Robot Learning Lab 2 - LQR and RL

Borella Simone (S317774)
Computer engineering - Artificial Intelligence
Polytechinc University of Turin, Italy
s317774@studenti.polito.it

1 Abstarct

The primary objective of this laboratory session is to cultivate familiarity with Reinforcement Learning (RL). This will be achieved through the exploration of a standard problem in control theory, the cartpole system. The comparative analysis between the RL controller and the Linear Quadratic Regulator (LQR) controller will be a central focus, elucidating the distinctive attributes.

2 Cartpole system

2.1 System description

The cartpole system is a classic problem in control theory. It consists of a cart that can move along a frictionless track, and a pole that is attached to the cart by a hinge. The goal is to balance the pole upright by applying horizontal forces to the cart.

The state of the system is typically described by its position (x), velocity ($\dot{x}$), angle (θ) and angular velocity ($\dot{\theta}$):

$$x = \begin{bmatrix} x & \dot{x} & \theta & \dot{\theta} \end{bmatrix}^T \tag{1}$$

The original system is non linear and the linearized state-space representation for the system can be written as follows:

$$\dot{x} = Ax + Bu \tag{2}$$

$$u = [F] \tag{3}$$

$$\mathbf{A} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & \frac{-g}{l(\frac{m_p}{m_c+m_p}-\frac{4}{3})} & 0 \end{bmatrix} \quad \text{and} \quad \mathbf{B} = \begin{bmatrix} 0 \\ \frac{1}{m_c+m_p} \\ 0 \\ \frac{1}{l(m_c+m_p)(\frac{m_p}{m_c+m_p}-\frac{4}{3})} \end{bmatrix} \tag{4}$$

Where:

- m_c = mass of the cart
- m_p = mass of the pole
- l = length of the pole
- g = acceleration due to gravity

2.2 Gym environment

To simulate the environment, we leveraged OpenAI's `gym` library, an open-source toolkit providing a range of environments for developing and testing reinforcement learning algorithms. This toolkit serves the dual purpose of facilitating the development and experimentation of the RL agent and enabling a comparison with the LQR controller.

The CartPole environment in `gym` is completely observable. In terms of the action space, the controller offers two discrete actions: pushing the cart either to the right or to the left.

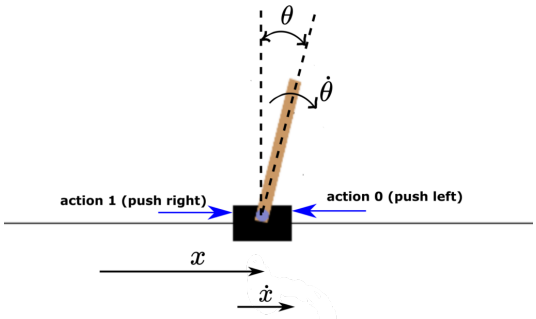

Figure 1: Gym environment for cartpole system

3 Linear Quadratic Regulator

3.1 Introduction

The Linear Quadratic Regulator (LQR) stands as an optimal control algorithm specifically crafted for linear, time-invariant systems. The primary objective of the LQR controller is to determine the optimal control input that minimizes a cost function. This cost function generally captures the balance between system performance and the exertion of control effort.

The function to be minimised is the following:

$$J = \int_0^\infty \left(x^T Q x + u^T R u\right) dt \tag{5}$$

Where Q is the state cost matrix, and R is the control cost matrix. The control law for an LQR controller is given by:

$$u = -Kx \tag{6}$$

where K is the feedback gain matrix. The optimal K matrix is the solution of the algebraic Riccati equation, which is given, in this specific problem, by:

$$A^T P + PA - PBR^{-1}B^T P + Q = 0 \tag{7}$$

The solution to the Riccati equation ensures that the closed-loop system is stable, and the quadratic cost function J is minimized. Once P is found, the optimal gain matrix K can be calculated as:

$$K = R^{-1}B^T P \tag{8}$$

3.2 Convergence

Simulating the system over 400 timesteps with weighting matrices $Q = 5 \cdot I_{4,4}$ and $R = 1 \cdot I_{1,1}$ (where $I_{n,n}$ denotes the identity matrix of size $[n, n]$) yields the summarized results illustrated in Figure 2.

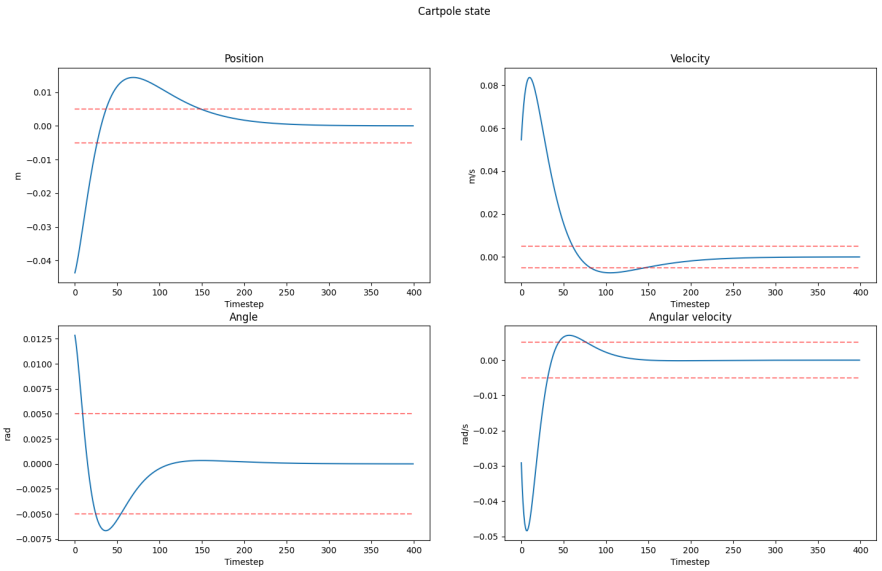

Figure 2: Evolution of the cartpole system state over the 400 timesteps with LQR controller

Achieving convergence to zero is a primary goal from a control perspective, meaning that the applied control actions effectively maintain the system in a desired state. The controller is designed to regulate the system, minimizing deviations from the desired state and facilitating the system's settlement at a stable equilibrium.

The convergence of the state is reached at timestep 150, taking as convergence range the interval $[-0.005, 0.005]$ for each dimension of the state, as illustrated in Figure 2.

As previously discussed, the controller's behavior is linked to the design of the state cost matrix Q and the control cost matrix R. Specifically, Q determines the degree to which we penalize deviations of the current state from the desired state, while R determines the penalty for non-zero control inputs. Maintaining Q constant and varying the values of R allows us to observe distinct behaviors. Larger R values result in increased penalties for controller actions, consequently slowing down the convergence process, whereas smaller R values expedite convergence. This observation is evident in Figure 3, where multiple R values are tested: $R_{\text{values}} = [0.01, 0.1, 10, 100]$. The behaviour of the controller for each R value can be observed by looking at the reaction of the force applied to the system, as illustrated in Figure 4.

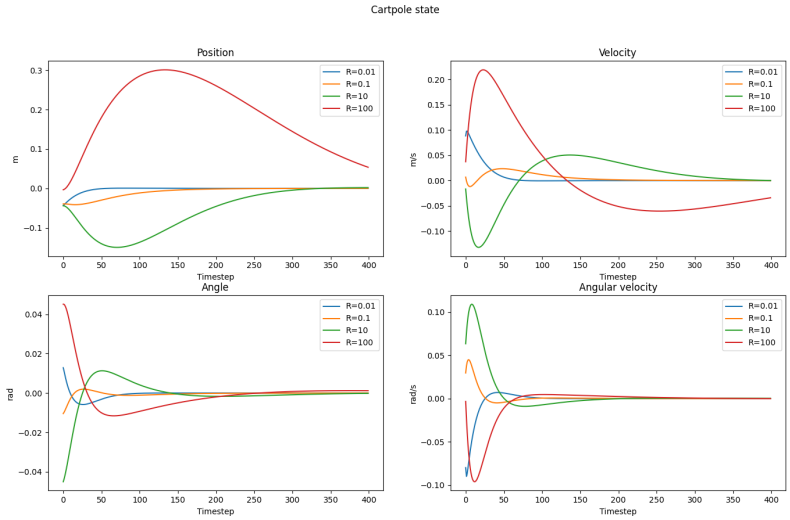

Figure 3: Evolution of the cartpole system state over the 400 timesteps with LQR controller, for different R values

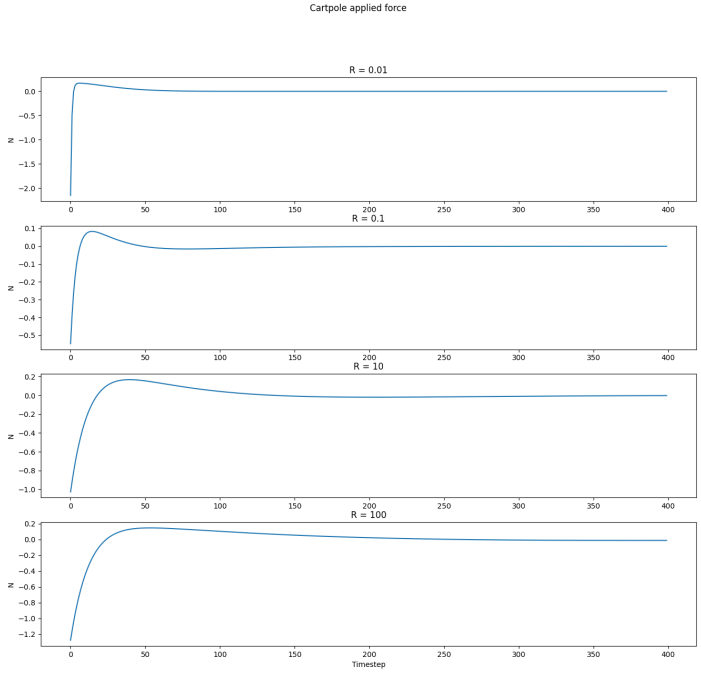

Figure 4: Force applied to cartpole system by LQR controller, for different R values

4 Reinforcement Learning controller

4.1 Introduction

Reinforcement learning (RL) is a type of machine learning paradigm where an agent learns to make decisions by interacting with an environment. The agent takes actions in the environment and receives feedback in the form of rewards or punishments. The goal of the agent is to learn a strategy, or policy, that maximizes the cumulative reward over time.

4.2 Cartpole environment

In the gym cartpole environment, the episode can end in a few different manners:

- 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 ends if the angle of the pole exceeds $\pm 0.21 rad$
- Position threshold: The episode ends if the x position exceeds ± 2.4 (when the cartpole gets of out of the rendered environment)

4.2.1 Base reward function

Figure 5 illustrates the training process with the base reward function provided by the gym cartpole environment. The reward results to be 1 if the cartpole state doesn't exceed limits and 0 otherwise. The total reward for an episode is the number of timesteps the carpole survive.

This way the RL agent will learn to maximise the reward function just simply by surviving the maximum amount of time in the simulation.

The trained model is saved as *model_params_base_500.ai*
Test results: Average test reward: 500.0 episode length: 500.0

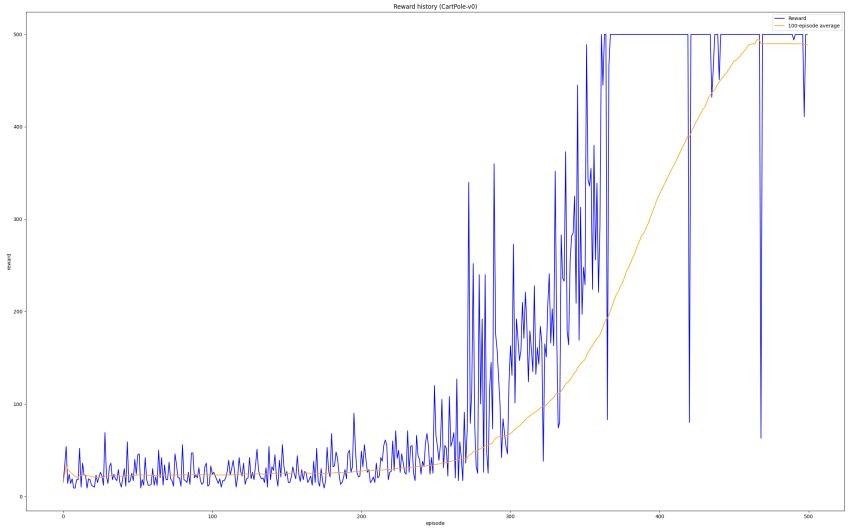

Figure 5: Training process with base reward function over 500 episodes

4.2.2 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 reinforcement learning, the agent's primary objective is to acquire an optimal policy that maximizes cumulative rewards over time. To establish a baseline or for early exploration purposes, a random policy is frequently employed. This random policy aids the agent in exploring the environment and gathering data crucial for estimating the values associated with different states and actions.

However, depending solely on a random policy is generally insufficient for achieving optimal performance in intricate environments. As the agent gains a deeper understanding of the environment through accumulated experience, it tends to refine its policy based on observed rewards and transitions. This evolution involves a shift from randomness towards more informed decision-making, marking a crucial aspect of the learning process.

The trained model is saved as *model_params_random.ai*
Test results: Average test reward: 126.2 episode length: 126.2

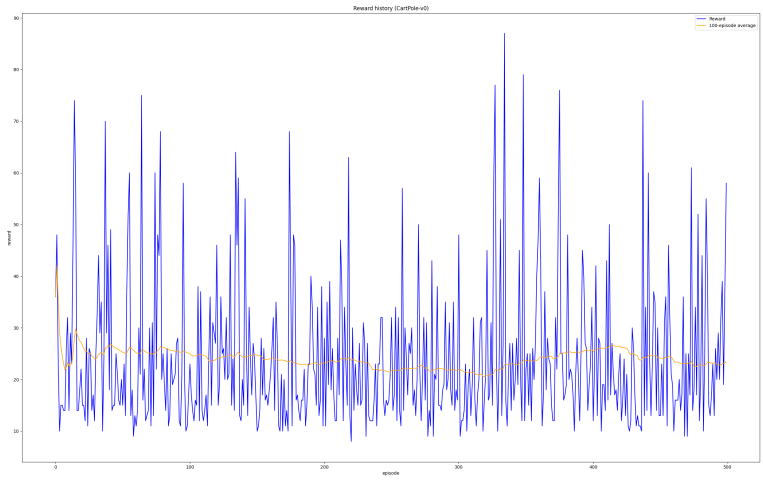

Figure 6: Training process with random policy

4.2.3 Base reward function - longer testing episode length

In this trial a model is trained with the base reward function and episode length equal to 200, and then tested with episode duration 500. 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.

The trained model is saved as *model_params_base_200.ai*
Test results: Average test reward: 500.0 episode length: 500.0

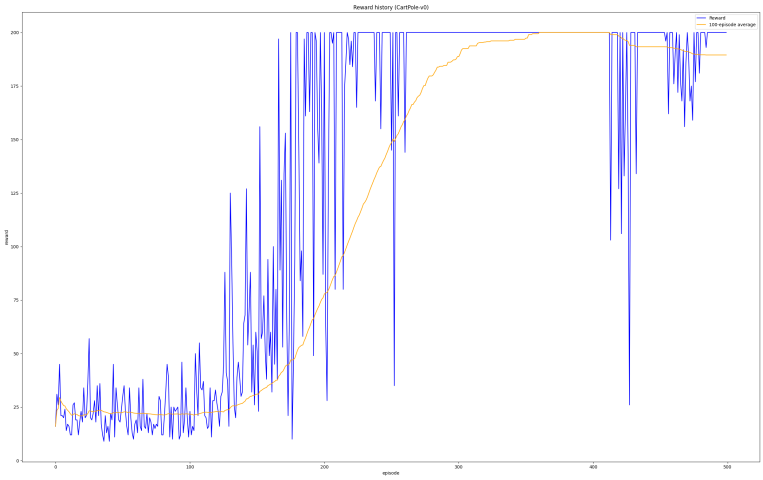

Figure 7: Training process with base reward function over 200 episodes

4.3 Repeatability

Trying to run the same training process for multiple time different behaviours can be observed.

The reason is the stochasticity of the exploration of the environment. The learning process firstly evolves randomly, because the agent doesn't know which way is positively rewarded yet. This exploration, in multiple runs, could be faster or slower based on how fast a new optimal state is explored for the first time. In more complex environments it can also happen that the agent falls into a local optimum before it ever experiences the reward leading him in the global optimum.

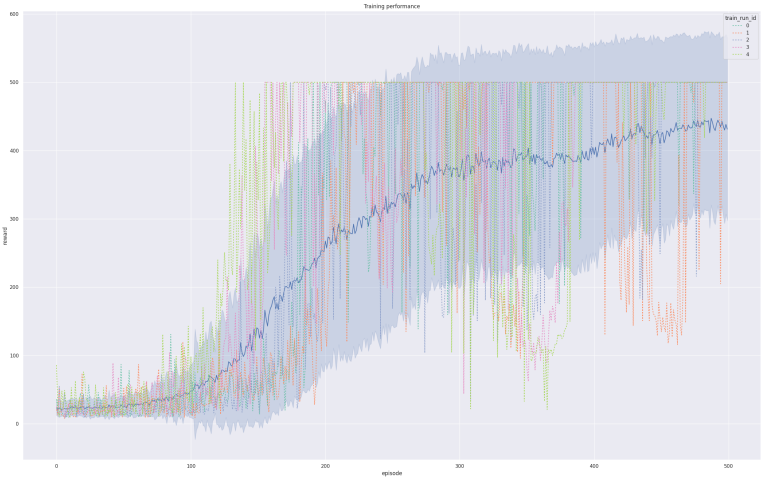

Figure 8: Multiple training process with base reward function

4.3.1 Reward function - arbitrary x0 position

To make the cartpole reach an arbitrary poistion x0 a reward function is designed to have its maximum when the position of the cart is exacly x0.

The function described below, and shown in Figure 9, is used as base function to evaluate reward factors.

$$r(x) = \frac{1}{|x| - 1} \tag{9}$$

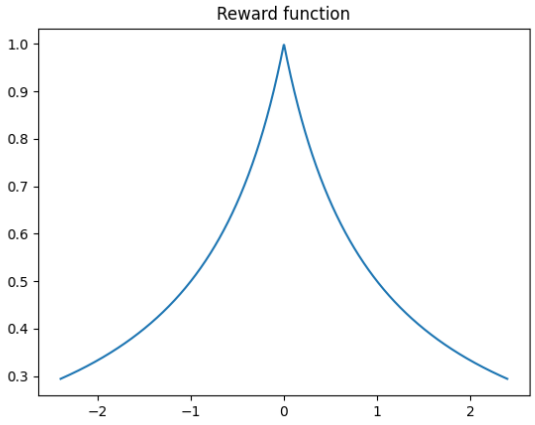

Figure 9: Reward factor function

The reward function is then computed giving more reward the smaller the distance between the current position (x) and $x0$:

$$r(x, v, \theta, \omega) = \frac{1}{|x - x0| - 1} \tag{10}$$

Training and testing (x0 = 0)

The trained model is saved as *model_params_x0_0.ai*
Test results: Average test reward: 481.6 episode length: 500.0

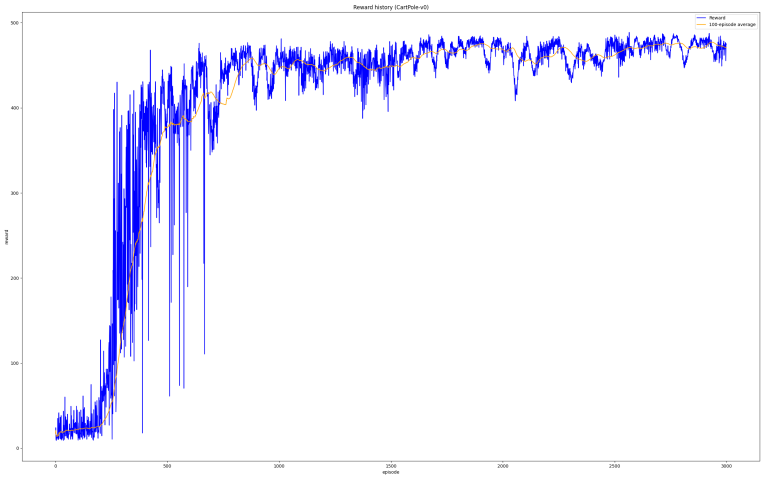

Figure 10: Training process with x0 = 0

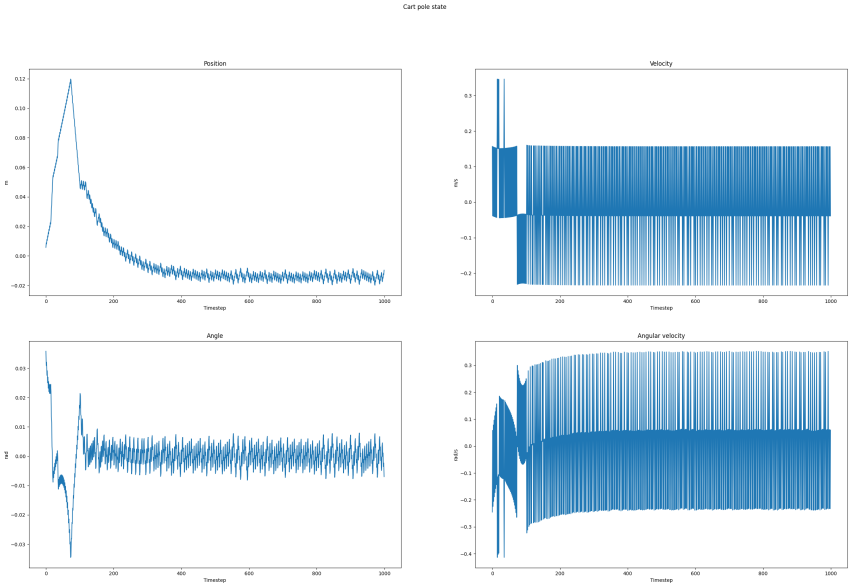

Figure 11: Test process with x0 = 0

Training and testing (x0 = 1.5)

The trained model is saved as *model_params_x0_1_5.ai*
Test results: Average test reward: 229.47 episode length: 500.0

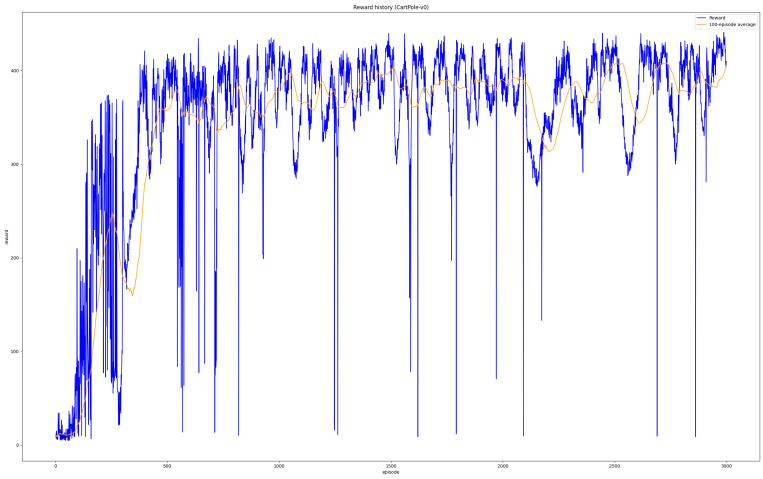

Figure 12: Training process with x0 = 1.5

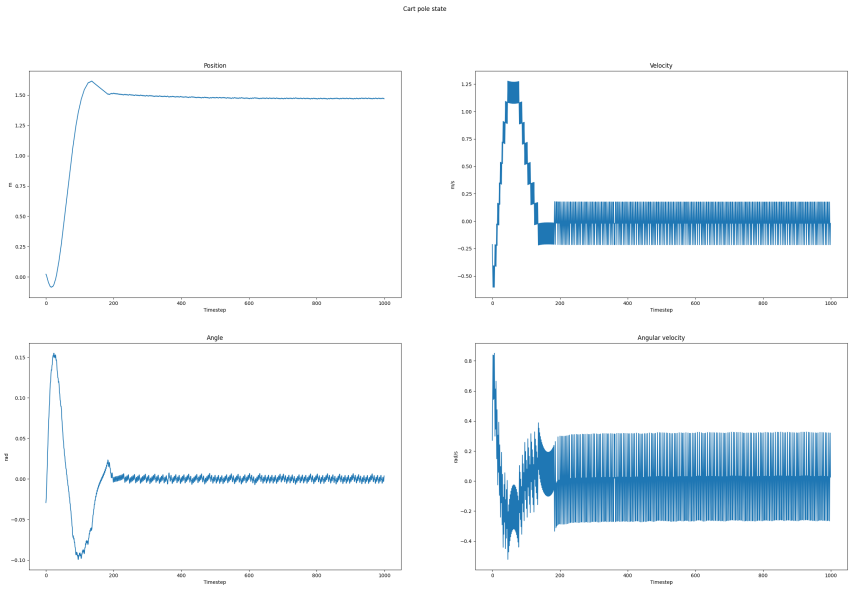

Figure 13: Test process with x0 = 1.5

4.3.2 Move the cartpole from the leftmost to rightmost side of the track and back

This particular task is harder than the other ones. The main issue with the possible solutions is that the agent has no indication of the time: the same state can be encountered both going in the right direction or going in the left direction. One solution to the problem is to force a sub-set of states to be encountered on the way to the right side and the remaining sub-set of states to be encountered on the way to the left side.

The reward function is composed by 4 factor, one for each state dimension. The factor involved (sorted by influence) are:

- **Direction factor:** the functions want to encourage moving to the right direction (dir_term), which is switched when the position of the cartpole reach the values ± 1.9.
- **Velocity factor:** the velocity should be the higher possible.
- **Angular velocity factor:** the angular velocity should the closest possible to 0 rad/s (this reward factor makes the system more stable).
- **Angle factor:** the angle should the closest possible to 0 rad (this reward factor makes the system more stable).

The reward function adopted for this kind of solution is the following:

$$direction = \{-1, +1\}$$

$$dir\_factor = 1 \quad \textit{if } sign(\frac{dir}{x}) > 0 \textit{ else} \quad 0 \longrightarrow \textit{right direction}$$

$$vel\_factor = |v| \longrightarrow \textit{maximum velocity}$$

$$angular\_vel\_factor = \frac{1}{|\omega| + 1} \longrightarrow \textit{minimum angular velocity}$$

$$angle\_factor = \frac{1}{|\theta| + 1} \longrightarrow \textit{minimum angle}$$

$$R_t = 8 \cdot dir\_factor + 4 \cdot vel\_factor + 2 \cdot angular\_vel\_factor + 0.5 \cdot angle\_factor$$

Multiplying the more important factors with higher values helps the training process to ensure first the correct direction, then the maximum velocity and than the stability given by the angular velocity and angular factors.

The training is processed with 10000 episodes with episode length equal to 3000 (Figure 14).

The trained model is saved as *model_params_right_left.ai*
Test results: Average test reward: 5512.87 episode length: 500.0

As a result the control behave correctly going from the left to the right and vice-versa reaching a maximum velocity of $2.2 m/s$.

Figure 15 illustrates more details about the result.

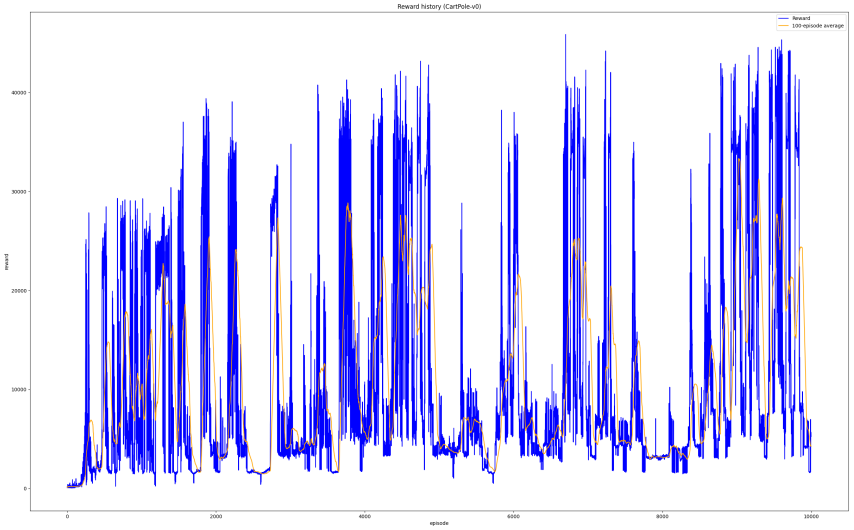

Figure 14: Training process

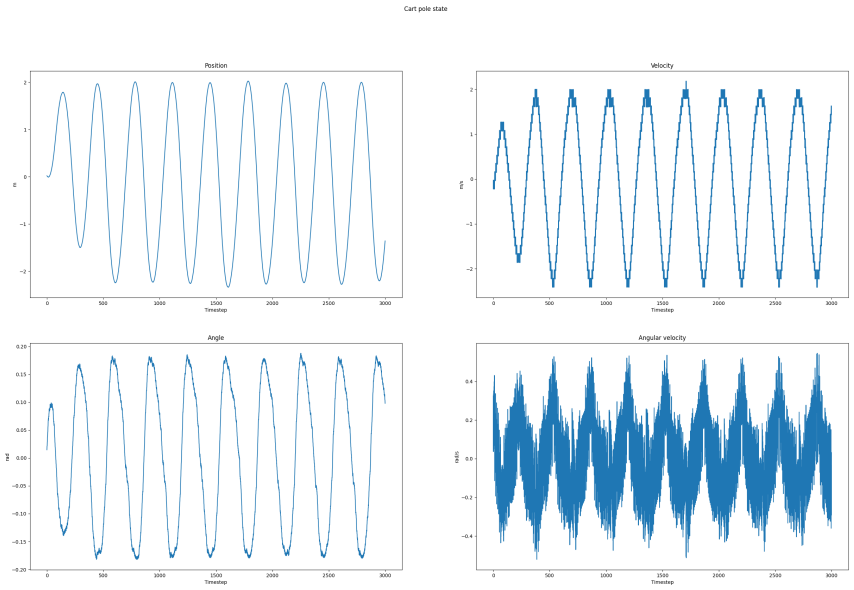

Figure 15: Testing process

4.3.3 Results

As a result here are described some aspect regarding the differences between LQR and RL control strategies:

- **Model dependency**: RL agents are usually model-free and do not rely on models of the system. They learn from interacting with the enviroment learning the best policy in order to maximize the reward function given by the environment. LQR requires a known linear model and computes a controller based on that model.
- **Adaptability**: The RL agent can adapt to unknown and complex non linear systems, while the LQR controller is specifically designed for linear systems and a using a linearization around a working point may cause a performance degrade if the system deviates significantly from linearity.
- **Learning and performances**: The RL agent has to interact with the environment and learn from the outcomes. This may be a long phase and optimality is not guaranteed. The LQR, instead, provides an analytical solution, making them computationally efficient and providing a clear understanding of the control law.

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.