# A Reinforcement Learning Approach to Ship Towing Using Two Tugboats

Rahul Kulkarni, Mohammad Saifullah Khan

December 3, 2024

**Abstract**

This project explores Multi-Agent Reinforcement Learning (MARL) techniques, specifically MADDPG and MAPPO, for towing ships using two tugboats in a custom simulation environment. The agents aim to collaboratively tow a ship to a dock while avoiding obstacles and maintaining safe rope lengths. Despite implementing realistic constraints and reward structures, results showed challenges in convergence, with agents underperforming compared to random trajectories. Future improvements include refining reward systems and incorporating dynamic obstacles, highlighting the challenges of applying MARL to maritime tasks.

## 1 Contributions

- Rahul Kulkarni - MAPPO
- Mohammad Saifullah Khan - MADDPG

## 2 Introduction

Towing large ships to specific target locations is a critical and challenging operation in maritime navigation. It requires precise coordination between tugboats to ensure efficient maneuvering while avoiding obstacles and maintaining safety. This report explores an approach using Reinforcement Learning (RL) techniques such as Multi-Agent Deep Deterministic Policy Gradient (MADDPG) and Multi-Agent Proximal Policy Optimization (MAPPO).
In this project, a custom simulation environment was developed to train and evaluate two tugboats working collaboratively to tow a ship. The simulation incorporates realistic constraints, such as high-dimensional continuous action spaces, dynamic environmental conditions, and agent coordination. Each tugboat operates as an independent agent with its own observation space, which includes information about the ship's position and orientation, the relative positions of the other tugboat, the distance to the target, and rope length. The action space consists of controlling the tugboat's velocities in two dimensions.

A reward system was designed to encourage the agents to minimize the distance to the target, avoid obstacles, and maintain a safe and effective rope length to tow the ship near the dock.
This report details the problem formulation, methodology, reinforcement learning models, results, and potential future improvements, providing an overview of how multi-agent reinforcement learning can be applied to real-world maritime operations.

# 3 Environment

The environment for the ship-towing task is a custom-designed simulation developed to replicate a maritime towing scenario using gymnasium. It provides a controlled setting for training and evaluating the performance of two tugboats, modeled as independent agents, in collaboratively towing a ship to a pre-defined target location.

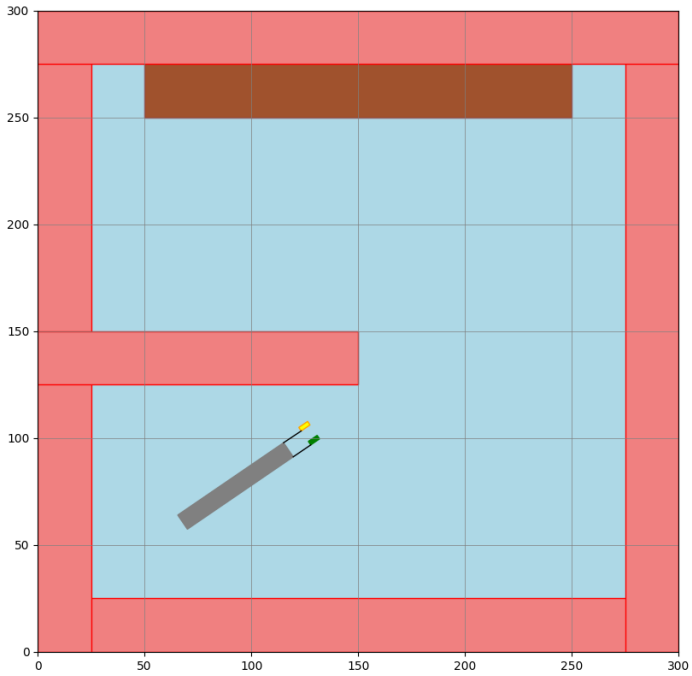



Figure 1: Environment. Dock (brown-colored), ship (grey-colored), tugboats (green and yellow-colored), obstacles (red-colored) and water (blue-colored)

Key features of the environment include:

### 3.1 Simulation Grid

The environment is designed as a $300 \times 300$ grid that simulates the water area where the ship and tugboats operate. Within this grid, obstacles are positioned, requiring the agents to plan their movements carefully to avoid collisions. Additionally, a designated dock region near the target serves as the goal location, where the ship must be successfully towed to complete the task.

### 3.2 Agents and Ship

Two tugboats are modeled as independent agents. The ship is modeled as an object influenced by the forces exerted through the ropes attached to the tugboats. The simulation incorporates physical properties such as mass, inertia, linear drag, and angular drag. The ship's movement is governed by equations of motion, which account for the forces applied by the tugboats via the ropes and the effects of water resistance, modeled using drag coefficients.

### 3.3 Target Region and Reward System

The target region, represented as a rectangular area within the grid, simulates the dock where the ship must be accurately positioned to complete the task. A reward system is implemented to guide the agents' actions, giving reward to minimize the distance between the ship and the target, and prevent rope overstretching. Proximity to obstacles incur penalties, encouraging agents to plan safe paths.

### 3.4 Visualization

The environment is visualized using Matplotlib, clearly representing the ship, tugboats, ropes, obstacles, and the target dock.

## 4 Problem Formulation

Grid size is $300 \times 300$.
Two tugboats in the environment are responsible for towing the ship to the dock.

- **Observation Space of each tugboat:** $(x_s, y_s, \theta_s, x_{t1}, y_{t1}, \theta_{t1}, x_{t2}, y_{t2}, \theta_{t2}, d_s, l)$.
- **Action Space of each tugboat:** $(v_x, v_y)$.

where,

- $(x_s, y_s, \theta_s)$ is the ship's position and orientation.
- $(x_{t1}, y_{t2}, \theta_{t1})$ is the tugboat's (own) position and orientation.
- $(x_{t2}, y_{t2}, \theta_{t2})$ is the other tugboat's position and orientation.

- $d_s$ is the Euclidean distance from the ship to the dock.
- $l$ is the rope length.
- $(v_x, v_y)$ is the tugboat's (own) velocity in x and y direction.

The reward structure is as follows:

$$R = R_{to\_target} + R_{proximity\_obs} + R_{rope\_length} + R_{in\_target} \tag{1}$$

where,

- $R_{to\_target} = \frac{d_s}{gm \times 0.004}$ is the penalty given to each tugboat for taking each step, which reduces as the ship moves close to the dock.
- $R_{proximity\_obs}$ is the penalty given to each tugboat for going close to the obstacles.
- $R_{rope\_length}$ is the penalty for stretching the rope beyond a certain threshold (which is set as 10.0 units here).
- $R_{in\_target}$ is the high positive reward given to each agent for taking the ship to the target location.

# 5 Approaches

## 5.1 MADDPG

**Algorithm 1** MADDPG

Initialize environment with 2 agents.
Initialize actors $\mu_i$, critics $Q_i$, and their target networks $\mu_i'$ and $Q_i'$ for each agent $i \in \{1, 2\}$.
Initialize replay buffer $\mathcal{D}$ for each agent $i$.
Set hyperparameters: learning rates $\alpha$, discount factor $\gamma$, soft update parameter $\tau$, batch size $B$, and noise parameters.
**for** episode = 1 to $N_{\text{episodes}}$ **do**
    Reset environment
    Initialize cumulative rewards for each agent: $R_i \leftarrow 0$ for $i \in \{1, 2\}$
    $t \leftarrow 0$
    **while** episode not done and $t <$ max_steps **do**
        **for** each agent $i \in \{1, 2\}$ **do**
            Compute action $a_i = \mu_i(s_i) + \text{noise}$
        **end for**
        Execute actions $\{a_1, a_2, \ldots, a_n\}$ in environment
        Observe next states $\{s_1', s_2', \ldots, s_n'\}$, rewards $\{r_1, r_2, \ldots, r_n\}$, and done flags
        **for** each agent $i \in \{1, 2\}$ **do**
            Store transitions $(s_i, a_i, r_i, s_i', \text{done})$ in replay buffer $\mathcal{D}_i$
            Update cumulative rewards $R_i \leftarrow R_i + r_i$
            $s_i \leftarrow s_i'$
        **end for**
        $t \leftarrow t + 1$
    **end while**
    **for** each agent $i \in \{1, 2\}$ **do**
        Sample a batch of $B$ transitions from $\mathcal{D}_i$: $(s, a, r, s', \text{done})$
        Compute target actions: $a_i' = \mu_i'(s_i')$
        Compute target Q-values:

$$y_i = r_i + \gamma(1 - \text{done})Q_i'(s_i', a_i')$$

        Compute current Q-values: $Q_i(s_i, a_i)$
        Update critic by minimizing loss:

$$\mathcal{L}_{\text{critic}} = \frac{1}{B}\sum(Q_i(s_i, a_i) - y_i)^2$$

        Update actor using policy gradient:

$$\mathcal{L}_{\text{actor}} = -\frac{1}{B}\sum Q_i(s_i, \mu_i(s_i))$$

        Perform gradient updates for actor and critic
        Update target networks using soft update:

$$\theta' \leftarrow \tau\theta + (1 - \tau)\theta'$$

    **end for**
    Log rewards and losses
    Save model checkpoints periodically
**end for**
**Return** trained actors $\{\mu_1, \mu_2, \ldots, \mu_n\}$

## 5.2 MAPPO

**Algorithm 2** Multi-Agent Proximal Policy Optimization (MAPPO)

**Input:** Environment, 2 agents, number of episodes $M$, max steps per episode $T$
**Hyperparameters:** Learning rate $\alpha$, discount factor $\gamma$, GAE factor $\lambda$, PPO clip ratio $\epsilon$
Initialize environment and MAPPO $\{\pi_{\theta_i}, V_{\phi_i}\}_{i=1}^{N}$
Initialize shared optimizer for agents with learning rate $\alpha$
**for** each episode $e \in \{1, 2, \ldots, M\}$ **do**
    Reset environment and observe initial states $\{s_i\}_{i=1}^{N}$
    Initialize memory $\mathcal{D}_i$ for each agent $i$
    $R_{\text{episode}} \leftarrow 0$ ▷ Initialize cumulative rewards
    **for** each time step $t \in \{1, 2, \ldots, T\}$ **do**
        **for** each agent $i \in \{1, 2\}$ **do**
            Compute action $a_i \sim \pi_{\theta_i}(a_i \mid s_i)$ and log-probabilities $\log \pi_{\theta_i}(a_i)$
        **end for**
        Execute joint actions $\{a_i\}_{i=1}^{N}$ in environment $\mathcal{E}$
        Observe next states $\{s_i'\}_{i=1}^{N}$, rewards $\{r_i\}_{i=1}^{N}$, and done flags $\{d_i\}_{i=1}^{N}$
        Store $\{(s_i, a_i, r_i, d_i, \log \pi_{\theta_i}(a_i))\}_{i=1}^{N}$ in $\mathcal{D}_i$
        $R_{\text{episode}} \leftarrow R_{\text{episode}} + \sum_{i=1}^{N} r_i$
        Update current state $s_i \leftarrow s_i'$ for each agent $i$
        **if** done = True for all agents **then**
            **break**
        **end if**
    **end for**
    **for** each agent $i \in \{1, 2\}$ **do**
        Compute Generalized Advantage Estimation (GAE) $\hat{A}_i$:

$$\delta_t = r_t + \gamma V_{\phi_i}(s_{t+1}) \cdot (1 - d_t) - V_{\phi_i}(s_t)$$

$$\hat{A}_i(t) = \delta_t + \gamma\lambda \cdot (1 - d_t) \cdot \hat{A}_i(t+1)$$

        Compute returns $R_t = \hat{A}_i(t) + V_{\phi_i}(s_t)$
        Optimize PPO objective for policy $\pi_{\theta_i}$:

$$\mathcal{L}_{\text{policy}} = -\mathbb{E}\Big[\min\big(r_t \hat{A}_i, \text{clip}(r_t, 1-\epsilon, 1+\epsilon)\hat{A}_i\big)\Big]$$

    where $r_t = \frac{\pi_{\theta_i}(a_i|s_i)}{\pi_{\theta_i}^{\text{old}}(a_i|s_i)}$
        Optimize value loss for critic $V_{\phi_i}$:

$$\mathcal{L}_{\text{value}} = \frac{1}{T}\sum_{t=1}^{T}\Big(V_{\phi_i}(s_t) - R_t\Big)^2$$

        Apply entropy regularization to encourage exploration:

$$\mathcal{L}_{\text{entropy}} = -\frac{\mathbb{E}}{6}\big[\log \pi_{\theta_i}(a_i \mid s_i)\big]$$

        Update parameters $\theta_i$ and $\phi_i$ via gradient descent:

$$\mathcal{L}_{\text{total}} = \mathcal{L}_{\text{policy}} + c_1\mathcal{L}_{\text{value}} - c_2\mathcal{L}_{\text{entropy}}$$

    **end for**
    Log episode reward $R_{\text{episode}}$ and training metrics
**end for**

# 6 Results

## 6.1 Random Trajectory

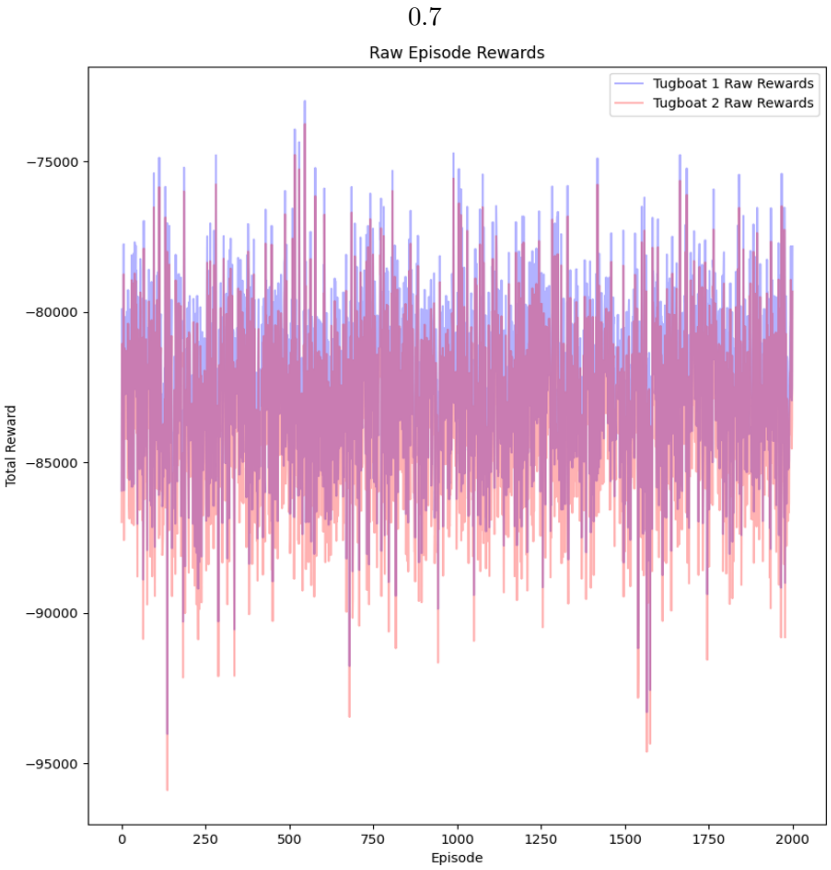
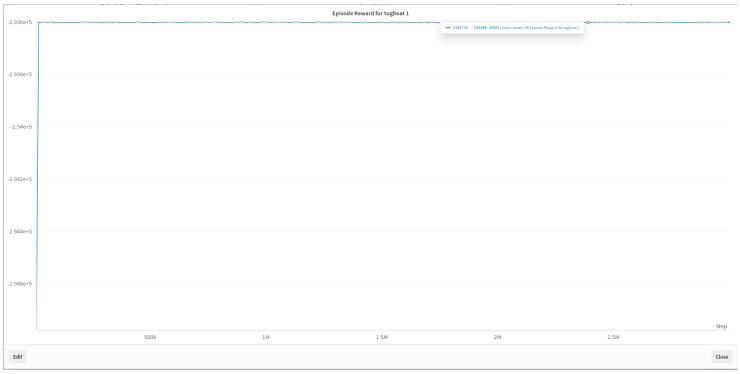



Figure 2: Reward accumulated by each agent for random trajectory.

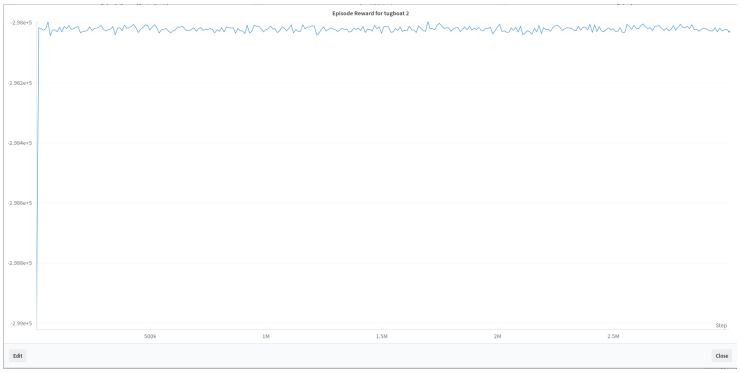
## 6.2 MADDPG

**Hyperparameters:**

- $\gamma = 0.99$
- $\alpha = 0.003$
- $\tau = 0.01$
- Training episodes = 300, with 2500 steps each

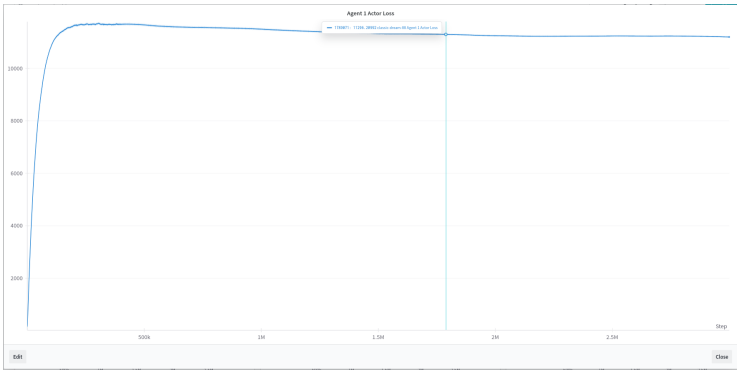



(a) Reward at each step for agent 1

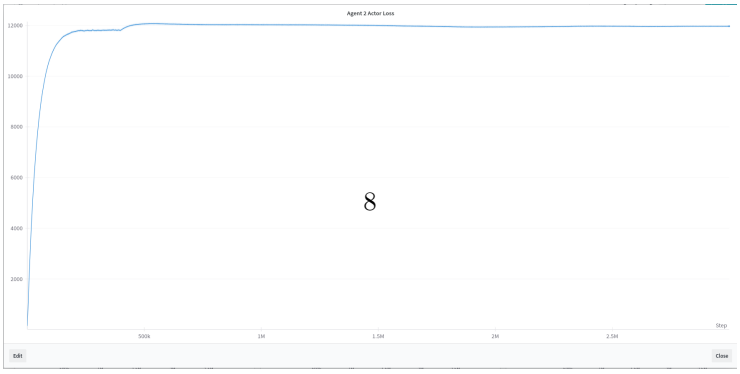



(b) Reward at each step for agent 2

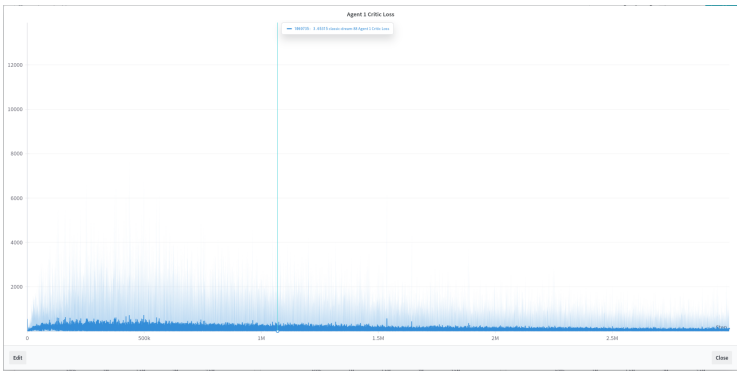
Figure 3: Reward accumulated by each agent



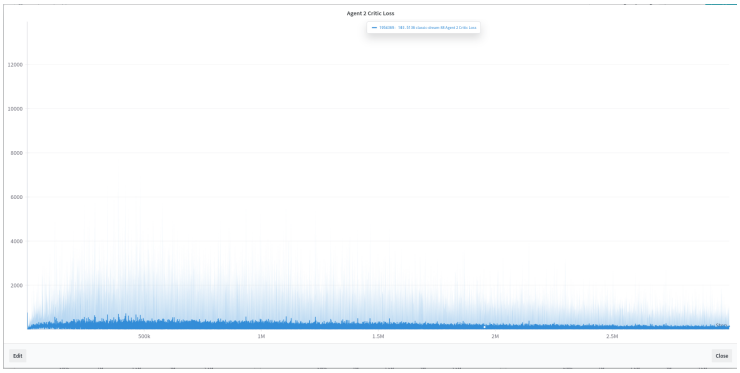

(a) Actor loss of agent 1




(b) Actor loss of agent 2




(a) Critic loss of agent 1




(b) Critic loss of agent 2

Figure 4: Critic loss of each agent each agent

The rewards accumulated by each agent are more in random trajectory as compared to MADDPG. The actor loss is not decreasing. The critic loss for each agent settled down but oscillating. From the results, it can be inferred that there is no convergence.

## 6.3 MAPPO

**Hyperparameters:**

- $\gamma = 0.99$
- $\alpha = 0.0003$
- $\epsilon$ (clip ratio) $= 0.2$
- $\lambda$ (GAE factor) $= 0.95$
- entropy $= 0.01$


Figure 6: Reward, entropy loss and value loss plots.

The entropy loss and value loss are increasing. The rewards accumulated by each agent are decreasing. The reward accumulated in random trajectory is higher than that of MAPPO. From the results, it can be inferred that there is no convergence.

# 7 Conclusion

The results of MADDPG and MAPPO are not satisfactory. The reward structure can be improved to improve the results. Randomly spawning the ship and the agents on the mapand training can also improve the results. After the results are improved, dynamic obstacles can be added and algorithms can be implemented.