Assignment for ME5406

Deep Learning for Robotics

Project 2

Continuous Control for Underwater Exploration Robot

Submitted by

Zhang Ceng

A0263127J

E1010485@u.nus.edu

2023/4/18

1. Introduction

Underwater exploration robots are used in many fields today, such as underwater resource extraction and salvage of sunken ships, and because of the unpredictable environment in the water, the robots are usually placed at the surface and landed at a designated location via pathway points to do the job. In this project, we simulate the use of an underwater robot with propellers and train the agent through RL to be able to perform the task described above.

In this project, we simulate an underwater environment as a two-dimensional plane. Due to the unique characteristics of this setting, a robot's descent is influenced by various factors, such as buoyancy and water flow thrust. To enhance control over its landing trajectory, the robot is equipped with directional boosters, with the switching between them forming the discrete action space in Reinforcement Learning (RL). Each time an action is executed, the robot's pose alters based on the action's outcome, resulting in a corresponding change in environmental state. As an RL agent, the robot's objective is to reach smoothly and accurately a specified, randomly determined position of each run at the bottom of the water while maintaining a reasonable overall trajectory by passing through the required waypoints.

For the implementation of RL, we selected two algorithms to train the agent, after thoroughly understanding their underlying principles and concepts. By deploying and training these algorithms, we compared their outcomes and chose the more effective one as our problem-solving approach. Simultaneously, to emphasize the advantages of the RL algorithm, we evaluated its performance against a traditional manual control method. Our findings demonstrate that the agent trained through RL exhibits superior performance in comparison, which is shown in higher success rate and the trained agent is able to reach the destination on the waterbed without extreme linear velocity to avoid crash at the same time passing through all the required sub-goals as bonus points.

2. Environment Implementation

2.1 Overview

As is shown in Figure 1, the plane world represents the underwater environment and each time the robot is dropped in the middle from the highest level. By the combination of gravity and floating it falls at a slower speed than a normal one, and the task is to enable the robot to land at the **desired position** show as the red flag stably, at the same time it is courage to pass through as many **waypoints** represented by yellow diamonds as possible to maintain a reasonable trajectory. To achieve this goal, the robot needs to adjust its pose along the way by switching the propellers mounted in different directions, at the same time, in order to better control the tilt angle of the robot, the side propellers are respectively installed at an angle of $\pm$ 45 degrees to the horizontal. And randomness is introduced to make it more challenging, like the **uncertain target position** and the red arrow which indicates **the direction of water flow** (no arrow indicates static water), which increase the difficulty of the task so that the agent can train a more robust policy.

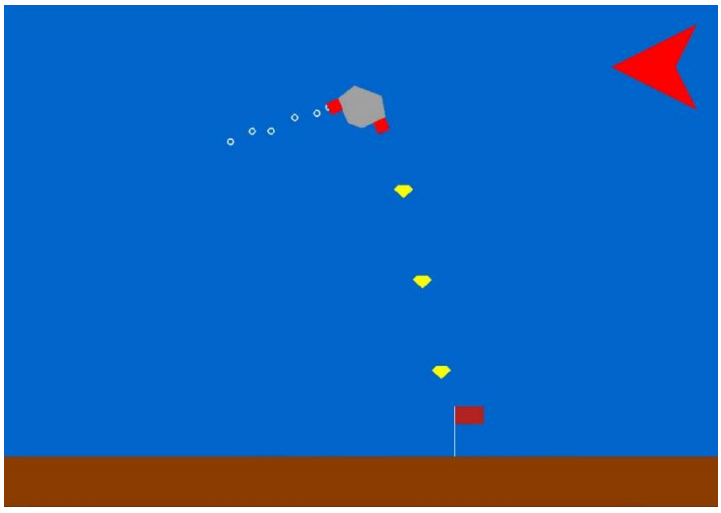

Fig. 1. Underwater Robot Environment

2.2 RL cast

After a brief introduction to the simulated environment above, this part introduces the settings of several important elements in RL cast, which play roles as essences in the subsequent training loop and learning process.

- State Space: This element includes some proprioceptive states of the robot and

environmental observations during operation, as shown in Table 1 in detail. For the sake of simplicity, here we choose 6 observations to consist of the state space.

State	Observation
State [1]	The horizontal distance from the robot to the destination
State [2]	The vertical distance from the robot to the destination
State [3]	The horizontal linear velocity of the robot
State [4]	The vertical linear velocity of the robot
State [5]	The tilt angle of the robot
State [6]	Distance to the nearest waypoint

Tab. 1. Observations of a state

- Action Space: To avoid complicated control logic, we choose discrete actions to achieve faster training speed. Here we use the numbers [0,1,2,3] to represent the switch status of the propeller in different directions.

Action	Observation
0	The robot does nothing and free falls
1	Switch on the right-side propeller
2	Switch the bottom propeller
3	Switch on the left-side propeller

Tab. 2. Components in action space

- Reward Structure: In order to drive the robot to successfully complete the task, the reward function is well-designed and mainly divided into dense rewards obtained at each step and sparse rewards given when certain conditions are met. Specifically, the calculation formulas are as follows in

$$R_{dense}(t) = -[K_p(d_t - d_{t-1}) + K_v(v_t - v_{t-1}) + K_\alpha(\alpha_t - \alpha_{t-1})]$$

where d_t is the distance between the robot and destination, v_t and α_t are the linear velocity and tilt angel of the robot at time t. And the sparse reward is defined as

$$R_{sparse} = \begin{cases} +100 & if\ robot\ passes\ through\ a\ waypoint \\ -200 & if\ robot\ lands\ with\ extreme\ velocity \end{cases}.$$

Besides above, energy consume of engines for each frame is also taken into consideration.

By designed the reward function the robot is encouraged to approach closer to the final goal and reduces its velocity and tilt angle each step while passing waypoints to earn positive rewards.

Once the definition of the above RL elements is done, the construction of the environment is roughly completed and the next step is to deploy the algorithm to train the agent.

3. RL algorithms

In this project we chose two popular RL algorithms, DQN and PPO, to train the agent in the task environment. And in this section, we give a brief introduction to its principles and set the important hyperparameters of their training and the structure of the neural networks.

3.1 Deep Q Network (DQN)

DQN is a combination of Q-learning and neural networks. In the past, Q-learning used Q-Table to store the Q value of each state-action pair. But it works only if the state and action space are discrete and the dimension is not high; if the state and action space are high-dimensional and continuous, there will be a curse of dimensionality problem, that is, as the number of dimensions increases, the amount of calculation increases exponentially. Therefore, DQN adapts Value Function Approximation technique to address this issue. Since it is not possible use a table to accurately store and represent the Q value, it uses a parameterized function to approximately represent the action value function, like

$$Q(s,a) \approx f(s,a,\theta).$$

The DQN algorithm implemented in this project consists of two networks, namely the training network and the target network, the latter copies the weight parameters of the former every once in a while and their share a same structure shown as below.

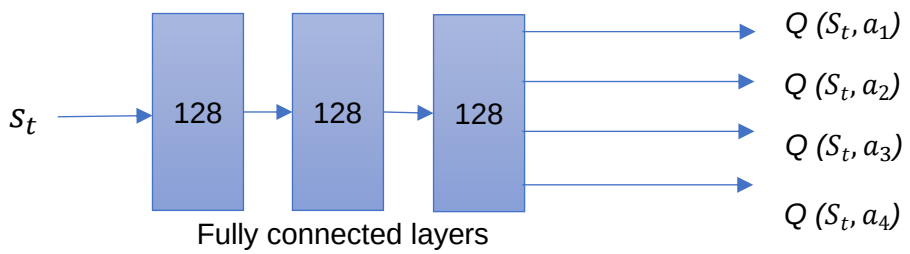

Fig. 2. Network structure of DQN

3.2 Proximal Policy Optimization (PPO)

PPO is an on-policy algorithm, which means it optimizes the policy directly based on the data collected from the current policy. The main idea behind PPO is to strike a balance between exploration and exploitation during the policy optimization process, by limiting the extent to which the policy can be updated in each iteration. This is achieved by using a surrogate objective function with a clipped probability ratio, which prevents overly large policy updates and helps to maintain stability during training thus its loss for the actor network is defined as

$$L^{CLIP}(\theta) = \hat{E}_t\left[\min\left(r_t(\theta)\right)\hat{A}_t, clip(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t\right]$$

where $\hat{A}_t$ is the Generalized Advantage Estimation and $r_t(\theta)$ is the ratio of current policy by the previous one. And the core components of the PPO algorithm include:

- Surrogate objective function: PPO optimizes a surrogate objective function that encourages taking actions with a higher probability under the new policy, while penalizing large deviations from the old policy.
- Clipping: The probability ratio is clipped to a range, such as [1-epsilon, 1+epsilon], where epsilon is a small value, like 0.1 or 0.2. This ensures that the policy updates are constrained, leading to more stable learning.
- Multiple epochs: PPO uses multiple epochs of stochastic gradient descent to update the policy and value function in each iteration, which can help improve sample efficiency.

In this project, the structure of implemented PPO consists of two parts, one is the actor network which outputs actions according to a given state, the other is the critic network that is able to estimate the value of the current states. The network structure is shown as below.

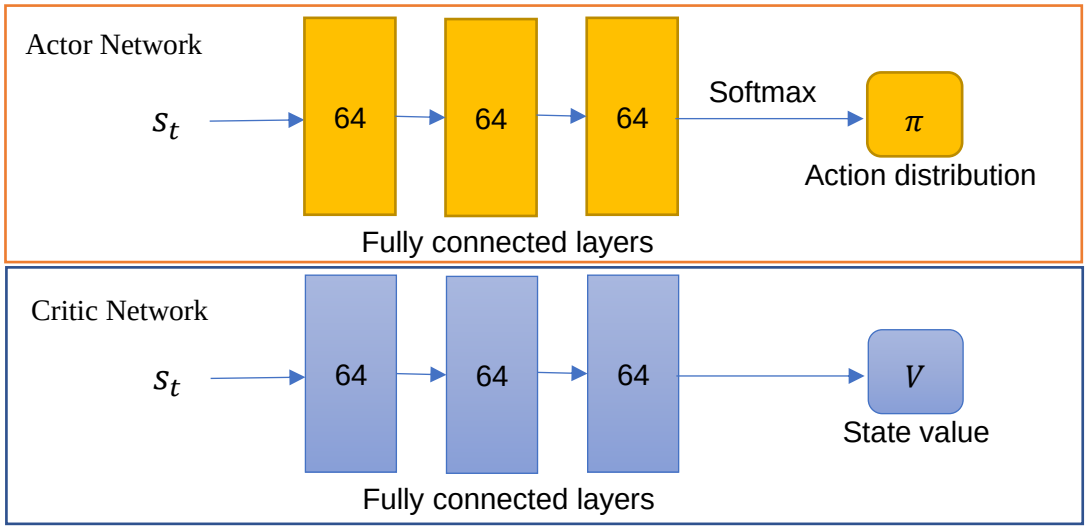

Fig. 3. Network structure of PPO

4. Training results

4.1 Training

In this part, the above two RL algorithms are deployed in the environment to train the agent to complete the task. For the RL algorithms, some of the training parameters are set as below.

Num of episode		5500	
Max episode length		1000	
DQN		PPO	
Learning rate	0.002	Actor learning rate	0.0003
		Critic learning rate	0.001
Discount factor	0.9	Discount factor	0.99
epsilon	0.9	clipping factor	0.2

Tab. 3. Hyperparameters Configuration

After training under the above parameter settings, the data records of the two algorithms are drawn in Figure 4 and Figure 5, which records the learning curve of the agent as the number of episodes increases.

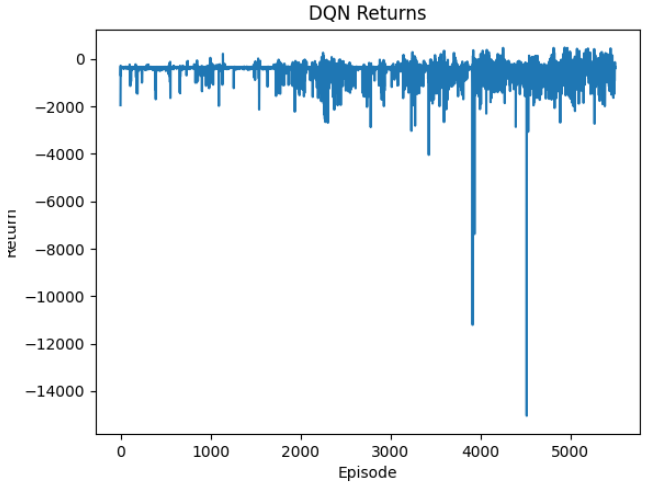

Fig. 4. Learning curve of DQN

It can be seen from the results that the agent trained by the DQN algorithm has never been able to learn the optimal strategy, and the sum of its rewards per round has always been kept at a small value, and it will not converge from the trend of the learning curve, so It can be judged that the training effect of the algorithm for this task environment is not satisfactory.

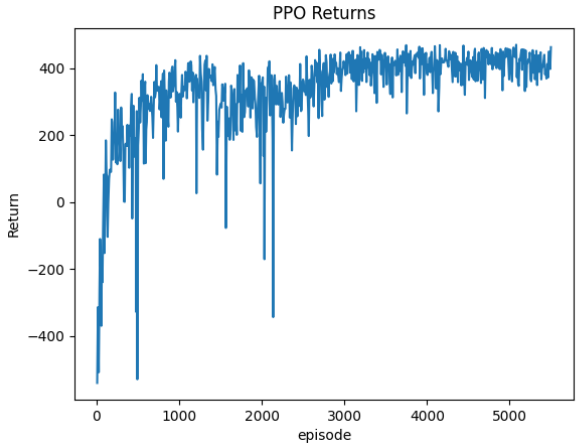

Fig. 5. Learning curve of PPO

In contrast, the learning curve of PPO shows a steady upward trend. Although there is large fluctuations in the early stage of training, the learning strategy gradually converges to the optimum as episodes increase. After the training, the agent can accumulate rewards close to 400 per round, which is already a level that can be judged as completing the task.

4.2 Testing

In order to intuitively evaluate the performance of the trained agent in this task, we used the models obtained by different algorithms to test for 20 episodes, and compared the RL-based methods on the reward value and success rate (accumulated reward within a single round larger than 300). Here a conventional control method by keyboard is also involved because it needs no training thus is a simpler way, but in this approach cannot accurately output the most reasonable action in each state by human in the continuous control, and the figure below shows the differences of these experimental techniques.

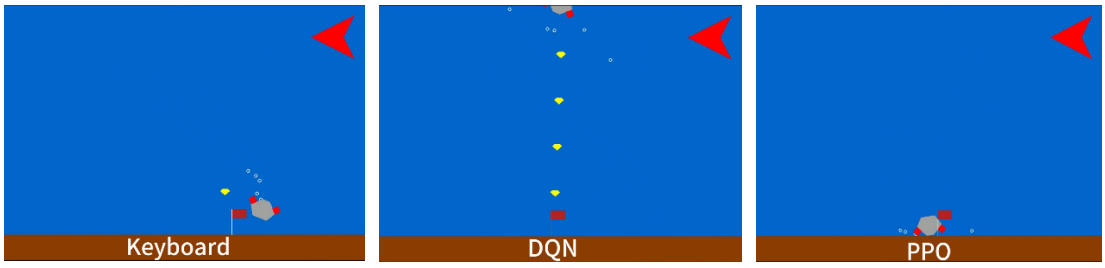

Fig. 6. Comparison between different methods

Method	Keyboard	DQN	PPO
Success rate	10%	0%	84%
Average reward	-1072.3	-300.4	401.3

Tab. 4. Comparison between different methods

5. Discussion and future work

In this project, a custom task environment is built up and 2 of the common RL algorithms are implemented to train the agent for a specified number of episodes. Due to the difficulty of this task and the existence of many disturbance factors in the environment, such as the randomness of the target and the influence of water flow, it is difficult to achieve the expected goal by using the traditional method by keyboard control. In contrast, the robot achieves different performances after training through RL algorithms. From the results, for DQN, the learning curve cannot be converged even after long-term training, and the final performance is that the robot keeps parking at the starting point and dare not move forward; after PPO training, the agent has learned a good strategy to resist the impact and get as many rewards as possible while reaching the destination, which has achieved a satisfactory result. The reason for this difference may be that the idea behind PPO is advanced with more structured networks, which makes its training more efficient.

There are also many challenges met in the implementation of the project, one of which is the logic of the step function, which plays important role in each state-action-state transition; in addition, whether the reward function is well designed directly affects the training performance. In this project, at the beginning the penalty for extreme linear velocity when landing is not set, so the robot after training directly falls to the destination at high speed and ignore the reward of way points, which is very dangerous for the actual robot, so it is necessary to enable the agent to complete the task with the expected trajectory, thus the design needs to take into account various factors in the environment to gradually induce it to step steadily towards the goal.

In addition, there are deficiencies in the design of this project. For example, the tilt angle of the robot when landing is not considered, so it sometimes arrives at the end in a strange pose. Another point is that the water flow in the environment is assumed to be uniform, but in reality its speed changes with the water depth, so these need to be taken into account in future work in order to be more adaptive in a practical case.