# Robot Learning Homework 2

**Matteo Zulian s310384**
Automation and Intelligent Cyber-Physical Systems
Politecnico di Torino

## 1 Introduction

This exercise deals with the Cart-Pole environment, a classic problem in reinforcement learning. Is asked to implement and evaluate various control strategies, including a Linear Quadratic Regulator (LQR) baseline, random policies and new reward functions, to understand their impact on the system's performance and learn about the fundamental concepts of reinforcement learning.
The Cartpole environment consists of a cart and a pole mounted on top of it and it can move either to the left or to the right. The goal is to balance the pole in a vertical position in order to prevent it from falling down. The cart should also stay within a limited distance from the center (trying to move outside screen boundaries is considered a failure). The observation is a four element vector:

$$states = \begin{bmatrix} x \\ \dot{x} \\ \theta \\ \dot{\theta} \end{bmatrix}$$

where $x$ is the position of the cart, $\dot{x}$ is its velocity, $\theta$ is the angle of the pole w.r.t. the vertical axis, and $\dot{\theta}$ is the angular velocity of the pole.

## 2 LQR

In a Linear Quadratic Regulator (LQR) controller, the objective is to minimize a cost function that represents a trade-off between the system's performance and control effort. The cost function typically takes the form:

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

where $x$ is the state vector, $u$ is the control input, $Q$ is the state cost matrix, and $R$ is the control cost matrix. The control law for an LQR controller is given by:

$$u = -Kx$$

where $K$ is the feedback gain matrix. To find the optimal $K$ matrix, you need to solve the algebraic Riccati equation. The Riccati equation for the LQR problem is given by:

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

Once $P$ is found, the optimal gain matrix $K$ can be calculated as:

$$K = R^{-1}B^T P$$

### 2.1 Results

After a proper design of the controller, the simulation were performed using OpenAI's library 'gym', which is an open-source toolkit that provides a collection of environments for developing and testing reinforcement learning algorithms. this is the result of a simulation of 400 time steps:

7th Conference on Robot Learning (CoRL 2023), Atlanta, USA.

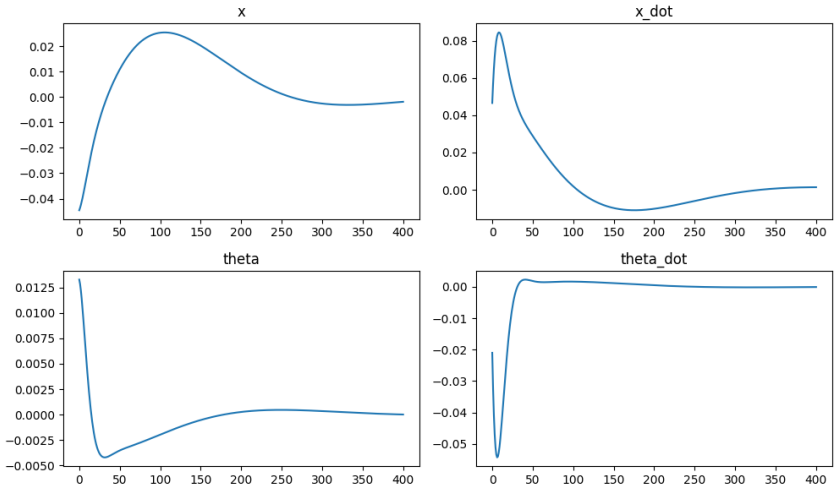


Figure 1: Plot with 400 time steps

### 2.2 Convergence

The convergence time is the time instant when the state gets in the range of $\pm$ 0.05 from 0 and doesn't come out. For the previous solution the convergence time of the states are 0 for $x$, 28 for $\dot{x}$, 0 for $\theta$ and 10 for $\dot{\theta}$.

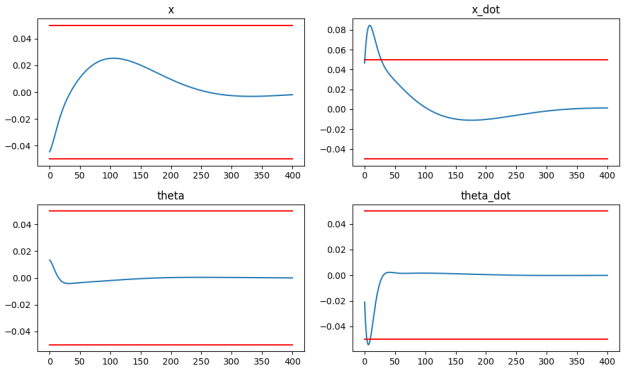


Figure 2: Convergence to $0 \pm 0.05$

From a control standpoint, achieving convergence to zero is a key objective because it means that the control actions applied to the system are effectively maintaining the system in a desired state. The controller is designed to regulate the system in such a way that deviations from the desired state are minimized, and the system settles at a stable equilibrium.

### 2.3 Control Cost Matrix $R$

The matrix $R$ in the cost function $J$ plays a crucial role in influencing the control effort, the cost associated with it is represented by the term $u^T Ru$. When $R$ is large, it means that there is a high penalty for using input control effort, and the controller will try to minimize it to reduce the overall cost, allowing larger state deviations, as shown in figure (3).

On the other hand a lower $R$ allows for more aggressive control but may lead to higher power consumption. It's intuitive that force applied and matrix R are inversely proportional.

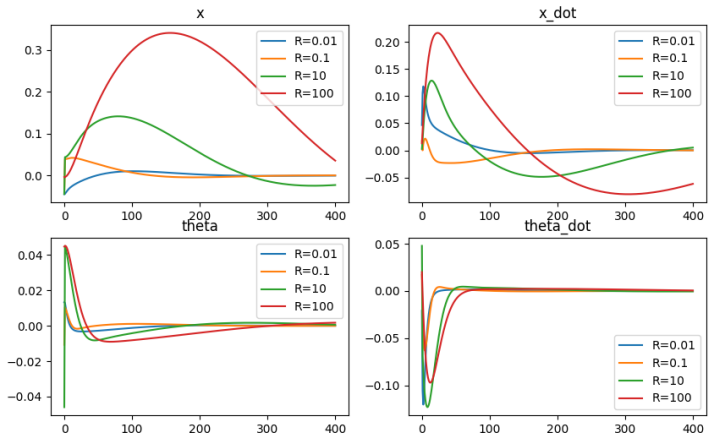


Figure 3: Plot with different R

## 3 Reinforcement Learning

In this section the goal is to train and test the controller in order to balance the pole. The controller learns through an iterative process of trial and error, adjusting its behavior based on the feedback it receives from the environment.
An episode ends after one or more of these conditions are met:

- **Time Limit**:
  The episode ends after a fixed number of time steps or frames. It ensures that episodes have a consistent duration.

- **Angle Threshold**:
  The episode terminates if the pole angle exceeds a certain threshold. For example, if the pole tilts too far from the vertical position, the episode ends.

- **Cart Position Threshold**:
  The episode terminates if the cart moves outside a specified position range. For instance, if the cart moves too far to the left or right, the episode may end.

This means that not every episode ends at the same time step.

### 3.1 Initial Result

In the beginning the reward function returns 1 each time the cart doesn't satisfy the previous conditions, giving as result the plot in figure (4) and an average test result of 500.0 out of an episode length of 500.0. Showing that the algorithm learned correctly how to balance the pole during the train phase.

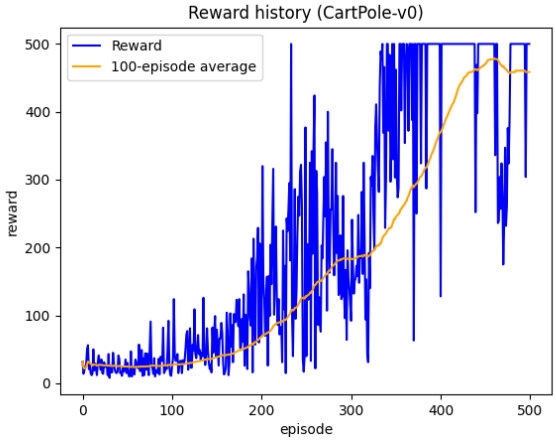


Figure 4

### 3.2 Random Policy

In reinforcement learning, a policy is a strategy or a mapping from states to actions that an agent uses to interact with its environment. A random policy, as the name suggests, is a policy where the agent selects its actions randomly, without taking into account the current state or any learned information.

In the context of reinforcement learning, the agent's goal is typically to learn an optimal policy that maximizes some notion of cumulative reward over time. A random policy is often used as a baseline or for exploration purposes in the early stages of learning. It helps the agent explore the environment and collect data that can be used to estimate the values of different states and actions.

However, relying on a purely random policy is generally not effective for achieving optimal performance in complex environments. As the agent learns more about the environment through experience, it typically updates its policy based on the observed rewards and transitions, moving away from randomness towards more informed decision-making.

In summary, a random policy in reinforcement learning is a policy where actions are chosen randomly, and it is often used for exploration purposes in the early stages of learning before the agent has acquired enough information to make more informed decisions.

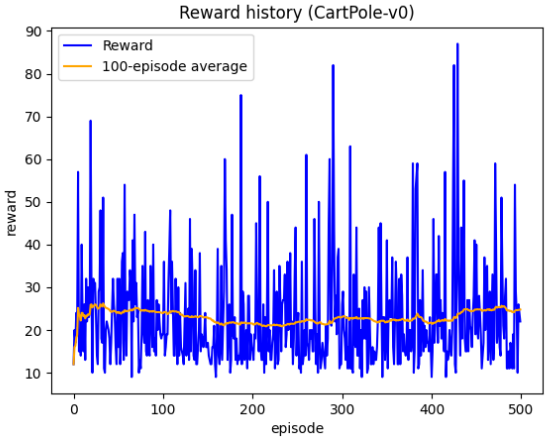


Figure 5

In figure (5) there is the result of a random policy which is clearly a worst solution than the previous one because the controller is stuck in the exploration phase and never starts the exploitation phase, in other words it doesn't improve. The average test reward is only 98.83 with a maximum possible episode length of 500.

### 3.3 Test the model for a longer period than Training

The model is trained with 200 time steps per episode and then tested with 500 time steps. Observing the model's behavior during the extended evaluation can provide insights into how well it generalizes to longer time horizons and it might reveal whether the model can adapt to situations not explicitly encountered during training. However the system doesn't change from 200 to 500 time steps so the algorithm managed to successfully complete the tests.

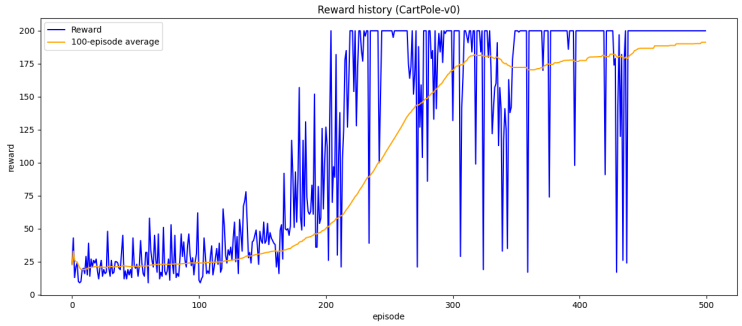


Figure 6

### 3.4 Repeatability

Stochasticity in reinforcement learning (RL) environments refers to randomness or variability in the outcomes of actions taken by the agent. These variations can arise from factors such as random initial conditions, stochastic dynamics, or random external events. Different runs of the same algorithm on the same task can lead to different outcomes due to the randomness in the environment. When comparing RL algorithms, it is often recommended to report results averaged over multiple runs. Algorithms that perform well on average and exhibit consistent behavior across different runs in the face of stochasticity are often considered more robust.

### 3.5 Reward Functions

The reward function can be exploited in order to obtain other behaviours, here are some examples:

- **Balance the cart in an arbitrary point** $x_0$
  the reward is given by the exponential decay function around $x_0$ : $e^{-\alpha|x-x_0|}$, where $x$ is the position of the cart and $\alpha$ tells the steepness of the curve. The idea behind is to reward the cart if it's near enough to $x_0$. If the objective is to balance the cart in the center of the screen, then $x_0 = 0$.

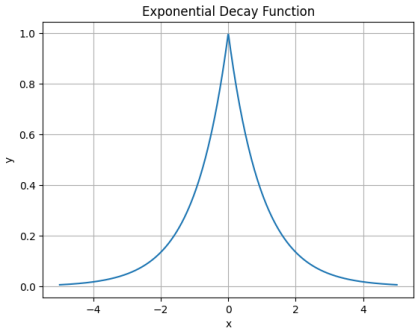


Figure 7

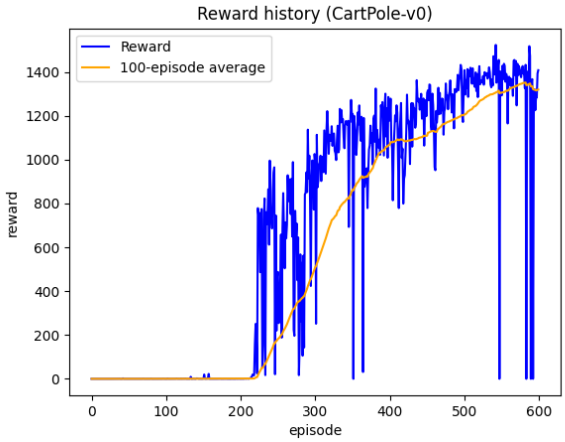


Figure 8

- **Move the cart side to side**
  This reward is more complicated than the previous one, the idea is to have a weighted reward for each state in order to impose the desired behaviour. The different rewards are: $4|\dot{x}|$, $\frac{1}{2(|\theta|+1)}$, $\frac{2}{(|\dot{\theta}|+1)}$, while for the position $x$ is important to know the current orientation of the movement, so is introduced a variable 'side' that tell the reward function whether the cart is moving left or right and rewards with weight 8.

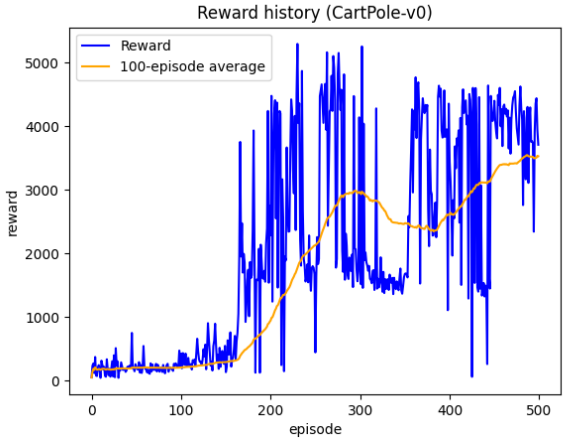


Figure 9

## 4 Extra

LQR controllers and RL agents are two distinct approaches to control systems each with its own advantages and characteristics.

LQR controllers have an analytical solution, making them computationally efficient and providing a clear understanding of the control law. When the system dynamics are accurately modeled as linear and the cost function is quadratic, they can provide optimal control solutions and they provide stability guarantees. On the other hand LQR controllers are specifically designed for linear systems. When dealing with nonlinear systems, they may require linearization around operating points, and their performance may degrade if the system deviates significantly from linearity.

RL agents can learn control policies directly from interactions with the environment without requiring an explicit model of the system dynamics, they can also adapt to nonlinear and complex systems. This characteristic makes theme more versatile than the LRQ, but one of the disadvantages is that RL methods often require a significant number of samples to learn effective control policies. Also RL methods generally do not provide explicit stability or optimality guarantees because the learned policies are data-driven and may not always conform to desired specifications.

In summary, the choice between an LQR controller and an RL agent depends on the characteristics of the robotic system and the control task at hand. LQR controllers are suitable for well-modeled linear systems with known dynamics, while RL agents are more versatile and adaptable to complex, nonlinear, and poorly understood systems. The trade-offs involve factors such as computational efficiency, stability guarantees, and the ability to handle uncertainties in system dynamics. The selection between LQR and RL is often made based on the specific challenges and requirements of the robotic control application.