National Technical University of Athens

School of Electrical and Computer Engineering

Reinforcement Learning Approach to Control an Inverted Pendulum

Neurofuzzy Control Semester Project

Dimitrios Kampanakis
03121012

Contents

1 Introduction

Machine learning plays a significant role in modern engineering, improving algorithmic performance and using models to solve specific tasks. Reinforcement learning is gaining attention in control theory; however, since this approach does not require any mathematical model, it may be perceived as a less intuitive control method.

The work presented in [[1]] addresses the issue, proposing a framework to produce experiments and simulations on a classic combination problem, the inverted pendulum, targeting both an intuitive approach with the use of Reinforcement Learning algorithms and providing the necessary material for the reproduction of the experiments and simulations. The study focuses on two algorithms: Q-learning and Deep Q-Networks (DQN).

The Inverted Pendulum (also known as "cart-pole") is selected as a system with easily derived mathematical equations that features nonlinearity and under-actuation. These properties make it an ideal candidate to gain comprehensive understanding on the control algorithms and to benchmark the control algorithms, before extending them to more complex systems.

2 Background and Theory

2.1 Inverted Pendulum

The Inverted Pendulum is a system consisting of a mass m located at the end of a rigid rod, which is considered to be massless and of length l. The other extremity of the rod is free to rotate on a motorized cart. The system is subjected to gravity g. As shown in 1 the cart is located at x(t) and ϑ(t) is the angle separating the rod to the downward vertical direction.

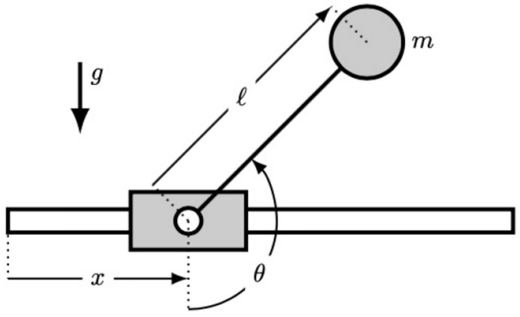


Figure 1: Inverted Pendulum

If the cart is actuated, the dynamics of the system can be described by the following equation:

$$\ddot{\theta} + k_v\dot{\theta} + \omega^2 \sin\theta + \frac{\ddot{x}}{\ell}\cos\theta = 0$$

where:

$\omega = \sqrt{g/l}$, the natural frequency of the pendulum k_v a viscous friction coefficient

It is noted that with a motionless cart the equation describes a damped oscilator and that the track in which the cart is allowed to move has a length of $2x_{max}$.

The purpose is to stabilize the pendulum in its unstable equilibrium position (ϑ=π) by controlling only the motion of the cart. For the experimental setup in the study case, the cart velocity ($\dot{x}$) and the control velocity are linked through the following equation:

$$\ddot{x} = \frac{1}{\tau}(\dot{x}_c - \dot{x}) - f_c \,\mathrm{sign}(\dot{x}) - f_d$$

where:

τ a relaxation time scale f_c, f_d two coefficients to account for the asymetric dry friction on the motorized base The target velocity is proportional to the applied voltage with a constant k_U:

$$\dot{x_c} = k_U U$$

2.2 Reinforcement Learning

Reinforcement learning is a type of machine learning in which an agent (a decision maker using an RL algorithm) learns to make optimal decisions by interacting with the environment (in our case the physical system). In the core of RL for each time step t a state s_t, an action we take a_t, a given reward r_{t+1} and the next state s_{t+1}. [3]

Reinforcement Learning uses the Markov Decision Process framework, in which we have a set of states, a set of actions, a reward and a policy. The policy is the probability of taking an action α, while in state s and determines the sequence of states and actions. The key assumption in MDPs is that the state s_t encodes all the information needed for the choice of the next action. The general objective is to find the policy $\pi^*(\alpha|s)$ that maximizes cumulative reward.

In order to find the best policy, a function designed as the discounted cumulative reward is needed:

$$R_i = r_i + \gamma r_{i+1} + \gamma^2 r_{i+2} + ...$$

where:

$\gamma \in (0, 1)$ is the discound factor, measuring the importance of future unitary rewards in the cumulative reward, with low γ translated to a short sighted "greedy" policy and a high γ meaning a more long-planning behaviour.

An "action-value" function computes the expected cumulative reward at the state s_i, if the action a_i is performed:

$$Q(s_i, a_i) = E[R_i(s_i, a_i)]$$

To determine this function can be difficult and time consuming, since it needs to visit all the states and actions and in a control task that may be long, updating the function Q only at the end of the experiment is not a good practice, so the equation can be rewritten as:

$$Q(s_i, a_i) \approx r_i + \gamma\, Q(s_{i+1}, a_{i+1})$$

where the reward is represented as the expected immediate reward and the cumulative discounted future reward.

2.3 Temporal Differencing (TD)

Temporal Difference (TD) learning updates value estimates incrementally at each time step, without requiring knowledge of the system dynamics or waiting for the termination of an episode. Unlike Monte Carlo methods, which update values only after observing complete returns, TD methods rely on bootstrapping, i.e., updating current estimates using previously learned estimates of future values.

The function described above can be updated iteratively:

$$Q(s_i, a_i) \leftarrow Q(s_i, a_i) + \lambda \, \Delta Q$$

where λ has the role of a time step (in [1] the symbol used is α, here it is swapped with λ to avoid confusion with the symbol of the actions)

Learning proceeds to reducing the error $\Delta Q = Q^* - Q$ over time.

2.4 Q-learning

Q-learning is an off-policy Temporal Difference control algorithm, as it learns the optimal action-value function independently of the behavior policy used to generate data. This allows the agent to explore the environment while simultaneously estimating the value of the optimal greedy policy.

Q-learning [4] tries to learn the optimal action-value function $Q^*(s, a)$ which implies the optimal policy:

$$Q^* \approx r_i + \gamma \max_{a'} Q(s_{i+1}, a')$$

where the best action for the next step is:

$$a^*_{i+1} = \arg\max_{a'} \, Q(s_{i+1}, a')$$

The update rule is:

$$Q(s_i, a_i) \leftarrow Q(s_i, a_i) + \alpha \Big(r_i + \gamma \max_{a'} Q(s_{i+1}, a') - Q(s_i, a_i) \Big)$$

where λ (matches α in [1]) models the learning rate

Q-learning chooses the best action most of the time, with a small possibility of exploring random actions, employing an ε-greedy policy, which promotes the exploration in the early stages and faster convergence at the end of the process, with ε values starting from close to 1 and decaying with the number of episodes as described in [2].

Q-learning is proven to converge to the optimal value function Q^* for finite Markov Decision Processes with discrete state and action spaces, provided that all state-action pairs are visited infinitely often and the learning rate satisfies standard stochastic approximation conditions [4]

The expectation of the cumulative reward is stored in a Q-table covering the state and action space. The dimensions of this matrix are $N_s \times N_a$, with N_s the number of discrete states and N_a the number of possible actions. A large state space and the limited memory of modern computing machines limit this approach.

2.5 Artificial Neural Networks (ANN)

Artificial Neural Networks (ANNs) are computational models inspired by the way biological neurons process information. An ANN consists of interconnected units, called neurons, organized in layers. Each neuron receives signals from neurons in the previous layer, multiplies them by adjustable weights, and combines them with an additional bias term. This weighted sum represents the neuron's internal activation.

The activation is then passed through a nonlinear function, known as the activation function, which determines the neuron's output. By stacking multiple layers of such neurons, neural networks are able to represent complex, nonlinear mappings between inputs and outputs. The parameters of the network, namely the weights and biases, are learned from data by minimizing a loss function that measures the discrepancy between the network's predictions and the desired targets.

In reinforcement learning, neural networks can be employed as function approximators to estimate value functions. In particular, Deep Q-Networks (DQN), which will be analyzed next use an artificial neural network to approximate the action–value function Q(s,a). Instead of maintaining a discrete Q-table, the network takes the current state as input and outputs a set of Q-values, one for each possible action.

The use of neural networks enables generalization across similar states, allowing the agent to learn effective control policies even when the state space is high-dimensional or continuous. Moreover, the nonlinear structure of ANNs makes them well suited to model the complex dynamics of systems such as the cart–pole, where linear approximations are insufficient.

2.6 Deep Q-Network (DQN)

Deep Q-Network replaces the Q-table with a neural network to address the limitation described in Q-learning. Neural networks are particularly well-suited for this task, as they can capture non-linear relationships between states and actions and generalize across similar states through shared network parameters. This enables more compact and expressive representations of the action-value function, significantly improving scalability and learning efficiency.

2.7 Overall Learning Process

The overall learning process is the following:

The state of the environment is measured. The agent updates the policy using either the Q-table or an artificial neural network to update the action-value function and chooses the next action $a \in \{-U, 0, U\}$. After a sampling time Δt, the state moves to the next time step and the cycle continues. The state and action spaces are analyzed thoroughly in the next chapter.

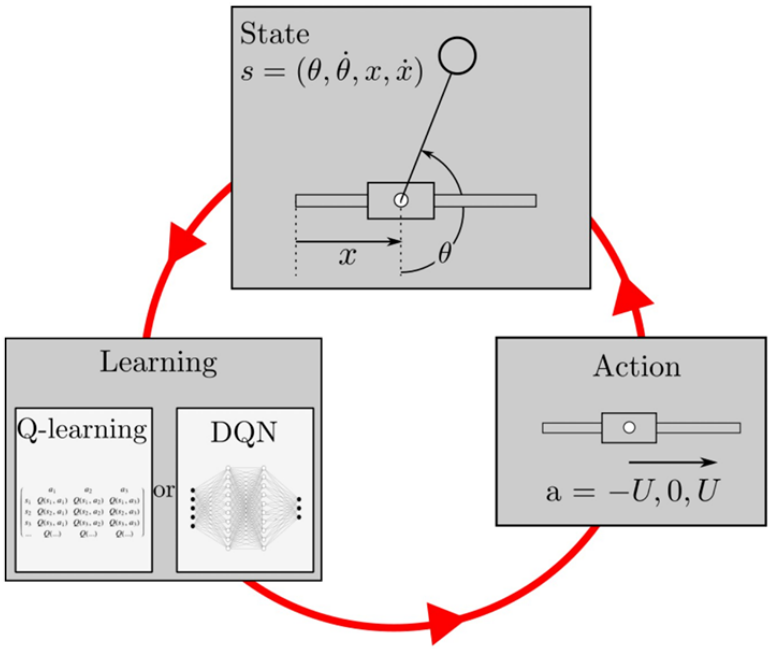


Figure 2: RL learning process

3 Exeriment Procedure - Simulation

3.1 Environment - Parameters

The objective is to maintain the pendulum at the target position ϑ=π, while the cart remains centered on the track (x = 0). In [1] both experiments and simulations are performed, however in this study case more attention is paid to the simulation part, since this can be reproduced.

At each time step the state for the study is described using the pendulum's orientation (ϑ) and angular velocity ($\dot{\theta}$) and the cart's position and velocity ($x, \dot{x}$):

$$s_i = \big(\theta(t_i),\, \dot{\theta}(t_i),\, x(t_i),\, \dot{x}(t_i)\big)$$

To avoid angle discontinuity, the agent is not given ϑ, but sin(ϑ) and cos(ϑ) instead.

At each time step, the agent can choose the applied voltage on the motor of the cart to move it to either direction or keep it in the current position:

$$a_i = \{-U, 0, U\}$$

with $U \in (0, 12V)$ being a fixed value for the applied voltage

The reward function that is chosen in [1] has a maximum of 1, when the pendulum is in ϑ=π and x=0:

$$r(\theta, x) = \frac{1}{2}\big(1 - \cos(\theta)\big) - \left(\frac{x}{x_0}\right)^2$$

From every episode a normalized return is computed as the cumulative reward of the episode divided by the maximum episode length to make the evaluation of the policy easier, with a better episode having a normalized return value closer to 1.

The episode is interrupted when at least on the following conditions is met:

1. The cart exceeds the physical boundaries. This results in a strong penalty (-400 reduction to the episode's reward).
2. The angular speed exeeds 14 rad/s, chosen according to the mechanical limit of the real system
3. The episode has reached the maximum duration $T_{ep} = 800\Delta t$, with Δt = 0.05s. This is chosen to avoid stucking to local minimums and add diversity to the learning process.

3.2 Simulation Setup

In [1] an experiment on the physical system is performed, in this project we try to produce similar results with a simulation of the physical system. In the following table, the values of several crucial parameters regarding the physical system used in the simulation process are shown:

pendulum mass (m)	0.075 kg
natural frequency (ω)	4.882 rad s^{-1}
viscous friction coefficient (k_u)	0.07 N s rad^{-1}
electro-mechanical time constant (τ)	0.0482 s
motor static gain (k_U)	0.051 m $s^{-1}V^{-1}$
static friction coefficient per unit mass (f_c)	1.166
static offset per unit mass (f_d)	-0.097 m s^{-2}
time step (Δt)	0.05 s
standard deviation for ϑ (σ_θ)	2.6 mrad

Table 1: Physical System Simulation Parameters

In the simulation process the measurement of the pendulum angle is updated like:

$$\theta_i^m = \theta_{i-1} + \dot{\theta}\,\Delta t$$

with Gaussian noise to center the measured value around the true value but randomly pertubed, so finally:

$$\theta_i \sim \mathcal{N}\big(\theta_i^m,\, \sigma_\theta^2\big)$$

There is no added noise to the measurement of the angular velocity, since as a consequence of a noisy ϑ, we get a noisy $\dot{\theta}$ with a noise of amplitude

$$\sigma_{\dot{\theta}} = \frac{\sigma_\theta}{\Delta t}$$

4 Simulation of the Experiment

The full working code for the simulation of the experiment can be found in [5] and the process is analyzed in the current chapter.

4.1 Step 1: Physical System

In the first part of the code the dynamics of the pole and the cart/actuator are modeled. We pay attention to the parameters of 1 and to make sure that the simulated system follows the equations of [1] as they are shown in previous chapters (taking into account the equation between cart velocity and control velocity). For numerical stability purposes, the observation space contains $\sin(\vartheta)$ and $\cos(\vartheta)$ and not ϑ. Gaussian noise is introduced to successfully simulate sensors that measure ϑ and $\dot{\theta}$ and will make noisy measurements.

4.2 Step 2: Reward Function, Episode Termination Logic

Following the parameters of the physical system, the track used for the cart extends in the [-0.35, 0.35]. For the completed environment we introduce the reward function discussed and the model the termination logic of the episodes with the corresponding penalty or reward.

For the realistic simulation of the physical system, for the experiments the starting position will be the upright position of the pole for the Q-learning and at rest for DQN (analyzed on the results chapter), in the center of the track. A small noise is added also to these observations to make the simulation seem more realistic and assist the exploration with diversity.

Following the analysis in [1] it is chosed $\sigma_\theta = 2.6mrad$, as measured for the real system. The voltage level for the actuator is selected following the corresponding analysis on the original study. For Q-learning a voltage 12V is selected. For DQN a robust behavior is secured for $U \geq 5.7V$ and the successful experiments are performed with U = 7.1 V, so this value is selected.

4.3 Step 3: Q-learning

The first algorithm that will be tested is Q-learning algorithm. For the discredization of the values used in the state we select nBins = (50, 50, 50, 10, 10), for state ($\cos(\vartheta)$, $\sin(\vartheta)$, $\dot{\theta}$, x, $\dot{x}$) as the best choice described in [1]. With this choice the Q table has 37.5 million entries and using 32bit float datatypes, the table size is 150 MB.

The hyperparameters of the learning process and the decaying method are derived from the paper's help file [2]:

learning rate : $\alpha = 0.01$
discound factor : $\gamma = 0.99$

$$\epsilon(n) = \max\left(\epsilon_{\min},\ \min\left(1,\ 1 - \frac{\log_{10}(n+1)}{d}\right)\right)$$

with:

- n: episode index
- $d = \dfrac{N_T}{10}$
- $\epsilon_{\min} = 0.1$

With the equation for the ε decay the ε-greedy algorithm described can be implemented. As suggested in the paper, for each episode the reward value returned is normalized with the max size of each episode. Moreover, a moving average is implemented over 300 episodes to smooth the data.

4.4 Step 4: DQN

In the second approach in [1] the Q-table is replaced by an Artificial Neural Network, named Deep Q Network (DQN). The DQN implemented has an input layer with 5 nodes (for 5 observations), 2 fully connected hidden layers with 256 nodes each and ReLU activation function and an output layer with 3 nodes for the action-value for each action.

Some stabilization techniques are implemented to keep the simulation aligned with the original paper's approach:

- We implement a replay buffer with a size of 50000 samples improving the data efficiency and stabilizing the training process.
- We perform gradient descend with the parameters in [2], using Huber loss and minibatching technique.
- We use fixed Q-targets (that are updated every 1000 time steps like the paper) to avoid a moving target problem and improve the convergence process.

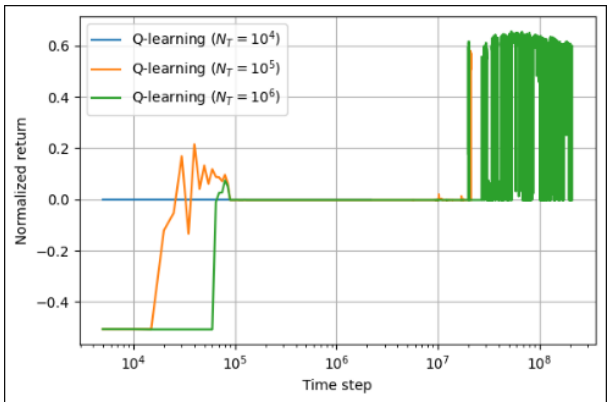
In the analysis that follows, the initial state is $(\vartheta,\ \dot{\theta},\ \mathrm{x},\ \dot{x}) = (\theta_0,\ 0,\ 0,\ 0)$, where θ_0 is a random variable following the normal distribution in $[-10^o, 10^o]$. As in the previous algorithm, here a moving average of 30 episodes is implemented, since DQN is more stable.

5 Results

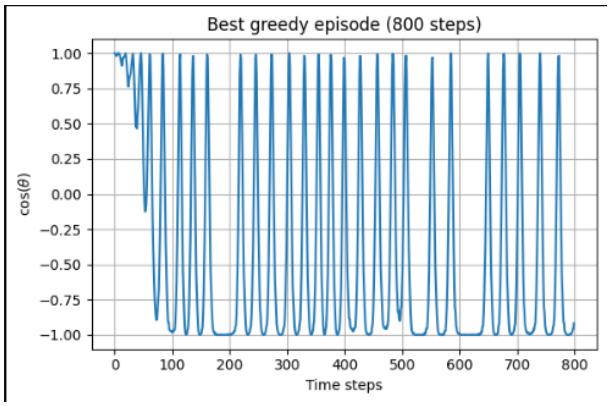
5.1 Q-learning

To faithfully reproduce the Q-learning protocol described in the reference study, several adjustments were introduced in the implementation. First, the training procedure was kept unchanged and initialized near the upright equilibrium, while the evaluation procedure was explicitly decoupled from training. In particular, policy evaluation was performed using a purely greedy policy and a downward initial condition, in accordance with the paper. Second, policy performance was assessed periodically every 5000 environment time steps by averaging the normalized return over 10 independent evaluation episodes. This evaluation metric was recorded as a function of the total number of time steps and used to generate learning curves on a logarithmic scale, matching the presentation of the original work. Finally, to assess the qualitative behavior of the learning policy, an additional inference run was conducted starting from the downward configuration during which the evolution of $\cos(\vartheta)$ was recorded over 800 time steps. This allowed visualization of swing-up and stabilization behavior without influencing the learning process.

Following the paper, the first simulation is performed with nBins equal to 10 for all the variables of the observation:



(a) Coarse discretization - Learning curves



(b) Coarse discretization - cos(ϑ) plot

Figure 3: Plots for coarse disretization

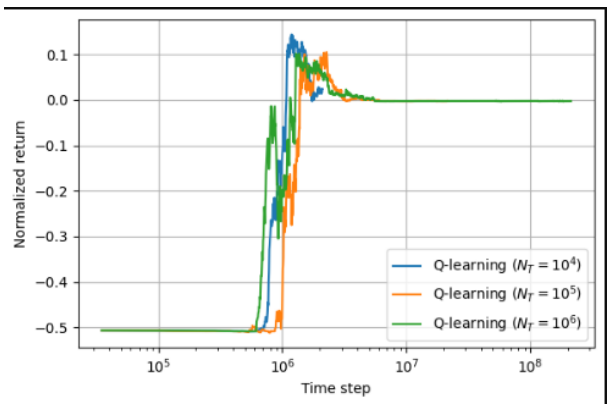
[1]

For a low number of time steps the normalized return stays close to its minimum. For 10^5 episodes the normalized return has a positive result after at least 10^4 time steps and it still never exceeds 0.2 until after 10^7 time steps. For a larger number of episodes, after 10^7 time steps we can see the normalized return reaching 0.6. We note that, using the same sampling interval to reach approximately 10^7 time steps would require nearly six days of uninterrupted real-world experimentation!

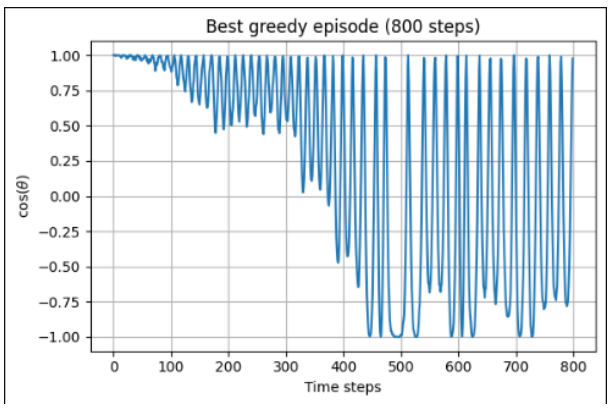
The second plot shows tabular Q-learning is able to achieve swing-up and stabilization within the considered training horizon. However, the resulting control policy remains highly oscillatory, highlighting the lack of robustness and scalability of tabular methods, in agreement with the conclusions of the reference study.

These considerations highlight the practical limitations of tabular Q-learning for this application.

A second experiment is performed with a finer discretization for the values of $\cos\vartheta$, $\sin\vartheta$ and $\dot{\theta}$ as analyzed before and is suggested in the study [1]:



(a) Fine discretization - Learning curves



(b) Fine discretization - cos(ϑ) plot

Figure 4: Plots for fine disretization

With finer discretization, tabular Q-learning exhibits significantly delayed and unstable learning behavior.

[1] For visualization purposes, learning curves are smoothed using a moving average over evaluation points; raw evaluation data exhibit high variance due to the intrinsic instability of tabular Q-learning.

It seems that after 10^6 time steps swing-up can be achieved, but the system cannot be stabilized resulting again in an oscilatory behavior.

Even with finer discretization Q-learning does not converge to an optimal policy. We can conclude that this process is sensitive to the discretization and the exponentially increasing dimensions of the Q-table make the learning more unstable.

Overall, Q-learning cannot robustly stabilize the inverted pendulum. This highlights the poor sample efficiency and scalability of tabular Q-learning for high-dimensional continuous control problems.

Due to computational constraints [2], Q-learning experiments were limited to 10^6 episodes, which was sufficient to observe convergence trends. With this training budget, simulations already required several hours of continuous computation. Extending the experiments to $N_T = 10^7$ episodes would result in execution times scaling to multiple days.

5.2 DQN

To be able to update the action-value function efficiently we need a better approximation to the function than the Q-table. We compare the results produced so far with the following analysis for the the DQN both in learning results and timing.

The training performed uses the hyperparameters in [2] and the maximal number of time steps is set to 150000. The initial conditions for the experiment have been analyzed before and the policy is evaluated every 5000 time steps. As in the original paper two distinct voltages were tested, 2.4 V and 7.1 V.

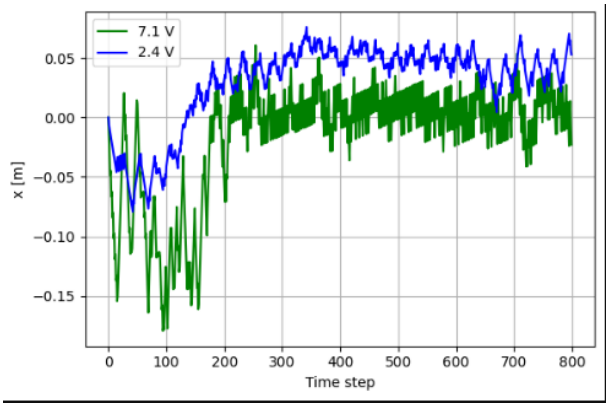
From a timing perspective, the results that follow were achieved in less time steps than with the Q-learning algorithm. Moreover, the work for every episode could be performed faster, since the training of the DQN can benefit from the use of a GPU minimizing the overall training time.

On the plots that follow, results similar to the paper ones are presented and analyzed. It is mentioned that the following plots are almost identical to the original results, validating a reproduction of the experimental results in [1].

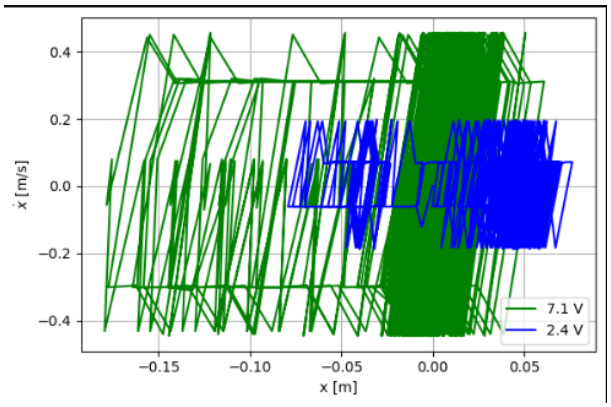
In plot (a) it seems that with the higher voltage level we have stronger oscilations but both cases converge to the same bounded region, stabilizing the cart under different actuation strengths. The second plot (b) shows that the cart dynamics are stable, even with higher voltage the trajectories remain inside the safe bounds.

Plot (c) is the most important to validate the successful learning process. When the cart is actuated with low voltage the pendulum swings slower and oscilates until the episode ends (analyzed before the optimal voltage level). For the higher voltage we can see some oscilations on the first time steps and then the pendulum is stabilized at around $\theta \approx \pi$. This result validates that the pendulum can be stabilized in its unstable equilibrium position.

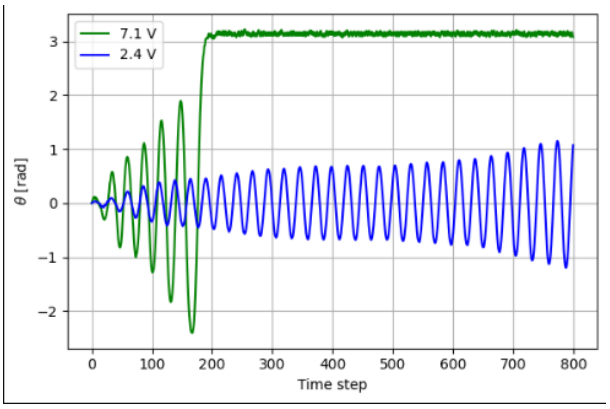
[2] All simulations were executed on a workstation equipped with an Intel i7-class CPU, 16 GB of RAM, and an NVIDIA RTX 4070 GPU. Empirical timing measurements indicate that tabular Q-learning with fine discretization scales approximately linearly with the number of episodes, making training horizons beyond 10^6 episodes computationally prohibitive within reasonable time frames.



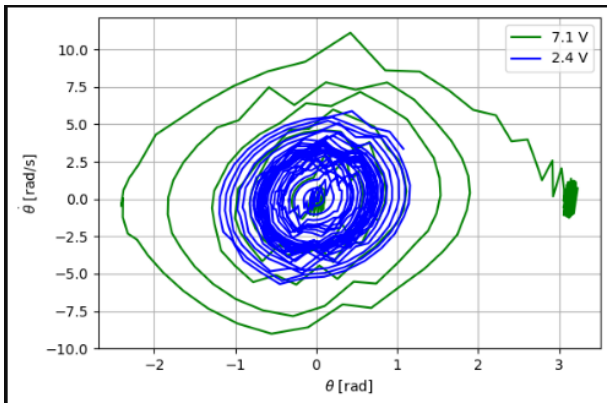
(a) Temporal evolution of the cart's position x during one episode



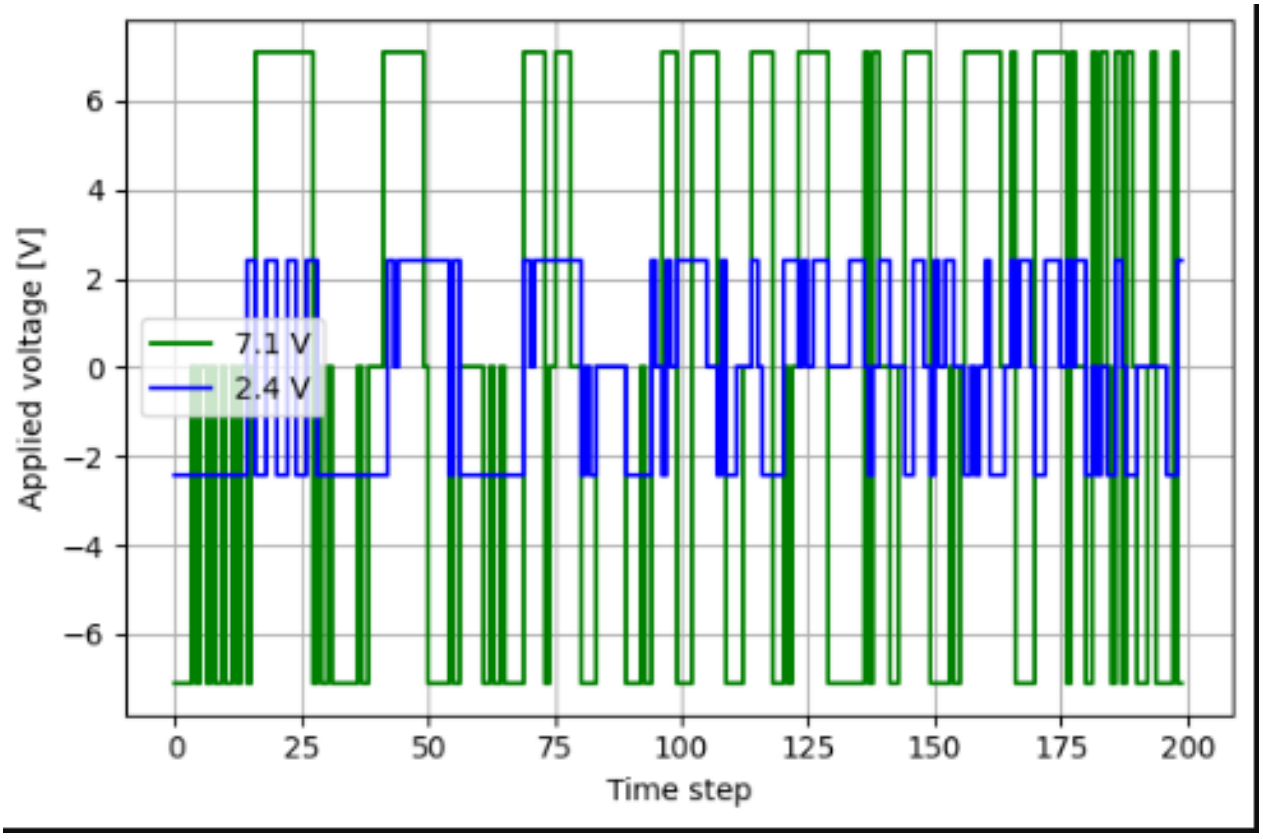
(b) Trajectory of the cart in the $(x, \dot{x})$ space



(c) Temporal evolution of the pendulum's angle ϑ during one episode



(d) trajectory of the pendulum in the $(\theta, \dot{\theta})$ space



(e) Temporal evolution of the applied voltage during the first 200 time steps

Figure 5: Results for DQN

Plot (d) confirms the asymptotic stabilization and fast convergence for the higher value of voltage level. It also visualizes the behavior of the pendulum when it the cart is underactuated. Finally, in plot (e) the policy can be shown, for the plot with the higher voltage level on the early steps the control results in the swing-up phase and on the later steps the switching becomes rapid to stabilize the pendulum upright.

6 Conclusion - Future Work

The article reviewed a classic problem in control theory with modern control techniques, aiming an intuitive approach on complex control theory. In this project the methodology of [1] is analyzed and the experimental results are reproduced successfully in simulation.

The comparative study between tabular Q-learning and Deep Q-Networks highlights the limitations of classical discrete-state approaches which despite theoretical convergence guarantees, suffer from poor scalability and strong sensitivity to discretization. In contrast, DQN demonstrates significantly improved performance and robustness through the use of non-linear function approximation and stabilization mechanisms, enabling effective swing-up and stabilization of the inverted pendulum.

Although Deep Q-Networks do not enjoy general convergence guarantees, the empirical results confirm that, when properly designed, they constitute a powerful and scalable alternative for control problems of increasing complexity. Building on these findings, future extensions of this work may include the investigation of andvanced DQN variants, such as Double DQN or Dueling Network architectures, as well as the application of the proposed framework to more complex systems, including underactuated robots and aerial platforms, as suggested in [1].

References

[1] S. Israilov, L. Fu, J. Sánchez-Rodríguez, F. Fusco, G. Allibert, C. Raufaste, and M. Argentina, "Reinforcement learning approach to control an inverted pendulum: A general framework for educational purposes," *PLOS ONE*, vol. 18, no. 2, e0280071, Feb. 2023, doi: 10.1371/journal.pone.0280071.

[2] S. Israilov, L. Fu, J. Sánchez-Rodríguez, F. Fusco, G. Allibert, C. Raufaste, and M. Argentina, "Supporting Information to: Reinforcement learning approach to control an inverted pendulum: a general framework for educational purposes," 10.1371/journal.pone.0280071.s001

[3] Richard S. Sutton and Andrew G. Barto, "Reinforcement Learning: An Introduction"

[4] Christopher J. C. H. Watkins & Peter Dayan, "Q-learning"

[5] https://github.com/jimkam24/Inverted-Pendulum-Reinforcement-Learning-Control-Simulation