Reinforcement Learning for Simulated Bicopter Control
CSE498 Fall 2023

Leonardo Santens

December 21, 2023

1 Introduction

This report outlines the development of a reinforcement learning model, specifically using the Soft Actor-Critic (SAC) algorithm, for controlling a bicopter in a simulated environment using CoppeliaSim.

The robot in CoppeliaSim uses the bicopter is modeled as a dynamic system with two propellers controlled by servo motors. The system's state is influenced by the forces generated by the propellers and the orientation controlled by the servos.

Code, video and document is linked on the github[1].

2 Goal of Project

The goal of the project is to successfully implement the Stable Baseline3 RL libraries with the CoppeliaSim remoteAPI for the purpose of learning to hover. The non-holonomic dynamics and lighter-than-air platform of the bicopter model meant that focusing on the height control would allow for rewards in attitude and altitude.

3 Comparison of Reinforcement Learning Algorithms

In choosing the most suitable reinforcement learning algorithm for bicopter control, we considered Deep Deterministic Policy Gradient (DDPG), Soft Actor-Critic (SAC), Proximal Policy Optimization (PPO), and Twin Delayed DDPG (TD3). SAC was ultimately selected due to its advantages in environments with continuous action spaces.

DDPG is an off-policy, actor-critic algorithm that uses deterministic policy. It is suitable for continuous action spaces but can struggle with exploration in complex environments.

PPO is an on-policy algorithm known for its balance between sample efficiency and simplicity. While effective in many scenarios, PPO may not provide the fine control required in complex dynamical systems.

TD3 is an improvement over DDPG, addressing its overestimation bias, but it retains the deterministic nature of DDPG, which can limit exploration.

SAC is an off-policy, actor-critic algorithm that optimizes a stochastic policy. SAC's key feature is its entropy regularization term, encouraging exploration, a crucial aspect for the bicopter's nuanced control:

$$\pi(a|s) = \mathrm{NN}_\theta(s), \quad Q(s,a) = \mathrm{NN}_\phi(s,a), \quad V(s) = \mathrm{NN}_\psi(s) \tag{1}$$

$$\tilde{R}(s,a) = R(s,a) + \alpha\mathcal{H}(\pi(\cdot|s)) \tag{2}$$

Here, $\mathcal{H}$ represents the entropy, and α is a coefficient balancing exploration and exploitation. SAC's ability to learn stochastic policies, combined with its exploration strategy, makes it particularly effective for the bicopter control problem, as detailed in [1].

[1]The source code for simulations and videos is available at `https://github.com/lsantens/CSE498` and the youtube video is at `https://youtu.be/Spl6SkKknp0` and `https://youtu.be/AKfSw_hi9YQ`

4 Evaluation and Experiments with SAC

Training the bicopter control model utilized the Soft Actor-Critic (SAC) algorithm implemented in Stable Baselines 3, chosen for its comprehensive, well-documented, and easy-to-use implementations of state-of-the-art RL algorithms. The model used an MLP policy and was trained for 10,000 timesteps.

SAC's actor-critic architecture, as learnt in class was valuable for exploitative behaviours. I wanted the robot to explore unusual attitudes to learn stability.

5 Reinforcement Learning Algorithm

5.1 Soft Actor-Critic (SAC) Policy $\pi(a|s)$

SAC optimizes a stochastic policy $\pi(a|s)$ which is typically represented by a neural network. The policy outputs a probability distribution over actions given the current state:

$$\pi(a|s) = \mathrm{NN}_\theta(s) \tag{3}$$

where $\mathrm{NN}_\theta(s)$ denotes the neural network parameterized by weights θ, and s is the state. The output a represents the action taken by the agent.

6 Model Components

The following components form the basis of the reinforcement learning model for the bicopter:

- **State Representation:** The state s of the bicopter is represented as a vector that includes its position, orientation, linear velocity, and angular velocity. This can be compactly represented as:

  $$s = [\mathbf{x}, \dot{\mathbf{x}}] \tag{4}$$

  where $\mathbf{x} = [p_x, p_y, p_z, \phi, \theta, \psi]$ includes the position and orientation, and $\dot{\mathbf{x}} = [v_x, v_y, v_z, \omega_\phi, \omega_\theta, \omega_\psi]$ represents the linear and angular velocities.

- **Action Space:** The action space a includes motor forces and servo angles:

  $$a = [f_1, f_2, t_1, t_2] \tag{5}$$

  where f_1, f_2 are motor forces, and t_1, t_2 are servo angles in the range 0 to π.

- **Reward Function:** The reward function $R(s, a)$ is designed with the following components:
  - Orientation Penalty: Penalizes the bicopter for having an aggressive orientation.
  - Angular Velocity Penalty: Penalizes the bicopter for having an aggressive angular velocity.
  - Height Reward: Rewards the bicopter based on its height above the ground level.
  - Ground Contact Penalty: Applies a significant penalty if the bicopter's altitude is too low.
  - Vertical Velocity Reward: Provides a reward for positive vertical velocity.
  - Distance to Goal Penalty: Penalizes the bicopter based on its distance from a goal position.

  Overall, the reward function encourages the bicopter to reach and maintain a desired altitude, penalizes aggressive movements and orientations, and rewards upward movements.

  The mathematical representation of the reward function is as follows:

  $$R(s,a) = \begin{cases} -d_{disttogoal} + \gamma \times H_{reward} - \alpha \times \omega_{orientation} \\ \quad -\beta \times \omega_{penalty} + \delta \times v_{vertical} - \zeta \times G_{contact}, & \text{if } p_z > 0.02 \\ -inf, & \text{reset} \end{cases} \tag{6}$$

  where DistToGoal $= ||\text{Goal} - p||$ is the distance to the goal, HeightReward is based on the bicopter's altitude, OrientationPenalty and AngularVelPenalty are calculated based on the orientation and angular velocities, VerticalVelReward is for upward movement, and GroundContactPenalty is applied when the bicopter is too close to the ground.

7 Evaluation and Experiments

The github repository includes videos of the learning process, dislocation issue and stable flight. Training for bicopter control utilized Stable Baselines 3's implementation of the Soft Actor-Critic (SAC) algorithm, with an MLP policy across 10,000 timesteps. During training, key hyperparameters such as the learning rate and action space variables were tuned.

The evaluation of the efficacy of the RL policy is based on the euclidean error from the goal position of 3 meters. During training in CoppeliaSim I was unable to finish training episodes to a satisfactory standard to deploy the learned values.

8 Encountered Challenges

During the project, several challenges were encountered:

- **CoppeliaSim Issues:** I ran into issues with the bicopter.ttt model dislocating. To combat fidelity issues I reduced the dt for simulating the robot. This partially solve the issue.
- **Python Library Limitations:** Connect the Stable Baseline3 was quite straight forward. However, I didn't understand how to use the trained model save from the SBL3 inside CoppeliaSim for future projects.
- **Bicopter Model Constraints:** The action space is where majority of my time was spent tuning in addition to the hyper-parameters. This was because with the rewards I set, even with a penalty for staying on the ground, if the action space allowed for motors to produce zero force often they would do so. I make the action space start at forces that would induces exploration of flight behaviours. The cost of this what that if the upper limit for the action space was too big, any initial exploration resulted in dislocated bicopter model that through the remoteAPI I wasn't able to reset.

  Another learning with the bicopter RL model was challenges with learned behaviours. I would have learned behaviour of reaching a desired height. Then in the next episode the robot would repeat it once. Following that episode the robot would dislocate it's parts immediately. I wasn't able to solve this issue.

9 Conclusion

The application of SAC for bicopter control in a simulated environment has shown promising potential. Future endeavors will focus on further refining how the model is implemented and exploring other RL algorithms. Perhaps focusing on more constrained actuation spaces could help the simulation for strange edge cases. Also trying the RL model on a different robot prior to implementing on the bicopter could have been a strategy.

References

[1] D. Patiño, S. Mayya, J. Calderon, K. Daniilidis, and D. Saldaña, "Learning to navigate in turbulent flows with aerial robot swarms: A cooperative deep reinforcement learning approach," *IEEE Robotics and Automation Letters*, vol. 8, pp. 4219–4226, July 2023.