Ocean Engineering 281 (2023) 114807

Contents lists available at ScienceDirect

Ocean Engineering

journal homepage: www.elsevier.com/locate/oceaneng

Dynamic trajectory planning for ships in dense environment using collision grid with deep reinforcement learning

R. Teitgen[a,b,*], B. Monsuez[a], R. Kukla[b], R. Pasquier[c], G. Foinet[c]

[a] *ENSTA Paris, 828 Bd des Maréchaux, 91120, Palaiseau, France*
[b] *Naval Group, Technopole de la Mer, 83190, Ollioules, France*
[c] *SIREHNA, Technocampus Ocean, 44340, Bouguenais, France*

ARTICLE INFO

Keywords:
Reinforcement learning
Collision grid
Collision avoidance
COLREGs
Proximal Policy Optimization

ABSTRACT

In areas with high maritime traffic, ship safety is of utmost importance when validating autonomous navigation models. While exact methods exist for specific situations, they are inadequate in a global context. This study employs an approximate deep reinforcement learning method to solve the navigation problem in a dense environment with numerous static and moving obstacles. Our model prioritizes ship integrity by enabling the agent to dynamically adapt its kinematics to its surroundings to reach a designated goal without colliding with obstacles. To achieve this, we incorporate collision grids in the form of danger zones as input to our model and train it using the proximal policy optimization algorithm. Additionally, we propose implementing the International Regulations for Preventing Collisions at Sea (COLREGs) in the collision grid as these navigation rules are necessary to obtain realistic behavior of an autonomous agent. The agent's performance is evaluated on a set of randomly generated scenarios operating in an environment complexity similar to the one used during training. These tests demonstrate that this type of data structure allows a trained agent to navigate in a dense environment while adhering to the COLREGs with a success rate of 94.69%.

1. Introduction

The primary goal of a ship is to ensure its safety. According to Baker and McCafferty (2005), human errors account for 80% of collisions and, therefore, there is a growing interest in developing autonomous navigation systems to perform parts of the navigation tasks. In dynamic environments where quick actions are required, autonomous systems must make effective decisions to ensure the safety of the ship.

Previous research, such as Statheros et al. (2008), has already shown that autonomous ship navigation is possible. To make relevant decisions, these systems had to consider various factors, including the COLREGs, ship properties, intentions, environment, and neighboring objects. These systems must solve several sub-problems, such as collision avoidance models that predict the trajectories of surrounding objects to build safe trajectory and avoid danger zones. However, the complexity of the environment and the number of objects make it challenging to consider all the objects, and exact methods struggle to cope with less ordinary situations.

In this context, we decided to use reinforcement learning (RL) (Sutton and Barto, 2018; Kaelbling et al., 1996) as an approximate method to improve existing algorithms. RL is a technique that enables an agent to learn how to navigate an environment by receiving rewards based on its actions and improving its behavior over time by maximizing the rewards it obtains. RL has been proven effective in playing video games such as GVGAI (Torrado et al., 2018) or Atari (Kaiser et al., 2019), mastering chess (Silver et al., 2017), performing network optimization (Luong et al., 2019), or cooperative robot navigation (Hu et al., 2020). Training algorithms have also evolved over time, with the Deep Q Neural Network (DQN) method taking over Q-learning by using neural networks to generalize the states of an environment with too many states to use a Q-table. Another significant development in RL is the Asynchronous Actor–Critic (A3C) algorithm (Mnih et al., 2016), which employs multiple agents simultaneously to optimize a global neural network. The Trust Region Policy Optimization (TRPO) (Schulman et al., 2015) can use a large number of experiments to improve behavior without compromising performance by optimizing the variation between the old and new policies. The Proximal Policy Optimization (PPO) (Schulman et al., 2017) is an enhanced version of TRPO, which simplifies the estimation of the gain of a new policy compared to the old one without reducing performance. In this paper, we will use the PPO method as the training algorithm. PPO has proven

* Corresponding author at: ENSTA Paris, 828 Bd des Maréchaux, 91120, Palaiseau, France.
E-mail addresses: raphael.teitgen@ensta-paris.fr (R. Teitgen), bruno.monsuez@ensta-paris.fr (B. Monsuez), romain.kukla@naval-group.com (R. Kukla), romain.pasquier@sirehna.com (R. Pasquier), gilles.foinet@sirehna.com (G. Foinet).

https://doi.org/10.1016/j.oceaneng.2023.114807
Received 14 December 2022; Received in revised form 29 March 2023; Accepted 8 May 2023
Available online 7 June 2023

to be at least as effective as TRPO or other RL algorithms, and it has demonstrated its efficacy in navigation tasks, particularly in maritime contexts, which is relevant to our work and enables us to compare it to similar studies.

This paper is structured as follows. Section 2 provides a literature review of collision avoidance methods for ships that incorporate COLREGs. Section 3 explains the construction of the collision grid and the embedding method of the COLREGs. Section 4 outlines the environment's design, its components, and the learning process. Section 5 presents the experiment's results and the behavior of a trained agent in specific situations. Finally, Section 6 summarizes the approach proposed in this work and discusses potential future improvements that could be made to the model.

2. Related works

Several studies have been conducted to develop methods for agents to determine their path while minimizing collision risks in uncertain environments. To address this problem, several tools have been proposed. Guo et al. (2020) used a reinforcement learning-based path planning approach, where they improved the Deep Deterministic Policy Gradient (DDPG) algorithm and designed a reward function that implements the Collision Regulations (COLREGs) to penalize unsafe behavior and ensure the ship's integrity. In another study, Shen et al. (2019a) used the Deep Q-Network (DQN) to develop obstacle avoidance strategies, validated their model through field trials and demonstrated its robustness in high-risk scenarios. The maneuverability space of a ship is dynamic and varies continuously depending on dynamics, intentions, and surrounding obstacles. To address this issue, Zhang et al. (2012) proposed a method that tailors the collision area around a ship according to COLREGs detection. They created a tool that defines the boundaries of a space where the ship will be in danger if an obstacle enters. Additionally, Sawada et al. (2021) used the Obstacle Zone by Target (OZT) technology to delineate danger zones in space and integrated OZTs as training data into a neural network with long-term memory (LSTM) using the Proximal Policy Optimization (PPO) method. They demonstrated that their model can pass the Imazu problem (Imazu, 1987), which is a set of COLREGs validation tests that challenges an agent to dodge an obstacle coming at it. Another approach proposed by Zhao and Roh (2019) divided the visibility space of a ship into four zones based on COLREGs for crossing, head-on, and overtaking. Moving obstacles were then evaluated based on two criteria: collision risks and potential COLREGs violations. Meyer et al. (2020) utilized the PPO algorithm to train an agent for obstacle avoidance and trajectory planning. Their reward function was based on the ship's compliance with its trajectory and minimizing collision risks. Real data from the Norwegian seas were used to validate their model since the collision risk in this area is particularly high. Another study by Li et al. (2021) investigated the feasibility of an unmanned system to navigate in uncertain environments. They divided the ship's field of view into four spaces, corresponding to the field of view dictated by the COLREGs, to separate the analysis of potential danger zones. The agent was trained through DQN and validated through high-risk collision scenarios. In another study, Bhopale et al. (2019) proposed a method for generating danger zones around obstacles for an Autonomous Underwater Vehicle (UAV). They created collision zones defined as extensions of a collision point, and the agent's objective was to avoid entering these zones. They used the QLearning algorithm to limit the possibility of entering a certain collision state. Moreover, Kuwata et al. (2014) proposed an approach using a collision cone based on the Velocity Approach method to provide early warning of possible collisions and to intuitively encode the COLREGs. This method makes it possible to project maneuverability zones on the trajectory to be followed by the ship.

3. Collision grid

In maritime navigation, operators must consider a variety of environmental factors to ensure the safe operation of their ships. With the environment constantly changing due to factors such as weather conditions, ship dynamics, and nearby obstacles, operators must select the most relevant information to ensure the safety of the ship, adhere to maritime regulations, and achieve their objectives. Therefore, it is necessary to develop a method that considers all observed factors to propose the most appropriate maneuver based on the current context. This work proposes a method for generating a collision grid that takes into account any object in the ship's field of view and evaluates its level of danger with respect to the ship's trajectory at each new observation generation. This approach overcomes two limitations observed in similar works. The first limitation concerns the discrimination of observed objects. Models such as neural networks have a fixed size of input data, and if the amount of observed data exceeds this size, a discrimination function must be modeled to remove part of the observation, such as selecting the most dangerous ship. As noted by Chun et al. (2021), this modeling can be difficult as the environment becomes more complex, and this discrimination can bias or even penalize the model's decision-making. The second limitation is the quantity of information that needs to be processed. Some works, such as Cheng and Zhang (2018), propose avoiding discrimination of information by providing a convolutional neural network (CNN) model with an image of the observed context, where all the observations are used, and there is no discrimination in pre-processing. However, the resolution of this image can introduce a significant amount of noise to the model, limiting its learning speed or efficiency. The proposed method overcomes these limitations by taking into account all observations without increasing the size of data used by the model and by limiting itself to the necessary information for decision-making.

3.1. Generate the grid

In order for an agent to learn through reinforcement learning (RL), it is essential to observe and interact with the environment to consolidate its behavior. To achieve this, it is crucial to determine what information the agent needs to receive and how to structure it, tailored to the specific problem it is attempting to solve. Unmanned surface vehicles (USVs) are a type of ship that does not require a crew to navigate and therefore must rely on sensors to capture surrounding information to maneuver according to the observed context. A range of sensors can be used, including Inertial Navigation Systems (INS) for estimating object movements, Laser Imaging (LiDAR) for determining object position with laser technology, and common devices such as cameras and microphones. In this context, we assume that the agent is equipped with the equivalent of an Automatic Identification System (AIS), which receives the position, direction, and speed of surrounding objects, as well as their type (static or moving objects).

To train a RL model, we need to define what constitutes an observation for the agent, meaning the information required for decision-making. The model is free to determine the best trajectory to take without any constraints on actions or trajectories based on the observed state. There are no checkpoints for the agent to follow, only an initial position and a destination to reach.

In the simulator, the agent's field of observation is defined by a fixed observation horizon set at initialization. All the elements on which the agent will base its behavior are extracted within this horizon, which is fixed at 120 m in this case. The first information the agent requires is the objective, denoted as $F_{x,y}$. If the objective's coordinates are beyond the horizon, a relative objective $F'_{x,y}$ within the range is recalculated at each new time step. The duration to reach the objective is counted in "time steps", which refer to the number of observations the agent makes during an episode. For instance, if an episode lasts for 50 time steps, the agent will have received 50 observations and executed 50 actions before reaching the objective or colliding with an obstacle. The relative objective is placed at mid-range to the horizon on the segment connecting the agent's position to the exact position of the objective. This position was determined based on experiments carried out with positions varying from 10% to 100% of the range, to find the optimal point. The experiments showed that if the position is too close to the agent, the agent is not cautious enough with surrounding obstacles and only focuses on reaching the objective. This results in models that have learned to reach the goal, but with a collision rate higher than an acceptable percentage. When the position is too close to the horizon, there are several problems. Firstly, if the position is between 90% and 100%, the objective may disappear from the observations due to the grid construction, which does not allow perfect tiling of the observation circle. If an object is not covered by a grid cell, even if it is below the horizon, it will not be considered. The second problem observed is that the further away from the center of the circle one goes, the greater the number of squares in the paving at the same distance. There will be more squares at 90% of the horizon where the relative target can appear than at 50% of the horizon. This increase leads to larger learning complexity as there is more information to process. Placing the relative objective at mid-distance strikes a balance between taking into account surrounding obstacles and pursuing the objective, resulting in better overall agent performance.

There are several pieces of information that our agent must consider, including collision points. First, we must take into account the environment's boundaries as well as static obstacles. The agent will perceive the borders, or "walls", as areas of absolute danger that it cannot access. Static obstacles are immobile points in space and will be represented by a single point. Moving obstacles are also important to consider and will be represented not only by their position at a given time but also by their next potential positions in three different directions $0°, -20°, +20°$. However, only positions that satisfy $n \mod 3 = 0$ will be retained to prevent overloading the collision grid generating algorithm. The set of possible collision points is taken into account only if their position is within or equal to the horizon. Furthermore, we decided to position the agent at $[0, 0]$ with a direction of $0°$ on the grid, and the relative position and direction of other objects within the horizon will be recalculated according to the agent's position. Therefore, the agent can only use points in space within the observation horizon and has no knowledge of the directions, speeds, or trajectory intentions of other objects. In other words, trajectory extrapolations are not considered real intentions, but rather linear predictions.

The neural network receives input data from a radar-like grid. The use of grids to represent environments has been previously studied by Boschian and Pruski (1993), who presented a method for representing a robot's 2D interaction space using a grid. This approach is extended in our work to include dynamically adaptable grids. Path planning solutions for automatic ship control using grids have also been proposed by Elfes (1989) for creating an occupancy grid from a robot's sensors, which can be used for trajectory planning algorithms by evaluating safe and dangerous areas. Additionally, Kim et al. (2014) used the Theta* algorithm to determine the optimal path for a ship based on a grid map. Woo and Kim (2020) adapted the idea of an occupancy grid for ship navigation using RL. They divided the observation space into three components, each of which is a grid for path tracking, fixed obstacles, and mobile obstacles. This approach allows for separate processing of information sources with CNN, which can then be merged together.

The collision grid is generated based on all the observations made by the agent. The circle-shaped observation horizon of the agent is divided into equally sized squares. The status of each square is determined based on the filling coordinates, which are the edges of the squares. If any element falls within the square, its status is changed. The size of the squares is determined by a trade-off between providing accurate information to the agent and computational performance in generating the grid. A grid with large squares could result in loss of accuracy, as any ship placed within a square would result in the entire square being filled. On the other hand, a grid with small squares would increase the size of input data and take much longer for the algorithm to converge. Therefore, for an observation horizon of radius $r_h = 120$ m, squares of side length $d_c = 15$ m were chosen, resulting in a grid containing 208 squares. Additional constraints, such as the collision zone of objects in the environment or the spacing of the ships' future positions, can be added to the tiling process.

Generating a perfect grid for the observation horizon is not feasible, especially with large squares. Additionally, a symmetrical grid can only be achieved if the ratio of the diameter of the horizon to the length of the square is an even integer. To generate the grid, we start from the lower left edge (i.e., coordinates $[-r_h; -r_h]$) and move from left to right and bottom to top with a step equal to the size of a square. If a square's center lies within the circle's area, the square is included in the grid. The filling ratio of the circle may vary depending on the square size. The algorithm for generating the grid is presented in Algorithm 1.

Algorithm 1 Generating the grid

```
$Grid = []$
$r_h \leftarrow horizon$
$d_c \leftarrow sizeof\,square$
$i \leftarrow -r_h$
**while** $i < r_h$ **do**
    $j \leftarrow -r_h$
    **while** $j < r_h$ **do**
        **if** $euclidean(0, i + d_c/2, 0, j + d_c/2, 0) < r_h$ **then**
            $Grid.append([i, j])$
        **end if**
        $j \leftarrow j + d_c$
    **end while**
    $i \leftarrow i + d_c$
**end while**
```

The grid is constructed at the beginning of the experiment and is reset to null values at every change of the environment state. Every object generated by the environment is then processed and mapped to the corresponding grid square (if it lies within the observation range). This information is stored in the table "$Objects\_environment$", where each index of the table is a sub-array containing a triplet $x_i, y_i, weight_i$, where i is the index of the table, x, y are the coordinates and $weight$ represents the level of attractiveness of the square. A higher weight indicates greater interest for the agent to reach that square, while a lower weight indicates a greater danger. The objective is assigned a weight of 1, walls and static objects are assigned a weight of 0.9, ships are weighted 0.8, and their linear predictions are weighted -0.8×0.85^j at each time step of the prediction, where j represents the time step. This weighting is applied in a cone that reflects the future positions of the ship. The intensity of the weights in the cone depends on the distance from the initial position, with the intensity decreasing as the distance from the origin increases. The cone is created by projecting the future ship positions in three directions, i.e., a modification of the direction of $dir_n + [0°, +20°, -20°]$, where dir_n represents the heading of ship n in degrees. The areas between these projections do not require weighting since it is assumed that the agent will not try to fit into empty spaces in the cone. An amplitude of $[-20°, +20°]$ also enables the agent to anticipate future changes in the trajectories of moving obstacles and thus adapt in advance to potential changes. The process of filling the grid based on object positions can be computationally intensive, especially when using a loop over the horizon. To optimize this, we developed Algorithm 3 which projects object coordinates onto the nearest multiple of the square size to determine which grid square the object belongs to, taking negative positions into account. We also generate a dictionary named $GridDict$ at the beginning of the experiment, which contains additional filling information for grid

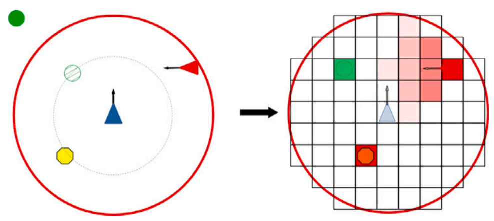

Fig. 1. Illustration of grid generation (right) from the agent's observations (left) of the environment. Our agent is in the center, the objective is in green, a fixed obstacle is in yellow and a ship entering the observation horizon is the red triangle.

positions. If a new object is added to a nonexistent location in this dictionary, the error is ignored and the experiment continues. This optimization reduces the experiment time while still providing the same results as the naive algorithm described in Algorithm 2. Fig. 2 displays an example of the generated grid based on the agent's observation. The left image provides a top view of the environment, while the right image shows the grid as a function of the agent's observation horizon, with the ship heading north. The relative objective on the port side and two moving obstacles with their respective projection cones placed in the grid are also shown.

Unlike the OZTs, the square filling approach is uniform, regardless of the obstacle's location within the square and the set of future ship movement projections. The approximations resulting from the uniform weighting simplify the hazard zone spaces significantly. Furthermore, the grid's pre-processed information is much less than that considered by a CNN. For instance, an image of size 50 × 50 pixels would have 2500 pixels to process, while our grid has only 208 inputs to process. Once the grid has been generated with all the visible objects on the agent's horizon, we average all the values contained in each square. Finally, we export these averages as a 1-dimensional matrix, representing the data observed by our agent at a specific time. Fig. 1 illustrates an example of the grid filling according to the environment state. On the left, the agent's observations are shown in a horizon defined by the large red circle. This information has already been pre-processed since the agent's angle is set to $0°$, and its position is $[0, 0]$. We observe the green target in the upper left corner, which is outside the horizon. Therefore, a relative target is placed in the same direction at 50% of the horizon, represented by the green hatched circle. If we follow the collision grid's construction, we see that the target fills one of the squares in green. A yellow static obstacle is observed behind the agent, and its location is filled in the collision grid as an absolute collision point. Finally, a moving obstacle enters the horizon to the starboard of the agent, and a collision cone has formed from the ship to the last position of the future projections of the ship's movements. It is important to note that this illustration of the grid generation does not consider any COLREG.

3.2. Implementing the COLREGs

The COLREGs, which are summarized in IMO (1972), are a set of regulations designed to prevent collisions at sea. These rules dictate how ships should behave in different environments and require the use of tools such as light and sound signals or ship controls to limit the risk of collisions with other vessels. They are international regulations that apply to all types of ships, regardless of their purpose or dynamics, to ensure consistency in maritime behavior. However, individual countries can make local adaptations to these regulations to enhance safety from their own perspective. The complete set of COLREGs rules can be found in DHS (2010). Our work focuses on a section of Section 2 of the maritime rules that pertains to a ship's behavior when other vessels are present in its field of observation. These rules are commonly studied in machine learning applications for autonomous ships. It is important

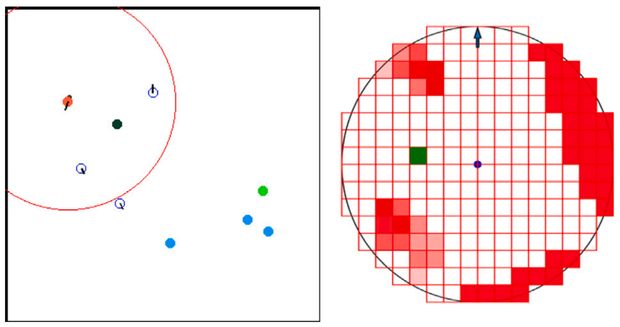

Fig. 2. Example of the view at a given moment of the simulator (left) and the collision grid (right) that the agent captures in the horizon. Static, moving obstacles or boundaries are in shades of red on the grid, the objective in green and the arrow on the grid indicates the direction of the agent. The agent is placed on the center of the grid, on the blue dot, the collision zones are therefore placed relatively to the position and direction of the agent.

Algorithm 2 Filling the grid - Naive Algorithm

```
Require: $Grid$
Require: $Objects\_environment$ as $objects$
  $Fill\_Grid = []$
  $d_c \leftarrow sizeof\,square$
  $i \leftarrow 0$
  while $i < Grid.size$ do
      $Fill\_Grid[0] = []$
      $j \leftarrow 0$
      while $j < objects.size$ do
          $x \leftarrow Grid[i][0]$
          $y \leftarrow Grid[i][1]$
          $m \leftarrow objects[j].x$
          $n \leftarrow objects[j].y$
          if $x - d_c/2 \le m$ and $x + d_c/2 \ge m$ and $y - d_c/2 \le n$ and $y + d_c/2 \ge n$
  then
              $Fill\_Grid[i].append([objects[j].weight])$
          end if
          $j \leftarrow j + 1$
      end while
      $i \leftarrow i + 1$
  end while
```

Algorithm 3 Filling the grid - Optimised Algorithm

```
  function CLOSESTMULTIPLE($n$,$mult$)
      return $(int)\ ((n + mult/2)\ /\ mult) * mult$
  end function

Require: $GridDict$
Require: $Objects\_environment$ as $objects$
  $Fill\_Grid = \{\}$
  $d_c \leftarrow sizeof\,square$
  $i \leftarrow 0$
  while $i < objects.size$ do
      $m \leftarrow objects[i].x$
      $n \leftarrow objects[i].y$
      $m \leftarrow sign(m) * closestMultiple(|m|, d_c)$
      $n \leftarrow sign(n) * closestMultiple(|n|, d_c)$
      $GridDict[m, n].append(objects[i].weight)$
      $i \leftarrow i + 1$
  end while
```

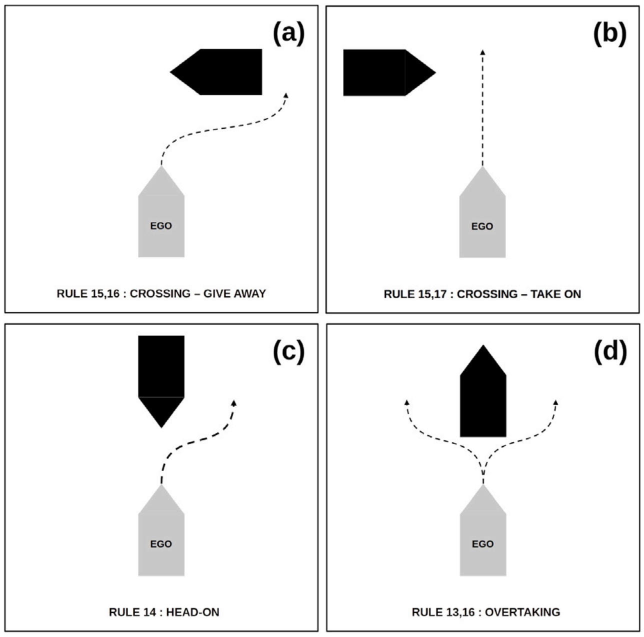
(a)
EGO
RULE 15,16 : CROSSING – GIVE AWAY
(b)
EGO
RULE 15,17 : CROSSING – TAKE ON
(c)
EGO
RULE 14 : HEAD-ON
(d)
EGO
RULE 13,16 : OVERTAKING

Fig. 3. Schematic diagrams of the four implemented Collision Regulations (COLREGs) when two ships are within each other's line of sight. The ship controlled by our agent is denoted as "Ego", and the arrow indicates the path that the agent must follow to comply with the observed COLREG.

to note that unlike our agent, the other ships in the environment do not adhere to the COLREGs and have random behaviors, meaning some maneuvers may not occur during this stage of the study.

To implement the COLREGs in our agent's behavior, there are various methods at our disposal. One way is to constrain the agent's actions at the operational level, meaning we block actions that would lead to a violation of the COLREGs or favor actions that lead to a safer trajectory. Another method is to modify the reward function by changing the reward the agent receives from observing the environment at each state change. Although there is a rich literature on implementing the COLREGs in this way, we decided against it. Tuning the reward function can be complicated and could introduce biases that weaken the learning process or even produce erroneous results. Instead, we considered modifying the agent's observation structure by incorporating the various COLREGs summarized in Fig. 3 into the inputs. In this figure, (a) and (b) represent the "crossing" rule where the rule in force is "priority to starboard": if two ships are crossing paths, the ship approaching from the starboard side maintains its course, and the ship on the port side must alter its course to pass behind the prioritized ship or turn around. Rule (c) is the "head-on" rule: when two ships approach each other from opposite directions, they both turn to starboard to pass each other on their port side. Lastly, rule (d) is the "overtaking" rule: if a ship intends to pass another ship and has sufficient speed, it must follow a parallel course or position itself far enough upstream of the overtaken ship so that the latter does not have to modify its course or speed. The figures presented here are intended only to illustrate what will be implemented later. Such maneuvers must be set up well in advance to limit changes in trajectory whenever possible.

The proposed approach aims to implement the COLREGs within the collision grid to address multiple constraints. Firstly, by not integrating the COLREGs into the reward function, we can handle each piece of information individually, rather than merging observations into a reward that is typically a sum of sub-rewards. This could cause the agent to overlook a COLREG if other information overrides the corresponding sub-reward with a positive sub-reward. Incorporating these rules into the inputs also brings clarity to the different actions the agent will take. As we enhance the observations with an overlay of information, the agent is constrained upstream of its decision-making, rather than downstream.

To implement each COLREG, we predict the future positions of ships to alter where the agent can navigate. By assigning weights to the cells associated with the projected tracks, we can affect the agent's perception of the environment and its danger level. For instance, let us assume an agent has learned that entering a weighted area results in a negative reward. If the same weight is assigned to the projected path of a neighboring ship, the agent will perceive an obstacle line ahead and avoid these coordinates, even if they are just potential dangers. This is how the COLREGs are implemented by adjusting the weights linked to the ships' projections, enabling the agent to learn how to recognize safe and hazardous areas while adhering to the navigation rules.

Fig. 4 illustrates the application of the COLREGs to the observations after modifying the weights. In the general case, the weight of each projection decreases linearly from the initial position of the ship, meaning that the danger decreases the further away from the obstacle. This applies to the three directions of the different projections d for a ship n. For the ith projection cell, the intensity of the danger zone is calculated using the equation $f_{n,dir_{0,20,-20}}(i) = 0.85^i$. For the "crossing" situation (a), where the agent does not have priority, all ships appearing in the upper right part of the agent's horizon, i.e. within the interval $[0°, 90°]$, should be taken into consideration. This first condition ensures that only ships in a position where they have priority are considered. The second condition is that the ship can potentially collide with the agent. If the intersection between the two paths is in front of the agent, i.e. under the horizon and in an angle included in the interval $[0°, 180°]$, we assign a maximum intensity of 1 to each projection. This creates zones of absolute danger in front of the agent, resulting in $f_{n,dir_{0,20,-20}}(i) = 1$. In the "crossing" case (b) where the agent has priority, we keep the initial weighting without reducing it to prevent the agent from taking riskier paths. Additionally, since the other ships have unpredictable intentions and do not respect COLREGs, the agent cannot ignore them. For the "head-on" rule (c), in case of crossing, the two ships must cross each other by their port side. The first step is to estimate whether the two ships are in a position to pass each other, and if so, the intensity of the danger zones concerned will be adapted. We check if their direction is opposite within a range of $[-20°, +20°]$, which is calculated with Eq. (1) where dir is the direction of an object in degrees. If the result is in the interval $[160°, 200°]$, we must also check that the position of the ship is in front of the agent, i.e. in the interval $[0°, 180°]$. If this condition is satisfied, then there is an intersection, and we modify the weights to minimize those on the port side of the obstacle ship and maximize those on the starboard side of the ship. This results in the following 3 equations: $f_{n,dir_0}(i) = 0.85^i$ for the front projection, $f_{n,dir_{20}}(i) = 0.7^i$ for the projection on the port side, and finally $f_{n,dir_{-20}}(i) = 1$ for the projection on the starboard side. Finally, the last COLREG "overtaking" (d) does not modify the initial weights because we want to be able to pass the overtaken ship without cutting its trajectory.

$$\begin{aligned} T'(agent_{dir}, ship_{dir}) &= max(agent_{dir}, ship_{dir}) \\ &\quad - min(agent_{dir}, ship_{dir}) \\ T &= T' \pm 20° \end{aligned} \tag{1}$$

4. Experimental setup

To train our model, we first define the environment and its constraints. Next, we choose the ship's dynamics, its objective, and the type of obstacles that will move and interact with the agent. For mobile obstacles, we believe it is more appropriate to give them random behaviors, which can be dangerous, in order to handle the worst possible situations. In a second step, we must process all the information received by the collision grid, as well as the weights to be given to each piece of information. We do not optimize the weights associated with the different objects as our goal is to validate the proposed approach, not fine-tune it. However, the agent's behavior could be improved and

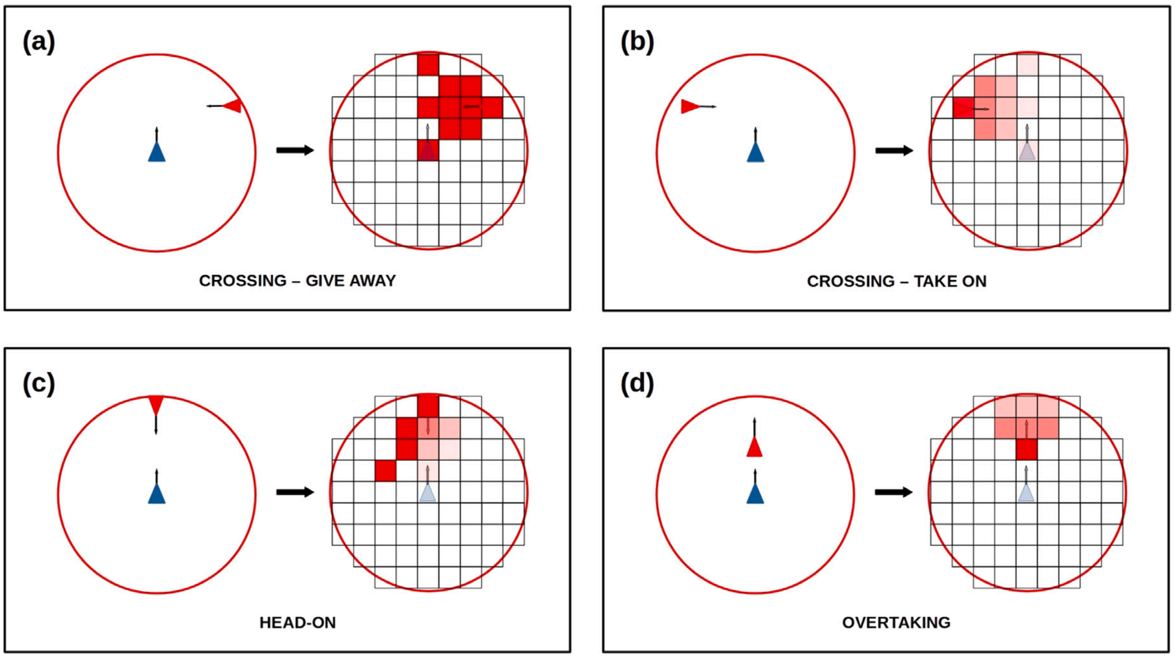

Fig. 4. Generation of collision zone weights in the grid for each COLREGs based on the observation of a moving obstacle. The ship piloted by the agent is depicted in blue, while the projections of the obstacle's future positions are shown in red. The opacity of the red cells corresponds to the level of danger posed by the obstacle, with darker shades indicating higher risk.

refined in such a way. Finally, we set the different hyperparameters of our model, such as the PPO algorithm and neural network structure, and determine the various actions the agent can perform according to its constraints. We train our agent over 1 million randomized episodes and record its performance evolution after each episode for further analysis.

4.1. *Simulator*

The simulator used for training the agent is a 2-dimensional space that measures $sizeMax \times sizeMax$ meters, where $sizeMax = 350$ m. The agent, which is a ship, operates in this environment with a variable direction expressed in degrees and a variable speed expressed in m s^{-1}. In each episode, a new objective is generated for the agent, which is a point in space that it must reach. The initial position of the agent and its objective are randomly generated, but we restrict the possibility of either element appearing too close to the edge. To ensure this, the coordinates x or y are generated within the interval $[20, sizeMax - 20]$ at the beginning of each episode. This constraint prevents episodes that are impossible to solve because the ship or the objective is generated within the hitbox boundaries. In this 2D space, there are two types of obstacles: static obstacles and moving obstacles. The position of static obstacles remains fixed throughout the episode. Moving obstacles have the same physical constraints as the agent but have no intention or objective. They follow a constant trajectory and have a probability of 5% of changing their heading at each time step. The behavior of these obstacles is not correlated to the agent and they can freely collide with or move away from it. If a moving obstacle touches an edge, it will reappear at a random position on the opposite edge. Collisions between obstacles or between an obstacle and a ship are ignored by the environment.

We added dynamics to the agent's behavior to make the simulator more realistic and incorporate constraints that affect moving objects. To achieve this, we used the first-order Nomoto model, as shown in Eq. (2), which provides an estimate of the rotation rate of the ship during maneuvers when it changes its heading. We modified this equation to apply it in discrete time to our simulator. The constants K and T in the equation affect the ship's dynamics, where T determines the stability of the rudder and K changes the difficulty of turning the ship. We needed to determine the best parameters for the experiment to run in a reasonable time, taking into account that ships with low maneuverability would require more space to perform a turn than those with high maneuverability. We compared the dynamics of some ships using the constants K and T studied in other papers. These included $K = 0.54, T = 0.35$ (Artyszuk 2016, Nomoto et al. 1957), $K = 4.9, T = 9.81$ (Artyszuk 2016), $K = 0.367, T = 0.421$; $K = 0.208, T = 0.064$; $K = 0.159, T = 0.048$ (Shen et al. 2019b). We also considered $K = 0.185, T = 107.8$; $K = -0.019, T = -153.7$ (Fossen 2011). We plotted the comparisons of the different trajectories as a function of the parameters T and K in Fig. 5. Based on this analysis, we selected the constants $K = 0.54, T = 0.35$ (blue curve), as they offered good maneuverability of the ship without making very tight turns too easy. These constants remained fixed for the entire duration of the experiment and could only be applied to ships with similar dynamics in real conditions.

$$\frac{r}{\delta}(s) = \frac{K}{1 + Ts} \tag{2}$$

To update the position of the ship, we calculate the rate of turn using a time step of $\delta_i = 1sec$, and then modify the ship's direction dir_i based on dir_{i-1} and the rate of turn. Using the ship's speed s, previous position x, y, and direction dir expressed in radians, we compute the new position using Eq. (3).

$$\begin{aligned} x_i &= s_i \times \cos(dir_i) + x_{i-1} \\ y_i &= s_i \times \sin(dir_i) + y_{i-1} \end{aligned} \tag{3}$$

4.2. *Reward evaluation*

Unless constrained by the episode duration defined in the learning or validation phases, the only way for the agent to terminate the episode is by colliding with an object in the environment, such as the objective, border, or an obstacle. To end the episode when reaching the boundary, crossing the space boundaries is sufficient, while entering a collision area, i.e., a hitbox, is necessary for other objects. The maximum Euclidean collision distance for the objective is slightly greater (15 m) than for other types of obstacles (13 m). This study does not use reward shaping (Ng et al., 1999), which means only the reward

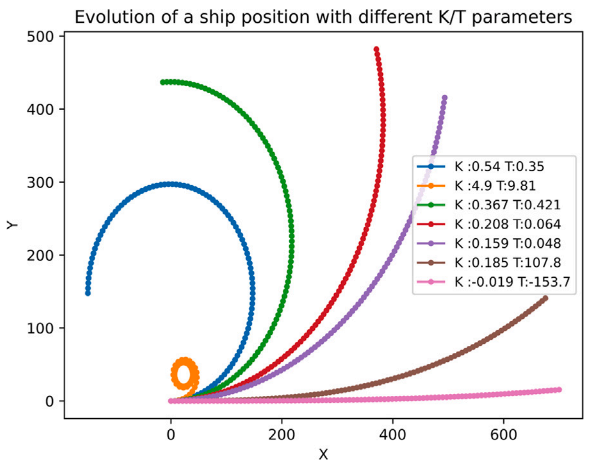

Fig. 5. Evolution of the position of a ship over 100-time steps by applying Nomoto's 1st order model as a function of different T, K. The ship maintains a constant speed while keeping its rudder angle fixed at 5°.

obtained at the end of the episode is considered. Although it is common to apply a modified reward to all observed states to accelerate learning and to evaluate certain states more accurately, it is not advisable in this study. The trajectory (time, speed, etc.) is not intended to be constrained too much, as it may lead to convergence towards dangerous states. Instead, the agent is taught to develop collision avoidance strategies even if it means taking longer paths.

The first justification is that a collision grid is used to evaluate the states' value, which clearly delimits the danger zones with negative weights and the interesting zones with positive or zero weights. This evaluation will be reflected in the network's updates, and the discounted reward will still assess the relevance of each transition leading to either a successful or a failed episode, even in the absence of a passive reward. Thus, a correlation can be made between the grid, its weights, and a discounted reward. The second justification is the complexity of constructing a passive reward function. It is difficult and ambiguous to determine the impact of each part of this function on the agent's convergence towards its goal. For instance, if the objective is too far away, a relative objective is displayed at 50% of the horizon, which further complicates the passive reward function. In the absence of obstacles, it is impossible to differentiate between a positive action towards the goal and a neutral or negative one when the grid remains unchanged. Thus, for such a transition, the reward becomes ambiguous, and providing a fixed reward would not align with the actual occurrence. Attempting to resolve this ambiguity by relying on the agent's heading is also insufficient since the rate of turn may not change the relative objective from one box to another. Even if a good evaluation of goal achievement was determined, it would still be necessary to assess the distance, potential collisions, and any violation of COLREG to establish a global reward. Since this information has already been considered in the grid at each step, including it in the reward would be redundant. Moreover, the weighting of different components is a significant disadvantage of passive rewards, and it is challenging to determine the appropriate weight for each component. The process of weighting the reward can be difficult as it involves a significant amount of hyperparameterization. Additionally, it may only be applicable to the particular problem at hand, making it difficult to reproduce with any modifications to the environment or actions taken by the agent. As a result, the agent can only receive a reward that is a constant value among three possibilities, which is determined by the type of collision. For instance, if the agent collides with the goal, it receives a reward of +5. However, if it collides with an obstacle or border, it receives a reward of −1, and in all other situations, the reward is 0. The equation for the reward function is shown in Eq. (4), and at the end of each

Table 1
Details of configuration of the environment used for training and evaluating the model.

Environment specifications	
Name	Value
Number of mobile obstacles	3
Number of static obstacles	3
Positive reward	5
Negative reward	−1
Null reward	0
Horizon of main ship (m)	120
Size of a square (m)	15
Obstacles hitbox radius (m)	15
Objective hitbox radius (m)	13

episode lasting t time steps, we compute the cumulative reward for each time step from $i = t - 1$ to $i = 0$ using the discount factor γ as $rw_i = \gamma \times rw_{i+1}$. There is no penalty for the number of time steps taken during an episode, so the agent is not penalized for using longer or shorter trajectories to reach its goal. The primary objective is therefore to reach the goal without colliding with any obstacles, and the only non-zero rewards come from positive or negative collisions that end an episode.

$$rw_{final} = \begin{cases} 5 & \text{if collision=objective} \\ -1 & \text{if collision=(ships or obstacles or walls)} \\ 0 & \text{otherwise} \end{cases} \tag{4}$$

5. Model validation

5.1. Agent training

The generation of our model will occur in two phases. The first phase involves the training of our model, while the second phase involves the evaluation of its performance. The training process will comprise one million episodes, each with a maximum duration of 300 time steps. The total training time will be approximately 52 h, which primarily depends on the number of squares in the grid and the number of obstacles in the environment. At the start of each episode, the agent's starting position, objective, and various obstacles in the environment will be randomly initialized. Thus, the learning environment will not remain static, and the parameters of the environment are summarized in Table 1. During both the learning and evaluation phases, at the start of each new episode, the position of all obstacles, as well as the rudder angle and direction for moving obstacles, are randomly reset, along with the position of the ship piloted by the agent and its target. This randomization of the environment allows us to observe the agent's ability to generalize its policy and perform well in previously unseen scenarios. The size of the environment is limited to 350 × 350 meters, with a maximum of three obstacles of each type. This constraint is expected to accelerate the learning process by enabling faster convergence and shorter episode durations. Additionally, limiting the number of obstacles should help the agent learn to avoid them without solely focusing on obstacle avoidance. Finally, since the input data is limited to a grid with a horizon, the size of the environment does not significantly impact the model's performance. While we can test the model on environments with varying sizes and obstacle densities, a restricted size during learning should result in faster convergence, albeit at the expense of partial observability of the environment by the agent.

To train the model using the PPO learning method, two neural networks need to be defined: the actor-network, which assigns weight to each action, and the critic-network, which evaluates the value of the agent's actions. Both networks have the same internal structure and the same number of inputs, namely, 3 hidden layers of 512 fully connected neurons. The input size is determined by the number of

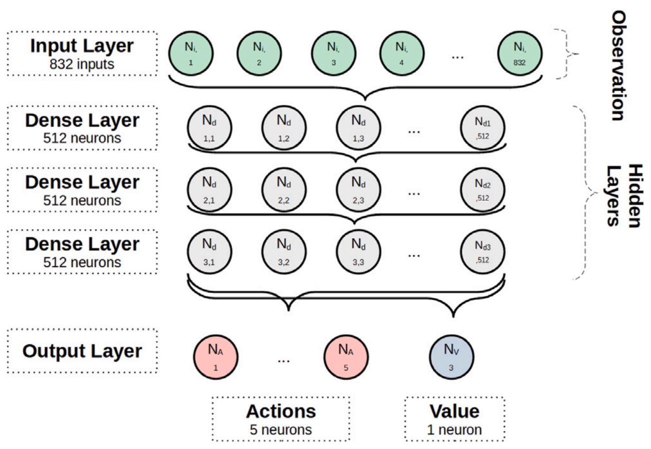

Fig. 6. Schematic representation of the neural network architecture used to train the learning model. Two networks are employed for the PPO algorithm, with identical structures except for the output layer. The actor-network consists of 5 neurons that correspond to the actions available to the agent for modifying its trajectory, while the critic network has 1 neuron for estimating the value function.

squares in the horizon, resulting in an observation size of 208. To estimate the movement of moving obstacles without using a Recurrent Neural Network (RNN) (Salehinejad et al. 2018), we implement short-term memory by stacking the last 4 observations, resulting in 832 inputs. The actor-network has 5 outputs, and the critic-network has 1 output. Fig. 6 illustrates the structure of the two neural networks used in the agent's training with PPO. The experiment will be conducted using an AMD Ryzen 9 3900X and an NVIDIA GeForce RTX 3090 running on Ubuntu 18.04.6. The algorithms are implemented in Python 3.6.9, using Torch 1.10 and CUDA 11.1.

The agent is trajectory-free, meaning it can adapt its trajectory towards its objective by adjusting its rudder angle or speed. Therefore, it does not follow a predetermined or passive reward-influenced trajectory, but can adjust its behavior based on the encountered state. The agent can perform five possible discrete actions for the actor-network, which are "accelerate", "decelerate", "turn to port", "turn to starboard", or "do nothing". When changing speed, the agent can modify its speed by ± 0.5 m s^{-1} within an interval of $[1-5]$ m s^{-1}. We do not want the agent to maneuver while immobile, and the maximum speed is a constraint due to the limitation of its field of observation and therefore its reaction time. For the helm maneuverability, its amplitude is limited in the interval $[-20°, +20°]$ and can only be changed by $-5°$ or $+5°$ at each time step to limit the possible reactions and physical constraints that the ship could have. The hyperparameters used for the PPO learning algorithm, which trains the neural networks, are summarized in Table 2, where the Adam optimizer with a beta of $(0.9, 0.999)$ is used.

5.2. Evaluation

Throughout the entire learning process, the type of collision, the duration of each episode (in time steps), and a binary value indicating the success or failure of each episode are saved. The analysis graphs presented in Fig. 7 are generated from this data and show the agent's learning progress over time. To enhance the readability of the different graphs, each displayed value for a given episode i is a rolling average calculated over the last 500 episodes, i.e., $\frac{1}{500}\sum_{n=i-500}^{i} object_n$, where $object$ is the relevant information calculated for each graph. For example, $steptime$ is used to plot the duration of each episode over time.

The upper part (TOP) of the figure presents the complete results of the experiment, while the lower part (BOT) displays only the evolution of the first 50,000 episodes. The first observation is that the agent has learned to reach its objective, and BOT shows that this learning was achieved in less than 20,000 episodes. The graph in the middle shows that there is a rapid increase in success rate in the first 20,000 episodes, followed by a gradual increase until reaching a success threshold of approximately 80% after 30,000 episodes. The graph in the bottom right indicates that in the first 10,000 episodes, the average time step increases from 120 to 150, indicating that the agent learns to reach its objective but takes more time to do so. Subsequently, the average time step decreases to 80 as the agent learns to find a more optimal trajectory without increasing negative collisions with the environment. The graph in the BOT left shows that the number of negative collisions is rare, and the episodes ending in "none" are at zero, indicating that the agent always tries to achieve its objective. TOP demonstrates that this rapid learning is not lost over time, and there is even a slight increase in performance since the central graph's slope is not completely flat. The TOP graphs also show that there is no loss of learning over time, and the entropy coefficient stabilizes the results without allowing risky explorations.

Table 2
Hyperparameters utilized by the PPO algorithm to train the model. The table presents various hyperparameters, including the number of episodes, the batch size, the learning rate, the discount factor, and the value of the clipping parameter. These hyperparameters play a critical role in determining the convergence and stability of the PPO algorithm, and their values are typically set based on prior experience with similar problems rather than using an automated hyperparameter optimization tool.

PPO Hyperparameters		
Name	Symbol	Value
Number max of episode	K	10^6
Batch size	T	3×10^4
Learning rate	η	0.0001
Discount Factor	γ	0.99
Clipping parameter	ϵ	0.1
Number minibatch	N	10
Value function coefficient	c_1	0.5
Entropy coefficient	c_2	0.01

A performance evaluation example is presented in Table 3. The environment settings are kept the same as the learning phase, and each evaluation is conducted with the same seed for comparison studies. The evaluations are performed at 1000, 10,000, 100,000, and 1 million episodes, with one evaluation representing the average performance of a run of 500 episodes. The table displays the accuracy rate, which is the percentage of successful episodes, the average duration of an episode measured in time steps, which refers to the average number of states observed by the agent before an episode ends, and the negative or zero collision rates, which represents the episodes where the maximum number of time steps has been reached without the agent reaching its goal or colliding with any obstacles.

It is important to note that some episodes are excluded from the performance calculation. Specifically, any episode lasting less than 20 time steps is removed from the calculations during evaluation. This is because it is believed that it takes at least 20 time steps at the beginning of an episode to start maneuvering if an obstacle is close to the agent. However, this estimate is highly dependent on the maneuverability capabilities of the agent, and the value would likely be much higher if the agent was less maneuverable. Episodes lasting less than 20 time steps leading to the goal are also ignored for consistency. The evaluation demonstrates that the agent continues to improve its behavior over time, and all cues are enhanced during the learning process. The summary Table 3 indicates that at the end of training, the agent shows an average performance of 94.69%. This means that only 5.31% of the 500 episodes tested, and exceeding the evaluation threshold of 20 time steps, resulted in negative collisions. The 500 evaluation scenarios are randomly generated and independent of those used during training. Accuracy is calculated as the number of episodes that have reached the objective divided by the total number of tested episodes, and the same method is used for calculating collision rates with fixed obstacles, ships, or walls. The evaluations are carried out using an

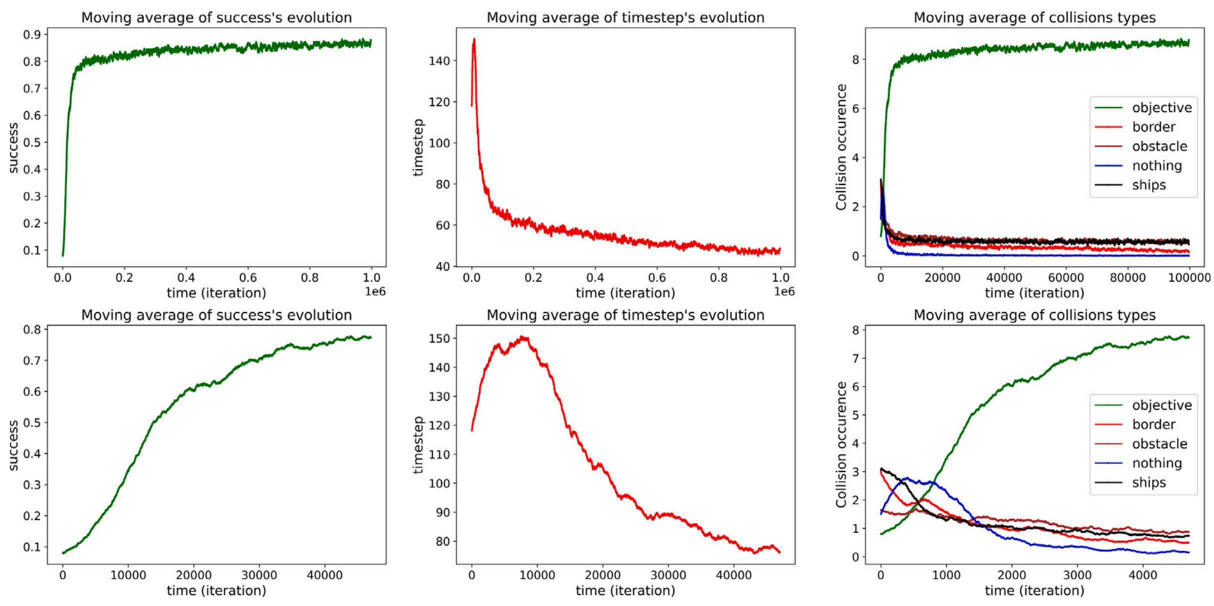

Fig. 7. Graphs depicting the evolution of the agent's learning over time. The left graph shows the distribution of collision types encountered by the agent, while the center graph displays the average number of episodes where the agent successfully reaches the objective. The right graph illustrates the average duration of each episode in terms of time steps. The top row of graphs represents the complete learning process, while the bottom row focuses on the first 50,000 episodes. These graphs provide valuable insights into the learning dynamics of the agent and the effectiveness of the training procedure.

Table 3
Analysis of the model's performance at different learning checkpoints, defined as a number of episodes during the training process. The table displays the average collision rates of the model over 500 episodes, with the performance being evaluated on the same seed. By using various learning checkpoints and systematically evaluating the model's performance at different stages of the training process, the table shows how the collision rates of the model evolve over time.

Performance evaluation				
	Number of episodes			
	10^3	10^4	10^5	10^6
Accuracy (%)	5.4	38.3	90.1	94.69
Mean time step	169	247	83	55
Ships collision (%)	30.3	21.3	2.5	2.5
Static obstacle collision (%)	14.5	18.9	7.1	2.8
Walls collision (%)	8.9	20.3	0.2	0
Zero collision (%)	40.9	1.2	0.3	0

Table 4
Evaluation of the model's performance at the end of the training process by varying the density of the environment, where a density of k means having k obstacles of each type. The table displays the average collision rates of the model over 500 episodes for different densities, allowing for a comprehensive analysis of the impact of environmental complexity on the model's performance. By systematically varying the density of obstacles and measuring the corresponding collision rates, the table provides insights into the robustness of the model to changing environmental conditions.

Performance based on density					
	Density				
	1	2	3	4	5
Accuracy (%)	98.6	96.8	94.69	88.9	83.89
Mean time step	45	54	55	63	71
Ships collision (%)	0.5	1.4	2.5	4.3	7.6
Static obstacle collision (%)	1	1.8	2.8	6.8	8.3

environment set up in the same way as during learning, as summarized in Table 1. The 500 evaluation scenarios were generated in a total of 26.58 s, using the same setup as the learning phase. Table 4 displays the progress of an agent's performance using the same policy while altering the obstacle density in the environment. The density refers to the number of obstacles for each type. To ensure comparability, the various tests conducted on different densities were carried out on identical environment dimensions, i.e., 350×350 m. These assessments were performed on the same model trained previously, with only the agent's ability to navigate in environments of different densities being tested, considering that it was trained in a density 3 environment. The number of fixed and mobile obstacles was kept the same. The results show that the overall performance of the agent deteriorates with an increase in density, and too many obstacles hinder the agent's ability to navigate.

We can demonstrate the agent's learned behavior through two scenarios detailed in Fig. 8. These scenarios provide a visual overview of the agent's behavior in the environment. The environment used in the demonstration scenarios has been enlarged to a square of size 800×800 m, with ten obstacles of each type. The ships' dynamics, hitboxes, appearance modes, and behaviors of the different objects are the same as those used during training. The two episodes presented above one another show the scenario's progression divided into four frames, each frame containing the same number of time steps, except the last one, which contains slightly more to display the entire episode. In each frame, the starting point is the large red dot, and the agent's last position in each image is the small red dot. Its previous positions are in yellow, and its objective is in green. We can also see the trajectories of mobile obstacles in blue and fixed obstacles in light blue. The two examples presented were selected based on their interest in analyzing the agent's behavior in only four frames. Sometimes the path to the objective is too short, making it interesting to analyze, or it enters a dense area where it is difficult to analyze its maneuvers with just one image.

In the first scenario, one can observe in frame 1 that the agent is already headed in the right direction at the start, and it moves straight towards its objective, making minor adjustments to stay on course. In frame 2, the agent tries to avoid a stationary and a moving obstacle by turning to port in a spiral, and then returning to its objective. In frame 3, a moving obstacle approaches the agent head-on, so the agent tries to dodge it by turning to port. We can see that the agent has turned almost 90° to avoid a collision, as a COLREG was activated, but it was forced to tack. Therefore, it chose to move far away from the ship to

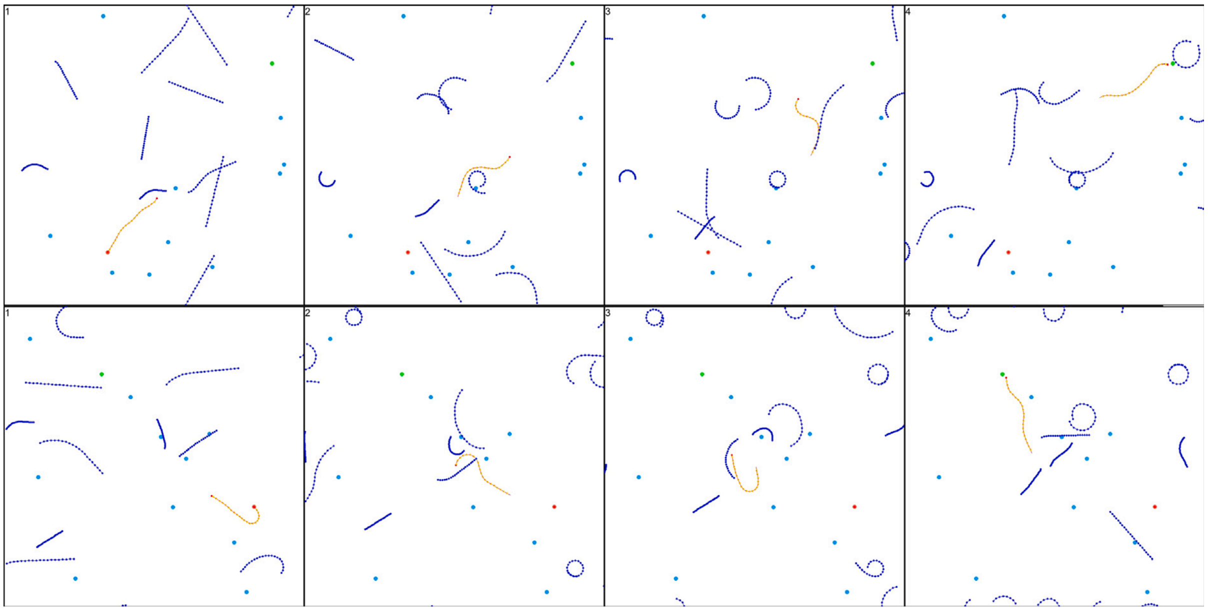

Fig. 8. An example of a sequence of two episodes, each divided into four frames, depicting the agent's trajectory (in red) as it attempts to navigate through a path with 10 obstacles of each type. The starting point is indicated by a large red dot and the objective is represented by green. Variations in the agent's trajectory are the result of evasive maneuvers it takes to avoid colliding with obstacles.

avoid a collision rather than crossing to starboard and then resuming its course. In frame 4, there are no more obstacles to alter its trajectory, so the agent reaches its goal without any further dodging maneuvers.

In the second scenario, one can observe that in frame 1, the agent has to maneuver because the random generation of its direction is opposite to its objective. Thus, it tries to place itself on course. Frame 2 shows that the agent encounters many collision risks and operates accordingly. First, a ship approaches the agent from starboard, so it deviates slightly to starboard to pass behind it with a safety margin. Afterward, it meets another ship that is spiraling, making it difficult to attempt a starboard maneuver, so it turns drastically to port, almost making a half-turn. Frame 3 shows the rectification of its trajectory due to the previous maneuver, and it finds its course. In the last frame, due to the absence of collision risk, it maintains its course almost constantly to reach its objective.

In this example, Fig. 9 demonstrates how the agent reacts and respects each COLREG. The agent starts from a large red dot and must reach a green dot, with its trajectory depicted in yellow and its last position represented by a small red dot. A blue ship appears on the horizon and causes the activation of a COLREG. The initial positions of the moving obstacle, the agent, and its goal were manually set to force the activation of the desired COLREG. In this case, the obstacle ship maintains a fixed speed and cannot change its direction. These examples do not validate that the COLREGs will be respected in all situations, but demonstrate that the proposed implementation is consistent. In the "Head-on" case, we observe that the agent changes its course to starboard to avoid the oncoming ship. In the "Overtaking" case, the agent attempts to overtake the ship, but realizes that it cannot place itself too early in front of the overtaken ship, so it changes its trajectory to move further away from the ship. In the "Crossing" case, where both priority situations are proposed, we observe that the agent does not maneuver too much when it has priority. However, it gets closer to the other ship, probably due to the fact that the target was no longer in its axis, and it attempted to turn to recalibrate but was too far away. The graph of the average episode duration shows that the agent tries to reach the target as quickly as possible, and a recalibration of its direction seems consistent with this but takes precedence over safety. Finally, the last case is when the agent does not have priority, and we can observe that it turns 90° to ensure that the other ship passes before continuing to its goal. Further work can be done to validate the agent's behavior in more complex and diverse situations.

6. Conclusion

In this study, we demonstrate that using a collision grid within an agent's observation horizon as an input structure enables the agent to navigate through dense environments and accomplish its objectives. The grid is created based on the agent's observation horizon and is updated at each iteration to reflect changes in the surrounding objects. Additionally, the potential future movements of mobile objects are considered to evaluate the collision risks, allowing the agent to anticipate movements and ensure its safety throughout its journey. To train the agent, we employed the PPO learning algorithm, which has been proven effective in solving similar navigation problems. In a dense environment, the agent was trained to reach its goal while avoiding collisions with randomly behaving objects. The results indicate that this learning approach enabled our agent to develop a behavior that yielded outstanding performance and a low risk of collision with its environment.

Having successfully overcome random behavior, the agent has acquired the ability to navigate safely in space while avoiding objects that may collide with it. During the evaluation phase, irrelevant episodes that did not represent the agent's behavior were removed, but collisions with mobile objects still occurred. It was observed that in some instances, the agent was unable to make a decision to prevent collisions, such as when a ship changed course at the last moment, and the agent was physically incapable of reacting quickly enough to avoid impact. Additionally, collisions occurred when objects approached the agent too quickly for it to react. In general, collisions with the environment happened when the agent was unable to avoid them. However, the collision grid facilitated efficient implementation of COLREGs, enabling the agent's behavior to be explained in situations where it may have been challenging to incorporate these rules into a complex reward function and ensure that they were accounted for.

There is potential for the model to be improved to reinforce the agent's behavior and enhance its realism. One possible improvement is to test the model on other training algorithms. It would also be relevant to make the model more robust and less sensitive to various adversarial attacks and potential data losses. Although the model has not experienced any data losses or sensor failures due to filling a whole grid square with a single point, such incidents could occur in real-world situations. As a result, additional sensors, such as LiDAR, could

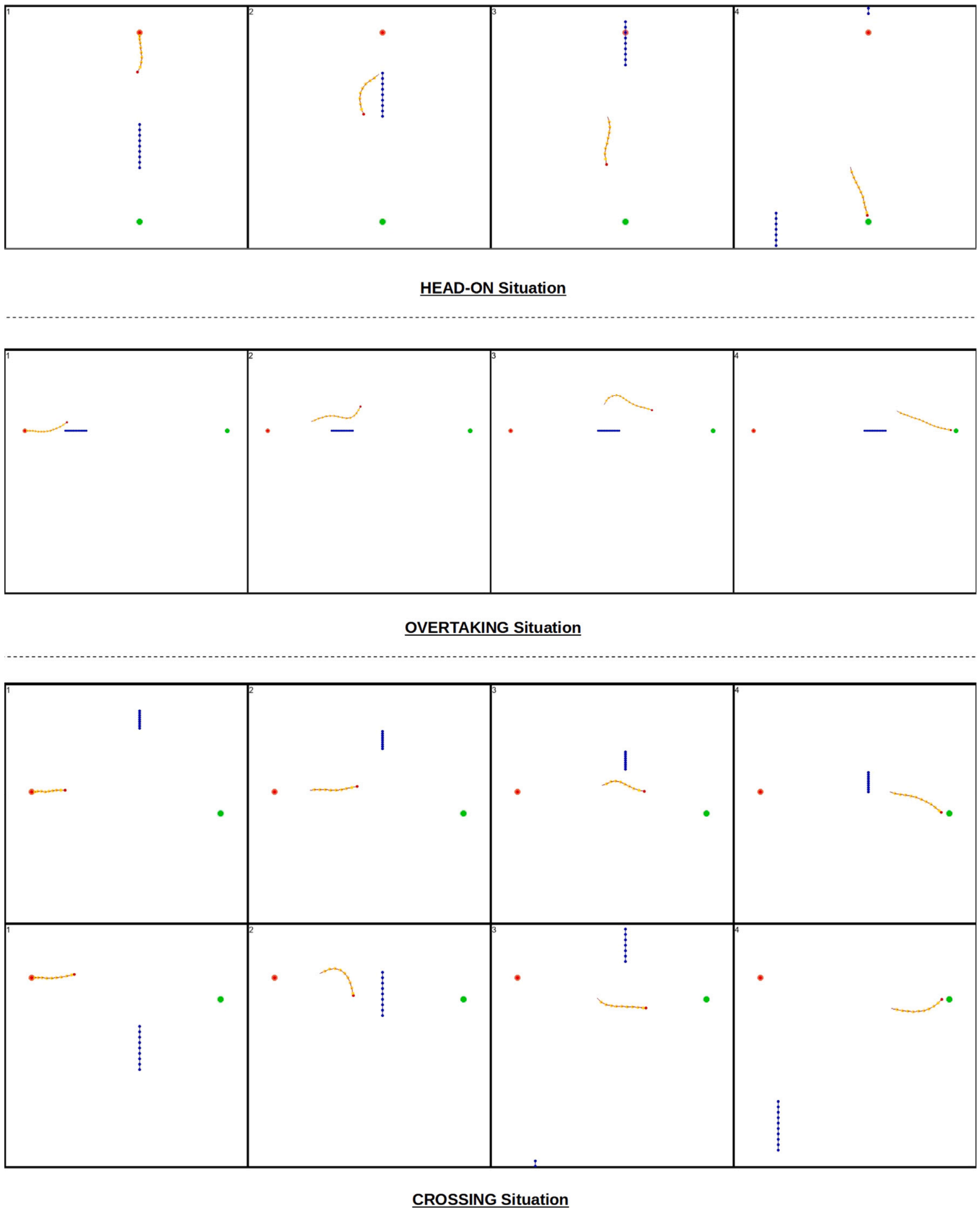

Fig. 9. Examples of situations where the trained agent must apply COLREGs, as implemented in this work. Each example is presented in four frames, depicting the progression of the episode from left to right. These examples were generated at the end of the experiment to evaluate the agent's ability to activate the appropriate COLREG in response to various scenarios.

be integrated to compensate for any information losses or improve the collision grid's information quality. The hitboxes for detecting possible collisions could be refined to better adapt to the trajectories of moving obstacles. Furthermore, the values within the collision grid could be propagated to their surroundings with weighted intensity, so that areas too close to ships are not considered safe by the agent. An evaluation of the model's performance with actual data from AIS could also be considered to validate the agent's maneuvers in real conditions and compare the approaches of different autonomous ships under the same conditions. Despite the promising results of this study, there is still work to be done to ensure the collision grids' reliability as input to the model.

CRediT authorship contribution statement

R. Teitgen: Conceived and designed the analysis, Collected the data, Contributed data or analysis tools, Performed the analysis, Writing – original draft. **B. Monsuez:** Contributed data or analysis tools, Performed the analysis, Writing – original draft. **R. Kukla:** Contributed data or analysis tools, Performed the analysis, Writing – original draft.

R. Pasquier: Contributed data or analysis tools, Other contribution. **G. Foinet:** Contributed data or analysis tools, Other contribution.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

No data was used for the research described in the article.

References

Artyszuk, J., 2016. Inherent properties of ship manoeuvring linear models in view of the full-mission model adjustment. TransNav Int. J. Mar. Navig. Saf. Sea Transp. 10 (4), 595–604.

Baker, C., McCafferty, D., 2005. Accident database review of human element concerns: What do the results mean for classification. In: Proc. Int Conf.'Human Factors in Ship Design and Operation. RINA Feb, Citeseer.

Bhopale, P., Kazi, F., Singh, N., 2019. Reinforcement learning based obstacle avoidance for autonomous underwater vehicle. J. Mar. Sci. Appl. 18 (2), 228–238.

Boschian, V., Pruski, A., 1993. Grid modeling of robot cells: A memory-efficient approach. J. Intell. Robot. Syst. 8 (2), 201–223.

Cheng, Y., Zhang, W., 2018. Concise deep reinforcement learning obstacle avoidance for underactuated unmanned marine vessels. Neurocomputing 272, 63–73.

Chun, D.-H., Roh, M.-I., Lee, H.-W., Ha, J., Yu, D., 2021. Deep reinforcement learning-based collision avoidance for an autonomous ship. Ocean Eng. 234 (109216), 109216.

DHS, 2010. International regulations for preventing collisions at sea. URL: https://web.archive.org/web/20100927052959/http://www.navcen.uscg.gov/?pageName=navRulesContent.

Elfes, A., 1989. Using occupancy grids for mobile robot perception and navigation. Computer (Long Beach Calif.) 22 (6), 46–57.

Fossen, T.I., 2011. Handbook of Marine Craft Hydrodynamics and Motion Control. John Wiley & Sons.

Guo, S., Zhang, X., Zheng, Y., Du, Y., 2020. An autonomous path planning model for unmanned ships based on deep reinforcement learning. Sensors 20 (2), 426.

Hu, J., Niu, H., Carrasco, J., Lennox, B., Arvin, F., 2020. Voronoi-based multi-robot autonomous exploration in unknown environments via deep reinforcement learning. IEEE Trans. Veh. Technol. 69 (12), 14413–14423.

Imazu, H., 1987. Research on Collision Avoidance Manoeuvre. Tokyo University of Marine Science and Technology, Tokyo, Japan.

IMO, 1972. Convention on the international regulations for preventing collisions at sea. URL: https://www.imo.org/en/About/Conventions/Pages/COLREG.aspx.

Kaelbling, L.P., Littman, M.L., Moore, A.W., 1996. Reinforcement learning: A survey. J. Artificial Intelligence Res. 4, 237–285.

Kaiser, L., Babaeizadeh, M., Milos, P., Osinski, B., Campbell, R.H., Czechowski, K., Erhan, D., Finn, C., Kozakowski, P., Levine, S., et al., 2019. Model-based reinforcement learning for atari. arXiv preprint arXiv:1903.00374.

Kim, H., Kim, D., Shin, J.-U., Kim, H., Myung, H., 2014. Angular rate-constrained path planning algorithm for unmanned surface vehicles. Ocean Eng. 84, 37–44.

Kuwata, Y., Wolf, M.T., Zarzhitsky, D., Huntsberger, T.L., 2014. Safe maritime autonomous navigation with COLREGS, using velocity obstacles. IEEE J. Ocean. Eng. 39 (1), 110–119.

Li, L., Wu, D., Huang, Y., Yuan, Z.-M., 2021. A path planning strategy unified with a COLREGS collision avoidance function based on deep reinforcement learning and artificial potential field. Appl. Ocean Res. 113, 102759.

Luong, N.C., Hoang, D.T., Gong, S., Niyato, D., Wang, P., Liang, Y.-C., Kim, D.I., 2019. Applications of deep reinforcement learning in communications and networking: A survey. IEEE Commun. Surv. Tutor. 21 (4), 3133–3174.

Meyer, E., Heiberg, A., Rasheed, A., San, O., 2020. COLREG-compliant collision avoidance for unmanned surface vehicle using deep reinforcement learning. IEEE Access 8, 165344–165364.

Mnih, V., Badia, A.P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., Silver, D., Kavukcuoglu, K., 2016. Asynchronous methods for deep reinforcement learning. In: International Conference on Machine Learning. PMLR, pp. 1928–1937.

Ng, A.Y., Harada, D., Russell, S., 1999. Policy invariance under reward transformations: Theory and application to reward shaping. In: Icml, Vol. 99. pp. 278–287.

Nomoto, K., Taguchi, T., Honda, K., Hirano, S., 1957. On the steering qualities of ships. Int. Shipbuild. Prog. 4 (35), 354–370.

Salehinejad, H., Sankar, S., Barfett, J., Colak, E., Valaee, S., 2018. Recent Advances in Recurrent Neural Networks. arXiv.

Sawada, R., Sato, K., Majima, T., 2021. Automatic ship collision avoidance using deep reinforcement learning with LSTM in continuous action spaces. J. Mar. Sci. Technol. 26 (2), 509–524.

Schulman, J., Levine, S., Abbeel, P., Jordan, M., Moritz, P., 2015. Trust region policy optimization. In: International Conference on Machine Learning. PMLR, pp. 1889–1897.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., Klimov, O., 2017. Proximal policy optimization algorithms. arXiv:1707.06347.

Shen, H., Hashimoto, H., Matsuda, A., Taniguchi, Y., Terada, D., Guo, C., 2019a. Automatic collision avoidance of multiple ships based on deep Q-learning. Appl. Ocean Res. 86, 268–288.

Shen, H., Hashimoto, H., Matsuda, A., Taniguchi, Y., Terada, D., Guo, C., 2019b. Automatic collision avoidance of multiple ships based on deep Q-learning. Appl. Ocean Res. 86, 268–288.

Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., et al., 2017. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. arXiv preprint arXiv:1712.01815.

Statheros, T., Howells, G., Maier, K.M., 2008. Autonomous ship collision avoidance navigation concepts, technologies and techniques. J. Navig. 61 (1), 129–142.

Sutton, R.S., Barto, A.G., 2018. Reinforcement Learning: An Introduction. MIT Press.

Torrado, R.R., Bontrager, P., Togelius, J., Liu, J., Perez-Liebana, D., 2018. Deep reinforcement learning for general video game AI. In: 2018 IEEE Conference on Computational Intelligence and Games. CIG, IEEE, pp. 1–8.

Woo, J., Kim, N., 2020. Collision avoidance for an unmanned surface vehicle using deep reinforcement learning. Ocean Eng. 199 (107001), 107001.

Zhang, J., Yan, X., Chen, X., Sang, L., Zhang, D., 2012. A novel approach for assistance with anti-collision decision making based on the international regulations for preventing collisions at sea. Proc. Inst. Mech. Eng. Part M 226 (3), 250–259.

Zhao, L., Roh, M.-I., 2019. COLREGs-compliant multiship collision avoidance based on deep reinforcement learning. Ocean Eng. 191 (106436), 106436.