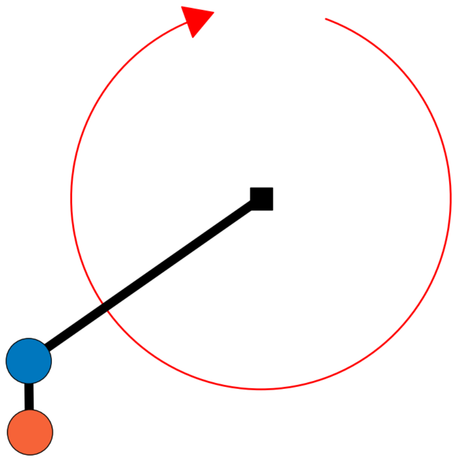
UNIVERSITÀ DEGLI STUDI DI PADOVA

DEPARTMENT OF COMPUTER ENGINEERING

SOLVING DOUBLE PENDULUM SWING-UP WITH REINFORCEMENT LEARNING



REINFORCEMENT LEARNING PROJECT

TRENTI STEFANO
2079450

2023 – 2024

1 Introduction

The goal of this project is that of creating a reinforcement learning based intelligent agent to accomplishes the swing-up movement of an under-actuated double pendulum system. The double pendulum is a 2-link system that is highly chaotic with nonlinear dynamics and in it‘s under-actuated versions, Pendubot and Acrobot, it is very challenging to balance on it‘s upright position. The codebase used for this project is a fork[2] of the Open Source Dual-Purpose Acrobot and Pendubot Platform created by the German Research Center for Artificial Intelligence (DFKI GmbH)[3], Bremen, Germany. To accomplish this task firstly a baseline controller was created and then a reinforcement learning agent was trained. For this task the Pendubot configuration was chosen as the main testing ground for the learning algorithms.

2 Baseline Controller

The baseline controller implemented for this project is an energy-based controller, the main goal is to swing the pendulum from the downward hanging position to the upright position by applying torque to the first link of the system. For the baseline controller, this torque is calculated based on the total energy of this system, the controller calculates for every time step the current kinetic and potential energy, then the control output is calculated based on the energy error between the current configuration and the goal configuration multiplied by a constant k_e:

$$u = k_e * energy\_error$$

This controller naturally drives the system toward the goal configuration but it is not able to stabilize it so a PID controller is utilized to bring the pendulum to the final position and a LQR controller is used for the final stabilization. This controller setup manages to stabilize the system to the goal in 24.64 seconds. The switching between the controllers is managed by reaching some set configurations.

3 Reinforcement Learning Agent - SAC

To complete this task SAC (Soft Actor-Critic) algorithm was chosen, because it is well-suited for continuous control tasks, particularly those involving underactuated systems like the pendubot. SAC, which is a model-free, off-policy algorithm, has many key advantages. Exploration-Exploitation Balance: SAC uses an entropy term to encourage exploration, which is crucial when dealing with complex, non-linear dynamics like those in a pendubot. Stability in Training: SAC optimizes both the actor and critic networks, offering more stable learning than simpler algorithms like DDPG. This stability is very important when training policies to solve tasks such as swing-up, where precise control and robustness are needed. Continuous Action Space Handling: The torque applied to the first joint of the pendubot is continuous, which SAC handles efficiently through its use of Gaussian policies. The Stable Baseline[1] implementation of the SAC algorithm was chosen for this project as this is a proven implementation which provides an efficient and stable way of training multiple instances using the Gymnasium framework. As a first step after choosing the RL architecture is the set up of the environment.

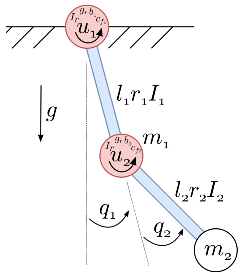
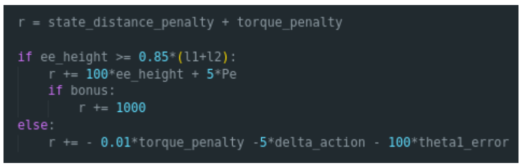


Figure 1: Double pendulum kinematic schematic ($q_1 = \theta_1, q_2 = \theta_2$)

A custom environment *Custom_Env.py* was created by modifying the provided one from the double pendulum package, the main modification was in the double_pendulum_dynamics_func class where the scaling/unscaling functions that convert between the normalized state (bounded in [-1, 1]) to physical state variables (angles and angular velocities / torques). The state representation is made up of four components: $[\theta_1, \theta_2, \dot{\theta_1}, \dot{\theta_2}]$. The angle θ_1 is bounded between $[0, 2\pi]$ with 0 being the downward position and π being the goal position, θ_2 instead is bounded between $[-\pi, \pi]$. The action space is instead represented by the torque value applied to the first joint bounded with the system‘s physical constraints with $max\, torque = 10$. The SAC algorithm is a Deep Reinforcement Learning algorithm that uses two neural networks (actor and critic) to learn the mapping from state to actions. Two Multilayered Perceptrons were used, both with three layers of 128 neurons each. A linear Learning Rate scheduler was implemented as a decay scheduler to allow for more aggressive updates at the beginning of training and more fine-tuned adjustments later, this was however bounded between $[0.01, 0.005]$ to prevent it from becoming too small. This works in combination of the environment‘s reset function which implements a curriculum learning type initialization together with some random noise to improve generalization. This works by initializing the pendulum position very close to the ultimate goal position early in the training and slowly moving the starting position closer to the downward position over 10 million iterations. This strategy of slowly increasing the difficulty of the task helps stabilize learning and reduces convergence time allowing higher exploration of the state space. The reward function was not provided and had to be designed. The chosen reward function is structured to balance two main objectives: achieving the swing-up and minimizing the energy needed to achieve this move. It combines a position based approach and an energy based approach to balance these two aspects and is composed of several elements:

- State distance penalty: It computes the quadratic form of the state error multiplying it by a weight matrix W which is a diagonal matrix where each element corresponds to a weight that will affect an element of the state representation. This encourages the agent to minimize the error between the current state and the goal state.
- Torque penalty: is computed also by a quadratic form combined with a tunable weight, this discourages the use of large torques lowering high energy consumption and aggressive torque maneuvers.
- End-effector height: if the end of the last link of the pendulum reaches a certain height threshold (85% of the max possible height) then a certain reward dependent on the system energy is applied, else a penalty is applied.
- Energy based penalty: the goal position of this system is the position that maximizes it‘s potential energy, this value is computed and used as a reward to drive the system towards the desired configuration.
- Action smoothness penalty: a small penalty is added proportionally to the difference between the current control action and the previous one. This is done to encourage smooth transitions and avoiding jerky controls.
- Region Of Attraction bonus: since this system will have a LQR controller for the final stabilization phase, a ROA was computed based on this LQR controller. If the system enters this region during training it will receive a very large bonus. The ROA is the region where the system can be stabilized by this controller.
- θ_1 error penalty: a large penalty dependent on the θ_1 error, only applied if the end effector is lower than the threshold.

This reward function should be well-suited for this particular task because it balances several factors important to swing-up control: accuracy in reaching the target state, smooth and efficient energy usage, penalizing excessive torque, and rewarding progress towards the goal. To stabilize the system with the LQR controller the computation of the Region of Attraction is necessary, due to compatibility issues with the pydrake package a custom function *Compute_ROA.py* was implemented to compute the necessary $\rho, S\, matrix$ values depending on the choice of Q and R weight matrices of the LQR controller and the pendubot parameters.



```
r = state_distance_penalty + torque_penalty

if ee_height >= 0.85*(l1+l2):
    r += 100*ee_height + 5*Pe
    if bonus:
        r += 1000
else:
    r += - 0.01*torque_penalty -5*delta_action - 100*theta1_error
```

Figure 2: Reward function

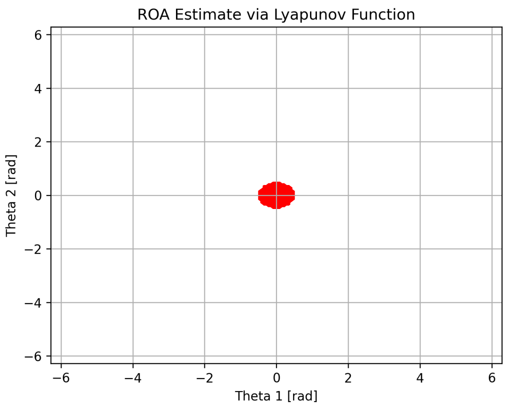


Figure 3: ROA of the LQR controller

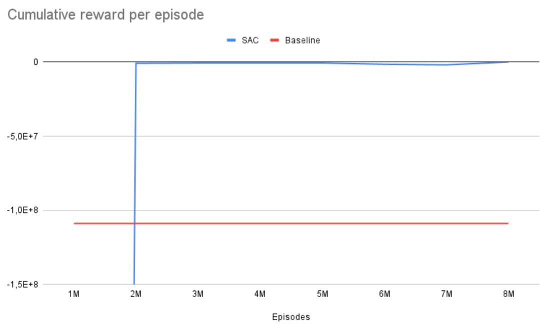
4 Results

The metrics of quality chosen for the comparison between the RL agent and the Baseline controller were: the time to achieve the swing-up and the cumulative reward obtained by simulating one episode and computing the sum of the reward obtained in each step.

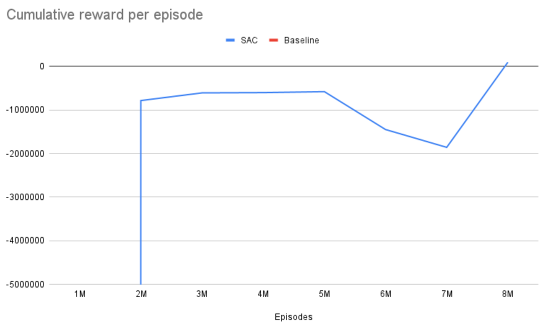
Controller	**Swing-up Time (s)**	**Reward**
Baseline Controller	24.64s	-1.09e8
SAC Agent	6.01s	9.41e4

Table 1: Comparison of Swing-up Time and Reward between Controllers

The SAC agent that had the best performance was the 181st version of the algorithm where in each version the reward function and other parameters where changed and tuned. This agent was trained for 8 million iterations and scored a cumulative reward for one episode of 94144 points. It manages to complete the swing up maneuver in 6.01 seconds and stabilize the pendulum. This is a much better score than the baseline controller that took 24.64 seconds to reach stability with cumulative reward of -1.089e+08. It is noticeable how all agents have much higher reward than the baseline after the 2 millionth episode.



(a) Normal view



(b) Zoom view

Figure 4: Cumulative reward for an episode depending on training episodes

5 Conclusions

This project has been very challenging, but in the end the solution found was positive. The limitations of the compute power available together with the long training session made the testing and tuning of the RL model very time consuming. Looking at the standard pendubot configuration used by the original creators of the double pendulum project, the one that was provided to us was even harder to control and stabilize given that the second link in our configuration is much shorter than the first link. This means that the second link has less inertia, and thus, it is more prone to quick changes in velocity or position due to small disturbances.

When the second link is short, the dynamics between the two links are less balanced, leading to more erratic movements and it has less potential to store and transfer energy during the swing-up process, which makes it harder to accumulate the energy necessary to bring both links to the upright position. In the end a solution for the swing up was found only after more than 180 versions of the training algorithm, the reason is that this system is very sensible and is very hard to precisely tune. The solution obtained for the SAC agent is however by no means a great solution, much faster and quicker movements are possible to achieve this motion and that could be possible with further tuning of the learning algorithm and exploration of the state space. A more precise computation of the ROA could also further improve the performance of the system. In conclusion the SAC agent was successfully designed and trained to accomplish the swing-up task of a pendubot system and it managed to do so in a much quicker and more energy efficient control sequence than the baseline controller.

References

[1] Ashley Hill et al. *Stable Baselines.* `https://github.com/hill-a/stable-baselines`. 2018.

[2] Niccolo Turcato. *double$_p$endulum.* `https://github.com/turcato-niccolo/double_pendulum`. 2023.

[3] Felix Wiebe et al. "Open Source Dual-Purpose Acrobot and Pendubot Platform: Benchmarking Control Algorithms for Underactuated Robotics". In: *IEEE Robotics Automation Magazine* 31.2 (2024), pp. 113–124. DOI: `10.1109/MRA.2023.3341257`.