MECH0020 Individual Project

2019/20

Student:	**Ding Ke**
Project Title:	**Deep reinforcement learning based path planning for underactuated autonomous cargo ships.** (Word count: 7466)
Supervisor:	**Dr. Liu Yuanchang**

Deep Reinforcement Learning Based Path Planning for Underactuated Autonomous Cargo Ships

Ding Ke, supervised by Dr. Liu Yuanchang

Abstract

Unmanned Surface Vehicles (USV) gains increasing attention thanks to the flexibility and cost-effectiveness of its applications in fields like maritime transport, oceanographic surveys. A quick and robust path planning to avoid obstacles is a key component to the USV's automated system. This project is mainly concerned with the problem of global path planning to avoid fixed and static obstacles. Traditionally, there have been many modelling approaches to the problem, but most often these precisely constructed models do not warrant much scalability or adaptability except for the exact scenario it is designed for. In this project, a data-driven approach based on reinforcement learning (RL) is investigated. By acting greedy with respect to the action values as the target policy for the temporal difference (TD) update, Q-learning which is a special case of experienced-based RL is chosen, as its strong performance displayed in some maze solving scenarios and Atari games. The design of state and action space critical to Q-learning requires domain expertise in terms of careful analysis of the ship dynamics.

Large and slow ship types like cargo ships are analysed, allowing some reasonable assumptions over ship dynamics to be made. The ship is modelled as a rectangular rigid body in which the surge speed is assumed to be constant. The only controllable surface is the rudder, and a range for rudder angle δ is set to bound the ship action space for later input to Q-learning. The control relationship between rudder angle and ship heading is modelled by the first order Nomoto KT system, where inertial effect and water disturbance are modelled by k, T coefficients in the parameterized form. A 2-D ship motion system is employed such that any state on the 2-D plane can be expressed with three parameters: x-coordinate, y-coordinate, and ship heading angle ψ. A discretization of state and action space is then carried out, mapping the continuous 2-D plane into a grid with a total of 100x100x36 discrete states, each allowing 5 possible rudder actions. In this work, two types of reward designs are considered. One consists of all constant rewards, the other involves a distance reward function that will apply more reward when approaching the target. Performances under both reward settings are compared. The algorithm's path planning capability has been tested and visualised on 4 different obstacle scenarios, serving to validate the good adaptability of the RL approach. The sum of rewards per episode curve is checked so that convergence is guaranteed.

Lastly, to further improve practicality of the proposed path planning, a concise Deep Q-Network (DQN) consisting of 3 fully connected layers with neuron configuration 64x32x32 is implemented. The DQN builds on the previous Q-learning settings, with modified rewards. The DQN is first tested on the standard 100x100 map, then on a larger 200x200 map. Planned trajectories are visualised, mean-square-error loss and average reward per episode are plotted to determine the convergence. An investigation on the hyperparameter selections for the neural network is also included. Finally, both algorithms' performances are analysed to arrive at a conclusion.

Keywords: Deep Reinforcement Learning; Nomoto model; path planning; obstacle avoidance; Q-learning; Deep Q-Network;

Acknowledgements

First of all, I would like to express my sincere thanks and gratitude towards my own supervisor Dr. Liu Yuanchang. During those many meetings we had, he was able to correct me at an essential time when I seemed to deviate too much from the core of the project. As a second-year ME undergrad with inadequate background knowledge for the computing nature of this project, almost a year ago I was only able to start to self-embark on a new quest of learning, facing an entirely new realm of knowledge, through my own interest, determination, ambitions and courage. Yet Dr. Liu deeply understands the value of self-determination from his own personal experience, and therefore he encouraged and supported me on my striving for knowledge where possible. I also want to thank him for advising me with a new approach of project-based learning to increase my efficiency, which proves to be invaluable to me not only for the success of this project, but also for future ones to come.

I also wish to express thanks to the Reinforcement Learning course by University of Alberta, Python Data Science course by Rice University and the organization of Coursera which hosts them. Without them, it would be so much harder of a task for me to get real fundamental grasp on python coding techniques for machine learning and reinforcement learning in particular. I finally want to thank "Morvan Python" for his DQN and tensorflow tutorials on his YouTube channel and website (https://morvanzhou.github.io/tutorials/machine-learning/reinforcement-learning/). Without these introductory tutorials, it would be impossible for me to begin such a gigantic task.

Declaration

I, DING KE, confirm that the work presented in this report is my own. Where information has been derived from other sources, I confirm that this has been indicated in the report.

Table of Contents

Nomenclature

x_0 :	Horizontal displacement in cartesian reference frame
y_0 :	Vertical displacement in cartesian reference frame
K:	Turning ability coefficient
MCA:	Minimum Collision Area
MDP:	Markov Decision Process
NN:	Neural Network
SGD:	Stochastic Gradient Descent
T:	Turing lag coefficient
CG:	Centre of Gravity
CGx:	Horizontal projection of Centre of Gravity
CGy:	Vertical projection of Centre of Gravity
x :	Horizontal displacement in ship-fixed reference frame
y :	Vertical displacement in ship-fixed reference frame
F_{x0} :	Total horizontal force exerted on ship hull in cartesian frame
F_{y0} :	Total vertical force exerted on ship hull in cartesian frame
N :	Total resultant moment on ship hull
I_{zz} :	Moment of inertia about z axis
δ :	Rudder angle of ship
ψ :	Course angle (heading) of ship
$\dot{\psi}$:	Yaw rate of ship (also could be represented as r)
$s_{(t)}$:	State at time step t
$s_{terminal}$:	State of termination of an episode
$a_{(t)}$:	Action at time step t
α :	Learning rate/step size parameter
$r_{(t)}$:	Reward at time step t
r_{max} :	Maximum cumulative reward of an episode
s' :	Next state
$G_{(t)}$:	Gain at time step t
$V_{(s)}$:	State value for state s
$\pi_{(a\|s)}$:	Policy of agent

γ :	Discount factor
p :	Transition probability (dynamics) of MDP
$\mathrm{Q}_{(\mathrm{s,a})}$:	Action value at state s choosing action a
$\mathrm{Q}'_{(\mathrm{s}',\mathrm{a})}$:	Next step action value at next state s'choosing action a
ε :	Epsilon greedy exploration probability
L :	Distance reward function
r_{L} :	Reward from distance reward function L
D :	Direction reward function
r_{D} :	Reward from direction reward function D

List of Figures

List of Tables

1. Introduction

1.1 Background and motives:

The emerging developments for USV in recent decades have brought considerable changes to maritime operations, namely in oceanographic surveys where Delfim USV was deployed for investigating hydrothermal activity in Portugal(Pascoal et al. 2000) and Messin USV was used in Germany for water ecological studies (Majohr et al. 2000). Possessing high flexibility and cost-effectiveness, the significance of employing USVs lies in that they can perform tasks in high-risk, adverse environments without jeopardizing human life (Naeem et al. 2006).

There is less research on larger USVs with for example cargo carrying capabilities that can perform long-range maritime transport missions. The reason behind is perhaps the low-cost, low-risk and easy-to-implement nature of small USVs. While the great tonnage and huge inertia for large cargo ships makes their automation a high risk maneuver leaving little margin for error.

Nevertheless, in this project, the tremendous benefits of automating maritime transport both in improved economics and most importantly in life-saving scenarios are recognised. An IEEE publication (Levander 2017) shows that, without need for crew compartment, more space can be assigned to cargo storage, and the silhouette can be optimised for a more streamlined profile, resulting in an increased cargo capacity and decreased wind drag.

1.2 Framework for autonomous ships:

From a research on ship collision avoidance system(Huang et al. 2020), a general framework for the autonomous control sequence can be laid out as in figure 1, displaying the two main components of autonomous ships: control and planning.

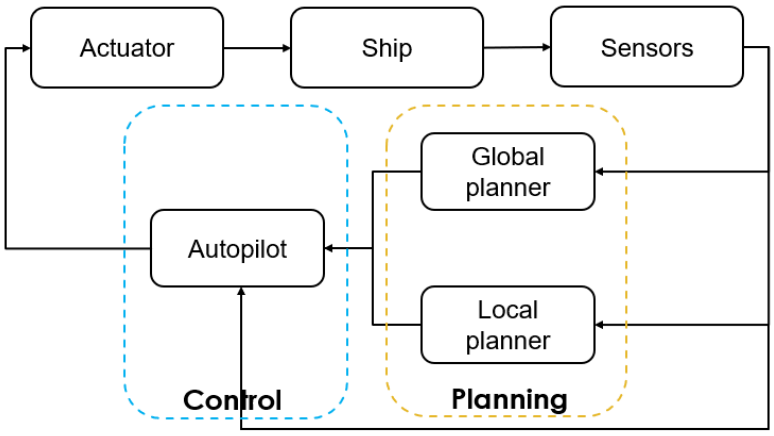

Figure 1. Block diagram for the autonomous ship's navigation and control.

1.3 Problem definition:

The intelligent system itself is required to be able to avoid obstacles, negotiate with dynamic environmental interferences, overcome uncertainty due to perception and select optimal control policies, among which a quick, robust path planning plays a vital role. Thus, in this project, a particular interest is placed upon the path planning, with some control considerations such as steer control and obstacle avoidance.

Buniyamin et al. (2011) illustrated clearly the three key properties of path planning in the work on mobile robots, on whether if the path planning is:

(1) static or dynamic
(2) local (online) or global (offline)
(3) heuristic or complete

For a global path planner, the information of environment is a priori, based on which a map/grid can be sketched out. For a local planner, it faces unknown environments and relies on observation from onboard sensors (Thi et al. 2016). Path planning is heuristic when it approximates the true value of optimality in order to gain efficiency. This property is closely related to the nature of algorithms deployed.

Due to the project scale limitation, this project is primarily concerned with the static, global and heuristic path planning problem that does not require any sensor input, for autonomous cargo ships. The problem is defined in this way so that it can be completely solved with the current computing power and resources available to the author.

1.4 Aims and objectives:

The aim is to develop a static, global and heuristic path planning solution for autonomous cargo ships in realistic settings. A number of objectives to fulfil are listed below:

(1) The solution must be shortest feasible path between current position to target destination.
(2) The planned path must avoid any obstacles present, by maintaining the obstacles outside the minimum collision area (MCA) of the ship.
(3) The path planning must take into account the realistic dynamics of cargo ships.
(4) The path planning must consider realistic environmental interferences i.e. due to waves
(5) The path planning must be concise and cheap to run, considering the implementation potential on an actual ship where computing power maybe limited.
(6) The path planning must also be highly generalizable, meaning it can easily adapt to different ships and environments.

2. Literature review

Because that the core principle of many path planning algorithms can transcend beyond boundaries of disciplines, and given the limited resources on ship path planning, literatures not only covering USV, but also on other robotic applications are analysed.

2.1 An overview of different methodologies:

Figure 2 classifies different path planning algorithms used across the fields of robotics. Traditionally, path planning is solved by deterministic methods like Artificial Potential Field (Sfeir et al., 2011), Rapid-exploring Random Tree (Lee et al. 2014) etc. Generally, these are all model-based approaches that require either a potential field, a set of rules or a tree search. Typically, most of these deterministic methods can achieve after a long calculation an optimal path, while avoiding obstacles, without considering the feasibility of the solution when applying to a real-world agent that be imposed with a series of dynamics constraints. Sometimes, deterministic methods may fail to produce a solution, due to problems like local minima. In recent decades, heuristic-based methods start to gain popularity. Methods such as A* algorithm (Erckens et al. 2010), neural network (Joshi et al. 2011), fuzzy logic (Hong et al. 2012), naturally inspired algorithms (X. Chen et al. 2013) have all been developed to solve path planning more efficiently. Additionally, the advent of data-driven approaches such as reinforcement learning (RL) with its derivatives like Q-learning (C. Chen et al. 2019), Deep Q Network (Y. Cheng et al. 2018), Deep Deterministic Policy Gradient (Guo et al. 2020), Actor-Critic (Woo et al. 2019) algorithm brings an entirely new realm of solutions to path planning.

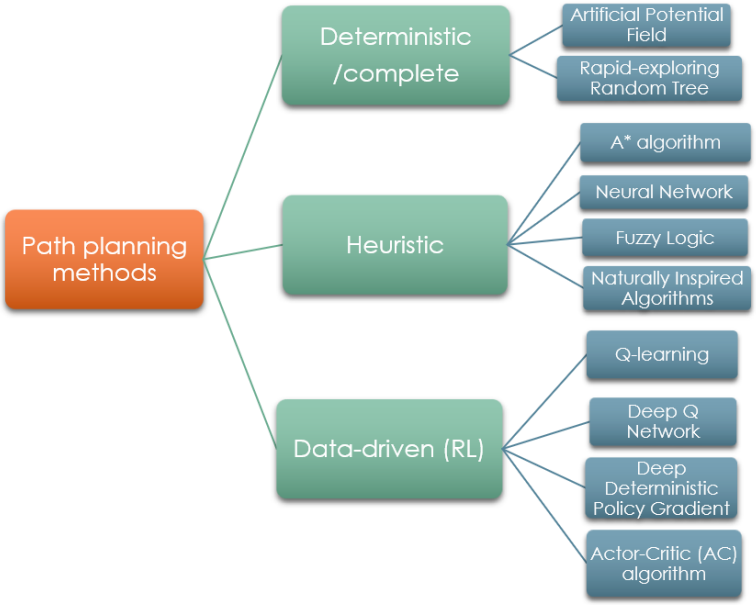

Figure 2 . Classes and types of path planning algorithms.

Preliminary literature reviews are carried on traditional and heuristic approaches (Appendix A and B).

2.2 Literature review on RL-based methods:

The superhuman performance of Q-learning in Go games (Silver et al. 2016) has offered researchers a new perspective in approaching the problem of ship path planning. As Carreras et al. (2005) pointed out in his work on RL-based AUV, in short, Q-learning is a special form of RL in that it observes states and take optimal actions then receives a reinforcement (reward) via agent-environment interaction. The off-policy nature of Q-learning also benefits agent's path planning in realistic settings because this allows each action at each step can be determined by the current value function without a prescribed

strategy. Q-learning combines the strength of sampling from Monte Carlo (MC) and bootstrapping from Dynamic Programming (DP) to simulate the agent's ability of learning from experience with efficient, finite gains even for non-episodic tasks. C. Chen et al. (2019) adapted tabular Q-learning to the problem of model-free path planning for smart cargo ships in a simulated grid waterway. In this work, path planning takes place in a 2D discretized grid. A series of steps are taken to simplify the simulation of ship dynamics and the environment to allow Q-learning to be easily fitted to solve the problem:

(1) The ship agent's control is modelled by the Nomoto first order system, where constant speed, small turning angles etc. are all realistically assumed for a cargo ship.
(2) The action space for Q-learning is discretized into a few possible rudder angle inputs, considering the turning difficulty of cargo ships.
(3) The state space for Q-learning is also designed to have discrete distance and angular representations.
(4) The constant negative reward setting for hitting obstacle and not meeting the target is designed to constraint the agent to pursue the path planning objectives.

Other more traditional algorithms like A* and RRT were also implemented in this work to contrast the Q-learning performance. As a result, from figure 3, the yellow line representing Q-learning produced a smoother path with fewer sharp turns compared to the blue and green lines, which is more practical in terms of conforming to the actual cargo ship's dynamics constraints.

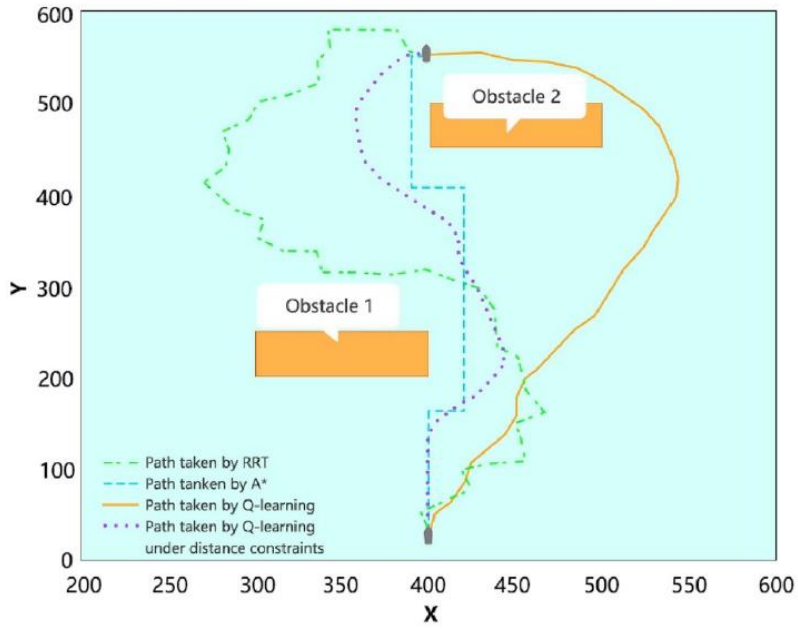

Figure 3 . A comparison of paths planned by Q-learning and other traditional algorithms (C. Chen et al. 2019).

The reliance on lookup table for Q-learning means it potentially faces the problem of exceeding memory space if the number of states gets too large. Deep Reinforcement Learning (DRL) combines the generalizing power of Neural Networks (NN) together with the decision-making ability of RL. Specifically, one type of DRL structure named Deep Q-Network (DQN) combines Deep Neural Networks (DNN) with Q-learning. The basic idea of DQN is the use of deep networks for a non-linear estimation of value function which in traditional Q-learning would be memorized in a tabulated form.

Training data is provided by the Q-learning's agent-environment interaction and a weight matrix is learned via backpropagation followed by SGD.

However, this naïve form of DQN will potentially lead to disastrous results. This is revealed in Sutton and Barto (2018), a book which is often referred to as the gold standard of RL. He mentions that the danger of instability and divergence will occur whenever three elements of RL are combined to form the "deadly triad":

- Function approximation, i.e. non-linear mapping of NNs
- Bootstrapping, i.e. the use of discounted gains in DP or Q-learning
- Off policy training, like that of Q-learning

Apparently, naïve DQN combines the traits of all three. This consequent problem of overfitting and divergence is overcome in DeepMind's success of adapting DRL to play Atari games (Mnih el at. 2013). In this ground-breaking work, experience replay is introduced perform batch updates in the Q-learning step. The significance of this move lies in the storage of past experiences forming the replay memory which proves to be vital in preventing overfitting caused by misdirection from recent experience. Target network is also used to stabilize the target function. This requires the use of two identical networks, one updating the other. The target network is the one with fixed parameters, the other one updates constantly. The latter one will only update the target network until a given number of steps has been reached.

One of the more prominent research in applying DQN to solve USV problem is Y. Cheng et al. (2018)'s DQN with state-of-the-art Convolutional Neural Network (CNN) obstacle avoidance architecture for unmanned marine vessels. This work stands out due to the real-time obstacle avoidance nature, which essentially means the computation for control policy takes place simultaneously with state data acquisition from onboard sensors. Some assumptions made are:

(1) The vessel is modelled with three degrees of freedom (DOF), associated motion in heave, roll and pitch are ignored.
(2) The time interval between control sequence are assumed to be constant.
(3) The agent can receive updated information at regular intervals.

Figure 4 illustrates the obstacle avoidance architecture. CNN performs the role of data fusion. Its design incorporates sparse connectivity and shared weights, enabling CNN to deal with high dimensional input space, which in this case allows the observation matrix to be designed very large. After a state ID for each input is generated by CNN, Q-learning can be used as the decision-making module. Experience replay and batch updates are added when using NN approximated action value function in minimizing the reward loss. To achieve realistic avoidance goals, a series of comprehensive reward functions are investigated in the work. For example, distance reward function L is used, computing distance to destination at every step. The closer the target, the less negative the L will become. A collision reward function is also used, computing how close the ship is from obstacle, the

closer the obstacle, the smaller the reward gets. Other reward functions like the drift reward function punishes the agent when sway speed far exceeds the surge.

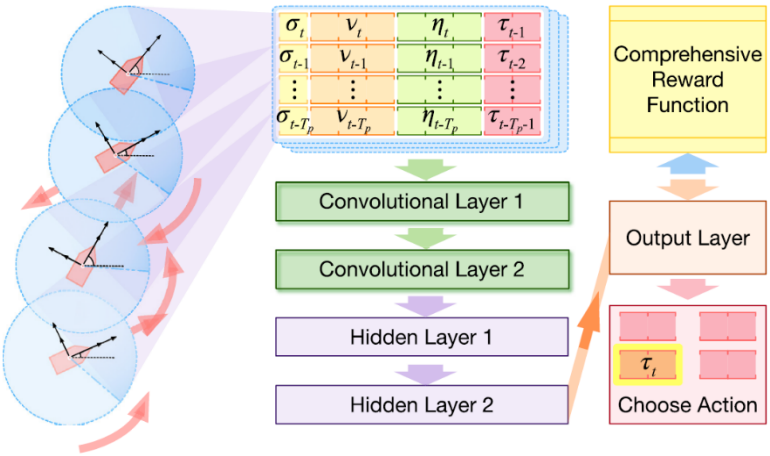

Figure 4. The DQN obstacle avoidance architecture with CNN (Y. Cheng et al. 2018).

To expand the DRL solution into the domain of continuous control, Deep Deterministic Policy Gradient (DDPG) is developed to accept continuous action space. But before DDPG is introduced in this report, its basis, Actor-Critic (AC) algorithm needs to be understood. The AC framework shown in figure 5 consists of a policy network and an evaluation network. The Actor network takes in state data and returns an action. Environment then generates a reward based on that action. The Critic network does the job of evaluation. It receives the action and the reward, computes an evaluation based on value function and improves the Actor network's action strategy. The strength of this structure is the achievement of optimal control policy with gradient estimation at a lower variance.

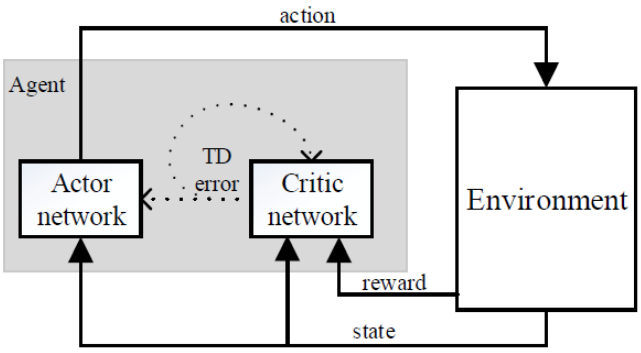

Figure 5. The standard AC structure (Guo et al. 2020).

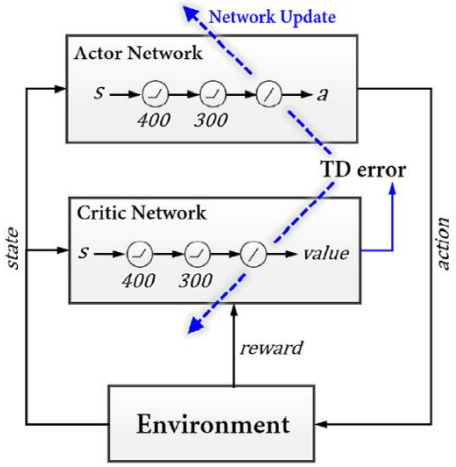

Figure 6. The DDPG control policy network used by Woo et al. (2019).

DDPG conceptually means the deep network structure follows the deterministic policy gradient method. DDPG shares the similar double network structure like DQN, with online network and target network. Action is randomly selected from the continuous action space, according to the learned action distribution. DDPG takes in state inputs as usual, but prior to action selection a random noise is added. Environment then responds by feeding a reward back. States, actions, rewards, next states are all stored in an experience buffer pool. Meanwhile, deep NN randomly changes actions by sampling from the experience pool. Its parameters are updated iteratively through Gradient Descent

(GD). In general, DDPG demonstrates high stability and accuracy, even when NN receives large scale inputs.

There are a number of successful applications of DDPG in ship path planning. Woo et al. (2019) used the DDPG to simulate the path following experience of USV during trials, using the DDPG structure shown in figure 6. The two feed-forward NNs each consists of two hidden layers with 400 and 300 neurons, with the same activation function rectified linear units (ReLU) used for each node. Guo et al (2020) first investigated the standard DDPG, then came up with a modified structure combining APF with DDPG for unmanned ship path planning under international maritime collision avoidance rules (COLREGS). The proposed APF-DDPG method proves to converge faster with higher stability. Zhao et al. (2019) also used DDPG to develop a multi-ship collision avoidance algorithm that is COLREGS compliant.

2.3 A conclusion from literature review:

Traditional, heuristic, and the more novel RL-based approaches to path planning for USV have been reviewed in the above sections. This project's approach to the problem will need to be determined by a qualitative cross-comparison of all those approaches reviewed above. For sake of simplicity, Q-learning, DQN, DDPG are all referred to Q-learning based methods as they all root from Q-learning.

Requirements/ (weightings)	APF	RRT	A* algorithm	Fuzz-neural method	Naturally inspired algorithm	Q-learning based methods
Ability to generalize (4)	3	1	2	5	4	6
Ability to consider realistic ship settings (3)	3	4	1	5	2	6
Performance of path planning (2)	2	4	1	5	3	6
Conciseness of algorithm, i.e. computationally light (1)	4	3	6	2	1	5
Weighted total	29	27	19	47	29	59

Table 1. The decision matrix for methodology selection.

Results shown in the decision matrix in table 1 are calculated by a multiplication of ranked importance weightings of requirements and the ranking of all six possible methods themselves. The requirement setup reflects the project objectives discussed in introduction. These rankings are obtained qualitatively from conclusions of the comprehensive literature review conducted above. As can be seen, Q-learning based methods have the highest weighted total. Therefore, this project will adopt Q-learning as the most suitable method to solve the problem of path planning for cargo ships. In particular, the literature provided by C.Chen et al. (2019) on Q-learning based path planning for autonomous cargo ships converges with the interest of this project, thus it will form an initial basis for the unfolding of this project.

3. Planning of work

3.1 Project plan:

Following the project objectives, and after extensive literature reviews, the project is planned with a flowchart shown in figure 7. The project plans to implement a tabular Q-learning agent first as the basic RL approach and then to proposes a DQN structure that is more applicable to real-world scenarios.

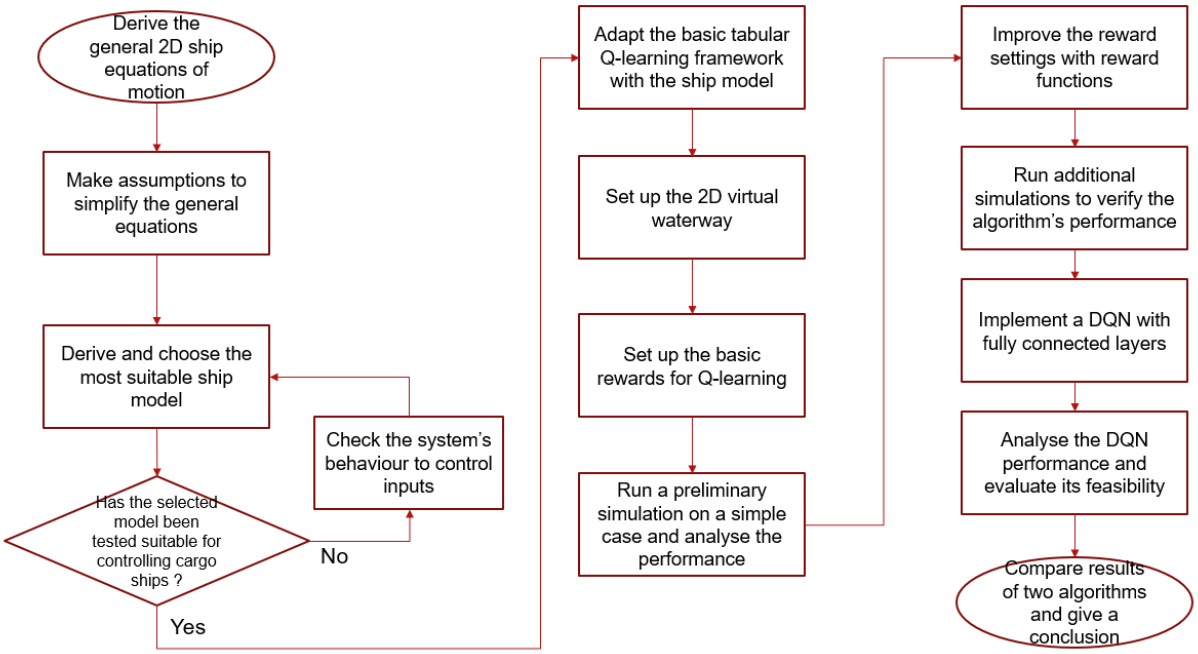

Figure 7. The project flowchart.

3.2 Success criteria:

Given all above, the project is successful if the below list is completed in the sequential order:

(1) A suitable model for cargo ship dynamics is chosen.
(2) The environment is reasonably set up.
(3) The reward setting is well established.
(4) The Q-learning algorithm can be fully implemented with a converged result.
(5) The path planning and obstacle avoidance by Q-learning is optimal.
(6) A simple DQN can be successfully implemented.
(7) The DQN can achieve an acceptable performance (although projected to be lower than Q-learning).

4.The proposed approach with Q-learning

4.1 The Q-learning environment and ship model setup:

First, a series of assumptions regarding realistic cargo ship dynamics can be made to simplify the problem without a great loss of generality:

- The ship is symmetric around the $x-z$ plane.
- The fixed origin of the ship body has the x, y coordinates of the Center of Gravity (CGx and CGy).
- The ship mass is homogeneously distributed.
- Constant average forward speed (surge).
- The sway speed v is small.
- The yaw rate r is small.
- No coupling from roll caused by the yaw rate.
- The rudder angle δ is small.

These assumptions for 2D ship model with constant surge and difficulty in steering are suitable for large and slow ships like cargo ships, and they are also made in Carrillo et al. (2018). The above assumptions also imply that for the 2D ship control with full control vector $[u\ v\ r]^T$, the sway control is not actuated ($v = 0$), meaning the ship is underactuated. The underactuated simplification is also adopted in the obstacle avoidance algorithm in (Y. Cheng et al. 2018).

Based on these assumptions, and deriving from the general ship motion equations (Appendix D), the ship's motion can then be modelled by the first order Nomoto model:

$$\psi = K\delta\left(t - T + T\,e^{\left(-\frac{t}{T}\right)}\right) \qquad (1)$$

The Nomoto model in eqn. (1) directly maps the control input rudder angle δ to course angle ψ. The control characteristics of the first order system of Nomoto KT model is investigated with matlab simulations (Appendix E) to ensure the suitability of using Nomoto model for the cargo ship focused in this project.

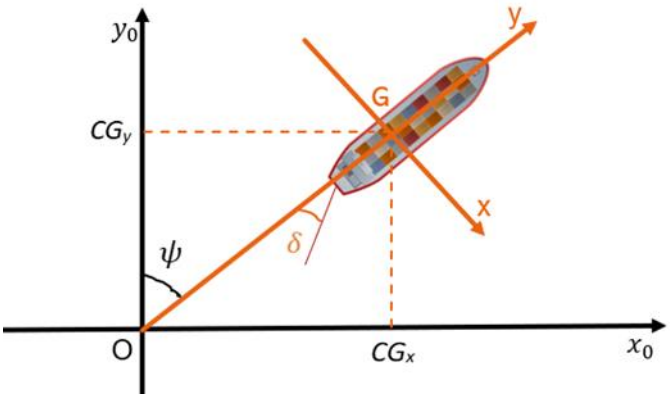

Figure 8. 2D Modelling of the cargo ship motion in both cartesian and ship-fixed frame, ship picture from (Anonymous, 2020).

Now, to incorporate the 2D ship motion model shown in figure 8 with Q-learning's framework, one of the most crucial consideration is the formulation of state space S and action space A, which is inevitably linked with domain expertise in ship dynamics. In this project, given the Nomoto model maps input to ship angles, the state space S is designed to be comprised three elements:

(1) Horizontal coordinate x of the ship in the 2D cartesian frame

(2) Vertical coordinate y of the ship in the 2D cartesian frame
(3) Course angle ψ of the agent

And the action space A is simply the range of rudder angle values δs that the ship agent could take.

Variable	Description
X_0OY_0	Represents the waterway on a 2-D cartesian plane
XGY	Represents the ship-fixed coordinate system w.r.t. the ship center of gravity G
CGx	The horizontal projection of the ship center of gravity
CGy	The vertical projection of the ship center of gravity
δ	The rudder angle of the ship
ψ	The course (heading angle) of the ship

Table 2. List of variables associated with the ship model.

For convenience of implementation, the state s of the ship in the 2D waterway can be represented by a tuple of format (x, y, ψ). Any point on the map hence any state of the ship can be represented by this format, from these kinematic relationships that were also adopted in C. Chen et al. (2019):

$$x_{(t+1)} = x_t + \int V sin(\psi) dt \quad (2)$$

$$y_{(t+1)} = y_t + \int V cos(\psi) dt \quad (3)$$

where V is the surge speed(constant), t means the current time step and $(t+1)$ is the next time step. Other variables are explained in table 2. Combining eqn. (2) and eqn. (3) with the previous Nomoto model:

$$\psi = K\delta\left(t - T + T\, e^{\left(-\frac{t}{T}\right)}\right) \quad (1)$$

The equation set made of eqn. (1), (2) and (3) allows ship agent to take an action by choosing a rudder angle input δ from the action space A, and compute all three of the elements required for the complete description of ship state in any time step t: $s_t = (x_t, y_t, \psi_t)$.

Exact size for the environment needs to be decided. In order to provide a large enough waterway for the ship agent to explore, but meanwhile to keep the size of state space S small to reduce the entry points to Q-table to alleviate computational burden, eventually for the 2D cartesian map the x-range is set be 100, y-range is 100, and the course angle ψ range is set to be 360°. Due to the discrete nature of Q-learning, a discretization of the state space S needs to be carried out. The x-y coordinates of the ship are discretized according to the process shown in figure 10, in which along both x and y direction, the 2D cartesian map is segmented into a discrete grid representation, with each grid having the size of 1x1. And within each one of those grids, as shown in figure 9, the third state element ship course ψ is divided into 36 independent slices, each representing 10° of course angle change, together making up the 360° as one revolution. After such discretization of the S, its size can be easily calculated as $size\ of\ S = (size\ of\ x) \times (size\ of\ y) \times (size\ of\ \psi) = 100 \times 100 \times 36 = 3.6 \times 10^5$.

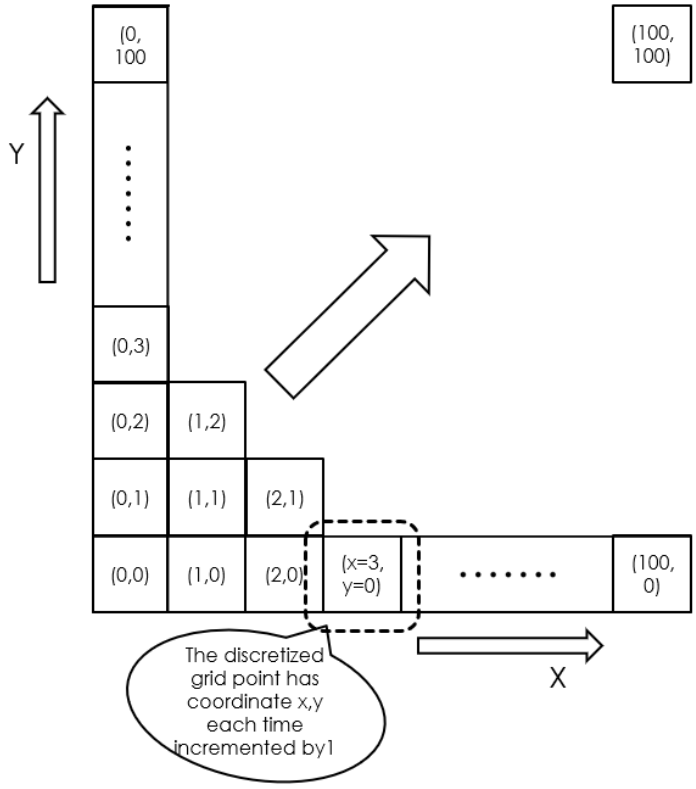

Figure 10. The discretization of the 2D cartesian map.

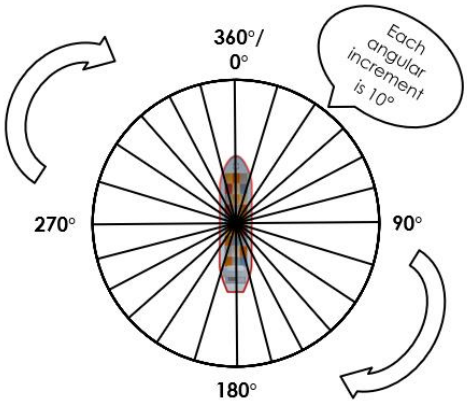

Figure 9. The discretization of the ψ, for every grid point shown in figure 10.

The range of action A is essentially the range of the rudder angle input δ, and its confinement will reflect the dynamics consideration of the cargo ship simulated in this project. As the primary reference of this project, in the work C. Chen et al. (2019) the maximum rudder angle input $\delta_{max} = \pm 35°$, and the action space A is discretized into a list of 5 possible rudder angle choices for the ship agent: $\partial \in [-35°, -15°, 0°, 15°, 35°] \; for \; all \; \delta \; in \; A$. In one of the pioneering work of development for autonomous ship (Sutton et al. 1997), a similar restriction on rudder servomechanism analysis for ship dynamics is adopted. It claimed that from an examination from rudder motion of a warship, the rudder rate is usually limited to $6°/s$ for the linear region and the absolute rudder turn cannot exceed $35°$. The $35°$ maximum rudder angle confinement is supported in other works, and it is reasonable to apply this kind of restriction to this project as cargo ships are typically large and slow and difficult to turn. After research, the same rudder angle discretization is adopted in this project, with $A = [-35°, -15°, 0°, 15°, 35°]$ and the size of A equals 5.

Now, the ship itself needs to be simulated in the environment as well. The ship which can be viewed as a near elliptic shape in a 2D x-y plane is simulated to be a rigid body rectangle with length L matching the exact length of the ellipse, and with width W matching the width of the ellipse, as shown in figure 11 (note that the ship itself will still be shown as a purple ellipse in later visualisations). The rigid body's center which is where the (x, y) coordinates are tracking to is located right at the center of gravity CG.

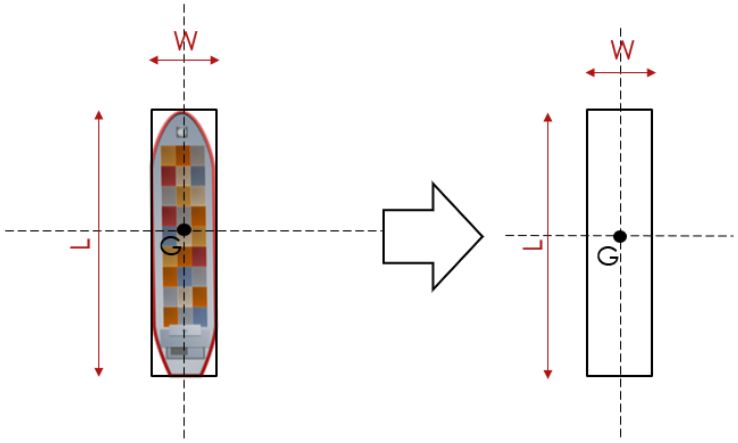

Figure 11. The modelling of ship object as a rectangular rigid body.

The dynamics of the cargo ship is chosen to be modelled by the first order Nomoto KT model, as mentioned before. To enable the simulation, the exact values of parameters K and T conventionally are obtained through a series of experiments on an actual scale-downed ship design in a towing tank. However, this project is rather more focused on the validation of the feasibility of the Q-learning based solution applying to the class of cargo ships. Therefore, pursuing the same goal as in C. Chen et al. (2019), the simulation will also be based on the same cargo ship design analysed in that work, meaning using the same K, T parameters:

- The turning ability coefficient K=0.08.
- The turning lag coefficient T=10.8.

4.2 The preliminary reward setup:

The design of the comprehensive reward R in a Q-learning algorithm is the driving force behind the agent to help achieve the desired objectives. The "unwanted" actions will receive far less rewards than the "correct" ones, and by always picking the action with the highest reward, the agent then can be led to the optimal solution. In this project, the objectives defined earlier can be condensed into two major points: (1) find the shortest path to target meanwhile (2) avoid all the obstacles on the way. To reflect these goals, the preliminary design for reward R is laid out in table 3.

There are in total 4 possible scenarios that an action taken might lead the ship agent into. In scenario 1, the agent takes an action that is able to lead agent to terminate at the target destination and the episode is "won", thus a very high positive reward of 1000 is designed to encourage such actions and propagate the increase in action values to other nearby states.

In scenario 2, the ship agent takes an action that causes the ship to hit an obstacle. A strict function is designed to detect if the ship has actually run into obstacle by detecting if any coordinates within the obstacle area has "intruded" into the minimum collision area (MCA) of the ship. The area of MCA is exactly covered by the rectangular rigid body model of the ship itself in figure 11. In order to force the agent to avoid such actions, a very negative reward of -1000 is applied.

Similarly, in scenario 3, the agent is out of bounds, receiving a negative reward of -100. The out-of-bound detection works in the same way as collision detection. The smaller magnitude of this reward is intentionally designed in this way to give the agent the "impression" that hitting an obstacle is more

severe than being out of the bound, so that whenever the agent confronts a situation where the only possible actions would either lead to hitting obstacles or exiting the map, the agent will still be able to firmly avoid the obstacle.

Finally, in scenario 4 where the ship moves and nothing happens, the agent receives a reward of -1, which is designed to ensure that for every extra step the agent takes, the cumulative reward will drop. This in effect will "hurry" the agent to force it to pick the shortest possible path.

Scenario number	Reward scenarios	Rewards	Episode status	Agent's next location
(1)	The ship's center of mass reaches the goal	receives a constant positive reward $r = 1000$	episode is terminated	none (as episode is ended)
(2)	Any side of the ship's rectangular rigid body touches or intersects with any edge of the obstacle present	receives a constant negative reward $r = -1000$	continues as usual	ship moved to the starting position and start over
(3)	Any side of the ship's rectangular rigid body touches or intersects with any boundary of the map	receives a constant negative reward $r = -100$	continues as usual	ship moved to the starting position and start over
(4)	The ship moves a step in open water	receives a constant negative reward $r = -1$	continues as usual	ship moved to next state according to the ship dynamics

Table 3. The initial design of the reward rules.

4.3 The agent's learning process:

Now, everything is set for the Q-learning algorithm. The implementation of all simulations in this project is done In Python3.6, as it is straightforward to use and provides many useful machine learning packages. The Q-learning algorithm starts by initializing an empty table for storage of action values for all states and all actions. Note for the terminal state there is no need for action value. The number of entry points of this table can be calculated by state space S size multiplied by action space A size which is $3.6 \times 10^5 \times 5 = 1.8 \times 10^6$. Then, the agent is simulated to pick actions and explore the environment. The agent-environment interaction generates the reward and next state and send this information back to the agent. The agent then updates its Q-table (action value table). The update rule is the core of the learning process and its defined as:

$$Q_{(s,a)} = Q_{(s,a)} + \alpha \cdot \left[r + \gamma \cdot max_a Q'_{(s',a)} - Q_{(s,a)}\right] \quad (4)$$

For completeness, a derivation of the Q-learning fundamentals is included (Appendix C). The episode terminates when agent reaches the target destination. The whole process repeats until the given number of training episodes has been reached. The final policy would theoretically be the optimal policy, as long as the action values have converged. Pseudo codes are shown below in Algorithm 1.

Algorithm 1. The Q-learning algorithm

---

1. Initialize parameters step size $\alpha \in (0,1]$, discount $\gamma < 1$, small epsilon $\varepsilon > 0$.
2. Initialize $Q_{(s,a)}$, for all states $s \in S$, for all actions $a \in A$
3. Initialize $Q_{(s_{terminal},a)} = 0$ (for all $a \in A$)
4. Loop for each episode:
5. Initialize S
6. Loop for each step of the episode:
7. Choose action $a_{(t)}$ from A, following the ε-greedy policy
8. Take the selected action $a_{(t)}$, observe reward $r_{(t)}$
9. Move to the next state $s_{(t+1)}$
10. Update the Q-table using equation (4)
11. Update $s_{(t)} = s_{(t+1)}$, $a_{(t)} = a_{(t+1)}$
12. Until $s_{(t)} = s_{terminal}$
13. Until the training episode number has been reached
14. Return the final policy: $\pi_{(s)} = \underset{a}{argmax}(Q_{(s,a)})$

One thing to note about the exploring-learning process of the agent is the epsilon(ε)-greedy action selection policy used in agent's every move. The reason behind its use directly links to the problem of exploration-exploitation trade-off, the ultimate RL struggle mentioned in Sutton and Barto (2018). The use of ε-greedy can balance this issue, by instructing the agent to select an action a in a way that:

$$\begin{cases} selects\ the\ a\ that\ maximises\ Q_{(s,a)}\ with\ probability\ (1-\varepsilon) \\ \quad randomly\ selects\ an\ a\ from\ A\ with\ probability\ \varepsilon \end{cases}$$

In this way, the agent can from time to time randomly explore other unvisited states, instead of being only "greedy" with respect to action values. This reduces the chance of agent being too "short-sighted" to only focus on short-term gains, thus increasing the chance of finding the global optimum. In this project, the value of ε is 0.1, which is the conventional value to use.

5. Result and analysis for Q-learning simulations

Using the adapted Q-learning approach introduced in the previous section, the actual ship path planning process can be simulated in different cases with maps filled with different obstacles settings. Note the maximum 2D map size for all cases is set to 100x100 as mentioned before. Each case differs from each other only at the establishment of obstacles and starting/destination locations for the agent. There are in total 4 different cases being tested in this section. Theoretically, the performance of the algorithm should be approximately on a level of parity for various cases as one of the advantages of Q-learning is that it is not environment-dependent.

5.1 The first simulation for a single obstacle case (case 1):

The first case is designed to be plain and simple to test the basic performance, also acting as a reference case. As can be seen from the virtual environment in figure 12, the ship agent's starting position shown as the white dot is set to be at (20,10) with the course heading directly to the north. The termination (target) shown as the green dot is set at (90,90) in the top right corner. The obstacle is designed to be a rectangle with length 50 and width 15, mimicking a small island or buoy. The obstacle is located right above the ship, with its lower edge less than 20 grid lengths away from the midship. For the sake of visualisation, the ship itself is shown every 5 steps in a form of a purple ellipse whose heading is the ship course angle ψ.

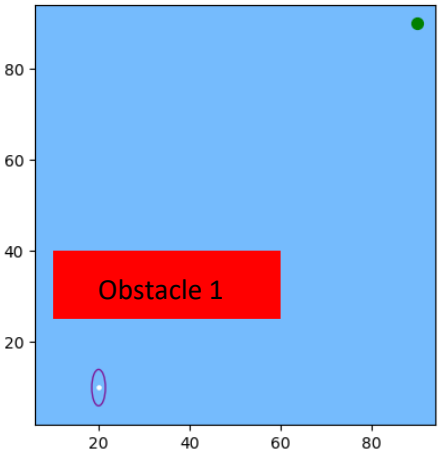

Figure 13. The single obstacle environment setting, for the first simulation case.

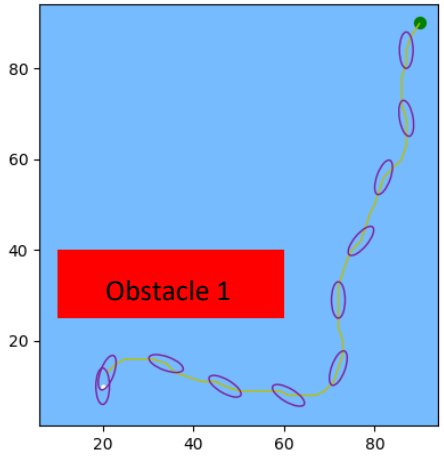

Figure 12. The final path planned by the agent, visualised in the virtual environment, for the first simulation case.

Table 4. The sequence of actions taken by the agent in the final policy.

The agent's final policy π_{best}											
Rudder angle taken at step (1-8)		Rudder angle taken at step (9-16)		Rudder angle taken at step (17-24)		Rudder angle taken at step (25-32)		Rudder angle taken at step (33-40)		Rudder angle taken at step (41-48)	
1.	35°	9.	0°	17.	-35°	25.	15°	33.	0°	41.	-35°
2.	35°	10.	-15°	18.	-35°	26.	15°	34.	15°	42.	-15°
3.	35°	11.	35°	19.	-35°	27.	0°	35.	-15°	43.	35°
4.	35°	12.	-15°	20.	-35°	28.	35°	36.	0°	44.	0°
5.	35°	13.	-35°	21.	-35°	29.	0°	37.	35°	45.	35°
6.	15°	14.	15°	22.	0°	30.	15°	38.	0°	46.	-35°
7.	35°	15.	0°	23.	-35°	31.	15°	39.	-35°	47.	35°
8.	-35°	16.	35°	24.	-35°	32.	-35°	40.	-15°	48.	Target reached

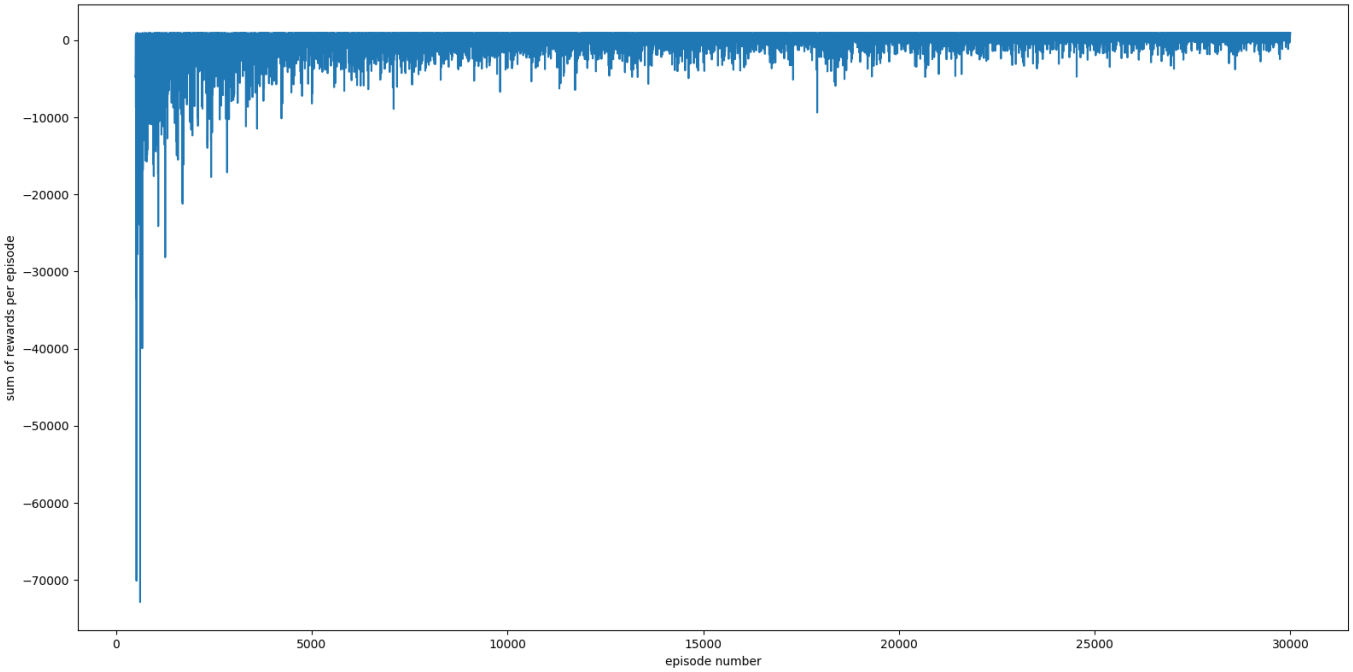

Figure 14. The reward curve for the first simulation case.

After 30,000 episodes of training, the agent picks the final path shown in figure 13. And the final control policy is listed in table 4, showing a sequence of actions selected by the agent. Figure 14 gives the proof of convergence (or near convergence) after 30,000 episodes of training.

First, the importance of the cumulative reward curve in figure 14 lies in its ability to detect whether if the agent's policy has stabilized and converged to an optimum. For the interest of visualisation, the reward sum is recorded only after the initial 500 episodes, because those extremely low values will break the plot as they tend to go too far down the y-axis. As the episode number increases, rewards become much less negative, and the magnitudes of oscillations stabilize. There is still a degree of oscillation, due to the fact that agent adopts the ε-greedy action selection policy at every step, meaning there is a 0.1 chance that agent chooses actions randomly and may reach undesired states. Overall, figure 14 confirms that the rewards are approaching a convergence, although the absolute convergence may be hard due to exploration caused by ε-greedy. The curve also shows that the 30,000 training number is sufficient since the curve does not change much after around 20,000 episodes. Further increase in the training times is discouraged as it will increase the computation time.

From the path planned in figure 13, it can be seen that the ship immediately takes some sharp turns starboard direction, heading north-east and east to avoid the obstacle. Then the ship breaks free to the left of the obstacle and reaches the target in a relatively straight path. This validates the basic performance of path planning and obstacle avoidance of the algorithm.

Table 4 shows that the agent takes a total of 47 steps to reach the target. There are a considerable number of 35° sharp rudder angle inputs, which is not so ideal for a real cargo ship. But considering the small size of the map and the proximity of the obstacle, it is an acceptable policy.

5.2 The improved reward setting with L reward function:

The initial all constant reward setting works quite well, as results from the first simulation can tell. However, in this project, a more dynamic reward setting i.e. the addition of reward function is investigated. The reward function proposed here is the distance reward function L, which is defined by:

$$L = \sqrt{(x_{target} - x_t)^2 - (y_{target} - y_t)^2} \qquad (5)$$

where x_t and y_t are the current location of the agent at step t. And the reward r_L calculated by L is:

$$r_L = multiplier \times L \qquad (6)$$

where the multiplier is negative and small to keep the r_L in scale with the maximum reward which is 1000 defined earlier in the project. This reward function will only be used in the scenario 4 in table 3, replacing the -1 constant reward. The design of this reward function reflects the primary objective of path planning which is to find the shortest path. Since L is essentially the distance between agent and target, the closer the agent comes to the target, the less negative the r_L term would become. The new reward setting can be summarised in table 5 below.

Scenario number	Reward scenarios	Rewards	Episode status	Agent's next location
(1)	The ship's center of mass reaches the goal	receives a constant positive reward $r = 1000$	episode is terminated	none (as episode is ended)
(2)	Any side of the ship's rectangular rigid body touches or intersects with any edge of the obstacle present	receives a constant negative reward $r = -1000$	continues as usual	ship moved to the starting position and start over
(3)	Any side of the ship's rectangular rigid body touches or intersects with any boundary of the map	receives a constant negative reward $r = -100$	continues as usual	ship moved to the starting position and start over
(4)	The ship moves a step in open water	receives reward r_L from distance reward function L	continues as usual	ship moved to next state according to the ship dynamics

Table 5. The new reward setting with a reward function L added.

To conclude which is the better reward setting, a cross-comparison is drawn between the reward setting with distance function L and the all-constant reward setting. The agent first interacts with the constant reward scheme then with the new scheme, for the same number of training episodes. In

both cases the agent uses the same map and starts and terminates at the same spots. The simulation is done using the exact same environment in case 1. Results are obtained after 30,000 episodes.

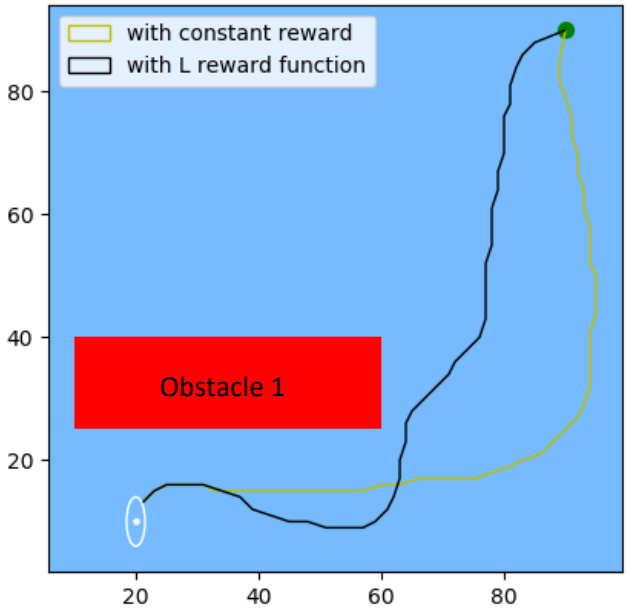

Figure 15. Comparison of paths planned from the two reward settings.

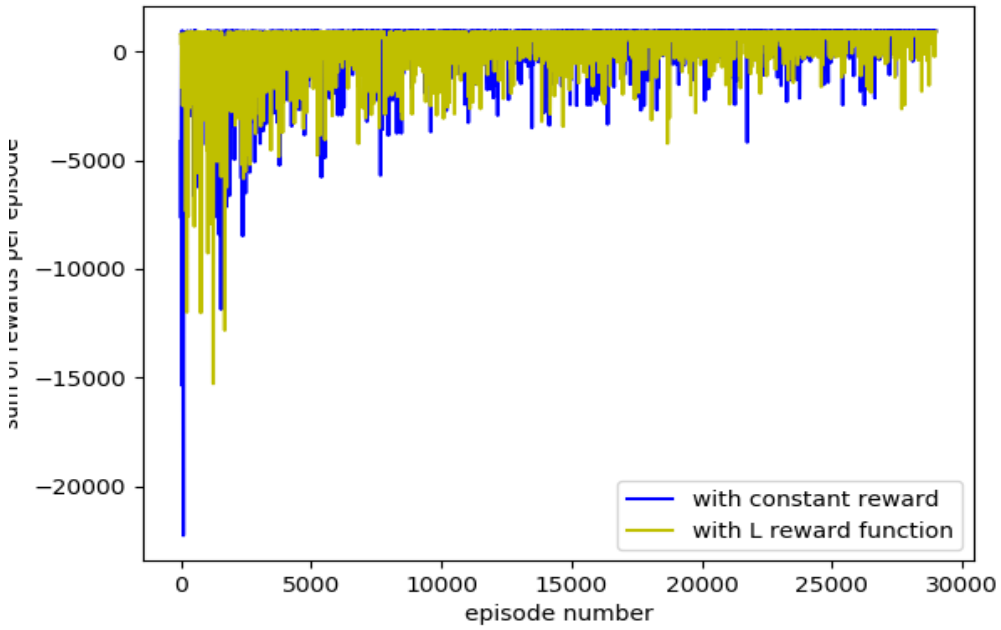

Figure 16. The reward curves for the comparison in figure 15.

From figure 15, the black curve representing the path planned with the new reward scheme, appears to be slightly shorter by taking the inner route than the yellow curve which is planned using the original scheme. In fact, agent with the L reward function (black) takes 2 less steps to terminate. A close look

at figure 16 reveals that the cumulative rewards for reward scheme with reward function (yellow) consistently appears to be less negative than the constant reward setting (blue), at all episode numbers. This implies that the addition of L reward function leads to a faster convergence. To reduce the effect of randomness in this comparison, the above simulation is repeated for 10 times, and the number of steps taken to terminate is recorded and aggregated for 10 times for each of the reward setting, shown in table 6.

Reward setting	Total steps taken for 10 simulations	Average steps needed to terminate
Constant rewards	471	47.1
With reward function L added	454	45.4

Table 6. The statistics of number of steps after 10 simulations.

It can be seen that on average, the addition of reward function L can help reduce the length of path planned by around 1.7 steps. Given all above, this proposed reward setting is more accurate in assigning the agent rewards at each step to entice it to target with a shorter path.

5.3 The simulation results for other cases:

Different obstacle loading conditions are also simulated. For the second case (case 2), there are now two rectangular obstacles being present in the environment, both of identical size with length 50 and width 10. They are placed in a manner to limit the possible paths to target to narrow passages and multiple sharp turns are required to pass. Compared with case 1 setting, this is a far more challenging scenario for the algorithm, causing a much longer computation time. The starting and target positions are identical to those in case 1. The training number is still 30,000 episodes.

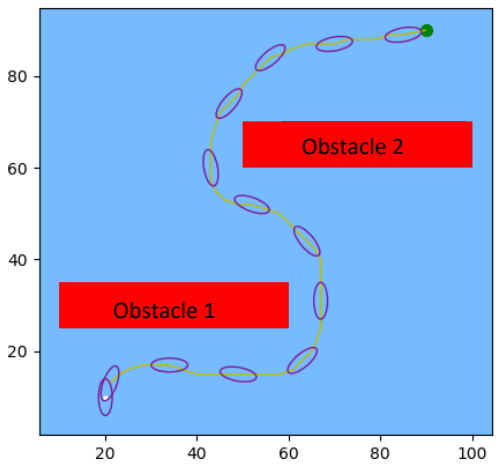

Figure 18. Simulation result for case 2, using constant reward setting for reference.

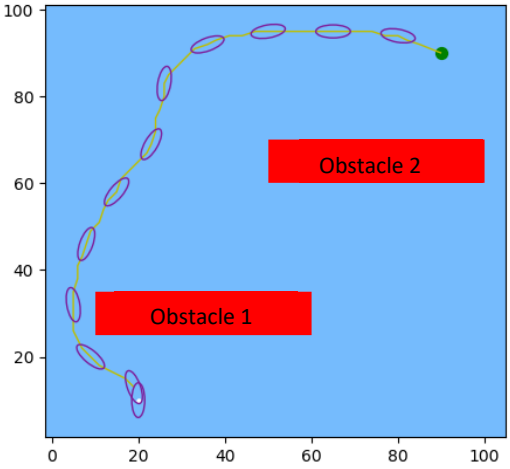

Figure 17. Simulation result for case 2, using L reward function setting.

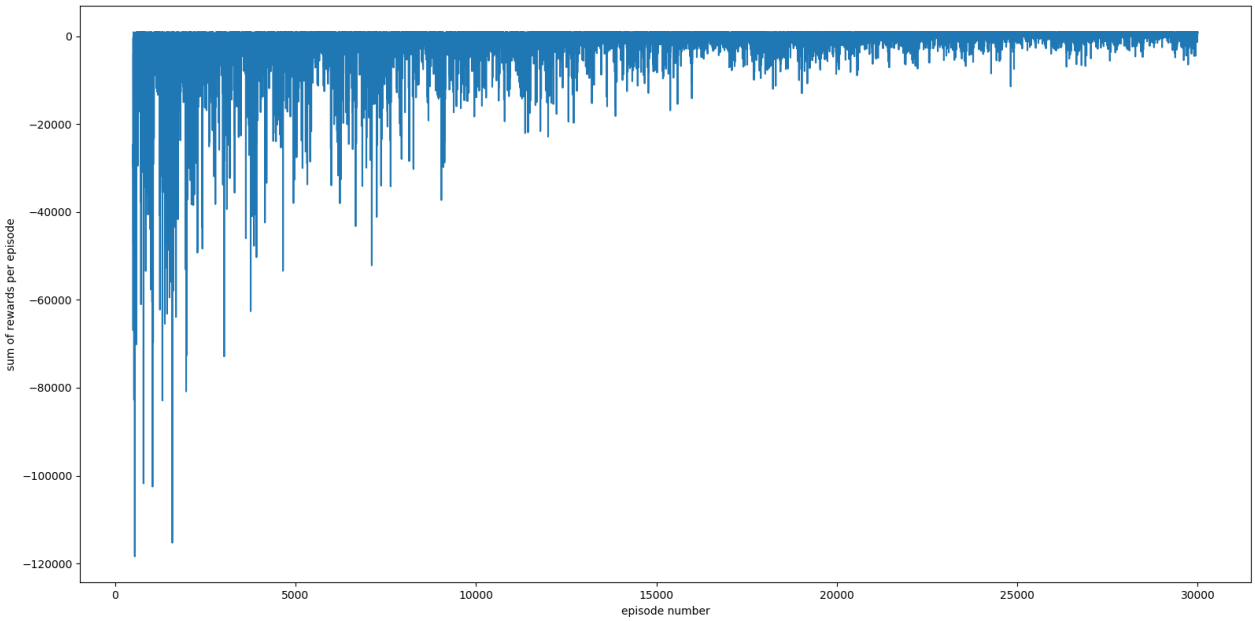

Figure 19.The reward curve for simulation in figure 18.

As a reference, figure 17 shows case 2 done with original rewards. Figure 18 shows the simulation of case 2 with the improved L reward function setting. Both paths successfully reach the target by executing a series of difficult maneuvers. The path produced using L reward function is slightly shorter as it skims through the narrow passage between obstacle 1's left edge and the boundary. Again, the shorter and smoother path in figure 18 verifies that the L reward function setting is the better one to use. Thus, all the rest of simulations (case 3 and 4) will be realised using the improved reward setting. Figure 19 confirms that the path planned in figure 18 is converging. It is also interesting to see that the initial rewards drop to much lower than that in figure 14 in case 1. This is because agent will initially bump into obstacles more in the more complex environments.

The third case (case 3) has the identical obstacle setting and target position in case 2, only differing at the agent's starting position. Instead of starting at (20,10), the agent now starts at (80,10), in the lower right corner.

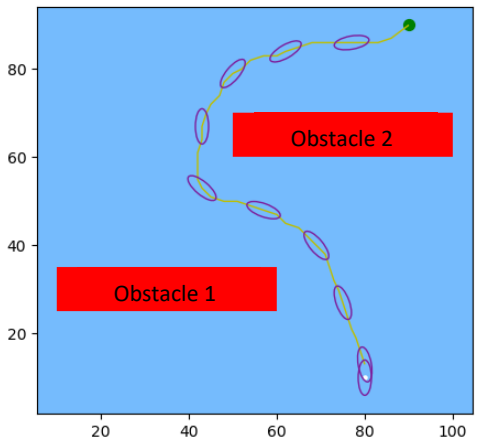

Figure 20. The result for simulation of case 3, with a different starting point.

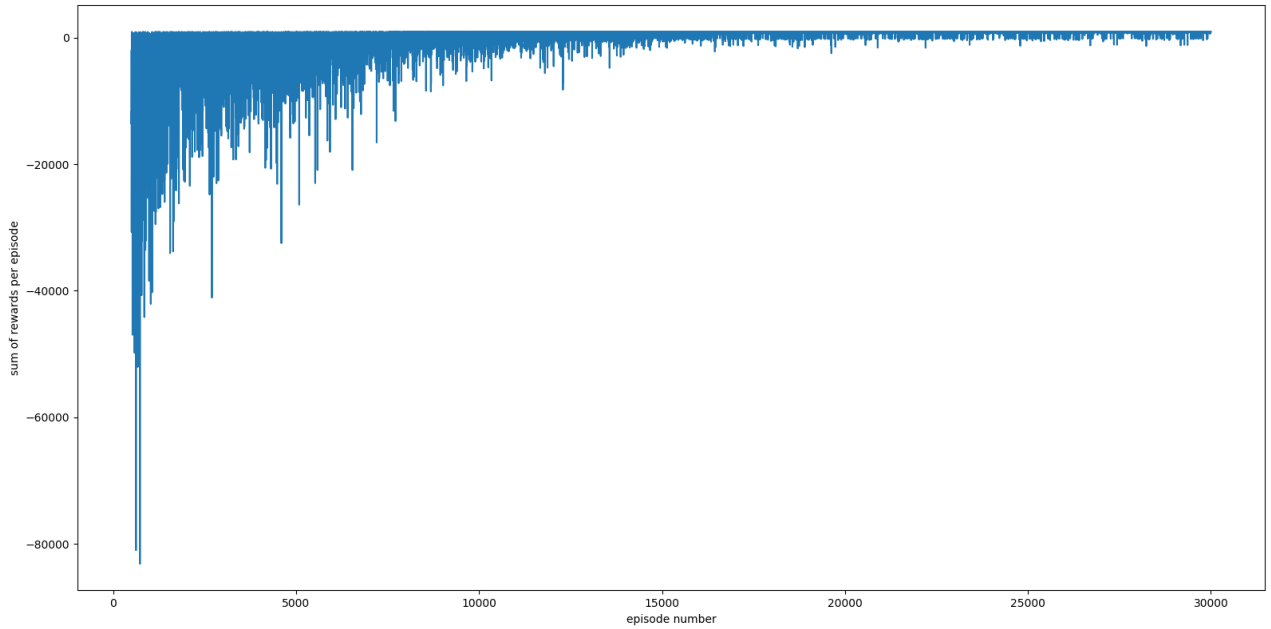

Figure 21. The reward curve for simulation in figure 20.

Figure 20 shows that regardless of the starting position, the algorithm can still find the optimal path to target. And the reward curve in figure 21 converges even more quickly than that in case 2.

Finally, case 4 loads the obstacles using a custom-built random obstacle generation of a random number of obstacles with random sizes at random location. Although an upper limit of 4 obstacles is set to avoid overfill the map so that agent might have nowhere to go. The starting and target positions are the same as case 3. Results are obtained after 30,000 episodes of training, shown in figure 22.

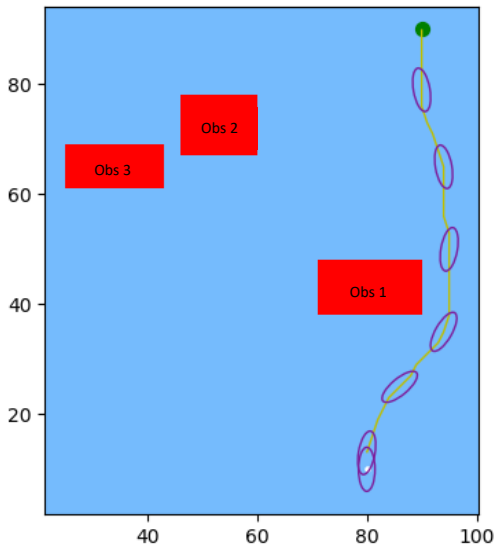

Figure 22. The simulation result for case 4, with random obstacles.

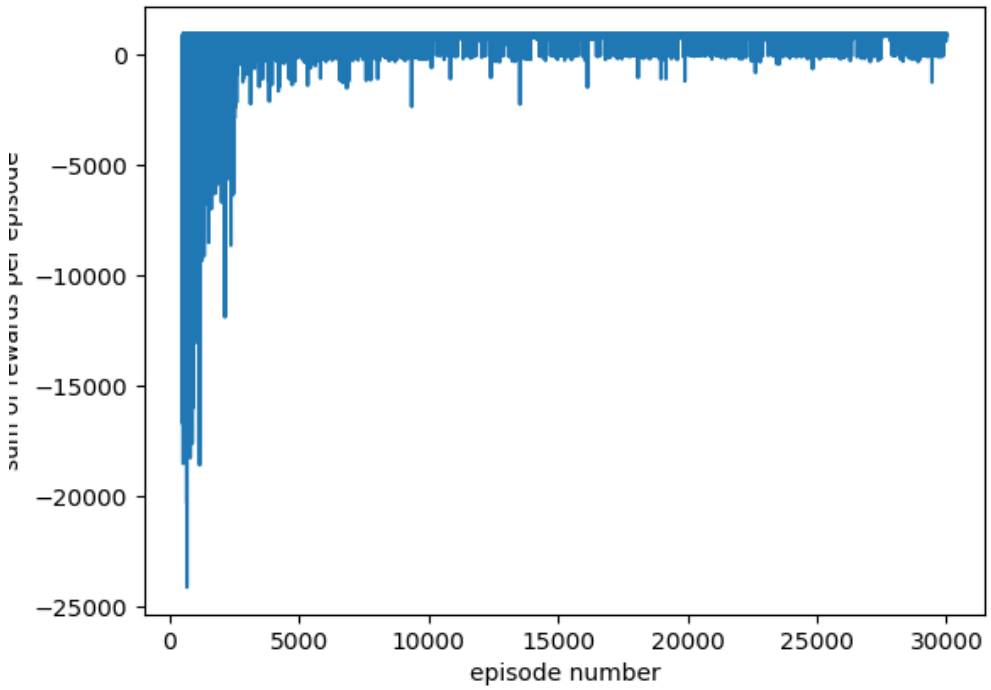

Figure 23. The reward curve for simulation in figure 22.

For the path planned in case 4, the agent is able to select a very straight-forward and efficient path that does not require much turning at all. And figure 23 shows the convergence.

6.The proposed DQN approach and results analysis

The memory limitation of lookup table makes Q-learning impractical for real-world applications. Thus, this project decides to expand on the C. Chen et al. (2019)'s basic Q-learning approach and fill the research gap to propose an innovative DQN architecture that, through NN's ability of generalisation, possesses realistic potential for onboard implementation.

6.1 DQN's environment setup:

The DQN's environment is the same as Q-learning before, including the same ship dynamics, state size of 100x100x36, rudder action discretization. However, from table 7, reward settings are changed with smaller rewards for all scenarios and an addition of a new direction reward function D:

$$D = \begin{bmatrix} \vec{x}_t \\ \vec{y}_t \end{bmatrix}^T \cdot \begin{bmatrix} \vec{x}_{target} \\ \vec{y}_{target} \end{bmatrix} \quad (7)$$

where $\vec{x}, \vec{y}$ are the horizontal/vertical directional vectors. And reward is:

$$r_D = multiplier \times D \quad (8)$$

This reward function gives higher rewards whenever agent's heading angle ψ is aligned with direction of the straight line connecting from starting to target position. The changes to small values with new reward function are designed to gently propagate the terminal rewards and generalize across states, as DQN is inherently sensitive to changes in serial rewards. Note the obstacle and boundary detections still work the same.

Scenario number	Reward scenarios	Rewards	Episode status	Agent's next location
(1)	The ship's center of mass reaches the goal	receives a constant positive reward $r = 10$	episode is terminated	none (as episode is ended)
(2)	Any side of the ship's rectangular rigid body touches or intersects with any edge of the obstacle present	receives a constant negative reward $r = -5$	continues as usual	ship moved to the starting position and start over
(3)	Any side of the ship's rectangular rigid body touches or intersects with any boundary of the map	receives a constant negative reward $r = -5$	continues as usual	ship moved to the starting position and start over
(4)	The ship moves a step in open water	receives distance reward r_L from distance reward function L	continues as usual	ship moved to next state according to the ship dynamics
		Receives directional reward r_D from reward function D	continues as usual	ship moved to next state according to the ship dynamics

Table 7. DQN's reward setting.

6.2 DQN's learning process:

The DQN's update rule is still fundamentally the same as Q-learning. But instead of a table, DQN uses deep NNs (larger than 2-layers) to approximate action values. The learning target is replaced by the Q-target $Q(s', a'; w_i)$. DQN can learn through updating NN's weight vectors w_i by minimizing loss function $L(w_i)$ iteratively (Y. Shen et al. 2017):

$$L(w_i) = \frac{1}{n}\sum_{1}^{n}[Q(s,a,w_i) - (r(s,a) + \gamma max_{a'}Q(s',a';w_i))]^2 \qquad (9)$$

Eqn. (9) defines the loss function for neural network. In this project, classic SGD is used as the lost function's optimizer. While the update continues, the change of weights for one episode might affect performances in other episodes. To overcome the issue, experience replay is added to the training of DQN.

Experience is stored in replay memory in order to train the agent with some less recent memory to break the temporal correlations. Specifically, experience in terms of four key elements (state, action, reward, next state) at time t is stored as $(\phi_t, a_t, r_t, \phi_{t+1})$ in experience history queue D_h. The symbol ϕ_t is equivalent to state s_t for the experience pool. Agent is trained with randomly sampled points from D_h.

In this project, prioritized experience replay is considered by adjusting the frequency of replay transitions to increase learning speed. Proposed by T. Schaul et al. (2015), this method calculates "importance" of transitions through some weighted metrics to minimise bias in distribution, in most cases greatly enhancing efficiency.

6.3 The proposed DQN structure:

In this project, a concise DQN structure is proposed with 3 fully connected layers (size 64x32x32) assisted by batch normalizations. A tensorboard visualisation of the detailed architecture is displayed in figure 24, "fc" referring to fully connected layer, "bn" referring to batch normalization layer, number of neurons can be seen in flow.

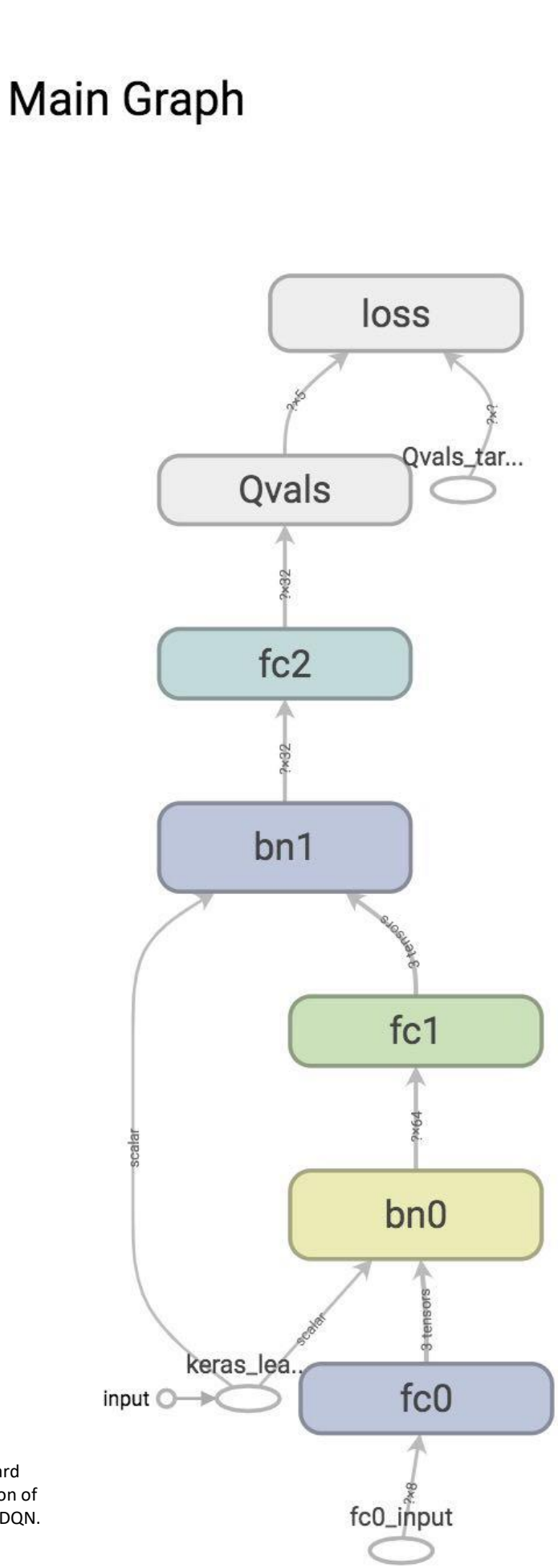

Figure 24. Tensorboard visualisation of proposed DQN.

6.4 Single-obstacle simulation result:

To first test the feasibility of proposed network, single obstacle case in figure 25 is considered. Loss (mean-square-error) and average reward (per episode) curves are plotted in figure 26 and 27. Tensorboard's smoothing is used to help better visualise trends.

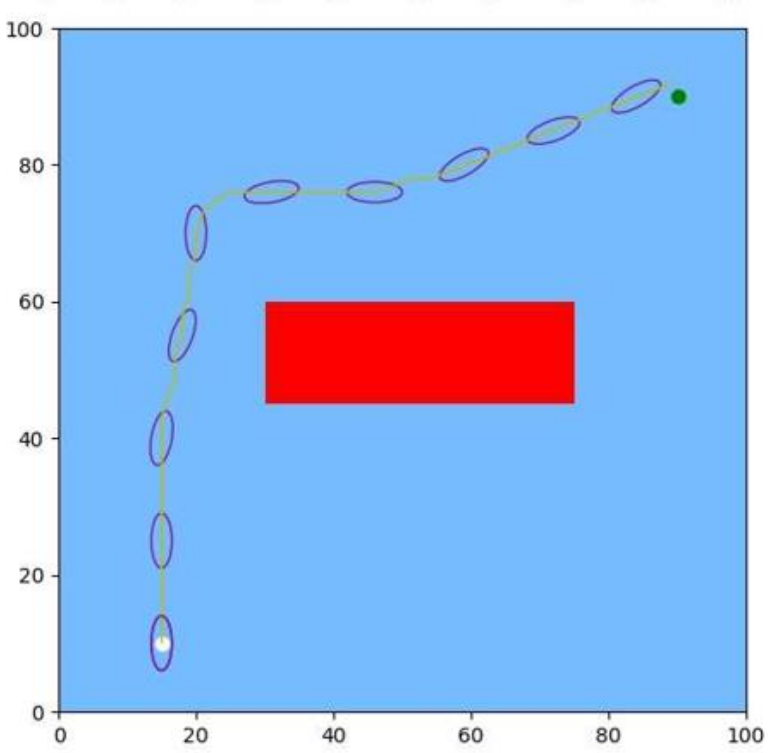

Figure 25. Single-obstacle map simulation.

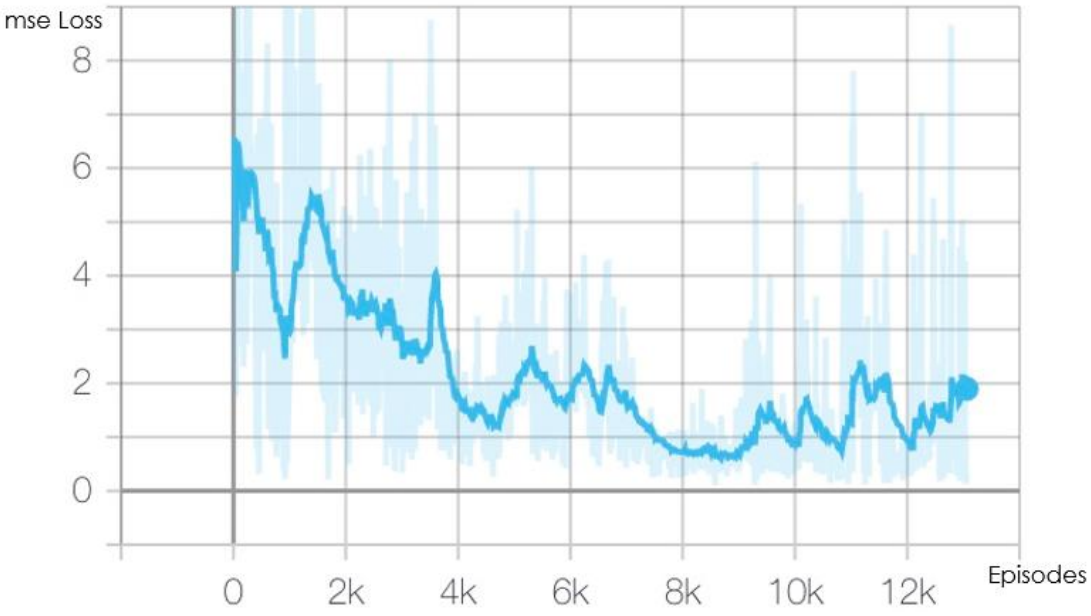

Figure 26. Loss curve for figure 25 simulation.

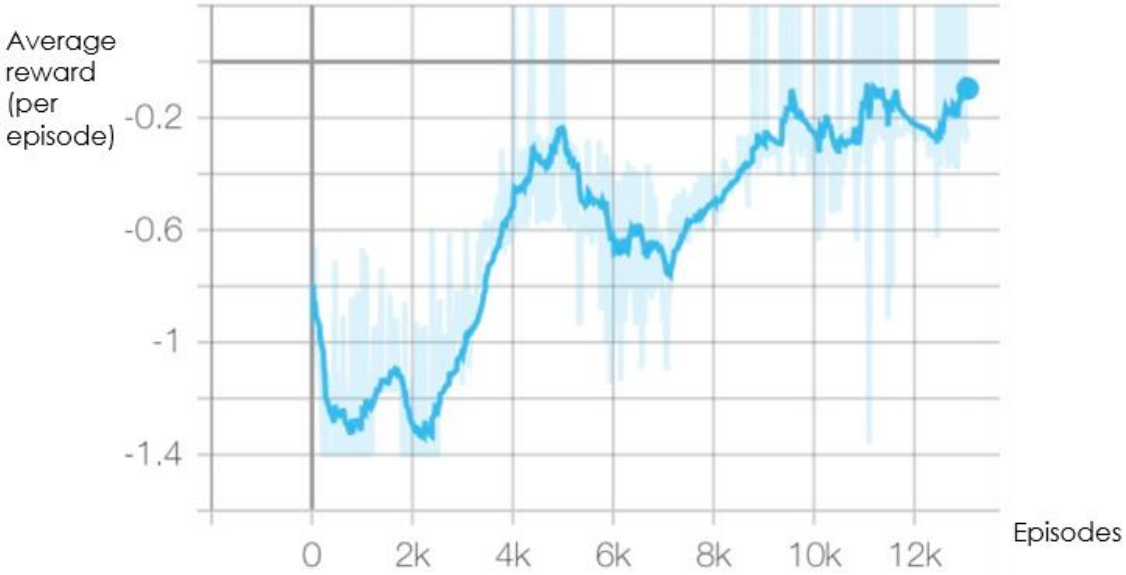

Figure 27. Reward curve for figure 25 simulation.

Figure 25's trajectory validates the proposed DQN's path planning ability. Figure 26 confirms the convergence of the result despite of heavy noise shown in the background. After smoothing the loss can be seen dropping to a steady level around 1.5. Figure 27 shows average reward's convergence, the sudden drop in between the reward curve is caused by agent's exploration of the right side of map. The general converging trend can be seen, with considerable noise due to probably the constant change in DQN's policy caused by epsilon.

6.5 Hyperparameter analysis:

The selections of number of neurons (NN size) of DQN and learning rate are analysed.

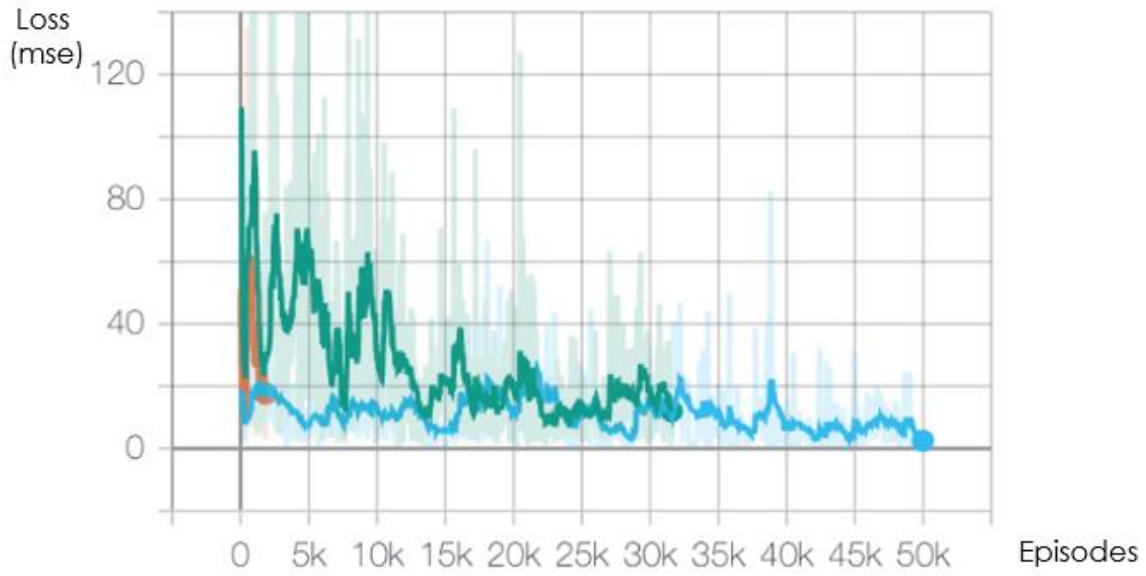

Figure 28. Loss curves for different neuron unit settings, green:32x16x16; blue:64x32x32.

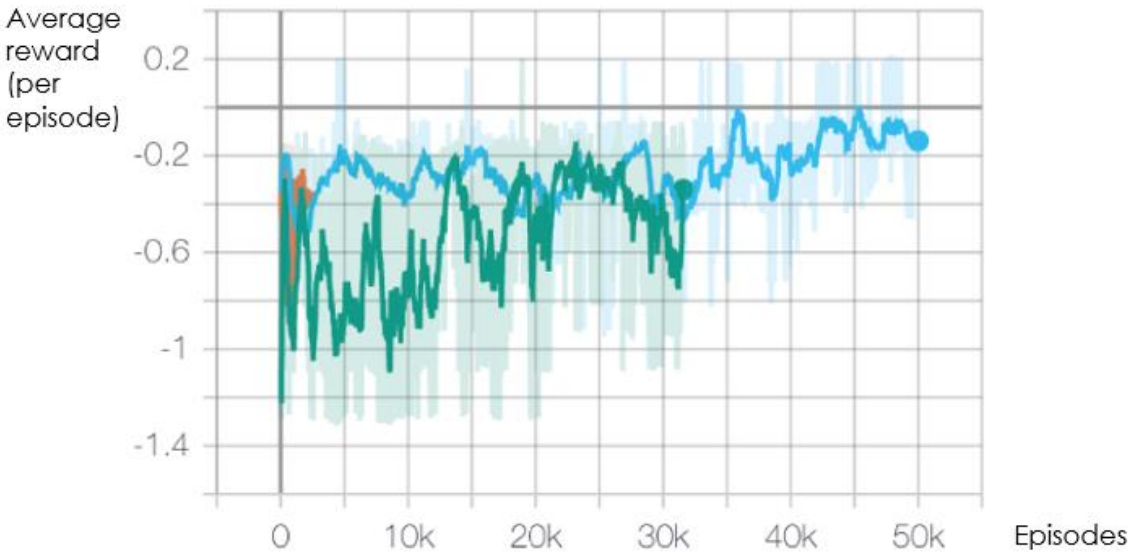

Figure 29. Reward curves for different units, green:32x16x16; blue:64x32x32.

Figure 28, 29 shows the larger neuron size (blue) has superior performance than smaller size (green), but it takes longer to converge.

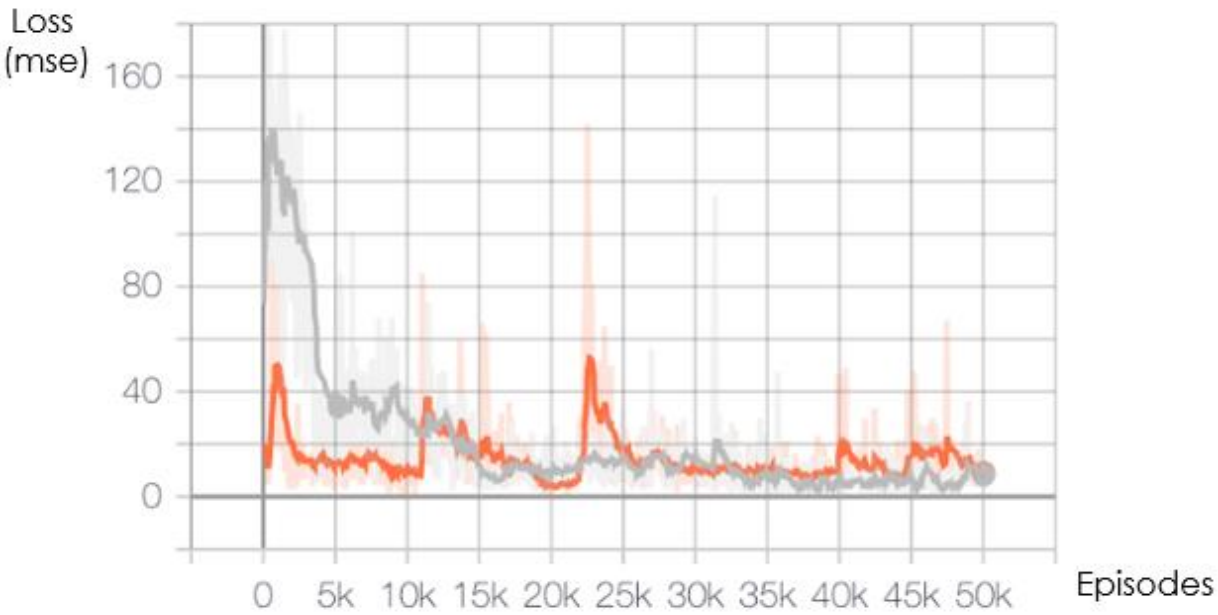

Figure 30. Loss curves for different learning rates, grey:4e-4; orange:1e-3.

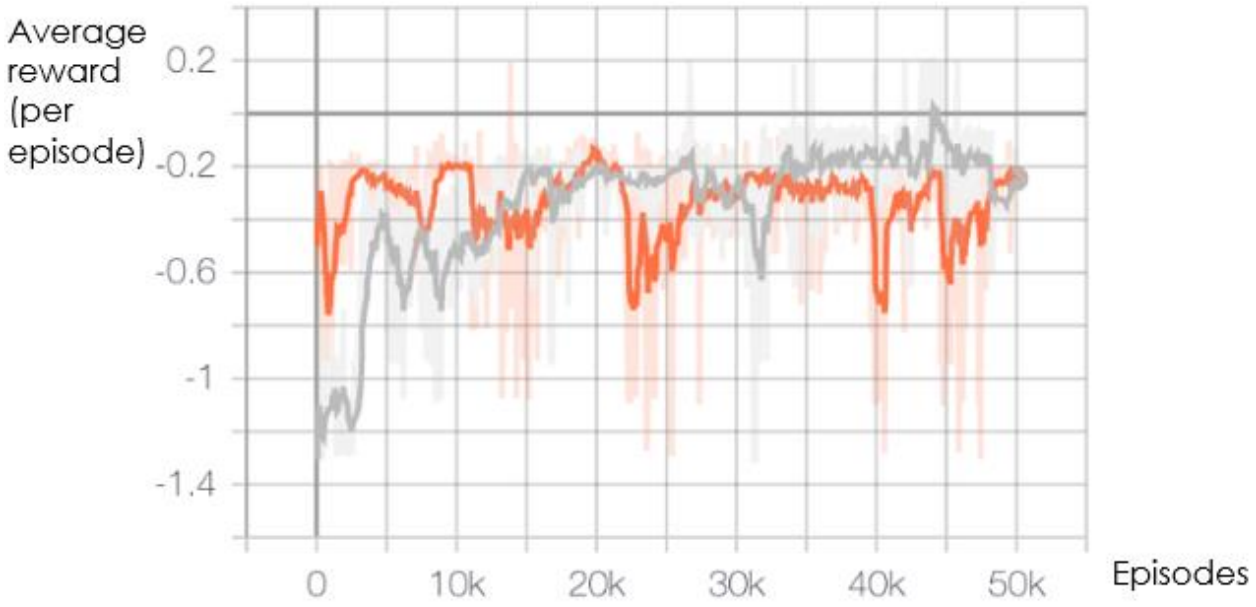

Figure 31. Reward curves for different learning rates, grey:4e-4; orange:1e-3.

Figure 30, 31 shows the higher learning rate (orange, 0.001) initially performs better, but fails to display convergence in the long term. Lower learning rate (grey, 0.0004) shows overall convergence, but requires adequate training.

Given above analysis, adequate NN size (64x32x32) and small learning rate (0.0004) are recommended for this project.

6.6 Large map simulations:

To test the DQN's real-world potential, a 200x200x36 map double the size before is simulated.

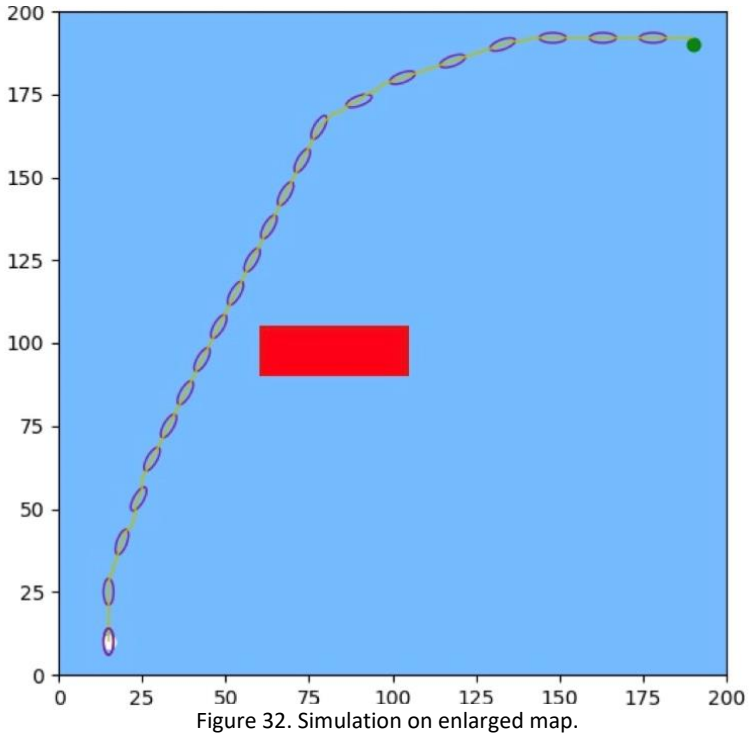

Figure 32. Simulation on enlarged map.

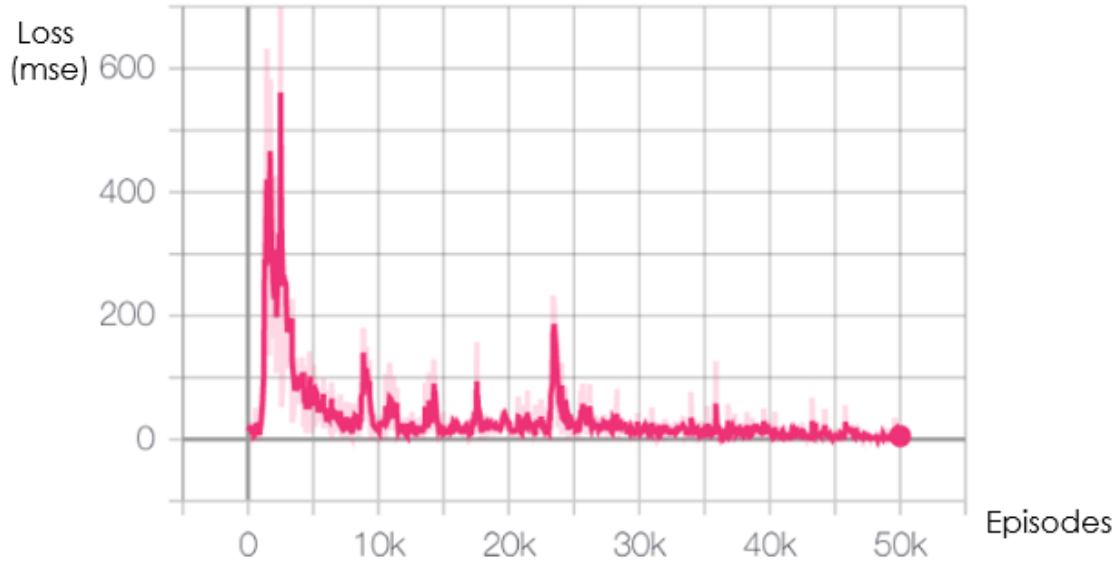

Figure 33. Loss curve for large map simulation.

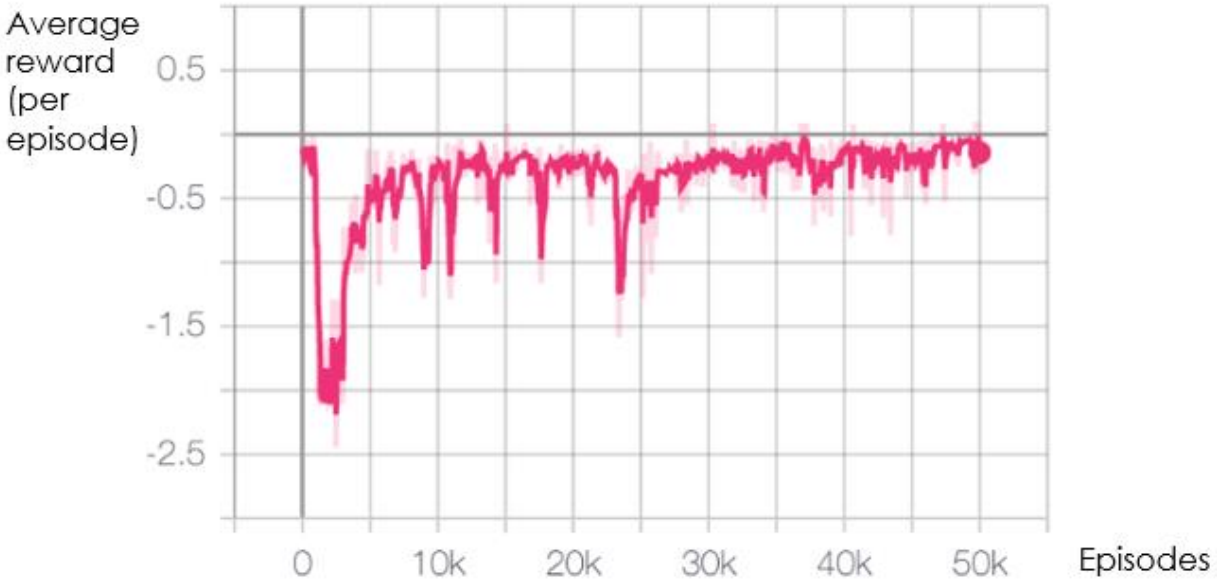

Figure 34. Reward curve for large map simulation.

Figure 32's path demonstrates positive performance in this enlarged map. Figure 33, 34 confirms convergence of the solution, proving the DQN's powerful generalization ability to maintain previous performance as state entries increase, while Q-learning would become much slower to memorize and query the Q-table.

7. Conclusion and future work

This project fills research gap of C. Chen et al. (2019)'s work by first building a Q-learning framework then proposing and studying DQN method based on it. The Q-learning exhibits good performance in small scale path planning, while the DQN also proves to be also capable, although more susceptible to reward noise. The proposed DQN exhibits eventual convergence, even as map increases, proving success of its implementation. The Q-learning's strength of "absolute" reward memorization can only benefit on miniature maps. DQN-based path planning would be much more efficient on large-scale maps, given adequate training. In general, the proposed approach with DQN proves more feasible for an onboard implementation and integration with other subsystems.

However, this project has been dealing with discrete spaces throughout, which is not ideal for steering of large ships with precise angle requirements. For future effort, advanced methods such as DDPG could be considered to solve this ship problem directly in continuous domain, without any discretization. Local path planning with simulated sensor inputs and dynamic obstacle avoidance could also be implemented, supplementing the existing framework.

8.Appendix A: Preliminary literature review on traditional methods

Artificial Potential Fields, known for their ease of implementation, can be thought as an interesting analogy to the theory of electric charges. It principally functions by assigning a virtual potential to target destination that can attract agent, while assigning a repelling potential to obstacles, thereby achieving global path planning with obstacle avoidance. According to Sfeir et al. (2011) who used AFP to guide mobile robots, the APF can be defined by equation (10):

$$\phi(x,y) = \phi_g(x,y) + \phi_o(x,y) \qquad (10)$$

This equation states that APF consist of two components: potentials of goal ϕ_g and potential of obstacles ϕ_o, and (x,y) represents the state of the agent is in. Specifically, the potential of goal ϕ_g can be analytically expressed, in equation (8):

$$\phi_g(x,y) = \frac{1}{2}\xi\left((x-x_g)^2 + (x-y_g)^2\right) \qquad (11)$$

The x_g, y_g represent the horizontal and vertical coordinates of the goal position. Now, equation (11) shares a very similar structure to the second order spring-mass system, suggesting the agent would behave just like a spring hung with a mass. This nature partly explains one of the APF's shortcomings, that is the susceptibility to oscillations. Arambula et al. (2004) also states that the primary disadvantage of APF is its tendency to be trapped by the local minima problem, due to the fact that the objective function is often multimodal, associating with many variables. Although D.H. Kim (2009) provides a solution to escape the local minima trap by introducing new sets of rules to guide the agent when stuck in situations where attraction and repulsion cancel out. In general, APF has good path-generating performance in simple settings, its shortcomings can be addressed at the cost of more computational complexity.

To improve performance in complex environments, sampling-based methods can be a potent choice. Rapid-exploring Random Tree (RRT) is one of the representatives. Lee et al. (2014) investigated the use of RRT in challenging environments, such as the problem of narrow passages. He stated that basic RRT randomly samples a point q_r through the space and finds its closest neighbour q_n according to random trees. The two points are then connected. If the line of connection is obstructed, a new point q_c that first comes into contact with the boundary of obstacle along the straight line of q_r-q_n is computed. Then q_c is added to the random tree and a new unobstructed connection between q_c and the original q_r is established, thus achieving obstacle avoidance. However, this basic RRT can take a long computation time, especially when obstacles are close to each other.

Algorithm 1 RRRT Planner

T.Init(q_{init})
repeat
  $q_r \leftarrow$ RandomState(); $q_n \leftarrow$ NearestNeighbor(q_r, T)
  if HaveCollisionFreePath ($\overline{q_n q_r}$) **then**
    $q_{new} \leftarrow$ RRTExtend(q_n, q_r)
    T.AddVertex(q_{new}); T.AddEdge(q_n, q_{new})
  else
    $q_c \leftarrow$ the first in-contact configuration from q_n to q_r
    OptimizedRetraction(q_c)
  end if
until a collision-free path between q_{init} and q_{goal} is found

Fig.A1. Pseudocodes of RRRT (Lee et al. 2014).

To improve this, the optimized Retraction Rapid-exploring Random Tree (RRRT) is proposed. RRRT works in the same way as RRT in generating random point q_r and its neighbour q_n and the first point q_c to touch obstacle. But if no collision occurs along the q_r-q_n path, it then samples a new point q_c' that minimizes the distance to q_r. Then it replaces the q_n with q_c' repeatedly until the distance of q_r-q_n has converged. The distance minimizing step in the loop portraits RRRT to have a greedy policy. But result shows that RRRT computes four time quicker than the basic version (Lee et al. 2014).

9.Appendix B: Preliminary literature review on heuristic methods

Heuristic method like A* algorithm demonstrates its strength in the path planning for an autonomous boat named Avalon (Erckens et al. 2010). The proposed path planning with A* algorithm utilizes the basic Dijkstra's graph search, where each grid point is searched and assigned a cost which is evaluated by a set of predefined functions. The optimal path is selected based on always following a neighbour with the lowest cost from starting location until finish. A* algorithm is heuristic, different from traditional methods mainly due to the addition of a heuristic function $h(n)$ that contributes to the cost calculation. This function can be designed to give the agent semi human-like logics and it can guarantee a solution. Li et al. (2017) has also successfully applied a modified A* algorithm to help solve seabed terrain matching navigation (STMN) for an Autonomous Underwater Vehicle (AUV). The A*algorithm is used to perform a grid search over the area of seabed terrain, using terrain-entropy and terrain-variance-entropy to determine the matching performance. Generally, A* algorithm performs well at a comprehensive grid point search in a given planar map where the obstacle conditions are simple, but it faces difficulty to adapt to a continuous control problem due to its discrete grid nature.

Fuzzy Logic algorithms (FL) use a set of if-then rules to give agent human-like logical thinking and human experience in path planning. Hong et al. (2012) illustrates the fundamental structure of FL. First the input is identified by the fuzzy set, before a process of fuzzification can be carried out. This process essentially converts input data into a set of fuzzy variables, according to the membership functions (fuzzy sets). Shape of those membership functions can be designed to be Gaussian, triangular or trapezoidal. A fuzzy rule-base is established consisting of sets of if-then logics. Inference is used to evaluate the transformed input data according to the rules, returning a combined result. The final step is defuzzification. It converts the fuzzy rule results back into normal data (crisp data) as the output. The entire flow can be visualised in figure A2.

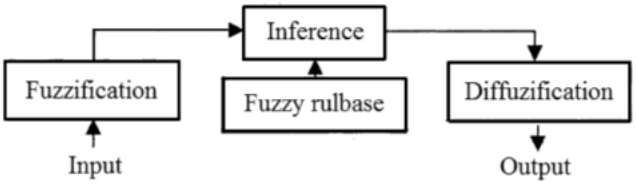

Fig.A2. Flow diagram of fuzzy logic control (Hong et al. 2012).

Fuzzy logic combined with neural networks takes advantage of both the human-like expertise and the non-linear mapping capability of neural networks. Joshi et al. (2011) proposed a neuro-fuzzy approach to path planning for a mobile robot. The proposed two-stage hybrid neuro-fuzzy system processes sensor information on environment via the neural network, while corrects decision making via the fuzzy logics. Figure A3 demonstrates the proposed combined architecture. Sensor data all form inputs to the neural network, simultaneously being cascaded to a fuzzy system which then outputs a control command based on fuzzy rules. The heading angle together with other state information are passed into the neural network which then outputs a Reference Heading Angle (RHA). As heading angle is an important control parameter in this case, the RHA supported by local sensor data produced by the neural network is then fed to the fuzzy system, improving the final inferred output. In general, the fuzzy-neuro method proves to be suitable for complex and unknown environments, exhibiting good performance in dealing with dynamic, even cluttered obstacle conditions. The combined method outperforms any of its individual constituent.

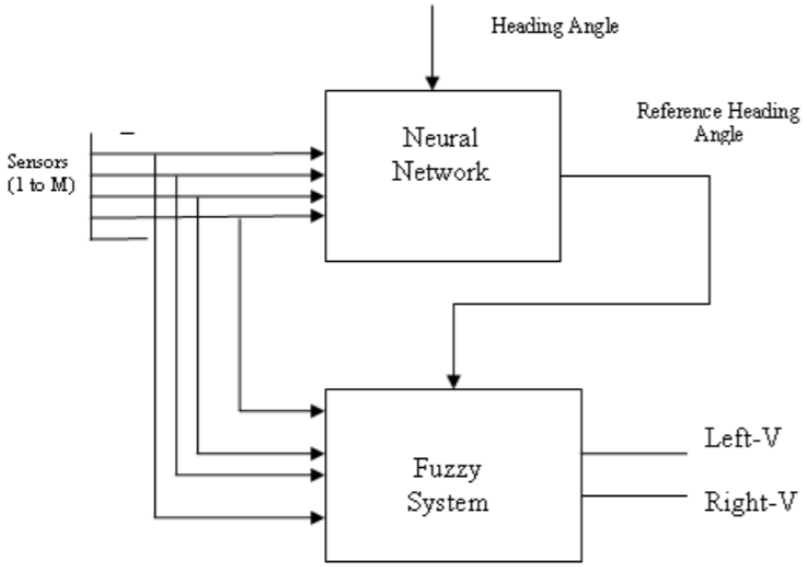

Fig.A3. Proposed two-stage neuro-fuzzy system for mobile robot (Joshi et al. 2011).

Biological inspiration can also become a driving force behind path planning, leading to the so-called naturally inspired algorithms. Ant Colony Optimization algorithm (ACO) is one of the more potent methods. X. Chen et al. (2013) investigated thoroughly the use ACO in path planning. According to the work, the ACO mimics the natural behaviour that when ant colony look for food, chemicals called "pheromone" are left to attract other ants. The intensity of pheromone is inversely proportional to the length of path. Conceptually, agent should compute a transitioning probability based on the intensity of pheromone it perceives at that location, also taking into account the pheromone evaporation rate which is a function of the inverse of path length. The beauty about this design is that shorter paths will be more frequently visited by the agent, causing pheromone intensity along that path to accumulate, until the agent has converged to the shortest path possible. However, with complexity of environment increases, ACO may take longer to compute and potentially get trapped in a local minima. Furthermore, if all paths are initialized to have the same pheromone intensity, ACO may fail to find a solution due to equal probability of transition at the beginning.

To remedy these failures, X. Chen et al. (2013) proposed a novel ACO that includes an additional map pre-processing stage. The proposed method first tags every node with pheromone (scent) information, recorded into a density matrix. Then agent moves according to the "1 minus search" strategy which forces the agent to move to a neighbour node with a lower intensity. The individual agent stops until the goal has been reached, updating the pheromone information on that path. The whole process repeats for other agents. The strategy allows search space to decrease dramatically compared to basic ACO, and it avoids the problem of inconsistency.

10.Appendix C: Derivation of mathematics for Q-learning

10.1 Markov Decision Process (MDP):

The underpinning framework for the Q-learning path planning is the MDP. Any Q-learning based problem must first be one that can be interpreted as an MDP problem. Defined by R. Bellman (1957), MDP introduced a way of modelling an agent's decision-making process in a discrete and stochastic environment. A set of variables of interest consists of:

- current state s
- current action a
- reward r
- next state s'
- transition probability p that defines the MDP dynamics from one state to the next state
- policy π the probability of agent choosing action a given state is in s

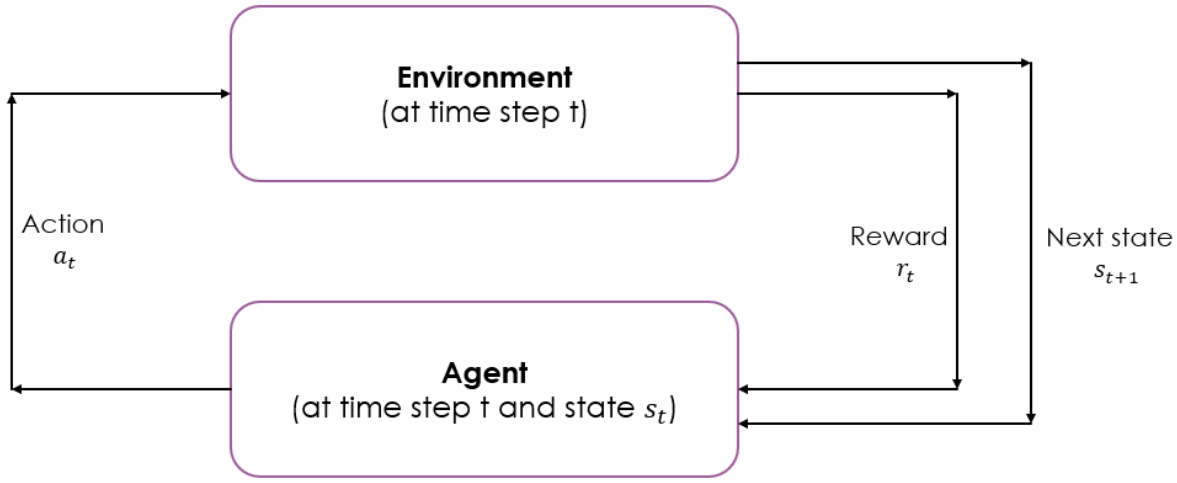

Fig.A4. The model of MDP dynamics, displaying the agent-environment interaction.

Figure A4 visualises the MDP model in which these variables interact with each other. At step t, the agent is state s_t. An action a_t chosen by the agent at s_t will receive its subsequent reward r_t from the environment and be moved into the next state s_{t+1} at the next step accordingly. The policy of the agent at time step t is defined by $\pi_t(a_t|s_t)$ which is the action conditional on the current state. The transition probability p describing the model dynamics can be mathematically defined as $p = p(s_{t+1}, r_t|\, s_t, a_t)$, which means the probability of agent receiving reward r_t and move into state s_{t+1} given the current state s_t and action a_t. The key concept here is that MDP assumes the transition only depends on the previous step tracing only one step backward. Note this memoryless nature is also named the Markov property in stochastic processes.

For an episodic task (task that has a clear terminal state), the gain G is just the sum of rewards for one episode. However for continuing tasks, there would be infinite sum of rewards (thus infinite gain):

$$G_t = R_{t+1} + R_{t+2} + R_{t+3} + \cdots \quad (12)$$

To get around this, a discount factor γ can be multiplied to each reward term:

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots, \ and\ 0 \leq \gamma < 1 \quad (13)$$

This makes the gain G finite as now the maximum reward is constant, which can be proved as:

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \leq \sum_{k=0}^{\infty} \gamma^k R_{max} \quad (14)$$

Where R_{max} is assumed to be the maximum possible reward which is also a constant that can be moved out of the summation, and the inequality in eqn. (14) above becomes:

$$G_t \leq R_{max} \sum_{k=0}^{\infty} \gamma^k \quad (15)$$

by the sum to infinity of γ:

$$\sum_{k=0}^{\infty} \gamma^k = \frac{1}{1-\gamma} \; as \; 0 \leq \gamma < 1 \quad (16)$$

Now suffice to say that the gain can be confined with:

$$G_t \leq R_{max} \frac{1}{1-\gamma} \quad (17)$$

The term on the right of the eqn. (17) comprises of all constants. Therefore, gain G_t is proved to be finite with the addition of discount factor γ.

And express return recursively:

$$G_t = R_{t+1} + \gamma G_{t+1} \quad (18)$$

The recursive expression of eqn. (18) greatly facilitate computation as now computer only has to store two data points and update them at every step. Finally, the goal of any agent in MDP is to maximise the cumulative reward (gain) in the long term.

10.2 Policy and value functions for MDP:

The policy $\pi(a|s)$ for an agent is the probability that action a is chosen at the given state s. And it can be seen that policy only depends on the current state s. In short, policy determines how action will be selected by the agent during the interaction.

Value functions can be divided into state value functions and action value functions. State value function is basically the expected value for the gain starting in state s and following policy π:

$$V_s = E[G_t|S_t = s], with\ respect\ to\ policy\ \pi \quad (19)$$

And state value function Vs will be 0 at terminal state, as the gain at that state is 0.

Similarly, the action value function is defined as the expected value for the return (gain) of taking action a in state s and following the agent's policy:

$$q(s,a) = E[G_t|S_t = s, A_t = a], with\ respect\ to\ policy\ \pi \quad (20)$$

The state and action value functions are created for the evaluation of the agent's policy.

10.3 Bellman equations and Bellman optimality equations:

Bellman equations provide a way to evaluate the state value and action value at a particular state. State value Bellman equation is defined as:

$$V_s = \sum_a \pi(a \Big| s) \cdot \sum_{s'} \sum_r p(s', r|s, a) \cdot [r + \gamma V_{s'}] \quad (21)$$

This is derived from the definition of state value function combining with the recursive return Gt:

$$G_t = R_{t+1} + \gamma G_{t+1} \quad (18)$$

The state value Bellman equation literally is implying that the expected return is the sum of all possible values weighted by the probability that they occur. And it can be used to represent an average of infinite number of future steps.

The action value Bellman equation is similarly defined as:

$$q_{(s,a)} = \sum_{s'} \sum_{r} p(s', r|s, a) \cdot \left[r + \gamma \cdot \sum_{a'} \pi(a'|s') \cdot q_{(s',a')} \right] \quad (22)$$

This is suggesting that the expected action value is a weighted average of all possible action values with the probability that they would occur. Note that the previous expressions for both Bellman state and action value functions are written in terms of the next step's state and value functions.

The optimality theorem states that there should exist an optimal policy and optimal value function so that the optimal state value function with respect to the optimal policy is greater than or at least equal to all other state value functions across the entire state space, and consequently leading to an optimal action value function as well. Therefore that optimal policy π^* shares an optimal state value function V^* and an optimal action value function q^*.

Accordingly, Bellman optimality equations can be derived from the original Bellman equations. The Bellman state value optimality equation can be rewritten as:

$$V_s^* = max_a \sum_{s'} \sum_{r} p(s', r|s, a) \cdot [r + \gamma V_{s'}^*] \quad (23)$$

The max(a) is implying an optimal deterministic policy that assigns probability 1 to an action a that achieves the highest action value, and 0 to all other actions. Therefore it replaces the summation sign earlier, only requires to compute for the maximum action term.

Similarly, Bellman action value optimality equation can be rewritten as:

$$q_{(s,a)}^* = \sum_{s'} \sum_{r} p(s', r|s, a) \cdot \left[r + \gamma \cdot max_{a'} \cdot q_{(s',a')}^* \right] \quad (24)$$

And from the action value optimality equation one can find the optimal policy directly by taking the argmax of the optimal action value function. However, the exact evaluation of any particular Bellman equation would require the knowledge of the MDP dynamics $p(s', r|s, a)$. This environment dynamics is not always so straightforward to derive, especially when dealing with more complex problems. This dilemma prompts the use of methods that do not rely so heavily on the model of environment, but instead can learn based on observation and experience.

10.4 Q-learning from Temporal Difference (TD) learning:

Temporal Difference (TD) learning uses bootstrapping to learn from experience much more efficiently instead of the complete search of Monte Carlo (MC). The concept of TD again builds on the important idea of "recursion". In general, a value from an action can be expressed as the average of all the rewards received by that action. And by re-expressing the value at one step in terms of the value of the precious step, the value function can be written as:

$$V_{t+1} = V_t + \alpha \cdot [G_t - V_t], \quad where\ \alpha\ is\ the\ constant\ stepsize\ parameter \quad (25)$$

By merging the recursive gain of eqn. (18) into the above equation, one can obtain the temporal difference error:

$$V_{t+1} = V_t + \alpha \cdot [R_{t+1} + \gamma V_{t+1} - V_t] \quad (26)$$

The reward term is the target, and the entire difference term within the bracket is the temporal difference.

From the above state value TD equation, the one-step temporal difference algorithm (TD0) for policy evaluation can be constructed, by looping over all steps of an episode and constantly updating the state values.

However, to solve for the control problem, the action value evaluation and update is required. Q learning takes care of this by selecting actions to maximise the action value Q as its target policy in the temporal difference calculation (Sutton and Barto, 2018), shown before as eqn. (4):

$$Q_{(s,a)} = Q_{(s,a)} + \alpha \cdot \left[r + \gamma \cdot max_a Q'_{(s',a)} - Q_{(s,a)}\right] \quad (4)$$

This update rule is exploited at each run of the algorithm to enable the ship agent to learn and update its experience, just like a human.

11.Appendix D: From general equations of ship motion to Nomoto model

Figure A5 illustrated a 6 degrees of freedom (6DOF) 3-dimensional (3D) system for modelling the general ship motion. For a rigid body ship, the dynamics should be described as motion along the three axes x, y, z corresponding to the surge u, sway v and heave w; and rotation about the three axes x, y, z corresponding to the roll ϕ, pitch γ and yaw ψ.

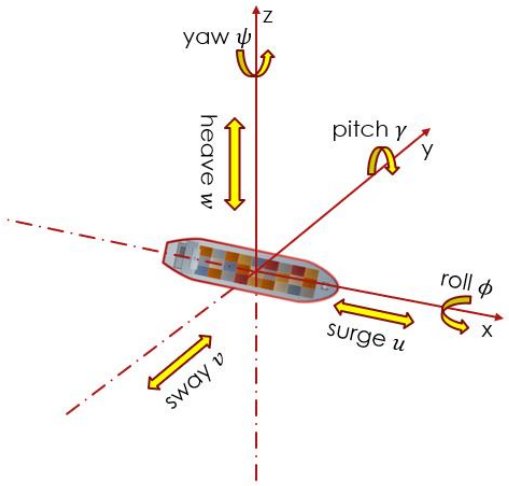

Fig.A5. Ship-fixed frame for general ship motion.

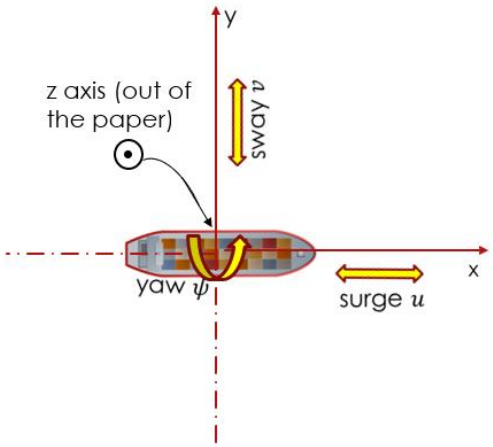

Fig.A6. Ship-fixed frame for 3DOF ship motion.

A common simplification to be made for modelling surface vessel is the reduction of DOFs from 6 to 3 by neglecting the heave w, roll ϕ, pitch γ. Both C. Chen et al. (2019) and Y. Cheng et al. (2018) adopted this assumption in their implementation of Q-learning based path planning and this assumption is also supported by Swarup et al. (2016) in the study of evolution of ship mathematical modelling. Therefore, in this project, the 6DOF system shown in figure A5 is confidently simplified to a problem that is 2D on the $x - y$ plane in nature. Common assumptions for 2D ship model are made in this project:

- The ship is symmetric around the $x - z$ plane.
- The fixed origin of the ship body has the x,y coordinates of the Center of Gravity (CGx and CGy).
- The ship mass is homogeneously distributed.

The simplified 3DOF model viewed on a $x - y$ plane is shown in figure A6, with a ship-fixed frame. Now, this 2D model can be mathematically represented according to Newton laws of motion, in a global cartesian frame:

$$\begin{cases} m\dfrac{d^2x_0}{dt^2} = F_{x0} \\ m\dfrac{d^2y_0}{dt^2} = F_{y0} \\ I_{zz}\dfrac{d^2\psi}{dt^2} = N \end{cases} \qquad (27)$$

where the suffix "0" represents the global cartesian reference frame, x_0 means the x-coordinate of the ship, y_0 is the y coordinate, F_{x0} means the net horizontal force exerting on the ship, F_{y0} is the net vertical force, I_{zz} is the moment of inertia about the z axis, ψ is the yaw, N is the total moment on the ship caused by F_{x0} and F_{y0} about the z axis. Equation (27) is constructed under the cartesian coordinate system, which is not so convenient as the previous ship analysis is based on the ship body frame. To unify the reference frame, a transformation from cartesian to ship-fixed frame yields:

$$\begin{cases} m(\dot{u} - vr) = F_x \\ m(\dot{v} + ur) = F_y \\ I_{zz}\dot{r} = N \end{cases} \qquad (28)$$

where $r = \dot{\psi}$ is the yaw rate, " $\cdot$ " represents the time derivative of the variable, F_x, F_y are net forces under ship-fixed frame, and N is the net moment about the z axis.

However, the general 2D model still contains a large number of "vague" terms that are not so easy to be specified. To mathematically solve the dynamics equation set in eqn. (28), an expression for the complex hydrodynamic forces is required. According to Swarup et al. (2016) the relationship between net forces and the ship dynamics variables can be approximated via a first-order Taylor expansion, first of all focus on N, which can be defined as the comprehensive function:

$$N = f_N(u, \dot{u}, v, \dot{v}, r, \dot{r}, \delta, \dot{\delta} \dots \dots) \qquad (29)$$

where δ is the rudder angle, using the Taylor expansion and removing higher order terms:

$$dN = \frac{\partial f_N}{\partial u} \cdot u + \frac{\partial f_N}{\partial \dot{u}} \cdot \dot{u} + \frac{\partial f_N}{\partial v} \cdot v + \frac{\partial f_N}{\partial \dot{v}} \cdot \dot{v} + \frac{\partial f_N}{\partial r} \cdot r + \frac{\partial f_N}{\partial \dot{r}} \cdot \dot{r} + \frac{\partial f_N}{\partial \delta} \cdot \delta + \frac{\partial f_N}{\partial \dot{\delta}} \cdot \dot{\delta} \qquad (30)$$

Now substituting (28) into the $I_{zz}\dot{r} = N$ equation in (26) yields:

$$dI_{zz}\dot{r} = \frac{\partial f_N}{\partial u} \cdot u + \frac{\partial f_N}{\partial \dot{u}} \cdot \dot{u} + \frac{\partial f_N}{\partial v} \cdot v + \frac{\partial f_N}{\partial \dot{v}} \cdot \dot{v} + \frac{\partial f_N}{\partial r} \cdot r + \frac{\partial f_N}{\partial \dot{r}} \cdot \dot{r} + \frac{\partial f_N}{\partial \delta} \cdot \delta + \frac{\partial f_N}{\partial \dot{\delta}} \cdot \dot{\delta} \qquad (31)$$

Equation (31) can be thought as a parametric form of expression for the ship dynamics. The determination of coefficients $\frac{\partial f_N}{\partial u}$, $\frac{\partial f_N}{\partial \dot{u}}$, $\frac{\partial f_N}{\partial v}$... etc. may require prediction from scale-down trials when the full-scale test is not possible, but inaccuracies may be introduced.

If the ship symmetry and constant surge assumption applies to equation (31), then the Davidson and Schiff Model (Davidson el at. 1946) can be derived as a simplification of eqn. (31):

$$I_{zz}\dot{r} = N_{\dot{\delta}} \cdot \dot{\delta} + N_v \cdot v + N_{\dot{v}} \cdot \dot{v} + N_r \cdot r + N_{\dot{r}} \cdot \dot{r} \qquad (32)$$

Where the capital N represents the hydrodynamic derivatives. The coefficients of the parametrised models in eqn. (32) and eqn. (31) are difficult to determine, thus this project intends to use simpler models.

Now, since this project is concerned with cargo ships that are typically slow and difficult to turn, it is reasonable that a set of additional assumptions about the cargo ship dynamics can be made:

- Constant average forward speed (surge).
- The sway speed v is small.
- The yaw rate r is small.
- No coupling from roll caused by the yaw rate.
- The rudder angle δ is small.

With the establishment of these assumptions, the Nomoto model (Nomoto et al., 1957) can be obtained from simplifying equation (32):

$$\dot{\psi} + \ddot{\psi}(\tau_1 + \tau_2) + \dddot{\psi} \cdot \tau_1\tau_2 = K(\delta + \tau_3\dot{\delta}) \qquad (33)$$

Conducting a Laplace transform on eqn. (33), assuming zero initial conditions:

$$\dot{\psi}_{(s)} + s\dot{\psi}_{(s)}(\tau_1 + \tau_2) + s^2\dot{\psi}_{(s)}\tau_1\tau_2 = K\left(\delta_{(s)} + s\delta_{(s)}\tau_3\right) \quad (34)$$

Rearranging (34), the transfer function can be found:

$$\frac{\dot{\psi}_{(s)}}{\delta_{(s)}} = \frac{K(\tau_3 s + 1)}{(\tau_2 s + 1)(\tau_1 s + 1)} \quad (35)$$

Equation (35) represents the second order Nomoto model. An interpretation of this system is provided in the work by Pradeep et al. (2015), the zero term $(\tau_3 s + 1)$ and the pole term $(\tau_1 s + 1)$ are due to the coupling from the sway with the yaw rate, and these two terms can nearly cancel each other out, thus eqn. (35) can be rewritten as after the pole-zero cancellation:

$$\frac{\dot{\psi}_{(s)}}{\delta_{(s)}} = \frac{K}{1 + Ts} \quad where\ T = \tau_1 \quad (36)$$

Equation (36) defines the first order Nomoto model, whose primary characteristics are represented by the two parameters K the turning ability coefficient, T the turning lag coefficient. Now converting back to the time domain to investigate the relationship between the yaw rate $r_{(t)}$ or $\dot{\psi}_{(t)}$ and the rudder angle $\delta_{(t)}$:

$$T\ddot{\psi}_{(t)} + \dot{\psi}_{(t)} = K\delta_{(t)} \quad (37)$$

And in the time domain, solve the differential equation in (37) by integrating the yaw rate $\dot{\psi}_{(t)}$ w.r.t. time twice, obtaining the major time domain input-output relationship defined previously in eqn.(1):

$$\psi = K\delta\left(t - T + T\,e^{\left(-\frac{t}{T}\right)}\right) \quad (1)$$

The significance of the obtained equation (1) lies in the fact that it directly relates the ship course angle ψ to the rudder angle δ which can be directly controllable. As Shen et al. (2005) has suggested, the input-output nature of first order Nomoto KT model differs from the state-space nature of the Davidson and Schiff Model or other more direct models. The difference is subtle in that the state-space model directly describes the dynamics variables of the ship motion as well as the disturbances caused by waves or wind, while the input-output Nomoto model combines the rest of the terms and maps the input which is rudder angle δ to the yaw rate r and eventually to the output which is the ship course angle ψ, as can be seen in figure A7. And the non-linear is still preserved by the differential equation in the much simpler Nomoto KT model in eqn. (37). The disturbance factors are summarised by the representation of an effective turning response lag. This lag is reflected in the model in terms of a disturbance rudder angle imposed by the addition of the K, T parameters. Note that the numerical values of K and T are ship-specific, meaning they could only be determined via experiments on the ship. Overall, first order Nomoto model provides a very simplified relation yet retains the realistic ship dynamics consideration, especially for large ships. But before it is adopted as the ship motion model to be used in the path planning simulation in this project, its control behaviour needs to be investigated.

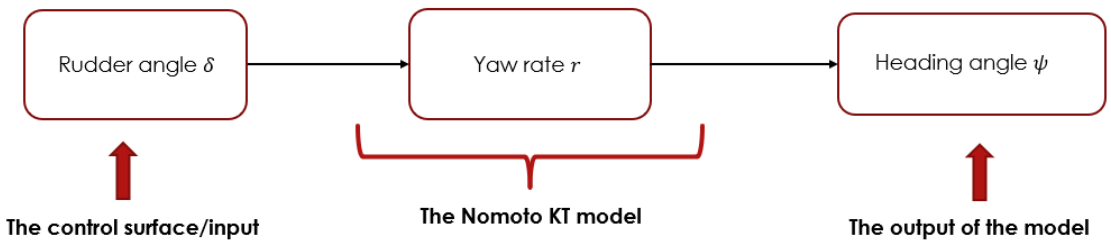

Fig.A7. The input-output response of Nomoto model.

12.Appendix E: System behaviour of the Nomoto model to control inputs

As this project's aim is to develop a path planning with control considerations, it is important to investigate the system's response behaviour to some typical control inputs. And for first order Nomoto model defined by the transfer function in eqn. (36), the only control input is the rudder angle, and the system response is the ship heading angle change. First, a general response analysis is carried out. Using the parameter values K=0.08, T=10.8 from realistic ship data used by C. Chen et al. (2020) the first order system's standard step and impulse responses can be visualised in matlab, in figure A8.

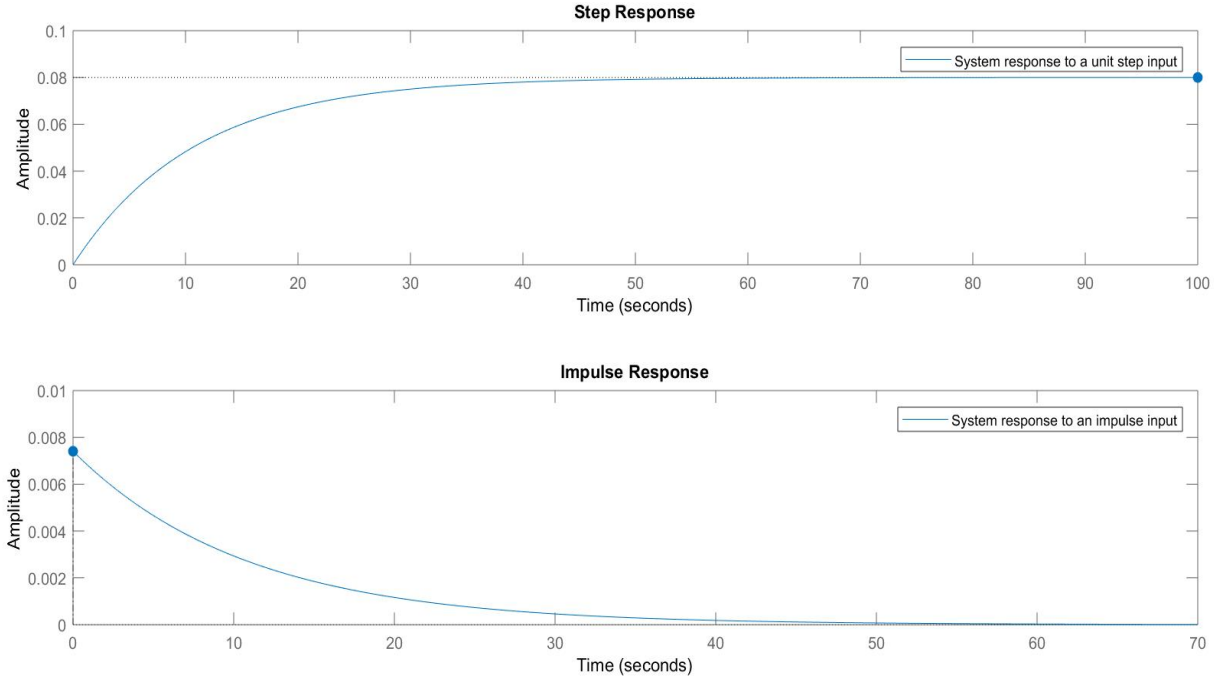

Fig.A8. The standard step and impulse response of the Nomoto system, generated by matlab.

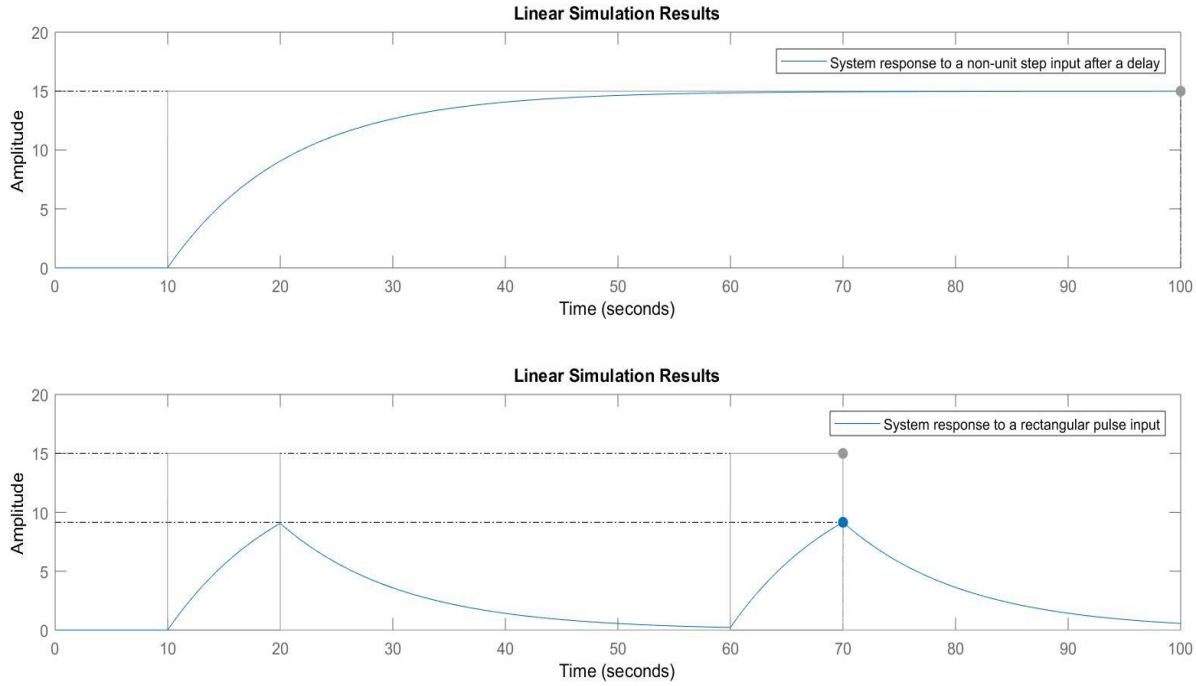

Fig.A9. The system response to a delayed step input and a rectangular pulse, simulated using matlab.

The unit step response shown in figure A8 proves the first order nature of the Nomoto KT model, with the peak response shown by the blue dot equals to the gain, and a time constant of about 10 seconds to reach the steady state with no overshoot. And the impulse response of the first order system shows the decaying trend.

Next, the more exact ship heading angle response to an actual rudder angle input signal in the time domain needs to be investigated. Particularly, the effect that the turning lag coefficient T produced on the first order system's $\psi_{(t)}$ response to $\delta_{(t)}$ is analysed, to verify the model's ability to simulate the lag in steering control that heavily affects cargo ships. To more realistically simulate the potential input signal that the real ship control might release, matlab's linear simulation function lsim is used, as it can take in both the system's transfer function and a simulated arbitrary signal. Two types of arbitrary signals are tested here: one is the step input taking a rudder angle value of $15°$ after a delay of 10 seconds; the other one is a rectangular pulse signal that takes the same rudder angle value $15°$ but only occurs in the time $10s < t < 20s$ and $10s < t < 20s$ and each lasting for only 10 seconds which is far less than the rise time of the system. For both signals the system response is allowed to be simulated for 100 seconds. The results are plotted in figure A9. As can be seen, for the step input, the system is able to follow the 10 seconds delay and reach the steady state at the maximum response. However, for the rectangular pulse input, the input signal lasts for too short to allow the ship to reach steady state, and response abruptly falls into a rapid decay afterwards, in the same way as it would be for an impulse.

Given the performance shown in figure A9, the first order Nomoto system demonstrates very simple and naïve behaviours to various signals, but it does factor in the turning difficulty of the cargo ship, by modelling the agent's inability to reach the maximum response and match the input rudder value if the input signal becomes realistic with a relatively short duration (realistic as in real ship operations the input cannot last infinitely). Overall, the Nomoto model is ultimately selected for use in this project due to its simplicity with an acceptable degree of realism in simulating cargo ship dynamics.

13.Appendix F: Matlab codes for simulations of KT Nomoto's system behaviour

```
s=tf('s');
K=0.08;   %define the turning ability coefficient
T=10.8;   %define the turning lag coefficnet
G_nomoto=K/(T*s+1);  %define the nomoto first order transfer
function
figure(1)
subplot(2,1,1)
step(G_nomoto)
legend('System response to a unit step input')
subplot(2,1,2)
impulse(G_nomoto)
legend('System response to an impulse input')
t=0:0.01:100;  %define the simulation time
step_sig=[];
for i=[1:length(t)]
    if i>=1000
        step_sig(i)=15;  %define the step signal as input
    else
        step_sig(i)=0;
    end
end
figure(2)
subplot(2,1,1)
% figure(1)
lsim((1/0.08)*G_nomoto,step_sig,t)   %simulate the system response
with lsim
ylim([0,20])
legend('System response to a non-unit step input after a delay')
% y=(1-exp(-(t/T)))*K*(5/0.08);
% plot(t,y)
rect_imp_sig=[];
for i=[1:length(t)]
    rect_imp_sig(i)=0;  %define the rectangular pulse signal as
input
end
for i=[6000:7000]
    rect_imp_sig(i)=15;  %add the rectangular pulse signal as input
end
for i=[1000:2000]
    rect_imp_sig(i)=15;  %add the rectangular pulse signal as input
end
subplot(2,1,2)
lsim((1/0.08)*G_nomoto,rect_imp_sig,t)   %simulate the system
response with lsim
ylim([0,20])
legend('System response to a rectangular pulse input')
```

14.Appendix G: Python codes for the Q-learning and DQN implementations

To clarify, due to formatting issues with the document, all python codes developed for this project are stored in the author's personal Github repository, link is below:

https://github.com/realdingke/undergraduate-projects-showcase

Prerequisites: for running any of the Q-learning file, python's basic scientific computing packages are enough.

- All codes for Q-learning implementation can be found in this link: https://github.com/realdingke/undergraduate-projects-showcase/tree/master/third%20year%20individual%20project
- For the Q-learning environment and agent setup, go to this file: https://github.com/realdingke/undergraduate-projects-showcase/blob/master/third%20year%20individual%20project/env_and_qagent_for_Q_learning.py
- For the Q-learning experiments, go to this file: https://github.com/realdingke/undergraduate-projects-showcase/blob/master/third%20year%20individual%20project/testplotpy.py

Prerequisites: for running of the DQN flow, tensorflow2.0 or above is **required.**

- All codes for the DQN implementation can be found here: https://github.com/realdingke/undergraduate-projects-showcase/tree/master/third%20year%20individual%20project/DQN
- The environment setup for DQN (contains a random map builder for training as well) can be found here: https://github.com/realdingke/undergraduate-projects-showcase/blob/master/third%20year%20individual%20project/DQN/codes/dqn_env.py
- The model of DQN itself can be found here: https://github.com/realdingke/undergraduate-projects-showcase/blob/master/third%20year%20individual%20project/DQN/codes/dqn_model.py
- For training of the model and experiments, go to here: https://github.com/realdingke/undergraduate-projects-showcase/blob/master/third%20year%20individual%20project/DQN/codes/dqn_train_experiments.py
  (Note: for complete training of the DQN, advanced computer hardware with high performance may be required.)

References

[Pascoal et al., 2000]: Pascoal, A., P. Oliveira, C. Silvestre, L. Sebastiao, M. Rufino, V. Barroso, J. Gomes, G. Ayela, P. Coince, M. Cardew, A. Ryan, H. Braithwaite, N. Cardew, J. Trepte, N. Seube, J. Champeau, P. Dhaussy, V. Sauce, R. Moitie, R. Santos, F. Cardigos, M. Brussienx and P. Dando (2000). Robotic Ocean Vehicles for Marine Science Applications: the European ASIMOV Project. 1. pp 409–415.

[Majohr et al., 2000]: Majohr, J., T. Bush and C. Korte (2000). Navigation and Automatic Control of the Measuring Dolphin (MESSINTM). pp 405–410.

[Naeem et al., 2006]: Naeem, W., Sutton, R., & Chudley, J. (2006). Modelling and control of an unmanned surface vehicle for environmental monitoring.

[Levander, 2017]: Levander, O. (2017). Autonomous ships on the high seas. IEEE Spectrum. 54(2), pp 26-31.

[Huang et al., 2020]: Huang, Y., Chen, L., Chen, P., Negenborn, R.R. and van Gelder, P. (2020) Ship collision avoidance methods: State-of-the-art. Safety Science [online], 121, pp 451–473. Available at: http://www.sciencedirect.com/ [Accessed 10 Mar 2020].

[Buniyamin et al., 2011]: Buniyamin, N., Ngah, W., Sariff, N. & Mohamad, Z.. (2011). A simple local path planning algorithm for autonomous mobile robots. International Journal of Systems Applications, Engineering & Development. 5, pp 151-159.

[Thi et al., 2016]: Thi, T. M., Cosmin C., Duc T. T., Robin D. K. (2016). Heuristic approaches in robot path planning: A survey, Robotics and Autonomous Systems. 86. pp 13-28.

[Sfeir et al., 2011]: Sfeir, J., Saad, M. & Saliah-Hassane, H. (2011) An improved Artificial Potential Field approach to real-time mobile robot path planning in an unknown environment. 2011 IEEE International Symposium on Robotic and Sensors Environments (ROSE). pp 208-213.

[Arambula et al., 2004]: Arambula C.F. & Padilla Castañeda, M.A. (2004) Autonomous robot navigation using adaptive potential fields, Mathematical and Computer Modelling 40(9-10), pp 1141-1156.

[Kim, 2009]: Kim, D.H. (2009) Escaping route method for a trap situation in local path planning Int. J. Control Autom. Syst. pp. 495-500.

[Erckens et al., 2010]: Erckens, H., Beusser, G., Pradalier, C. and Siegwart, R. Y. (2010) Avalon. IEEE Robotics & Automation Magazine. 17(1), pp 45-54.

[Li et al., 2017]: Li, Y., Ma, T., Chen, P., Jiang, Y., Wang, R. and Zhang, Q. (2017) Autonomous underwater vehicle optimal path planning method for seabed terrain matching navigation. Ocean Engineering. 133, pp 107-115.

[Lee et al., 2014]: Lee, J., Kwon, O., Zhang, L. and Yoon, S. (2014) A Selective Retraction-Based RRT Planner for Various Environments. IEEE Transactions on Robotics. 30(4), pp. 1002-1011.

[Hong et al., 2012]: Hong, T.S., Nakhaeinia, D., Karasfi, B. (2012) Application of fuzzy logic in mobile robot navigation Fuzzy Logic - Controls, Concepts, Theories and Applications. pp. 21-36.

[Joshi et al., 2011]: Joshi, M.M., Zaveri, M.A. (2011) Reactive navigation of autonomous mobile robot using neuron-fuzzy System Int. J. Robot. Autom. 2(3), pp. 128-145.

[X. Chen et al., 2013]: Chen, X., Kong, Y., Fang, X. et al. (2013). A fast two-stage ACO algorithm for robotic path planning. Neural Comput & Applic. 22, pp 313–319.

[Silver et al., 2016]: Silver, D., Huang, A., Maddison, C.J., Guez, A., Sifre, L., van den Driessche, G., Schrittwieser, J. (2016) Mastering the game of Go with deep neural networks and tree search. Nature. 529, pp 484-489.

[C. Chen et al., 2019]: Chen, C., Chen, X., Ma, F., Zeng, X. and Wang, J. (2019) A knowledge-free path planning approach for smart ships based on reinforcement learning. Ocean Engineering [online], 189. Available at: http://www.sciencedirect.com/ [Accessed 1 Mar 2020].

[Carreras et al., 2005]: Carreras, M. & Yuh, J. & Ridao, P. (2005). A Behavior-Based Scheme Using Reinforcement Learning for Autonomous Underwater Vehicles. IEEE Journal of Oceanic Engineering. 30, pp 416 - 427.

[Mnih et al., 2013]: Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D. and Riedmiller, M. (2013) Playing Atari with deep reinforcement learning. 1312.5602.

[Y. Cheng et al., 2018]: Cheng, Y. & Zhang, W. (2018) Concise deep reinforcement learning obstacle avoidance for underactuated unmanned marine vessels. Neurocomputing [online], 272, pp. 63–73. Available at:http://www.sciencedirect.com/ [Accessed 1 Oct 2019].

[Sutton and Barto, 2018]: Sutton, R.S. & Barto, A.G. (2018) Reinforcement Learning An Introduction. 2nd ed.

[Guo et al., 2020]: Guo, A., Zhang, Z., Zheng, X., and Du, D. (2020). An Autonomous Path Planning Model for Unmanned Ships Based on Deep Reinforcement Learning. Sensors. 20, pp 426.

[Woo et al., 2019]: Woo, J., Yu, C. and Kim, N. (2019) Deep reinforcement learning-based controller for path following of an unmanned surface vehicle, Ocean Engineering. 183, pp 155-166.

[Zhao et al., 2019]: Zhao, L. and Roh, M. (2019) COLREGs-compliant multiship collision avoidance based on deep reinforcement learning, Ocean Engineering. 191.

[Bellman, 1957]: Bellman, R. (1957) A Markovian decision process. J. Math. Mech. pp 679–684.

[Anonymous, 2020]: Container Ship Top Red Icon [online image] Available at:://www.iconarchive.com/show/transporter-icons-by-icons-land/Container-Ship-Top-Red-icon.html [Accessed 3 Mar 2020].

[Swarup et al., 2016]: Swarup, D. (2016). Evolution of Ship's Mathematical Model from Control Point of View Lt Cdr Swarup Das.

[Carrillo et al., 2018]: Carrillo, S. & Contreras, J. (2018) Obtaining First and Second Order Nomoto Models of a Fluvial Support Patrol using Identifcation Techniques. Ship Science & Technology [online], 11, pp 19-28. Available at: https://doi.org/10.25043/19098642.160 [Accessed 10 Mar 2020].

[Davidson el at., 1946]: Davidson, K. S. M. and Schiff, L. I. (1946) Turning and course keeping qualities. Transaction of Society of Naval Architecture and Marine Engineering. 54, pp 354–370.

[Nomoto et al., 1957]: Nomoto, K., Taguchi, T., Honda, K. and Hirans, S. (1957) On the steering qualities of ship. International Shipbuilding Progress. 4, pp 354–370.

[Pradeep et al., 2015]: Pradeep, M., Panigrah, S., Swarup, D., Mechanical, D., Pune, M. & Email, E. (2015). Ships Steering Autopilot Design by Nomoto Model. International Journal of Mechanical Engineering and Robotics (IJMER). 3, pp 2321-5747.

[Shen et al., 2005]: Shen, Z., Guo, C. & Yuan, S. (2005) Reinforcement learning control for ship steering using recursive least-squares algorithm. IFAC Proceedings Volumes [online], 38(1), pp.133-138. Available at: https://doi.org/10.3182/20050703-6-CZ-1902.00243 [Accessed 10 Mar 2020].

[Sutton et al., 1997]: Sutton, R., Roberts, G.N., Taylor, S.D.H. (1997) Tuning fuzzy ship autopilots using artificial neural networks. Transactions of the Institute of Measurement and Control. 19(2), pp. 94-106.

[Schaul et al., 2015]: Schaul, T., Quan, J., Antonoglou, I. & Silver, D. (2015). Prioritized Experience Replay.

[Shen et al., 2017]: Shen, Y., Zhao, N., Xia, M., & Du, X. (2017). A Deep Q-Learning Network for Ship Stowage Planning Problem. Polish Maritime Research. 24(3), pp. 102-109.