IE4100R BENG DISSERTATION

A COLREGs-compliant multi-ship collision avoidance approach based on Deep Reinforcement Learning

He Zhenyu

DEPARTMENT OF INDUSTRIAL SYSTEMS ENGINEERING AND MANAGEMENT, NATIONAL UNIVERSITY OF SINGAPORE

(ACADEMIC YEAR 22/23 SEMESTER 2)

A COLREGs-compliant multi-ship collision avoidance approach based on Deep Reinforcement Learning

Submitted by
He Zhenyu

Department of Industrial Systems Engineering and Management

In partial fulfilment of requirements for
the Degree of Bachelor of Engineering
National University of Singapore

Acknowledgement

Throughout this project, I am extremely grateful for the advice given by my supervisor, Dr. Li Haobin. Especially through the beginning of this project when I was very uncertain about the directions of this project and what should be.

Also, Dr. Keramat Hasani is someone who has provided me with immense support and guidance throughout this project. Providing me with important and crucial feedback and suggestions during multiple occasions of meetings throughout my entire year of the project. Learning resources as well as technical examples shown to me by Dr. Keramat Hasani were also extremely helpful for me in my project. Thus, I would like to express my sincere appreciation for Dr. Keramat Hasani’s support and help throughout this year.

Someone I would also like to thank is Prof. Chew Ek Peng, while we were only able to speak on 2 occasions at the beginning of this project, Prof Chew provided me with valuable advice for me to start this project right.

Lastly, I would like to thank NUS for the offline and online resources that were useful in completing this project.

Table of Content

Summary

In maritime navigation, it is crucial for ships traveling at sea to be able to navigate safely and arrive to their destination in a timely and safe manner.

However, in most cases of maritime incidents of ship collisions, it is caused by human errors for not being able to correctly assess the situation and take appropriate actions to avoid collisions or high-risk encounters with other ships nearby.

To avoid such dangerous situations, this research aims to seek suitable autonomous methods through deep reinforcement learning such that ships can take autonomous actions and navigate in the safest way possible. While there are some autonomous methods that exist, this research aims specifically to not just avoid collisions, but to also abide by COLREGS rules which is a set of navigation regulations ships travelling at sea have to follow. Also, this project aims to seek a cooperative collision avoidance method in which ships work together to achieve collaborative safety instead of prioritizing their own safety and possibly putting other ships at risk.

In this research, a suitable Deep Reinforcement Learning Environment is developed, which takes into consideration multiple safety concerns during ship navigation all while avoiding any violations of COLREGS rules.

The validity of this research will be evaluated based on measuring the number of collisions that occur during millions of episodes of testing as well other factors such as the number of times violations of COLREGS rules occurred.

Lastly, based on the results of such measurements, the method proposed in this research can successfully avoid collisions at sea and avoid COLREGS rules violation for certain risky scenarios occurring at sea.

List of Figures

List of Tables

1. Introduction

In this section, the motivation for this research will be covered. As well as an introduction to existing collision avoidance methods at sea and a general introduction of COLREGS(Convention on the International Regulations for Preventing Collisions at Sea) rules relevant to this research. Algorithms and functions used in this paper will be based on COLREGS defined in this section.

1.1 Problem Description and Introduction to COLREGS

In a complex sea environment, traveling ships can be prone to high-risk encounters with other ships. While there are guidelines existing to respond to such encounters, these guidelines are intentionally vague, and interpretations are made based on personal subjectiveness and experiences. [1] Thus, causing sub-optimal decisions and therefore requiring autonomous methods which can fairly navigate through close ship encounters.

The COLREGS is a set of rules written for ships travelling at sea to comply with for navigation safety. A total of 41 COLREGS rules can be classified into six different categories below:

Rules 1-3 [General]

Rules 4-19 [Steering and Sailing]

Rules 20-31 [Light and Shapes]

Rules 32-36 [Sound and Light]

Rule 38 [Exemption]

Rules 39-41 [Verification of compliance]

The set of rules relevant to navigation actions and this research can be summarized into four different situations.

Situation 1: Head-on

In a head-on situation, defined by COLREGS rule 14, two vessels meeting on reciprocal paths involving collision risks should navigate to starboard and pass on the port side of the other.[2]

Situation 2: Overtaking

Defined by COLREGS rules 13,16,17, a ship is deemed to be overtaking when coming to another vessel more than 22.5° abaft its beam.[2]

Situation 3, Situation 4: Crossing

In crossing situations, defined by rules 15,16,17, the vessel which has the other on her own starboard side shall keep out of the way and shall, if the circumstances of the case admit, avoid crossing ahead of the other vessel.[2]

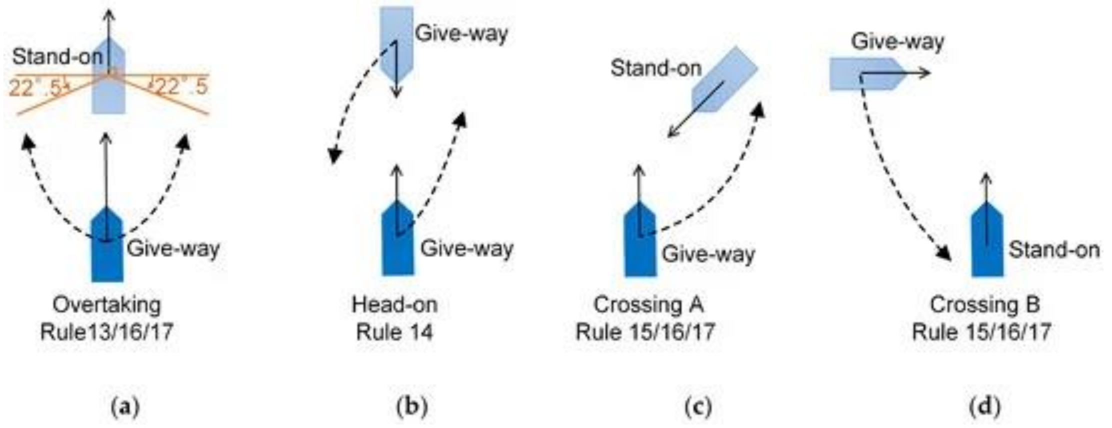


Figure 1.1 Illustration of COLREGS Rule 13-17[3]

1.2 Existing Methods

With regard to the problem aforementioned, several existing autonomous methods involving the use of (DRL)deep reinforcement learning have been implemented.
However, most of these methods had limited effectiveness due to either of the reasons listed below:

1) **Methods developed are made for 1-to-1 encounters**

   Early works on DRL-based ship collision avoidance were mostly developed based on 1-to-1 scenarios which are useful in determining the suitability of the implementation of DRL-based methods for ship collision avoidance. However, in real-life conditions, especially in complex waterways, numerous ships would need to avoid collision with each other at the same time. Thus models developed for 1-to-1 encounters would be ineffective in these scenarios.

2) **Methods developed did not consider COLREGS**

   Studies that did not consider COLREGS are able to avoid collisions but since ships navigating at sea should abide by COLREGS rules, these methods are not able to be put into practice

3) **Single-Agent (1-to-many) methods**

   Some studies are based on single-agent DRL models which focus on just one single ship trying to avoid every other ship instead of all the ships trying to avoid each other simultaneously. In a realistic event, ships would simultaneously take action and avoid each other in a cooperative manner.

Thus, due to the limitations of some existing methods, this research aims to research a suitable multi-agent DRL method that can be implemented for a multi-ship situation that complies with COLREG rules.

2. Literature Review

In this section, existing research and work will be looked into based on two different aspects. Learning algorithms used such as the use of model-based algorithms versus model-free algorithms, as well as learning environment set-ups, mainly on COLREGS interpretation and how rewards functions are defined.

2.1 Learning Algorithms

Among current studies, Model-Free learning algorithms such as DQN(Deep-Q-Network)[4] and PPO(Proximal Policy Optimization)[5] are mostly used.

Model-based algorithms were shown to be relatively easier to implement and have a faster learning rate in studies using MPC (Model Predictive Control) algorithm [6]. However, Model-Based methods react poorly to sudden unexpected changes in the environment [7]. Thus, in a ship collision avoidance context where the reaction to an unexpected situation is crucial, Model-Free learning algorithms are used in most studies.

Among Model-Free learning algorithms, both policy-based and value-based algorithms are common in most studies. There are no studies clearly pointing out which types of algorithms are more suitable for ship collision avoidance problems. General deep reinforcement learning studies on other topics have suggested that value-based algorithms like DQN tend to have a slower convergence and difficulty in learning complex environments, whereas policy-based algorithms like PPO have disadvantages such as tendency to converge to a local minimum, have a higher variance and require more computational power. [8]

However, suitability of different types of algorithms largely depends on the specific environment and problem. Differences in performance of different types of DRL algorithms will be discussed in later sections.

2.2 Environment

Different learning algorithms mentioned in the previous section cannot be fairly compared due to their different learning environment set-up such as how reward functions are defined or how a collision is defined.

In deep reinforcement learning, an agent interacts with the environment, takes actions, and changes the state of the environment which the environment then returns a reward and the new environment state for the agents' action taken. How such agent-environment interactions are defined affects the training of DRL agents and their suitability to be used in collision-avoidance scenarios.

In terms of observation space, observation spaces used in different studies include images of a Top-Down view, relative position and states of other ships, or relative position and states of the closest ship.[10] Observation spaces do not affect the learning results of the DRL agents but overly large observation could cause unnecessarily long convergence time and learning time for the agent.

Most existing studies differ in terms of the reward functions defined for their environments. Since COLREGS does not have a standard risk evaluation method, the environment needs a suitable definition in terms of a collision and risk level to learn from a suitable reward function.

2.3 Risk Assessment

In real life conditions where ships are navigating close to each other, the safety of ships are not just determined by the occurrences of collision. It is important to assess its risk of collision with other ships nearby to prevent collision as well as high risk situations when travelling near other ships. So that even before collision happens, ships can be at a safe distance for other ships. Without assessing risk to other ships, ships may navigate dangerously close to other ships as long as there are no collisions occurring. Therefore, risk assessment would be needed in most collision avoidance studies with real-life implementations.

In existing studies mostly use two different methods for risk assessment. Using either the domain of the ship[11] or the CPA(Closest Point of Approach) between ships[12] or both[10].

The domain of the ship refers to a specified area around the ship and calculates collision risk based on distance once another ship enters its domain.

In comparison, CPA includes DCPA (Distance at Closest Point of Approach) and TCPA(Time at Closest Point of Approach), which is the closest point between two ships based on the ship's current speed and directions.

Studies that combine both methods start calculating CPA only when another ship enters its domain[10].

Overly complex risk assessment methods paired with a large observation space and action space can cause difficulty in agent convergence as mentioned in previous sections. The computational power needed would also be excessively high[10]. The effects of a different risk assessment approach will be discussed in the *Methodology* and *Results* Section.

3. Methodology

In this section, the methodology used in this research will be covered. Covering the environment set-up, reward function, agent-environment interaction, and DRL algorithms used.

3.1 Environment setup

As covered in the previous section, the environment setup is important as it affects the rate of convergence for the DRL agent and whether the agent will convert to a local optimal or have any undesired behaviors.

3.1.1 Observation Space

In this research, a [*n*,9] array(*n*, number of ships) is used as the observation space for the environment after testing against environments that use image input as the observation space.

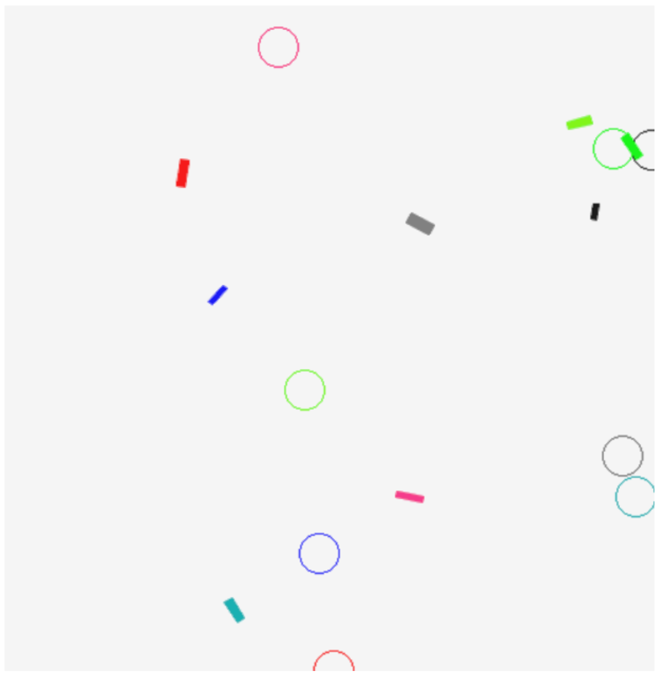


Figure 3.1 Snapshot of Image used for Training

Environments using a top-down image input as observation spaces were unable to show any convergence or learning when tested with different DRL algorithms and neural networks within 1 million episodes. Due to the difficulty of learning information such as speed using an image. Observation space testing is done with simplified reward spaces where agents are only rewarded for reaching their destination.

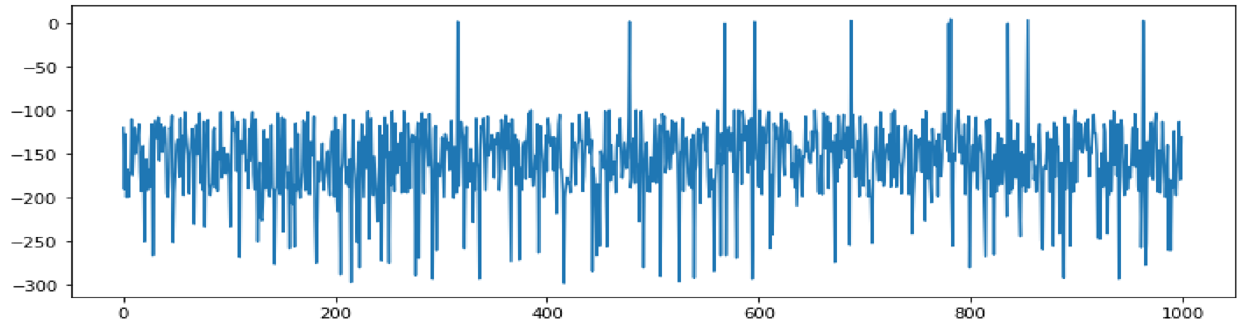


Fig 3.2 DQN Model with Image Observation Space (1 Million Timesteps)

DRL Algorithms	Convergence	Timesteps	Number of Ships Tested
DQN	No	10 million	2,4
A2C (Actor-Critic)	No	10 million	2,4
PPO	No	10 million	2,4
A2C (LSTM)	No	10 million	2,4

Table 3.1 Convergence of DRL algorithms with Image Observation Space

With a [*n*,*9*] observation space, algorithms were able to show signs of convergence within 500 thousand episodes when tested with a simple two-ship scenario and simplified reward functions.

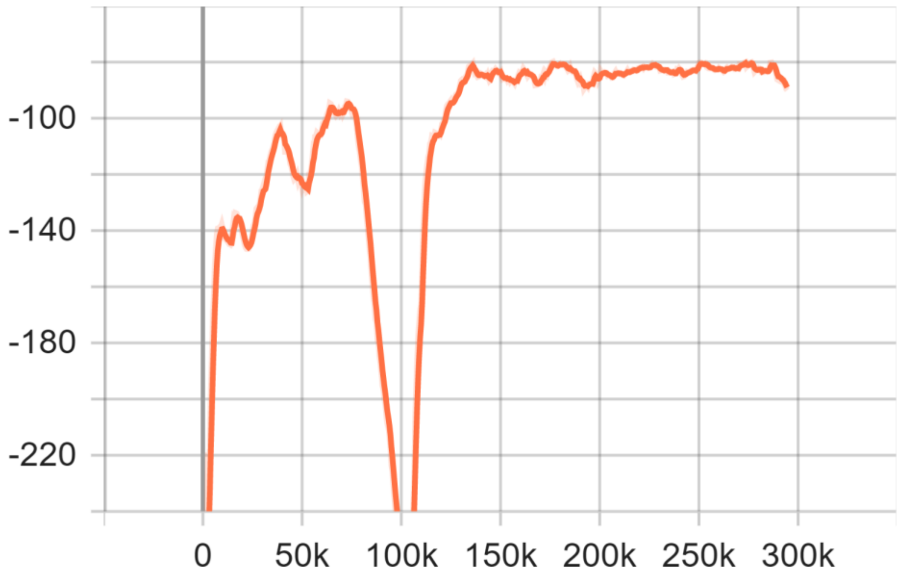


Fig 3.3 A2C Model with [n,9] observation space

In the [*n*,*9*] observation space, *n* represents the number of ships tested in the environment. Within each row, the observation space contains the following state information of each ship:

1. x-coordinate

2. y-coordinate
3. Speed
4. Length of Ship
5. Width of Ship
6. x-coordinate of destination
7. y-coordinate of destination
8. Bearing (Ships Heading Angle)
9. Status (Boolean value indicating whether the ship has arrived at its destination)

3.1.2 Action Spaces

Action spaces are the possible actions that can be taken by the agent at each timestep that changes the state of the environment.

In this research, the following actions can be taken by the agent:

1. Steer Anti-clockwise
2. Steer Clockwise
3. Accelerate
4. Decelerate
5. No action

After each time step, the location of the ships changes according to actions taken during the time steps.

In realistic conditions, resultant speed and position are affected by factors such as its initial momentum or draft force at sea. For example, accelerating by χ at will not lead to an increase in speed by χ immediately and exactly in the next time step. However, in this research, actions taken are assumed to have an immediate and direct effect on the ships' speed and position to avoid complications in this research.

3.1.3 Reward Functions

Reward values are values fed to agents by the environment after each time step and where the agent learns about the environment.

The reward functions are the main factors that determine the desired behavior of the agents and the learning process of DRL algorithms. Thus defining suitable reward functions will be the main focus of this research.

Poorly defined reward functions will cause the agent to converge to an local optimal or exhibit certain undesired behaviors which will be covered later on in the *Results* section.

The suitability of the reward functions is evaluated based on the following data collected from testing and training:

1. **Ability to converge within 30 Million timesteps**

   Considering available computational power and training time available for this research, training timesteps are limited to 30 Million timesteps(~16 hours in training). Agents that are unable to learn within 30 Million timesteps may have too complicated reward functions for this research.

2. **Convergence to a local optimal**
3. **Exhibit undesired behaviors**

   Some reward functions results in the agent learning unexpected policies that converge to local optimal and exhibit undesired behaviors. Such cases will be covered in the *results* section.

4. **The number of collisions that occur during 1 Million random test cases**

   Avoiding collisions is the main aim of this research, thus collisions that occur during test runs should be as close to zero as possible.

5. **The number of arrivals during 1 Million random test cases**

   Certain reward functions will cause the agent to be too avoidant of collisions to a point where agents do not care about arrivals. More will be discussed in the *Results* section. Thus, good reward functions should see all the ships arriving at their destination safely.

6. **The number of COLREG violations during 1 Million random test cases**

   Abiding by COLREG rules is another aim of this research, thus violations that occur during test runs should be as close to zero as possible.

7. **Timesteps taken for all ships to arrive at the target location**

   To avoid ships taking sub-optimal or unnecessary long paths to get to their destination, the time taken for ships to arrive at their target location would also be considered.

The following reward functions will be tested in different combinations with each other and evaluated based on their performance. Unsuitable reward functions will be eliminated and the finalized reward functions will be discussed in the *Results* section.

1. **Penalty for collision**

   A range of penalty values from -1 to - 100 are tested to find a suitable penalty value that does not overly prioritize collision avoidance but prioritizes it enough to navigate to the target location safely.

2. **Penalty for time steps taken**

   Different time steps penalties are tested to avoid ships taking sub-optimal or unnecessary long paths. The penalty should not be too large such that it disregards navigation safety in order to arrive quicker.

3. **Penalty for Collision Risks based on DCPA, TCPA (Risk Assessment/Risk Factor)**

   DCPA and TCPA are important factors as it evaluates non-collision risks when ships are travelling near each other, especially in real-life conditions where the such evaluation of non-collision risk are important for safe navigation.

   To calculate DCPA and TCPA , it would require the x-coordinate, y-coordinate , heading angle , and speed of two ships.

   Let x_i, y_i, θ_i, s_i be x-coordinate, y-coordinate , heading angle , and speed of ship i.

$$D_x = x_1 - x_2$$
$$D_y = y_1 - y_2$$
$$V_x = s_2 \cdot cos(\theta_2) - s_1 \cdot cos(\theta_1)$$
$$V_y = s_2 \cdot sin(\theta_2) - s_1 \cdot sin(\theta_1)$$

$$\text{TCPA} = -(D_x \cdot V_x + D_y \cdot V_y) \div (V_x{}^2 + V_y{}^2)$$
$$\text{DCPA} = \sqrt{(D_x + V_x \cdot TCPA)^2 + (D_y + V_y \cdot TCPA)^2}$$

4. **Penalty for COLREG Violation, Different Penalties for Different Violations**

5. **Penalty for COLREG Violation, Same Penalty for All Violations**

   Having different penalties for different COLREG violations may not be needed and may lead to overly complex reward functions. Penalty 4 and Penalty 5 are tested to determine how COLREG Violations should be penalized.

6. **Penalty for Moving Further Away from Target Location**

This penalty may be needed to avoid sparse rewards[14] which refers to a type of reward signal in reinforcement learning where the agent receives a reward only when it achieves a particular goal or reaches certain milestones(arrival or collision), rather than receiving a reward for every action it takes. Sparse rewards can make it challenging for the agent to learn the desired behavior because it may not receive enough feedback to learn which actions are most effective in achieving the goal. This can lead to slower learning or the agent getting stuck in a suboptimal policy.

7. **Reward for Arriving at Target Location**

   Similar to the penalty for collision, values from -1 to - 100 are tested to find a suitable reward value such that it does not over or under-prioritize arriving at the target location.

8. **Reward for Moving Closer to Target Location**

   Similar to Penalty 6, this reward function may be needed to avoid sparse rewards[14].

9. **Reward for staying on Course**

   Although not part of the COLREGS rules used in this research , the COLREGS advices against changing courses and speed unnecessarily when not trying to avoid collision.

10. **Penalty, Same Ship Domain for all Ships**

11. **Penalty, Different Ship Domain based on Ships Size and Speed**

    As discussed in the *Literature Review* section, some studies adopt the ship domain method to evaluate non-collision risk when ships enter other ships' domains. To determine which method of ship domain assessment is suitable for this research, both Penalty 10 and Penalty 11 are tested.

3.2 Alternative Method: Action Masking

The aim of this research is to find a DRL method for multiship collision avoidance while abiding by COLREGS rules. There are two methods to abide by COLREGS.
One method is to penalize agents for violating COLREGS rules as described previously. Another method is to restrict the agents from performing actions that violate COLREGS rules entirely. Thus action masking will be performed and agents will not be able to perform actions that violate COLREGS rules.

Using the same DRL algorithm and environment as the first method, the action masking method was able to converge to the same level but faster than the first method.
However, due to the nature of this research, where collisions have much higher consequences than violating COLREGS rules in real-life situations, action masking on COLREG violations would be unsuitable. Agents should be, able to violate COLREGS rules in order to prevent collisions in real-life situations.

3.3 Neural Network, Algorithm Parameters, and Training

In this research, off-policy(DQN)[13], on-policy(PPO)[14], and mixed-policy(A2C)[15] are implemented to test its suitability for a multi-ship collision avoidance situation.

All of the DRL training was done with a 2-layer [64,64] Multi-layer Perceptron neural network with the respective hyperparameters listed below. Hyperparameters unspecified in the following table follow the implementation available in Appendix C,D and E.

<u>DQN</u>

Learning Rate	0.0001
Warm-up Steps	50000
Batch Size	32
Discount Factor	0.99
Training Frequency	4
Gradient Steps	1
Epsilon Decay	0.3
Target Network Interval	10000

Table 3.2. DQN hyperparameters

<u>A2C</u>

Learning Rate	0.0007
Update Steps	5
Discount Factor	0.99
Gradient Clipping	0.5
Entropy coefficient	0.1
Value function coefficient	0.5

Table 3.3 A2C hyperparameters

<u>PPO</u>

Learning rate	0.0003
Update Steps	2048
Batch Size	64
Epochs	10
Discount factor	0.99
Gradient Clipping	0.5
Entropy Coefficient	0.1
Value coefficient	0.5

Table 3.4 PPO hyperparameters

Training episodes were trained using a four-ship environment where each ship's position and destination as well as its speed and size are randomly reset after each episode.

Initial training episodes were done with an eight-ship environment but the DQN model was unable to fully converge within 10 Million timesteps as observation space and action space may be too complex and require more than 10 Million timesteps to train.

For example, a four-ship environment has an action space array of length = 4, each index ranges from 0-4. There are 625 total possible combinations for the action output by the agent. When it increases to five-ships, possible combinations increase to 3125. If the number of ships increases, the action spaces and observation spaces required to learn and explore increase exponentially. Thus for the purpose of computational power and time in this research only, scenarios are trained and tested using a four-ship environment.

In DQN, the agent selects an action based on an epsilon-greedy policy[8] where the agent has a chance of *epsilon* to select a random action instead of selecting an action that maximizes the return. Whereas *epsilon decay* is the rate at which the *epsilon* value decreases. Thus a higher *epsilon decay* value would mean that the *epsilon* value decreases slower and the agent would take more random exploration during the training process.

When training in a four-ship environment, *epsilon decay* less than 0.3 was unable to properly train the DQN agents as the agents stopped exploring before they can fully learn the environment.

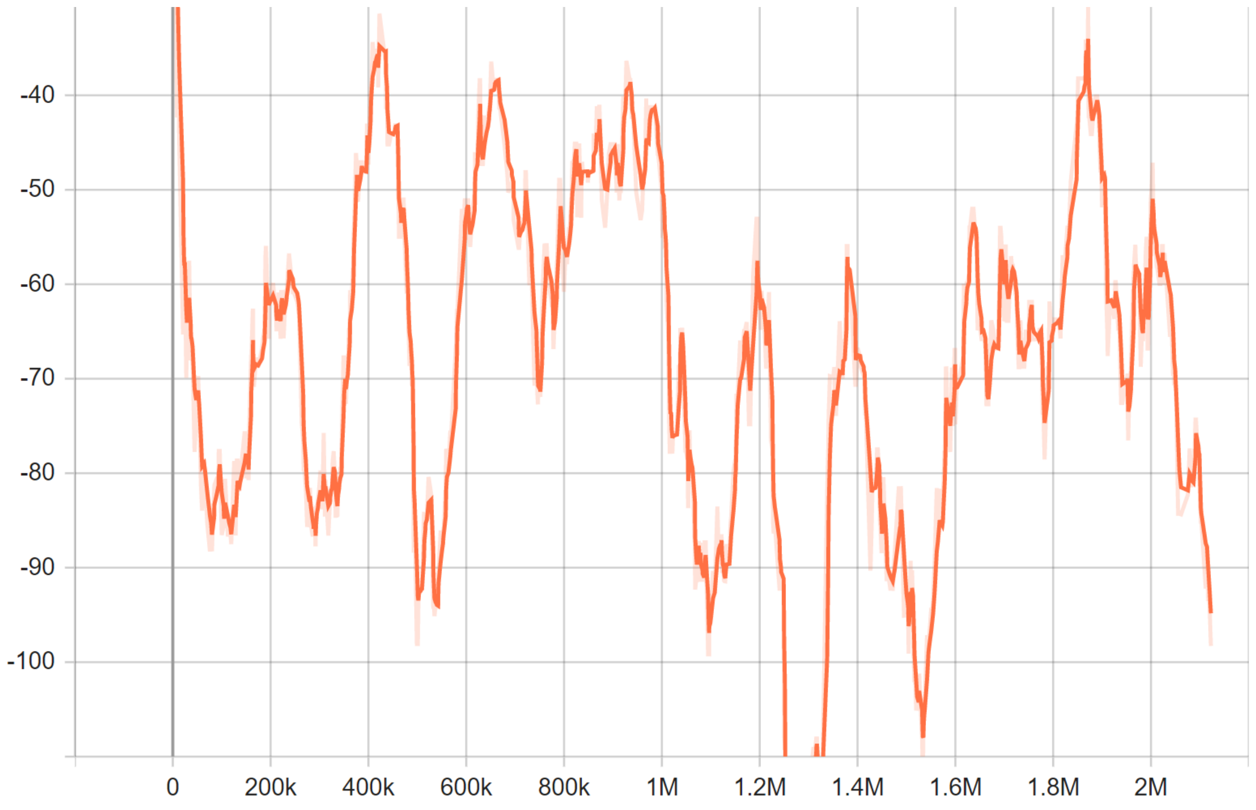


Fig 3.4 DQN policy with epsilon decay = 0.1

DQN agent was only able to converge after *epsilon decay* increased to 0.3.

Similarly, for PPO and A2C, their agents were unable to converge successfully regardless of the reward functions used. Both PPO and A2C were seen to converge into a local optimal due to insufficient exploration during training. Thus the entropy coefficient was increased to 0.1 to allow for more exploration by the agent.

For detailed training progress of hyperparameters tuning, training results are available in Appendix A.

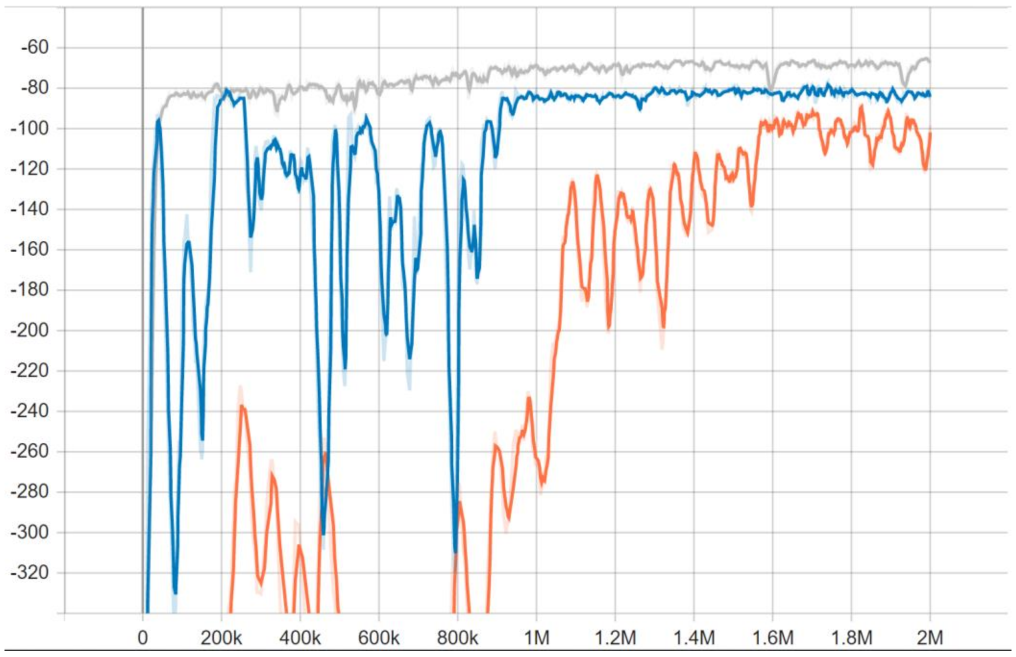
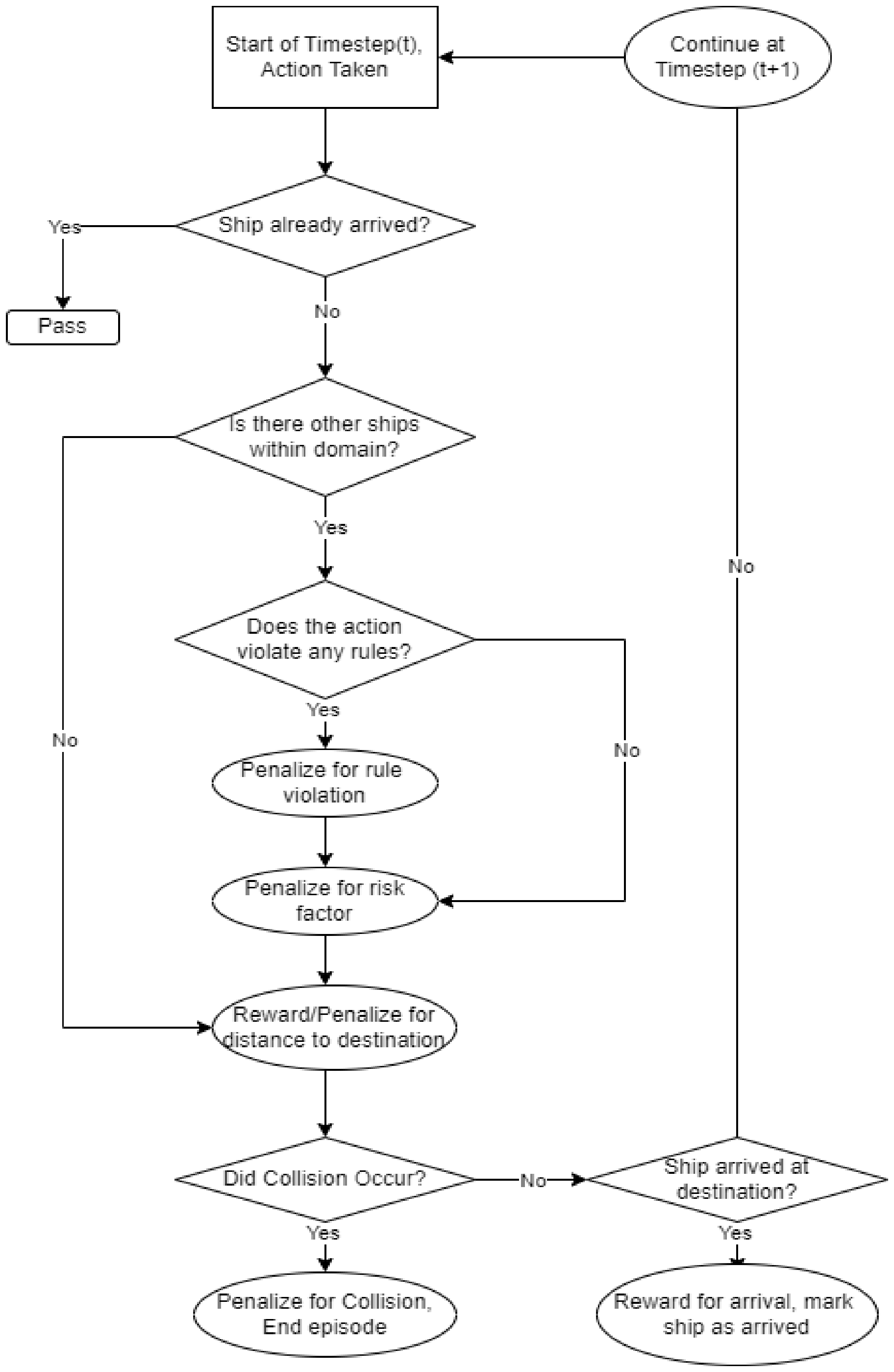


Fig 3.5 PPO, A2C , DQN Algorithm with 2 Million timesteps with simplified reward

With the algorithms' hyperparameters finalized for fair testing, reward functions discussed previously were repeatedly tested using different reward values. The outcome will be discussed in the next section.

4. Results

The final reward functions that lead to the most optimal behavior are as shown in this diagram.



Start of Timestep(t), Action Taken
Continue at Timestep (t+1)
Ship already arrived?
Yes
Pass
No
Is there other ships within domain?
Yes
No
Does the action violate any rules?
Yes
No
Penalize for rule violation
Penalize for risk factor
Reward/Penalize for distance to destination
Did Collision Occur?
No
Ship arrived at destination?
Yes
No
Yes
Penalize for Collision, End episode
Reward for arrival, mark ship as arrived

3.6 Flowchart of Final Reward Functions at each Timestep

As discussed in the *Methodology* section, reward functions that are too complex or unnecessary for the agent to achieve optimal behavior will be eliminated.

In the following segment, reward functions that lead to suboptimal behavior, undesired behavior, overly complex reward function, and unnecessary reward functions will be discussed.

4.1 Local Optimal and Undesired Behavior

During training and testing, agents that were unable to achieve desired performance exhibited one of the following behaviors during visual rendering.

1. **Perpetually moving in away from other ships**

In this case, the ships were observed to move away from other ships and maintain that direction until the end of episodes. With no change in direction or speed after turning away from other ships.

Agents exhibited this behavior when penalty values for collision are significantly higher than the reward values given for arrival at the target location. Thus the agent converges to a local optimal where agents avoid any chances of collision penalty but also never arrive at the target location. After repeated training experiments with different combinations of reward and penalty values, agents were able to improve on this when reward values and penalty values for arrival and collision are of equal magnitude.

[insert picture of the flat line , moving away graph]

2. **Perpetually moving in circles**

This is another occurrence of a local optimal when the reward functions for moving closer to the target location are included. This caused the agent to exploit this reward by moving closer to the target and then moving away from the target and then moving closer to the target again repeatedly to indefinitely receive the reward for moving closer until the episode ends.

Reward and Penalty values for moving closer and away from the target are adjusted repeatedly but this problem persisted thus this reward function is removed from the final reward functions. However, this reward function was initially introduced to prevent sparse rewards[14] as discussed in the *Methodology* Section. Removing this reward function led to difficulty in learning for the agent thus a similar reward function needs to be added in order to provide constant feedback for the agent's actions and prevent sparse rewards. By including a reward function that penalizes the agent based on current distance to the target instead of moving closer compared to the last time step, the training was able to avoid sparse reward and getting stuck in this local optimal.

3. **Collides intentionally with other ships**

In cases where the time step penalty for each time step is too high relative to the collision penalty , agents learnt to intentionally collide with other ships in order to end the episode early. For example,

Time Step Penalty	-0.1 per time step
Time Steps per Episode	4000
Collision Penalty	-10

Table 4.1 Timesteps and Collision Penalty

A full exploration without any collision or arrival would result in a final reward value of -400 whereas an immediate collision incurs a penalty of -10 and immediately ends the episode.
To avoid this, time step penalty were decreased and a time step penalty of -0.005 was tested to be suitable for the environment used in this research.

4.2 Overly Complex Reward Functions

As explained in the Methodology section, training time steps are limited to 30 million timesteps in research due to limitations in computational power and time available. Thus the following reward functions are removed as it was too complicated for the agent to learn within 30 million steps.

1. Different Penalties for Different COLREG Violations
2. Reward for Staying on Course
3. Different Ship Domain for Ships of Different Sizes and Shapes

However, this does not mean that such reward functions are not suitable for multi-ship collision avoidance agents. It is that including such reward functions may take much longer to train the agent and its suitability cannot be concluded in this research.

Training progress is available in Appendix B.

4.3 Evaluation

To evaluate the effectiveness of the trained models, the models were evaluated for 1 million random episodes and measured the number of collisions that occurred during testing, average reward, and timesteps taken for all ships to arrive at their destination.

Over 1 million random episodes of a four-ship environment:

Model	Average reward	Average Collision	Average Time steps per episode average	Average COLREG Violations per episode (Violating multiple rules at the same instance counts as 1)	Average Number of Ships reach before episode end
PPO	556.6714	0	189.7677	0.7338	3.9914
A2C	548.4162	0.000002	190.1101	0.8211	3.9892
DQN	526.94515	0	190.277	0.7927	3.9911

Table 4.2 Testing Result

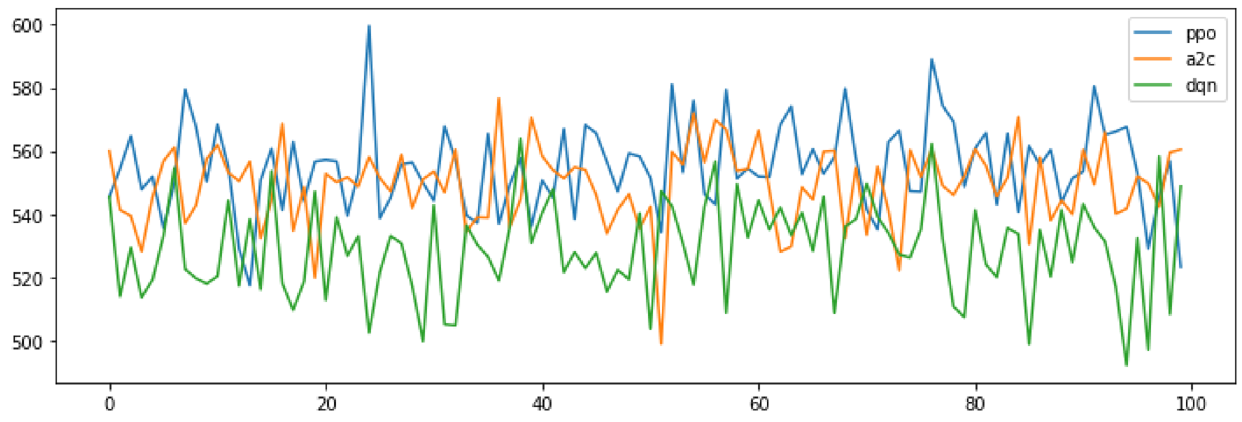


Fig 4.1 Average reward

Table 4 includes the testing result of successful training with finalized reward functions. Training cases of other reward functions which failed to achieve optimal solutions are available in Appendix B.

To test the validity of training methods when applied to other environments, the same training methods were repeated with 3,4 and 5-ship environments.

As discussed in the previous section, action masking methods may not be suitable due to real-life implementation of ship-collision studies.

However, for the purpose of this research, the action masking method was tested directly against a regular model with no action masking.

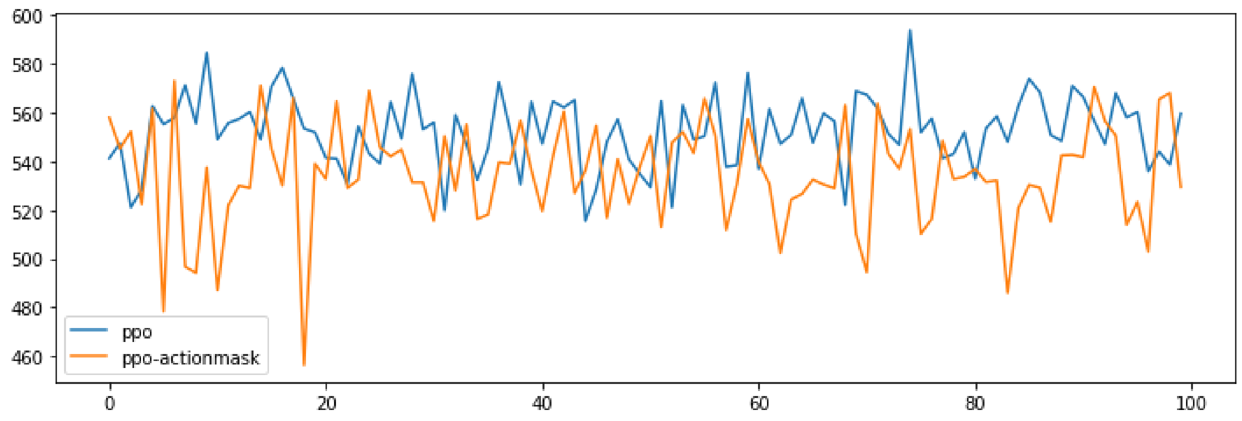


Fig 4.2 Action masking vs Regular Environment

Model	Average reward	Average Collision	Average Time Taken	Average COLREG Violations per episode (Violating multiple rules at the same instance counts as 1)	Average Number of Ships reach before episode end
PPO	553.398	0	188.8365	0.7925	3.9833
PPO-Actionmasking	523.4623	0.00074	184.3314	0.0239	3.9734

Table 4.3 Action Masking Comparison

When compared to a regular environment, the action-masked environment has a 5.7% lower average reward when using the same DRL algorithm. It has a slightly higher collision rate but collision avoidance is the focus of this research and 74 counts of collision in a million episodes show that action-masked methods would not be suitable for ship collision avoidance studies. There are still some counts of violations despite action masking due to some occurrences where episodes randomly reset at a state where violations are already occurring at the initial steps.

5. Conclusion and Recommendation

In this research, a suitable learning environment for multi-ship collision avoidance was proposed and different DRL methods were compared to by training and testing on this environment.

To create an environment suitable for learning, this research first looked at an observation space that provides enough information for DRL agents to learn from but is not too complex so that agents can learn and converge efficiently.

Determining a suitable reward functions and values were the main focus of this research. Different reward functions and values repeatedly experimented with, such that DRL agents can achieve a desired behavior and converge within 30 Million time steps as defined in this research.. Rewards such as ship domains and separate COLREGS violation penalties used in other existing studies for single-ship environments were found to be too complex for a multi-ship environment when the agent fails to converge within 30 million episodes. Methods to avoid poor convergence and undesired behaviours were also discussed in this paper such as the magnitude of different reward values relative to one another.

Different methods to achieve COLREGS compliance were also looked into. While no existing studies use the action masking method discussed in this research due to impracticality in real-life scenarios, this research looked into the effect of action masking can have in a theoretical situation. This concluded that the action masking method learns similarly well as regular penalty methods. It performs poorer in collision avoidance but performs much better in abiding by COLREGS rules.

Different types of DRL algorithms were compared to, and the on-policy method (PPO) performed best both in terms of learning convergence and actual testing.

However, the proposed environment and algorithms have certain limitations that can be and should be improved on.

As stated repeatedly in this research, training time steps were limited to 30 million time steps due to limited computational power and training time. Some factors in this research were evaluated based on the fact that they were not suitable for training within 30 million time steps. However, further studies and testing can be done with longer training time steps and may have a better collision avoidance than this research which is done under a 30 million time steps constraint.

As mentioned in the *Methodology* Section, the actions taken by the ships are assumed to have an immediate and direct impact on the resultant position and velocity. However, this assumption can be unrealistic as explained in the *Methodology* sections. Thus, improvement can be made by removing this assumption. Further work can be done with more realistic physics calculations for the ship's resultant position and velocity based on the actions taken. For example, considering drag force and momentum , hydrodynamic forces [16] and other factors like water resistant[10] to achieve realistic results.

Another improvement that can be made in this research is to study more complex environments with more ships. This research is primarily tested with scenarios of four ships avoiding collision with one another. Simpler scenarios with fewer ships or more complex scenarios with many more ships may result in a different conclusion than this research. Especially in terms of suitable DRL algorithms. As explained in the *Literature Review* section, some algorithms work better with less complex scenarios while other algorithms work better with more complex scenarios.

Also, since most real-life maritime accidents involving collisions occur in restricted waters with different conditions and rules, further work can be done on studying collision avoidance in certain restricted waters with different conditions and other rules to abide by.

References

[1]
ASHIM KUMAR DEBNATH, “TRAFFIC-CONFLICT-BASED MODELING OF COLLISION RISK IN PORT WATERS,” PHD Thesis, National University of Singapore, 2009.

[2]
“Convention on the International Regulations for Preventing Collisions at Sea, 1972.” International Marine Organization, 2018.

[3]
H. Lyu and Y. Yin, “Fast Path Planning for Autonomous Ships in Restricted Waters,” *Applied Sciences*, vol. 8, no. 12, p. 2592, Dec. 2018, doi: 10.3390/app8122592.

[4]
H. Shen, H. Hashimoto, A. Matsuda, Y. Taniguchi, D. Terada, and C. Guo, “Automatic collision avoidance of multiple ships based on deep Q-learning,” *Applied Ocean Research*, vol. 86, pp. 268–288, May 2019, doi: 10.1016/j.apor.2019.02.020.

[5]
M. Abdelaal and S. Schön, “Predictive Path Following and Collision Avoidance of Autonomous Connected Vehicles,” *Algorithms*, vol. 13, no. 3, p. 52, Feb. 2020, doi: 10.3390/a13030052.

[6]
T. A. Johansen, T. Perez, and A. Cristofaro, “Ship Collision Avoidance and COLREGS Compliance Using Simulation-Based Control Behavior Selection With Predictive Hazard Assessment,” *IEEE Trans. Intell. Transport. Syst.*, vol. 17, no. 12, pp. 3407–3422, Dec. 2016, doi: 10.1109/TITS.2016.2551780.

[7]
E. F. Camacho and C. Bordons, *Model Predictive Control*. London: Springer London, 1999. doi: 10.1007/978-1-4471-3398-8.

[8]
R. S. Sutton and A. G. Barto, *Reinforcement learning: an introduction*, Second edition. Cambridge, Massachusetts: The MIT Press, 2018.

[9]
H. Mao, M. Alizadeh, I. Menache, and S. Kandula, “Resource Management with Deep Reinforcement Learning,” in *Proceedings of the 15th ACM Workshop on Hot Topics in Networks*, Atlanta GA USA, Nov. 2016, pp. 50–56. doi: 10.1145/3005745.3005750.

[10]
D.-H. Chun, M.-I. Roh, H.-W. Lee, J. Ha, and D. Yu, “Deep reinforcement learning-based collision avoidance for an autonomous ship,” *Ocean Engineering*, vol. 234, p. 109216, Aug. 2021, doi: 10.1016/j.oceaneng.2021.109216.

[11]
M. Junmin *et al.*, “Mechanism of dynamic automatic collision avoidance and the optimal route in multi-ship encounter situations,” *J Mar Sci Technol*, vol. 26, no. 1, pp. 141–158, Mar. 2021, doi: 10.1007/s00773-020-00727-4.

[12]
K. Woerner, M. R. Benjamin, M. Novitzky, and J. J. Leonard, “Quantifying protocol evaluation for autonomous collision avoidance: Toward establishing COLREGS compliance metrics,” *Auton Robot*, vol. 43, no. 4, pp. 967–991, Apr. 2019, doi: 10.1007/s10514-018-9765-y.

[13]
V. Mnih *et al.*, “Playing Atari with Deep Reinforcement Learning.” arXiv, Dec. 19, 2013. Accessed: Feb. 07, 2023. [Online]. Available: http://arxiv.org/abs/1312.5602

[14]
J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, “Proximal Policy Optimization Algorithms.” arXiv, Aug. 28, 2017. Accessed: Feb. 07, 2023. [Online]. Available: http://arxiv.org/abs/1707.06347

[15]
V. Mnih *et al.*, “Asynchronous Methods for Deep Reinforcement Learning.” arXiv, Jun. 16, 2016. Accessed: Feb. 07, 2023. [Online]. Available: http://arxiv.org/abs/1602.01783

[16]
Y Nakiri and K Kijima, “Prediction method of ship manoeuvrability in deep and shallow waters,” 1990.

Appendix A: Hyperparameter Tuning

Appendix A shows some of the hyperparameters settings training result mentioned in *Methodology* section that failed to converge within a maximum of 30 million steps.

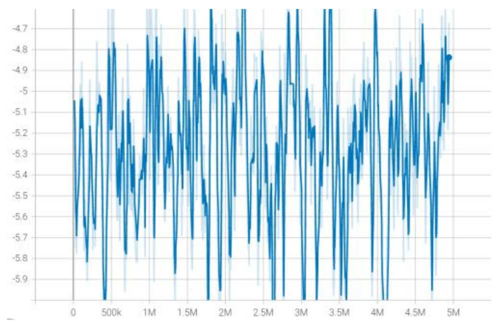


Fig A.1 PPO algorithm with ent_coef =0, Clipping = 0.2

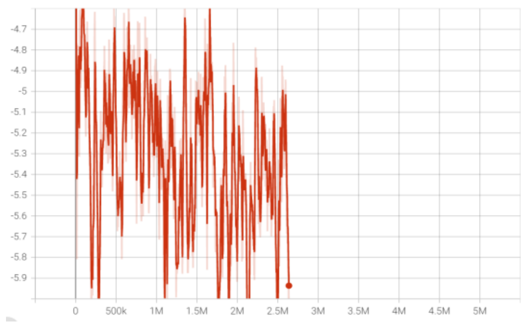


Fig A.2 PPO algorithm with ent_coef =0.05, Clipping=0.2

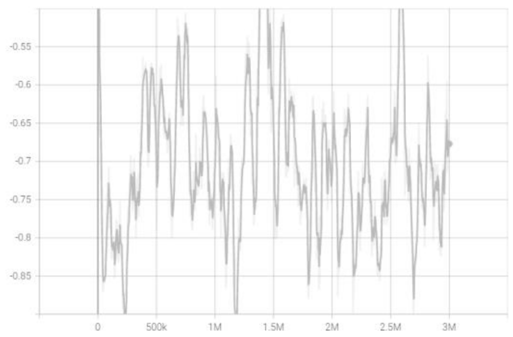


Fig A.3 PPO algorithm with ent_coef = 0 , Clipping = 0.1

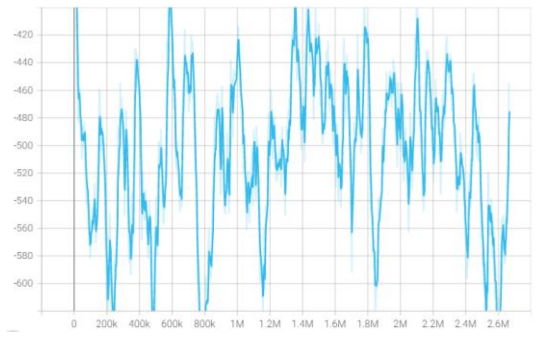


Fig A.4 DQN algorithm with Epsilon = 0.2

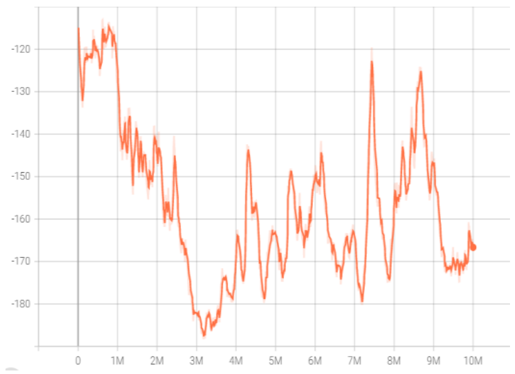


Fig A.5 A2C Algorithm with ent_coef = 0

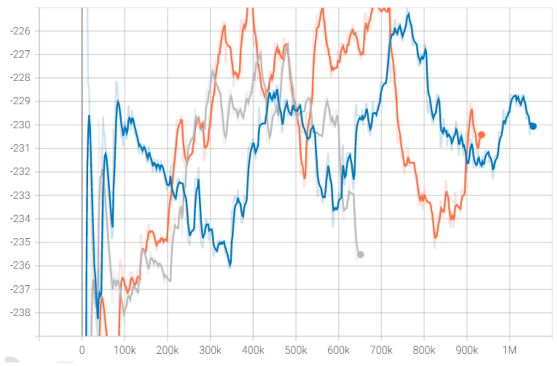


Fig A.6 Collated results of other hyperparameters

Appendix B: Training Progress for Different Reward Functions

Appendix B shows the training progress of some reward functions which were deemed unsuitable in this research. Specifically, those that arrive at a certain local optimal or undesired behavior as described in *Methodology section.*

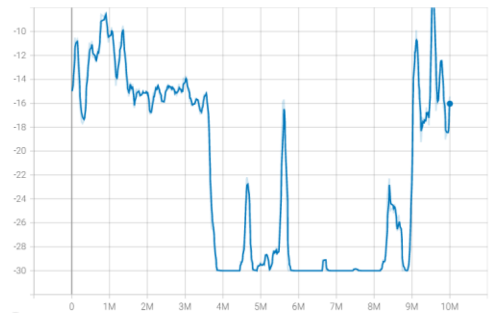


Fig A.7 Undesired behavior, intentionally colliding

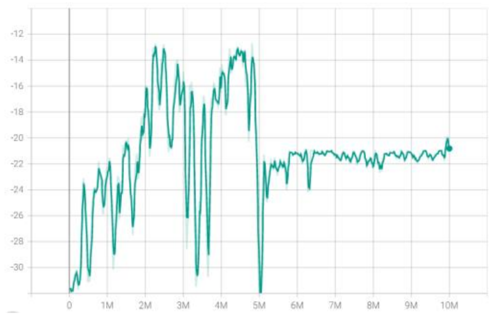


Fig A.8 Local optimal, Perpetually moving in circles

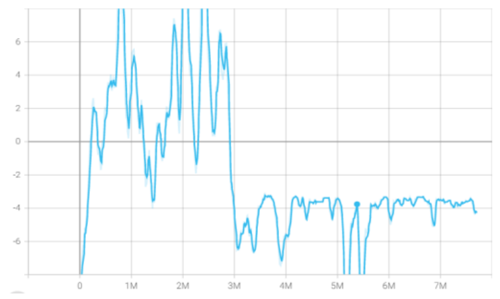


Fig A.9 Local optimal, perpetually moving away from other ships.

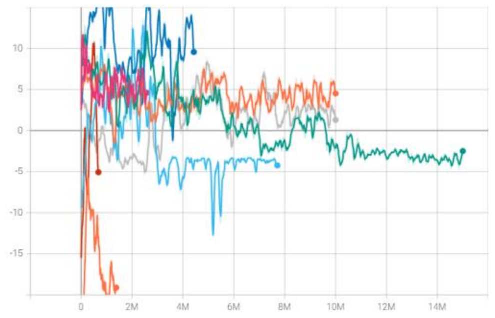


Fig A.10 Collated results of other reward functions.

Appendix C: DQN pseudocode

DQN algorithm used in this research is based entirely on this paper[13]

```
1. Initialize replay memory D to capacity N
2. Initialize action-value function Q with random weights
3. Initialize target action-value function Q' with the same weights as Q
4. Set hyperparameters:
    1. learning rate alpha
    2. discount factor gamma
    3. exploration rate epsilon
    4. replay memory batch size B
    5. target network update frequency C
5. For each episode:
    1. Observe initial state s
    2. Repeat until episode ends:
    3. With probability epsilon, select a random action a
    4. Otherwise, select action a = argmax Q(s, a; theta) with current action-value
       function Q
    5. Execute action a in the environment, observe reward r and next state s'
    6. Store transition (s, a, r, s') in replay memory D
    7. Sample a minibatch of transitions (s, a, r, s') from D
    8. Compute target values y_j for each transition in the minibatch:
        i.  If s' is a terminal state, y_j = r_j
        ii. Otherwise, y_j = r_j + gamma * max Q'(s', a'; theta')
    9. Update the action-value function Q by minimizing the mean squared error
       between the predicted Q values and the target values:
        i.  Q(s, a; theta) = Q(s, a; theta) - alpha * (Q(s, a; theta) - y_j)^2
    10. Every C steps, copy the weights from Q to Q'
    11. Set s = s'
6. Repeat from step 5 until the action-value function Q has converged or a time limit is
   reached.
```

Appendix D: PPO pseudocode

PPO algorithm used in this research is based entirely on this paper[14] and made use of stable_baseline3 python package. Hyperparameters which are unspecified in the *Methodologies* Section follow the default hyperparameters of stable_baseline3 package.

```
1. Initialize neural network policy with random weights
2. Initialize value function with random weights
3. Set hyperparameters:
4. learning rate for policy and value function updates
    1. discount factor gamma
    2. number of epochs to train on each batch of data
    3. clip range epsilon for PPO objective
5. Collect a batch of data by running the policy in the environment
6. Compute advantages for each timestep in the batch using the value function
7. Normalize advantages to have zero mean and unit variance
8. For each epoch:
    1. Shuffle the batch of data
    2. Split the data into minibatches
    3. For each minibatch:
    4. Compute the current policy's action probabilities and log probabilities for the
       observations in the minibatch
    5. Compute the ratio of the new probabilities to the old probabilities for each action
       taken in the minibatch
    6. Compute the clipped objective as the minimum of the ratio multiplied by the
       advantages and the ratio clipped at (1 - epsilon) to (1 + epsilon) multiplied by the
       advantages
    7. Compute the policy loss as the negative mean of the clipped objective
    8. Update the policy using gradient descent on the policy loss
    9. Compute the value function loss as the mean squared error between the predicted
       values and the actual values
    10. Update the value function using gradient descent on the value function loss
9. Repeat from step 4 until the policy has converged or a time limit is reached.
```

Appendix E: A2C pseudocode

A2C algorithm used in this research is based entirely on this paper[15] and made use of stable_baseline3 python package.Hyperparameters which are unspecified in the *Methodologies* Section follow the default hyperparameters of stable_baseline3 package.

```
1. Initialize neural network policy pi and value function V with random weights
2. Set hyperparameters:
    1. learning rate alpha for both pi and V updates
    2. discount factor gamma
    3. number of timesteps to collect per episode
    4. number of epochs to train on each batch of data
3. For each episode:
    1. Reset environment to initial state s
    2. Repeat until episode ends:
    3. Sample an action a ~ pi(s; theta)
    4. Execute action a in the environment, observe reward r and next state s'
    5. Store transition (s, a, r, s') in a buffer
    6. If the buffer contains enough timesteps to train on:
    7. Compute the advantages A(s, a) for each timestep in the buffer using the value
       function V
    8. Compute the discounted rewards R(s) for each timestep in the buffer
    9. Normalize the advantages to have zero mean and unit variance
    10. For each epoch:
        i. Shuffle the buffer
        ii. Split the buffer into minibatches
        iii. For each minibatch:
            1. Compute the current policy's action probabilities and log
               probabilities for the observations in the minibatch
            2. Compute the policy loss as the negative mean of the product of the
               advantages and the log probabilities
            3. Compute the value function loss as the mean squared error
               between the predicted values and the actual values
            4. Update the policy and value function using gradient descent on
               their respective losses
    11. Clear the buffer
    12. Set s = s'
4. Repeat from step 3 until the policy and value function have converged or a time limit is
   reached.
```