A Deep Reinforcement Learning approach for Autonomous Docking of an Underactuated Surface Vessel

Daniel Filipe Norte Fortunato
daniel.fortunato@tecnico.ulisboa.pt

Instituto Superior Técnico, Lisboa, Portugal

February 2021

Abstract

With the growing interest in autonomous navigation systems, advanced vessel motion control techniques are being developed to ensure that vessels can independently control their own actions, especially in tasks where high precision is required. One of these tasks is the docking procedure. The objective of this task is to approach the vessel to the docking position in safety, gradually decreasing the speed until a complete stop. At low velocities the low propulsion on the rudder reduces the vessel's maneuverability. This combined with the high density of obstacles in the harbor environment makes the docking task one of the most challenging tasks in a maritime setting. In this research, we propose to solve this problem by, first, constructing a realistic 3D harbor environment. Next, due to the learning abilities and the model-free approach, Deep Reinforcement Learning techniques were used to develop an action-planning guidance layer for an underactuated vessel, without a prior path-planning. It was used a Duelling Network in combination with the Double Deep Q-Network algorithm to train the agent. The model was tested for three scenarios: 1) and 2) test the agent's ability to approach the docking position when it is already aligned with it; 3) test the capacity of the agent to make the whole docking maneuver, by starting perpendicular to it. Regarding safety and robustness, results showed good performance in all scenarios, while showing some limitations in control smoothness.
Keywords: Autonomous Docking, Reinforcement Learning, LiDAR, Duelling Network

1. Introduction

With the density of maritime traffic increasing, various types of marine accidents frequently occur [11]. 80% of marine accidents are caused by human factors, according to the world maritime disaster records that were filed by international maritime organization (IMO) from 1978 to 2008 [20].

The harbor areas are particularly dangerous due to the space limitations, reduced maneuverability caused by the low velocities, heavy traffic and external perturbations, such as wind and currents [31]. The docking procedure, although handled successfully on a daily basis, mistakes and accidents are frequent phenomenons resulting in injuries of both personnel and equipment [25]. Therefore, improving the autonomous navigation level of vessels has become an urgent problem to be solved, specially in these stressful tasks that are more prone to errors.

In order to build an autonomous docking system, a robust control technique is required to approximate the vessel to the quay. [34] presents a review on the research on motion control of autonomous surface ships. In this research are presented several control techniques for autonomous ships, such as Proportional Integral Derivative Algorithm (PID), Model Predictive Control Algorithm (MPC) and other optimization algorithms. Techniques based on Artificial Intelligence (AI) were also presented, focusing on Deep Learning (DL) and Deep Reinforcement Learning (Deep RL).

Starting by the PID controller, [8] and [15] propose a PID controller approach for automatic berthing of a catamaran and a ship, respectively. Although the controllers on both researches prove a satisfactory behaviour, with smooth maneuvers, this approach cannot be used in a practical ship berthing system because because it requires manual adjustments to compensate for changes in environment (e.g., wind and sea state) and in the dynamics of the ship (such as speed, draught, and water depth) [41].

Staying in the classical methods, research such as [22], [4] and [19] propose an optimal control method for performing autonomous docking of marine vessels using nonlinear model predictive controller (NLMPC). Despite optimization-based control techniques have been the focus of much research and shown very good results in the maritime field,

this method is heavily dependant on the model of the system and on its accuracy [9]. This motivates the use of intelligent control approaches, such as Machine Learning (ML) and Deep Reinforcement Learning (Deep RL) techniques, as they do not require the exact modeling of the system [14] (model-free).

In such a case, an ANN-based (Artificial Neural Network) approaches have been one of the most commonly used methods due to their learning ability and mimicking actions of the human brain when performing stages of ship docking [19]. Research such as [39], [41], [13], [14], [33] and [28] are some of the examples that proposed an ANN-based approach to solve the autonomous docking problem. However, the drawback of the ANN-based approaches is being dependent not only on the ANN's structure but also on the reliability of the training data.

Contrary to ANN-based approaches, the advantage of using Deep RL is that it does not require any expert to directly teach the agent how to behave under particular conditions [2] since it learns through trial and error interaction with the environment. Although in the maritime field, Deep RL techniques have also received attention in path following [37][40], path planning [**?**][5] and collision avoidance [36] [38] [6], autonomous docking has received little attention. Research such as [17], [26], [18], [2], [3] proposed RL-based approaches to solve the autonomous docking. Although these approaches have shown very good results compared to classical methods, much remains to be done.

While MPC-based methods are heavily dependant on a system model and ANN-based methods need reliable data to train the network, Deep RL approaches do not need any of them. This was the main motivation in using Deep RL to solve this problem. To the best of our knowledge, information from the LiDAR sensor was also not used in the state space for the specific task of autonomous docking. Moreover, most research is done to solve the problem of cargo ships docking at the container port and, to the best of our knowledge, nothing has been done to dock small boats at the marina, which is more restricted due to the high density of obstacles. The main contributions of this research are the following:

- Construction of a simulated harbor for small boats in V-REP framework [27], including the simulation of the vessel's dynamics system and a LiDAR simulation;
- Use a discrete action space Deep RL algorithm (Dueling DDQN [35]) to teach a small vessel to perform optimal docking maneuvers in the simulated harbor;
- Use extra information from the LiDAR sensor in the state space as it provides very important information about the distance to the environment around the vessel.

This research is organized as follows: Initially, **Section 2** presents the the theoretical background needed in order to be able to follow the solutions presented in this research. Next, **Section 3** presents the implementation details. **Section 4** presents the results of our approach for two different scenarios Finally, **Section 5** presents the conclusions and future work.

2. Background

In this Section, it will be first presented the mathematical model of the surface vessel and later the theory behind the reinforcement learning techniques used in this research.

2.1. Mathematical Modeling of a Surface Vehicle

In maneuvering, a marine vehicle experiences motion in 6 degrees of freedom (DOFs) [10]. When designing control systems for marine craft, models with reduced order are often used since most vehicles do not have actuation in all DOF. In this section the motions of the vessel are decoupled according to 3 DOF, where it will be only used the *surge*, *sway*, and *yaw*.

2.1.1 Fossen's Robot-Like Vectorial Model for Marine Vehicles

Robot-like vectorial model is adopted as a standard by the international community due to its easy implementation [10]. The vectorial model can be written as follows:

$$\boldsymbol{M}(\boldsymbol{q})\ddot{\boldsymbol{q}} + \boldsymbol{C}(\boldsymbol{q}, \dot{\boldsymbol{q}}) = \boldsymbol{\tau} \tag{1}$$

This model was used as motivation to derive a compact marine craft model in 3 DOFs using a vectorial setting. A complete 3 DOF vectorial setting for marine vehicles was derived in [10]:

$$\boldsymbol{M}\dot{\boldsymbol{\nu}} + \boldsymbol{C}(\boldsymbol{\nu})\boldsymbol{\nu} + \boldsymbol{D}(\boldsymbol{\nu})\boldsymbol{\nu} + \boldsymbol{g}(\boldsymbol{\eta}) + \boldsymbol{g}_0 = \boldsymbol{\tau} \tag{2}$$

where

$$\boldsymbol{\eta} = [x, y, \psi]^{\mathrm{T}} \tag{3}$$

$$\boldsymbol{\nu} = [u, v, r]^{\mathrm{T}} \tag{4}$$

are vectors of generalized position/Euler angles and velocities, respectively, used to describe motions in 3 DOF. Similarly, $\boldsymbol{\tau}$ is a vector of generalized forces and moments in 3 DOF. The matrices $\boldsymbol{M}$, $\boldsymbol{C}(\boldsymbol{\nu})$ and $\boldsymbol{D}(\boldsymbol{\nu})$ denote inertia, Coriolis and

damping, respecively, while $\boldsymbol{g}(\boldsymbol{\eta})$ is a vector of generalized gravitational and buoyancy forces. Static restoring forces and moments due to ballast systems and water tanks are collected in the term $\boldsymbol{g}_0$.

2.1.2 Kinematics

Position and orientation of the vehicle are calculated in North-East-Down (NED) frame. Inertial sensors whose measurements are located in the body-fixed frame must be converted to NED frame (local navigation frame). This transformation is done with

$$\dot{\boldsymbol{\eta}} = \boldsymbol{R}(\psi)\boldsymbol{\nu} \tag{5}$$

Explicitly,

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\psi} \end{bmatrix} = \begin{bmatrix} \cos(\psi) & -\sin(\psi) & 0 \\ \sin(\psi) & \cos(\psi) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} u \\ v \\ r \end{bmatrix} \tag{6}$$

where $\boldsymbol{R}(\psi)$ is the transformation matrix that is used to transform linear and angular velocities from the body fixed-frame to the NED fixed-frame.

2.1.3 Rigid-Body Kinetics

Kinetics, contrary to kinematics, analyses the forces causing the motion. [10] showed that the rigid-body kinetics of a vessel can be expressed in the following vectorial setting:

$$\boldsymbol{M}_{RB}\dot{\boldsymbol{\nu}} + \boldsymbol{C}_{RB}(\boldsymbol{\nu})\boldsymbol{\nu} = \boldsymbol{\tau}_{RB} \tag{7}$$

where, $\boldsymbol{M}_{RB}$ is the rigid-body mass matrix, $\boldsymbol{C}_{RB}$ is the rigid-body Coriolis and centripetal matrix, and $\boldsymbol{\tau}_{RB}$ is the generalized vector of external forces and moments expressed in the body frame.

The matrices associated with rigid-body kinetics can be described as:

$$\boldsymbol{M}_{RB} = \begin{bmatrix} m & 0 & 0 \\ 0 & m & 0 \\ 0 & 0 & I_z \end{bmatrix} \tag{8}$$

where m is the mass of the vehicle and I_z is the moment of inertia around the z-axis.

$$\boldsymbol{C}_{RB}(\boldsymbol{\nu}) = \begin{bmatrix} 0 & -mr & 0 \\ mr & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \tag{9}$$

2.1.4 Added Mass Dynamics

The general equations of motion of a marine vehicle partially immersed in water contain terms which are due to the added mass effect. In the structure of the current model, it is desirable to divide the forces due to the virtual mass into added mass inertia matrix, $\boldsymbol{M}_A$, and added Coriolis and centripetal matrix, $\boldsymbol{C}_A(\boldsymbol{\nu})$. The virtual mass matrices are written as follows:

$$\boldsymbol{M}_A = \begin{bmatrix} -X_{\dot{u}} & 0 & 0 \\ 0 & -Y_{\dot{v}} & -Y_{\dot{r}} \\ 0 & -Y_{\dot{r}} & -N_{\dot{r}} \end{bmatrix} \tag{10}$$

$$\boldsymbol{C}_A(\boldsymbol{\nu}) = \begin{bmatrix} 0 & 0 & Y_{\dot{v}}v + Y_{\dot{r}}r \\ 0 & 0 & -X_{\dot{u}}u \\ -Y_{\dot{v}}v - Y_{\dot{r}}r & X_{\dot{u}}u & 0 \end{bmatrix} \tag{11}$$

According to [10], the motion equation (2), the matrices $\boldsymbol{M} = \boldsymbol{M}_{RB} + \boldsymbol{M}_A$ and $\boldsymbol{C}(\boldsymbol{\nu}) = \boldsymbol{C}_{RB}(\boldsymbol{\nu}) + \boldsymbol{C}_A(\boldsymbol{\nu})$ can now be described.

2.1.5 Hydrodynamic Damping Forces

Several damping effects result from the interaction between a maritime vehicle and the water. In this research, only linear damping and the drag force will be considered. In many cases, it is convenient to write total hydrodynamic damping as:

$$\boldsymbol{D}(\boldsymbol{\nu}) = \boldsymbol{D} + \boldsymbol{D}_n(\boldsymbol{\nu}) \tag{12}$$

where $\boldsymbol{D}$ is the linear damping matrix due to potential damping and possible skin friction and $\boldsymbol{D}_n(\boldsymbol{\nu})$ is the nonlinear damping matrix due to quadratic damping and higher-order terms. These damping matrices can be written as follows:

$$\boldsymbol{D} = \begin{bmatrix} X_u & 0 & 0 \\ 0 & Y_v & Y_r \\ 0 & N_v & N_r \end{bmatrix} \tag{13}$$

$$\boldsymbol{D}_n(\boldsymbol{\nu}) = \begin{bmatrix} X_{u^2}|u| & 0 & 0 \\ 0 & Y_{v^2}|v| & 0 \\ 0 & 0 & N_{r^2}|r| \end{bmatrix} \tag{14}$$

2.1.6 Engine Thrust

The boat model is equipped with a propeller of variable direction and propulsion magnitude, as shown in the Figure 1. Therefore, the thrust that propels the boat is modeled according to:

$$\boldsymbol{\tau} = \begin{bmatrix} P\cos\phi \\ -P\sin\phi \\ aP\sin\phi \end{bmatrix} \tag{15}$$

where P is the propeller force [N], ϕ is the rudder angle [º], and a is the distance between the propeller and the center of gravity of the vehicle [m].

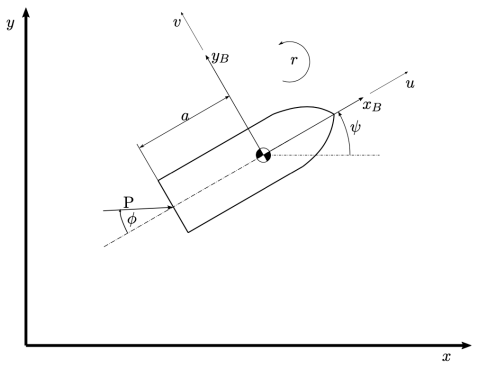

Figure 1: Schematic of the considered boat model

2.2. Reinforcement Learning

We consider a sequential decision making setup, in which an agent interacts with an environment over discrete time steps, see [29] for an introduction. In the research, for example, the agent perceives a state $s_t \in \mathcal{S}$ consisting of vessel information, such as distance and orientation relative to the desired position, velocities and previous control commands, and additional LiDAR information. The agent then chooses an action from a discrete set $a_t \in \mathcal{A} = \{1, ..., |\mathcal{A}|\}$ and observes a reward signal r_t.

The agent seeks maximize the expected discounted return, where we define the discounted return as $R_t = \sum_{\tau=t}^{\infty} \gamma^{\tau-t} r_\tau$. In this formulation, $\gamma \in [0, 1]$ is a discount factor that trades-off the importance of immediate and future rewards.

For an agent behaving according to a stochastic policy π, the values of the state-action pair (s, a) and the state s are defined as follows:

$$Q^\pi(s, a) = \mathbb{E}[R_t | s_t = s, a_t = a, \pi] \tag{16}$$

$$V^\pi(s) = \mathbb{E}_{a\sim\pi(s)}[Q^\pi(s, a)]. \tag{17}$$

The preceding state-action value function (Q function for short) can be computed recursively with dynamic programming:

$$Q^\pi(s, a) = \mathbb{E}_{s'}[r_t + \gamma \mathbb{E}_{a\sim\pi(s')}[Q^\pi(s', a')]|s, a, \pi] \tag{18}$$

We define the optimal $Q^*(s, a) = \max_\pi Q^\pi(s, a)$. Under the deterministic policy $a = \arg\max_{a'\in\mathcal{A}} Q^*(s, a')$, it follows that $V^*(s) = \max_a Q^*(s, a)$. From this, it also follows that the optimal Q function satisfies the Bellman equation:

$$Q^*(s, a) = \mathbb{E}_{s'}[r + \gamma \max_{a'} Q^*(s', a')|s, a] \tag{19}$$

We define another important quantity, the *advantage function*, relating the value and Q functions:

$$A^\pi(s, a) = Q^\pi(s, a) - V^\pi(s) \tag{20}$$

Intuitively, the value function V measures the how good it is to be in a particular state s. The Q function, however, measures the value of choosing a particular action when in this state. The advantage function subtracts the value of the state from the Q function to obtain a relative measure of the importance of each action.

2.2.1 Deep Q-Networks

The value functions as described in the preceding section are high dimensional objects. To approximate them, we can use a deep Q-network: $Q(s, a; \theta)$ with parameters θ. To estimate this network, we optimize the following sequence of loss functions at iteration i:

$$L_i(\theta_i) = \mathbb{E}_{s,a,r,s'}[(y_i^{DQN} - Q(s, a; \theta_i))^2], \tag{21}$$

with

$$y_i^{DQN} = r + \gamma \max_{a'} Q(s', a'; \theta^-) \tag{22}$$

where θ^- represents the parameters of a fixed and separate *target network*. A key innovation in [24] was to freeze the parameters of the target network $Q(s', a'; \theta^-)$ for a fixed number of iterations while updating the *online network* $Q(s, a; \theta_i)$ by gradient descent (This greatly improves the stability of the algorithm).

Another key ingredient behind the success of DQN is *experience replay* [24]. During learning, the agent accumulates a dataset $\mathcal{D}_t = \{e_1, e_2, ..., e_t\}$ of experiences $e_t = (s_t, a_t, r_t, s_{t+1})$ from many episodes. When training the Q-network, instead only using the current experience as prescribed by standard temporal difference learning, the network is trained by sampling mini-batches of experiences from $\mathcal{D}$ uniformly at random, which increases data efficiency and reduces the correlation among the samples used in the update.

2.2.2 Double Deep Q-Networks

The previous section described the main components of DQN as presented in [24]. In this paper we use the improved Double DQN (DDQN) learning algorithm presented in [32]. In Q-learning and DQN, the max operator uses the same values to both select and evaluate an action. This can therefore lead to overoptimistic value estimates [32]. To mitigate this problem, DDQN uses the following target:

$$y_i^{DDQN} = r + \gamma Q(s', \arg\max_{a'} Q(s', a'; \theta_i); \theta^-). \quad (23)$$

DDQN is the same as for DQN [24], but with the target y_i^{DQN} is replaced by y_i^{DDQN}.

2.2.3 Duelling Deep Q-Networks

In this research we used the Duelling Network proposed by [35]. The network architecture is practically the same as in the original DQN [24]. However, instead of following the fully connected layers with the output, it instead uses two sequences (or streams) of fully connected layers (Figure 2). The streams are constructed such that they have the capability of providing separate estimates of the value and advantage functions. Finally, the two streams are combined to produce a single output Q function. As in [24], the output of the network is a set of Q values, one for each action. Since the output of the dueling network is a Q function, it can be trained with the many existing algorithms. In this research it was used the DDQN presented in the Section 2.2.2.

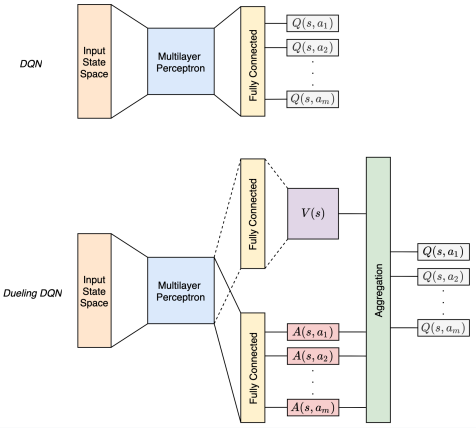

Figure 2: Single stream Q-network (top) and the dueling Q-network (bottom). The dueling network has two streams to separately estimate the state-value $V(s)$ and the advantages for each action $A(s, a_1), ..., A(s, a_m)$; Both networks output Q-values for each action.

The motivation behind this dueling network was, by explicitly separating the two estimators, the dueling architecture can learn which states are (or are not) valuable, without having to learn the effect of each action for each state, which accelerates the learning process.

3. Implementation

As stated before, the objective of this research is to teach a small vessel to perform optimal docking maneuvers using Deep RL. The agent is trained to approach the dock position in a safe and robust manner by directly mapping, at each time step, the input states to the control variables (Propeller Force and Rudder Angle) using a Deep RL agent. The closed loop system is constructed around the vessel model and the LiDAR sensor to accurately represent the network input states. The state variables are further fed to the RL agent that generates the optimal control commands at each time step. The signal flow is represented in Figure 3.

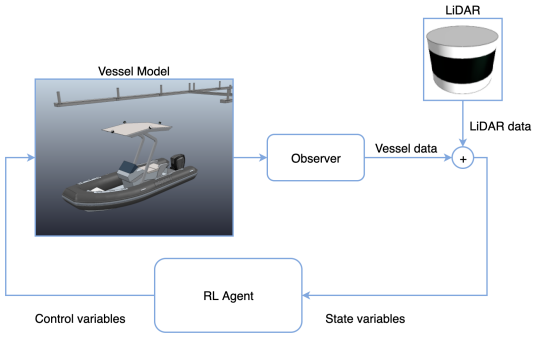

Figure 3: Signal flow in the closed-loop system.

3.1. Simulator

To build the harbor environment (Figure 4) it was used 3D models from Sketchup 3D Warehouse and it was built with the aim of resembling the generic harbours.

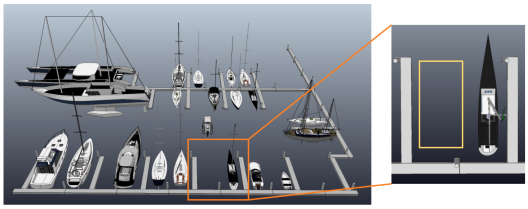

Figure 4: Harbor Simulation; Docking spot.

The agent model employed in the experiments is a small vessel, with $m = 150$ kg, 3.0m of length and 1.0m of width. Since the hydrodynamic parameters presented in the Section 2.1 must be estimated in real experiments, for this research it was used the parameters already estimated by [7]. Just as proposed by [7], the model used in this research also controls the propeller force and the rudder angle.

In order to have a perception of the whole environment around the agent, 4 LiDAR sensors were mounted on top of the boat with specific positions and orientations that allows all the objects to be detected.

The positions and orientations of the 4 sensors relative to the vessel's body-frame are specified in

the Appendix A, where α, β, γ are the rotations around the x-axis, y-axis and z-axis, respectively.

3.2. Problem Modeling as MDP

To achieve this using RL it is first necessary to design the learning strategy. In this section, first, the state space variables are presented, as well as the pre-processing approach for the LiDAR data. Next, the control variables discretization will be explained. Topics such as reward function design and network architecture will also be covered in this section. In the end, the training specifications will be presented.

3.2.1 State Space Representation

The intuition behind the state space representation was to make it as generic as possible in order to be able to transfer the network model to other environments, thus creating the advantage of not having to re-train the model when changing environments. With this in mind, we used the relative distance δd and relative orientation $\delta\psi$ to the dock as the first two state variables. The computation of these two variables can be seen in Equation (24) and (25):

$$\delta d = \sqrt{(x_{boat} - x_{dock})^2 + (y_{boat} - y_{dock})^2} \quad (24)$$

$$\delta\psi = \psi_{boat} - \psi_{dock}, \quad (25)$$

where, (x_{boat}, y_{boat}) and ψ_{boat} are the boat position and orientation, respectively, and (x_{dock}, y_{dock}) and ψ_{dock} are the desired docking position and orientation, respectively.

Since the agent needs to know the boat's current speed to make the decision to reduce it near the final position, in this research, variables such as surge velocity u, sway velocity v, and angular velocity r were added to the state space. To learn which actions lead to which states, the last step control commands P_{i-1} and ϕ_{i-1} are also added to the state space.

Data from LiDAR is also relevant as it provides information on the distance between the boat and other obstacles. In this research, first, the 3D point cloud is transformed into a 2D point cloud that is further segmented in sections using Python's pandas library [23]. This is done by, first, calculating the polar coordinate $\theta = \text{atan2}(y, x)$ of the 2D point cloud. It is possible to segment the point cloud according to the θ values using pandas library function $cut()$. After the θ segmentation it is computed the closest point to the vessel for each point cloud section using k-Nearest Neighbors [16]. The distance to the closest point is then saved for every section and added to the state space. In this research, the point cloud is segmented according to θ in 8 sections, resulting in 8 distances. The work flow for the point cloud pre-processing can be seen in the Figure 5.

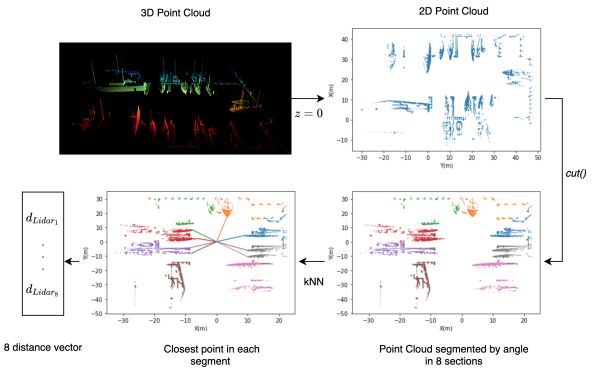

Figure 5: Point Cloud Pre-Processing.

To conclude, the state s_t can be represented as a vector of size 15 as follows:

$$s_t = [\delta d, \delta\psi, u, v, r, P_{i-1}, \phi_{i-1}, d_{Lidar_{1,\dots,8}}]. \quad (26)$$

3.2.2 Action Space

As state before, control variables used in this research are the propeller force P and the rudder angle ϕ. Generally, the maximum allowed speed inside the harbors is 3 knots (kn), which is equivalent to ∼1.5 m/s. However, in the last docking phase, where the boat is already approaching the berthing position, it is necessary that this speed is lower. Due to this restriction, the propeller control command was limited to $P \in [-50, 50N]$ which leads to a maximum speed of approximately 0.4 m/s. The rudder angle will be limited to $\phi \in [-30°, 30°]$.

Since the network presented in Section 2.2.3 has discrete action space it is necessary to discretize the control variables. The propeller force and the rudder angle were both discretized in 10N and 10^{o} intervals, respectively. Multiplying the 11 possible actions of the propeller by the 7 possible actions of the rudder gives the total number of 77 possible actions.

3.2.3 Reward Function

In the last phase of docking, it is possible to formulate the problem as an approximation to the final position and orientation, gradually decreasing the speed, until stopping completely in the desired pose. With this in mind, the reward function designed for this task can be seen in the Equation (27).

$$r = \begin{cases} -r_{collision}, & \text{if collision} \\ r_{final}, & \text{if final pose} \\ d_{term} + \psi_{term} + u_{term}, & \text{else} \end{cases} \quad (27)$$

where,

$$d_{term} = -\delta d * r_{distance}, \tag{28}$$

$$\psi_{term} = -|\delta\psi| * r_{orientation}, \tag{29}$$

$$v_{term} = -\frac{1}{\delta d} * \sqrt{u^2 + v^2} * r_{velocity}. \tag{30}$$

The intuition behind this reward design was to give a very high reward if the agent reaches the final state and a very high penalty if a collision happens. The rewards for these two states are represented in the Equation (27) as r_{final} and $r_{collision}$, respectively. The final state is reached each time (i) the relative distance to the final position is less than 1m, (ii) the relative orientation is less than 0.07 rad and (iii) the speed is less than 0.1m/s. At each time step, if the agent does not reach the final state or does not collide with any objects in the environment, the reward is given by the last equation in (27). The terms in (28) and (29) are responsible for penalizing the relative distance and relative orientation to the final position, respectively, thus encouraging the selection of actions that approximate the agent to the final pose. The term in (30) penalizes the velocity linearly as a function of the distance.

The chosen reward parameters are the following: (i) $r_{collision} = 30000$; (ii) $r_{final} = 30000$; (iii) $r_{distance} = 150$; (iv) $r_{orientation} = 100$; (v) $r_{velocity} = 20$.

3.2.4 Network Architecture

The neural network adopted is a regular Multilayer Perceptron (MLP) [12] with input of dimension 15 (number of state variables) and output of dimension 77 (number of actions). The number of hidden layers was based on another work [1] whose RL-based control task was formulated in a similar way. That research used 3 hidden layers with the 256,128, and 64 perceptrons, respectively.

As state before, the dueling architecture consists of two streams that represent the value and advantage functions, while sharing common layers. For this research, the dueling network architecture is as follows:

- The shared layers have the same layout has the work done by [1] (256x128x64);
- The advantage functions comes after a fully connected layer with size 77, which is the size of the action space;
- The value function also comes after a fully connected layer with size 64.

The adopted activation function was the ReLu (Rectifier Linear Unit) [21] and the RMSProp optimization algorithm [30].

3.2.5 Training parameters

Hyperparameter tunning is also an important task for good performance in RL models. All the parameters used for this research and its respective values are the following: (i) Discount Factor $\gamma = 0.99$; (ii) Experience Replay Buffer = 500,000; (iii) Batch Size = 32; (iv) Target Update Rate = 500; (v) Learning Rate $\alpha = 0.001$.

3.3. Scenarios

For this research, two scenarios where taken into account. In the first scenario, a model will be tested in which, in training, the agent started each episode aligned with the dock (Figure 6). In the second one, each episode will start with the vessel in a position perpendicular to the docking position (Figure 7).

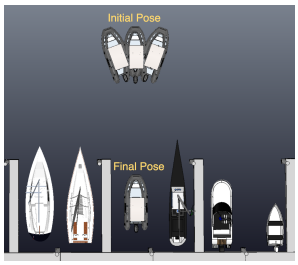

Figure 6: Scenario 1.

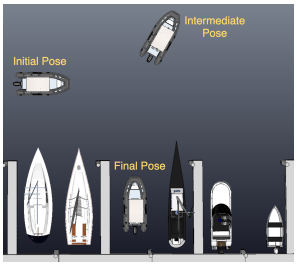

Figure 7: Scenario 2.

To complete the task of the third scenario it is necessary to make some adjustments to the problem formulation. These changes will be explained in the next section.

3.3.1 Scenario 3 Adjustments

A variable s_{task} was added that can only have the value of 0 or 1, depending on the task that the agent is completing (0 if the agent's goal is to navigate away from the dock to be aligned with the final pose; 1 if the agent's goal is to approach the final pose in reverse). The state space is then similar to the representation shown in Equation (26) with the addition of this binary variable s_{task}, making the state space vector with size 16, as shown bellow:

$$s_t = [s_{task}, \delta d, \delta\psi, u, v, r, P_{i-1}, \phi_{i-1}, d_{Lidar_{1,\dots,8}}]. \tag{31}$$

There are also some changes that need to be made to the reward function, as can be seen in Equation (32).

$$r = \begin{cases} -r_{collision}, & \text{if collision} \\ r_{final}, & \text{if final pose} \\ r_{pre-final}, & \text{if intrm. pose} \\ d_{term_0} + \psi_{term_0} + u_{term_0}, & \text{if task 0} \\ d_{term_1} + \psi_{term_1} + u_{term_1}, & \text{if task 1} \end{cases} \tag{32}$$

where $r_{pre-final} = 30000$ is the reward that the agent receives when it reaches the intermediate pose, and the index 0 and 1 in the last two equations in (32) define weather the reward function is penalizing the distance and the orientation relative to the intermediate position (0) or the final position (1). The rest of the parameters are the same as in Equation (27).

4. Results

Results and commentaries for both scenarios are presented in this Section.

4.1. Scenario 1

In this scenario, for each episode the vessel starts already aligned with dock, with a small perturbation imposed in the pose, choosing one of the following poses: (i) $(x_1, y_1, \psi_1) = (20, -2, 200)$, (ii) $(x_2, y_2, \psi_2) = (20, 0, 180)$, (iii) $(x_3, y_3, \psi_3) = (20, 2, 160)$. The goal is to reach the final pose $(x, y, \psi) = (36, 0, 180)$.

The model was trained for a total of about 4700 episodes and the agent's navigation behaviour for this model can be seen in Figure 8.

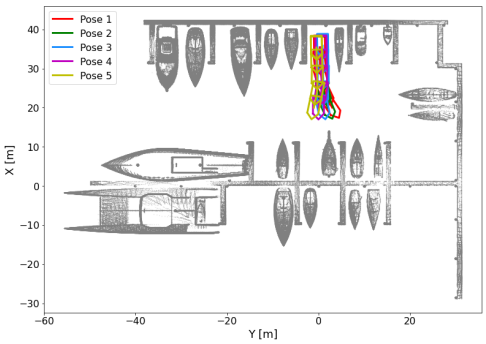

Figure 8: Vessel's pose (x, y, ψ).

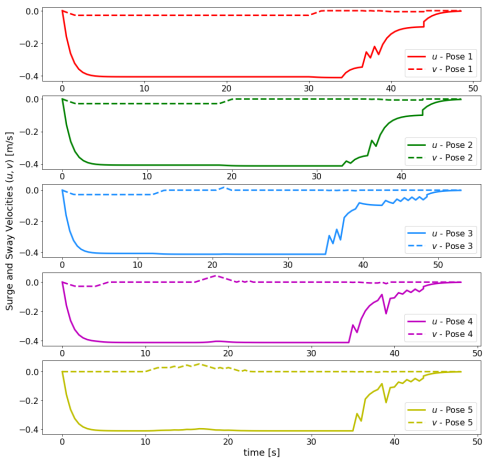

Figure 9: Surge and Sway Velocities (u, v) [m/s].

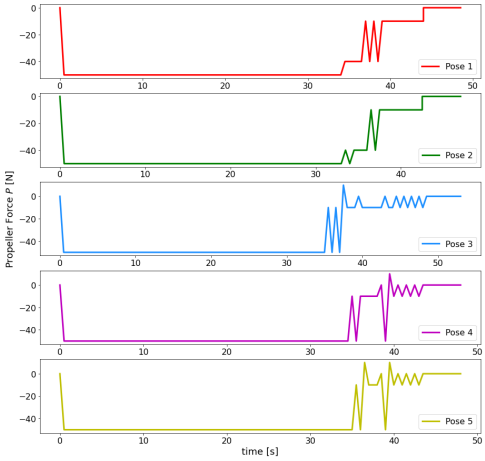

Figure 10: Propeller Force P [N].

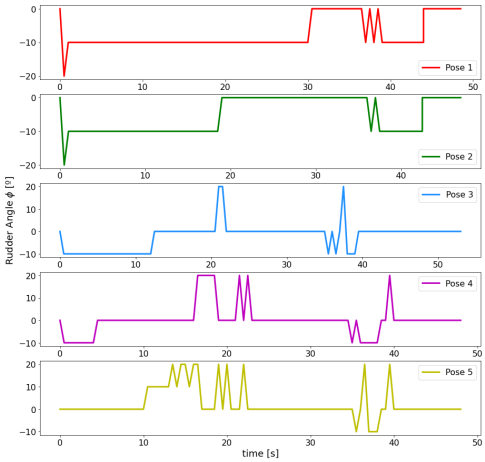

Figure 11: Rudder Angle ϕ [º].

It can be observed in Figure 8 that the agent was successful in safely approaching the desired position in reverse, starting from 5 different positions, with 3 of these positions not appearing in training.

It can be observed in the plots of the surge and sway velocities (Figure 9), that there is a tendency for the agent to put the maximum speed (due to the negative reward given at each step) until the moment in which the velocity penalty becomes too high and then gradually decrease the speed until the final position.

Despite this decrease in speed being accompanied by a somewhat sudden change in propeller force, as can be seen in Figure 10, this does not have a very strong influence on speed, since the agent is acting at low speeds.

4.2. Scenario 2

In this scenario, at each episode, the agent always starts at the same position and orientation $(x, y, \psi) = (23, -10, 90)$. At the beginning of the episode the variable that determines which is the

agent's task has the value 0, and as long as it stays at 0 the reward function will penalize the distance and orientation relative to the intermediate position that is defined as $(x, y, \psi) = (18, 3, 150)$. As soon as this variable changes from 0 to 1, the reward function then penalizes the distance and orientation relative to the final position defined as $(x, y, \psi) = (36, 0, 180)$.

In an attempt to eliminate sudden movements near the final position, the value of the parameter $r_{velocity}$ was changed from 20 to 10 in this Scenario, thus taking away some influence of velocity in the reward function. The model was trained for a total of about 4000 episodes and the agent's navigation behaviour for this model can be seen in Figure 12.

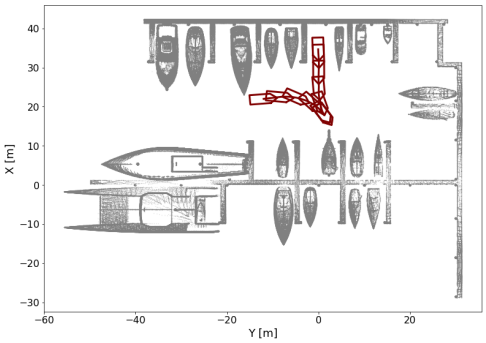

Figure 12: Vessel's pose (s, y, ψ).

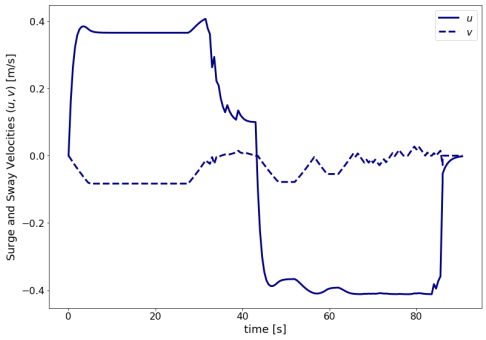

Figure 13: Surge and Sway Velocities (u, v) [m/s].

Starting perpendicularly positioned in relation to the dock, one can see from Figure 12 that the behavior of the agent is very similar to that of a pilot performing this task in real life. The agent first navigates away from the dock in order to be aligned with it and, as soon as the variable s_{task} changes from 0 to 1, the agent starts to navigate towards the final position.

It is also possible to see in the Figure 13 that, at the end of the first task the agent only reduced the speed when it was almost reaching the intermediate position. However, at the end of the second task,

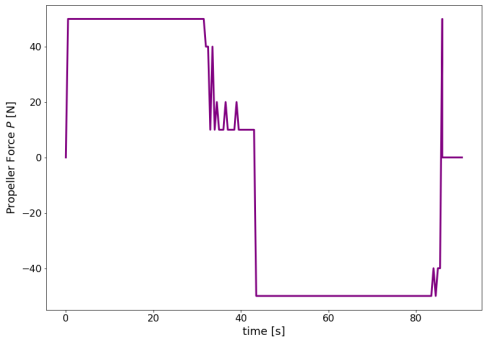

Figure 14: Propeller Force P [N].

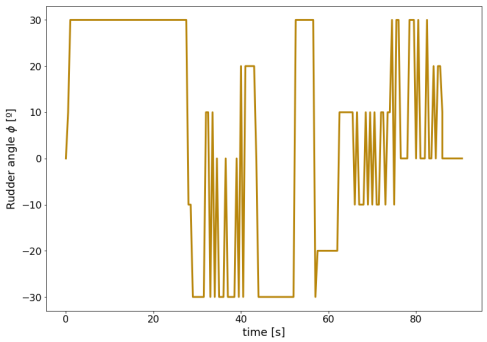

Figure 15: Rudder Angle ϕ [º].

the agent learned a behavior that is actually used by motor boats in docking, which is to keep the velocity constant until the vessel is very close to the final position and when that position is reached a sudden change is made from reverse navigation to forward navigation. This way the inertia that is associated with the constant speed of the boat is cancelled and, consequently, there is a safer stop. One could argue that this late reduction in velocity near the final position can be dangerous in real tests, however, since in this research the velocity was limited to 0.4 m/s, that is not a serious concern.

Compared to the previous scenarios, this reduction in the velocity penalty parameter led to a more desirable behavior, without many abrupt changes in propeller control. However, the ideal behavior thought for this task would be a gradual and smooth reduction of speed.

To complement the tests done here, a test for a scenario that was never seen by the agent during training (Figure 16) is done in order to demonstrate the generalization of the trained model.

This test was able to demonstrate that, even training the model starting always in the same position with the objective of always docking in the same position, the agent managed to dock in an opposite position to the training dock, having been successful in making a maneuver that had never appeared during the training.

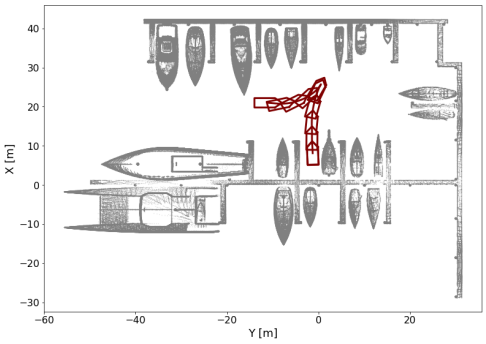

Figure 16: Vessel's pose (x, y, ψ) for docking space in the opposite side.

The general behavior of the agent was positive, but there were limitations. Although the sudden changes in propeller control have practically disappeared with the reduction of the velocity penalty parameter, these sudden changes have not disappeared in rudder control (Figure 15). Despite this behaviour being justified by the lack of maneuverability at low speeds, it is not ideal.

Another limitation to report is that the agent can only reach the final position in safety if its initial position is within 3 meters relative to the initial training position. Further away than this threshold the agent's behavior is one of the following: (i) reaches the tolerance of the intermediate position in a part of the dock where it becomes difficult to approach the final position without a collision; (ii) the agent fails to reach the intermediate position, and, consequently, the variable s_{task} does not change from 0 to 1, but it does not collide with any obstacle either.

5. Conclusions

This research has studied applications of machine learning in a marine setting, specifically Deep RL applied as an action-planning guidance layer for the vessel. To the best of our knowledge, this was also the first time that a Duelling Network combined with LiDAR data was used to solve the autonomous docking problem. From the observed results, it is clear how Deep RL methods are able to teach an agent to learn optimal docking maneuvers through collected experience.

The results reflect how the agent's behavior is highly dependent on reward function design and action space definition. In particular, inspecting the results from the first scenario it is clear how the agent's behaviour reflects the structure and magnitude of the reward components.

Some limited generalization was observed in the first scenario, where the agent was trained for 3 positions, being successful in others slightly different. It was also observed in the second scenario, in which, although it was trained for only one position, the agent was successful in docking in positions that had never been seen in the training. In particular, in positions that needed a maneuver to the opposite side of the dock.

Although the model had good performance results, limitations were found in scenarios where the robustness of the model was being tested for initial positions much further away from the initial training position. As a result of this, it was possible to observe that without a pre-defined path it becomes difficult for the agent to be successful in conditions where it starts far from the docking position.

6. Future Work

The first limitation lies in the collision that occurred when the test episode in Scenario 2 started in a position further away than the initial training position. Future work involves testing the influence of the information coming from the LiDAR in collision avoidance occasions.

In the second scenario, although in state space only the distance to the final position is present, one could argue that changing the vessel's behaviour from forward to reverse navigation only when the agent reaches the intermediate position is a suboptimal solution and can be considered a path-following approach. It remains to be proven that it is possible to design the reward function so that this change is made automatically by the agent, thus achieving an optimal solution.

An analysis that can also be interesting in the future is to compare the performance of this model, which was trained in a network with a discrete action space, with models trained in networks with continuous action space.

To conclude, in order to make the environment as realistic as possible, future work also involves adding external disturbances, such as wind, currents, LiDAR measurements noise, as well as a more complex vessel model (6 DOF) and an actuation delay in control commands.

Acknowledgements

A special thanks to my family and friends, to Professor Alexandre Bernardino, and to my colleagues at BSB AI with whom I had the pleasure of working on this very enriching project.

References

[1] R. P. Abou Rejaili and J. M. P. Figueiredo. Deep reinforcement learning algorithms for ship navigation in restricted waters. *Mecatrone*, 3(1), 2018.

[2] J. Amendola, E. A. Tannuri, F. G. Cozman, and A. H. Reali Costa. Port channel navigation subjected to environmental conditions using reinforcement learning. In *ASME 2019*

38th International Conference on Ocean, Offshore and Arctic Engineering. American Society of Mechanical Engineers Digital Collection, 2019.

[3] E. Anderlini, G. G. Parker, and G. Thomas. Docking control of an autonomous underwater vehicle using reinforcement learning. *Applied Sciences*, 9(17):3456, 2019.

[4] G. Bitar, A. B. Martinsen, A. M. Lekkas, and M. Breivik. Trajectory planning and control for automatic docking of asvs with full-scale experiments. *arXiv preprint arXiv:2004.07793*, 2020.

[5] C. Chen, X.-Q. Chen, F. Ma, X.-J. Zeng, and J. Wang. A knowledge-free path planning approach for smart ships based on reinforcement learning. *Ocean Engineering*, 189:106299, 2019.

[6] C. Chen, F. Ma, J. Liub, R. R. Negenborn, Y. Liu, and X. Yan. Controlling a cargo ship without human experience based on deep q-network. *Journal of Intelligent & Fuzzy Systems*, (Preprint):1–17, 2020.

[7] J. Eilbrecht. Time optimal control of an unmanned ocean surface vessel. 2014.

[8] V. Ferrari, S. Sutulo, and C. G. Soares. Preliminary investigation on automatic berthing of waterjet catamaran. In *Maritime Technology and Engineering*, pages 1105–1112. Taylor & Francis Group London, UK, 2015.

[9] B. Foss and T. A. N. Heirung. Merging optimization and control. *Lecture Notes*, 2013.

[10] T. I. Fossen. *Handbook of marine craft hydrodynamics and motion control*. John Wiley & Sons, 2011.

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

[12] S. Haykin. *Neural Networks: A Comprehensive Foundation*. Prentice Hall PTR, USA, 2nd edition, 1998.

[13] N. Im and K. Hasegawa. Automatic ship berthing using parallel neural controller. *IFAC Proceedings Volumes*, 34(7):51–57, 2001.

[14] N.-K. Im and V.-S. Nguyen. Artificial neural network controller for automatic ship berthing using head-up coordinate system. *International Journal of Naval Architecture and Ocean Engineering*, 10(3):235–249, 2018.

[15] Y. B. Kim, Y. W. Choi, H. Kawai, et al. A study on automatic ship berthing system design. In *2009 International Conference on Networking, Sensing and Control*, pages 181–184. IEEE, 2009.

[16] L. Kozma. k nearest neighbors algorithm (knn). *Helsinki University of Technology*, 2008.

[17] M. Łacki. Reinforcement learning in ship handling. *TransNav: International Journal on Marine Navigation and Safety of Sea Transportation*, 2(2), 2008.

[18] A. Layek, N. A. Vien, T. Chung, et al. Deep reinforcement learning algorithms for steering an underactuated ship. In *2017 IEEE International Conference on Multisensor Fusion and Integration for Intelligent Systems (MFI)*, pages 602–607. IEEE, 2017.

[19] S. Li, J. Liu, R. R. Negenborn, and Q. Wu. Automatic docking for underactuated ships based on multi-objective nonlinear model predictive control. *IEEE Access*, 8:70044–70057, 2020.

[20] D. Liu, Z. Zheng, and Z. Wu. Risk analysis system of underway ships in heavy sea. *J. Traffic Transp. Eng*, 4:100–102, 2004.

[21] A. L. Maas, A. Y. Hannun, and A. Y. Ng. Rectifier nonlinearities improve neural network acoustic models. *in ICML Workshop on Deep Learning for Audio, Speech and Language Processing*, 28, 2013.

[22] A. B. Martinsen, A. M. Lekkas, and S. Gros. Autonomous docking using direct optimal control. *IFAC-PapersOnLine*, 52(21):97–102, 2019.

[23] W. McKinney et al. pandas: a foundational python library for data analysis and statistics. *Python for High Performance and Scientific Computing*, 14(9):1–9, 2011.

[24] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.

[25] B. E. W. Mothes. Reinforcement learning for autodocking of surface vessels. Master's thesis, NTNU, 2019.

[26] A. Rak and W. Gierusz. Reinforcement learning in discrete and continuous domains applied to ship trajectory generation. *Polish Maritime Research*, 19(Special):31–36, 2012.

[27] E. Rohmer, S. P. Singh, and M. Freese. V-rep: A versatile and scalable robot simulation framework. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1321–1326. IEEE, 2013.

[28] Y. Shuai, G. Li, X. Cheng, R. Skulstad, J. Xu, H. Liu, and H. Zhang. An efficient neural-network based approach to automatic ship docking. *Ocean Engineering*, 191:106514, 2019.

[29] R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[30] T. Tieleman and G. Hinton. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, 4(2):26–31, 2012.

[31] A. Trzuskowsky, C. Hoelper, and D. Abel. Anchor: navigation, routing and collision warning during operations in harbors. *IFAC-PapersOnLine*, 49(23):220–225, 2016.

[32] H. Van Hasselt, A. Guez, and D. Silver. Deep reinforcement learning with double q-learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 30, 2016.

[33] V.-C. D. Van-Suong Nguyen and N.-K. Im. Development of automatic ship berthing system using artificial neural network and distance measurement system. *International journal of Fuzzy logic and Intelligent systems*, 18(1):41–49, 2018.

[34] L. Wang, Q. Wu, J. Liu, S. Li, and R. R. Negenborn. State-of-the-art research on motion control of maritime autonomous surface ships. *Journal of Marine Science and Engineering*, 7(12):438, 2019.

[35] Z. Wang, T. Schaul, M. Hessel, H. Hasselt, M. Lanctot, and N. Freitas. Dueling network architectures for deep reinforcement learning. In *International conference on machine learning*, pages 1995–2003. PMLR, 2016.

[36] J. Woo and N. Kim. Collision avoidance for an unmanned surface vehicle using deep reinforcement learning. *Ocean Engineering*, 199:107001, 2020.

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

[38] X. Wu, H. Chen, C. Chen, M. Zhong, S. Xie, Y. Guo, and H. Fujita. The autonomous navigation and obstacle avoidance for usvs with anoa deep reinforcement learning method. *Knowledge-Based Systems*, 196:105201, 2020.

[39] H. Yamato. Automatic berthing by the neural controller. In *Proc. Of Ninth Ship Control Systems Symposium*, volume 3, pages 3183–3201, 1990.

[40] L. Zhang, L. Qiao, J. Chen, and W. Zhang. Neural-network-based reinforcement learning control for path following of underactuated ships. In *2016 35th Chinese Control Conference (CCC)*, pages 5786–5791. IEEE, 2016.

[41] Y. Zhang, G. E. Hearn, and P. Sen. A multivariable neural controller for automatic ship berthing. *IEEE Control Systems Magazine*, 17(4):31–45, 1997.

A. Appendix

	Rotation [º]			Translation [m]		
Lidar Number	α	β	γ	x	y	z
1	0	27	0	1.0	0.0	3.0
2	0	50	90	0.0	0.5	3.0
3	0	35	180	-1.0	0.0	3.0
4	0	50	270	0.0	-0.5	3.0

Table 1: Transformations between each LiDAR body-frame and Agent body-frame.