Robot Learning A.Y. 24/25 - Lab 1

Mattia Sabato
DAUIN, Dipartimento di Automatica e Informatica
Politecnico di Torino, Italy
mattia.sabato@studenti.polito.it

Abstract: The following paper reports a comparison of approaches based on a Linear Quadratic Regulator (LQR) and Reinforcement Learning (RL) for controlling an agent in the Cart-Pole environment. Several experiments are run to validate congruency between the theory and the practice, allowing for the exploration of some alternatives which may promote or discourage certain behaviors, thus providing insights into the strengths and limitations of each approach.

1 Introduction

The Cart-Pole environment is a well-known benchmark in RL [1], originally introduced in [2] and currently implemented in the Gym framework [3]. The setup involves a cart that can move horizontally with a pole attached to it, free to rotate around its pivot point. The system's state is defined by four variables: the cart's position and velocity, as well as the pole's angle and angular velocity. The agent controlling the cart has only two possible actions: applying a force to move the cart either left or right. Many physical dynamics such as accelerations, inertia, and forces are thus hidden, and must be inferred through interaction, often modeled using a Neural Network (NN) as a function approximator. Each episode begins with the cart positioned near the center of the track and concludes when either the cart moves beyond a predefined boundary or the pole's angle exceeds a critical threshold. For more specific details, we refer to the Gym documentation. This report explores various approaches to solving the Cart-Pole balancing problem. Starting with a baseline method using an LQR [4], we progress to RL-based techniques that autonomously learn optimal behaviors, or policies, designed to maximize cumulative rewards defined by user-specified criteria.

2 LQR applied to the Cartpole environment

An LQR is a method within the framework of optimal control which involves designing a controller K that selects actions to minimize a user-defined cost function. This cost is formulated using two matrices: Q, which penalizes deviations in the system's state, and R, which penalizes the magnitude of the control inputs. By adjusting the values in Q and R, the user can influence the behavior of the controller K, resulting in different control strategies and dynamics.

When applying an LQR to the Cart-Pole environment, it is important to account for the fact that the system is inherently nonlinear. The first step, therefore, is to linearize the system. While this linearization is only an approximation, it generally produces satisfactory results for small deviations from the equilibrium. The control problem then reduces to minimizing a quadratic cost function of the form:

$$J(\mathbf{x}, \mathbf{u}; Q, R) = \int_0^\infty (\mathbf{x}^T Q \mathbf{x} + \mathbf{u}^T R \mathbf{u}) dt \tag{1}$$

where $\mathbf{x}$ represents the state vector (i.e., cart position, cart velocity, pole angle, and angular velocity), and $\mathbf{u}$ represents the control input (i.e., the force applied on the cart).

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

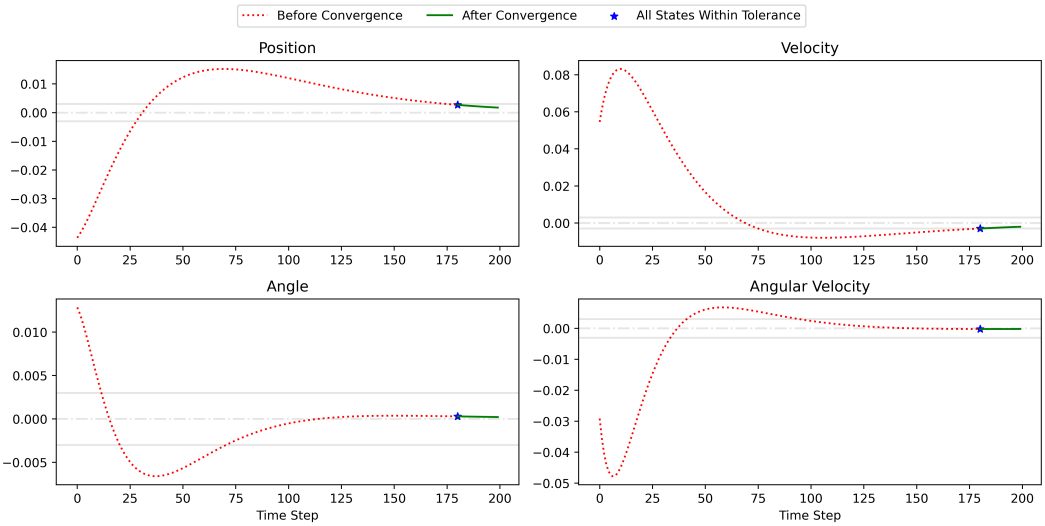

Figure 1: Behavior of the state variables over time under the LQR controller with $\rho = 1$. The gray lines indicate the range within which the values are considered to have converged.

We run several experiments on the Cart-Pole environment by setting $Q = 5I, Q \in \mathbb{R}^{4\times4}, R = \rho I, I \in \mathbb{R}^{1\times1}$. When fixing $\rho = 1$, the system converges efficiently to a stable configuration, as shown in Figure 1. All components of the state vector approach zero, demonstrating successful stabilization. In this configuration, it is observed that the system requires only 180 timesteps for all state variables to converge within the range $[-0.003, 0.003]$. The convergence to the null vector is expected from a control standpoint, in which the controller aims to reach a stable configuration in which all states are equal to zero, i.e. an equilibrium point.

It is important to highlight that the results depend significantly on the magnitude of the values in the R matrix, both quantitatively and qualitatively. As discussed earlier when introducing the LQR and as evident from Equation (1), higher values in R impose a stronger penalty on the control input. This relationship leads to smaller input magnitudes as R increases. To validate this expected relationship, we repeated the experiment several times by considering $\rho \in \{0.01, 0.1, 1, 10, 100\}$. The results are depicted in Figure 2, where some curves have been inverted for clarity to better illustrate the trend. As expected, the experimental results confirm that the magnitude of R and the input values, specifically the forces applied, are inversely proportional. This alignment between theory and observed behavior demonstrates the consistency of the LQR framework in controlling the system.

3 Reinforcement Learning

RL is a powerful branch of Artificial Intelligence where an agent operates within an environment, often unmodeled, to maximize a cumulative reward. The underlying structure of the environment is typically defined by a Markov Decision Process (MDP), although its dynamics is usually unknown to the agent. This lack of prior knowledge necessitates exploration of the state space to discover the optimal behavior. The main goal in RL is to derive a policy $\pi(s) = \mathbb{P}(a|s)$ that represents the probability of selecting action a when in state s. In the context of the Cart-Pole environment, the action space is discrete, with $a \in \{\text{go left}, \text{go right}\}$. However, the state space is continuous and is characterized by variables such as cart position, velocity, pole angle, and angular velocity. To handle this complexity and leverage its strong generalization capabilities, a NN parameterized by w is used to approximate the policy. The result is denoted as π_w, allowing the agent to make decisions that effectively approximate the optimal policy in such a dynamic system.

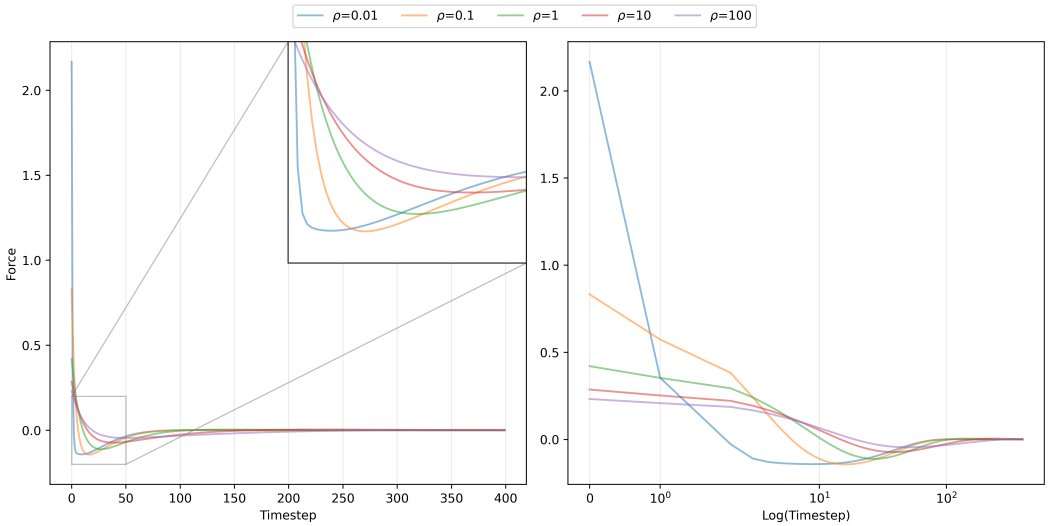

Figure 2: Force trends observed for different values of R, with the effects most noticeable during the initial transient. The empirical results confirm the theoretically predicted inverse proportionality.

3.1 Baseline and training issues

As an initial baseline, we tested the agent's performance using a simple reward function: $+1$ for each timestep the pole remains balanced and 0 otherwise. This approach aligns with the standard setup in the Gym environment and is commonly used to encourage the agent to maximize the duration for which the pole remains upright. Each episode ends at 500 timesteps, although it can end earlier if $|\theta| > 12°$ (pole angle exceeds a threshold) or $|x| > 2.4$ (cart position goes out of bounds). Early in training, episodes often terminate prematurely as the agent initially struggles to determine optimal actions. Over time, as the agent gathers experience, the length of the episode is expected to gradually approach the maximum of 500 timesteps. This improvement reflects the NN-based policy π_w learning the appropriate latent representations and optimal actions through trial and error. Initially, the agent's behavior is nearly random, but this transient phase should end quickly as training progresses and the policy becomes more refined. To contextualize the agent's performance, we compared it against a random policy, π_u. The results, presented in Figure 3, demonstrate that the random policy leads to significantly worse performance, not being able to ever increase the reward, providing further empirical evidence of the NN's added value. Unlike the random policy, the NN-based one not only behaves more intelligently, but also optimally adapts to the task, achieving far superior results. This reflects also when testing them after the training, where, on average, π_w easily reach the maximum reward of 500 and π_u does not actually go above 110. Due to the complex nonlinear nature, the problem of balancing the pole can not be solved through random actions, as expected.

A natural question arises: when should training stop, and how many timesteps are sufficient to learn a generalizable policy π_w? Unfortunately, there is no universal answer, as the required numbers depend on factors such as the task complexity and the state space's dimensionality. However, in our relatively simple Cart-Pole environment, we can analyze how performance evolves as the maximum number of training timesteps varies. Figure 3 shows the average reward and standard deviation across multiple runs, each trained with a different maximum timestep limit. Performance stabilizes quickly in shorter training ones, where the model tends to overfit on the limited states it encounters within the restricted time. Interestingly, stabilization generally occurs after a number of episodes proportional to the maximum timestep limit. This trend is motivated by the increased exploration afforded by longer episodes, which enables the agent to see (and thus, learn) a broader range of states. While this extended exploration requires more time to converge, it also facilitates better generalization when the policy is tested. Table 1 provides an analysis of how varying the number of

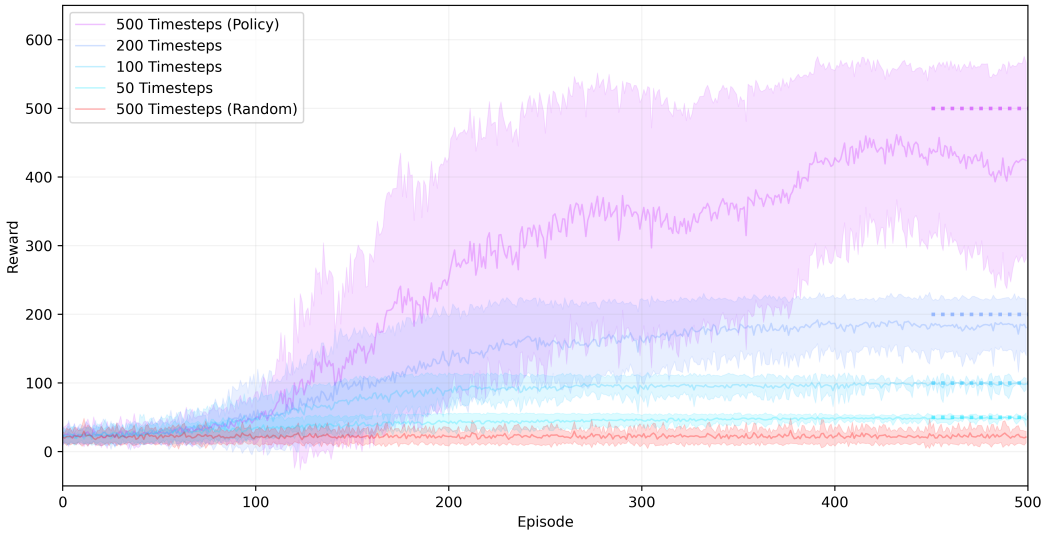

Figure 3: Average total reward and variance computed over 30 independent runs. The contrasting behaviors highlight the wealth of knowledge and information exploited by π_w, which improves progressively with each episode, compared to the lack of intelligent or optimal behavior in π_u. Dotted lines indicate the maximum achievable reward for each timestep limit. While all curves eventually converge to it, a greater number of episodes may be required for settings with an higher limit.

training episodes impacts the performance. All experiments were conducted with a fixed limit of 200 timesteps per episode, training three agents for five runs and varying the numbers of episodes. For each agent, the mean and standard deviation were calculated, and an overall average was computed for each training episode count. The results indicate that, as expected, increasing the number of training episodes improves generalization. Notably, this appears to be at least as significant as the effect of increasing the number of timesteps. For instance, when trained for 500 episodes, two out of three agents demonstrate a remarkably good generalization during testing, even with a timesteps limit far exceeding that encountered during training. This outcome is quite interesting, despite some observed variability in qualitative behavior, which may seem less stable or reliable than its counterpart trained on 500 timesteps. This is motivated by a lack of convergence which results in suboptimal and different behaviors during testing. This does not mean that the agent is not able to eventually reach an high reward, but it achieves so by exploiting each time policies which are drastically different from each other and that could have eventually converged to an optimal one by providing more time and data.

When comparing the reward trends across different runs of the training, a notable issue is the non-deterministic nature of the results. While the overall performance may converge after many episodes, variability between runs remains. This can be attributed to several factors, one of the most significant being the inherent stochasticity in RL algorithms, particularly when employing stochastic policies. The lasts introduce randomness in action selection, which encourages exploration but also leads to suboptimal choices at times. This randomness is crucial for effective exploration and ultimately aids convergence, as it helps the agent discover a broader range of state-action pairs. However, it also means that each training run is unique and not easily reproducible unless controlled via a fixed random seed. This variability is further amplified when using Monte Carlo (MC) estimation techniques, which are characterized by higher variance compared to other methods such as Temporal Difference (TD) learning or its n-step variants. MC methods rely on complete episode rollouts to estimate returns, and their variance arises from the dependence on full trajectories and the inherent uncertainty in the environment's dynamics. In contrast, TD methods update estimates incrementally, reducing variance at the cost of potentially introducing bias.

#training_episodes	AgentID	Average Reward	Average Across Agents
200	1	413.09 ± 06.68	364.49
	2	496.99 ± 00.93	
	3	210.39 ± 02.19	
300	1	411.09 ± 10.99	374.13
	2	211.31 ± 01.24	
	3	500.00 ± 00.00	
500	1	489.80 ± 02.93	418.99
	2	500.00 ± 00.00	
	3	267.18 ± 01.90	

Table 1: Average reward computed with varying numbers of training episodes, each limited to 200 timesteps, and evaluated over 500. Reducing the number of training episodes significantly impacts the performance when testing compared to simply limiting the timesteps. Still the agent is able to achieve a quite satisfactory reward even with as few as 200 episodes.

3.2 Reward function variants

The behavior of an agent, defined by its learned policy, is heavily influenced by the reward function underlying the environment. Since the agent's goal is to maximize the cumulative reward, it must act in a manner tailored to the specific reward structure, even without having explicit knowledge of the reward function itself. To study how different reward formulations affect the agent's behavior, we experimented with the following two reward functions:

1. **Biased**: We force the agent to be as close as possible to the states for which $x \approx C$, where $C \in (-2.4, 2.4)$ is a given constant. To get the desired behaviour, we used the following formula:
$$R_{t+1}(x_t; C) = 1 + |C| - |x_t - C| + \mathbb{1}_{|x_t - C| \leq 0.05} * 5 \tag{2}$$
2. **Slide**: The agent is supposed to slide from one end of the track to the other in the fastest way. The following formula has shown to achieve the desired behavior in an extremely general way:
$$R_{t+1}(x_t, \dot{x}_t; \mathcal{C}_k) = 4 - |\mathcal{C}_k - x_t| + \mathbb{1}_{|\dot{x}_t| > 0.5} * 1 - \mathbb{1}_{|x_t| > 2.2} * 100 + \mathbb{1}_{|\mathcal{C}_k - x_t| \leq 0.05} * 500 \tag{3}$$
The value $\mathcal{C}_k$ belongs to the vector (and not the range) $\mathcal{C} = [-1.8, 1.8]$, which represents the *checkpoints* that the agent must reach in order to maximize its reward. These checkpoints are strategically chosen positions within the environment that the agent should aim for, encouraging the agent to explore and move towards specific locations. The reward structure associated with the checkpoints is as follows:
    - **High Bonus for Reaching a Checkpoint**: If the agent reaches the current checkpoint $\mathcal{C}_k$, it receives a significant bonus of +500. This encourages the agent to move toward them and recognize them as favorable positions.
    - **Penalty for Moving Beyond the Checkpoint**: If the agent moves too far beyond the current checkpoint, it begins to incur a penalty of -100 for each timestep spent beyond it. This discourages the agent from overshooting the checkpoint and encourages precise control.
    - **Checkpoint Alternation**: The index $k = 1, 2$ alternates every time the agent reaches the current checkpoint, meaning that once one checkpoint is reached, the agent is tasked with reaching the other. This alternation creates a dynamic goal for the agent to move back and forth between the two checkpoints.
    - **Linear Penalization Based on Distance to Checkpoint**: A linear penalty is applied to the agent based on the distance between the current position x and the next desired

checkpoint $\mathcal{C}_k$. The closer the agent is to the next checkpoint, the higher the reward for getting closer, and the greater the penalty if it moves away.

- **Rewarding High Speeds**: The agent is incentivized with a small reward of +1 to move quickly toward its goals. However, it must simultaneously keep the pole upward — challenging particularly during the early stages of training.

The overall strategy is to encourage the agent to visit known "good" positions — the checkpoints near the borders of the environment — and once the agent reaches one of them, it is incentivized to quickly move to the other opposite to again receive the high reward. While the reward function described promotes the desired sliding behavior, the underlying model (the NN) lacks information about the direction in which the agent should move, as it only receives the state vector $[x, \dot{x}, \theta, \dot{\theta}]$. From this state alone, the agent cannot determine which checkpoint it should pursue, since there is no explicit time component or directional cue. This lack of directional information makes it difficult for the agent to understand whether it should move towards the left or right checkpoint, as the state vector does not differentiate between the two directions. To address this limitation, two additional one-hot-encoded input neurons are introduced to the NN. These neurons alternate between values of 1 and 0 indicating to the agent which direction it should move towards. Specifically, one-hot encoding allows the model to distinguish between the two checkpoints and learn which side of the environment it should aim for. For instance, a state of $[0, \dot{x}, \theta, \dot{\theta}]$ provides no directional information about which checkpoint to pursue. On the other hand, a state like $[0, \dot{x}, \theta, \dot{\theta}, 1, 0]$ informs the NN that the agent should move towards the left side, thus adjusting its actions accordingly. This mechanism augments the original state vector by adding contextual information, with the two additional neurons alternating their values based on proximity to the current desired checkpoint. A graphical representation of this is shown in Figure 4.

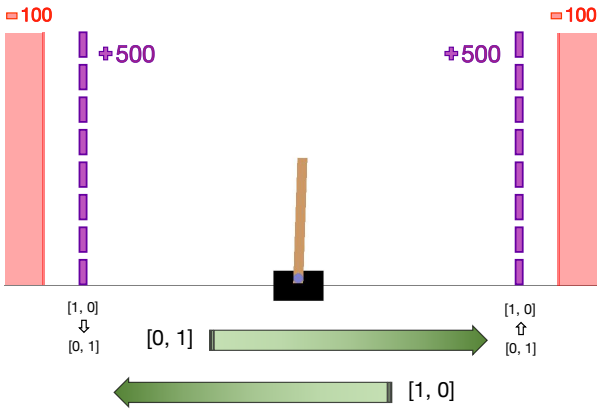

Figure 4: Graphical representation of the points of interest for the third custom reward function. By an alternation of goals, the agent is constantly trying to move from one end to the other in the fastest way.

An important further modification to the NN architecture involves adding an additional layer with the same number (12) of input and output neurons as the preceding layer and replacing the Rectified Linear Unit (ReLU) activation function with the sigmoid function. The additional layer increases the network's expressivity without significantly increasing computational cost. The choice of the sigmoid activation function is motivated by its ability to solve the issues associated with the reward function's potential peaks, which could lead to gradient explosions during backpropagation. The ReLU, while widely used, is susceptible to the *dying ReLU* problem, where large gradients may turn off neurons permanently, reducing the network's capacity to learn effectively. Considering also the fact that the problem is quite symmetric in the x state variable, we could expect certain neurons to deactivate during the initial movement towards one side (e.g., sliding left) and never re-

cover when moving towards the other side. In contrast, the sigmoid, despite its saturation behavior, always maintains a non-zero gradient. Although small, this gradient can still facilitate learning, especially in a shallow network like this one, where high reward values can counterbalance the limitations of sigmoid saturation.

By testing the different reward functions, different behaviors emerge as expected. Whereas the biased one is quite straightforward, the one that implements the sliding behavior requires a careful design and more expensive computations. As shown in Figure 5, where we considered 10 distinct runs over 2,750 episodes of 2,500 timesteps each, the training procedure can be characterized by an extreme variance. This can be motivated not only by the large magnitude of some components of the reward function, but also by the fact that the desired behavior emerges only after some time has elapsed since the beginning of each episode. Thus, to understand whether or not the agent is behaving as expected, some time must pass, and the way in which this happens may be extremely variable from run to run.

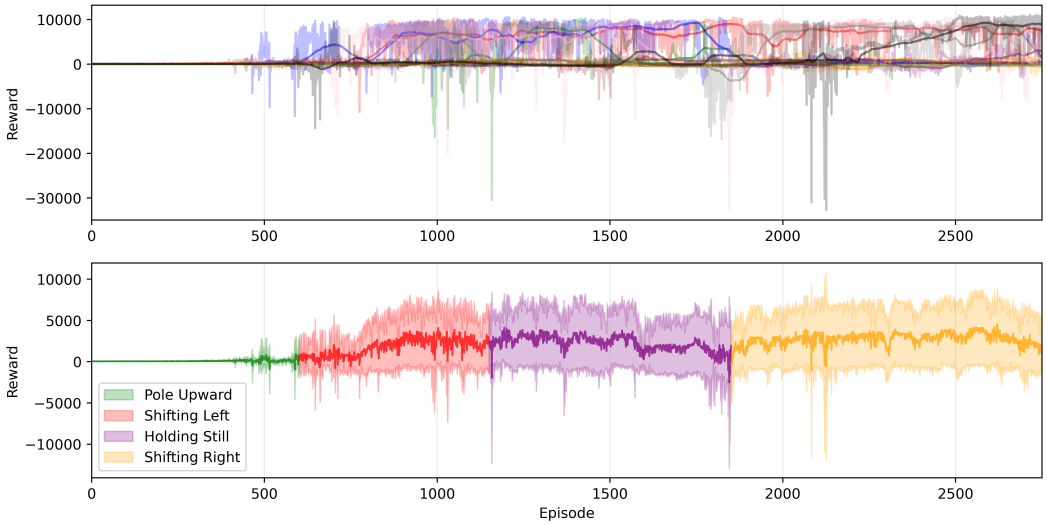

Figure 5: Average total reward and variance computed over 10 independent runs. While the MC method used and the shape of the reward function can make training unstable, increasing the number of episodes and timesteps — set here to 2,750 and 2,500, respectively — enables the discover of an effective and general policy.

To select the best policy, we adopted an early stopping strategy, saving the NN's weights whenever a higher reward than the previous maximum achieved was observed. This approach enhances robustness against training instabilities and episode limits, ensuring that the selected policy reflects the best behavior, *regardless of when it was encountered during training*. To illustrate the agent's learning process, in Figure 5 we also highlight the distinct phases encountered during training. Initially, it focuses on balancing the pole. It then learns that moving to the left yields a high reward and starts prioritizing reaching that position quickly. However, excessive speed and prolonged stay in that area result in significant penalties cumulated over time. Finally, it discovers that a more effective strategy involves sliding from one end to the other in a repetitive pattern throughout the episode.

Once an effective policy is identified, it is evaluated over 100 episodes, each with a 3,000 timesteps limit (we recall the agent has been trained on 2,500 timesteps). Performance and relevant statistics are presented in Figure 6. Remarkably, the agent *independently learns* to mimic the behavior of a Harmonic Oscillator with a high degree of accuracy. The state statistics exhibit the expected oscillatory patterns, with the smoothness of the curves reflecting the policy's optimality. This optimality not only enables efficient sliding but achieves it in a highly effective manner. Peaks in the reward correspond to the agent reaching checkpoints, specifically, the track's extremes where the cart moves, whereas absolute values of the velocity close to 1.5 show that, while achieving the task,

it pays attention in doing so in the fastest way as well. Finally, the averaged actions, calculated using a sliding window of 10 instances, further highlight the agent's understanding of the environment by prioritising actions based on its position and the next checkpoint to reach.

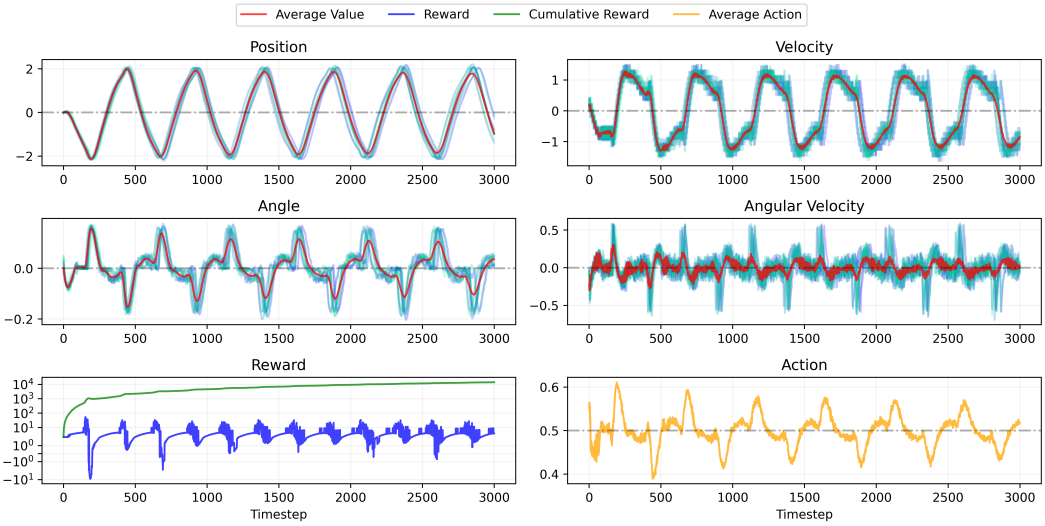

Figure 6: On top and in the middle statistics collected over 10 independent runs, on the bottom both the instantaneous and cumulative reward over one episode. The experiments show a behavior perfectly aligned with the desired one, even though tested over a larger timestep limit. Given the robustness of the strategy adopted, we expect the agent to be able to generalize to longer horizons as well.

4 Conclusion

When comparing an LQR- to an RL-based agent, each approach offers distinct advantages and disadvantages. The use of a fixed model in an LQR facilitates faster convergence, as demonstrated in our experiments, where only 180 timesteps were needed to reach a stable configuration. In contrast, RL-based approaches demand significant training time and computational effort before achieving a satisfactory policy. For scenarios with limited time, an LQR provides an efficient, ready-to-use solution, requiring minimal effort beyond tuning the Q and R matrices. However, the reliance on linear approximations in can lead to degraded performance in more complex scenarios where nonlinear interactions dominate. In such cases, the expressive capacity of an LQR may fail, and a learning-based agent capable of autonomously adapting to complex dynamics can offer a more effective solution. Finally, the LQR approach depends on the availability of a model, which may not always be feasible, particularly for systems with complex or poorly understood dynamics. RL, on the other hand, excels in precisely those situations where developing an accurate model is impractical, as it learns directly from interaction with the environment. This adaptability makes RL a powerful choice for tackling complex, nonlinear, or unmodeled control problems. Another advantage of RL is its ability to uncover *unexpected behaviors* that can outperform already established ones. Thus, while both approaches aim to solve the same problem, an RL-based agent, through continuous interaction with the environment, has the potential to discover novel and innovative strategies.

References

[1] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, 2nd edition, 2018. URL http://incompleteideas.net/book/the-book-2nd.html.

[2] A. G. Barto, R. S. Sutton, and C. W. Anderson. Neuronlike Adaptive Elements That Can Solve Difficult Learning Control Problems. *IEEE Transactions on Systems, Man, and Cybernetics*, 13 (5):834–846, 1983. doi:10.1109/TSMC.1983.6313077.

[3] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba. OpenAI Gym. GitHub repository, 2016. URL https://github.com/openai/gym.

[4] R. W. Brockett. Linear Quadratic Regulators: A Survey. *IEEE Transactions on Automatic Control*, 28(10):1001–1011, 1983. doi:10.1109/TAC.1983.1103094.