Deep Reinforcement Learning for AI-Driven Sea Navigation

Mojmír Horváth

Abstract

This paper presents an AI-driven sea navigation system utilizing deep reinforcement learning techniques. A custom environment simulating maritime conditions is developed, and a Deep Deterministic Policy Gradient (DDPG) agent is trained to navigate from a starting point to a goal location while considering environmental factors such as wind speed, wave height, and depth. The reward system is meticulously designed to encourage efficient and safe navigation, penalizing undesirable behaviors and rewarding progress towards the goal.

1 Introduction

Maritime navigation is a complex task influenced by various environmental factors like wind, waves, and ocean depth. Traditional navigation methods rely heavily on human expertise and heuristic approaches. This project aims to automate sea navigation using deep reinforcement learning, providing a framework where an AI agent learns optimal navigation strategies through interaction with a simulated environment.

2 Environment Description

A custom OpenAI Gym environment, `SeaNavigationEnv`, is developed to simulate maritime conditions. The environment provides realistic feedback to the agent, including wave heights, wind speeds, and ocean depths at specific coordinates and times.

2.1 State Space

The state space is a continuous space represented by a 6-dimensional vector:

$$s = [\Delta\text{lat}, \Delta\text{lon}, \text{heading}, \text{speed}, \text{distance\_to\_goal}, \text{bearing\_to\_goal}] \tag{1}$$

where:

- Δlat and Δlon are the differences in latitude and longitude from the starting position.
- heading is the current heading of the ship in degrees.

- speed is the current speed in knots.
- distance_to_goal is the haversine distance to the goal.
- bearing_to_goal is the bearing from the current position to the goal.

2.2 Action Space

The action space is a continuous 2-dimensional vector:

$$a = [\Delta\text{heading}, \Delta\text{speed}] \tag{2}$$

where:

- Δheading $\in [-30°, 30°]$ is the change in heading.
- Δspeed $\in [-5 \text{ knots}, 5 \text{ knots}]$ is the change in speed.

2.3 Dynamics

The ship's movement is updated based on the current heading and speed. The position is calculated using basic trigonometry and converted into latitude and longitude changes.

2.4 Environmental Feedback

At each step, environmental data such as depth, wind speed, wind direction, wave height, and wave period are retrieved based on the current position and time.

3 Reward System

The reward function is designed to guide the agent towards efficient and safe navigation. It consists of several components:

3.1 Distance to Goal

The agent is rewarded for decreasing the distance to the goal:

$$r_{\text{distance}} = \alpha(\Delta\text{distance}) \tag{3}$$

where α is a scaling factor and Δdistance is the change in distance to the goal since the last step.

3.2 Heading Alignment

Penalties are applied for large differences between the current heading and the bearing to the goal:

$$r_{\text{heading}} = -\beta|\text{heading\_difference}| \tag{4}$$

where β is a scaling factor.

3.3 Environmental Factors

Rewards or penalties are applied based on environmental conditions:

- **Wave Score (r_{wave})**: Encourages navigation in favorable wave conditions.
- **Wind Score for Kites ($r_{\text{wind\_kite}}$)**: Rewards using wind effectively.
- **Wave Height Score ($r_{\text{wave\_height}}$)**: Penalizes dangerous wave heights.
- **Wind Power Score ($r_{\text{wind\_power}}$)**: Rewards efficient use of wind power.

3.4 Fuel Consumption

A penalty is applied proportional to fuel consumption:

$$r_{\text{fuel}} = -\gamma \times \text{fuel\_consumption} \tag{5}$$

3.5 Terminal Rewards

Additional rewards or penalties at termination:

- Large positive reward if the agent reaches within 10 km of the goal.
- Penalty if the ship runs aground or stops moving.

3.6 Total Reward

The total reward at each step is the sum of all components:

$$r = r_{\text{distance}} + r_{\text{heading}} + r_{\text{wave}} + r_{\text{wind\_kite}} + r_{\text{wave\_height}} + r_{\text{wind\_power}} + r_{\text{fuel}} + r_{\text{terminal}} \tag{6}$$

4 Algorithm: Deep Deterministic Policy Gradient (DDPG)

DDPG is an actor-critic algorithm suitable for continuous action spaces. It consists of:

- **Actor Network**: Proposes actions given the current state.
- **Critic Network**: Estimates the Q-value of state-action pairs.
- **Experience Replay**: Stores experiences to break correlation between samples.
- **Target Networks**: Stabilize training by slowly tracking the learned networks.

4.1 Actor Network Architecture

The actor network maps states to actions:

$$x_1 = \tanh(W_1 s + b_1) \tag{7}$$
$$x_2 = \tanh(W_2 x_1 + b_2) \tag{8}$$
$$a = \text{action\_low} + (\tanh(W_3 x_2 + b_3) + 1) \times 0.5 \times (\text{action\_high} - \text{action\_low}) \tag{9}$$

4.2 Critic Network Architecture

The critic network evaluates state-action pairs:

$$x_1 = \text{ReLU}(W_1[s; a] + b_1) \tag{10}$$
$$x_2 = \text{ReLU}(W_2 x_1 + b_2) \tag{11}$$
$$Q(s, a) = W_3 x_2 + b_3 \tag{12}$$

4.3 Training Procedure

Algorithm 1 DDPG Training Loop

```
 1: Initialize actor network $\mu(s|\theta^\mu)$ and critic network $Q(s,a|\theta^Q)$ with weights $\theta^\mu$ and $\theta^Q$
 2: Initialize target networks $\theta^{\mu'} \leftarrow \theta^\mu$, $\theta^{Q'} \leftarrow \theta^Q$
 3: Initialize replay buffer $\mathcal{D}$
 4: for episode = 1 to $M$ do
 5:     Reset environment and get initial state $s_0$
 6:     for t = 1 to max_steps do
 7:         Select action $a_t = \mu(s_t|\theta^\mu) + \mathcal{N}_t$ (with exploration noise)
 8:         Execute action $a_t$ and observe reward $r_t$ and new state $s_{t+1}$
 9:         Store transition $(s_t, a_t, r_t, s_{t+1})$ in $\mathcal{D}$
10:         Sample minibatch of $N$ transitions from $\mathcal{D}$
11:         Set $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$
12:         Update critic by minimizing loss $L = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i|\theta^Q))^2$
13:         Update actor using policy gradient $\nabla_{\theta^\mu} J \approx \frac{1}{N}\sum_i \nabla_a Q(s,a|\theta^Q)|_{s=s_i,a=\mu(s_i)} \nabla_{\theta^\mu}\mu(s|\theta^\mu)|_{s_i}$
14:         Soft update target networks:
                $\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$   (13)
                $\theta^{\mu'} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'}$   (14)
15:     end for
16: end for
```

5 Training

The agent is trained for 1000 episodes. Key hyperparameters include:

- **Discount Factor (γ)**: 0.99
- **Soft Update Parameter (τ)**: 0.001
- **Batch Size**: 64
- **Actor Learning Rate**: 1×10^{-5}
- **Critic Learning Rate**: 3×10^{-5}

5.1 Exploration Strategy

An Ornstein-Uhlenbeck process is used to add temporally correlated noise to the actions, promoting exploration in the continuous action space.

6 Results

During training, the agent learns to navigate towards the goal while optimizing the reward components. The distance to the goal decreases over episodes, and the total reward shows an increasing trend.

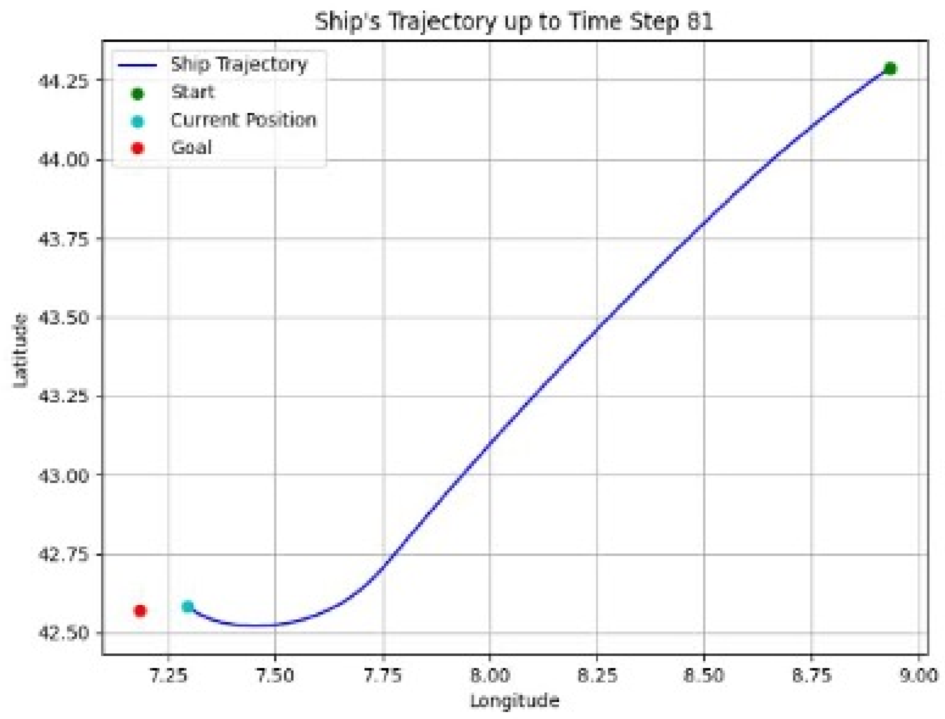

Figure 1: Example Route from A to B

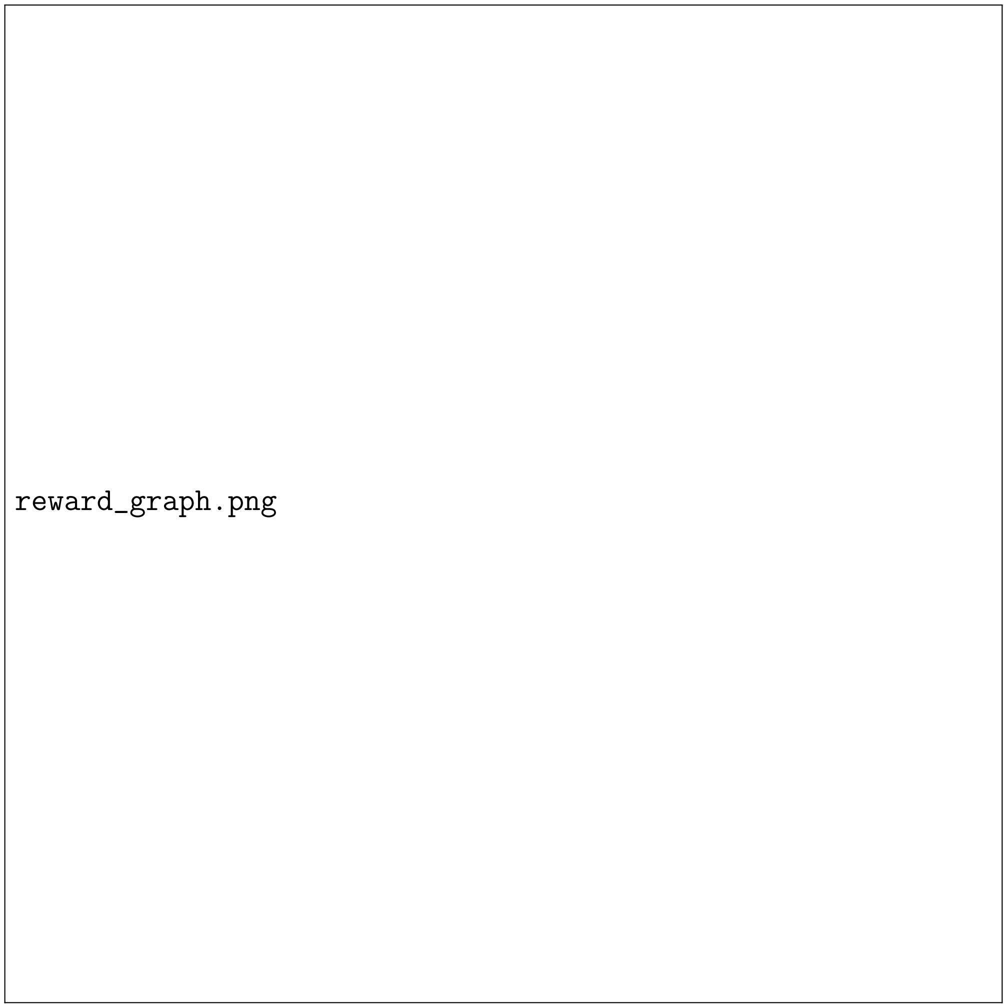

Figure 2: Total Reward over Episodes

7 Conclusion

The project demonstrates the potential of deep reinforcement learning in automating complex tasks like sea navigation. By carefully designing the reward function and environment dynamics, the agent effectively learns to make navigation decisions that consider environmental factors and safety constraints.

8 Future Work

Future enhancements may include:

- Incorporating more realistic environmental models.
- Extending the action space to include more navigational controls.
- Testing the agent in varied geographic locations and conditions.