11767 - Autonomous Smart Robots
Master's Degree in Intelligent Systems
Assignment 1

Xisco Bonnín - Alberto Ortiz

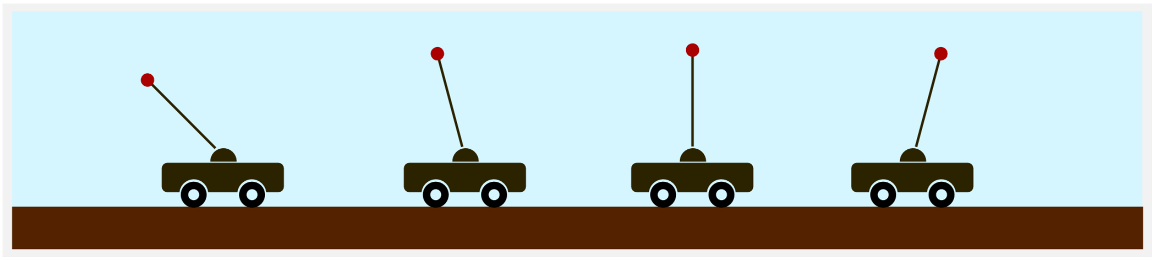
University of the Balearic Islands

May 4th 2023

Inverted pendulum

The inverted pendulum is a classic control problem. The first solution to this problem was provided by James Roberge in 1960. The pendulum consists of a pole placed upright on a cart, to which it is fixed by means of a joint or hinge. **The problem consists in moving the cart in order to keep the pole upright**.

The system state comprises the angle $\phi \in \left[\frac{-\pi}{2}, \frac{\pi}{2}\right]$ (rad), which is zero when the pole is completely vertical, and the angular velocity $\dot{\phi} \in [-\pi, \pi]$ (rad/sec).



There are different parameters which affect the system dynamics: the mass of the pole *m*, the mass of the cart *M*, the length of the pole *d* and the time step Δt, all expressed in the International System of Units. The angle ϕ and the angular velocity $\dot{\phi}$ are updated for time $t+1$ as indicated in the following equations:

$$\phi_{t+1} = \phi_t + \dot{\phi}_{t+1}\Delta t,$$

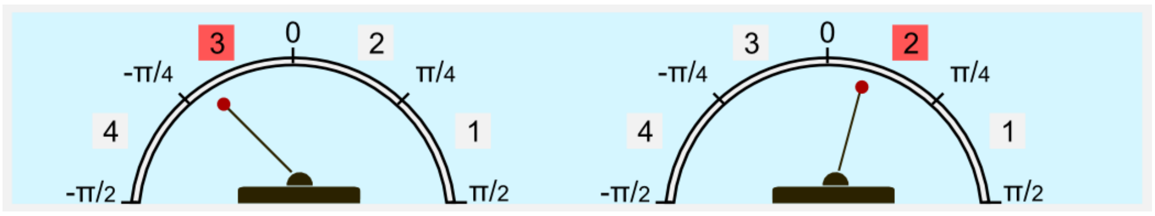
$$\dot{\phi}_{t+1} = \dot{\phi}_t + \frac{g\sin(\phi_t) - \alpha\, m\, d\, (\dot{\phi}_t)^2 \sin(2\phi_t)/2 + \alpha\cos(\phi_t) a_t}{4d/3 - \alpha\, m\, d\, \sin^2(\phi_t)}\Delta t$$

where $\alpha = 1/(M+m)$, and a_t is the action applied at instant t. This actions is a force in Newtons which is applied to the cart, for example, due to a motor that moves it forward or backward as needed.

1 Approach using Reinforcement Learning

In order to solve the problem of the inverted pendulum by reinforcement learning, it is necessary to define how we calculate the reward, and how we discretize the state and the set of actions. We can consider that the reward associated with a state is given by the cosine of the angle ϕ, so that the larger the angle, the lower the reward. In other words, the reward is 0.0 when the stick is completely horizontal, and 1.0 when its position is completely vertical.

The angle and the angular velocity can be discretized into regions of equal dimensions. For example, the following figure shows the case of discretizing the angle into four regions. When the inclination of the stick is $\frac{-\pi}{5}$ rad. (case of the left), we can say that it is in the third region, whereas if its inclination is $\frac{\pi}{6}$ rad. (case on the right), we can say that it is in the second region. As for the set of actions, for simplicity, we use the discrete set of values $[-50, 0, 50]$ (Newtons).



The *inverted_pendulum.py* file contains a Python script that defines the *InvertedPendulum* class. This class contains the methods *reset()*, *step()* and *render()* which allow us, respectively, to prepare a new simulation episode, to move the cart in order to execute a simulation step, and to generate a gif file with the simulation result. This animation is made using the *Matplotlib* library and should be interpreted as the view of a camera which is always looking at the union between the cart and the pole. In order to create a new simulation, it is necessary to create a new instance of *InvertedPendulum*, defining its parameters (masses, pole length and time increment). Here is an example:

```
from inverted_pendulum import InvertedPendulum

# Defining a new environment with pre-defined parameters
my_pole = InvertedPendulum( pole_mass=2.0,
                            cart_mass=8.0,
                            pole_lenght=0.5,
                            delta_t=0.1)
```

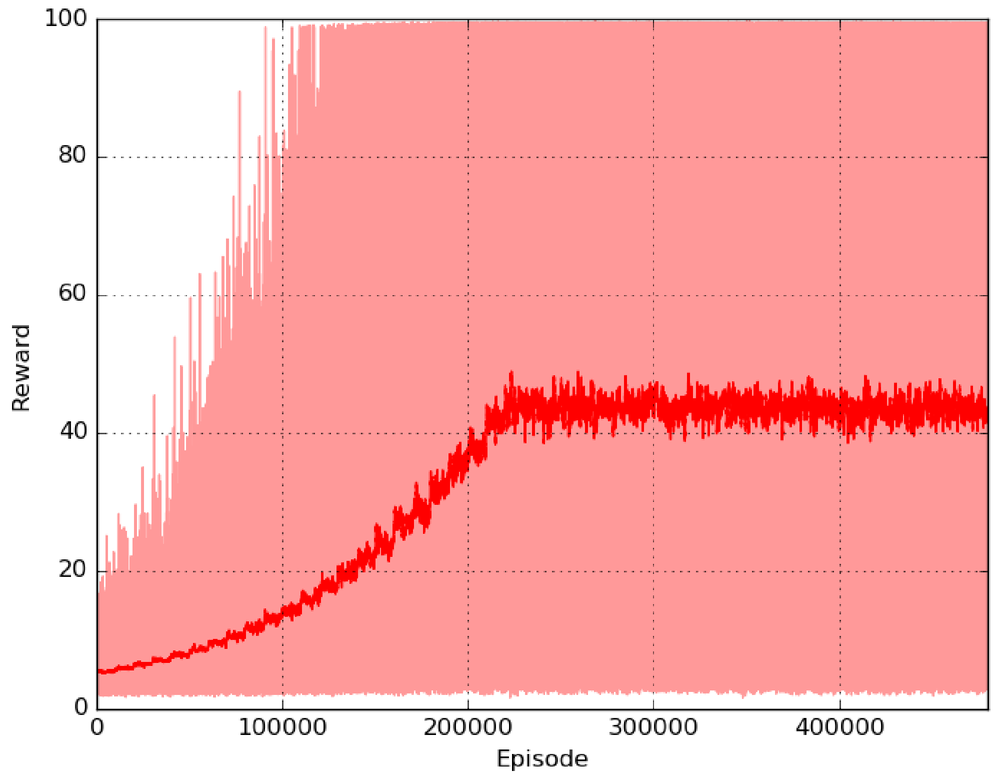
The goal is to prevent the pendulum to fall during a **10 seconds** simulation (100 steps with $\Delta t = 0.1$s). If the pole falls, the simulation episode ends.

As a first test, we can evaluate performance using a random policy. You can do this by running the *random_agent_inverted_pendulum.py* file. As you can see if you run this code, the performance of a random policy is not at all satisfactory, with very short episodes.

2 Solution using Monte Carlo for control

As you can imagine, the optimal policy is to compensate for the angle and speed variations so that the pole is vertical for as long as possible. In the *montecarlo_control_inverted_pendulum.py* file you will find a solution using **Monte Carlo for control using first visit**. This code trains the policy during 5×10^5 episodes, using a *depreciation factor* $\gamma = 0.999$ and discretizing the angle and the angular velocity in 12 equal regions each (parameter *tot_bins*).

In order to favor the exploration, a ϵ-greedy strategy with a linear decay from 0.99 to 0.1 is used. In addition, the *exploratory starts* technique is used to start each episode with a random pole tilt. Each episode is simulated for 10 seconds (100 steps with $\Delta t = 0.1$s). Note that the maximum reward that can be obtained is 100, which is obtained when the pole remains completely upright during the 100 steps. In the following image you can see how the algorithm provides good results, providing an average reward of 45 from, more or less, episode 220000 (the red line indicates the mean of all the 100 rewards corresponding to all the steps in the same episode)). That is, the pendulum is either vertical or nearly vertical, more or less the half of the episode, or the entire episode is tilted between ±63° ($\cos^{-1} 0.45 = 63$).



The resulting policy is quite satisfactory and, with a favorable initial tilt, it is able to keep the stick up for the entire episode (10 seconds).

Now is your turn to run *montecarlo_control_inverted_pendulum.py* file. You will see that during training, three files are generated and updated periodically:

- *inverted_pendulum.gif*: an animation corresponding to the last policy used.
- *reward.png*: a plot providing the mean of all the rewards for all the simulated episodes.
- *step.png*: a plot providing the mean of the number of steps for all the simulated episodes.

Perform the following tasks:

Task 1: Analyze the aforementioned python codes: *inverted_pendulum.py*, *random_agent _inverted_pendulum.py* and *montecarlo_control_inverted_pendulum.py*. Try to understand the code and compare the results obtained with the ones obtained using the random policy.

Task 2: Modify the code *montecarlo_control_inverted_pendulum.py* setting the pole and cart parameters to the values assigned to your group number (you can find the table at the end of the document). You will use these values from now on. Perform a new training using your settings and save the resulting files *inverted_pendulum.gif*, *reward.png* and *step.png* to compare with future results.

3 Solution using SARSA

Task 3: Using the course notes as a guide, and starting from the Monte Carlo code, implement a new solution using the method **SARSA(0)** in a file called *sarsa_0_inverted_pendulum.py*. As in Monte Carlo code, use the ϵ-greedy strategy with a linear decay from 0.99 to 0.1, and using the exploratory starts technique. Once the code is implemented, train for a sufficient number of episodes and save the resulting files *inverted_ pendulum.gif*, *reward.png* and *step.png* to compare with future results.

Task 4: Implement a new solution using the method **SARSA(λ)** in a file called *sarsa_lambda _inverted_pendulum.py*. You can start from SARSA(0), incorporating the concept of elegibility traces. To be precise, implement replacing traces + clearing traces of other actions. Besides, keep on using the ϵ-greedy strategy with linear decay and make use of exploratory starts to facilitate the exploration. Once the code is implemented, train for a sufficient number of episodes and safe the resulting files.

Task 5: Incorporate in *sarsa_lambda_inverted_pendulum.py* the possibility to try other ϵ-greedy strategies. To be precise, implement the following:

- ϵ with linear decay from 0.99 to 0.1 (already implemented),
- ϵ fixed to a certain value (try with different values to find a good one) and
- ϵ with decay according to
$$\epsilon(ep) = max\left\{\mu_\epsilon,\ \epsilon \cdot d^{ep}\right\},$$
where ep is the episode number and ϵ, d and μ_ϵ need to be configured.

Note that we must be able to choose one of the three strategies mentioned by using a parameter in the *main()* function. Once the different strategies are implemented and configured, perform trainings for each of them and save the resulting files.

4 Comparative assessment and report

Task 6: Write a report reviewing each one of the tasks. You must describe the changes you have made at each task, and what improvements you achieved with these changes. In particular, we are interested in comparing the different methods or strategies, showing that you understand what you have done. You have to make this comparison in terms of methods/ideas and in terms of the results you have obtained with the implementation of these ideas. Include reward and step graphs for the different versions/configurations.

Submission

The laboratory assignment must be submitted via UIBDigital before May 24th (Wednesday) at 23:55. You will find a link for that purpose. The submission will consist of a compressed file (zip) containing all the Python codes you have developed, as well as the pdf file corresponding to the report.

Parameters

Group	M	m	d	Δt
1	7	1.5	0.4	0.1
2	8	1.5	0.4	0.1
3	9	1.5	0.4	0.1
4	7	1.5	0.5	0.1
5	8	1.5	0.5	0.1
6	9	1.5	0.5	0.1
7	7	2.0	0.4	0.1
8	8	2.0	0.4	0.1
9	9	2.0	0.4	0.1
10	7	2.0	0.5	0.1
11	8	2.0	0.5	0.1
12	9	2.0	0.5	0.1
13	7	2.5	0.4	0.1
14	8	2.5	0.4	0.1