Max Cart Displacement (m)	Average Control Effort (N)	Duration of Stability (s)	Stability Score
[1, 1, 10, 10]	5.628	2.500	3.752	39.93	3.69
[10, 10, 1, 1]	7.442	0.386	8.524	120.01	7.31
[10, 10, 50, 50]	9.808	0.444	13.515	120.00	6.48
[5, 5, 20, 20]	3.220	0.141	6.899	120.01	8.73
[5, 5, 30, 30]	5.079	0.206	8.801	120.00	8.13
[10, 10, 30, 30]	4.846	0.128	10.636	120.00	8.24

Table 1: Performance metrics of LQR controller for varying Q values (Fixed R = 0.1).

1.4.4 Varying R while keeping Q constant at [5,5,20,20]

After selecting the best-performing Q matrix ($Q = [5, 5, 20, 20]$) from the first step, we then vary R to study how different control effort penalties affect system stability and performance.

R Value	Duration (s)	Max Cart Displacement (m)	Max Pole Angle Deviation (°)	Average Control Effort (N)	Stability Score
0.05	120.00	0.081	3.102	5.659	8.93
0.10	120.00	0.202	4.782	8.056	8.24
0.50	24.97	2.500	5.844	2.916	3.69
1.00	13.51	2.500	9.913	6.612	2.69

Table 2: Performance metrics of LQR controller for varying R values (Fixed Q = [5,5,20,20]).

1.5 Performance Plots

1.5.1 LQR Performance for Varying Q and R Matrices

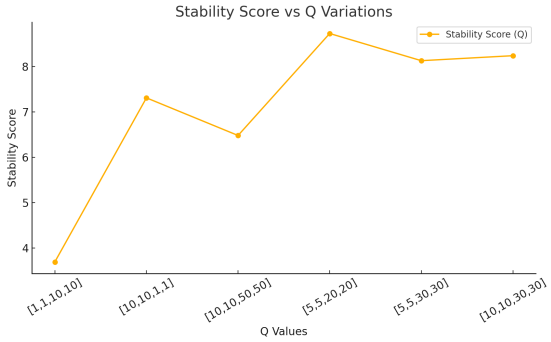

Figure 2: Stability Score vs Q Variations

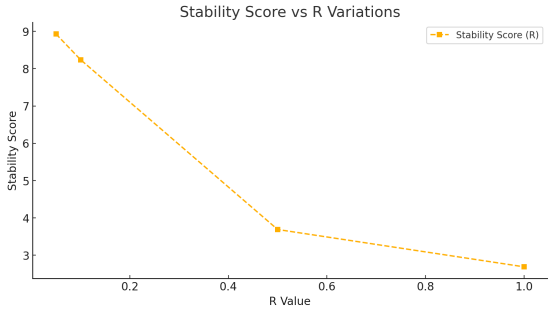

Figure 3: Stability Score vs R Variations

1.5.2 Default vs Optimal LQR Controller Performance

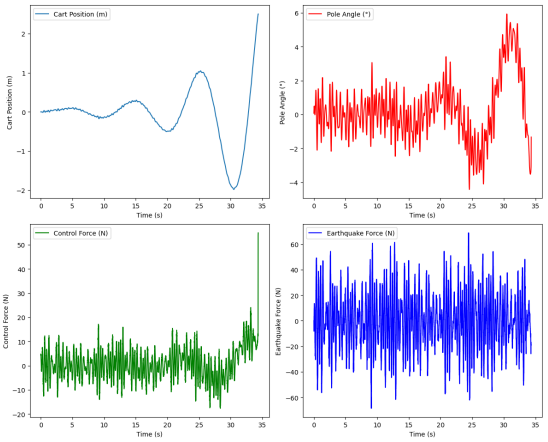

Figure 4: Default LQR Controller (Q = [1,1,10,10], R = 0.1)

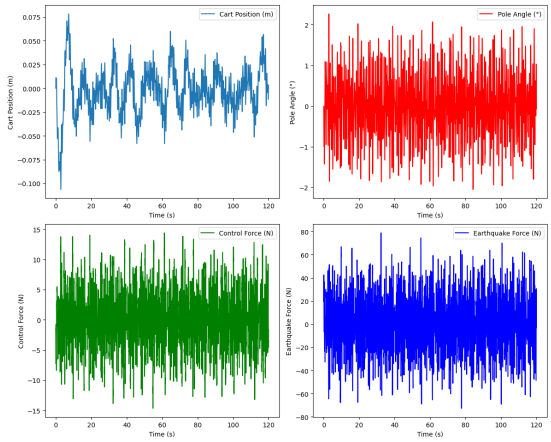

Figure 5: Optimal LQR Controller (Q = [5,5,20,20], R = 0.05)

1.5.3 Controller Output Logs

Default Controller (Q = [1,1,10,10], R = 0.1)

```
[lqr_controller]: Duration of stable operation: 39.93 s
[lqr_controller]: Maximum cart displacement: 2.500 m
[lqr_controller]: Maximum pendulum angle deviation: 5.628°
[lqr_controller]: Average control effort: 3.752 N
[lqr_controller]: Stability score: 3.69/10
```

Optimal Controller (Q = [5,5,20,20], R = 0.05)

```
[lqr_controller]: Duration of stable operation: 120.00 s
[lqr_controller]: Maximum cart displacement: 0.081 m
[lqr_controller]: Maximum pendulum angle deviation: 3.102°
[lqr_controller]: Average control effort: 5.659 N
[lqr_controller]: Stability score: 8.93/10
```

1.6 Discussion of Controller Behavior

- The initial LQR controller with $Q = [1, 1, 10, 10]$ and $R = 0.1$ exhibited poor stability:
  - The cart reached its physical limits with a displacement of 2.5 meters.
  - The pendulum deviated significantly by 5.628°.
  - The system failed after only 39.93 seconds.
- By tuning the Q matrix to $Q = [5, 5, 20, 20]$, stability improved significantly:
  - The maximum cart displacement was reduced to 0.081 m.
  - The pendulum remained within 3.102° of deviation.
  - The system remained stable for the full 120-second duration.
- Further tuning of the R value confirmed:
  - A smaller R value allows for more aggressive control.
  - This leads to better stabilization.
  - The final selection of $R = 0.05$ with $Q = [5, 5, 20, 20]$ provided the best trade-off between stability and control effort.

1.7 Video Demonstration

A video demonstration of the LQR controller shows the cart-pole system remaining stable for a duration of 120 seconds, after which the simulation ends and displays performance metrics.

Video Link: `https://drive.google.com/file/d/1xEGF7jzGVaF6QiLa4eflu6lhcKIFgfFK/view?usp=drive_link`

2 Deep Q-Network (DQN) for Cart-Pole with Earthquake Forces

2.1 Introduction

This section explores reinforcement learning using **Deep Q-Networks (DQN)** in the Gym CartPole environment with added earthquake disturbances. The goal is to analyze the model's ability to stabilize the system under external forces.

2.2 Gym CartPole Environment with Earthquake Forces

The base environment used is the **Gym CartPole-v1** simulation. We introduce **earthquake disturbances** into the environment to analyze the model's performance under dynamic external forces. The earthquake forces are applied at random intervals to simulate real-world seismic events.

2.3 DQN Implementation

The DQN implementation consists of three main components:

2.3.1 DQN Agent

The deep Q-network (DQN) is defined using a fully connected neural network that approximates the Q-value function. The key components of the agent include:

- **Q-Network:** Learns the state-action value function.
- **Target Network:** Stabilizes training by using a delayed update mechanism.
- **Experience Replay:** Stores past experiences to decorrelate samples.
- **Epsilon-Greedy Policy:** Balances exploration and exploitation.

2.4 Installation and Execution

To train and evaluate the DQN model, a Python environment with the required dependencies must be set up. The following steps outline the installation and execution process.

2.4.1 Setting Up the Environment

The project requires Python packages such as PyTorch and Gymnasium. It is recommended to use a virtual environment to manage dependencies.

Steps to Install and Set Up the Environment:

```
# Create and activate a virtual environment
python -m venv dqn_env
source dqn_env/bin/activate  # On Windows, use: dqn_env\Scripts\activate

# Install required packages
pip install torch gymnasium numpy matplotlib
```

2.4.2 Running the Training Script

Once the dependencies are installed, the model can be trained using the following command:

```
python dqn_train.py
```

The training script:

- Initializes the Gym CartPole environment.
- Trains the DQN model using an epsilon-greedy policy.
- Logs training progress, including rewards per episode.
- Saves the trained model to a file.

Expected Output: At the end of training, the model weights are saved as:

```
dqn_model_trained.pth
```

2.4.3 Evaluating the Trained Model

Once training is complete, the trained model can be evaluated using the following command:

```
python dqn_evaluate.py
```

The evaluation script:

- Loads the saved model (`dqn_model_trained.pth`).
- Runs test episodes to check the agent's performance.
- Evaluates the model with and without earthquake forces.

2.5 Training and Evaluation Without Earthquake Forces

The agent is first trained in a **zero-earthquake environment** to establish a baseline.

2.5.1 Hyperparameters for Training Without Earthquake Forces

Parameter	Value
Learning Rate	0.001
Discount Factor (Gamma)	0.99
Replay Buffer Size	100,000
Batch Size	64
Target Network Update Frequency	1000 steps
Epsilon Decay	0.999
Number of Episodes	500

Table 3: Hyperparameters for DQN training without earthquake forces.

2.5.2 Rewards Function

The reward function encourages the agent to maintain pole stability while preventing excessive cart movement. It is defined as follows:

$$R(s, a) = 1.0 + (1.0 - 2.5 \cdot \text{pole\_angle}) + \text{cart\_stability}$$

where:

- **1.0** is a base reward for surviving each timestep.
- **(1.0 - 2.5 * pole angle)** penalizes deviations from the upright position.
- **cart_stability** introduces an additional penalty for excessive movement.
- **Negative rewards are clipped to zero** to prevent excessive penalties.

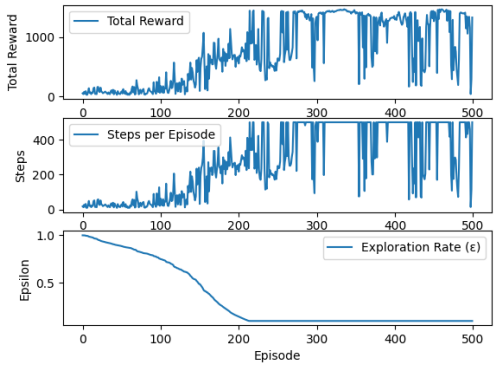

Figure 6: DQN training results without earthquake forces.

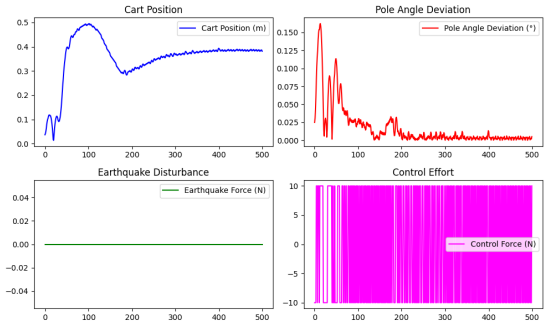

Figure 7: DQN evaluation results without earthquake forces.

2.6 Training and Evaluation With Earthquake Forces

After verifying the model works in a standard environment, it is retrained with an **earthquake force of 15N**.

2.6.1 Hyperparameters for Training With Earthquake Forces (15N)

Parameter	Value
Learning Rate	0.001
Discount Factor (Gamma)	0.99
Replay Buffer Size	100,000
Batch Size	64
Target Network Update Frequency	1000 steps
Epsilon Decay	0.9999
Number of Episodes	15000

Table 4: Hyperparameters for DQN training with earthquake forces (15N).

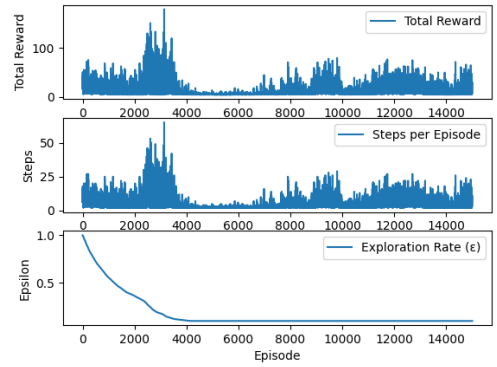

Figure 8: DQN training results with earthquake forces (15N).

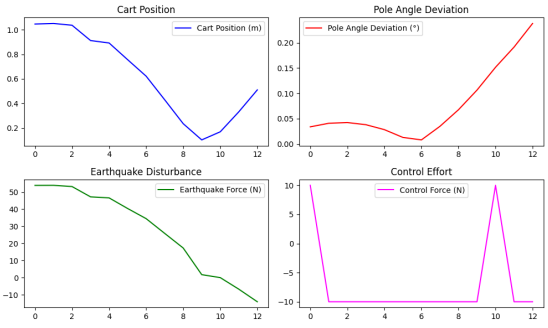

Figure 9: DQN evaluation results with earthquake forces (15N).

2.7 Challenges with Discrete Actions and the Need for Continuous Control

The results indicate that due to the **discrete nature of Gym's CartPole environment**, the model fails to learn an effective stabilization strategy. The CartPole environment in Gym only allows:

- **Discrete actions (0 or 1)** (move left or right).
- **Force application limited to ±10N**, which is insufficient to counteract a 15N earthquake force.

To handle **continuous external forces**, a custom continuous environment was attempted but faced several implementation challenges. Instead, we explore an alternative solution using **Proximal Policy Optimization (PPO) in Isaac Sim**.

2.8 Next Steps: PPO Implementation in Isaac Sim

Since Gym's CartPole environment does not support continuous control natively, we transition to **Isaac Sim's CartPole implementation**, which allows for continuous action space. Using PPO, we aim to:

- Define a **continuous action space** for force application.
- Utilize **Isaac Sim's physics engine** for accurate dynamics.
- Train the model using **policy-based reinforcement learning**.

2.9 Video Demonstration

Video Link: `https://drive.google.com/file/d/11T0wvyiKSeA-mdnwOzyDbZnRdW_gHfer/view?usp=drive_link` The video demonstrates the evaluation of the trained DQN model under two different conditions:

- **Without Earthquake Forces:** The first part of the video shows the cart-pole system being evaluated using a model trained in a standard environment with no external disturbances.
- **With Earthquake Forces (15N):** The second part of the video presents the evaluation of the model after training with an earthquake force of 15N. The results indicate that the training did not converge successfully due to the limitations of the discrete action space in Gym's CartPole environment.

3 Proximal Policy Optimization (PPO) for Cart-Pole with Earthquake Forces

3.1 Introduction

This section explores reinforcement learning using **Proximal Policy Optimization (PPO)** in **Isaac Sim** and **Isaac Lab** to stabilize the cart-pole system under earthquake forces. Unlike the previous DQN implementation, which struggled due to discrete action limitations, PPO supports continuous control, making it better suited for this problem.

3.2 Pipeline Overview

An example provided by Isaac Lab is modified to introduce **earthquake forces**, allowing the model to learn how to maintain equilibrium under external disturbances.

1. Modify the existing **Cart-Pole environment** to apply earthquake forces.
2. Train the model using **PPO**, which optimizes the policy network through a clipped objective function.
3. Evaluate the trained model under different earthquake intensities.
4. Compare performance metrics, including displacement, pole deviation, earthquake forces, and control effort.

3.3 Installation and Execution

3.3.1 Requirements

To train and evaluate the PPO model, Isaac Lab and Isaac Sim must be installed. The custom environment file `c1_env.py` should be placed in the following directory:

```
isaaclab/source/isaaclab_tasks/isaaclab_tasks/direct/cartpole
```

3.3.2 Modifying __init__.py

Modify the `__init__.py` file in the same directory to include the following registration:

```python
from .cartpole_env import C1Env, C1EnvCfg

gym.register(
    id="Isaac-C1-Direct-v0",
    entry_point=f"{__name__}.cartpole_env:C1Env",
    disable_env_checker=True,
    kwargs={
        "env_cfg_entry_point": f"{__name__}.cartpole_env:C1EnvCfg",
        "rl_games_cfg_entry_point": f"{rl_games_ppo_cfg.__name__}:rl_games_ppo_cfg.yaml",
        "rsl_rl_cfg_entry_point": f"{rsl_rl_ppo_cfg.__name__}.rsl_rl_ppo_cfg:CartpolePPORunnerCfg",
        "skrl_cfg_entry_point": f"{skrl_ppo_cfg.__name__}:skrl_ppo_cfg.yaml",
        "sb3_cfg_entry_point": f"{sb3_ppo_cfg.__name__}:sb3_ppo_cfg.yaml",
    },
)
```

3.3.3 Training the PPO Model

To train the model, execute the following command:

```
./isaaclab.sh -p scripts/reinforcement_learning/sb3/train.py
--task Isaac-C1-Direct-v0 --num_envs 64
```

3.3.4 Evaluating the PPO Model

After training, the model can be evaluated using:

```
./isaaclab.sh -p scripts/reinforcement_learning/sb3/play.py
--task Isaac-C1-Direct-v0 --num_envs 64
```

3.4 PPO Model characteristics

The PPO algorithm is trained on the modified Isaac Sim environment, which introduces earthquake forces. The model is trained to minimize pole deviation and maintain stability.

3.4.1 Hyperparameters

Parameter	Value
Learning Rate	0.0003
Discount Factor (Gamma)	0.99
Clip Range	0.2
GAE Lambda	0.95
Number of Environments	64
Total Timesteps	2 million

Table 5: Hyperparameters for PPO training in Isaac Sim.

3.4.2 Reward Function

The reward function encourages the agent to maintain pole stability while penalizing excessive displacement and control effort.

$$R(s,a) = 1.0 - (|\text{pole\_angle}| \times 2.0) - (|\text{cart\_displacement}| \times 1.5) - (\text{control\_effort} \times 0.1)$$

where:

- **1.0** is a base reward for keeping the pole upright.
- **Pole deviation is penalized** using $|\text{pole\_angle}| \times 2.0$.
- **Cart displacement is penalized** using $|\text{cart\_displacement}| \times 1.5$.
- **Control effort is minimized** to encourage energy-efficient stabilization.

3.5 Training and Performance Evaluation

The PPO model is trained, and its performance is evaluated using two key plots. The training process successfully completes after **979 iterations**, at which point the simulation reaches a satisfactory policy.

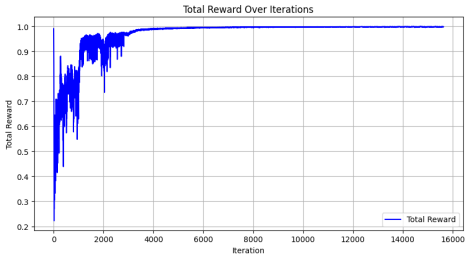

Figure 10: PPO training results showing total rewards over iterations. The training successfully converges after 979 iterations.

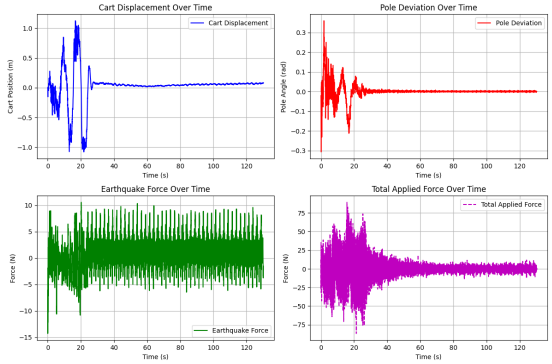

Figure 11: Performance metrics including cart displacement, pole deviation, earthquake forces, and control effort.

3.6 Generalization Beyond Training Conditions

After training the PPO model with an earthquake force of 15N, further tests were conducted by increasing the earthquake magnitude to **50N**. Despite not being trained for such high disturbances, the model performed exceptionally well, demonstrating its ability to generalize beyond its training conditions.

3.7 Video Demonstration

Video Link: `https://drive.google.com/file/d/1mombskk7TNwtgn8APkGX-z-WUXRHNO-1/view?usp=sharing` The video includes:

- **Training process clips**, showing how the model learns over time.
- **Evaluation after training with 15N**, demonstrating successful stabilization.
- **Testing with an increased earthquake magnitude of 50N**, proving the model's generalization capability.

4 Conclusion

In this study, we explored different control strategies for stabilizing a cart-pole system under earthquake-induced disturbances.

4.1 LQR Controller

The **LQR approach** provided stable control by tuning Q and R matrices. The best-performing configuration ($Q = [5, 5, 20, 20]$, $R = 0.05$) resulted in **minimal displacement and longest stability duration (120s)**. However, LQR requires prior knowledge of system dynamics and is **not adaptable** to varying external forces.

4.2 Deep Q-Network (DQN)

DQN was trained in the **Gym CartPole-v1** environment, first without earthquakes and then with a **15N earthquake force**. The model performed well under normal conditions but **failed to converge when subjected to earthquake forces**. However, this limitation was not due to the DQN algorithm itself but rather due to the **constraints of the Gym CartPole environment**, which only allows:

- **Discrete actions (0 or 1)** (move left or right).
- **Force application limited to $\pm$10N**, which is insufficient to counteract a 15N earthquake force.

As a result, the model was unable to generate the required counteracting forces, leading to instability.

4.3 Proximal Policy Optimization (PPO) in Isaac Sim

To overcome the limitations of Gym's discrete action space, we implemented **PPO in Isaac Sim**, which allows for **continuous force application**. The model successfully:

- Stabilized the cart-pole under **15N earthquake disturbances**.
- Generalized to **50N earthquake forces**, despite never being trained for such high magnitudes.

This demonstrates PPO's ability to handle dynamic external disturbances and its advantage over discrete-action-based approaches.