TÉCNICO LISBOA

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

Daniel Filipe Norte Fortunato

Thesis to obtain the Master of Science Degree in

Electrical and Computer Engineering

Supervisor(s): Prof. Alexandre José Malheiro Bernardino
Eng. Diogo Costa Arreda

Examination Committee

Chairperson: Prof. José Eduardo Charters Ribeiro da Cunha Sanguino
Supervisor: Prof. Alexandre José Malheiro Bernardino
Member of the Committee: Prof. João Fernando Cardoso Silva Sequeira

February 2021

Declaração

Declaro que o presente documento é um trabalho original da minha autoria e que cumpre todos os requisitos do Código de Conduta e Boas Práticas da Universidade de Lisboa.

Declaration

I declare that this document is an original work of my own authorship and that it fulfills all the requirements of the Code of Conduct and Good Practices of the Universidade de Lisboa.

To my grandparents.

Acknowledgments

Em primeiro lugar, queria expressar o meu sentido agradecimento aos meus companheiros de curso. Luís Ferreira, José Nobre, Rafael Vaz, Vasco Candeias, Rafael Forte, André Silva, Pedro Fareleira, Pedro Santos, José Lopes, Pedro Custódio, David Ribeiro, Bernardo Santana, Luís Pujol, sem vocês o Técnico teria sido muito menos tolerável. Não conseguia imaginar um grupo melhor para ter partilhado estes 5 anos de alegrias e desalentos. Aproveito também para dar um especial agradecimento aos membros deste grupo que nunca faltaram a um ”encontro informal” na zona dos churrascos do IST. Pedro Custódio, Bernardo Santana, José Lopes, David Ribeiro, Luís Pujol, que as memórias dos célebres churrascos sejam preservadas por muitos mais anos.

Aos amigos de infância. Francisco Nunes, um obrigado por me mostrares que não importa de onde viemos, mas sim para onde vamos. Tens uma ambição inigualável que eu espero um dia conseguir alcançar. Pedro Formigo, meu Pikus, obrigado pelos 5 anos a partilhar casa. Foste um grande apoio e soubeste sempre o que dizer nos momentos de maior desespero. A tua sensatez e sabedoria levar-te-ão longe. José Pereira, meu amigo e antigo colega de trabalho. São incontáveis as horas e as ideias de negócio que tivemos juntos. Obrigado por me ensinares que a inteligência pode vir em várias formas. A tua sempre me fascinou.

Aos meus pais. Que me possibilitaram tudo sem pedir nada em troca. Que me passaram valores como a perseverança e a disciplina. Que me ensinaram a trabalhar arduamente, mas que acima de tudo me asseguraram um ambiente seguro para crescer. Sinto-me um privilegiado por ser vosso filho e ficar-vos-ei eternamente grato por tudo.

Ao meu irmão João, que desde cedo mostrou uma motivação incomparável para seguir os seus sonhos. Um obrigado pelo modelo de determinação que levarei sempre para a vida.

Ao Diogo Arreda e a todos os meus colegas da BSB AI com quem tive o prazer de trabalhar durante o último ano. Um obrigado pelos conselhos e pelo ambiente enriquecedor que se fazia sentir todos os dias no escritório.

Um agradecimento especial também à Beatriz e à sua família. Foram um apoio fundamental nos momentos de maior indecisão sobre a mudança de curso, que sem o qual eu não estaria aqui hoje. Levarei para a vida a vossa bondade como um modelo a seguir.

À Joana Morgado. Um grande obrigado pela companhia durante os dias de quarentena e por ouvires sempre as minhas inquietações. Certamente que esses dias teriam sido mais penosos sem o teu apoio.

Queria também agradecer ao Professor Alexandre Bernardino. Foi preciso fazer um ajuste enorme na orientação deste ano, mas você mostrou ser incansável na ajuda. Um obrigado pela exigência e por ser sempre tão prestável.

Aos meus avós José e Maria que, infelizmente, não estão presentes para ver esta conquista. Avô, a tua abstrata demonstração de apoio não passou sem ser reconhecida. Avó, levarei para a vida a sua compaixão e benevolência.

E por último, um obrigado enorme à minha irmã gémea Cátia. A minha orientadora na vida. A pessoa com quem mantenho desde sempre uma competição saudável. Ensinaste-me a querer mais e

a ser melhor. Se foi possível chegar aqui, a ti o devo. Desde os dias em que me tiravas todas as dúvidas de matemática, até aos dias de maior desmotivação nesta dissertação. Olharei sempre para ti com um enorme sentimento de admiração, e serás sempre o meu maior exemplo de resiliência. Mesmo não continuando a partilhar Chocapic, espero conseguir continuar a partilhar todas as conquistas contigo. É impagável todo o teu suporte. Love you, Caca.

Resumo

Com o crescente interesse em sistemas de navegação autónomos, estão a ser desenvolvidas técnicas avançadas de embarcações para assegurar que as estas possam controlar independentemente as suas próprias acções, especialmente em tarefas onde é necessária uma alta precisão. Uma destas tarefas é o procedimento de atracagem. O objectivo desta tarefa é a aproximação da embarcação à posição de atracagem em segurança, diminuindo gradualmente a velocidade até uma paragem completa. A baixas velocidades, a baixa propulsão no leme reduz a manobrabilidade da embarcação. Isto combinado com a alta densidade de obstáculos no ambiente portuário torna a tarefa de atracagem uma das tarefas mais desafiantes num cenário marítimo. Nesta investigação, propomo-nos resolver este problema, primeiro, através da construção de um ambiente portuário 3D realista. Em seguida, devido às capacidades de aprendizagem e à abordagem *model-free*, para esta investigação foram utilizadas técnicas de Aprendizagem de Reforço para desenvolver uma sistema de planeamento de acções para um navio subactuado, sem um planeamento prévio do percurso. Foi utilizada uma *Duelling Network* em combinação com o algoritmo *Double Deep Q-Network* para treinar o agente. O modelo foi testado para três cenários: 1) e 2) testam a capacidade do agente para se aproximar da posição de atracagem quando já está alinhado com esta; 3) testa a capacidade do agente para fazer toda a manobra de atracagem, começando perpendicularmente a esta. Relativamente à segurança e robustez, os resultados mostraram bom desempenho em todos os cenários, mas mostraram algumas limitações na suavidade do controlo.

Palavras-chave: Atracagem Autónoma, Aprendizagem por Reforço, LiDAR, Duelling Network

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

Contents

List of Tables

List of Figures

Nomenclature

Abbreviations

ANN	Artificial Neural Network.
AUV	Autonomous Underwater Vehicle.
DDPG	Deep Deterministic Policy Gradient.
DDQN	Double Deep Q-Network.
DL	Deep Learning.
DOF	Degrees of Freedom.
DP	Dynamic Programming.
DQN	Deep Q-Network.
FC	Fully-Connected.
IMO	International Maritime Organization.
LOS	Line-of-Sight.
MASS	Motion Control of Maritime Autonomous Surface Ships.
MC	Monte Carlo.
MDP	Markov Decision Process.
ML	Machine Learning.
MLP	Multilayer Perceptron.
MPC	Model Predictive Control.
NED	North-East-Down.
NLMPC	Nonlinear Model Predictive Control.
NN	Neural Network.
PID	Proportional Integral Derivative.

ReLU Rectifier Linear Unit.

RL Reinforcement Learning.

TD Temporal-Difference.

ToF Time-of-Flight.

USV Unmanned Surface Vessel.

Greek symbols

α Learning rate.

$\boldsymbol{\eta}$ Vector of generalized position/Euler angles.

$\boldsymbol{\nu}$ Vector of generalized velocities.

$\boldsymbol{\tau}$ Vector of generalized forces and moments acting on the vehicle.

$\delta\psi$ Orientation relative to the dock.

$\delta\psi_0$ Orientation relative to the intermediate position.

$\delta\psi_1$ Orientation relative to the final position.

δd Distance relative to the dock.

δd_0 Distance relative to the intermediate position.

δd_1 Distance relative to the final position.

δ_t Temporal-Difference error.

ϵ Random action probability.

γ Discount rate.

∇ Gradient.

ϕ Rudder Angle.

ϕ, θ, ψ Angular coordinates in NED frame.

ϕ_{i-1} Last rudder command.

π Policy.

π_* Optimal policy.

ρ, θ Polar coordinates.

Roman symbols

$\boldsymbol{C}$ Coriolis Matrix.

$\boldsymbol{C}_A$ Added Coriolis and centripetal matrix.

$\boldsymbol{C}_{RB}$ Rigid-body Coriolis and centripetal matrix.

$\boldsymbol{D}$ Linear Damping Matrix.

$\boldsymbol{D}_n$ Nonlinear Damping Matrix.

$\boldsymbol{g}$ Vector of generalized gravitational and buoyancy forces.

$\boldsymbol{g}_0$ Static restoring forces and moments due to ballast systems and water tanks.

$\boldsymbol{I}_g$ Inertia Matrix.

$\boldsymbol{M}$ System Inertia Matrix.

$\boldsymbol{M}_A$ Added mass inertia matrix.

$\boldsymbol{M}_{RB}$ Rigid-body mass matrix.

$\boldsymbol{q}$ Vector of joint angles.

$\boldsymbol{R}$ Transformation matrix that transforms linear and angular velocities from the body fixed-frame to the NED fixed-frame.

$\mathcal{A}$ Action Space.

$\mathcal{S}$ State Space.

a Distance between the propeller and the center of gravity of the vessel.

$A(s, a)$ Advantage.

A_t Agent's action.

D_t Replay memory buffer.

e_t Experience tuple.

G_t Sum of discounted rewards.

I_x, I_y, I_z Moments of inertia about x, y, and z.

I_{xy}, I_{xy}, I_{xz} Products of inertia.

L Loss.

m Mass of the vehicle.

P Propeller Force.

$p(s, a)$ Behaviour distribution.

p, q, r Angular velocities in roll, pitch and yaw.

P_{i-1} Last propeller command.

$Q(S_t, A_t)$ Q-value.

$Q^*(S_t, A_t)$ Target Q-value.

q_* Optimal action-value function.

q_π Action-value function.

R_t Agent's Reward.

$r_{collision}$ Penalty given if there is a collision.

$r_{distance}$ Reward parameter responsible to penalize the distance.

r_{final} Reward given the agent reaches the final docking position.

$r_{orientation}$ Reward parameter responsible to penalize the orientation.

$r_{pre-final}$ Reward given the agent reaches the intermediate position.

$r_{velocity}$ Reward parameter responsible to penalize the velocity.

S_t Agent's current state.

S_{t+1} Agent's next state.

s_{task} Variable that defines in which docking task the agent is.

S_T Terminal state.

T Terminal step.

u, v, w Linear velocities in surge, sway and heave.

$V(s)$ State-Value.

v_* Optimal state-value function.

v_π State-value function.

x, y, z Cartesian coordinates in NED frame.

Subscripts

i Computational indexes.

Superscripts

T Transpose.

Chapter 1

Introduction

In the last few years, there has been an increasing interest in using Machine Learning (ML) techniques in the context of marine systems research. While automated marine navigation systems are well established for open sea navigation, the docking process has resisted automation (that is, the process of taking a ship to its final position in a port) [1].

In practice, the docking procedure performed by a skilled pilot usually consists of three steps [2]: firstly, when the vessel has sufficient acceleration to alter its course with the rudder, the course of the vessel is changed to the desired docking direction. Secondly, the vessel decelerates as it approaches the final position. Thirdly, the engine is stopped at the appropriate time, and the ship moves to the berth with the remaining speed. A demonstration of the docking procedure made by the Volvo Penta's pioneering self-docking technology [3] is shown in the Figure 1.1.

Figure 1.1: Volvo Penta's self-docking technology [3].

Docking a vessel in the harbor is known as one of the most challenging tasks in a maritime setting [2][4][5]. An important restriction in this process is the loss of rudder efficiency as the vessel slows down since there is a reduction in the water flow passing through the rudder, which, consequently, reduces maneuverability.

Since the culmination of almost any voyage executed by a surface vessel includes bringing the vessel to a stationary docking position where the maneuverability is restricted and prone to collisions, automatic vessel docking techniques are required in order to ensure maritime safety.

In order to build an autonomous docking system, a robust control technique is required to approximate the vessel to the quay. Wang et al. [6] presents a review on the research on motion control of autonomous surface ships, in which 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).

RL can not only make use of existing data but also acquire new data by exploring the environment. It is a dynamic learning process of trial and error that does not require previous information of what the learning agent should exactly do at a given moment, thus it is recommended for problems where there are no available data of how the agent should behave, but there is a reasonable notion of how better certain actions or states are over others.

With the proposal of the Deep Q-Network (DQN) [7][8] and its outstanding performance in the game AlphaGo [9][10], there have been several efforts to transfer Deep RL techniques to other areas. Compared with the traditional algorithms based on prior knowledge, Deep RL has a greater ability to adapt to complex environments. Due to this, for this research, it was chosen a Deep RL approach to tackle the autonomous docking problem.

Deep RL algorithms have already been applied to both discrete and continuous action spaces. Research such as Stopforth and Moodley [11] shows that the convergence for discrete action space networks is faster than in continuous networks since there are less possible state-action pairs to map. This motivated the use a discrete action space Deep RL algorithm (Dueling Double Deep Q-Network [12]) in this research which will be applied to produce a stable and reliable controller for a surface vessel to autonomously perform a docking procedure. To achieve this, first, an harbor simulation was developed in the V-REP framework [13], which replicates a generic harbor. This simulator will include 3D realistic meshes of the harbor structure and other docked vessels, the mathematical model that simulates the vessel's dynamic system, and the 3D LiDAR simulation already provided by the V-REP framework [13].

The learning agent acts on the rudder angle and propeller force of an underactuated vessel initially positioned at the entrance of a docking position and its goal it to reach the docking position without colliding with the harbor structures or other parked vessels. The state space variables for this research include the relative distance and orientation to the final position, the vessel velocities, the last rudder and propeller command, and lastly, distance information at several orientations around the vessel provided by the LiDAR sensor. Since the Duelling Network used in this research has a discrete action space, the problem representation incorporates discrete rudder angle and propeller force actuation and discrete time steps between actions that closely match the action of a human pilot during the maneuver.

In this section, the motivation behind the creation of a docking system for autonomous vessels and the detailed objectives of this research will be presented. In order to assess the current state of marine vessel autonomy, a literature review will also be conducted, followed by the detailed contribution of this

research. In the end, the thesis outline will be presented.

1.1 Motivation

With the density of maritime traffic increasing, various types of marine accidents frequently occur [14]. 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 [15].

The harbor areas are particularly dangerous due to the space limitations, reduced maneuverability caused by the low velocities, heavy traffic and external perturbations, like wind and currents [16]. The docking procedure is a very good example of a task performed within this environments where, although handled successfully on a daily basis, mistakes and accidents are frequent phenomenons resulting in injuries of both personnel and equipment [17]. 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.

Moreover, merchant shipping is the lifeblood of the world economy, carrying 90% of international trade [18] and, as stated before, going from point A to point B always presupposes that the vessel will enter a restricted navigation area and stop at a docking spot. For ships in harbors this process often requires the on-bringing and collaboration with a harbor pilot possessing knowledge regarding the particular waterway such as depth, currents, and further local hazards. However, even though availability and complexity of technology has shown exponential increase the last years, it is difficult to argue there exists a substitution for the local experience a harbor pilot can provide [17]. In another perspective however, effects such as depth and currents, given specific factors, such as time of day, tend to hold certain patterns. Thus, rises the question if necessary experience to properly maneuver the vessel in harbours also can be extended to autonomous systems. This places the focus on artificial learning systems such as machine learning, and more specifically reinforcement learning, by letting the vessel gather its own experience, generally in a simulation due to safety reasons.

1.2 State-of-the-art

In order to assess the current state of marine vessel autonomy a literature review was conducted in preparation of this thesis. This review discovered several papers suggesting classical methods as well as machine learning techniques in order to increase autonomy in the docking procedure. The review of classic methods will be brief as more emphasis will be given to machine learning based methods.

Over the course of this review, the terms ”docking” and ”berth” are alternated. Although berthing is referred more to ships were maneuverability is much more complex, it makes sense to include in this review since the problem is similar.

Starting to introducing the control techniques available in the maritime field, Wang et al. [6] elaborates on the state-of-the-art on motion control of maritime autonomous surface ships (MASS). The methods

used are ranging from the classical controllers, such PID control and optimal control, to Machine Learning techniques, such as Deep Learning and Deep Reinforcement Learning.

Starting by the classical methods presented in Wang et al. [6], a proportional–integral–derivative controller (PID controller) is a controller that continuously calculates an error value as the difference between a desired setpoint and a measured process variable and applies a correction based on proportional, integral, and derivative terms (denoted P, I, and D respectively) [19].

The approaches presented in Ferrari et al. [5] and Kim et al. [20], propose a PID controller for automatic berthing of a catamaran and a ship, respectively. In [5], two controllers are designed: one for the propulsor thrust that receives the velocity error and one for the rudder angle that receives the heading error given by the Line-of-Sight (LOS) algorithm [21]. In [20], the same approach is used but, since it is a ship and not a catamaran, bow and stern thrusters are controlled, instead of the propulsor thrust and the rudder angle.

While PID techniques perform reasonably well in a specified performance envelope, their overall effectiveness is limited. This is due to the fact that classical PID autopilots require 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) [22]. Moreover, in order to complete the entire docking task using the PID, it is necessary to make a previous path-planning and follow it.

Staying in the classical methods, another approach used in control theory is the optimal control. Optimal control theory is a branch of mathematical optimization that deals with finding a control for a dynamical system over a period of time such that an objective function is optimized [23]. More specifically model predictive control (MPC), which is an optimization-based control strategy that could provide a flexible framework to deal with complicated and nonlinear system dynamics and system constraints [24]. It has been extensively studied in trajectory tracking control problem of ships [25][26][27].

Transferring the MPC method to the docking task, in Martinsen et al. [28] the authors propose a method for performing autonomous docking of marine vessels using a nonlinear model predictive controller (NLMPC) in a task framed as a dynamic positioning problem, with the addition of spatial constraints that ensure collision avoidance. That is achieved by choosing an operating region that encompasses the desired docking position, while not intersecting with obstacles or land. The cost function is then composed of (i) the difference between the vessel's pose and the desired pose, (ii) the velocity and (iii) the force of the thrusters. The optimization problem constraints are (i) the vessel's dynamic model constraints, (ii) the spatial constraints given by the chosen convex region, (iii) the control saturation constraints and (iv) the initial conditions over a time horizon. Although the proposed method works very well with the vessel approaching the target poses without violating the spatial constraints, solving the open loop optimization problem takes 2 to 4 seconds. However, the main drawback of this approach is being dependent on a model and its accuracy, which is not ideal in an highly dynamic environment.

Bitar et al. [24] proposed to solve the problem of the environmental disturbances mentioned in Martinsen et al. [28] by, instead of running the trajectory planner as an MPC controller with the inputs directly, the state trajectory given by an optimization-based trajectory planner is sent to a trajectory-tracking PID controller to account for disturbances and unmodeled dynamics, separating the trajectory planning from

the motion control. This separation gave several advantages, comparing to the Martinsen et al. [28] research, since the PID controller accounts for steady-state disturbances and corrects for modeling errors, as opposed to the MPC approach, and having a trajectory-tracking controller as the bottom control layer makes the docking system more robust to situations where the solver fails to find a feasible solution. This approach was tested experimentally in confined waters in Trondheim, Norway, and produced safe maneuvers, avoiding collision with static obstacles and completing the docking phase. Overall, this method has proven to be suitable for the docking procedure. The only drawbacks was the requirement of a geographic map, which in this case was known a priori, leaving for future work the integration with maps obtained from exteroceptive sensors, such as lidar or radar, through simultaneous localization and mapping techniques.

In an NLMPC problem, like the last approaches presented, the objective function usually consists of a weighted sum of different terms including the ship's position deviations, course deviations and velocity deviations from desired values. The selection of different weights generates different solutions. Selecting an appropriate set of weights can be a complicated and time consuming task, since is usually done by trial and error. Li et al. [29] proposes to solve this parameter tuning problem by using a multi-objective optimization strategy in the design of the MPC controller to provide optimal ship rudder angles and propeller revolution rate in a ship docking process. This is done through the division of the overall objective function into several sub-functions, ranked from the most important to the least important. An optimal solution is found for the first and most important objective function. Then the second most important objective function is minimized by adding a new constraint which ensures that the value of of the first objective function could preserve its optimal value, i.e., it should not be worse than the value of the solution in the previous optimization problem. This is the solution of the original problem and the whole process repeats at each time interval. Although this method showed higher computation time, since the controller needs to solve three optimization problems instead of one, the results showed better performance than the direct NLMPC method and removed the need for the time consuming task of parameter tuning.

Despite optimization-based control techniques have been the focus of much research and shown very good results in the maritime field, this method is dependant on the system's model. Though simply having a model is not enough, they are also dependant on this model being very accurate to get successful results [30]. This means that if the model deviates from the actual physical system it might be impossible to get the desired results using optimization. This motivates the application of intelligent control approaches, such as machine learning and deep reinforcement learning techniques, as they do not require the exact modeling of the system [2].

In such a case, an ANN-based (Artificial Neural Network) approach with the ability to learn the underlying nature from maneuvering data, provides a potential solution for automatic ship docking [31]. ANN's 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 [29]. Moreover as the network learns from examples and adapts to each situation based on its past experience, it can generalize knowledge to produce adequate responses to any unknown situation. Thus, ANN is expected to solve complex

problems like automatic berthing, that are difficult to manage by other controlling systems [32].

Following Nguyen [33] review on ANN-based techniques for automatic berthing, the most pertinent methods will be presented in the following lines. Starting from the beginning, the first automatic berthing system based on ANN was demonstrated in 1990's by Yamato [34]. In this work, the inputs of the neural network included the ship coordinates, ship heading, ship velocities, and beam distances. The advantage of this approach is the proposed system working as a human brain when maneuvering the ship in the berthing process. However, this research did not consider to external disturbances, such as wind and currents.

Later, Zhang et al. [22] proposed a multivariable controller for ship berthing with (i) the ship state (pose and velocities), (ii) the desired states and (iii) the control signal at previous steps as input, and the (iv) propeller revolution and the rudder angle as output. This research presents and online training approach, instead of the offline approach proposed by Nguyen [33]. An advantage of such online approach is the complete removal of any dependence upon the proper identification of the mathematical model expressing the dynamics of the ship. The other advantage is that the network is trained by directly evaluating the performance error of the controller and therefore the need for off-line training and "trainer" associated with supervised control can be removed. Furthermore, because the network training is undertaken repeatedly in each control cycle, any changes to the ship dynamics can be identified online, and hence the control strategy is useful for time-varying systems under changeable environmental conditions. However, to carry out this idea, a route planning block is required to create waypoints, and thus, this is a track-keeping control and not a ship berthing control.

In a subsequent work, [35] introduced a parallel controller with two sub-networks in a hidden layer. The input of the network includes the ship state (pose and velocities) as well as the distance to the berth position. That research has separated the network hidden layers that control the engine and the rudder. The engine control would not be affected by the heading angle, lateral speed, angular velocity, etc., when the ship is far away from the wharf, since when speed reduction is needed, the pilot just takes the remaining distance to the goal and the current speed into considerations. The result showed that performance of the parallel controller is better than the central one. Successful berthing has been accomplished even if under a different starting point from the ones provided in the training data. Moreover, disturbances, such as wind and current were also included in the research.

The drawback is, since the global pose of the boat is used as the network input, these methods do not work in other ports, where the position of the berth is different from the ones used in the training data. Im and Nguyen [2] and [36] proposed to solve this problem by including the relative bearing and distance to the berth in the input data, instead of the ship's global pose. This controller is proposed in order to control a ship into a berth in different ports without retraining the ANN structure. Optimal docking maneuvers are learned from a set of time series of successful berthing maneuvering processes executed by a skilled captain and tested in different ports and terminals. The results verify that the proposed ANN controller can control the ship into the berth in the original port (training data) and different ports or multi-terminals without retraining the ANN structure, which makes it more time-effective that the ones in previous research. The drawback, however, lies in the difficulty to collect large number of real ship

docking data under different cases of environmental disturbances for training.

Shuai et al. [31] proposed to solve the data collection problem and proposed a joystick implementation in simulation in order to provide manual maneuvering with the objective of collecting sufficient and reliable data from successful maneuvers. The simulation module consists of a ship model and environmental disturbance models and training data was created through a manual controller, which was a skilled captain using a joystick to control the ship's rudder and thrust. A data processing module was also proposed in order to use an optimal subset from the training data for ship control. This module computed the correlation between two features to detect the dependency of two input parameters and removed one of them. It was also computed the influence of each input on the output and removed the ones where the influence was close to zero. An ANN with parallel structure, similar to the one presented in [35], is also proposed to control the ship's thrust and rudder, respectively. Results show that under no wind and constant wind conditions, this approach works very well, with only small fluctuations in command of rudder angle and thruster speed, while under dynamic wind environment, the performance is inferior but still can steer the ship into dock.

The robustness of the ANN-based approaches depend not only on the ANN's structure but also on the training data. The training data can be either from real ship docking operation [2], or from reliable simulation. The former has high reliability, but is difficult to be collected under different environmental disturbances; the latter is most frequently used [31] due to its high efficiency, but the reliability of the training data is dependent on the fidelity of the ship model, which can be complex and time-consuming.

Contrary to ANN-based approaches, the advantage of using reinforcement learning is that it does not require any expert to directly teach the agent how to behave under particular conditions [37]. Although the ANN-based approaches have shown very promising results, there has been a demand to introduce RL techniques in the maritime field, since the agent collects data in an online manner through trial and error interaction with the environment and it is not necessary to deal with the data collecting complexity.

Reinforcement learning has attracted extensive attention in recent years, with the proposal of Deep Q-Network (DQN) [7][8] and the outstanding performance in the game AlphaGo [9][10]. Although in the maritime field, RL techniques have also received attention in path following [38][39], path planning [**?**][40] and collision avoidance [41] [42] [43], autonomous docking has received little attention.

Even though RL is a promising approach for autonomous docking without previous trajectory samples, little has been done in this direction [1]. Exceptions can be found in the work of Łacki [44] where a simulated model of a ship is treated as an agent, which through environmental sensing learns itself to navigate through restricted waters selecting an optimum trajectory. The decision phase of the task is to observe current state and choose one of the available actions. Variables such as (i) the position of the ship, (ii) ship's orientation and (iii) angular velocity were selected to represent the state space. Constant velocity and discretized rudder angles were considered to simplify the problem.

Later, Rak and Gierusz [45], also presented the application of the reinforcement learning algorithms to the task of autonomous determination of the ship trajectory during the in-harbour and harbour approaching manoeuvres. A discrete (Q-learning) and a continuous (Least-Squares Policy Iteration) algorithm were tested for this task, showing very promising results.

Although the last two researches showed very promising results, both chose tabular techniques to solve the maneuvering problem. This tabular approach uses a table as the brain of the agent. This table has size $n \times m$, where n is the number of possible states and m is the number of possible actions. Each cell in this table contains a value for being in a certain state and performing an action. This approach is efficient in environments with small state-space and action-space, but becomes unfeasible when the table becomes too big. Due to this, Layek et al. [46] proposed two distinct Deep RL-based controllers with continuous action-space to pass a ship through a specified gate. The controllers were developed based on two Deep RL algorithms: Deep Deterministic Policy Gradient (DDPG) and Normalized Advantage Function (NAF). The state space variables were the position, orientation and the turning rate and the action space variable is the continuous angle velocity (or rudder angle). The results showed that Deep RL algorithms have been proven to help the ship to pass a specified gate. The ship was able to reach the gate in short time, while starting at any position and orientation. However, for this scenario, the learned controller needs to satisfy multiple objectives (pass multiple gates), which is hard to achieve for a primitive Deep RL.

Later, Amendola et al. [37] also proposed a machine learning agent for navigating a vessel in restricted waters. More specifically, in a restricted and narrow channel, subjected to environmental conditions. The agent is trained to control the rudder of the ship in order to keep the vessel inside the channel with minimum distance from the center line and to reach the final part of the channel with a prescribed thruster rotation level. The state-space variables were the distance to the center of the line, the course of the ship, the angular velocity and the last rudder command given. The action space is the rudder level discretized in 5 values. This approach enabled human-style maneuvering through discrete rudder levels and discrete interval between actions. The problem representation allowed the trained agent to navigate with variations on environment conditions and managed to be even extended to channels with different length and smooth width variations.

Research in the docking task for autonomous underwater vehicles (AUVs) using Deep RL has also been done. Anderlini et al. [47] presented for the first time Deep RL strategies to control the docking of an AUV onto a fixed platform in a simulation environment. Two RL algorithms were tested: one with continuous action space (DDPG) and one with discrete action space (DQN). The RL approach was compared with a PID controller and with optimal control. The authors concluded that, although RL algorithms present a very high computational cost at training time, the deployment time is five order of magnitude faster than optimal control, enabling an online implementation. Furthermore, its model-free nature provides a general framework that can be easily adapted to the control of different systems.

After this literature review, it is possible to conclude that there is a great density of research aimed at solving the autonomous docking problem with classical methods, such as MPC, and ML methods, such as ANNs, with training data collected from real or simulated data maneuvers done by experienced pilots. However, Deep RL methods have seen little attention. 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, since they are model-free algorithms that learn to choose optimal actions via interaction with the environment.

Adding to the lack of research done in Deep RL-based controllers for docking systems, the non reliance on a model and the fact that no data collection from successful docking maneuvers is needed were the main motivation to use Deep RL to solve this problem. As far as our knowledge goes, information from the LiDAR sensor was also never 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. Although this environment is very restricted and the docking problem is very complex to solve due to the low speeds that lead to low maneuverability, it becomes more complex when small boats, such as sailing boats, and yachts need to dock at the marina. This is a much more restricted environment with a lot more obstacles and small distances between them and the vessel, comparing with the cargo ports.

The main contributions of this research are the following:

- Construction of a simulated harbor for small boats in V-REP framework [13], including the simulation of the vessel's dynamics system and a LiDAR simulation;
- Use a discrete action space Deep RL algorithm (Dueling DDQN [12]) 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.

1.3 Objectives

The primary objective of this project is to study how Deep RL can be used to make the control system of an autonomous vessel learn optimal docking maneuvers. To dock a vessel safely, the vessel's states, including position, heading angle, and velocity need to be controlled appropriately by the rudder and the propeller. This research formulates a nonlinear vessel model that uses rudder–propeller configuration. That is, the propeller force and the rudder angle are the main control inputs, which is in compliance with ship docking practise. It will be developed a simulator on V-REP robot simulation framework [13] that will contain this vessel configuration as well as the simulation of the LiDAR that will be used to add extra information about the environment in the agent's state space.

It must be noted that shallow water or quay effects [48] and external disturbances, such as wind [49] and current [50], can influence the motion of the vessel but these effects are not considered in this research.

The intention is not to develop a docking system to be fully functional in the real world, but rather to build a proof of concept and generate understanding and insight in the field of machine learning applied to a marine setting. On the basis of this, final numerical results from training are of less importance, but rather the observed possibilities of improvement and behaviour according to action spaces and reward functions.

This research will refer only to the final phase of approaching the vessel to the wharf and not the entire harbor navigation.

1.4 Thesis Outline

This research is organized as follows:

- Initially, **Chapter 2** presents the the theoretical background needed in order to be able to follow the solutions presented in this research, such as the vessel's mathematical model and the RL techniques;
- Next, **Chapter 3** presents the implementation details about the simulator, the neural network architecture, as well as state space action space and reward function;
- **Chapter 4** presents the results of our approach for two different scenarios, as well as commentaries on the performance of each one and limitations. A general comment that summarizes the overall performance is done in the end;
- Finally, **Chapter 5** presents the conclusions and future work.

Chapter 2

Theoretical Background

The following chapter will contain an explanation of various topics and theories needed in order to be able to follow the solutions presented in this Master's thesis. A mathematical model of the surface vessel as well as the theory behind the reinforcement learning techniques used in this research will be presented.

2.1 Mathematical Modeling of a Surface Vehicle

In maneuvering, a marine vehicle experiences motion in 6 degrees of freedom (DOFs) [51]. The DOFs are the set of independent displacements and rotations that specify completely the displaced positional and orientation of the vehicle. The motion in the horizontal plane is referred to as *surge* (longitudinal motion, usually superimposed on the steady propulsive motion) and *sway* (sideways motion). *Yaw* (rotation about the vertical axis) describes the heading of the vehicle. The remaining three DOFS are *roll* (rotation about the longitudinal axis), *pitch* (rotation about the transverse axis) and *heave* (vertical motion) (Figure 2.1).

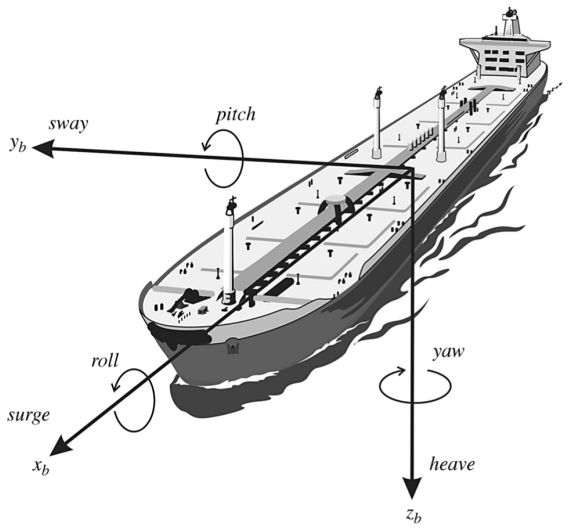

Figure 2.1: Vessel motion in 6 degrees of freedom [51].

When designing control systems for marine craft, models with reduced order are often used since most vehicles do not have actuation in all DOF [51]. Due to this, and because in the harbor environment the *roll*, *pitch* and *heave* are negligible, 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*.

A mathematical model for an USV based on Fossen's research [51] is used in this chapter. The model is obtained from the kinematics of a physical system representing a USV and comprises external forces acting on the hull including torque commands for controlling the vehicle. The body of the boat is taken as rigid, and the equations of motion, which are used to construct the mathematical model, are based on rigid body theory.

This model includes rigid body dynamics: mass and inertia, centripetal and Coriolis forces and damping forces. Furthermore, the vehicle can be exited by some external forces and moments due to various sources: thrusters and air drag due to vehicle speed.

This chapter covers the following subjects in detail. First of all, Fossen's robot-like vectorial model which is used to define the kinetic of the sea surface vehicle will be explained briefly. After that, kinematics will be expressed for vector transformation from body to inertial frame, and coordinate transformation between these two frames is shortly explained. Rigid body dynamics and forces will be discussed as well.

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

A robot-like vectorial model is adopted as a standard by the international community due to its easy implementation [51]. The system can be easily examined for stability and motion characteristics because of the properties of inertia, Coriolis and centripetal matrices. These properties are also useful for computational purposes to reduce the number of coefficients needed to design controllers. The vectorial model can be written as follows:

$$\boldsymbol{M}(\boldsymbol{q})\ddot{\boldsymbol{q}} + \boldsymbol{C}(\boldsymbol{q}, \dot{\boldsymbol{q}}) = \boldsymbol{\tau} \tag{2.1}$$

This model was used as motivation to derive a compact marine craft model in 6 DOFs using a vectorial setting. In the robot model $\boldsymbol{q}$ is a vector of joint angles, $\boldsymbol{\tau}$ represents the moments and forces acting on the vehicle, while $\boldsymbol{M}$ and $\boldsymbol{C}$ denote the system inertia and Coriolis matrices, respectively.

Based on these ideas, a complete 6 DOF vectorial setting for marine vehicles was derived in Fossen [51]:

$$\boldsymbol{M}\dot{\boldsymbol{\nu}} + \boldsymbol{C}(\boldsymbol{\nu})\boldsymbol{\nu} + \boldsymbol{D}(\boldsymbol{\nu})\boldsymbol{\nu} + \boldsymbol{g}(\boldsymbol{\eta}) + \boldsymbol{g}_0 = \boldsymbol{\tau} \tag{2.2}$$

where

$$\boldsymbol{\eta} = [x, y, z, \phi, \theta, \psi]^{\mathsf{T}} \tag{2.3}$$

$$\boldsymbol{\nu} = [u, v, w, p, q, r]^{\mathsf{T}} \tag{2.4}$$

are vectors of generalized position/Euler angles and velocities, respectively, used to describe motions in 6 DOF. Similarly, $\boldsymbol{\tau}$ is a vector of generalized forces and moments in 6 DOF. The matrices $\boldsymbol{M}$, $\boldsymbol{C}(\boldsymbol{\nu})$ and $\boldsymbol{D}(\boldsymbol{\nu})$ denote inertia, Coriolis and damping, respectively, 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 $\boldsymbol{g}_0$.

As mentioned before, a simpler version of this model will be considered. The vector of generalized gravitational and buoyancy forces $\boldsymbol{g}(\boldsymbol{\eta})$ as well as the static restoring forces and moments due to ballast systems and water tanks $\boldsymbol{g}_0$ will not be used in this research. The motion will be described in 3 DOF, instead of 6 DOF. Hereupon, the (2.2), (2.3), and (2.4) equations will be respectively modified into:

$$\boldsymbol{M}\dot{\boldsymbol{\nu}} + \boldsymbol{C}(\boldsymbol{\nu})\boldsymbol{\nu} + \boldsymbol{D}(\boldsymbol{\nu})\boldsymbol{\nu} = \boldsymbol{\tau} \tag{2.5}$$

where

$$\boldsymbol{\eta} = [x, y, \psi]^{\mathsf{T}} \tag{2.6}$$

$$\boldsymbol{\nu} = [u, v, r]^{\mathsf{T}}. \tag{2.7}$$

2.1.2 Kinematics

Kinematics only handle geometrical aspects of motion without knowledge of the forces acting on the vehicle [51]. 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. This transformation is done with

$$\dot{\boldsymbol{\eta}} = \boldsymbol{R}(\psi)\boldsymbol{\nu} \tag{2.8}$$

Explicitly,

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\psi} \end{bmatrix} = \begin{bmatrix} \cos(\psi) & -\sin(\psi) & 0 \\ \sin(\psi) & \cos(\psi) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} u \\ v \\ r \end{bmatrix} \tag{2.9}$$

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. Fossen [51] showed that the rigid-body kinetics of a vessel can be expressed in the following vectorial setting:

$$\boldsymbol{M}_{RB}\dot{\boldsymbol{\nu}} + \boldsymbol{C}_{RB}(\boldsymbol{\nu})\boldsymbol{\nu} = \boldsymbol{\tau}_{RB} \tag{2.10}$$

where, $\boldsymbol{M}_{RB}$ is the rigid-body mass matrix, $\boldsymbol{C}_{RB}$ is the rigid-body Coriolis and centripetal matrix, $\boldsymbol{\nu} = [u, v, r]^{\mathsf{T}}$ is the generalized velocity vector expressed in the body frame, and $\boldsymbol{\tau}_{RB}$ is the generalized vector of external forces and moments expressed in the body frame.

The equation of motion are usually represented in two body-fixed reference points: **CO** - origin of the body frame; **CG** - center of gravity. Figure 2.2 shows a diagram of these two frames.

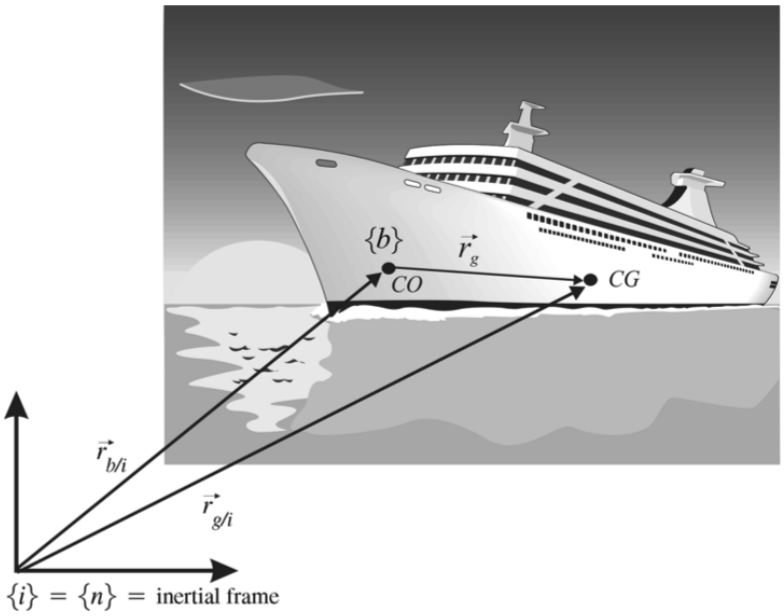

Figure 2.2: Center of gravity and origin fixed-frames diagram [51].

The inertia matrix $\boldsymbol{I}_g$ is defined as

$$\boldsymbol{I}_g = \begin{bmatrix} I_x & -I_{xy} & -I_{xz} \\ -I_{yx} & I_y & -I_{yz} \\ -I_{zx} & -I_{zy} & I_z \end{bmatrix} \tag{2.11}$$

where I_x, I_y and I_z are the moments of inertia about x, y and z-axes, respectively, in the body fixed frame, and $I_{xy} = I_{yx}$, $I_{xz} = I_{zx}$ and $I_{yz} = I_{zy}$ are the products of inertia.

It is common to assume that the vehicle has homogeneous mass distribution and xz-plane symmetry so that

$$I_{xy} = I_{yz} = 0 \tag{2.12}$$

In this research we assume that the body frame coordinate origin coincide with the center of gravity,

such that $y_g = x_g = 0$, where y_g and x_g are the distances in x and y between the center of gravity and the origin of the body fixed-frame. Under the previously stated assumptions, the matrices associated with rigid-body kinetics can be described as:

$$\boldsymbol{M}_{RB} = \begin{bmatrix} m & 0 & 0 \\ 0 & m & 0 \\ 0 & 0 & I_z \end{bmatrix} \tag{2.13}$$

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

$$\boldsymbol{C}_{RB}(\boldsymbol{\nu}) = \begin{bmatrix} 0 & -mr & 0 \\ mr & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \tag{2.14}$$

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. What this actually means is that: a body moving in a fluid behaves as if it has more mass than it actually has. This increase is termed added mass [52], sometimes is called virtual mass. This dynamics include inertia, centripetal and Coriolis matrices which are derived from the Kirchoff's kinetic energy approach [53].

Since any motion of the marine craft will induce a motion in the otherwise stationary fluid, the fluid must move aside and then close behind the craft in order to let the craft pass through the fluid. As a consequence, the fluid motion gains kinetics energy that would not have otherwise.

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, $\boldsymbol{C}_A(\boldsymbol{\nu})$. According to Fossen [51], the virtual mass matrices are written as follows:

$$\boldsymbol{M}_A = \begin{bmatrix} -X_{\dot{u}} & 0 & 0 \\ 0 & -Y_{\dot{v}} & -Y_{\dot{r}} \\ 0 & -Y_{\dot{r}} & -N_{\dot{r}} \end{bmatrix} \tag{2.15}$$

$$\boldsymbol{C}_A(\boldsymbol{\nu}) = \begin{bmatrix} 0 & 0 & Y_{\dot{v}}v + Y_{\dot{r}}r \\ 0 & 0 & -X_{\dot{u}}u \\ -Y_{\dot{v}}v - Y_{\dot{r}}r & X_{\dot{u}}u & 0 \end{bmatrix} \tag{2.16}$$

According to Fossen [51], the matrices $\boldsymbol{M}$ and $\boldsymbol{C}(\boldsymbol{\nu})$ can be described from the equations (2.13), (2.14), (2.15) and (2.16) as follows:

$$\boldsymbol{M} = \boldsymbol{M}_{RB} + \boldsymbol{M}_A = \begin{bmatrix} m - X_{\dot{u}} & 0 & 0 \\ 0 & m - Y_{\dot{v}} & -Y_{\dot{r}} \\ 0 & -Y_{\dot{r}} & I_z - N_{\dot{r}} \end{bmatrix} \tag{2.17}$$

$$\boldsymbol{C}(\boldsymbol{\nu}) = \boldsymbol{C}_{RB}(\boldsymbol{\nu}) + \boldsymbol{C}_A(\boldsymbol{\nu}) = \begin{bmatrix} 0 & -mr & Y_{\dot{v}}v + Y_{\dot{r}}r \\ mr & 0 & -X_{\dot{u}}u \\ -Y_{\dot{v}}v - Y_{\dot{r}}r & X_{\dot{u}}u & 0 \end{bmatrix} \tag{2.18}$$

2.1.5 Hydrodynamic Damping Forces

Several damping effects result from the interaction between a maritime vehicle and the water. In this thesis, only linear damping and the drag force will be considered, both depending on the relative velocity between vehicle and the water. For linear damping of low-speed, it is assumed that the surge speed can be decoupled from the sway and yaw motions [51].

The different damping terms contribute to both linear and quadratic damping. However, it is difficult to separate these effects. In many cases, it is convenient to write total hydrodynamic damping as [51]:

$$\boldsymbol{D}(\boldsymbol{\nu}) = \boldsymbol{D} + \boldsymbol{D}_n(\boldsymbol{\nu}) \tag{2.19}$$

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. According to Fossen [51], these damping matrices can be written as follows:

$$\boldsymbol{D} = \begin{bmatrix} X_u & 0 & 0 \\ 0 & Y_v & Y_r \\ 0 & N_v & N_r \end{bmatrix} \tag{2.20}$$

$$\boldsymbol{D}_n(\boldsymbol{\nu}) = \begin{bmatrix} X_{u^2}|u| & 0 & 0 \\ 0 & Y_{v^2}|v| & 0 \\ 0 & 0 & N_{r^2}|r| \end{bmatrix} \tag{2.21}$$

Hereupon, the equation (2.19) can be reformulated as follows:

$$\boldsymbol{D}(\boldsymbol{\nu}) = \begin{bmatrix} X_u + X_{u^2}|u| & 0 & 0 \\ 0 & Y_v + Y_{v^2}|v| & Y_r \\ 0 & N_v & N_r + N_{r^2}|r| \end{bmatrix} \tag{2.22}$$

2.1.6 Engine Thrust

The boat model is equipped with a fixed-axis propeller of variable angle direction ([-30º, 30º]) and propulsion magnitude ([-50N, 50N]), as shown in the Figure 2.3. Therefore, the thrust that propels the boat is modeled according to:

$$\boldsymbol{\tau} = \begin{bmatrix} P\cos\phi \\ -P\sin\phi \\ aP\sin\phi \end{bmatrix} \tag{2.23}$$

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].

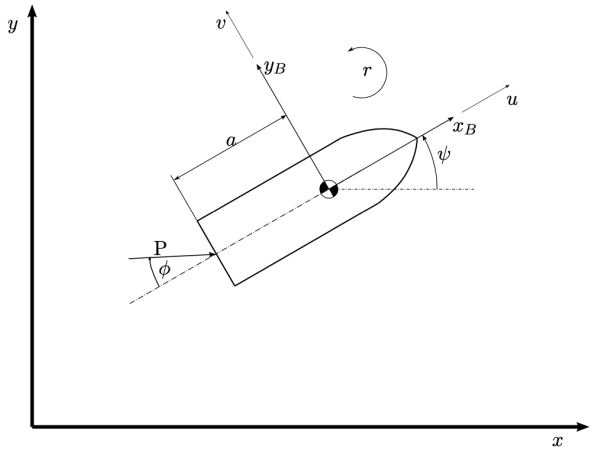

Figure 2.3: Schematic of the considered boat model

2.2 Reinforcement Learning

Machine Learning (ML) can be described as branch within artificial intelligence that deals with a computers ability to improve over time, and learn without being explicitly programmed [17]. ML can further be categorized into four majorly recognized sub fields:

- **Supervised Learning** - these models learn from the labeled dataset and then are used to predict future events. For the training procedure, the input is a known training dataset with its corresponding labels, and the learning algorithm produces an inferred function to finally make predictions about some new unseen observations that one can give to the model. The model is able to provide targets for any new input after sufficient training. Some examples of models that belong to this family are the following: SVC [54], LDA [55], SVR [56], random forests [57], etc.

- **Unsupervised Learning** - this field studies how systems can infer a function to describe a hidden structure from unlabeled data. The system does not predict the right output, but instead, it explores the data and can draw inferences from datasets to describe hidden structures from unlabeled data. Some examples of models that belong to this family are the following: PCA [58], K-means [59], DBSCAN [60], etc.

- **Semi-Supervised Learning** - This family is categorized between the supervised and unsupervised learning families. The semi-supervised models use both labeled and unlabeled data for training.

- **Reinforcement Learning** - This family of models consists of algorithms that use the estimated errors as rewards or penalties. If the error is big, then the penalty is high and the reward low. If the error is small, then the penalty is low and the reward high. This family of models allows the

automatic determination of the ideal behavior within a specific context in order to maximize the desired performance. Reward feedback is required for the model to learn which action is the best. An example of a model that belong to this family is the Q-learning [61]. This research will focus on this field.

2.2.1 Agent-Environment Interactions

Reinforcement learning (RL) is currently one of the most active and fast developing sub fields of ML [62]. In recent years, it has been successfully applied to solve large scale real world, complex decision making problems, including playing board games such as Go (AlphaGo/AlphaGo Zero) [9] [10], achieving human-level performance in video games [8], and driving autonomously [63] [64]. An RL agent does not pre-specify a structural model, instead, gradually learns the best strategies based on trial and error, through interactions with the environment and receiving feedback from those interactions, in order to improve the overall performance.

This technique presents a way to solve Markov Decision Process (MDP) problems built on the core idea of taking advantage of newly gathered information from a previously unknown environment [65]. The approach of RL is teaching an agent the mapping connection between the states and the actions with the aim of maximizing a reward.

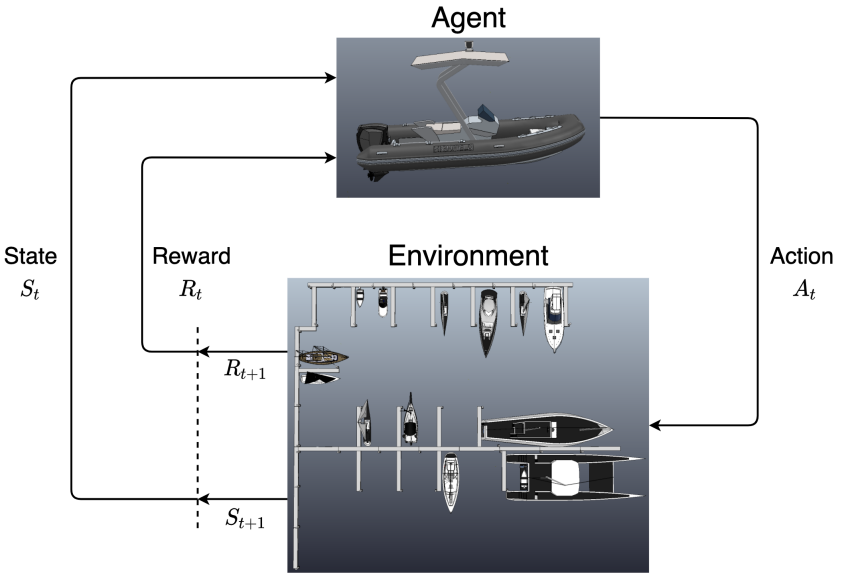

Figure 2.4: Standard RL Model

Some key terms that describe the elements of a RL problem are:

- **Environment:** Physical world in which the agent operates
- **State:** Current situation of the agent
- **Reward:** Feedback from the environment
- **Policy:** Method to map agent's state to actions

- **Value:** Future reward that an agent would receive by taking an action in a particular state

Figure 2.4 represents the basic idea and elements involved in a reinforcement learning model. On each step, information about the current state S_t is fed to the agent as input. The agent then, based on it's current behavior, or so called *policy* π, decides on an action A_t to return as output. The change that specific action makes to the state of the environment is returned to the agent as a scalar reinforcement signal, i.e reward R_t. The objective of the process is for the agent's behaviour to choose actions that maximize the sum of rewards.

Unlike the other fields of ML such as supervised learning, where a batch of training data is available for the agent, RL methods depend on gathering this data in the training process. The agent must learn its policy through trial-and-error interactions with the environment to find optimal solutions. In order to enable a system to have these algorithms applied, it is required to model the states S_t, the actions A_t, and the rewards R_t. This is commonly done using an MDP, where RL techniques are assumed to be in.

2.2.2 Markov Decision Process

A Markov Decision Process (MDP) can be defined a as non deterministic graph representation of the environment [66]. MDPs are designed to be a straightforward framing of the problem of learning from interaction to achieve a goal [67]. More specifically, the agent and the environment interact at each of a sequence of discrete time steps, $t = 0, 1, 2, 3, ...$ At each time step t, the agent receives some representation of the environments state S_t, and for that state selects an actions A_t. One time step later, as a consequence of his action, it receives a reward R_{t+1}, and transits to a new state S_{t+1}. This interaction give rise to a sequence that begins like this:

$$S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2, R_3, ... \tag{2.24}$$

In a MDP, the sets of states, actions, and rewards ($\mathcal{S}$, $\mathcal{A}$, and $\mathcal{R}$) all have finite number of elements. In this case, the random variables R_t and S_t have well defined discrete probability distributions dependent only on the preceding state and action. That is, for particular values of these random variables, $s' \in \mathcal{S}$ and $r \in \mathcal{R}$, there is a probability of those values occurring at time t, given particular values of the preceding state and action:

$$p(s', r|s, a) \doteq P[S_t = s', R_t = r|S_{t-1} = s, A_{t-1} = a] \tag{2.25}$$

for all s', $s \in \mathcal{S}$, $r \in \mathcal{R}$, and $a \in \mathcal{A}$. The function p completely characterizes the environment's dynamics [67]. That is, the probability of each possible value for S_t and R_t depends only on the previous state and action, S_{t-1} and A_{t-1}. This is a restriction not on the decision process, but on the state, which must include information about all past interactions between the agent and the environment that make a difference for the future. The state is known to have the *Markov property* if the previous condition is true.

An MDP can be represented as a directed graph (Figure 2.5), where the vertices are made up of the states space $\mathcal{S}$ and the set of actions available at each particular state and time.

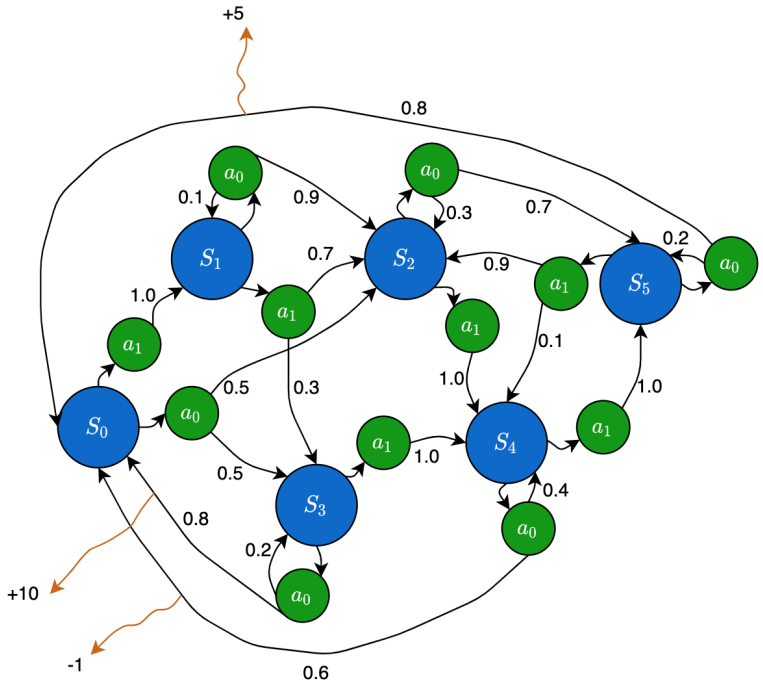

Figure 2.5: Example of an MDP with state space $S = \{s_0, s_1, s_2, s_3, s_4, s_5\}$ and action space $A = \{a_0, a_1\}$. Orange arrows show examples of returned reward for that specific action.

Executing an action a at time t causes a transition of the system from a state s to another state s' and receives an immediate reward R_t based on this transition:

$$r(s, a) \doteq \mathbb{E}[R_t | S_{t-1} = s, A_{t-1} = a] \tag{2.26}$$

In order to choose actions a policy π is defined as a probability distribution over actions given states, which defines the behaviour of the agent:

$$\pi(a|s) = P[a_t = a | s_t = s] \tag{2.27}$$

Formally, a policy maps states to probabilities of selecting each possible action. If an agent is following a policy π at time t, $\pi(a|s)$ is the probability that $A_t = a$ if $S_t = s$.

Hereupon, the objective is to develop a policy that will maximize the cumulative reward, or expected return, the agents receives. In the simplest case the return G_t is the sum of the rewards:

$$G_t = R_{t+1} + R_{t+2} + R_{t+3} + ... + R_T \tag{2.28}$$

where R_{t+1}, R_{t+2}, R_{t+3}, ..., are the rewards received after time step t, and T is the final time step.

The equation (2.28) can be problematic for continuing tasks, such as on-going process-control tasks for robots with a long life span, because, for continuous tasks, the final time step would be $T = \infty$, and

the return, which is what it set up to be maximized, could easily be infinite.

The concept that solves this issue is called the *discounted return*. The agent, instead, tries to select actions that maximize the discounted rewards that it receives over the future:

$$G_t \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ... = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}, \tag{2.29}$$

where γ is a parameter, $0 \leq \gamma \leq 1$, called the *discount rate*.

The discount rate determines the present value of future rewards. If $\gamma = 0$, the agent focus only on maximizing immediate rewards: its objective is to learn how to choose A_t so as to maximize only R_{t+1}. As γ approaches 1, the agent takes future rewards into account more strongly.

Returns at successive time steps are related to each other in a way that is important for the theory and algorithms of reinforcement learning:

$$\begin{aligned} G_t &= R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + ... \\ &= R_{t+1} + \gamma(R_{t+2} + \gamma R_{t+3} + \gamma^2 R_{t+4} + ...) \\ &= R_{t+1} + \gamma G_{t+1} \end{aligned} \tag{2.30}$$

2.2.3 Policies and Value Functions

Almost all reinforcement learning algorithms involve estimating *value functions* [67] - functions of states (or state-action pairs) that estimate how good it is for the agent to be in a given state (or how good it is to perform a given action in a given state). This notion of "how good" defines the expected future rewards.

Value functions are defined with respect to policies. The *value function* $v_\pi(s)$ of a state s under a policy π is the expected return when starting in s and following π. For MDPs, v_π is defined by

$$v_\pi(s) \doteq \mathbb{E}_\pi[G_t|S_t = s] = \mathbb{E}_\pi[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}|S_t = s] \tag{2.31}$$

where $\mathbb{E}_\pi[.]$ is the expected value of a random variable given that the agent follows the policy π, and t is any time step. The function v_π is called the *state-value function for policy* π.

Similarly, the value of taking an action a in the state s under policy π is also defined as $q_\pi(s, a)$:

$$q_\pi(s, a) \doteq \mathbb{E}_\pi[G_t|S_t = s, A_t = a] = \mathbb{E}_\pi[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}|S_t = s, A_t = a] \tag{2.32}$$

The function q_π is called the *action-value function for policy* π.

A fundamental property of value functions used in reinforcement learning is that they satisfy recursive relationships similar to the one demonstrated in equation (2.30). According to Sutton and Barto [67], for any policy π and any state s, the value of s and the value of its possible successor states hold a condition given by:

$$
\begin{aligned}
v_\pi(s) &\doteq \mathbb{E}_\pi[G_t|S_t = s] \\
&= \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1}|S_t = s] \\
&= \sum_a \pi(a|s) \sum_{s'} \sum_r p(s', r|s, a)[r + \gamma \mathbb{E}_\pi[G_{t+1}|S_{t+1} = s']] \\
&= \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)[r + \gamma v_\pi(s')], \text{for all } s \in \mathcal{S},
\end{aligned}
\tag{2.33}
$$

Equation (2.33) is the *Bellman equation for* v_π. It expresses a relationship between the value of a state and the values of its successor states. In the same way, the *Bellman equation for* q_π can be expressed as:

$$
\begin{aligned}
q_\pi(s, a) &\doteq \mathbb{E}_\pi[G_t|S_t = s, A_t = a] \\
&= \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1}|S_t = s, A_t = a] \\
&= \sum_{s'} \sum_r p(s', r|s, a)[r + \gamma \mathbb{E}_\pi[G_{t+1}|S_{t+1} = s', A_{t+1} = a']] \\
&= \sum_{s',r} p(s', r|s, a)[r + \gamma q_\pi(s', a')], \text{for all } s \in \mathcal{S},
\end{aligned}
\tag{2.34}
$$

2.2.4 Optimal Policies and Optimal Value Functions

Solving a RL task means finding a policy that achieves a lot of reward over the long run. A policy π is defined to be better than or equal to a policy π' if its expected return is greater than or equal to that of π' for all states. There is always at least one policy that is better than or equal to all other policies [67], called the *optimal policy* π_*. The same state-value function is shared between them, called the *optimal state-value function* v_*, and it is defined as

$$
v_*(s) = \max_\pi v_\pi(s), \tag{2.35}
$$

for all $s \in \mathcal{S}$.

In the same way, optimal policies also share the same *optimal action-value function* q_*, defined as

$$
q_*(s, a) = \max_\pi q_\pi(s, a), \tag{2.36}
$$

for all $s \in \mathcal{S}$ and $a \in \mathcal{A}(s)$. For the state-action pair (s, a), this function outputs the expected return for taking an action a in state s following an optimal policy π_*. Hereupon, the equation (2.36) can be rewritten in terms of v_* as follows:

$$
q_*(s, a) = \mathbb{E}[R_{t+1} + \gamma v_*(S_{t+1})|S_t = s, A_t = a] \tag{2.37}
$$

The value function v_* of a policy must satisfy the condition by the Bellman equation (2.33) for state values. This condition can be rewritten without a reference to any specific policy as follows:

$$
\begin{aligned}
v_*(s) &= \max_{a \in \mathcal{A}(s)} q_{\pi_*}(s,a) \\
&= \max_a \mathbb{E}_{\pi_*}[G_t | S_t = s, A_t = a] \\
&= \max_a \mathbb{E}_{\pi_*}[R_{t+1} + \gamma G_{t+1} | S_t = s, A_t = a] \\
&= \max_a \mathbb{E}[R_{t+1} + \gamma v_*(S_{t+1}) | S_t = s, A_t = a] \\
&= \max_a \sum_{s',r} p(s', r | s, a)[r + \gamma v_*(s')]
\end{aligned}
\tag{2.38}
$$

This is the Bellman equation for v_*, or the *Bellman optimality equation*. This equation expresses the fact that the value of a state under an optimal policy must equal to the expected return for the best action from that state.

In the same way, the *Bellman optimality equation* for q_* can be expressed as follows:

$$
\begin{aligned}
q_*(s,a) &= \mathbb{E}[R_{t+1} + \gamma \max_{a'} q_*(S_{t+1}, a') | S_t = s, A_t = a] \\
&= \sum_{s',r} p(s', r | s, a)[r + \gamma \max_{a'} q_*(s', a')]
\end{aligned}
\tag{2.39}
$$

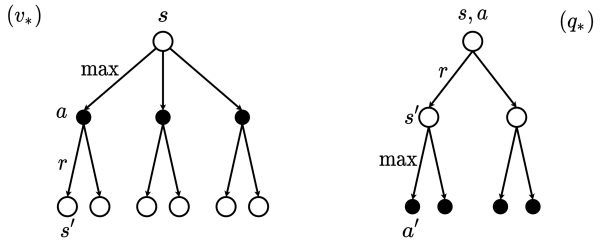

Figure 2.6: v_* and q_* diagrams

The diagrams in the Figure 2.6 show the spans of future states and actions considered in the Bellman optimality equations for v_* and q_*. The diagram on the left represents the Bellman optimality equation (2.38) and the diagram on the right represents the Bellman optimality equation (2.39).

2.2.5 Model-based vs. Model-free

RL techniques are categorized into two different classes: model-free and model-based [68]. Model-based approaches assume that the agent has knowledge of the environment. That is, the agent knows how the executed actions affect the current state. Through this the agent attempts to learn a transition probability $P(S_{t+1}|S_t, A_t)$ based on the current state-action pair (S_t, A_t) onto the succeeding state S_{t+1}. The agent then uses this learned model to make predictions about the future states as well as future rewards.

On the contrary, model-free approaches do not have knowledge of the environment, and instead, rely only on trial and error to update its knowledge. These approaches do not require any previous

information of the model because the agent estimates the policy itself.

2.2.6 Reward Function

Unlike supervised learning, which uses labeled data to measure the performance of the model, RL methods require a different way to measure the performance of the agent's actions. This measure is done by the reward function, which is used as a tool to guide the agent toward the desired behaviour [69]. The agent's goal is to maximize this reward.

RL agents can often find new strategies to collect rewards from the environment. This can be a clear strength, but it can also lead to undesirable and dangerous solutions. This is why it is so important to be careful designing the reward function [67].

For simple problems with a small number of steps it might be enough to return a single reward when the agent reaches a terminal state S_T. Performance can then be improved through backpropagation of this delayed reward [17]. However, for problems with many steps this might result in extensive process and problems in optimal convergence. In such cases a continuously collected reward can be applied in order to facilitate the learning process. This approach often can be a bad idea once it originates local optimums where the agent finds ways to optimize these sub goals while missing the real intention [70]. Such problems can also occur due to conflicting goals [71].

As mentioned before, rewards can backpropagate and lead to convergence of the learning process. However, there is a lot of uncertainty present in the future rewards. To account for this, the reward function is discounted by a factor $0 \leq \gamma^t \leq 1$, where t is the number of time steps into the future. This makes the total returned reward at time t defined as

$$G_t \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ... = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}, \tag{2.40}$$

as mentioned in section 2.2.2.

2.2.7 Exploration vs Exploitation

Unlike in other kinds of learning, one of the challenges that arise in RL is the trade-off between *exploration* and *exploitation*. To obtain a lot of reward, the agent must prefer actions that it has tried in the past and showed to be effective in maximizing it. However, in order to discover such actions, it has to try actions that it has not selected before. In summary, the agent has to exploit what is has already experienced in order to obtain reward, but it also has to explore in order to select better actions in the future [67]. The agent must try a wide range of actions and gradually favor those that appear to be the best.

At any time step t, if estimates of the action values are maintained, there is at least one action whose estimated value is greatest. These actions are called *greedy* actions. When one of these actions are selected, the agent is said to be *exploiting* his current knowledge of the action values. If instead, one of the nongreedy actions are selected, then the agent is said to be *exploring*, because it enables him to

improve the estimate of the nongreedy action's value. Exploitation is the right thing to do to maximize the expected reward on the one step, but exploration may produce the greater total reward in the long run [67].

The simplest action selection rule is to select one of the actions with the highest estimated value, that is, one of the greedy actions as defined in the last paragraph. This *greedy* action selection method as

$$A_t \doteq \arg\max_a Q_t(a), \tag{2.41}$$

where $\arg\max_a$ denotes the action a for which the expression that follows is maximized. Greedy action selection always exploits current knowledge to maximize immediate reward; it does not take into account other inferior actions to see if they might be better in the long run. A simple alternative to this is to select a greedy action most of the time, but occasionally, with small probability ϵ, select a random action instead. This method is called ϵ-*greedy*. An advantage of these methods is that, as the experience increases, every action will be sampled an infinite amount of times, ensuring that all the $Q_t(a)$ converge to q_* [67].

A popular approach for this exploration-exploitation approach trade-off is to initialize ϵ with higher values and decrease it as training evolves. This approach enables the agent to do a lot of exploration in the beginning and, as training progresses, exploration decreases and the agent exploits the most promising actions.

2.2.8 Temporal-Difference Learning

Temporal-difference (TD) learning is a combination of Monte Carlo (MC) and dynamic programming (DP) techniques. Like MC methods, TD methods can learn directly from raw experience without a model of the environment's dynamics. Like DP, TD methods updates estimates based in part on other learned estimates, not learning by the difference of the final outcomes of the process, but instead evaluating the differences between each update step (bootstrapping).

Both TD and MC methods use experience to solve the prediction problem [67]. Both methods update their estimate V for the nonterminal states S_t, given some experience following a policy π. MC methods wait until the return following the visit is known and then use that return as a target for $V(S_t)$. This is written as

$$V(S_t) \leftarrow V(S_t) + \alpha[G_t - V(S_t)] \tag{2.42}$$

where G_t is the actual return following time t, and α is a constant step-size parameter. Although MC methods must wait until the end of the episode to determine the increment to $V(S_t)$ (only then is G_t known), TD methods need to wait only until the next time step. At time $t+1$ a target is immediately formed and made an update using the observed R_{t+1} and the estimate V_{t+1}. The simplest TD method update is written as follows:

$$V(S_t) \leftarrow V(S_t) + \alpha[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)]. \tag{2.43}$$

This update is made immediately on transition to S_{t+1} and receiving R_{t+1}. In summary, the target for the MC update is G_t, whereas the target for the TD update is $R_{t+1} + \gamma V(S_{t+1})$. This TD method is called $TD(0)$. The Algorithm 1 shows the pseudo-code for the TD(0).

Algorithm 1 *one-step TD* for estimating v_π

Input: the policy π to be evaluated
Algorithm parameter: step size $\alpha \in [0, 1]$
Initialize $V(s)$, for all $s \in \mathcal{S}^+$, arbitrarily except that V(*terminal*) $= 0$

Loop for each episode:
    Initialize S
    Loop for each step of episode:
        $A \leftarrow$ action given by π for S
        Take action A, observe R, S'
        $V(s) \leftarrow V(s) + \alpha[R + \gamma V(S') - V(S)]$
        $S \leftarrow S'$
    until S is terminal

Note that the quantity in brackets in (2.43) is an error, computing the difference between the estimated value $V(S_t)$ and the better estimate $R_{t+1} + \gamma V(S_{t+1})$. This difference is called the *TD error*:

$$\delta_t \doteq R_{t+1} + \gamma V(S_{t+1}) - V(S_t) \tag{2.44}$$

where δ_t is the error in $V(S_t)$, available at time $t + 1$.

The MC error can be written as a sum of TD errors:

$$\begin{aligned} G_t - V(S_t) &= R_{t+1} + \gamma G_{t+1} - V(S_t) + \gamma V(S_{t+1}) - \gamma V(S_{t+1}) \\ &= \delta_t + \gamma(G_{t+1} - V(S_{t+1})) \\ &= \delta_t + \gamma\delta_{t+1} + \gamma^2(G_{t+2} - V(S_{t+2})) \\ &= \delta_t + \gamma\delta_{t+1} + \gamma^2\delta_{t+2} + ... + \gamma^{T-t-1}\delta_{T-1} + \gamma^{T-t}(G_T - V(S_T)) \\ &= \delta_t + \gamma\delta_{t+1} + \gamma^2\delta_{t+2} + ... + \gamma^{T-t-1}\delta_{T-1} + \gamma^{T-t}(0 - 0) \\ &= \sum_{k=t}^{T-1} = \gamma^{k-t}\delta_k. \end{aligned} \tag{2.45}$$

In summary, TD methods have an advantage over DP methods since they do not need to model the environment. TD methods also have an advantage over MC methods since the latter needs to wait until the end of an episode while the former does not need to. Another advantage of this method, presented in [67], is that it is proved to converge to v_π.

2.2.9 Q-learning

Q-learning [61] is a model-free and off-policy algorithm commonly used in RL problems. It provides agents with the capability of learning to act optimally in Markovian domains by experiencing the consequences of actions, without requiring them to build maps of the domains [61].

Learning proceeds similarly to Sutton's method [67] of TD: by letting the agent select an action given the state the immediate consequence is evaluated through basic reward/penalty feedback based on a reward function. Using this reward, combined with an initial or previously record value-estimate of the state to which it is taken, the agent instantly updates the value of the executed action in the previous state, the *Q-value*. The step-by-step diagram of this algorithm can be seen in the Figure 2.7.

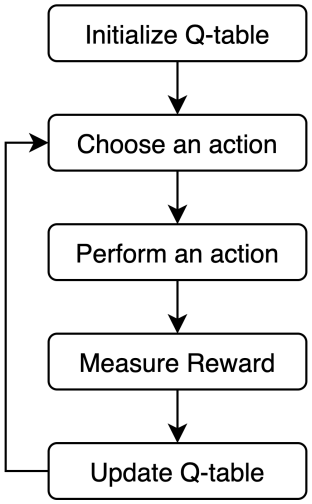

Figure 2.7: Q-learning diagram

The first step is to initialize the Q-table. This table is the brain of the agent and has size $n \times m$, where n is the number of states and m is the number of possible actions. Each cell in this table contains the Q-value for being in a certain state S_t and performing an action A_t, following a certain policy π. This is the value that is going to be updated at each step. The Q-table initialization can be seen in the Table 2.1. Since the agent does not know anything about the environment, every Q-value for each state-action pair is initialized with zero.

	A_0	A_1	A_2	A_3	A_4	A_5	...	A_m
S_0	0	0	0	0	0	0	...	0
S_1	0	0	0	0	0	0	...	0
S_2	0	0	0	0	0	0	...	0
S_3	0	0	0	0	0	0	...	0
S_4	0	0	0	0	0	0	...	0
S_5	0	0	0	0	0	0	...	0
...	...	...	...	...	...	...	...	...
S_n	0	0	0	0	0	0	...	0

Table 2.1: Q-table initialization

The second step is to choose an action and perform it. The ϵ-*greedy* strategy comes in to play here. In the beginning, the ϵ rates will be higher. The agent will explore the environment and randomly choose actions. As the agent explores the environment, the epsilon rate decreases and the agent starts to exploit the environment.

By performing an action A_t in a given state S_t there is a transition for another state S_{t+1}. That transition is evaluated by the reward R_t. Having the next state and the reward it is now possible to compute the target Q-value $Q^*(S_t, A_t)$ using the Bellman Equation, as seen in the section 2.2.4:

$$Q^*(S_t, A_t) = R_t + \gamma \max_{A_{t+1}} Q(S_{t+1}, A_{t+1}), \tag{2.46}$$

where $\max_{a_{t+1}} Q(S_{t+1}, A_{t+1})$ is the maximum expected future reward, that is, the optimal Q-Value for the state the agent end up in.

This target value is the equivalent of a "ground truth" in supervised learning. This is the value that the agent is trying to predict for that state-action pair.

The last step is the Q-table update. This update is made by the following equation:

$$Q^{new}(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_t + \gamma \max_{A_{t+1}} Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)] \tag{2.47}$$

where,

- $Q^{new}(S_t, A_t)$, is the new Q-value for that state-action pair
- $Q(S_t, A_t)$, is the current Q-value
- α, is the learning rate. The learning rate or step size determines to what extent newly acquired information overrides old information. A factor of 0 makes the agent learn nothing (exclusively exploiting prior knowledge), while a factor of 1 makes the agent consider only the most recent information (ignoring prior knowledge to explore possibilities).
- $R_t + \gamma \max_{A_{t+1}} Q(S_{t+1}, A_{t+1})$, is the target Q-value, as previously stated.

All the steps, except the Q-table initialization, are repeated until the end of training. After that the Q-table is updated as represented by the example in the Table 2.2.

	A_0	A_1	A_2	A_3	A_4	A_5	...	A_m
S_0	0.4	1.3	-1.7	7.0	4.2	-3.3	...	14.3
S_1	21.3	-3.2	-0.2	-32.4	0.9	-0.7	...	3.4
S_2	18.1	24.9	-10.0	0.3	7.2	4.9	...	0.4
S_3	11.4	5.3	-1.9	49.2	-3.2	-1.9	...	16.7
S_4	-0.4	37.8	22.2	-3.0	1.8	45.2	...	-10.6
S_5	29.3	-32.2	-10.2	3.4	30.9	-10.8	...	-3.1
...	...	...	...	...	...	...	...	...
S_n	1.3	-26.4	-10.2	12.4	23.7	-0.2	...	13.4

Table 2.2: Example of Q-table after training

The pseudo-code for the Q-learning algorithm can be seen in the Algorithm 2.

Algorithm 2 Q-learning

Algorithm parameters: step size $\alpha \in [0, 1]$, small $\epsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+$, $a \in \mathcal{A}(s)$, arbitrarily except that $Q(\textit{terminal}, .) = 0$

Loop for each episode:
    Initialize S
    Loop for each step of episode:
        Choose A from S using policy derived from Q (e.g., ϵ-*greedy*)
        Take action A, observe R, S'
        $Q(S, A) \leftarrow Q(S, A) + \alpha[R + \gamma \max_a Q(S', a) - Q(S, A)]$
        $S \leftarrow S'$
    until S is terminal

The general Q-learning has been shown to converge to the optimal action-value function with probability 1 given two conditions: all action values are represented discretely, and these actions are repeatedly sampled in all states [61].

2.3 Deep Reinforcement Learning

Q-learning is a simple yet powerful algorithm that gives the agent the capability to figure out exactly which action to perform with the help of a table. But what happens when the environment is too complex and this table becomes too long? Lets imagine an environment with 10,000 states and 1,000 actions per state. This would create a table of 10 million cells. This would create two problems:

- First, the amount of memory required to save and update that table would increase as the number of states increases;
- Second, the amount of time required to explore each state to create the required Q-table would be unrealistic.

The emergence of these two problems led to the replacement of this table by a neural network. This method of approximating the Q-values with machine learning models was first introduced by DeepMind in [7] and [8].

2.3.1 Deep Q-Network (DQN)

In Deep Q-learning, a neural network is used to approximate the Q-value function. The state is given as the input and the Q-values for every possible action are given as the output. The differences between Q-learning and Deep Q-learning are illustrated in the Figure 2.8.

A Q-Network can be trained by minimising a sequence of loss function $L_i(\theta_i)$ that changes at each iteration i,

$$L_i(\theta_i) = \mathbb{E}_{s,a\sim p(.)}[(y_i - Q(s, a; \theta_i))^2], \tag{2.48}$$

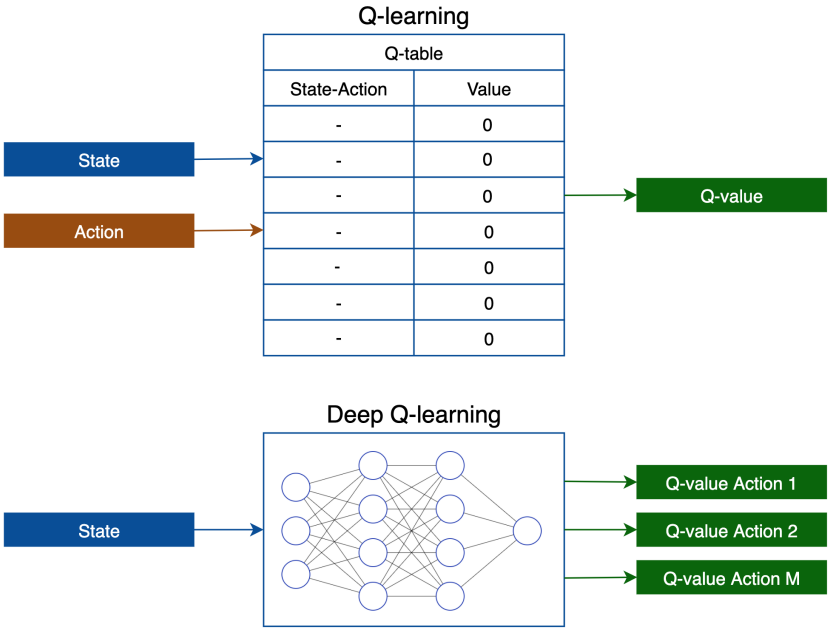

Figure 2.8: Differences between Q-learning and Deep Q-learning

where,

$$y_i = \mathbb{E}_{s' \sim \mathcal{E}}[r + \gamma \max_{a'} Q(s', a'; \theta_{i-1}) | s, a], \tag{2.49}$$

is the target for iteration i. The parameters from the iteration θ_{i-1} are held fixed when optimising the loss function. Note that the targets depend on the network weights; this is in contrast with the targets used for supervised learning, which are fixed before learning begins. Differentiating the loss function with respect to the weights, the gradient is written as follows:

$$\nabla_{\theta_i} L_i(\theta_i) = \mathbb{E}_{s,a \sim p(); s' \sim \mathcal{E}}[(r + \gamma \max_{a'} Q(s', a'; \theta_{i-1}) - Q(s, a; \theta_i)) \nabla_{\theta_i} Q(s, a; \theta_i)]. \tag{2.50}$$

Algorithm 3 Deep Q-learning with Experience Replay

Initialize replay memory $\mathcal{D}$ to capacity N
Initialize action-value function Q with random weights
for episode $= 1, M$ **do**
    Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
    for $t = 1, T$ **do**
        With probability ϵ select a random action a_t
        otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
        Execute action a_t in emulator and observe reward r_t and image x_{t+1}
        Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
        Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $\mathcal{D}$
        Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$
        Set $y_i = \begin{cases} r_j, & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta), & \text{for non-terminal } \phi_{j+1} \end{cases}$
        Perform a gradient descent step on $(y_i - Q(\phi_j, a_j; \theta))^2$ according to equation 2.50
    end for
end for

The full algorithm for training deep Q-networks is presented in Algorithm 3. The algorithm is different from standard online Q-learning in two ways to make it suitable for training large neural networks without diverging. First, it is used a technique known as experience replay [72] in which it is stored the agent's experiences at each time-step, $e_t. = (s_t, a_t, r_t, s_{t+1})$, in a buffer $D_t = \{e_1, ..., e_t\}$ pooled over many episodes (where the end of an episode occurs when a terminal state is reached) into a replay memory. During the inner loop of the algorithm, it is applied Q-learning updates, or minibatch updates, to samples of experience, $(s, a, r, s') \sim U(D)$, drawn at random from the pool of stored samples. This method has various advantages over standard online Q-learning:

1. Each step of experience is potentially used in many weight updates, which allows for greater data efficiency.

2. Learning directly from consecutive samples is inefficient, due to the strong correlations between the samples; choosing random samples breaks these correlations.

3. When learning on-policy the current parameters determine the next data sample that the parameters are trained on. For example, if the maximizing action is to move left then the training samples will be dominated by samples from the left-hand side; if the maximizing action then switches to the right then the training distribution will also switch. It is easy to see how parameters could get stuck in a poor local minimum, or even diverge [73]. By using experience replay the behaviour distribution is averaged over many of its previous states, smoothing out learning and avoiding oscillations or diverge in the parameters. Note that when learning by experience replay, it is necessary to learn off-policy, because the current parameters are different to those used to generate the sample, which motivates the choice of Q-learning.

The second modification to online Q-learning aimed to improve stability of the method with neural networks is to use a separate network for generating the targets y_i in the Q-learning update. More precisely, every C updates the network Q is cloned to obtain a target network $\hat{Q}$ and use this target network for generating the Q-learning targets y_i for the following C updates to Q. This modification makes the algorithm more stable compared to standard online Q-learning, where an update that increases $Q(s_t, a_t)$ often also increases $Q(s_{t+1}, a)$ for all a and hence also increases the target y_i, leading to oscillations or diverge of the policy. Generating the targets using an older set of weights adds a delay between the time an update to Q is made and the time the update affects the targets y_i, making divergence or oscillations much more unlikely.

2.3.2 Double Deep Q-Network (DDQN)

The max operator in standard Q-learning and DQN, in equation (2.46) and (2.49), uses the same values both to select and to evaluate an action. This makes it more likely to select overestimated values, resulting the overoptimistic value estimates [74]. To prevent this, the selection and the evaluation can be decoupled. This is the idea behind Double Q-learning [75].

Algorithm 4 Double Q-learning

Initialize Q^A, Q^B, s
repeat
  Choose a, based on $Q^A(s,\cdot)$ and $Q^B(s,\cdot)$, observe r, s'
  Choose (e.g. random) either UPDATE(A) or UPDATE(B)
  if UPDATE(A) **then**
    Define $a^* = \arg\max_a Q^A(s',a)$
    $Q^A(s,a) \leftarrow Q^A(s,s) + \alpha(s,a)(r + \gamma Q^B(s',a^*) - Q^A(s,a))$
  else if UPDATE(B) **then**
    Define $b^* = \arg\max_a Q^B(s',a)$
    $Q^B(s,a) \leftarrow Q^B(s,s) + \alpha(s,a)(r + \gamma Q^A(s',b^*) - Q^B(s,a))$
  end if
  $s \leftarrow s'$
until end

In the original Double Q-learning algorithm (Algorithm 4), two value functions are learned by assigning each experience randomly to update one of the two value functions, such that there are two sets of weights, $\boldsymbol{\theta}$ and $\boldsymbol{\theta'}$. For each update, one set of weights is used to determine the greedy policy and the other to determine its value.

For a clear comparison, by untwisting the selection and evaluation in Q-learning, its target (2.46) can be rewritten as

$$Y_t^Q = R_{t+1} + \gamma Q(S_{t+1}, \arg\max_a Q(S_{t+1}, a; \boldsymbol{\theta_t}); \boldsymbol{\theta_t}) \tag{2.51}$$

The Double Q-learning target can then be written as

$$Y_t^{DoubleQ} = R_{t+1} + \gamma Q(S_{t+1}, \arg\max_a Q(S_{t+1}, a; \boldsymbol{\theta_t}); \boldsymbol{\theta'_t}) \tag{2.52}$$

The selection of the action depend on the online weights $\boldsymbol{\theta_t}$. This means that, similar to Q-learning, the value of the greedy policy is still being estimated by the current values, as defined by $\boldsymbol{\theta_t}$. However, to evaluate the value of this policy it is used the second set of weights $\boldsymbol{\theta'_t}$. This second set of weights can be updated symmetrically by switching the roles of $\boldsymbol{\theta_t}$ and $\boldsymbol{\theta'_t}$.

2.3.3 Dueling Deep Q-Network (Dueling DQN)

Wang et al. [12] proposed a dueling architecture, which separates the representation of state values and (state-dependent) action advantages. The dueling architecture consists of two streams that represent the value and advantage functions, while sharing a common layer. The two streams are combined via a special aggregating layer to produce an estimate of the state-action value function Q as shown in Figure 2.9.

In Figure 2.9 it is clear that, instead of following the Multilayer perceptron (MLP) with a single sequence of fully-connected (FC) layers, the network uses two streams of fully connected layers. 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 Mnih et al. [8], the output of the network is a set of Q values, one for each action.

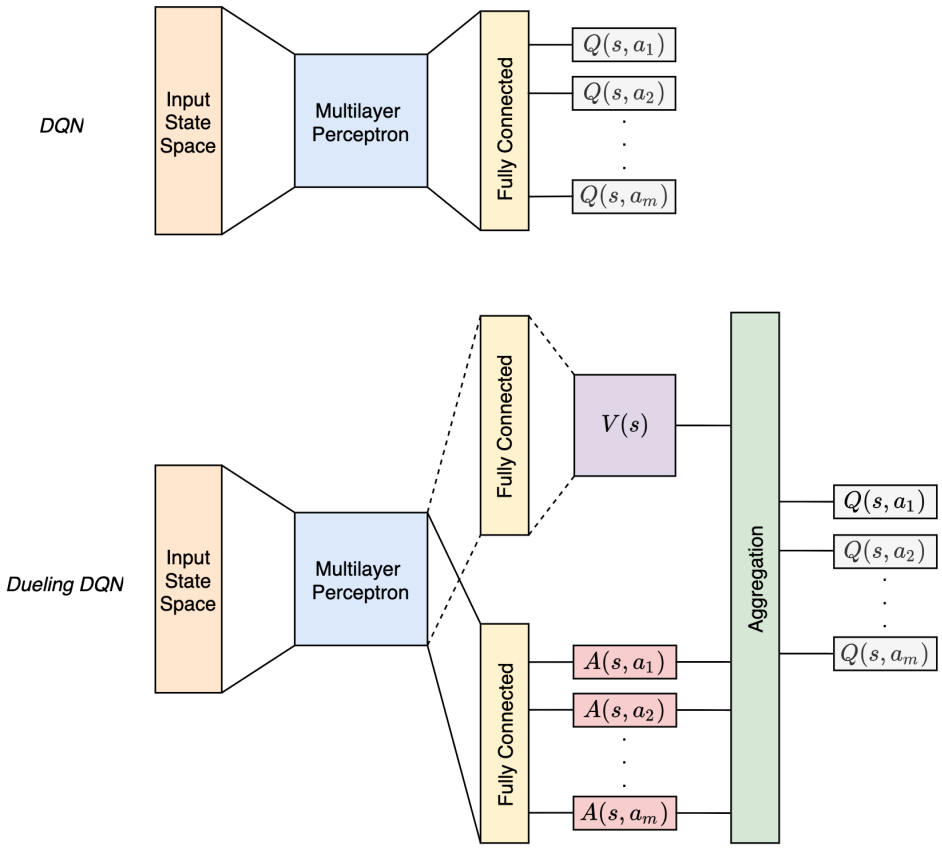

Figure 2.9: Single stream Q-network (top) and the dueling Q-network (bottom). The dueling network has two streams to separately estimate the state-value $V(s)$ and the advantages for each action $A(s, a_1), ..., A(s, a_m)$; Both networks output Q-values for each action.

In order to understand this architecture it is important to define the *advantage function*, that is represented by the red block in the Figure 2.9. This function can be written as follows:

$$A^{\pi}(s, a) = Q^{\pi}(s, a) - V^{\pi}(s), \tag{2.53}$$

where,

$$Q^{\pi}(s, a) = \mathbb{E}[R_t | s_t = s, a_t = a, \pi] \tag{2.54}$$

$$V^{\pi}(s) = \mathbb{E}_{a \sim \pi(s)}[Q^{\pi}(s, a)], \tag{2.55}$$

define the values of the state-action pair (s, a) and the state s, respectively.

Considering the network in Figure 2.9, one stream of fully-connected layers outputs a scalar $V(s; \theta, \beta)$ and the other stream outputs an $|\mathcal{A}|$-dimensional vector $A(s, a; \theta, \alpha)$. Here, θ denotes the parameters of the network, while α and β are the parameters of the two streams of fully-connected layers.

The aggregation module, shown in color green in Figure 2.9, is written as follows:

$$Q(s, a; \theta, \alpha, \beta) = V(s; \theta, \beta) + (A(s, a; \theta, \alpha) - \frac{1}{\mathcal{A}} \sum_{a'} A(s, a'; \theta, \alpha)) \tag{2.56}$$

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.

As presented in [12], the dueling architecture shares the same input-output interface with standard Q networks, which makes it is possible to use all Q networks learning algorithms, such as DDQN and SARSA, to train the dueling architecture.

In this research, the dueling architecture presented in this section will be used in combination with the DDQN algorithm [74].

Chapter 3

Implementation

As stated before, the objective of this research is to teach a small underactuated vessel to perform docking maneuvers in a simulated harbor environment 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 3.1.

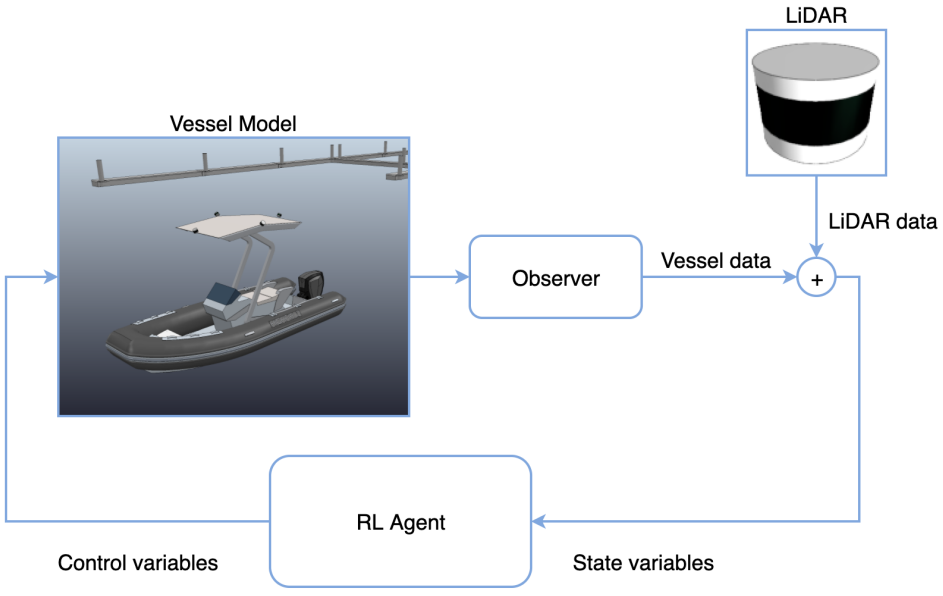

Figure 3.1: Signal flow in the closed-loop system.

In this chapter, an explanation of how the simulated environment was developed is first presented. It is then shown how the problem was formulated with RL, as well as which simplifications were assumed. In the end, the different scenarios that are going to be tested are also presented.

3.1 V-REP (CoppeliaSim) Simulator

To build the harbor environment (Figure 3.2) it was used 3D models from Sketchup 3D Warehouse. The 3D models used, as well as its references are presented in the Appendix A. This environment was built with the aim of resembling the generic harbours where the moored boats are sailing boats, small boats, yachts and catamarans. The arrangement of the docking spots was also taken into account, since the harbor structure does not generally separate the berthing place for each vessel. In this research, the harbor structure separates the docking spots two by two, as can be seen in the Figure 3.2.

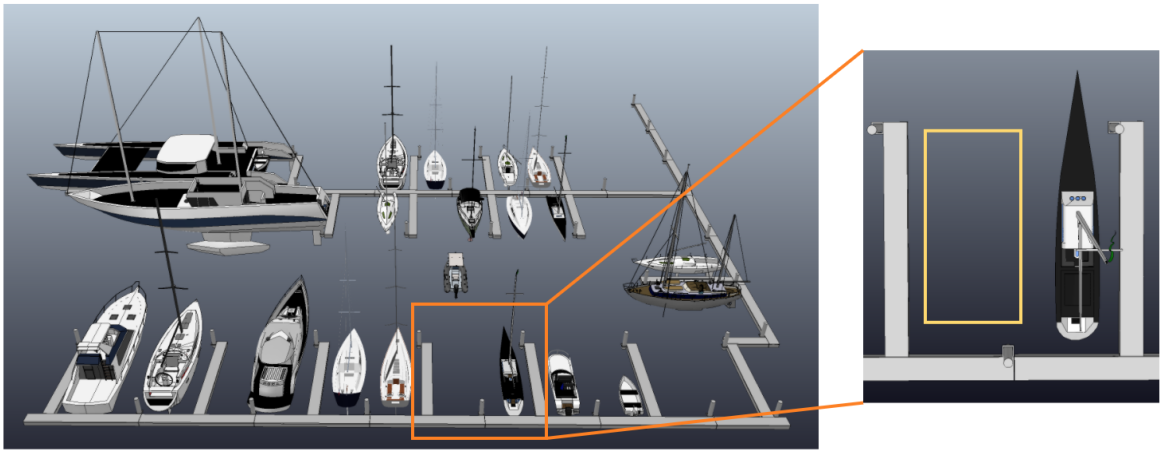

Figure 3.2: Harbor Simulation; Docking spot.

In order to simplify the problem, external disturbances, such as winds and currents, were not considered in this research.

3.1.1 Vessel system

The agent model employed in the experiments is a small vessel, whose main properties are listed in Table 3.1. Since the hydrodynamic parameters presented in the Section 2.1 must be estimated in real experiments, for this research, the parameters already estimated by Eilbrecht [76] were used.

As proposed by Eilbrecht [76], the model used in this research also controls the propeller force and the rudder angle. However, for the docking task the speeds have to be lower than in open-sea navigation. With this in mind, in this research, instead of the propeller force being limited to $P \in [-200, 200N]$, it will be limited to $P \in [-50, 50N]$ due to speed limitations inside the harbour. The limits in the rudder angle have also been changed to $\phi \in [-30^{o}, 30^{o}]$, instead of $\phi \in [-35^{o}, 35^{o}]$, so that a more simplified control discretization is possible, as will be explained further in the Section 3.2.2.

Generally, the harbor environment is very restricted and has many obstacles, with very short distances between them and the boat. 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. Information from this sensors will then be used as the state space variables, as will be further explained in Section 3.2.1.

Parameter	Value	Description
m	150.0 kg	Mass
L	3.0 m	Length
B	1.0 m	Width
T	0.3 m	Draft
T_s	0.5 s	Simulation time step
I_z	68.2 kg m^2	Moment of inertia
$X_{\dot{u}}$	-18.3 kg	Hydrodynamic parameter
$Y_{\dot{v}}$	-120.6 kg	Hydrodynamic parameter
$Y_{\dot{r}}$	0 kg m	Hydrodynamic parameter
$N_{\dot{r}}$	-31.8 kg m^2	Hydrodynamic parameter
X_{u^2}	66.0 kg/m	Hydrodynamic parameter
X_u	94.2 kg/s	Hydrodynamic parameter
Y_{v^2}	395.8 kg/m	Hydrodynamic parameter
Y_v	395.8 kg/s	Hydrodynamic parameter
N_{r^2}	245,1 kg m^2	Hydrodynamic parameter
N_r	942,5 kg m^2/s	Hydrodynamic parameter
Y_r	245,1 kg m/s	Hydrodynamic parameter
N_v	0 kg m/s	Hydrodynamic parameter
a	1.5 m	Distance between the engine and the center of gravity

Table 3.1: Parameters of the boat's dynamic system.

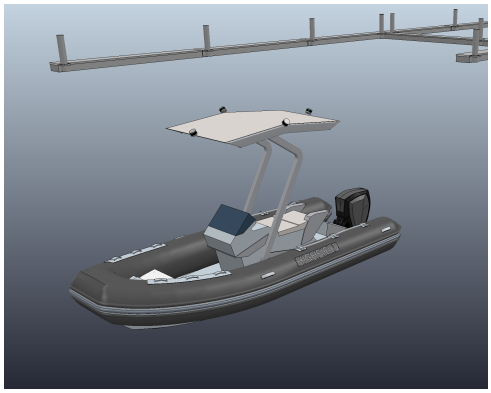

Figure 3.3: Agent.

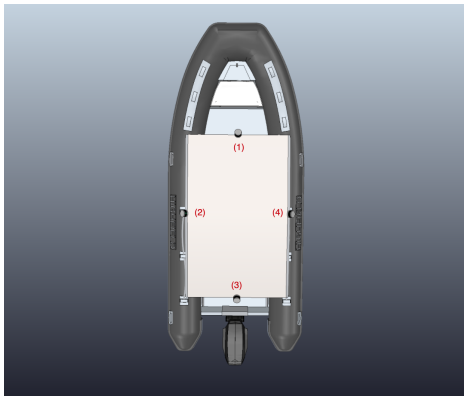

Figure 3.4: LiDAR Arrangement. Numbers 1 to 4 represent the LiDAR positions.

	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 3.2: Transformations between each LiDAR body-frame and Agent body-frame.

The LiDAR arrangement can be seen in the Figure (3.3) and (3.4), and the positions and orientations of the 4 sensors relative to the vessel's body-frame are specified in the Table 3.2, where α, β, γ are the rotations between each LiDAR body-frame and the vessel body-frame around the x-axis, y-axis and

z-axis, respectively.

The transformation between each LiDAR body-frame and the vessel body-frame is done using the homogeneous transformation matrix represented in Equation 3.1:

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix}_{Boat \leftarrow Lidar_i} = \begin{bmatrix} \mathbf{R} & \mathbf{T} \\ \mathbf{0} & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix}_{Lidar_i} \tag{3.1}$$

where,

$$\mathbf{R} = \mathbf{R}_z(\gamma)\mathbf{R}_y(\beta)\mathbf{R}_x(\alpha) = \begin{bmatrix} c_\gamma c_\beta & c_\gamma s_\beta s_\alpha - c_\alpha s_\gamma & s_\gamma s_\alpha + c_\gamma c_\alpha s_\beta \\ c_\beta s_\gamma & c_\gamma c_\alpha + s_\gamma s_\beta s_\alpha & c_\alpha s_\gamma s_\beta - c_\gamma s_\alpha \\ -s_\beta & c_\beta s_\alpha & c_\beta c_\alpha \end{bmatrix} \tag{3.2}$$

and,

$$\mathbf{T} = \begin{bmatrix} t_x \\ t_y \\ t_z \end{bmatrix} \tag{3.3}$$

are the rotation and translation matrices, respectively, between each LiDAR body-frame and the vessel body-frame, and $i = 1, ..., 4$ is the number of each LiDAR listed in Table 3.2.

3.1.2 LiDAR

LiDAR works in a similar way to Radar and Sonar yet uses light waves from a laser, instead of radio or sound waves. Its operational principle, Time-of-Flight (ToF), provides distance information by measuring the travel rime of emitted light and is described by the following equation (3.4):

$$R = \frac{c\Delta T}{2} \tag{3.4}$$

where R - range [m], c - light propagation velocity [m/s], ΔT - round trip time [s].

This sensor offers the ability to acquire massive scale 3D geometrical information of the surrounding scene using single mounted sensor system. In addition to real-time friendly acquisitions speed, 3D LiDAR is also very precise. Compared to conventional camera systems, 3D LiDAR provides lower resolution but is highly robust against unstable illumination and generally provides a larger horizontal field of view.

The LiDAR model used in this research is the Velodyne VLP-16 (Figure 3.5) and the specificifications are listed in the Table (3.3).

Specifications	Value
Channels (lasers)	16
Range	100m
Accuracy	$\pm$ 3cm
Vertical FoV	+15º to -15º
Horizontal FoV	360º
Angular Resolution	0.1º - 0.4º
Rotation Rate	5-20Hz
Power Consumption	8W

Table 3.3: Velodyne VLP-16 Specifications

Figure 3.5: Velodyne VLP-16

This particular LiDAR can fire around 300,000 pulses per second. Each of these measurements, or returns, can then be processed into a 3D visualization known as a 'point cloud'. An example of the output of this sensor, given by the harbor simulator, is shown in the Figure 3.6.

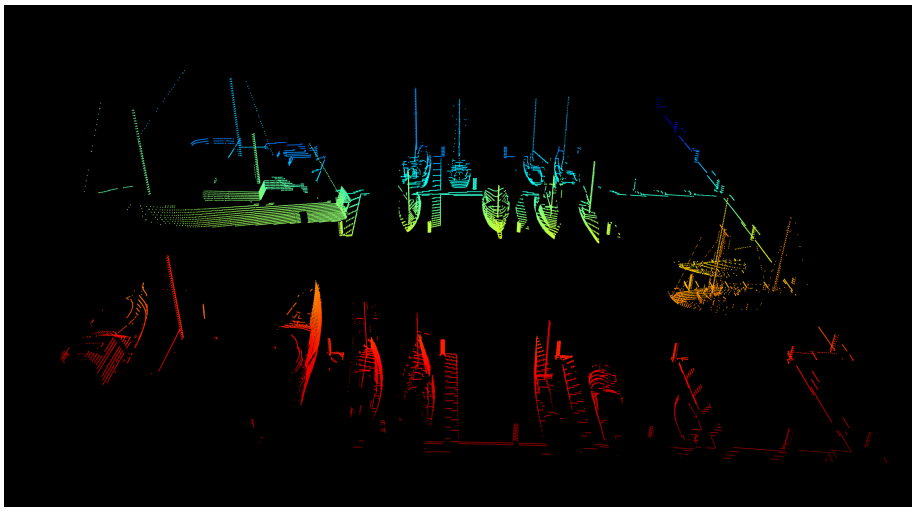

Figure 3.6: Simulator 3D Point cloud.

Further, in Section 3.2.2, the pre-processing done to the 3D point cloud from raw data to the network input format will be shown.

3.1.3 V-REP communications

V-REP framework [13] offers a set of three communication techniques to execute the control code of the simulation model:

- **The control code is executed on another machine.** This represents a distinct machine connected to the simulator machine via a specific network (e.g. socket, serial port, etc.). This allows the control code to be native and running on the original hardware while reducing the computing load on the simulation machine. However, the drawback is that imposes limitations in regards to synchronization with the simulation loop.

- **The control code is executed on the same machine, but in another process than the simulation loop.** This approach can also benefit from a reduced load on the CPU cores, but comes accompanied with the lack of synchronization, also mentioned in the last above approach.
- **The control code is executed on the same machine and in the same thread as the simulation loop.** This allows synchronization with the simulation loop and has the advantage of the absence of any execution or communication lag. However, is only possible with an increased load on the simulation loop CPU core.

In this research the control code will be executed in the same machine, but in an exterior Python script using a Remote API client, which is provided by the V-REP framework, to communicate with the simulator.

The remote API interface in V-REP allows interactions with V-REP or a simulation from an external entity via socket communication. It is composed by remote API server services and remote API clients. The client side can be embedded as a small footprint code (C/C++, Python, Java, Matlab, etc.) in virtually any hardware including real robots, and allows remote function calling, as well as fast data streaming back and forth. On the client side, functions are called almost as regular functions, with two exceptions: remote API functions accept an additional argument which is the operation mode, and return a error code. The operation mode allows calling functions as blocking (will wait until the server replies), or non-blocking (will read streamed commands from a buffer, or start/stop a streaming service on the server side). The ease of use of the remote API and its availability on all platforms makes it an interesting alternative to the ROS interface and was the main motivation to use in this research.

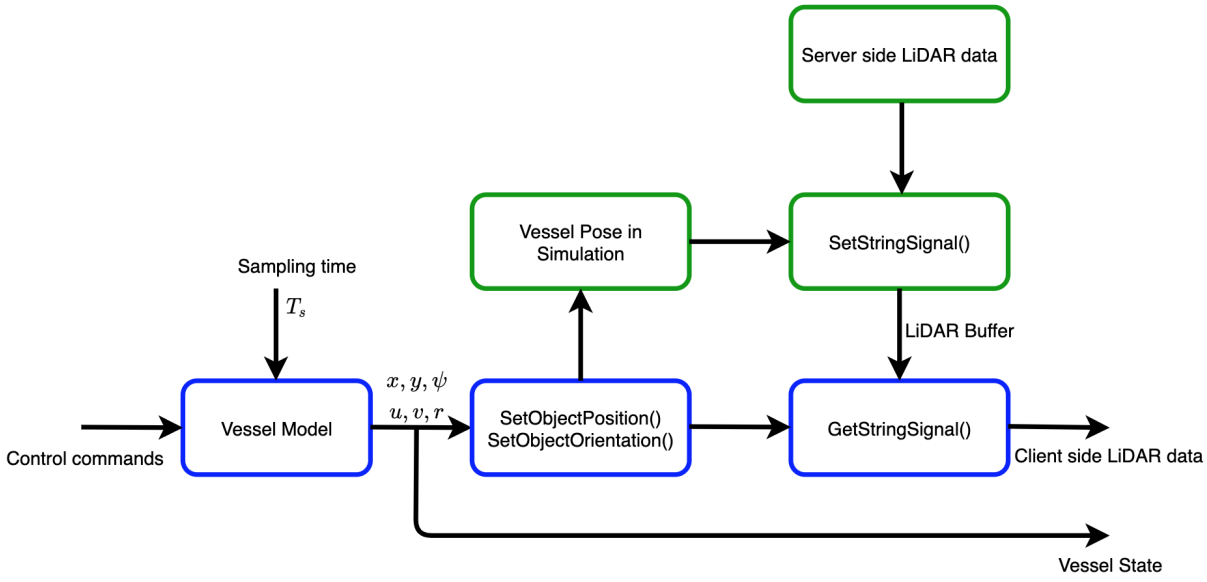

Figure 3.7: Remote API communication. The modules in green are ran in the server side and the blue modules are ran in the client server.

In order to accelerate the training process, a different approach was taken. Instead of building the response of the boat dynamic system to the control commands in the simulation, this is computed in the external script to save time. The computed vessel state (position and orientation) is then set in the

simulation using the a remote API function that changes the vessel's position and orientation without the need to wait for the next time step, as would be necessary if the response of the dynamic system to the control commands was built in the simulation. After setting the pose computed by the external script, the LiDAR data is acquired from a string buffer that is being streamed in the server side and sent to the client side. This procedure is then repeated throughout the training process in a synchronous manner so that there is no discrepancy between the LiDAR data and the vessel state. The signal flow of the communication between the external script and the simulator can be seen in the Figure 3.7. The modules in green are ran in the server side and the blue modules are ran in the client server. The client is also responsible for running the RL module that trains the neural network in order to benefit from the reduction of computational load in the simulation.

3.2 Problem Modeling as MDP

The task of the agent is to approximate the vessel to the berth position in a safe and robust manner, and to achieve this using RL it is first necessary to design the learning strategy. In this section, this learning strategy will be presented. First, the state space variables are presented, as well as the pre-processing approach for the LiDAR data. Next, since the network that will be used has a discrete action space, 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, such as exploration steps, discount factor, learning rate, etc., will be discussed.

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. As a result, the relative distance δd and relative orientation $\delta\psi$ to the dock were added to the state space. The computation of these two variables can be seen in Equation (3.5) and (3.6):

$$\delta d = \sqrt{(x_{boat} - x_{dock})^2 + (y_{boat} - y_{dock})^2} \tag{3.5}$$

$$\delta\psi = \psi_{boat} - \psi_{dock}, \tag{3.6}$$

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.

As stated before, to successfully dock a vessel it is necessary to reduce the speed near the final position in order to stop the boat in the right position, as well as avoid collisions with the harbor structures or with other parked boats. 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 a 3D point cloud pre-processing was done in order to simplify the distance information to other obstacles. First, the 3D point cloud acquired in each LiDAR is transformed to the vessel's body frame using the Equation (3.1), as explained in the Section 3.1.1. The 3D point cloud is then transformed into a 2D point cloud by simply putting the coordinate $z = 0$. The 2D Point Cloud is further segmented in sections using Python's pandas library [77]. This is done by, first, calculating the polar coordinates (ρ, θ) of the 2D point cloud (Figure 3.8), where,

$$\rho = \sqrt{x^2 + y^2}, \tag{3.7}$$

$$\theta = \text{atan2}(y, x). \tag{3.8}$$

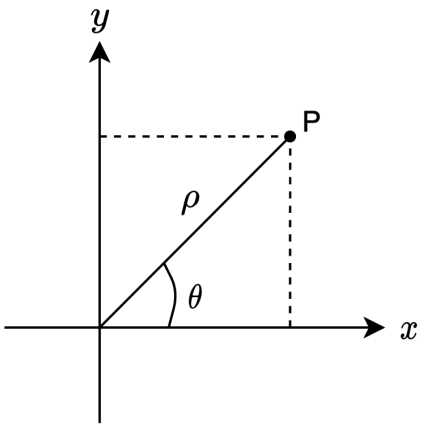

Figure 3.8: Polar coordinates.

Now that all the points are transformed in polar coordinates, it is possible to segment the point cloud according to the θ values using pandas library function $cut()$. This function has as input parameter the number of equal segments that will divide the point cloud and returns an index for each point that tells to which segment that point belongs to. After the θ segmentation the closest point to the vessel for each point cloud section is computed using k-Nearest Neighbors [78]. The distance to the closest point is then saved for every section, and these distances are 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 3.9.

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

$$s_t = [\delta d, \delta\psi, u, v, r, P_{i-1}, \phi_{i-1}, d_{Lidar_1}, ..., d_{Lidar_8}]. \tag{3.9}$$

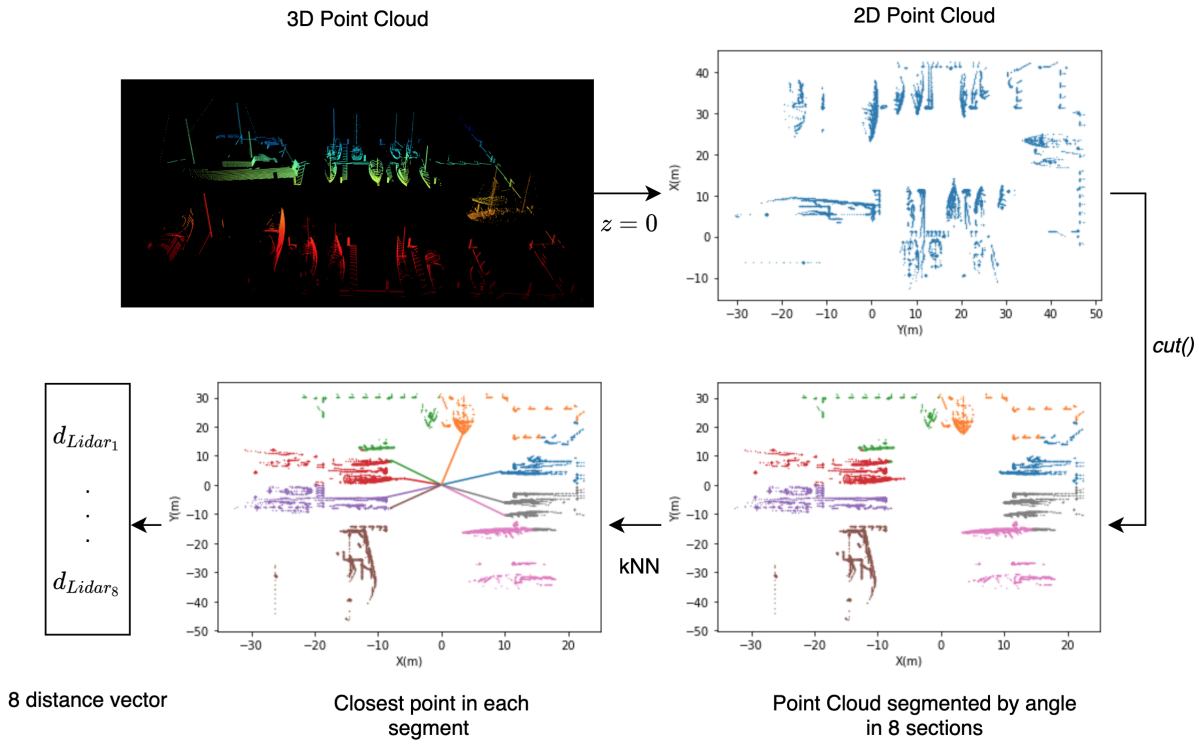

Figure 3.9: Point Cloud Pre-Processing.

3.2.2 Action Space

As stated 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 $\sim$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 so that, even if there is a collision with the harbour structure or another parked vessel, the crew is not endangered and the physical damage of the boat or structure is not catastrophic.

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, as represented in the Figure 3.10.

The rudder angle will be limited to $\phi \in [-30^o, 30^o]$. The original model from Eilbrecht [76] was limited to $\phi \in [-35^o, 35^o]$, but in order to simplify the action space, in this research it were considered different boundaries.

Since the network presented in Section 2.3.3 has a discrete action space it is necessary to discretize the control variables. The propeller force and the rudder angle were both discretized in 10N and 10º intervals, respectively, as shown in the Table 3.4.

Control Variables	Discretized Values
Propeller Force [N]	-50 -40 -30 -20 -10 0 10 20 30 40 50
Rudder Angle [º]	-30 -20 -10 0 10 20 30

Table 3.4: Control Discretization.

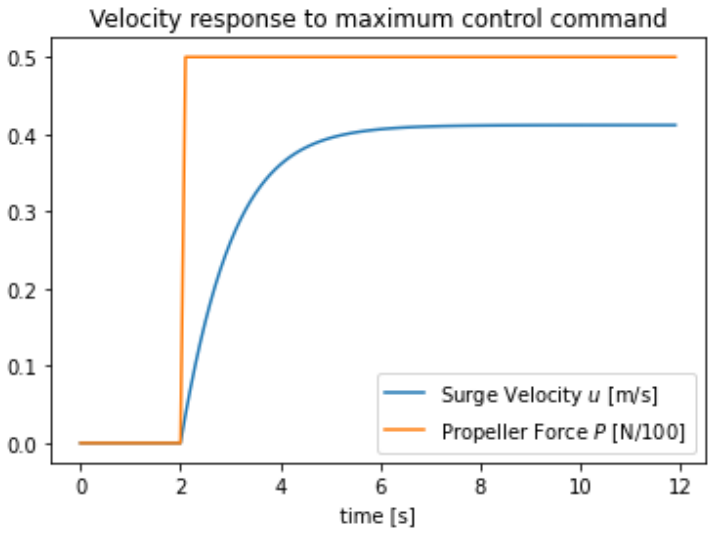

Figure 3.10: Velocity response to maximum control command.

The network only chooses one action at each step, so it is necessary to make a combination of all possible actions from each control variable. Multiplying the 11 possible actions of the propeller by the 7 possible actions of the rudder gives a total number of 77 possible actions. In Figure 3.11 it is shown the distribution of the actions in the neuronal network output vector.

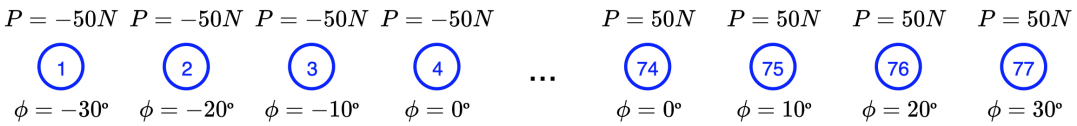

Figure 3.11: Distribution of the network output vector.

3.2.3 Reward function

The ultimate goal of RL is to maximize returned reward, as this serves as a measure of performance through training. Consequently the reward shapes the behaviour of the system and sets terms for convergence.

It is important to design the reward function to give gradual feedback and let the agent know it's getting better or worse. Two approaches can take place here: One can give positive rewards or negative rewards, which makes a huge difference in the agent's behaviour. In the former, the system will try to accumulate as much reward as possible and is encouraged to avoid terminal states unless they yield very high reward. By contrast, in the latter, the negative rewards incentivize the agent to get done as quickly as possible because it is constantly getting a penalty, so it is encouraged to reach a terminal state as quickly as possible to avoid accumulating penalties.

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 (3.10).

$$r = \begin{cases} -r_{collision}, & \text{if collision} \\ r_{final}, & \text{if final pose} \\ -\delta d * r_{distance} - |\delta\psi| * r_{orientation} - \frac{1}{\delta d} * \sqrt{u^2 + v^2} * r_{velocity}, & \text{else} \end{cases} \tag{3.10}$$

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 (3.10) as r_{final} and $r_{collision}$, respectively. The final state is reached each time the relative distance to the final position is less than 1m, the relative orientation is less than 0.07 rad and the speed is less than 0.1m/s, and the collisions are detected using the collision detection tool already available in the V-REP framework [13]. 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 (3.10). The first ($-\delta d * r_{distance}$) and second ($-|\delta\psi| * r_{orientation}$) terms 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 last term ($-\frac{1}{\delta d} * \sqrt{u^2 + v^2} * r_{velocity}$) penalizes the velocity linearly as a function of the distance. As a consequence of this, it encourages the agent to reduce the the velocity as it approaches the final pose.

As mentioned above, negative rewards cause the agent to choose actions that will lead him as quickly as possible to end the episode. With this in mind, it is necessary that the penalty given for a collision to be greater than the sum of penalties that the agent accumulates during the episode in which he does not reach the desired position. Otherwise, as the agent is constantly losing points, if the collision penalty is too low, it can happen the undesirable case in which the agent learns that it is more rewarding to arrive at a collision than to continue the episode.

Another subject to be taken into account is the comparison between the speed penalty and the distance penalty. The ideal behavior of the agent is to approach the final position, gradually decreasing the speed. With this in mind, the distance penalty must be greater than the speed penalty so that the agent does not have the undesirable behavior of stopping near the desired position without actually reaching it.

The chosen reward parameters can be seen in the Table 3.5.

Reward parameter	Value
$r_{collision}$	30 000
r_{final}	30 000
$r_{distance}$	150
$r_{orientation}$	100
$r_{velocity}$	20

Table 3.5: Reward Parameters.

3.2.4 Exploration

The exploration policy during the training was the ϵ-*greedy* policy. This policy selects a greedy action most of the time, but occasionally, with small probability ϵ, selects a random action instead, as repre-

sented in Equation (3.11). An advantage of this method is that, in the limit as the experience increases, every action will be sampled an infinite amount of times, ensuring the training convergence.

$$A_t = \begin{cases} \max Q_t(s,a), & \text{with probability } 1-\epsilon \\ rand(a) & \text{with probability } \epsilon \end{cases} \tag{3.11}$$

In this research, the ϵ was set to 1 in the beginning and gradually decayed for 500,000 steps during training, until it reached the minimum ϵ defined, which was 0.1. This gave a random behaviour to the agent in the beginning, which is a good approach since the agent does not know anything about the environment, and gradually made him exploit the actions where the agent obtained the most reward.

3.2.5 Network Architecture

The neural network adopted is a regular Multilayer Perceptron (MLP) [79] 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 [80] 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 stated 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 Abou Rejaili and Figueiredo [80] (256x128x64);
- The advantage function comes after a fully connected layer with size 77, which is the size of the action space.
- The value function comes after a fully connected layer with size 64.

The adopted activation function was the ReLu (Rectifier Linear Unit) [81], and the RMSProp optimization algorithm [82] was chosen. The network architecture diagram can be seen in the Figure 3.12.

3.2.6 Training parameters and specifications

Nowadays there are many tools, such as Tensorflow [83], Pytorch [84] and Keras [85], that help keep neural network design as high level as possible in order to simplify the training process and focus on the macro problems, instead of focusing on the details of micro functions. In this research, the following tools were used to train the Deep RL model:

- Tensorflow 1.15.0
- Keras 2.1.6

The model was trained on a computer with the following CPU and RAM specifications, respectively:

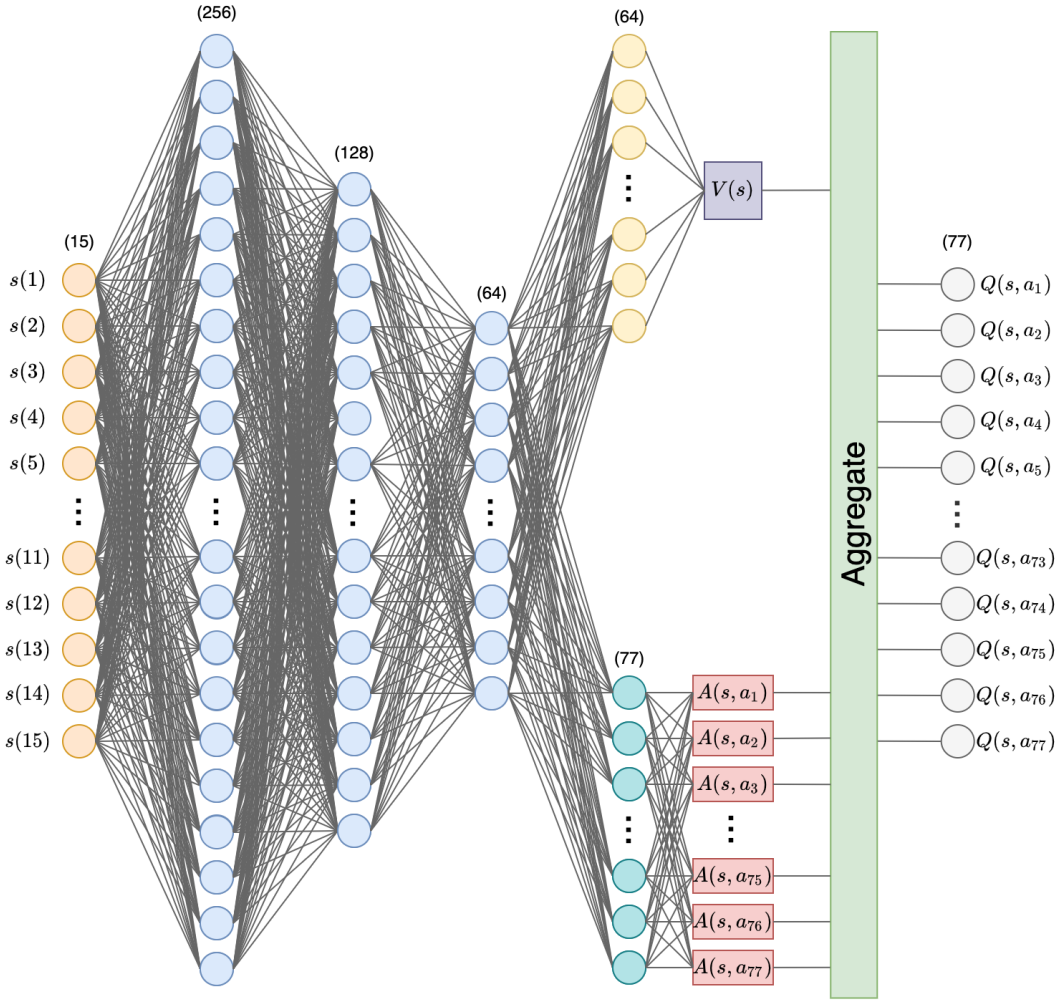

Figure 3.12: Neural Network Architecture. Orange: input state (size 15); Blue: shared MLP (size 256x128x64); Yellow: fully connected layers that precede the value function (size 64); Darker Green: fully connected layer that precede the advantage function (size 77); Purple: value function (size 1); Red: advantage function; Lighter Green: aggregation module; Grey: network output - Q-values of every action (size 77).

- Intel Core i5-8400 CPU @ 2.80GHz x 6
- 11,6 GiB RAM

Hyperparameter tunning is also an important task for good performance in RL models. Although an introduction to parameters that can influence the network performance is made in Section 2.2, for the rest of this section an explanatory summary of the function of each of the parameters will be made and its value will be presented, as well as a justification for it.

Starting by the discount factor $\gamma \in [0, 1]$, this parameter essentially determines how much the RL agent cares about rewards in the distant future relative to those in the immediate future. Values close to 1 value more rewards in the distant future than values close to 0. Since the goal is to get that last very high reward given when reaching the desired pose in safety, the agent must care about the rewards in the distant future and not choose actions that maximize the immediate reward, since these actions may not be the ones that lead him to the maximum accumulated reward. With this in mind, for this research,

it was chosen a discount factor of $\gamma = 0.99$.

The experience replay buffer size is also a parameter that needs to be tuned. This buffer saves the experiences tuples (s_t, a_t, r, s_{t+1}) at each training step. The largest the experience replay, the less likely the agent will sample correlated elements, hence the more stable the training of the Neural Network (NN) will be. However, a large experience replay also requires a lot of memory and it might slow down training. For this research, an experience replay buffer of size 500,000 was chosen as the best value to cope with this trade off.

As stated before, at each time step an experience batch is sampled randomly and the experiences sampled are used to train the network. The batch size should not be too small because then the model takes too long to converge, but it should not be too large as well because it can cause the model to be unable to generalize well on data that hasn't seen before. With this in mind, the batch size chosen in this research was 32, which is the size usually adopted in RL problems.

Another parameter that needs to be taken into account is the target update rate. This parameter sets the periodicity at which the target network is updated. As stated before, the target network is a copy of the online network that is held constant to serve as a stable target for learning for some fixed number of time steps. A low periodicity can lead to instability in the training process, while a large rate can lead to a slow training, so for this research was chosen a target update rate of 500 steps.

The learning rate for the neural network was set to η = 0.0001 and its weights were randomly initialized with uniform distribution (Glorot uniform initializer [86]) limited by a range inversely proportional to the number of inputs and outputs of the each perceptron .

3.3 Scenarios

In this research, three scenarios were taken into account:

- **In the first scenario** (Figure 3.13), the agent starts each episode aligned with the desired docking pose, 16 meters away and with a relative 20-degree orientation between the vessel and the docking orientation. This is a simple first scenario that will test if the agent is able to approach the vessel to the desired pose, without collisions with the harbor structure or other parked vessels. It will also test the ability to gradually slow down near the final pose, as well as the ability to stop.

- **Next**, in the second scenario (Figure 3.14) the agent will start aligned with the mooring pose but from slightly different positions and orientations at each episode in order to prove that the agent can berth from any aligned pose.

- **Lastly**, in the third scenario (Figure 3.15) the agent will start in a position perpendicular to the docking spot. This last task is more complex since, in order to dock the boat in reverse, first, the vessel must move away from the dock to get to the right orientation, and then navigate in reverse to the desired pose.

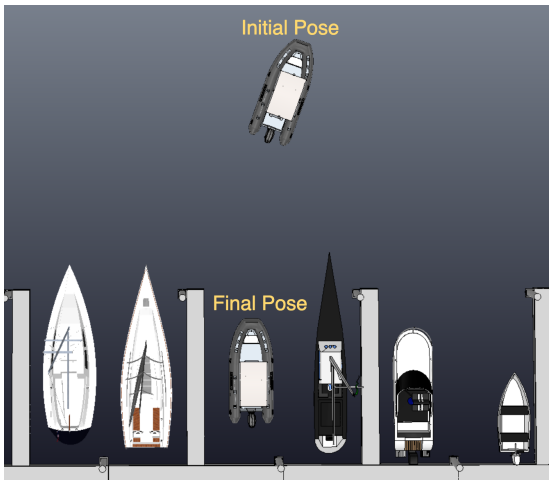

Figure 3.13: Scenario 1.

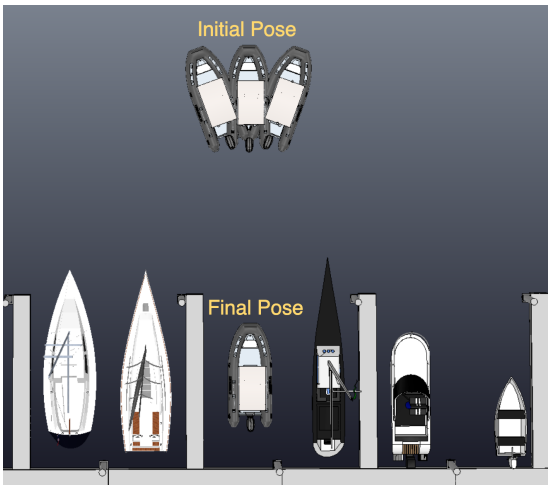

Figure 3.14: Scenario 2.

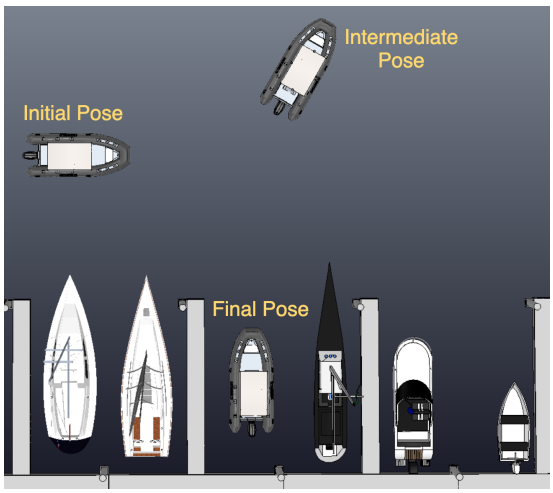

Figure 3.15: Scenario 3.

In all the scenarios the goal is for the agent to be able to dock in reverse. This is an approach taken in most harbors by most vessels since: 1) backing in means an easier departure, driving out forward; 2) with stern boarding platforms, backing in facilitates loading and the access to the cockpit; 3) there are constraints related to power connection. The Table 3.6 shows the initial, intermediate and final poses for each scenario.

Scenarios	Initial Pose			Intermediate Pose			Final Pose		
	x	y	ψ	x	y	ψ	x	y	ψ
Scenario 1	20	2	150	-	-	-	36	0	180
Scenario 2	20 20 20	2 0 -2	160 180 200	-	-	-	36	0	180
Scenario 3	23	-10	90	18	3	150	36	0	180

Table 3.6: Scenario Specifications.

3.3.1 Scenario 3 adjustments

In order to be able to dock starting from a perpendicular position, it is necessary to formulate the problem in a different way. The solution to this problem was inspired by the research of Polvara et al. [87] which uses Deep RL to train two different networks to land a drone on the deck of an Unmanned Surface Vehicle. One network is trained to align the drone with the platform and the other one is trained to do the descend maneuver. However, in this research a one-net approach was adopted with a subtle change in the state space representation. A variable was added that can only have the value of 0 or 1, depending on the task that the agent is completing. The value 0 is given when the agent's goal is to navigate away from the dock so as to be aligned with the final pose, and the value 1 is given when the agent's goal is to approach the final pose in reverse. This way the agent knows at all times what its goal is, without needing another network or a path-following approach. The state space is then similar to the representation shown in Equation (3.9) but with the addition of this binary variable s_{task}, making the state space vector with size 16, as shown bellow:

$$s_t = [s_{task}, \delta d, \delta\psi, u, v, r, P_{i-1}, \phi_{i-1}, d_{Lidar_1}, ..., d_{Lidar_8}]. \tag{3.12}$$

There are also some adjustments that need to be made to the reward function, as can be seen in Equation (3.13):

$$r = \begin{cases} -r_{collision}, & \text{if collision} \\ r_{pre-final}, & \text{if intermediate pose} \\ r_{final}, & \text{if final pose} \\ -\delta d_0 * r_{distance} - |\delta\psi_0| * r_{orientation} - \frac{1}{\delta d_0} * \sqrt{u^2+v^2} * r_{velocity}, & \text{if phase 0} \\ -\delta d_1 * r_{distance} - |\delta\psi_1| * r_{orientation} - \frac{1}{\delta d_1} * \sqrt{u^2+v^2} * r_{velocity}, & \text{if phase 1} \end{cases} \tag{3.13}$$

where,

- δd_0 and δd_1 are the relative distances to the intermediate position and final position, respectively;
- $\delta\psi_0$ $\delta\psi_1$ are the relative orientations between the vessel and intermediate pose and final pose, respectively;
- $r_{pre-final}$ is the reward that the agent receives when it reaches the intermediate pose. When the agent reaches this state, the s_{task} variable also changes to 1.

The intuition behind this reward function adjustment was the same as the reward function presented in the Equation (3.10), but instead of having only one phase it has two. The goal of the first phase is to approach the intermediate pose by navigating forward and decrease the speed near this position so that the boat is aligned with the berthing position and can reverse smoothly. With this in mind, the reward function for this phase will penalize the distances and orientations to the intermediate position, as well as the velocity near this position. Once the agent reaches this position, the s_{task} variable changes to

1 and, consequently, its goal and reward function also change. For this final phase the goal is also to approximate the vessel to the final position by gradually decreasing its velocity until a full stop, but now navigating in reverse. To achieve this behaviour, the reward function penalizes, at this stage, the distance and orientation to the final position, as well as the velocity near this position.

In the next Chapter, results for these 3 scenarios are shown.

Chapter 4

Results

Usually the boat enters the harbor in a position perpendicular to the final position. In order to berth the boat in reverse, it is necessary that the agent is able to complete two different tasks. In the first phase it is necessary to move away from the final position in order to be aligned with the dock, and in the second phase it is necessary to navigate the vessel in reverse decreasing the speed when approaching the desired berthing position.

In this Chapter, these two tasks will be tested in an iterative way. First, in Section 4.1, a model will be tested in which, in training, the agent started each episode aligned with the dock (Figure 3.13), always starting from the same position and orientation, with the aim of approaching the final position in reverse. With this we test the agent's ability to approach the final position in safety, as well as his ability to slow down until a complete stop.

Secondly, in Section 4.2, a model in which the agent was trained starting again each episode aligned with the final position (Figure 3.14) will be tested, but this time from three different positions and orientations, as explained in Table 3.6. This way, we test the agent's ability to approach the final position, even when the boat changes position and orientation because of eventual external disturbances, such as wind and currents.

Lastly, in Section 4.3, the entire berthing process will be tested. For this, each episode will start with the boat in a position perpendicular to the docking position (Figure 3.15). With this scenario we test the agent's ability to complete the entire docking process, formulating the problem in the same way as in the previous scenarios, making only the adjustments represented in Section 3.3.1. That is, (i) adding the extra variable s_{task} that defines in which task the agent is, (ii) the reward adjustment that optimizes the navigation for each task. As a way to demonstrate the generalization of the trained model, in this scenario it will also be tested the ability of the agent to dock in positions that are different from the positions used in the training episodes.

The results for the three scenarios will be presented in three steps:

- **First**, the training performance will be shown through the analysis of the plots of the reward, the number of steps per episode and ϵ-value (defines the probability of a random action) in each episode.

- **Next**, the agent's behaviour will be analyzed. For this purpose, the plots of (i) the position, (ii) the relative distance and (iii) orientation to the final position, (iv) the velocities, and (v) the control variables will be shown.

- **Finally**, and just for the Scenario 2 and 3, the generalization of trained model will be analyzed by testing the agent's ability to perform the docking procedure in positions that are different than the positions used in the training. For this purpose, the same analysis for the agent's behaviour is done, but for different docking positions.

The following sections present the results of training in a docking scenario. Here the most significant measures are presented and discussed, while additional supporting results can be found in Appendix B.

4.1 Scenario 1

As stated before, for this scenario the agent starts each episode always aligned with the docking position (Figure 3.13) and the goal is to approach the vessel to the final position in a safe manner, reducing the velocity until a complete stop.

For each episode the vessel starts at the pose $(x, y, \psi) = (20, 2, 160^{\text{o}})$ and the goal is to reach the final pose $(x, y, \psi) = (36, 0, 180^{\text{o}})$. The episode ends if one of the conditions bellow is satisfied:

- the agent reaches the final pose with $\sqrt{u^2 + v^2} \leq 0.1$ m/s;

- there is a collision;

- the agent reaches the maximum number of 500 steps for each episode.

This model was trained for a total of 4000 episodes and the ϵ_{decay} was calculated with the following equation:

$$\epsilon_{decay} = \frac{\epsilon_{start} - \epsilon_{end}}{n_{exploration}}, \tag{4.1}$$

where, ϵ_{start} is the ϵ-value in the beginning of the training, ϵ_{end} is the value at which the ϵ-value stops decaying and stays constant, and $n_{exploration}$ is the number of exploration steps. At each step the ϵ is updated using the the equation written bellow:

$$\epsilon_i = \epsilon_{i-1} - \epsilon_{decay}, \tag{4.2}$$

where, ϵ_{i-1} is the ϵ-value of the previous step. This update is done until the ϵ-value reaches ϵ_{end} and it remains constant until the end the training. The values for ϵ_{start}, ϵ_{end} and $n_{exploration}$ parameters can be seen in the Table 4.1.

The total reward, the number of steps and the ϵ-value for each training episode can be seen in Figures (4.1), (4.2) and (4.3), respectively. Since both the total reward and the number of steps have a lot of noise due to random actions, it is also presented a smoothed version of these plots. This can

ϵ_{start}	ϵ_{end}	$n_{exploration}$
1.0	0.01	500 000

Table 4.1: ϵ parameters for Scenario 1.

be seen in the Figures (4.1) and (4.2), where the lighter color is the raw data and the darker color is the smoothed data. This was done in order to see more clearly the trend of the training throughout the episodes.

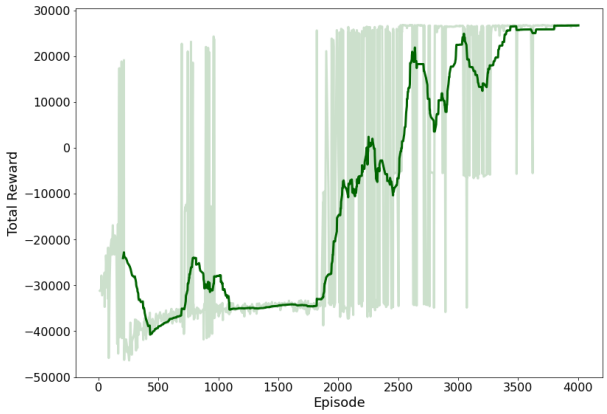

Figure 4.1: Total reward during training.

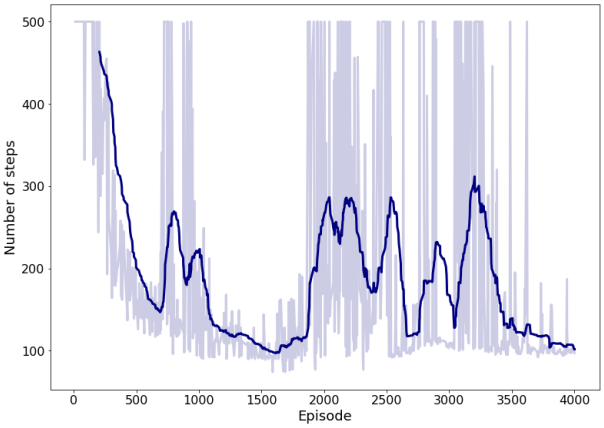

Figure 4.2: Episode duration during training.

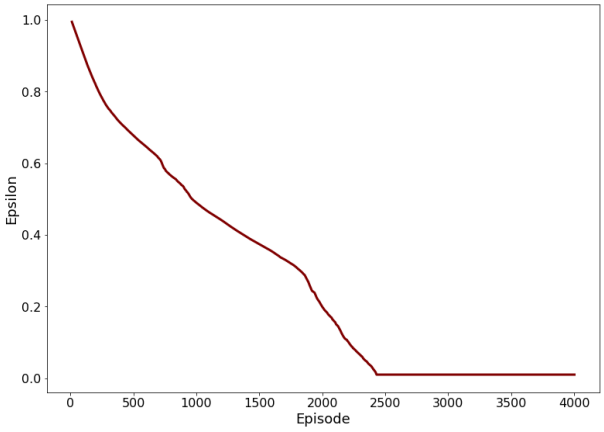

Figure 4.3: ϵ-value during training.

Since the reward difference between episodes in which the agent collides (-30 000) and episodes in which the agent reaches the final position (+30 000) is very large, it is possible to make a more detailed analysis of what happened during the training. Very high peaks during the training are equivalent to episodes in which the agent reaches the final position with the desired speed, and, on the contrary, very low pits are equivalent to episodes in which the agent collided with some obstacle.

As the agent only has random actions at the beginning of the training due to the high value of ϵ, it is possible to see in the Figure 4.2 that the initial episodes are ended by the condition of the maximum number of steps up to approximately episode 200. Near this episode it is also possible to see in Figure 4.1 that the agent reached the final pose, but the following episodes were dominated by

collisions until approximately episode 2000. From this episode onwards, the reward started to increase, and the episodes where the agent collided started to be less frequent, and, on the contrary, the episodes where the agent reached the final pose started to be more and more frequent, as well as the episodes that end due to the maximum number of steps.

From episode 2500 onwards, it can be seen in Figure 4.3 that the ϵ-value stopped decreasing. It is also from this episode on that the collisions begin to be almost non-existent and the agent concentrates more on the actions that lead to the final position or on actions that do not lead to collisions. The training was stopped when the agent had already reached the final position in the last 400 episodes in order to conclude that the model had already converged. This convergence can be seen in Figure 4.1, where the reward remained constant during the final episodes, not showing a tendency to increase, and in Figure 4.2, where the duration of each episode remained practically constant. This can conclude that the agent was not able to find a more optimal solution.

The next section shows the agent's behavior for the model saved in the last episode.

4.1.1 Agent's behaviour

Once again, the agent's goal in this scenario is to approach the final position safely, as well as to reduce the speed near this position until a complete stop. The plots of the vessel's position and orientation in relation to the dock (Figure 4.4), the distance and orientation in relation to the final position (Figures 4.5 and 4.6), the velocities (Figures 4.7 and 4.8), and the control variables (Figures 4.9 and 4.10) will be presented in this section in order to test whether the agent's behavior is the desired.

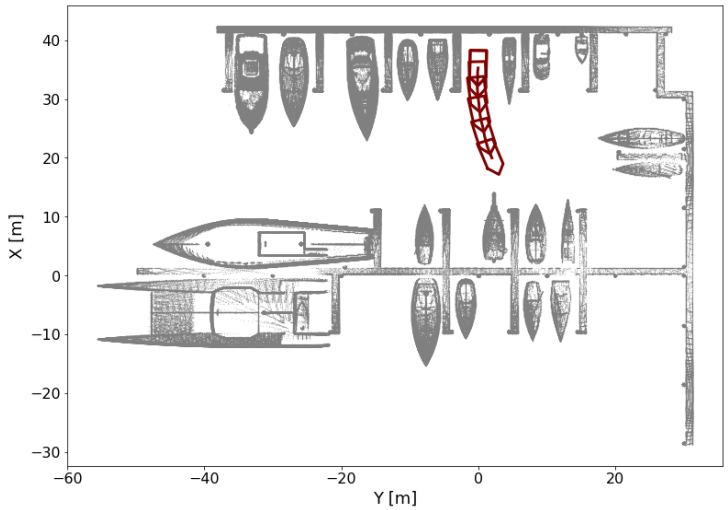

Figure 4.4: Vessel's pose (x, y, ψ).

The agent's navigation behaviour can be seen in Figure 4.4. It can be observed that the agent was successful in safely approaching the desired position in reverse. It can also be seen from the relative position and orientation plots (Figures 4.5 and 4.6) that there was a gradual approach both in distance and in orientation that leads the agent to reach the final position and maximize the accumulated reward, rather than the instant reward.

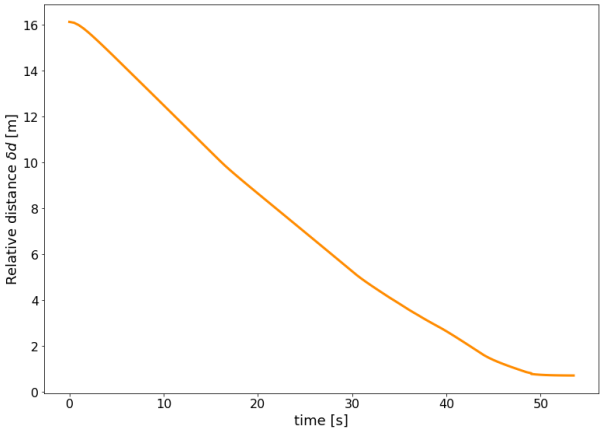

Figure 4.5: Relative distance δd [m].

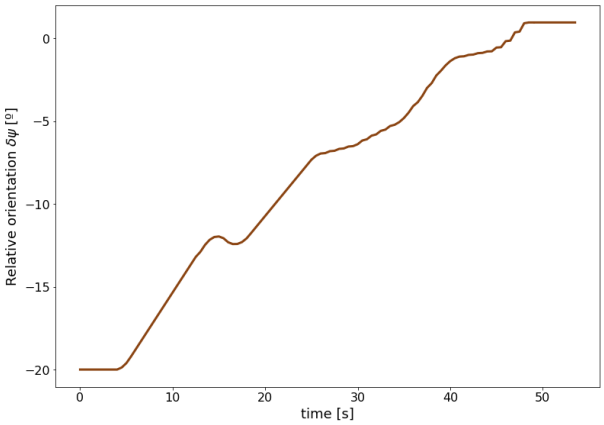

Figure 4.6: Relative orientation $\delta\psi$ [º].

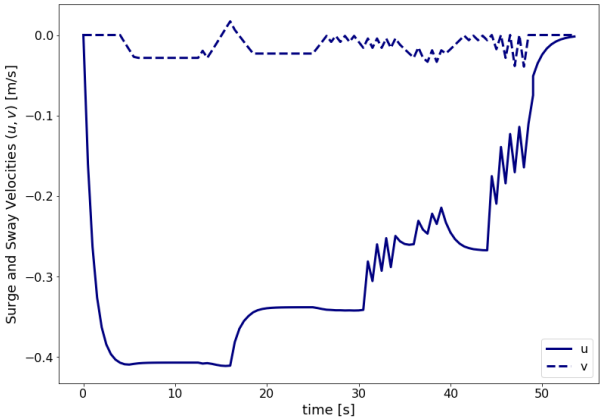

Figure 4.7: Surge and Sway Velocities (u, v) [m/s].

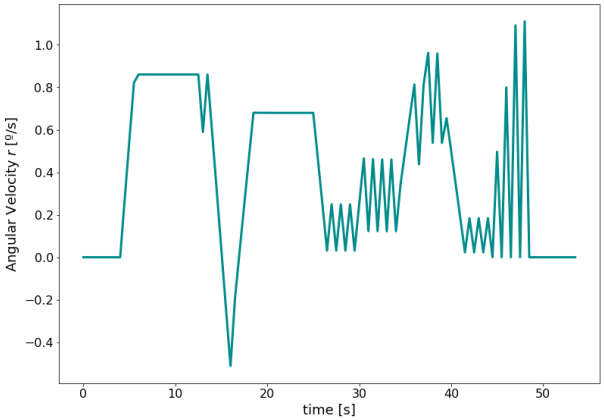

Figure 4.8: Angular velocity r [º/s].

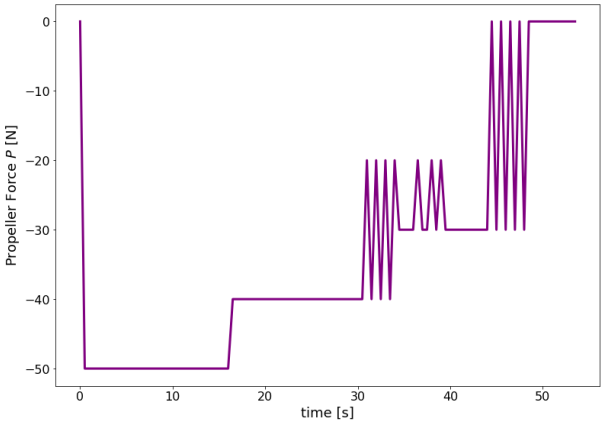

Figure 4.9: Propeller Force P [N].

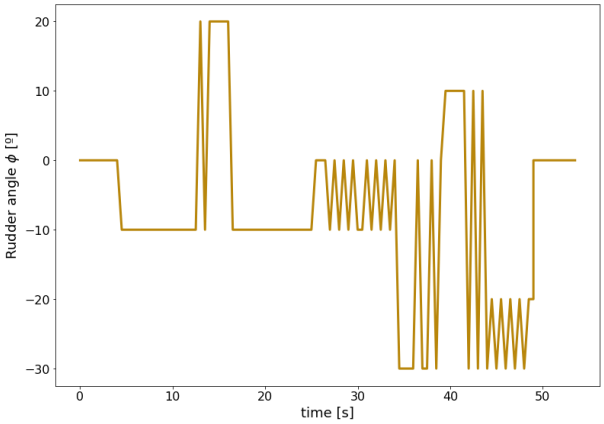

Figure 4.10: Rudder Angle ϕ [º].

Analyzing also the surge and sway velocities, it can be seen in Figure 4.7 that there is a demand by the agent to maximize the reward, by putting the maximum speed until the moment where the velocity penalty is too high and it is more rewarding to reduce to a lower speed. This behavior can be explained by the reward function that was designed for this research. Since, at each step, the agent is always loosing points due to the negative reward, it will always try to act in order to decrease the number of steps until the end of the episode.

The last desired behavior to be analyzed is the gradual decrease in the velocity. This can be seen in Figure 4.7 from the second 15 onwards, where there is a gradual decrease in the velocity when the the vessel's is 10 meters away from the final position. Although this behavior is accompanied by some sudden changes in the control variables (Figures 4.9 and 4.10), it is possible to observe the desired tendency in the decrease of speed until it stops completely.

4.2 Scenario 2

For this scenario, the same training approach as the previous scenario (Section 4.1) was chosen. The agent also starts each episode always aligned with the docking position and the goal is to approach the vessel to the final position in a safe manner, reducing the velocity until a complete stop, but at the beginning of each episode a small perturbation is imposed in the pose, choosing from one of the poses represented in Table 4.2, instead of choosing always the same one. The goal is to reach the final pose $(x, y, \psi) = (36, 0, 180^{o})$.

	Pose 1	Pose 2	Pose 3
x	20	20	20
y	-2	0	2
ψ	200º	180º	160º

Table 4.2: Initial poses for scenario 2.

Each episode ends if one of the conditions stated in the Section 4.1 is satisfied. The model was trained for a total of about 4700 episodes and the ϵ_{decay} is the same as in the previous scenario, with the same ϵ parameters represented in the Table 4.1.

The total reward, the number of steps and the ϵ-value for each training episode can be seen in Figures (4.11), (4.12) and (4.13), respectively. As in the previous scenario, both the total reward and the number of steps were smoothed in order to analyze more easily the training trend. The lighter color is the raw data and the darker color is the smoothed data.

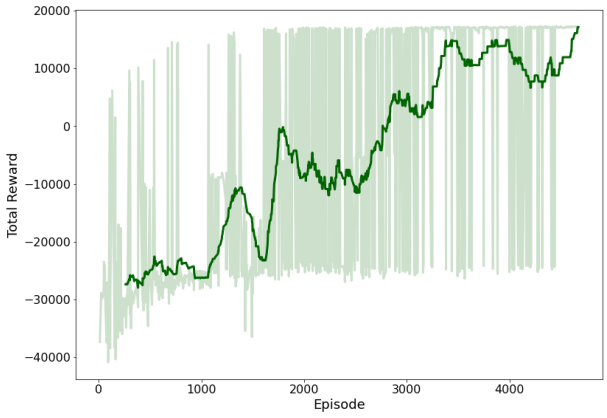

Figure 4.11: Total reward during training.

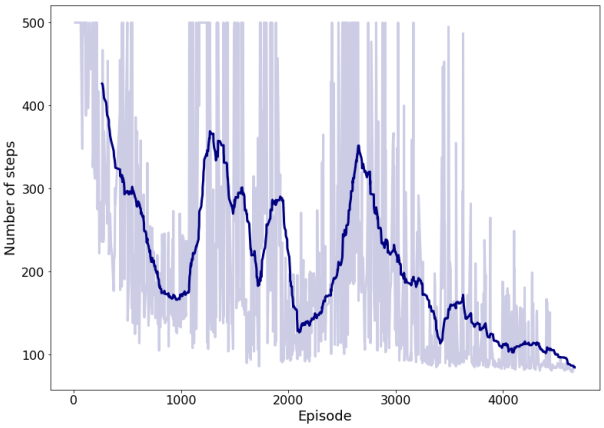

Figure 4.12: Episode duration during training.

As in the previous scenario, very high peaks during the training are equivalent to episodes in which

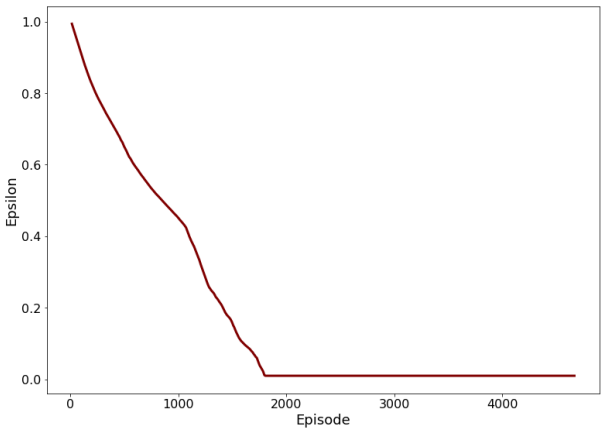

Figure 4.13: ϵ-value during training.

the agent reaches the final position with the desired speed, and, on the contrary, very low pits are equivalent to episodes in which the agent collided with some obstacle.

The training performance for this scenario is not very different from the previous scenario. It can be seen in the Figure 4.12 that the initial episodes are also ended by the condition of the maximum number of steps up to approximately episode 200 due to the very high randomness of actions in the beginning of the training. Near this episode the agent started to reach the final position occasionally, but, as in the previous scenario, the following episodes were dominated by collisions near the final position, in a search to reach it. From around episode 1600 onwards, the reward started to increase, with more frequent episodes where the agent reaches the final position and less frequent episodes where there is a collision, as can be seen in Figure 4.11. From episode 2000 onwards, the ϵ-value stopped decreasing and remained constant at 0.01 until the end of the training, as can be seen in the Figure 4.13. This is equivalent to a 1% probability of the agent triggering a random action until the end of the training.

The reward showed the tendency to increase until the end of the training, as can be seen in the Figure 4.11. The training was stopped at the end of 4700 episodes, when the agent had already reached the final position for 200 consecutive episodes, which demonstrates the model convergence. This convergence can be seen in Figure 4.12, where, the in final episodes, the agent showed the ability to reach the final position in safety, but the duration of the episode remained constant, which shows that the agent was no longer able to find a more optimal solution. This can also be seen in Figure 4.11, where the reward was kept constant during the final episodes, not showing a tendency to increase.

The next section shows the agent's behavior for the model saved in the last episode.

4.2.1 Agent's behaviour

Once again, the agent's goal in this scenario is to approach the final position safely, as well as to reduce the speed near this position until a complete stop. However, as in this scenario the agent was trained for more than one initial position, then the behavior of the agent was tested for 5 different initial positions. In order to demonstrate the slight generalization of the model, 3 of them were different from the initial positions chosen in the training. The initial positions chosen for this test are represented in the Table

4.3, together with the colors of the plots chosen for each of them.

	Pose 1	Pose 2	Pose 3	Pose 4	Pose 5
x	20	20	20	20	20
y	3	2	1	0	-1
ψ	150º	160º	170º	180º	190
color	Red	Green	Blue	Purple	Yellow

Table 4.3: Initial poses for the test in scenario 2.

The plots of the vessel's position and orientation in relation to the dock (Figure 4.14), the distance and orientation in relation to the final position (Figures 4.15 and 4.16), the velocities (Figures 4.17 and 4.18), and the control variables (Figures 4.19 and 4.20) will be represented in this section.

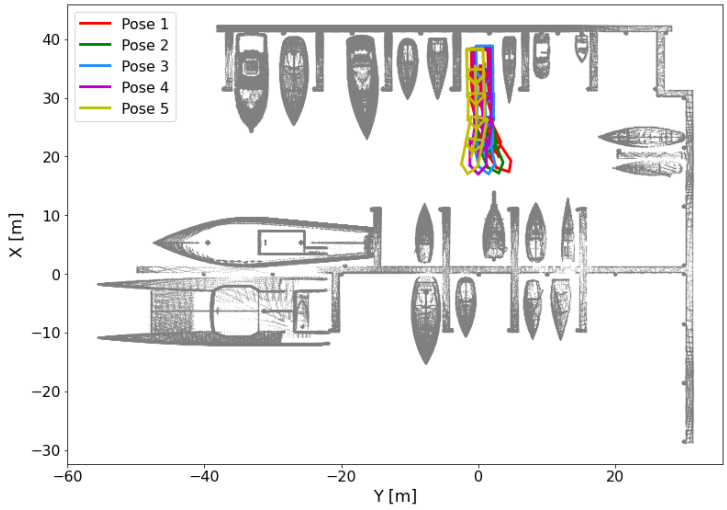

Figure 4.14: Vessel's pose (x, y, ψ).

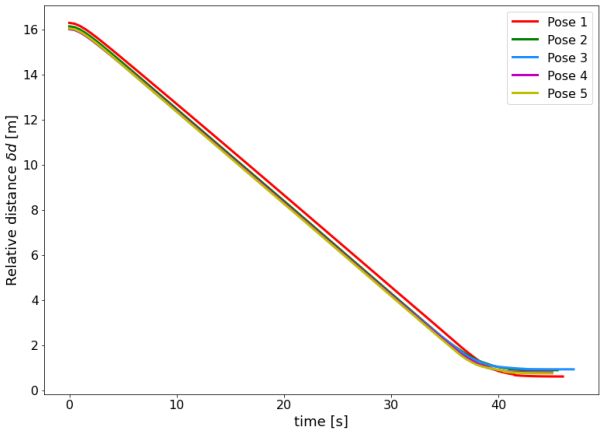

Figure 4.15: Relative distance δd [m].

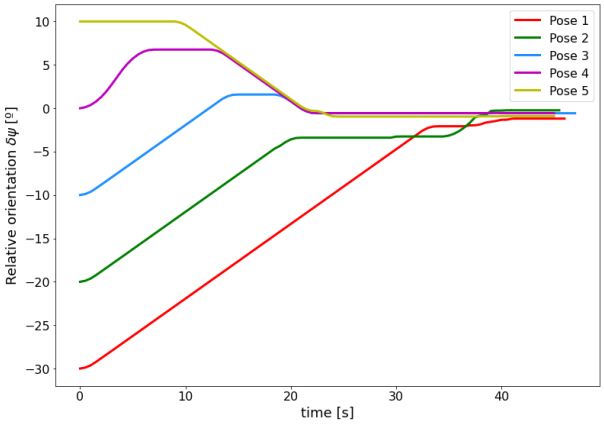

Figure 4.16: Relative orientation $\delta\psi$ [º].

The agent's navigation behaviour can be seen in Figure 4.14. It can also be seen that the agent was successful in safely approaching the desired position in reverse, starting from the 5 different positions.

As in the previous scenario, it can also be seen in the relative position and orientation plots (Figures 4.15 and 4.16) that there was a gradual approach both in distance and in orientation that leads the agent

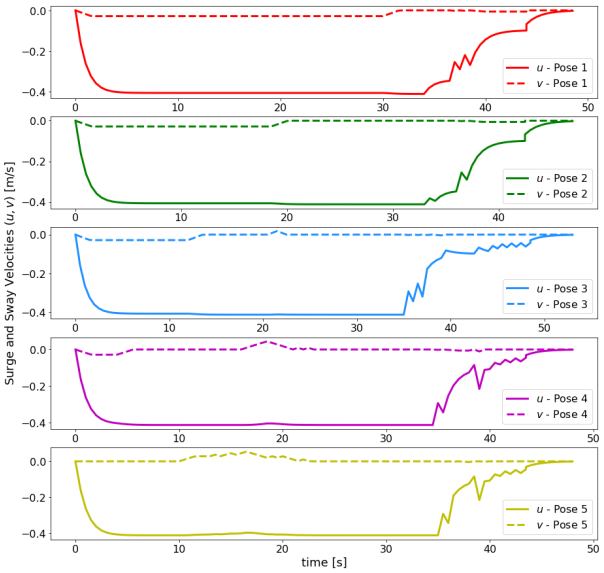

Figure 4.17: Surge and Sway Velocities (u, v) [m/s].

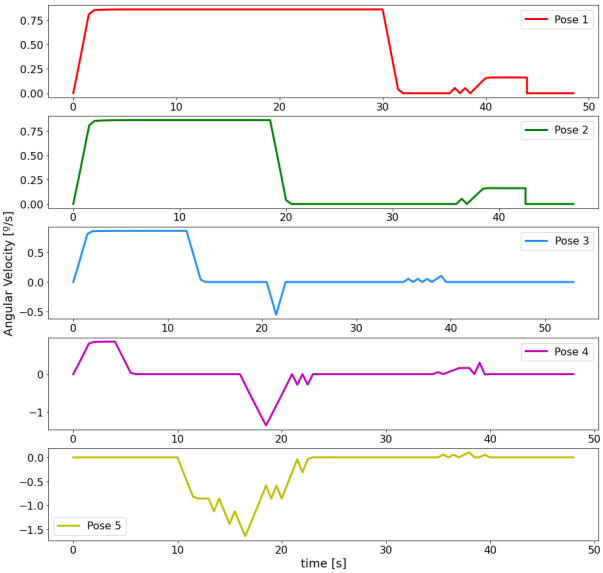

Figure 4.18: Angular velocity r [°/s].

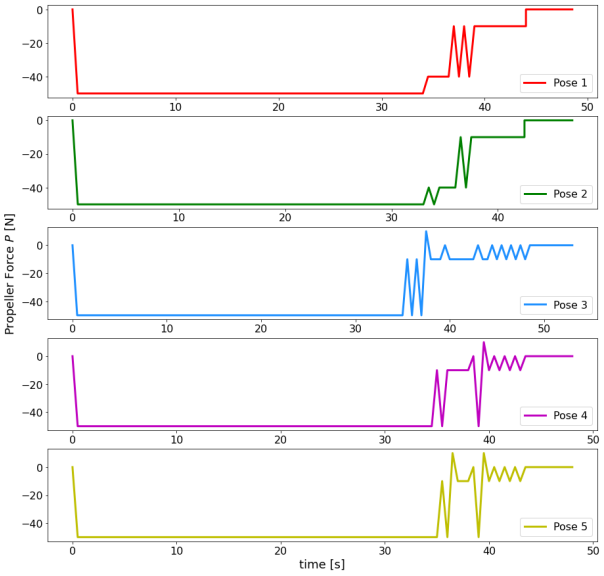

Figure 4.19: Propeller Force P [N].

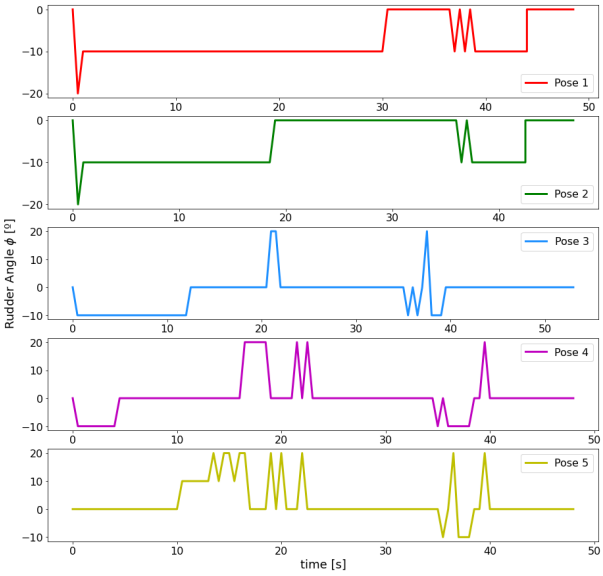

Figure 4.20: Rudder Angle ϕ [°].

to reach the final position and maximize the accumulated reward, rather than the instant reward.

It is also possible to see in the plots of the surge and sway velocities (Figure 4.17), that there is a tendency for the agent to put the maximum speed until the moment in which the velocity penalty becomes too high and then the agent gradually decreases the speed until the final position, as happened in the previous scenario. The first part where the agent navigates at maximum speed can again be explained by the design of the reward function. Since, at each step, the agent is losing points due to the negative reward function, he then tries to get as fast as possible to the end of the episode in order to decrease the number of steps where a negative reward is given. However, when the agent approaches the final position the velocity penalty starts to increase, which leads the agent to decrease the velocity, since it is more rewarding, until a complete stop.

Although this decrease in speed is accompanied by a somewhat sudden change in propeller force,

as can be seen in Figure 4.19, this does not have a very strong influence on speed, since the agent is acting at low speeds. Also, comparing with the previous scenario, it can be concluded that, in this model, the changes in propeller force and rudder angle (Figures 4.19 and 4.20) are less abrupt.

4.3 Scenario 3

Finally, for this final scenario, an extra task was added: start the navigation from a position perpendicular to the docking position and navigate first to a position aligned with the dock, in order to berth the vessel in reverse. To solve this navigation approach, generally chosen by most motorboats, a variable was added to tell which of the tasks the agent is completing. If the agent is navigating forward in order to align with the dock, this variable has a value of 0. If the agent is navigating in reverse towards the final docking position, this variable has the value of 1. This way, the agent always knows what its goal is.

The training approach chosen for this scenario was more or less the same chosen for Scenario 1 presented in Section 4.1. At each episode the agent always starts at the same position and orientation $(x, y, \psi) = (23, -10, 90^{\text{o}})$. 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 $(x, y, \psi) = (18, 3, 150^{\text{o}})$. This position was chosen because it was one of the initial positions in Scenario 2 where the agent was successful. The variable s_{task} changes to 1 as soon as all of the the following conditions are met:

- The distance relative to the intermediate position is less than or equal to 1,5 m;
- The orientation relative to the intermediate position is less than or equal to 5º;
- The speed of the agent ($\sqrt{u^2 + v^2}$) is less than 0.1 m/s.

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 by $(x, y, \psi) = (36, 0, 180^{\text{o}})$.

A small change in the reward function was also introduced. In an attempt to eliminate sudden movements near the final position, the value of the parameter $r_{velocity}$ in Table 3.5 was changed from 20 to 10, thus taking away some influence of velocity in the reward function.

Each episode ends if one of the stop conditions stated in the Section 4.1 is satisfied. The model was trained for a total of about 4000 episodes and the epsilon decay is the same as in the Scenario 1, with the same ϵ parameters represented in the Table 4.1.

An additional plot that shows in which task the agent finished the episode was added in this analysis in order to be able to see more clearly what happened during the training.

The total reward, the number of steps and the plot that shows in which task the agent finished for each training episode can be seen in Figures (4.21), (4.22) and (4.23), respectively. As in the previous scenarios, since these three metrics have a lot of noise due to the agent's random actions, a smoothing of these plots was done to make clear the tendency of the data.

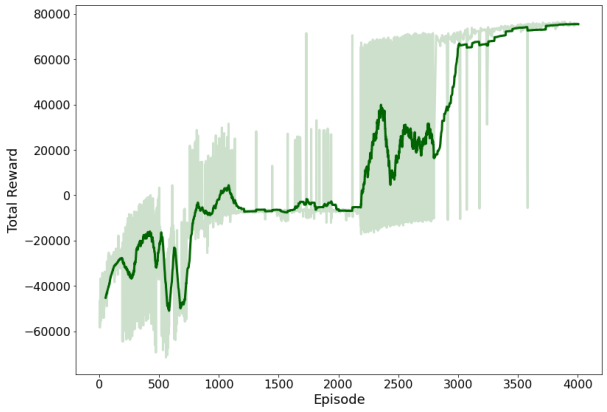

Figure 4.21: Total reward during training.

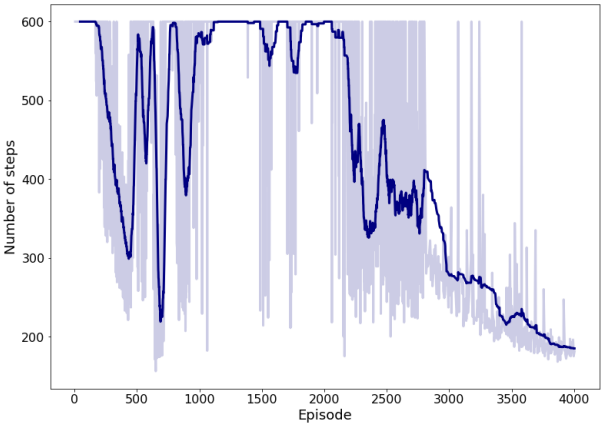

Figure 4.22: Episode duration during training.

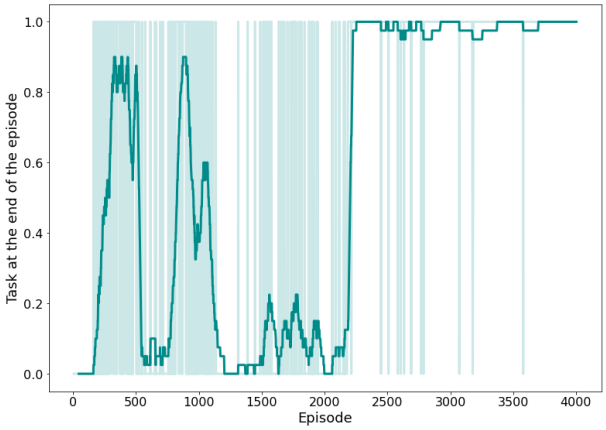

Figure 4.23: Task at the end of the episode: 0 for the task of aligning the vessel with the dock; 1 for the task of approaching the vessel to the final position.

As in the previous scenarios, very high peaks in the reward plot (Figure 4.21) during the training are equivalent to episodes in which the agent reaches the final position with the desired speed, and, on the contrary, very low pits are equivalent to episodes in which the agent collided with some obstacle. In this scenario are present intermediate peaks that are equivalent to episodes in which the agent reaches the intermediate position. When the agent reaches this position it receives a very high reward, but the episode does not end and the agents starts getting penalized for being distant from the final position.

Starting with the plot of the episode's duration (Figure 4.22), it can be seen that, as in the previous scenarios, the initial episodes end with the condition of maximum number of steps. From about the 200 episode onwards the agent often started to reach the intermediate position, as it can be seen in Figure 4.23, since there were many episodes that ended with the value of the variable s_{task} in 1. This can also be seen in Figure 4.21, where there is an increase in the episode's reward from this episode up to the 500 episode.

From episode 500 up to episode 750 the arrival to the intermediate position was less frequent and started to occur more collisions. This trend was then countered until approximately episode 1200 where the agent reached significantly more times the intermediate position without collisions, as can be seen by a significant increase in reward in Figure 4.21.

However, approximately from episode 1200 until episode 2200, the agent seems to have found a local minimum, where the episodes were dominated by a non-ideal behavior where the agent ended each episode with the maximum step condition, not even reaching the intermediate position, as can be seen by Figure 4.23 where most of the episodes in this range ended with the variable s_{task} at 0. This trend was then countered by an increase in the number of episodes in which the agent, not only reaches the intermediate position, but also reaches the final position safely. From then on, the agent was divided between actions that lead the boat to the final position safely and actions that lead to collisions near the final position.

At the end of the training, from approximately episode 3000, it is possible to see that collisions are much less frequent and the training begins to stabilize. This stabilization can be seen first in Figure 4.21, where the reward remained stable until the end of the training, and in Figure 4.22, where the duration of the final episodes also remained constant, thus concluding that the agent did not find any sequence of actions that would lead him to decrease the execution time of the task and, consequently, that would lead him to have a higher reward. In short, he did not find a better optimal solution.

The next section shows the agent's behavior for the model saved in the last episode.

4.3.1 Agent's behaviour

Once again, the agent's goal in this scenario is to first approach an intermediate position that puts the vessel aligned with the dock, and then approach to the final position in reverse. In order to analyze whether the agent is behaving according to this description, the plots of the vessel's position and orientation in relation to the dock (Figure 4.24), the distance and orientation in relation to the final position (Figures 4.25 and 4.26), the velocities (Figures 4.27 and 4.28), and the control variables (Figures 4.29 and 4.30) will be represented in this section.

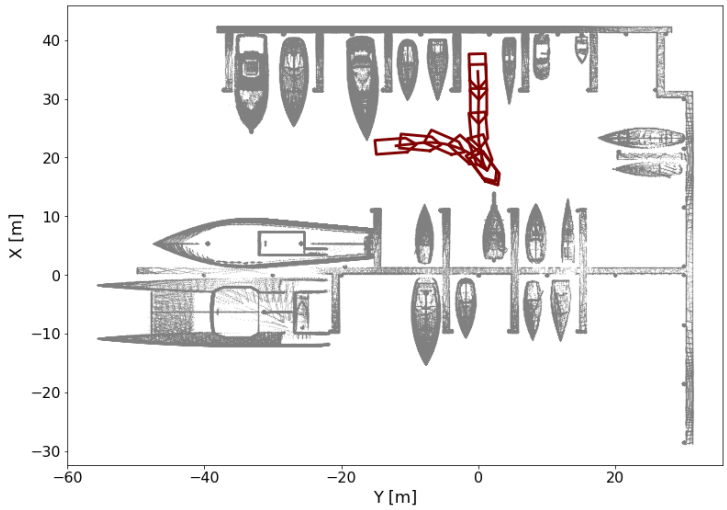

Figure 4.24: Vessel's pose (s, y, ψ).

The agent's navigation behaviour can be seen in Figure 4.24. It is necessary to emphasize that these initial, intermediate and final positions are the same used in the training episodes. Regardless, it can be concluded that the agent was successful in performing the proposed task in safety.

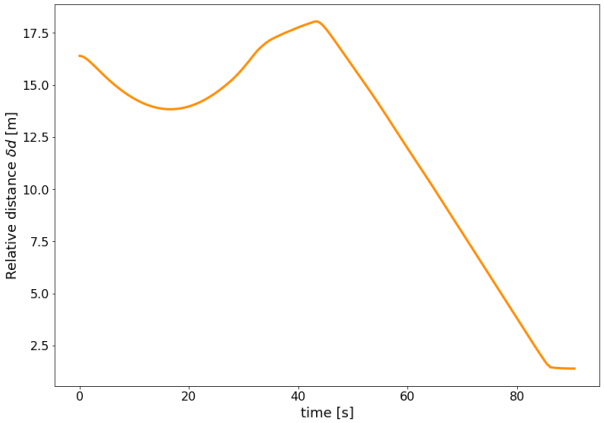

Figure 4.25: Relative distance δd [m].

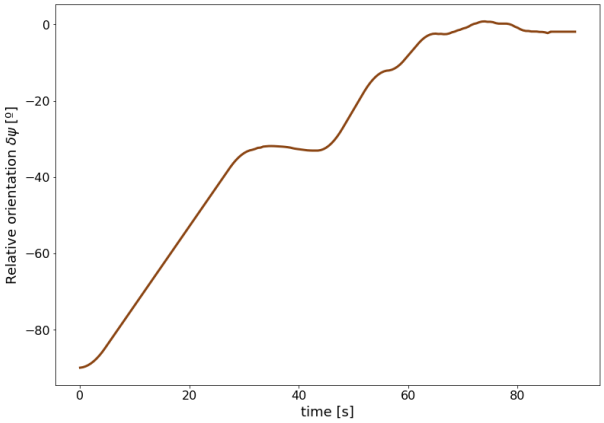

Figure 4.26: Relative orientation $\delta\psi$ [º].

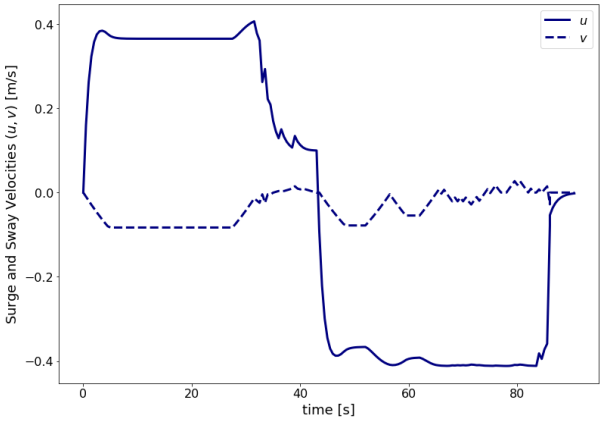

Figure 4.27: Surge and Sway Velocities (u, v) [m/s].

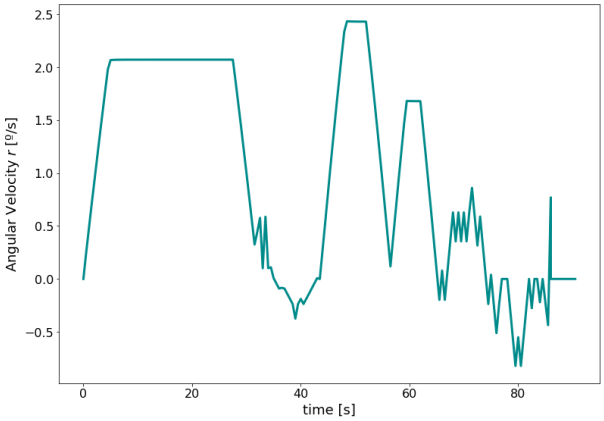

Figure 4.28: Angular Velocity r [º/s].

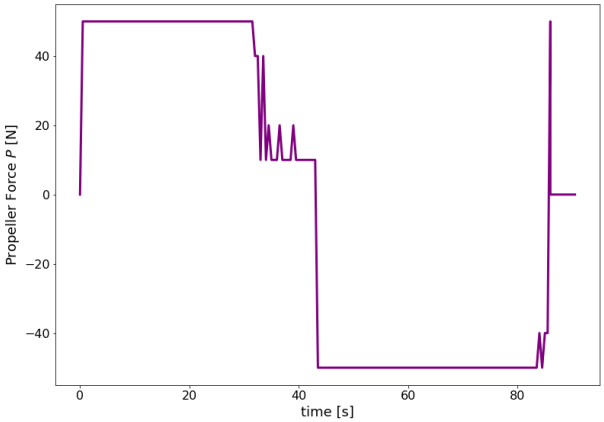

Figure 4.29: Propeller Force P [N].

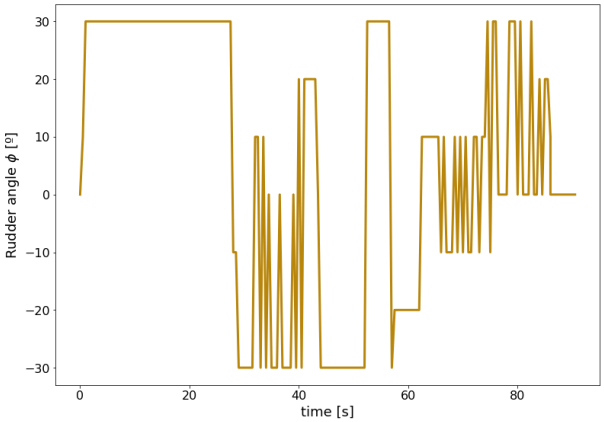

Figure 4.30: Rudder Angle ϕ [º].

Starting perpendicularly positioned in relation to the dock, one can see from Figure 4.25 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 approaches to the final position. It can also be seen in Figure 4.25 that the change in the value of this variable took place around the second 40, once the agent stopped moving away from the dock and started to decrease the distance to the final position.

Looking also at the relative orientation (Figure 4.26) and the rudder angle (Figure 4.30) during the first phase, one can see that the agent reached the desired orientation before reaching the intermediate position since it activated the maximum rudder orientation right from the start. The orientation then remained constant until the end of the first task. As soon as the second task is started, it can be seen a steeper steer at the beginning to bring the boat in line with the berthing orientation. The rest of the steer was done more slowly and gradually until it remained constant when the desired orientation had already been reached.

Analyzing the global behavior of the agent in approaching the desired distance and orientation from Figures 4.24, 4.25 and 4.26, one can conclude that it was successful, and there was a gradual and safe approach, comparable with the behavior of a pilot in a real docking task.

Looking at the velocity (Figure 4.27) and the propeller control (Figure 4.29), and comparing with the other scenarios, it is possible to see that the agent doesn't start reducing speed early anymore, since the speed penalty in the reward function has been reduced to half, showing almost no influence on the agent's behavior. However, it is also possible to see that the sudden control on the propeller force practically ceased to exist with the change made to the velocity parameter in the reward function.

It is also possible to see in the Figure 4.27 that, for both the first and the second task, the agent decides to put the maximum speed until near the end of each task due to the negative reward function. At the end of the first task the agent only reduced the speed when he was almost reaching the intermediate position. However, at the end of the second task, the agent learned a behavior that is usually 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. Sudden changes of control in this velocity range do not have a worrying influence on the velocity of the system.

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. In Section 4.4 a solution to this problem will be proposed, as well as an overall commentary on the performance of the various scenarios tested here.

To complement the tests done here, the next section shows a test for a scenario that was never seen by the agent during training in order to demonstrate the generalization of the trained model.

4.3.2 Non trained scenario

The model trained for the previous scenario was only trained for that specific initial, intermediate and final positions. However, in order to prove the generalization of the trained model, in this section, the

model is tested for the following initial, intermediate and final positions:

	Initial Pose	Intermediate Pose	Final Pose
x	21	25	7
y	-11	0	-2
ψ	90º	50º	0º

Table 4.4: Initial poses for second test in scenario 3.

As it can be seen, although the starting position is practically the same used in training, the intermediate position is very different, with a difference of 100º compared to the intermediate orientation used in the training. The final position is also very different, being on the opposite side of the dock used in the training. The difference between the final orientations is 180º. In short, it is necessary to make the navigation contrary to what was trained, which makes this a good test to analyze the generalization of the trained model.

Once again, the agent's goal in this scenario is to first approach an intermediate position that puts the vessel aligned with the dock, and then approach to the final position in reverse. In order to analyze whether the agent is behaving according to this description, as well as the generalization of the model, the plots of the vessel's position and orientation in relation to the dock (Figure 4.31), the distance and orientation in relation to the final position (Figures 4.32 and 4.33), the velocities (Figures 4.34 and 4.35), and the control variables (Figures 4.36 and 4.37) will be represented in this section.

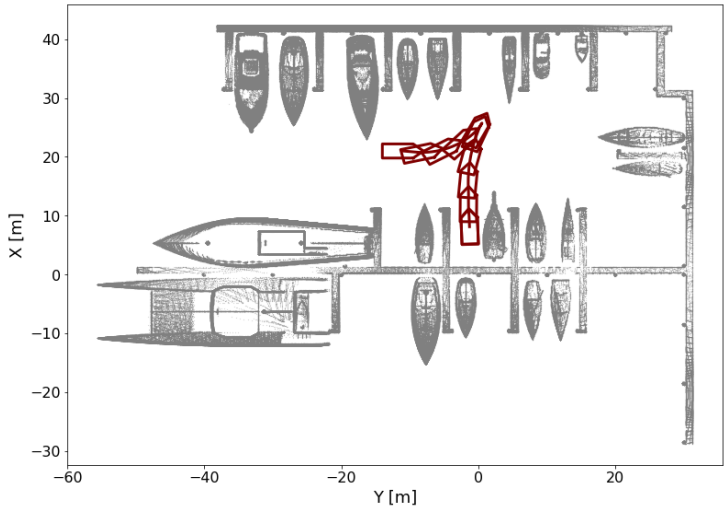

Figure 4.31: Vessel's pose (x, y, ψ).

The agent's navigation behaviour can be seen in Figure 4.31. Comparing with the agent's behavior in the training scenario it can be seen that there are not many differences in the navigation, both in the first and second berthing phases, having a gradual approach to the intermediate and final position, as can be seen in Figures 4.32 and 4.33, respectively.

In Figure 4.32 it can be seen the same behavior presented in the training scenario, where the agent has a movement similar to that of a pilot completing the berthing task in a real test. There is first a forward navigation, away from the dock, to approach the intermediate position in order to then make the

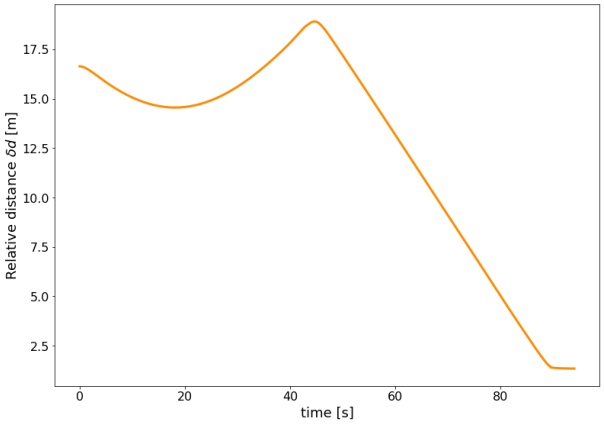

Figure 4.32: Relative distance δd [m].

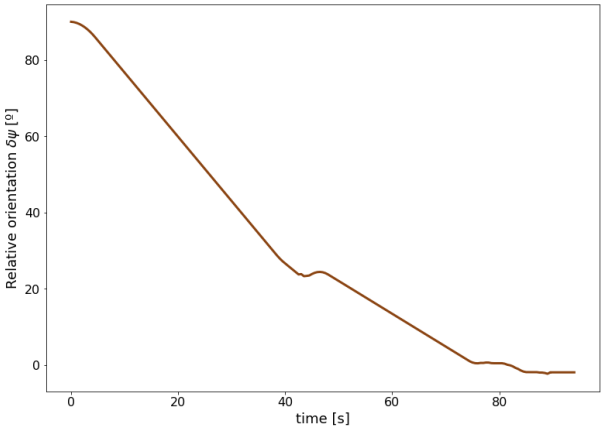

Figure 4.33: Relative orientation $\delta\psi$ [º].

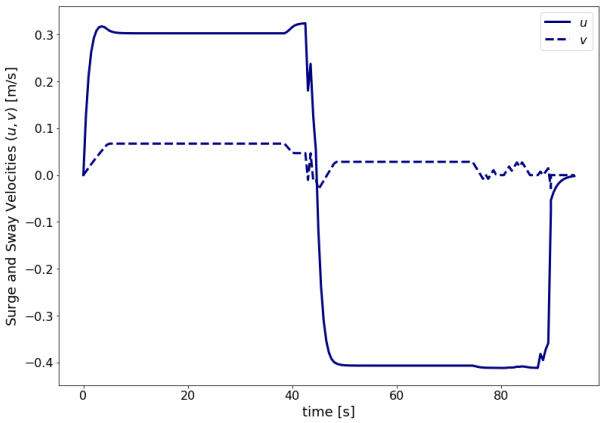

Figure 4.34: Surge and Sway Velocities (u, v) [m/s].

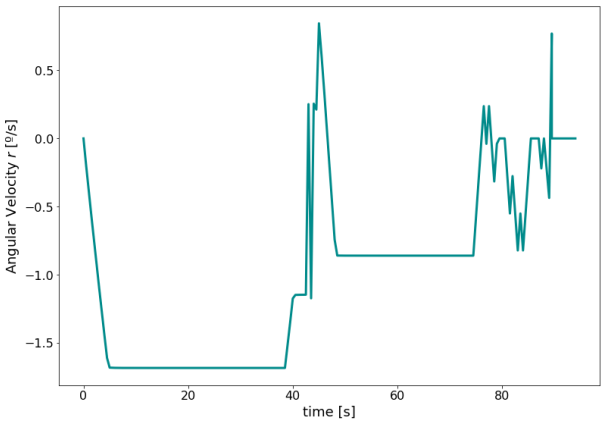

Figure 4.35: Angular Velocity r [º/s].

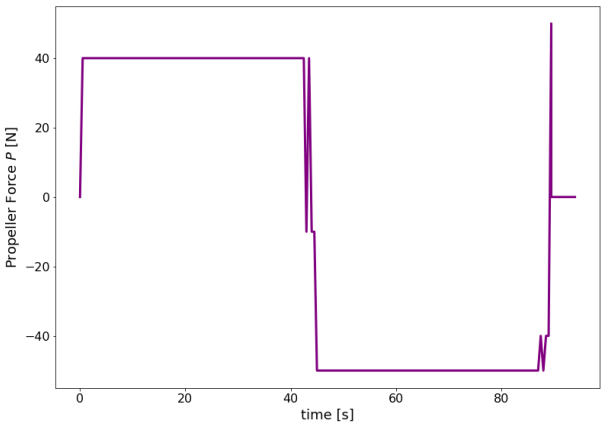

Figure 4.36: Propeller Force P [N].

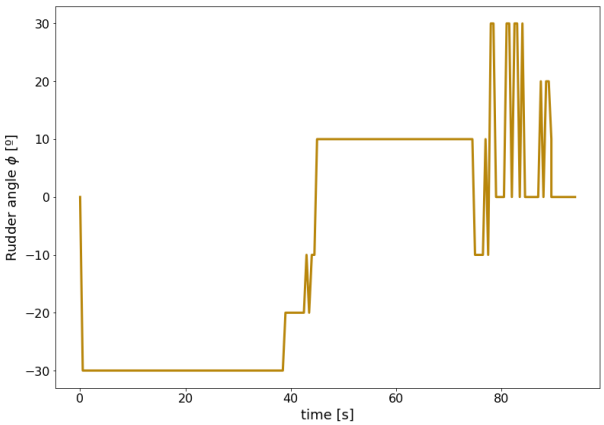

Figure 4.37: Rudder Angle ϕ [º].

approach maneuver to the final position in reverse.

The velocity behaviour (Figure 4.34) is essentially the same, since in both phases the agent puts the maximum velocity until it gets close to the intermediate and final position. At the end of the episode it is possible to see again a sudden change from a negative force to a positive force on the propeller (Figure 4.36). As stated in Section 4.3.1, this way the inertia that is associated with the constant speed of the boat is cancelled and,consequently, there is a safer stop.

Comparing also the rudder angle control signals for both scenarios one can conclude that, for this test scenario, the control is much less abrupt, and only at the end of the episode can one see a more frequent change in control.

In short, 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.

In Appendix B an extra test made for a different docking position is presented in order to complement the generalization of the model.

4.3.3 Training and Inference Time

During the training, each step took approximately 0.2s in the machine mentioned in Section 3.2.6. This duration includes the network forward pass, the backpropagation and all the necessary computations for the simulator. With this step duration it was possible to train the network for each scenario for approximately 48 hours.

To prove that this model can be deployed in a system to work in real time, a test was done during an episode, which measures the time it takes the network to make a forward pass. The total elapsed time spent on the forward pass for each step in that episode can be seen in Figure 4.38.

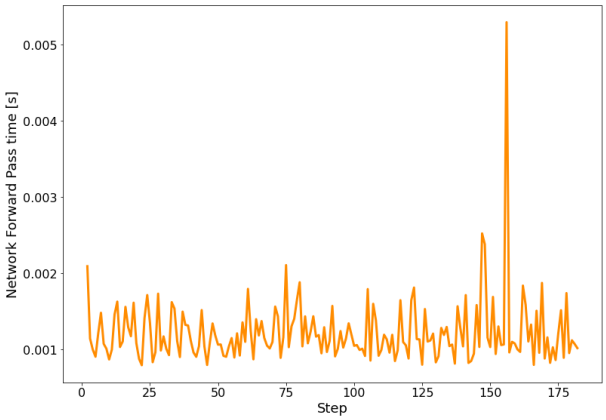

Figure 4.38: Network Forward Pass duration [s].

This gives a mean duration of 0.002s per step, which makes this approach real-time friendly.

4.4 General Discussion

To test the Deep RL model we started by trying to solve the simplest problem of the docking task - be able to dock the vessel when it is already aligned with the docking spot. In this test it was possible to observe that the agent was successful in finding the final position safely and consistently. Velocity and control plots show that, although there is a gradual decrease in velocity near the final position, this is accompanied by some sudden changes in control.

The intuition behind the reward function represented in Equation (3.10), was so that the behavior of the agent was the following:

1. The vessel needs to approximate to the final position;
2. The velocity needs to decrease gradually until a full stop.

The penalty for absolute speed was motivated by the second condition represented above. Since the desired behavior would be a gradual decrease in speed, then the intuition was to multiply by the velocity penalty another term that increases linearly with the distance that the vessel is from the final position. This way, the penalization would increase gradually as the vessel approached the final position. A gradual velocity decrease near the final position was observed but, although, in this research, only low speeds were considered, this is not an ideal behaviour for the agent.

The added task in Scenario 3 puts the agent's docking ability to the test, even starting from a position perpendicular to the dock. In this approach the agent needs to execute two different maneuvers: first it has to move away from the final position with a forward navigation in order to be aligned with the dock, and then approach the final position in reverse until a full stop.

Results for this scenario show that by adding one more variable in state space, which tells which task the agent is performing, and an intermediate position for the vessel to reach, it was possible to teach the actions that maximize the reward for both tasks. It was also possible to observe that changing this variable from 0 to 1 triggers the agent to change its behaviour, by making a forward navigation in the first task and automatically changing the behaviour to a reverse navigation as soon as the variable changes value.

The agent was successful in the proposed task, gradually approaching both the intermediate and the final position in safety. It was also possible to observe a decrease in the sudden control of the propeller, compared to the previous scenarios. This behaviour was due to the reduction to half of the parameter that multiplied by the velocity term in the reward function. This reduction made the agent learn to always put the maximum velocity in order to reduce the number of steps in which he receives a negative reward until near the final position. Near the final position, in order to reach the tolerance given for the end of the episode and since the influence of velocity was reduced, the decrease in velocity was no longer gradual. Instead, the agent had a behavior that is similar to that of a pilot in a real task - to give a negative force to the propeller until it gets close to the final position and, when it is very close, changes from a negative force to a positive force in order to counteract the inertia of the boat and fully stop it.

In addition to security and consistency, it was also observed that the model managed to generalize the behaviour even for scenarios that were significantly different from the ones seen in training. We trained the model to converge to only one position, and it managed to learn enough to converge to positions on the opposite side of the harbor. This fact is important because we managed to reach a more general model without training for all possible scenarios, which would lead to a longer training time.

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 pa-

rameter, these sudden changes have not disappeared in rudder control. 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:

- The agent 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 (Figure 4.39);
- The agent fails to reach the intermediate position but does not collide with any obstacle (Figure 4.40).

In Figure 4.39 and 4.40 it is possible to see the agent's behaviour in the two conditions in which the agent is not successful. Before analyzing what happened, it is pertinent to emphasize that the change of the variable s_{task} happens when there is a tolerance of 2.0 m in the relative distance and 5º of the orientation relative to the intermediate position. With this in mind, it is possible to see in Figure 4.39 that the agent reached this tolerance in a position of the dock where it became complicated to have safe navigation in reverse, which lead to a collision with the dock. This happened because the agent tried to replicate the successful training actions, that is, steer the vessel to reach the intermediate position in the minimum number of steps possible. Although the intermediate position was reached, it was reached in a different position from training because the agent started in a farther initial position, which caused the variable s_{task} to be changed from 0 to 1 in a position where it was difficult for the agent approach the dock in safety.

In the scenario represented in Figure 4.40 it is possible to observe the behavior of the agent when it starts so far from the initial training position that it cannot even reach the intermediate position. On the positive side, although it cannot reach the intermediate position, there was no collision either. This happens, once again, because the agent tries to replicate the training scenario, but since it starts the episode so far from the initial training position, it cannot succeed.

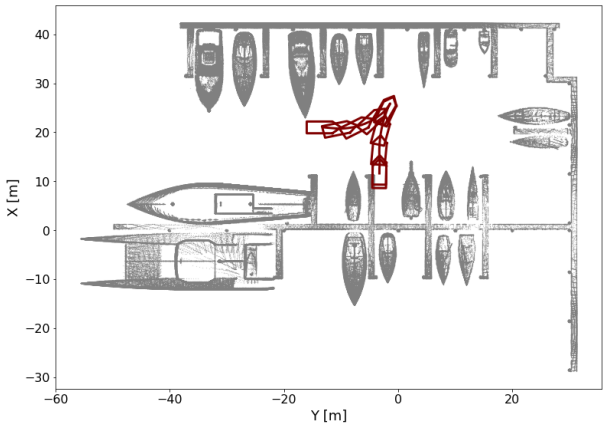

Figure 4.39: Collision case.

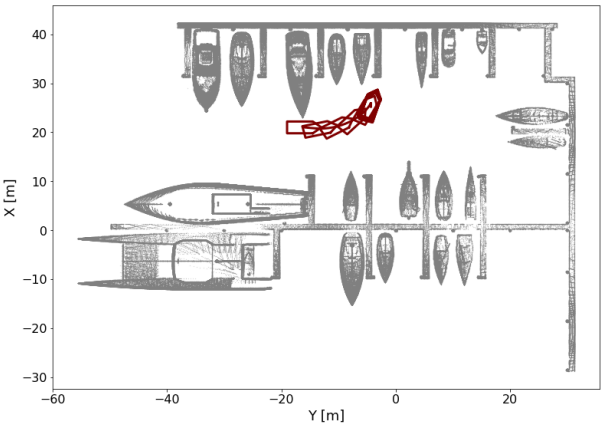

Figure 4.40: Case where the intermediate position was not reached.

These two scenarios show very well what are the limitations of the model. Since our approach is

mapping the state of the boat directly to the control, without prior planning of the path the boat will take, it is difficult to make the model be able to succeed starting in positions very different from those for which it was trained.

In short, and since the purpose of this research was not to make the entire harbor navigation, it is possible to conclude that the agent was successful in the proposed task. However, there is room to continue to investigate ways to improve the behavior of the agent. More details on approaches that could improve its performance will be explained in Section 5.2.

Chapter 5

Conclusions

5.1 Achievements

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. A series of tasks have been tested and presented subject to performance and behavior. 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 based on rewards of desired monitoring variables and without having prior knowledge of the environment.

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 two scenarios it is clear how the agent's behaviour reflects the structure and magnitude of the reward components. The distance and orientation penalties make the agent navigate towards the final position as fast as possible. This behavior is then countered with a gradual decrease in speed when the agent approaches the final position. The fact that the velocity penalty is much lower than the penalty that is being given for position and orientation (see Table 3.5), makes the agent only feel the need to slow down when the former becomes so high that it is more rewarding to slow down and reach the final position in fewer steps.

In machine learning models it is also very important to prove the generalization of the model. That is, to prove that training for a limited amount of conditions it is possible to generalize the model's behavior to other conditions that were not seen in the training. In short, to prove that the model was not overfitted for the specific training task, but can also solve tasks for which it was never trained to accomplish.

Some limited generalization was observed in the second scenario, where the agent was trained for 3 positions, being successful in other positions slightly different from the ones present during the training. It was also possible to observe this generalization in the third scenario, in which, at each episode, the agent always started in the same position with the purpose of docking in the same place. 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 addition, it was successful in positions that needed a maneuver to the opposite

side of the dock.

The state space variables choice was motivated by this same generalization, having in mind a robust model that could dock safely in all docks. In addition, this generalization is important because, training the model only for a restricted amount of conditions, instead of training for all possible conditions, causes the training time to be reduced. Another advantage is that it proves the model's robustness, as well as its operation under a wide range of conditions.

Although the model had good performance results, it was possible to find some limitations in the last scenario, in addition to those that had already been found in the first two. These 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. With this it was possible to conclude that without a pre-defined path it becomes difficult for the agent to be successful in conditions where it starts far from the docking spot.

To conclude, the docking task was successful, in an approach where state space variables make it possible to generalize the model to several docking spots without prior path planning. In addition, Deep RL techniques have shown that not only is it possible to use these techniques to bring the boat closer to the docking position in safety when it is already aligned with the dock, but it is also possible to execute the entire docking maneuver, when the boat is perpendicular to the dock and needs to navigate first to a intermediate position in order to align itself with a the dock so the reverse navigation is possible.

5.2 Future Work

One can observe some limitations, despite the good performance of the agent in the presented test scenarios. In this section, these limitations will be presented in a summarized way, as well as the proposals to try to solve them.

The first limitation lies in sudden changes in control. Although these changes practically disappeared when the velocity penalty in the reward function was reduced, the sudden changes in the rudder remained. Future work will involve adding terms to the reward function that penalize this behavior.

The second limitation is associated with the collision that occurred when the test episode started in a position further away than the initial training position. For the episodes in which the agent started less than 5 meters from the initial training position, it was possible to observe success in the docking maneuver, however, there was a collision with the dock in a test episode in which it started further than the 5 meters threshold. In comparison, starting even further away causes the agent not to reach the intermediate position, but a collision does not occur either. With this in mind, future work also involves testing the influence of the information coming from the LiDAR in collision avoidance occasions.

In Scenario 3, 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 ($s_{task} = 0 \rightarrow 1$) 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 in navigation 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, and as mentioned before, one of the greatest challenges of docking vessels is the adverse external disturbances that make this task prone to collisions with the harbor structures or other parked vessels. With that said, and in order to make the environment as realistic as possible, future work also involves adding external disturbances, such as wind and currents, and with this analyze the model performance in contexts more similar to the real world. Other perturbations such as, LiDAR measurements noise also remained to be added in this research, as well as a more complex vessel model and an actuation delay in control commands.

Bibliography

[1] J. Amendola, E. A. Tannuri, F. G. Cozman, and A. H. Reali. Batch reinforcement learning of feasible trajectories in a ship maneuvering simulator. In *Anais do XV Encontro Nacional de Inteligência Artificial e Computacional*, pages 263–274. SBC, 2018.

[2] 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.

[3] BoatU.S. Self-docking boats: What's next?, 2018. URL `https://www.boatus.com/magazine/2018/august/volvo-penta-self-docking-boats.asp`.

[4] A. Maki, N. Sakamoto, Y. Akimoto, H. Nishikawa, and N. Umeda. Application of optimal control theory based on the evolution strategy (cma-es) to automatic berthing. *Journal of Marine Science and Technology*, 25(1):221–233, 2020.

[5] 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.

[6] 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.

[7] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[8] 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.

[9] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489, 2016.

[10] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, et al. Mastering the game of go without human knowledge. *nature*, 550(7676): 354–359, 2017.

[11] J. Stopforth and D. Moodley. Continuous versus discrete action spaces for deep reinforcement learning. *CEUR Workshop Proceedings*, 2540:2–4, 2019.

[12] 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.

[13] 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.

[14] 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.

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

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

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

[18] Wikipedia contributors. Freight transport — Wikipedia, the free encyclopedia, 2020. URL `https://en.wikipedia.org/w/index.php?title=Freight_transport&oldid=969820580`.

[19] K. J. Åström and T. Hägglund. *PID controllers: theory, design, and tuning*, volume 2. Instrument society of America Research Triangle Park, NC, 1995.

[20] 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.

[21] T. I. Fossen, M. Breivik, and R. Skjetne. Line-of-sight path following of underactuated marine craft. *IFAC proceedings volumes*, 36(21):211–216, 2003.

[22] 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.

[23] D. G. Luenberger. *Introduction to Dynamic Systems: Theory, Models and Applications*. Wiley, 1979.

[24] 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.

[25] S.-R. Oh and J. Sun. Path following of underactuated marine surface vessels using line-of-sight based model predictive control. *Ocean Engineering*, 37(2-3):289–295, 2010.

[26] B. J. Guerreiro, C. Silvestre, R. Cunha, and A. Pascoal. Trajectory tracking nonlinear model predictive control for autonomous surface craft. *IEEE Transactions on Control Systems Technology*, 22 (6):2160–2175, 2014.

[27] M. Abdelaal, M. Fränzle, and A. Hahn. Nonlinear model predictive control for trajectory tracking and collision avoidance of underactuated vessels with disturbances. *Ocean Engineering*, 160:168–180, 2018.

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

[29] 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.

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

[31] 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.

[32] Y. A. Ahmed and K. Hasegawa. Automatic ship berthing using artificial neural network trained by consistent teaching data using nonlinear programming method. *Engineering applications of artificial intelligence*, 26(10):2287–2304, 2013.

[33] V. S. Nguyen. A review of automatic berthing systems based on artificial neural networks for marine ship. *International Journal of Civil Engineering and Technology (IJCIET)*, 2019.

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

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

[36] 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.

[37] 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.

[38] 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.

[39] 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.

[40] 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.

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

[42] 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.

[43] 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.

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

[45] 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.

[46] 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.

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

[48] X. Zhou, S. Sutulo, and C. G. Soares. Simulation of hydrodynamic interaction forces acting on a ship sailing across a submerged bank or an approach channel. *Ocean Engineering*, 103:103–113, 2015.

[49] A. Wnek, A. Paço, X. Zhou, and C. G. Soares. Numerical and experimental analysis of the wind forces acting on lng carrier. In *Proceedings of the 5th European conference on computational fluid dynamics (ECCOMAS CFD2010)*, 2010.

[50] V. Ferrari, L. Moreira, S. Sutulo, and C. Guedes Soares. Influence of sea current on manoeuvring of a surface autonomous model. *Maritime Engineering and Technology. London, UK: Taylor & Francis Group*, pages 173–180, 2012.

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

[52] A. W. Browning. A mathematical model to simulate small boat behaviour. *Simulation*, 56(5):329–336, 1991.

[53] F. H. Imlay. The complete expressions for added mass of a rigid body moving in an ideal fluid. Technical report, DAVID TAYLOR MODEL BASIN WASHINGTON DC, 1961.

[54] C.-W. Hsu, C.-C. Chang, C.-J. Lin, et al. A practical guide to support vector classification, 2003.

[55] S. Balakrishnama and A. Ganapathiraju. Linear discriminant analysis-a brief tutorial. *Institute for Signal and information Processing*, 18(1998):1–8, 1998.

[56] M. Awad and R. Khanna. Support vector regression. In *Efficient learning machines*, pages 67–80. Springer, 2015.

[57] A. Cutler, D. R. Cutler, and J. R. Stevens. Random forests. In *Ensemble machine learning*, pages 157–175. Springer, 2012.

[58] J. Shlens. A tutorial on principal component analysis. *CoRR*, 2014.

[59] A. Likas, N. Vlassis, and J. J. Verbeek. The global k-means clustering algorithm. *Pattern recognition*, 36(2):451–461, 2003.

[60] K. Khan, S. U. Rehman, K. Aziz, S. Fong, and S. Sarasvady. Dbscan: Past, present and future. In *The fifth international conference on the applications of digital information and web technologies (ICADIWT 2014)*, pages 232–238. IEEE, 2014.

[61] C. J. Watkins and P. Dayan. Q-learning. *Machine learning*, 8(3-4):279–292, 1992.

[62] H. Wang, T. Zariphopoulou, and X. Zhou. Exploration Versus Exploitation in Reinforcement Learning: A Stochastic Control Approach. *SSRN Electronic Journal*, pages 1–33, 2019.

[63] S. Levine, C. Finn, T. Darrell, and P. Abbeel. End-to-end training of deep visuomotor policies. *The Journal of Machine Learning Research*, 17(1):1334–1373, 2016.

[64] P. Mirowski, R. Pascanu, F. Viola, H. Soyer, A. J. Ballard, A. Banino, M. Denil, R. Goroshin, L. Sifre, K. Kavukcuoglu, et al. Learning to navigate in complex environments. *5th International Conference on Learning Representations, ICLR 2017 - Conference Track Proceedings*, 2017.

[65] F. Kunz. An introduction to temporal difference learning. In *Seminar on autonomous learning systems*, pages 21–22, 2000.

[66] R. Bellman. A markovian decision process. *Journal of mathematics and mechanics*, 6(5):679–684, 1957.

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

[68] Q. J. Huys, A. Cruickshank, and P. Seriès. Reward-based learning, model-based and model-free. In *Encyclopedia of Computational Neuroscience*, pages 2634–2641. Springer New York, 2015.

[69] C. Daniel, M. Viering, J. Metz, O. Kroemer, and J. Peters. Active reward learning. In *Robotics: Science and systems*, volume 98, 2014.

[70] S. Marsland. *Machine learning: an algorithmic perspective*. CRC press, 2015.

[71] A. Nowé, P. Vrancx, and Y.-M. De Hauwere. Game theory and multi-agent reinforcement learning. In *Reinforcement Learning*, pages 441–470. Springer, 2012.

[72] L.-J. Lin. Reinforcement learning for robots using neural networks. Technical report, Carnegie-Mellon Univ Pittsburgh PA School of Computer Science, 1993.

[73] J. N. Tsitsiklis and B. Van Roy. An analysis of temporal-difference learning with function approximation. *IEEE transactions on automatic control*, 42(5):674–690, 1997.

[74] 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.

[75] H. Hasselt. Double q-learning. *Advances in neural information processing systems*, 23:2613–2621, 2010.

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

[77] 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.

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

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

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

[81] 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.

[82] 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.

[83] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, et al. Tensorflow: A system for large-scale machine learning. In *12th {USENIX} symposium on operating systems design and implementation ({OSDI} 16)*, pages 265–283, 2016.

[84] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. In *Advances in neural information processing systems*, pages 8026–8037, 2019.

[85] A. Gulli and S. Pal. *Deep learning with Keras*. Packt Publishing Ltd, 2017.

[86] X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256. JMLR Workshop and Conference Proceedings, 2010.

[87] R. Polvara, S. Sharma, J. Wan, A. Manning, and R. Sutton. Autonomous vehicular landings on the deck of an unmanned surface vehicle using deep reinforcement learning. 2019.

Appendix A

3D Models

A.1 Vessels

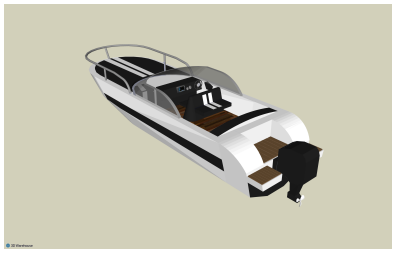

Figure A.1: Vessel 1

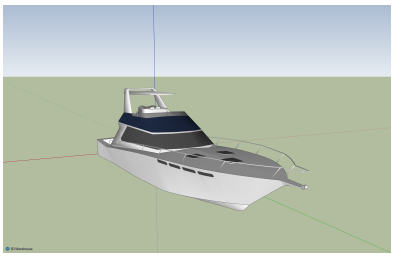

Figure A.2: Vessel 2

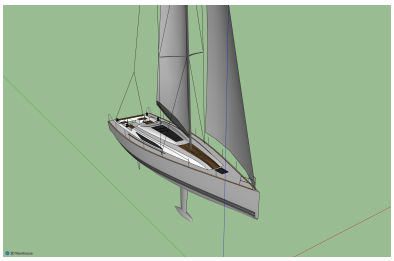

Figure A.3: Vessel 3

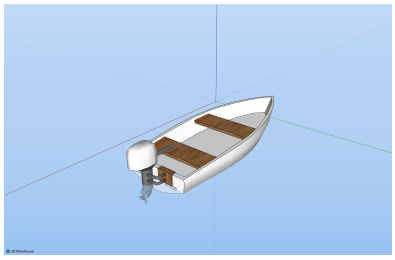

Figure A.4: Vessel 4

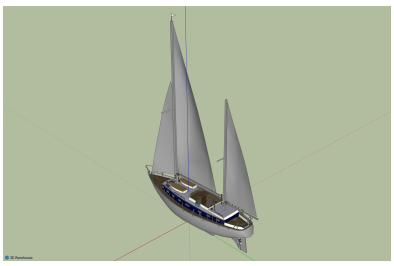

Figure A.5: Vessel 5

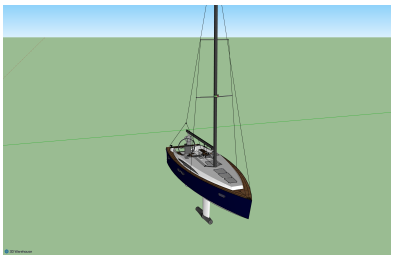

Figure A.6: Vessel 6

Figure A.7: Vessel 7

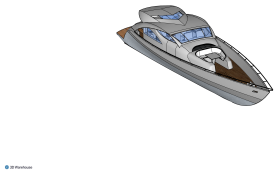

Figure A.8: Vessel 8

Figure A.9: Vessel 9

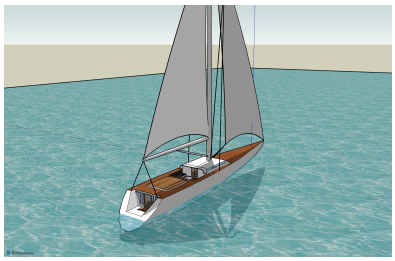

Figure A.10: Vessel 10

Figure A.11: Vessel 11

Figure A.12: Vessel 12

A.2 Docking structure

Figure A.13: Harbor Structure.

A.3 References

This section presents the 3D models references.

Figure A.1 - https://3dwarehouse.sketchup.com/model/4af786ed4226705279863338881ed398/boat

Figure A.2 - https://3dwarehouse.sketchup.com/model/378b6d20f4f5adfcb177fd3e81fc3749/Boat

Figure A.3 - https://3dwarehouse.sketchup.com/model/c8399301566f2c21eb610cbf0d8cfdd8/boat

Figure A.4 - https://3dwarehouse.sketchup.com/model/aa6a256cce6f8248b1bb46d2556ba67d/Boat

Figure A.5 - https://3dwarehouse.sketchup.com/model/8c73670f79af4d9b6a34aa94ca8a3355/Sail-Boat

Figure A.6 - https://3dwarehouse.sketchup.com/model/f1b69ee1dafd8e51fdb45062ff586ea0/Northcity-50

Figure A.7 - https://3dwarehouse.sketchup.com/model/9be050a5277076604c366679c5ee56ad/Voilier-Suspens-84

Figure A.8 - https://3dwarehouse.sketchup.com/model/a04d6b9a-7c6a-45b5-8b5d-d8fb2352b789/LANCHA-YACHT

Figure A.9 - https://3dwarehouse.sketchup.com/model/76bfbb3c-a253-481b-ad73-dc24e82aa7ab/Inflatable-Motor-Boat

Figure A.10 - https://3dwarehouse.sketchup.com/model/ec76d47241f37d8149f0e3dd7767881/My-first-Yatch

Figure A.11 - https://3dwarehouse.sketchup.com/model/7b7847ccb4f15fa9b1bb46d2556ba67d/Catamaran

Figure A.12 - https://3dwarehouse.sketchup.com/model/d317c39473534f97b1bb46d2556ba67d/Sail-Yacht

Figure A.13 - https://3dwarehouse.sketchup.com/model/b5be14e6ffd711b6b1bb46d2556ba67d/Port

Appendix B

Extra Results

In this section are presented extra results that were not presented in Section 4.

B.1

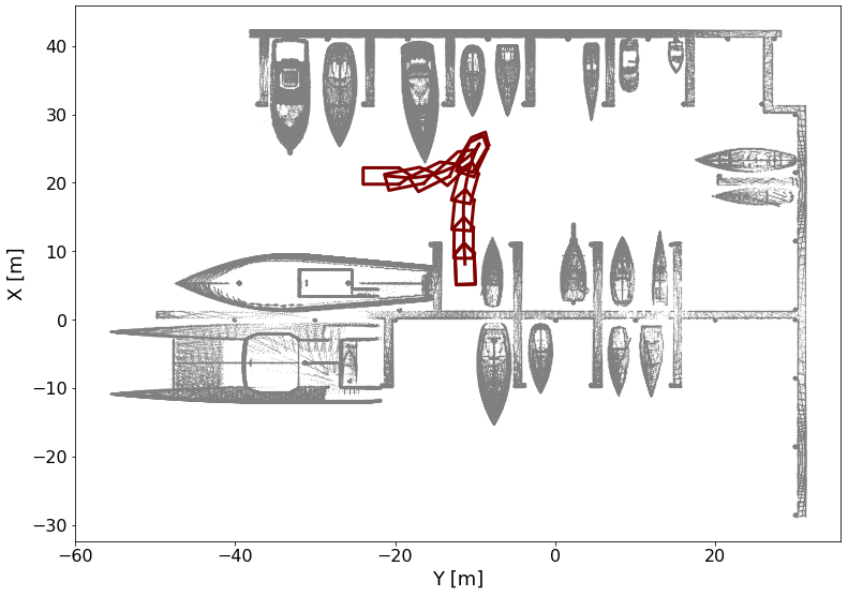

Figure B.1: Vessel's Pose (x, y, ψ).

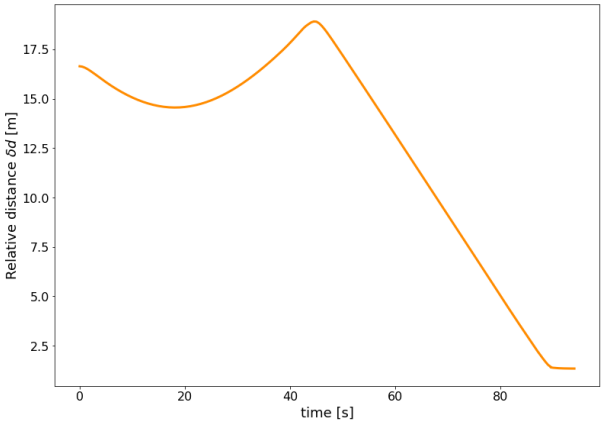

Figure B.2: Relative distance δd [m].

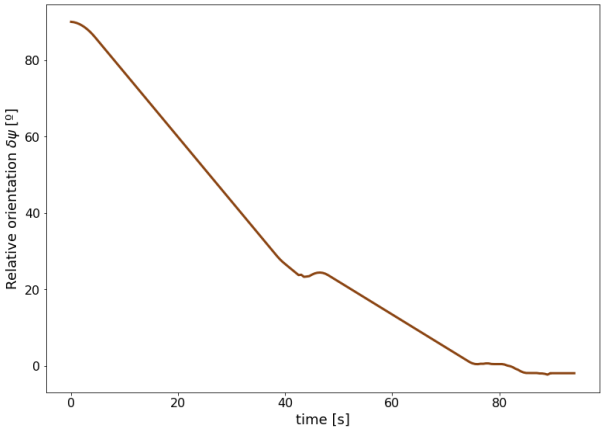

Figure B.3: Relative orientation $\delta\psi$ [º].

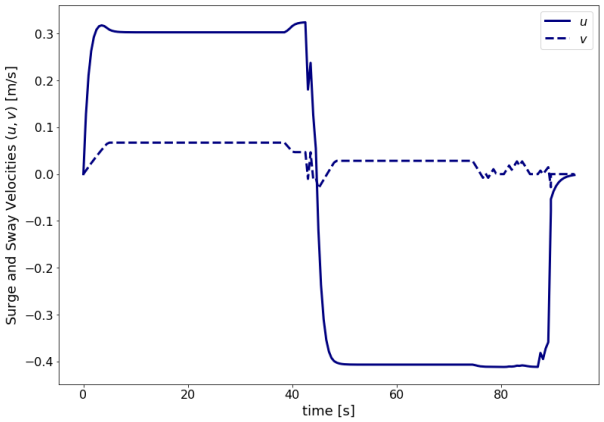

Figure B.4: Surge and Sway Velocities (u, v) [m/s].

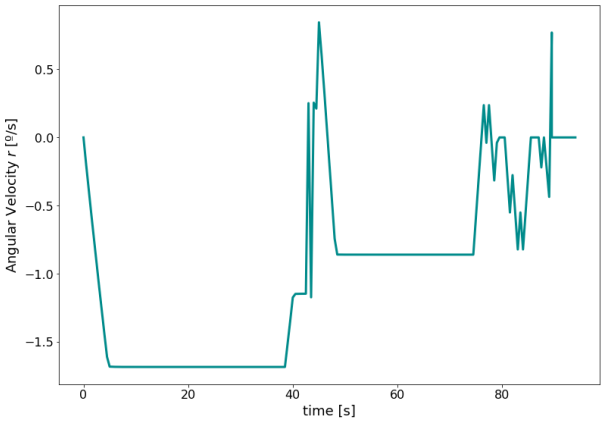

Figure B.5: Angular Velocity r [º/s].

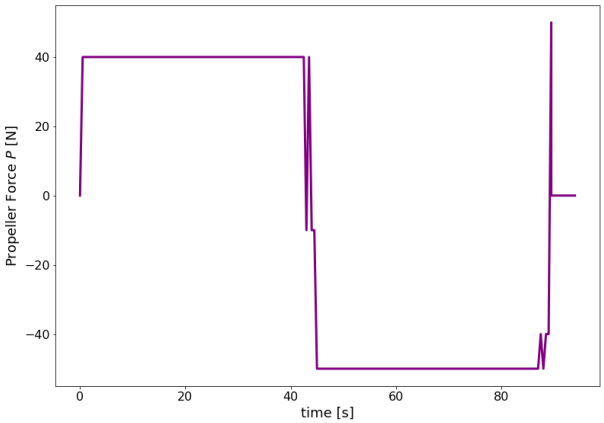

Figure B.6: Propeller Force P [N].

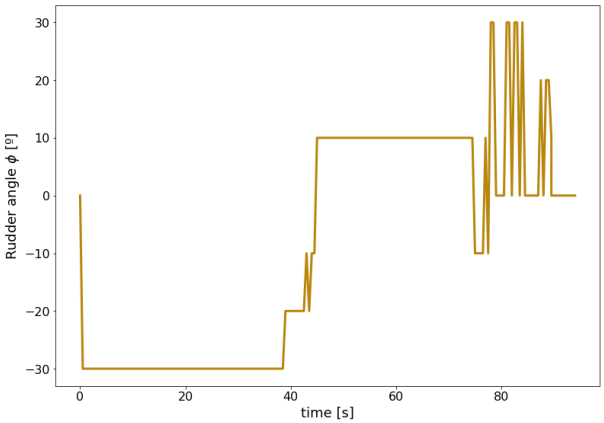

Figure B.7: Rudder Angle ϕ [º].