Pendubot swing-up using Soft Actor-Critic

Dejan Dichoski[†]

***Abstract*—This project addresses the swing-up movement of an underactuated, torque-limited Pendubot using a reinforcement learning-based agent. The agent is trained using the Soft Actor-Critic (SAC) algorithm and is compared against a point PID controller as a baseline. Initial training trials with reinforcement learning revealed several challenges, including potential entrapment in local minima, difficulty in maintaining stability at the highest point for extended periods, and continuous movement in either direction for the entire duration of the trial. These challenges have been addressed, and the findings are reported along with the best-performing model, which was able to perform the swing-up in 1.205 seconds.**

***Index Terms*—SAC; Reinforcement Learning; Pendubot; Double Pendulum; Point PID Controller**

I. Introduction

The Pendubot (short for PENDUlum RoBOT) is a two-link and two-degree-of-freedom planar robot with a single actuator on the shoulder (first joint) and the elbow (second joint) passive. A swing-up represents moving the robot from its stable downward hanging position to its unstable inverted position (unstable equilibrium) and balance it vertically [1]. The challenge of controlling such systems lies in the fact that they have fewer actuators than degrees of freedom.

The Pendubot system is highly nonlinear and exhibits complex dynamic behaviors, making it an ideal candidate for advanced control strategies. To perform the swing-up, first a heuristic policy is developed, which is a set of rules that don't involve any learning. Then a reinforcement learning-based agent is trained, and its performance is benchmarked against the baseline. For stabilization after reaching the goal, a linear quadratic regulator (LQR) controller is used.

To perform the experiments, a custom simulation environment is used, developed by the German Research Center for Artificial Intelligence, but it incorporates standard OpenAI Gym features. Additionally, as an input, a yaml file is given, containing the physical characteristics of the double pendulum plant. To model the dynamics of the double pendulum, 15 parameters are needed, and they are given [2]. These include 8 link parameters, namely masses (m_1, m_2), lengths (l_1, l_2), center of masses (r_1, r_2), inertias (I_1, I_2) for the two links, and 6 actuator parameters, namely motor inertia I_r, gear ratio gr, coulomb friction (cf_1, cf_2), viscous friction (b_1, b_2) for the two joints, and gravity g. Additionally, the torque of the shoulder is limited to 10.0, and the max velocity to 20.0. The Pendubot is simulated with a Runge-Kutta integrator with a timestep of $dt = 0.001s$ for $T = 5s$.

Four variables completely describe the state of the double pendulum. These include (see Fig. 1) the two angles (θ_1 and θ_2) expressed in radians and the angular velocities of the two pendulums (ω_1 and ω_2) expressed in rad/s. θ_1 is the difference from the vertical, and θ_2 from the first rod.

[†]Department of Mathematics, University of Padova, email: dejan.dichoski@studenti.unipd.it

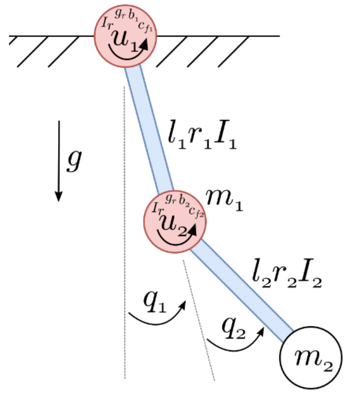

Fig. 1: Double pendulum

So, the whole state is defined as $\mathbf{x} = [\theta_1, \theta_2, \omega_1, \omega_2]$. The initial stable position of the Pendubot is $\mathbf{x}_0 = [0.0, 0.0, 0.0, 0.0]^T$, while the goal state is defined as $\mathbf{x}_{goal} = [\pi, 0.0, 0.0, 0.0]^T$.

For the LQR controller, the operating point is chosen at the goal position, around which the system can be linearized. The region of attraction is computed, allowing us to define switching conditions between the main controller and the LQR as stabilator.

II. Baseline

The initial phase of this project involved defining a heuristic policy to serve as a baseline for benchmarking the RL agent. Several baselines were developed, starting from the simplest approaches up to our final best-performing policy.

Among the baselines that were discarded were the following:

- *A dummy controller*: It applies the maximum torque endlessly;
- *A charging controller*: First, it applies max torque in one direction for a specific charging time, and then switches direction, applying max torque in the other direction;
- *An energy-based controller*: It computes the target energy, which is equal to the maximum potential energy of the plant (at the upward position). Then, at each timestep, the current energy is calculated as the sum of potential and kinetic energy. Finally, the difference between the target and current energy is computed and a torque proportional to this difference is applied in the direction that increases energy.

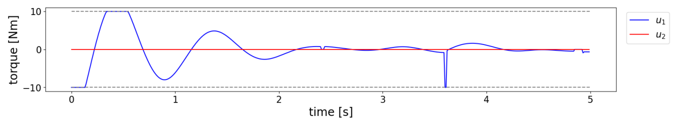

Fig. 2: Torque over time for the Point PID controller

However, as these may work for the simple pendulum, they couldn't solve the double pendulum swing-up task.

The baseline that outperformed all of the above-mentioned approaches, and was chosen for benchmarking the RL agent against, was a *Point PID (Proportional - integral - derivative) controller*. For this controller, the non-negative proportional K_P, integral K_i and derivative K_d coefficients had to be tuned manually [3]. The final values were chosen as: K_p = 6.1, K_d = 0.2 and K_i = 0.1 [4]. The torque that this controller applies on the shoulder over time is displayed in Figure 2. We can observe that it starts by applying maximum torque, and then the torque decreases gradually, slowly approaching zero.

Due to the highly nonlinear dynamics of the double pendulum system, especially during the transition from downward to upright positions, formulating effective rules for this problem proved to be not a naive task. The Point PID controller serves as a decent baseline, and motivates the development of a robust RL agent.

III. Soft Actor-Critic

Various reinforcement learning algorithms can be applied for learning this task, among which Soft Actor-Critic (SAC) was chosen. The reasoning behind this was that SAC is a very successful reinforcement learning algorithm that attains state-of-the-art performance in continuous control tasks (like robotic locomotion and manipulation) [5]. In our scenario, the state (the angles and velocity) and action (torque) variables are continuous real numbers (normalized between -1 and 1). For coding the algorithm, the [6] version was used.

The biggest feature of SAC is that it uses a modified RL objective function [7]. Instead of only seeking to maximize the lifetime rewards, SAC also seeks to maximize the entropy (randomness) of the policy. So, during training, the policy maximizes a trade-off between expected return and entropy. This has a close connection to the exploration-exploitation trade-off: increasing entropy results in more exploration, which can accelerate learning later on. It can also prevent the policy from prematurely converging to a bad local optimum [8].

A. Reward function

The reward function was designed to guide the agent's behavior to achieve stability at the goal position. For stabilization, a LQR controller is used, whose ROA is known. The termination function checks whether the plant has entered the ROA. Additionally, simple zero reset and noisy reset functions are used, which set the initial state to a fixed position (downward equilibrium) and add a small random perturbation to it.

In both the reward and termination functions, the normalized state and action (between -1 and 1) were mapped back to the original physical state quantities. θ_1 was mapped to the range [0, 2π], θ_2 to [-π, π], w_1, w_2 to [-max_velocity, max_velocity], and the action to [-torque_limit, torque_limit]. By doing this, we obtain a difference of π for θ_1 and θ_2 to the goal state when the system is in the initial state.

The reward function consists of three parts. First, a negative quadratic cost function is utilized to encourage smooth swinging away from the initial state. It is formulated as follows:

$$\text{reward}(x,u) = (x - x_{\text{goal}})^T Q (x - x_{\text{goal}}) - u^T R u, \qquad (1)$$

where Q is a diagonal matrix and R is a scalar, since we use only one actuator. These values were carefully chosen to give higher penalty to the angles for deviations from the goal state, and lower penalty to the angular velocities, and torque applied.

Second, the y-coordinates of both pendulums are calculated in order to measure how close the system is to the goal. When the first ball has exceeded the threshold set as $0.8 \cdot l_1$, where l_1 is the maximum height it can achieve (it is the length of the rod), we also check whether the second ball is above the first ball. The second check is crucial because, without it, the system learns to stabilize at the goal position but acts as a single pendulum - the second ball is pointing downward. When this condition is satisfied, we give a reward proportional to how high the second ball is. We also penalize for high velocity to prevent the system from continuously spinning. This reward is applied on top of the first reward, like this:

```
if $y_1 > 0.8 \cdot l_1$ and $y_2 > y_1$:
    reward($x$, action) += $[y_2/(l_1 + l_2)] \cdot 1e^3$
    if $w_1 + w_2 > 8.0$:
        reward(x, action) -= $(w_1 + w_2 - 8.0) \cdot 1e^2$
```

Third, if the agent reaches the region of attraction of the LQR controller, a high reward is given to stimulate it to remain there. This is achieved by just adding $r_{\text{LQR}} = 1e4$ to the final reward.

B. Hyperparameter tuning

During the training of the SAC agent, several hyperparameters were tuned:

- Total learning timesteps: $1e6$, $1e7$, $3e7$;
- Number of environments: 10, 50, 100;
- Timesteps per episode: 500, 1000, 2000;
- Control frequency: 100Hz (dt = 0.01), 1kHz (dt = 0.001);
- Learning rate: 0.01, 0.001, 0.0003;
- Gamma: 0.99, 1.0;
- Quadratic cost parameters:
  - Q1 = Q2: 100.0, 10.0;
  - Q3 = Q4: 1.0, 0.1;
  - R: 0.01, 0.001, 0.0001;

All experiments were performed with a fixed seed of 0, for reproducibility. Initial trials were performed for $1e6$ learning timesteps, but this was insufficient. Increasing the value allowed the agent to explore the state space better and improve its performance. Also, increasing the number of environments gave an improvement in performance while not increasing the training time too much.

For the control frequency and number of maximum timesteps per episode, it was expected that allowing for longer episodes, by increasing the frequency (to 1 kHz instead of 100 Hz) and the maximum episode timesteps (to 2000 instead of 500), would result in better performance. However, in practice, the best-performing agent was trained on shorter episodes, with dt = 0.01 and max_steps = 500, i.e., an episode length of 5 seconds, which was actually the same as the test configuration.

Varying gamma didn't impact the results much, so the default value of 0.99 was kept. On the other hand, the highest value (0.01) for the learning rate gave the best results. When using a smaller learning rate, the most common scenario was a negative reward during the whole training, suggesting entraptment in local minima.

Finally, the quadratic cost parameters were tuned, and several combinations were experimented with. In all cases, the strategy was similar: assign a higher penalty to the angles of the pendulums θ_1 and θ_2, and a lower penalty to the angular velocities and the action, i.e., torque applied. The idea behind this approach was to stimulate fast early movement towards the goal state and then penalize high velocity later when the condition $0.8 < y_1 < y_2$ is satisfied. The best performing hyperparameters were $Q_1 = 10.0$, $Q_2 = 10.0$, $Q_3 = 0.1$, $Q_4 = 0.1$, and $R = 0.0001$.

IV. Results

The point PID controller was able to perform the swing up in 4.75s; however, due to high oscillations, it didn't reach the ROA of the LQR controller and was therefore not able to remain in the unstable equilibrium. If we apply the same reward that was used during the SAC training to evaluate this controller, we obtain a positive mean reward (return). During the initial steps of the episode, it just collects the negative quadratic reward, but it quickly approaches the states near the goal state. The latter positive rewards negate the initial negative rewards, thus giving a final positive return of $103320.86 \approx 1.03e5$.

During training, the RL agent interacts with the environment, collects experiences, and updates its policy. Periodically, the agent's performance is evaluated without policy updates. The rollout/ep_rew_mean metric records the average reward per episode during training, while the eval/mean_reward metric records the average reward during evaluation episodes.

Figure 3 shows the average returns of the SAC agent obtained per episode during the training and validation rollouts, respectively, compared to the point PID controller, which is represented by the red dashed horizontal line. Ignoring the outliers, we can observe that in the first 1e7 steps, the reward is negative. Then from 1e7 to 2e7, there is an almost steady growth, which suggests that the agent is improving the policy over time, and soon it outperforms the baseline. Finally, in the last time steps, the reward saturates near 3e5.

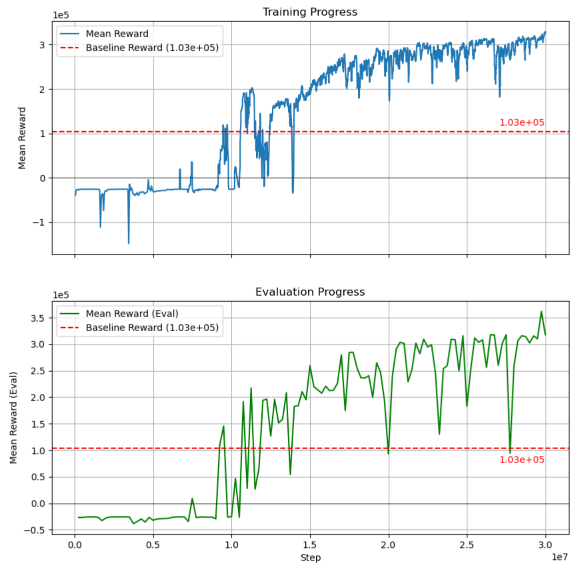

Fig. 3: Comparison of training (rollout/ep_rew_mean) and evaluation (eval/mean_reward) rollouts performance vs the baseline

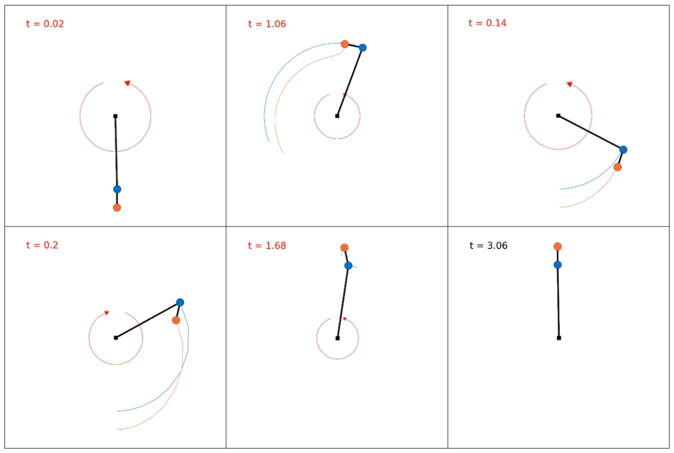

Fig. 4: Simulation results of the SAC agent in an ideal environment

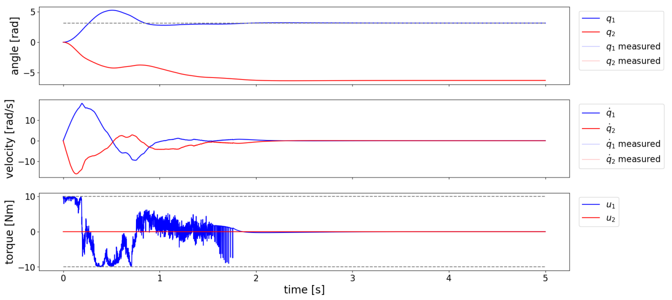

Fig. 5: Swing up and stabilization trajectory of the Pendubot

The episode length for testing was set to 5s, and dt to 0.001. When applying the learned policy using SAC, this agent is able to perform the swing up, as shown in Figure 4. To perform the swing up, the agent takes 1.205 seconds, as shown in Figure 5. Soon after this, the LQR controller is activated, keeping the system in its goal state.

V. Conclusion

In this project, an agent was trained to perform the swing-up task for the Pendubot using the Soft Actor-Critic (SAC) algorithm. The reward function consisted of multiple parts: a negative quadratic cost, a bonus for reaching a certain height, a velocity penalty, and a bonus for reaching the Region of Attraction (ROA) of the Linear Quadratic Regulator (LQR) controller, which was used for stabilization. The performance of this agent was benchmarked against a point PID controller with carefully tuned parameters. Both agents were able to perform the swing-up task, but the RL agent approached the goal position more quickly, thereby obtaining higher rewards during the episode. Additionally, the RL agent was able to reach the ROA of the LQR controller, unlike the baseline policy, which did not. Moreover, the RL agent learned to stabilize on its own, as demonstrated in longer episodes without a combined controller, using only the SAC controller.

Despite the commendable performance of the SAC agent, it is far from ideal. One area for future improvement is the time needed for the swing-up. Exploring other reward functions could be beneficial. The inclusion of height in the reward function was a good direction, and experimenting with different “if“ conditions and additional bonus/penalty mechanisms could yield better results. Furthermore, other model-free RL algorithms, such as Deep Q-Networks (DQN) [9], as well as model-based approaches like MC-PILCO [10], are worth investigating, as they have proven effective for similar systems. Finally, this research should be extended to the acrobot, which has the elbow actuated instead of the shoulder.

References

[1] A. Fumagalli, C. Romaniello, and V. M. Vitale, “Trajectory optimization of an acrobot swing up using iterative linear quadratic regulator (ilqr),” 2023.

[2] DFKI RIC Underactuated Lab, “The double pendulum system,” 2022.

[3] Bobby Rodriguez, “PID Control Explained,” 2023.

[4] DFKI, “PointPIDController,” 2024.

[5] Chris Yoon, “In-depth review of soft actor-critic,” 2019.

[6] Stable Baselines3, “Sac,” 2021.

[7] Vaishak V.Kumar, “Soft actor-critic demystified,” 2019.

[8] OpenAI, “Soft actor-critic,” 2018.

[9] Z. B. Hazem, “Study of q-learning and deep q-network learning control for a rotary inverted pendulum system,” *Discover Applied Sciences*, vol. 6, p. 49, feb 2024.

[10] F. Amadio, A. Dalla Libera, R. Antonello, D. Nikovski, R. Carli, and D. Romeres, “Model-based policy search using monte carlo gradient estimation with real systems application,” *IEEE Transactions on Robotics*, vol. 38, pp. 3879–3898, Dec. 2022.